Curso 2003/04
CIENCIAS Y TECNOLOGÍAS/2
I.S.B.N.: 84-7756-588-0

EVELIO JOSÉ GONZÁLEZ GONZÁLEZ

Diseño e implementación de una arquitectura multipropósito basada en agentes inteligentes: aplicación a la planificación automática de agendas y al control de procesos

Director
ALBERTO FRANCISCO HAMILTON CASTRO

SERVICIO DE PUBLICACIONES
UNIVERSIDAD DE LA LAGUNA

SOPORTES AUDIOVISUALES E INFORMÁTICOS
Serie Tesis Doctorales

A Laura,
quien siempre
siembra mi vida
de momentos felices

Numerosos son los pasos necesarios en el camino de una tesis doctoral y mucha la suerte de contar con tantas personas y de tanto valor a tu lado. Valgan estas pocas líneas, siempre insuficientes, para mostrar mi agradecimiento a algunas de estas personas.

En primer lugar, quisiera agradecer al Dr. D. Alberto F. Hamilton Castro su labor de dirección, su profesionalidad, su rigurosidad, su minuciosidad y sobre todo su dedicación a este trabajo.

Especial agradecimiento merece el Dr. D. Lorenzo Moreno Ruiz, por abrirme las puertas de la carrera universitaria, así como contribuir a mi formación docente e investigadora.

A los miembros del grupo de Computadoras y Control de la Universidad de La Laguna: Dr. D. Leopoldo Acosta Sánchez, Dr. D. José Luis Sánchez de la Rosa, Dr. D. José Demetrio Piñeiro Vera, Dr. D. Juan Albino Méndez, Dr. D. Graciliano Nicolás Marichal Plasencia, Dra. Dña. Rosa María Aguilar Chinea, Dr. D. José Sigut Saavedra, Dr. D. José Ignacio Estévez Damas, Dra. Dña. Carina Soledad González, Dr. D. Roberto Luis Marichal Plasencia, Dra. Dña. Marta Sigut Saavedra, Dr. D. José Ramón Llata García, D. Juan Julián Merino Rubio, D. Santiago Torres Álvarez, Dra. Dña. Silvia Alayón Miranda, D. Héctor Reboso Morales, D. Sergio Hernández Alonso, D. José Julio Rodríguez Bello, D. Carlos Martín Galán, Dña. Vanessa Muñoz Cruz, D. Jesús M. Torres Jorge, D. Jonay Toledo Carrillo, D. Roberto Betancor Bonilla, D. Agustín José Padrón y D. Manuel Fernández Vera. A todos gracias por su ayuda y compañerismo.

A mis padres, Evelio y María del Carmen, por darme la vida y su comprensión en todo momento.

A mi hermano Víctor Manuel y a su esposa Beatriz por su apoyo incondicional.

A mis amigos Miguel, Nieves Luz, Yurena, Chema, Raquel, Eduardo, Mónica, Penélope, María José, Ricardo, Estrella, Helen, Cristina y Clara, por su interés y ofrecerme tan buenos momentos.

A Laura. ¿Cómo agradecerle en tan poco espacio todo su apoyo, su comprensión, su interés, su paciencia? Aunque sea imposible, desde estas líneas, muchísimas gracias.

Índice general

Introducción **19**

1. Programación Orientada a Objetos **23**
1.1. Programación Orientada a Objetos (OOP) y Java 23
1.1.1. Abstracción 24
1.1.2. Encapsulado 24
1.1.3. Herencia 24
1.1.4. Polimorfismo 25
1.1.5. Persistencia 25
1.1.6. Ocultación 25
1.1.7. Concurrencia 26
1.1.8. Clases abstractas 26
1.1.9. Java 26
1.1.9.1. Orígenes de Java 26
1.1.9.2. Características de Java 27
1.1.9.3. Tratamiento de Excepciones 28
1.2. Sistemas de Objetos Distribuidos: RMI y CORBA 29
1.2.1. Invocación Remota de Métodos (RMI) 30
1.2.1.1. *Stubs* y *Skeletons* 30
1.2.2. Arquitectura de Intermediación de Petición de Objetos Comunes. (CORBA) 32
1.2.2.1. El lenguaje de Definición de Interfaz (IDL) 32
1.2.2.2. *Stubs* y *Skeletons* 33
1.2.2.3. El Intermediario de Petición de Objeto (ORB) . . . 33
1.2.2.4. Servicios CORBA 34

2. Ontologías **37**
2.1. Web Semántica 37
2.2. Ontologías: Definición y clasificación 38
2.2.1. Ontologías y la Web Semántica 38
2.2.2. Ontologías y Conceptualización 38
2.2.3. Componentes de una Ontología 39
2.2.4. Clasificación de Ontologías 40
2.2.4.1. Clasificación según su dependencia de un dominio o tarea. 40
2.2.4.2. Clasificación según su grado de axiomatización. . . 40
2.2.4.3. Ontologías de meta-nivel 41
2.2.5. Principios Útiles para el Desarrollo de Ontologías 41

2.2.6. Diseño y Elaboración de Ontologías 41
2.3. Propuestas no Basadas en Lenguajes de Marcas 42
2.3.1. Dublin Core Metadata Initiative, *DCMI* 42
2.3.2. Cyc 44
2.3.3. Open Knowledge Base Connectivity (OKBC) 47
2.3.3.1. Breve Resumen de Sintaxis OKBC 48
2.3.4. Formato de Intercambio de Conocimiento (KIF) 49
2.3.4.1. KIF Lineal y KIF estructurado 50
2.3.4.2. Base 51
2.3.4.3. Lógica 51
2.3.4.4. Metaconocimiento 52
2.4. Lenguajes de marcas 53
2.4.1. Standard Generalized Markup Language, SGML 55
2.4.1.1. Estructuras SGML 56
2.4.1.2. Inconvenientes de SGML 62
2.4.2. Lenguaje de Marcas de Hipertexto (HTML) 62
2.4.2.1. Puntos débiles del HTML 63
2.4.3. Lenguaje de Marcas Extensible (XML) 63
2.4.3.1. Ventajas de XML 64
2.4.3.2. Características para un documento XML *bien formado* 64
2.4.3.3. Espacios de nombres XML 66
2.4.3.4. La declaración XML 66
2.4.3.5. XSL y XSLT 67
2.4.3.6. XML Schemas 69
2.4.4. Simple HTML Ontology Extensions, SHOE 72
2.4.4.1. Breve descripción de la Sintaxis SHOE 73
2.4.5. Lenguaje de intercambio de ontologías basado en XML (XOL) 76
2.4.5.1. Sintaxis XOL 76
2.4.6. Marco de Descripción de Recursos (RDF) 79
2.4.6.1. Ventajas y desventajas del uso del RDF respecto al XML 82
2.4.6.2. Sintaxis RDF 83
2.4.6.3. Cosificación 86
2.4.7. RDFS 87
2.4.8. Capa de Inferencia de Ontologías (OIL) 88
2.4.8.1. Standard OIL 90
2.4.8.2. Instance OIL 93
2.4.8.3. Limitaciones de OIL 93
2.4.9. DAML+OIL 94
2.4.9.1. Sintaxis 94
2.4.9.2. Lenguajes basados en DAML+OIL 98
2.4.9.3. Herramientas DAML+OIL 99
2.4.10. Lenguaje Web de Ontologías (OWL) 99
2.4.10.1. Sintaxis 100
2.4.10.2. OWL Lite, OWL DL, OWL Full 105
2.4.10.3. Ejemplo OWL 106
2.5. Herramientas de Ontologías y Lenguajes de Marcas 108

2.5.1. OpenCyc 109
2.5.2. Castor 109
2.5.3. Jena 109
2.5.3.1. RDF API 110
2.5.3.2. ARP Parser 110
2.5.3.3. RDQL query language 111
2.5.3.4. DAML API 111
2.5.3.5. Jena 2 Ontology API 111
2.5.4. OilEd 111
2.5.4.1. Elemento de Razonamiento FaCT 113
2.5.5. Protégé-2000 114
2.5.6. OntoEdit 116
2.5.7. DAML Viewer 117
2.5.8. VisioDAML 119
2.5.9. AeroDAML 119
2.5.10. TRIPLE 121
2.5.11. DAML+OIL Ontology Checker 121
2.5.12. DAML Validator 121
2.5.13. OWL Converter 123

3. Agentes y Sistemas Multiagente 125
3.1. El concepto de agente 125
3.1.1. Definiciones de agente 126
3.1.2. Características de un agente 127
3.1.3. Actitudes mentales de los agentes: el modelo BDI 128
3.1.3.1. Noción débil y noción fuerte de agente 130
3.2. Taxonomía de los agentes 130
3.3. Sistemas Multiagente 131
3.4. Lenguajes de comunicación de agentes 133
3.4.1. La teoría del habla (*Speech Act Theory)* 133
3.4.2. KQML 134
3.4.2.1. Performativas KQML 134
3.4.2.2. Parámetros de un mensaje KQML 136
3.4.2.3. Conversaciones KQML 137
3.4.2.4. Capas KQML 137
3.4.2.5. Aspectos a mejorar en KQML 138
3.4.3. FIPA-ACL 139
3.4.3.1. Tipo de mensaje en FIPA-ACL: el campo *performative* 139
3.4.3.2. Parámetros de un mensaje FIPA-ACL 140
3.4.3.3. Protocolos de interacción FIPA-ACL 142
3.4.4. Comparativa entre KQML y FIPA-ACL 143
3.5. Arquitecturas de MAS 145
3.5.1. Ejemplo de arquitectura de MAS con KQML 146
3.5.1.1. KQML Router 146
3.5.1.2. KQML Facilitator 147
3.5.1.3. KQML KRIL 147
3.5.2. Arquitectura de una plataforma FIPA: Gestión de Agentes . . 148

3.5.2.1. Agente 149
3.5.2.2. Facilitador de Directorios (DF) 149
3.5.2.3. Sistema Gestor de Agentes (AMS) 149
3.5.2.4. Servicio de Transporte de Mensajes (MTS) 151
3.5.2.5. Software 152
3.5.2.6. Plataforma de Agentes (AP) 152
3.5.2.7. El ciclo de vida de un agente 152
3.5.3. El modelo OMG MASIF 154
3.5.3.1. Transferencia de agentes 158
3.6. Restantes Especificaciones del Estándar FIPA 158
3.6.1. Especificaciones FIPA sobre un Aplicación Asistente Personal 160
3.6.1.1. Análisis General 160
3.6.1.2. Servicios de Directorio 160
3.6.1.3. Servicios de Programación de Reuniones 161
3.6.1.4. Servicios de Gestión de la Información 161
3.6.1.5. Servicio de Planificación de Viajes 162
3.6.1.6. Ontología de PA 162
3.6.2. Especificaciones FIPA sobre el Servicio de Ontologías en un MAS 162
3.6.2.1. Servicio de Ontologías 163
3.6.2.2. Relaciones entre Ontologías 166
3.6.2.3. Ejemplo de Utilidad del OA en un Sistema Multiagente 166
3.7. Programación orientada a agentes, AOP 167
3.8. Marcos de Trabajo de Agentes 169
3.8.1. FIPA-OS 169
3.8.1.1. Breve Descripción 170
3.8.1.2. Modelo de ejecución de un agente FIPA-OS 172
3.8.1.3. Proyectos de sistemas multiagente que emplean FIPA-OS 172
3.8.2. JADE 173
3.8.2.1. Breve Descripción 173
3.8.2.2. Modelo de ejecución de un agente JADE 175
3.8.2.3. Proyectos de sistemas multiagente que emplean JADE 176
3.8.3. Zeus 177
3.8.3.1. Breve Descripción 177
3.8.3.2. Modelo de ejecución de un agente Zeus 178
3.8.3.3. Proyectos de sistemas multiagente que emplean Zeus 179
3.8.4. JAT*Lite* 179
3.8.4.1. Breve Descripción 179
3.8.4.2. Arquitectura JAT*Lite* 180
3.8.4.3. Proyectos de sistemas multiagente que emplean JAT*Lite* 181
3.8.5. Jackal 181
3.8.5.1. Breve Descripción 182
3.8.6. Grasshopper 183
3.8.6.1. Breve Descripción 183

3.8.6.2. Proyectos de sistemas multiagente que emplean Grasshopper . . . 185
3.8.7. Otras herramientas MAS . . . 185

4. Sistema MASplan **189**
4.1. Consideraciones Previas . . . 189
4.1.1. Negociación en sistemas multiagente . . . 189
4.1.2. Mecanismos de votación . . . 191
4.1.2.1. Métodos preferenciales de votación . . . 192
4.1.2.2. Métodos no preferenciales de votación . . . 196
4.1.3. Mecanismos de negociación basados en subastas . . . 196
4.1.4. Mecanismos basados en regateo . . . 197
4.1.4.1. Modelo de negociación bilateral . . . 197
4.1.4.2. Tácticas dependientes del tiempo . . . 199
4.1.4.3. Tácticas dependientes de los recursos . . . 200
4.1.4.4. Tácticas dependientes del comportamiento . . . 201
4.1.5. Redes de Contrato . . . 202
4.1.6. Coaliciones . . . 202
4.1.7. Negociación basada en argumentación . . . 203
4.1.8. Teoría de Juegos . . . 203
4.1.9. Aprendizaje mediante construcción de árboles de identificación 204
4.2. Aplicaciones MAS similares . . . 209
4.2.1. Visitorbot . . . 209
4.2.2. El modelo de Sandip Sen . . . 210
4.2.3. *Electric Elves* . . . 213
4.3. Introducción a MASplan . . . 215
4.4. Elección de Herramientas . . . 217
4.4.1. Marcos de trabajo MAS . . . 217
4.4.2. Herramientas de Ontologías . . . 220
4.4.3. Lenguaje de Programación . . . 221
4.5. Arquitectura de agentes . . . 222
4.6. Diseño de Ontología . . . 225
4.6.1. Definición de usuarios CyC . . . 226
4.6.2. Definición de recursos comunes . . . 230
4.6.3. Definición de ocupaciones de horario . . . 231
4.6.4. Definición de tramos horarios . . . 231
4.6.5. Otras definiciones . . . 233
4.7. Inicialización del Sistema . . . 234
4.7.1. Inicialización de un Agente de Usuario . . . 237
4.8. Estructura de la Interfaz de Usuario . . . 238
4.9. Planificación de una reunión en MASplan . . . 241
4.9.1. Determinación del horario más adecuado . . . 241
4.9.2. Flujo de mensajes entre los agentes de MASplan . . . 246
4.10. Estructura de Implementación de los Árboles de Identificación . . . 248
4.11. Negociación de Recursos . . . 252
4.11.1. Mecanismo de Regateo . . . 253
4.11.2. Flujo de mensajes en la Negociación de Recursos . . . 256

4.11.3. Flujo de mensajes en el Cambio de Factor de Egoísmo 257
4.12. Evaluación del Sistema 260
4.12.1. Evaluación de la robustez 260
4.12.2. Evaluación del comportamiento del sistema 260
4.13. Inclusión de un nuevo agente en MASplan 261
4.13.1. Código Java de implementación del nuevo agente 261
4.13.2. Resto de modificaciones 263

5. Sistema MASCONTROL 265
5.1. Consideraciones Previas 266
5.1.1. Definiciones previas 266
5.1.2. Modelado de Sistemas 268
5.1.2.1. Función de Transferencia en Sistemas Continuos . . 268
5.1.2.2. Representación en Variables de Estado en Sistemas Continuos 269
5.1.2.3. Función de Transferencia en Sistemas Discretos . . 270
5.1.2.4. Representación en Variables de Estado en Sistemas Discretos 270
5.1.2.5. Forma Canónica Controlable 270
5.1.3. Controladores Implementados 271
5.1.3.1. Controlador PID 271
5.1.3.2. Realimentación por variables de estado. Asignación de polos. 272
5.1.4. Identificación de Sistemas 273
5.1.4.1. Introducción 273
5.1.4.2. Consideraciones sobre la entrada en un proceso de identificación 273
5.1.4.3. Nomenclatura 274
5.1.4.4. Criterios de Estimación de Parámetros 274
5.1.4.5. Modelo de Función de Transferencia Racional . . . 274
5.1.5. Identificación recursiva en tiempo real 278
5.1.5.1. Factor de olvido 279
5.1.6. Reguladores autoajustables (*Self-Tunning Regulators*) 279
5.2. Empleo de *Evenet2000* en problemas de Control 280
5.2.1. Simulación de Sistemas 280
5.2.2. Controladores basados en redes neuronales 283
5.2.3. Identificación de Sistemas 285
5.2.4. Función de Coste 286
5.2.5. Optimización en Problemas de Control 286
5.3. Otras aplicaciones de los MAS al control de procesos 287
5.4. Implementación de un MAS con KQML 291
5.5. Introducción al Sistema MASCONTROL para el control de procesos . 295
5.6. Elección de herramientas 295
5.7. Diseño de Ontología 296
5.8. Arquitectura del sistema 301
5.9. Flujo de mensajes 305
5.9.1. Flujo de mensajes en el proceso de identificación del sistema . 305

5.9.2. Flujo de mensajes en el proceso de optimización de los parámetros del controlador 306
5.9.3. Flujo de mensajes en el proceso de estimación del estancamiento de la entrada y la salida de la planta 307
5.10. Fichero de perfiles 308
5.11. Resultados 309
5.11.1. Planta de Tanques Interconectados 309
5.11.1.1. Descripción de la planta 309
5.11.1.2. Resultados 312
5.11.2. Planta de Control de Nivel de Agua en un Depósito 320
5.11.2.1. Descripción de la planta 320
5.11.2.2. Resultados 321
5.12. Inclusión de un nuevo agente en MASCONTROL 322
5.12.1. Código Java de implementación del nuevo agente 324
5.12.2. Resto de modificaciones 326

Conclusiones y Líneas Abiertas **327**

A. Performativas o actos de comunicación KQML **331**

B. Marco para la semántica KQML **335**

C. Actos de comunicación FIPA **337**

D. Marco para la semántica FIPA ACL **339**
D.1. Definiciones básicas SL 339
D.2. Abreviaturas 340
D.3. Operadores Referenciales 341
D.4. Ejemplos 341

E. Protocolos de Interacción FIPA **343**
E.1. Protocolo de Interacción *FIPA Propose* 343
E.2. Protocolo de Interacción *FIPA Request* 343
E.3. Protocolo de Interacción *FIPA Request When* 344
E.4. Protocolo de Interacción *FIPA Query* 345
E.5. Protocolo de Interacción *FIPA Contract Net* 346
E.6. Protocolo de Interacción *FIPA Iterated Contract Net* 347
E.7. Protocolo de Interacción *FIPA English Auction* 348
E.8. Protocolo de Interacción *FIPA Dutch Auction* 349
E.9. Protocolo de Interacción *FIPA Brokering* 350
E.10. Protocolo de Interacción *FIPA Recruiting* 351
E.11. Protocolo de Interacción *FIPA Subscribe* 352

F. Requisitos obligatorios FIPA para una plataforma de agentes **355**
F.1. Requisitos de Interoperabilidad 355
F.2. Requisitos del agente FIPA 355
F.3. Requisitos sobre ACL y mensajes 355
F.4. Requisitos sobre el AMS 356

F.5. Requisitos sobre el ACC 356
F.6. Requisitos sobre el DF 356

G. Requisitos opcionales FIPA para una plataforma de agentes 357
G.1. Requisitos de Interoperabilidad 357
G.2. Requisitos del agente FIPA 357
G.3. Requisitos sobre ACL y mensajes 357
G.4. Requisitos sobre Movilidad 357
G.5. Requisitos sobre el AMS 357
G.6. Requisitos sobre el ACC 358
G.7. Requisitos sobre el DF 358

H. Introducción a las Redes Neuronales 359
H.1. Representación de una neurona artificial 360
H.2. Topologías de las Redes Neuronales 362
H.3. Entrenamiento de las Redes Neuronales 363
H.4. Métodos de Entrenamiento de Redes Neuronales 364
H.4.1. BackPropagation y Real Time Recurrent Learning 365
H.4.2. Gradiente Descendente 370
H.4.3. Método del gradiente conjugado de Fletcher-Reeves 371
H.4.4. Método del Camino Aleatorio 374
H.4.5. Método del Enfriamiento Progresivo 375
H.4.6. Algoritmos Genéticos 377
H.5. Algoritmos de Optimización del Paso de Entrenamiento 379
H.5.1. Búsqueda Inicial 380
H.5.2. Método de interpolación parabólica 380
H.5.3. Método de la Sección de Oro 381

I. Grafo de Flujo de Señal y su aplicación a redes neuronales 383
I.1. Grafo de Flujo de Señal 383
I.2. Representación de Redes Neuronales en Diagramas de Bloque 388
I.3. Reglas de construcción de la Red Recíproca 389

J. Librería y Entorno de Cálculo para el Entrenamiento de Redes Neuronales Evenet2000 393
J.1. Elementos de Red 394
J.2. Cálculo de la salida y propagación del error 396
J.3. Cálculo de la salida de la red: el método `dimeSalida`. 397
J.4. Propagación de los términos de la derivada: métodos `recibeDerivada` y `mandaDerivadaAnteriores` 399
J.5. El objeto `Red` 401
J.6. Construcción de una librería de cálculo 402
J.6.1. La clase `FuncionCoste` 403
J.6.2. La clase `Algoritmo` 403
J.6.3. La clase `AlgoritmoOptimizacionPaso` 404
J.6.4. La clase `Problema` 405
J.6.5. Ejemplo de entrenamiento 405

J.6.6. Inclusión del *Real Time Recurrent Learning* (RTRL) 410
J.7. El paquete `interfase` 411
J.8. El paquete `editorGrafico` 412

Bibliografía **415**

Índice alfabético **431**

Introducción

El concepto de *agente* ha cobrado una enorme importancia en nuestros días dentro de numerosos campos. Así se emplean en problemas muy diversos como el manejo y control de la información en redes de computadoras, la planificación de la agenda personal, la compra de billetes de avión o el transporte militar.

Su origen se sitúa en el año 1977, en el que Carl Hewitt propone la idea de un objeto auto-contenido, interactivo y ejecutado de modo concurrente llamado *actor* definido como

> un agente computacional que tiene una dirección y un comportamiento. Los actores se comunican mediante el paso de mensajes y llevan a cabo sus acciones de forma concurrente.

Sin embargo, a pesar de lo extendido de su aplicación, su definición es todavía objeto de discusión y análisis. De hecho no existe una definición académica aceptada por todo el mundo. Como razones esgrimidas para ello destacan dos. En primer lugar, los investigadores que trabajan en el campo de los agentes *no poseen en exclusiva* el término. Por otro lado, el término "agente" se ha convertido en un paraguas donde se cobijan investigaciones muy heterogéneas, y que se sienten cómodas bajo ella. Esto explica la gran cantidad de autores que han aportado su propia definición. La mayoría de estas definiciones suelen incidir en aspectos como la autonomía del sistema o su capacidad de reacción ante cambios producidos en su entorno.

Sin embargo, son muy pocos los campos en que un único agente basta para llevar a cabo una tarea compleja. Lo normal es crear una *sociedad* de agentes que se comuniquen entre ellos, colaboren y se coordinen en la realización de la tarea: los *sistemas multiagente* (MAS).

Los objetivos iniciales del presente proyecto de tesis han sido los siguientes:

- Estudio en profundidad de los aspectos relacionados con el campo de los sistemas de agentes inteligentes y de la tecnología asociada: arquitecturas, lenguajes de comunicación, marcos de trabajo para la implementación.
- Determinación, a partir de dicho estudio, de las tecnologías consideradas como más avanzadas y que ofrecen más posibilidades.
- Desarrollo de una nueva plataforma de agentes que reúna los aspectos más avanzados de cada una de las arquitecturas estudiadas.
- Aplicación de la nueva plataforma diseñada en dos campos lo suficiente diversos como para demostrar el poder y versatilidad de la misma, y de los sistemas multiagente en general. En concreto, una aplicación de gestión de agendas y otra del área de Control de Sistemas.

Tras los estudios y la realización de un primer prototipo, tal como se mostrará en el transcurso de este proyecto de tesis, estos objetivos iniciales evolucionaron. Se constató la existencia de un estándar de MAS que se encontraba al nivel de nuestras exigencias iniciales. Por otro lado, se descubrió la gran potencia de incluir lenguajes de representación del conocimiento en los MAS.

En cuanto a la estructura del presente proyecto de tesis, antes de los apartados dedicados a la implementación de estos sistemas y los resultados obtenidos, emplearemos algunos capítulos para introducir una serie de aspectos generales que, tras el estudio realizado, consideramos importantes.

Dedicaremos el primer capítulo a introducir al lector en la programación orientada a objetos, el lenguaje Java y los sistemas de objetos distribuidos (RMI y CORBA). El motivo de este capítulo son las continuas referencias a estos conceptos a lo largo de este trabajo. Además, el Java ofrece numerosas características para el desarrollo de sistemas multiagentes, hasta el punto de que la inmensa mayoría de las herramientas para la construcción de estos sistemas se encuentran implementadas en este lenguaje. Su principal ventaja es que es multiplataforma, es decir, un software independiente de la arquitectura en la que se ejecuta.

El siguiente capítulo se centra en el estudio del concepto de ontología, que se ha demostrado, a raíz del estudio realizado, como un aspecto puntero en la tecnología de MAS, puesto que permite una interoperabilidad más racional entre los agentes, dotando de capacidad de inteligencia e inferencia al sistema y alejándolo, de esta forma, de una mera resolución distribuida de problemas. Por tanto, su utilización proporcionará una mayor potencialidad a la arquitectura de agentes. Comenzaremos dicho capítulo presentando los conceptos de Web Semántica (cuyo propósito es codificar la información de un modo entendible para las máquinas) y del propio término de *ontología*, así como sus componentes y su clasificación según diversos criterios. Seguiremos con la descripción, siempre de una forma breve, de diversas iniciativas para aumentar la información extraíble de un documento por una máquina. Debido a su empleo en este trabajo, analizaremos con un mayor detalle los denominados lenguajes de marcas. Cada vez más surgen nuevas formas de codificación de ontologías basadas en este tipo de lenguajes ya que: ofrecen una alta capacidad de representación, son fácilmente manejables y entendibles, y existen numerosas herramientas para su procesamiento. El recorrido por este tipo de lenguajes se lleva a cabo en un orden creciente de riqueza semántica: SGML, XML, RDF, DAML+OIL y OWL. La integración de estos lenguajes es uno de los puntos novedosos de este trabajo y que se ha demostrado como enormemente útil, puesto que permite a un sistema multiagente manejar información no declarada de forma explícita.

El último capítulo introductorio está dedicado a la presentación de los aspectos relativos al concepto de agente y sistema multiagente. En primer lugar, se presenta una noción de lo que significa un agente. A pesar de que, como se ha indicado, no existe una definición universalmente aceptada, sí que se pueden extraer un conjunto de características que debiera tener un agente. A continuación justificaremos su utilización centrándonos en dos aspectos fundamentales de estos sistemas: la comunicación y la arquitectura. En este sentido, se estudiarán los estándares más extendidos (KQML, OMG MASIF...). Dedicaremos una especial atención al estándar FIPA (avalado por una organización compuesta por numerosas e importantes universidades y empresas) debido a su importancia y su repercusión en el trabajo

presentado en este proyecto de tesis. Se ha escogido dicho estándar para los desarrollos realizados porque se encuentra al nivel de las necesidades y objetivos de este trabajo. Asimismo también se describirán las diferentes herramientas existentes para la implementación de sistemas multiagente.

En el siguiente capítulo se presenta la aplicación de MAS, denominada MASplan, para la planificación de agendas en el escenario de un grupo de investigación universitario, concretamente el Grupo de Computadoras y Control (CyC) de la Universidad de La Laguna, del que el autor es miembro. Este sistema se encarga de determinar el mejor horario para la organización de reuniones y de proporcionar un mecanismo de reserva de los recursos comunes dentro del grupo CyC. Se ha escogido este tipo de problema por ser un campo clásico en el campo de las aplicaciones MAS así como por el gran número y variedad de agentes involucrados y tareas asumidas. Previamente a la descripción del MAS, dedicaremos una especial atención al estudio de las tácticas implementadas, concretamente los mecanismos de votación y de regateo, que son fundamentales para la negociación entre agentes. Para completar la base teórica necesaria, se describe la técnica de los árboles de identificación para la generación de reglas a partir de un conjunto de datos observados. Esta técnica permite dotar de cierta inteligencia a los agentes en el MAS. El resto del capítulo se dedica a los aspectos de diseño e implementación, para la cual se ha empleado principalmente una herramienta de construcción basada en el estándar FIPA (FIPA-OS), una ontología desarrollada en un lenguaje de marcas de alta riqueza semántica (DAML+OIL) y un procesador para dicho lenguaje (Jena).

En el último capítulo se detalla la otra implementación realizada: el sistema MAS para la identificación y control de procesos, MASCONTROL. De este modo se entronca con la disciplina de control, una de las áreas principales del trabajo del Grupo de Computadoras y Control de la Universidad de la Laguna, en cuyo seno se ha desarrollado el presente proyecto de tesis. Como se ha indicado, el sistema MASCONTROL busca realizar simultáneamente los procesos de identificación y control de una planta. Esto se consigue adoptando una configuración basada en los reguladores autoajustables (STR), que determinase los mejores parámetros para diferentes controladores: P, PI y asignación de polos. En su implementación se ha empleado la herramienta de entrenamiento de redes neuronales *Evenet2000* por lo que será descrito su empleo en problemas relacionados con el control de procesos. Dicha herramienta había sido desarrollada por parte del autor de este proyecto de tesis en trabajos previos en el seno del Grupo de Computadoras y Control de la Universidad de la Laguna.

Al final del capítulo correspondiente a cada implementación se ha descrito el fácil proceso de inclusión de otros agentes, pudiendo estar codificados en otros lenguajes diferentes al Java y/o por parte de otros programadores. Éste es uno de los puntos más interesantes de los sistemas multiagente: la posibilidad de ampliación según nuevo paradigma de programación, la programación orientada a agentes.

El presente trabajo se completa con una serie de apéndices, como elemento de consulta, referidos a aspectos concretos como las especificaciones KQML y FIPA (requerimientos, actos de comunicación, protocolos de interacción...), así como a aspectos relativos a la herramienta *Evenet2000*, como las redes neuronales y la técnica de Grafo de Flujo de Señal.

1. Programación Orientada a Objetos

Dedicaremos este primer capítulo a introducir al lector en la programación orientada a objetos, el lenguaje Java y los sistemas de objetos distribuidos (RMI y CORBA). Consideramos esta introducción necesaria puesto que se van a hacer continuas referencias a estos conceptos a lo largo de este trabajo, especialmente al lenguaje de programación Java. Este lenguaje ofrece numerosas características para el desarrollo de sistemas multiagentes, hasta el punto de que la inmensa mayoría de las herramientas para la construcción de estos sistemas, tal como se verá en capítulos posteriores, se encuentran implementadas en este lenguaje. La principal ventaja de Java es que es multiplataforma, es decir, un software independiente de la arquitectura en la que se ejecuta. Esto lo diferencia de un programa escrito en C o C++ que, cuando se cambia la plataforma, debe ser recompilado y seguramente modificado. Basándose en dicha ventaja, un sistema multiagente programado en Java puede ser distribuido en varias máquinas, incluso con diferentes sistemas operativos. El capítulo se complementa con la comunicación entre sistemas de objetos distribuidos, concretamente la Invocación de Métodos Remotos (RMI) y el Common Object Request Broker Architecture (CORBA).

1.1. Programación Orientada a Objetos (OOP) y Java

La programación orientada a objetos (*Object-Oriented Programming*, OOP) tiene sus orígenes a finales de la década de los 60 con el lenguaje llamado Simula. Este lenguaje introdujo la idea de objetos para simular entidades del mundo real. En la primera mitad de los 70, otro lenguaje, Smalltalk profundizó en la idea de emplear objetos del mundo real en el desarrollo de aplicaciones. A mediados de la década de los 80, surgieron otros lenguajes orientados a objetos como C++ y Eiffel.

El 23 de mayo de 1995, Sun *Microsystems* anunció oficialmente el lanzamiento del lenguaje Java, basado en C++, pero eliminando todas las características que comprometían su modelo de orientación a objetos.

Los siguientes subapartados describen los términos básicos de la OOP. Existen diversas opiniones sobre cuáles son las verdaderas características que determinan la programación orientada a objetos. Algunos autores afirman que un lenguaje es orientado a objetos simplemente si soporta la abstracción y el encapsulado. En el lado contrario están los que exigen además conceptos como la herencia, el polimorfismo, la ocultación dinámica y la persistencia. En la subsección 1.1.9 se hablará en detalle del lenguaje de programación Java, en el cual están implementadas la mayoría de las herramientas MAS.

1.1.1. Abstracción

Para explicar este concepto, considérese el ejemplo de una persona cualquiera y de un programa que gestione las cuentas corrientes de los clientes de un banco. La persona considerada tiene numerosas características como su color de pelo, peso, altura, edad y muchas más. Sin embargo, los encargados de desarrollar el programa no están interesados en detalles como el color de ojos del cliente, sino en cuánto dinero retira, cuál es su sueldo o si tiene créditos pendientes. En otras palabras: no están interesados en el cliente "real" (con todas sus características), sino en una "abstracción" del mismo con las características necesarias para el programa.

La OOP proporciona al programador una capacidad de representación de la realidad que va mucho más allá que la de los lenguajes tradicionales (ensamblador, FORTRAN, C, PROLOG), siendo lo suficientemente general para que no se vea restringido a un tipo particular de situaciones. Los elementos en el espacio problema y su representación en el espacio solución (programa) se llaman *objetos*.

La idea que subyace en este tipo de programación es que se pueda describir el problema a resolver en términos de sí mismo, lo que tiene una correspondencia más cercana al mundo real: los seres humanos no dividen el mundo en dos partes (datos y funciones, como sucede en la programación estructurada), sino que lo perciben como un conjunto de objetos, cada uno con un estado (descrito por sus *atributos*) y un conjunto de acciones que producen cambios en él (*métodos*).

Estos atributos y métodos de un objeto se condensan en su *clase*. Una clase es una especie de "plantilla" a partir de la cual se producen objetos del mismo tipo conteniendo todos los datos y métodos comunes.

1.1.2. Encapsulado

Para comprender el concepto de encapsulado, considérese el siguiente ejemplo: cuando se conduce un automóvil, sus atributos y sus métodos son percibidos como una unidad. Así, solamente se puede cambiar una de sus propiedades (por ejemplo su velocidad) a través de uno de sus métodos (acelerar/frenar). Del mismo modo, el estado actual del objeto influye en el resultado de sus acciones. Por ejemplo, los resultados de la acción *acelerar* no es la misma si está metida una marcha o no.

La OOP refleja esta propiedad integrando en un objeto dado todos los atributos y métodos relativos al mismo. La única manera en que se puede modificar el estado de un objeto es mediante la invocación de los métodos pertinentes. A este modo de estructuración es a lo que se conoce como "encapsulado".

1.1.3. Herencia

Considérense las clases *Automóvil* y *Camión*. Muchos de los atributos y de los métodos de *Camión* son idénticos o parecidos a los de *Automóvil*. Anteriormente a la OOP, la manera de implementar la clase *Camión* consistía en replicar el código y modificar lo necesario. Sin embargo, lo ideal consistiría en derivar la nueva clase directamente de la antigua, añadiendo algunas nuevas funciones y/o atributos, pero a la vez manteniendo las propiedades de la clase anterior.

La OOP proporciona este recurso mediante la *herencia*. Es un mecanismo que permite reutilizar código, indicando cómo una clase se diferencia de otra definida

previamente, tomando parte de sus características (o todas) y cambiando o ampliando el resto. Para que sea efectiva, no obstante, el diseño de la jerarquía de clases debe ser muy preciso.

Así, en el ejemplo anterior, existen atributos característicos de los camiones que no poseen los automóviles y viceversa, por lo que no parece conveniente la herencia de una clase por parte de la otra. Es más adecuado definir una nueva clase llamada *VehículoTerrestre* de la cual pueden heredar propiedades ambas clases. A la clase "madre" en un proceso de herencia se le llama *superclase*, mientras que las clases que derivan de una superclase son sus *subclases.*

1.1.4. Polimorfismo

El polimorfismo al que se refiere este apartado consiste en las múltiples formas que pueden adquirir los atributos y/o métodos según la clase que los contiene. Así, cada una de las subclases de la misma superclase puede redefinir y/o extender los términos de la superclase. De esta forma, invocaciones de métodos del mismo nombre en diferentes clases implicarán efectos diferentes.

Cuando una subclase redefine un método de su superclase emplea el mismo nombre que ésta. Si por el contrario, el comportamiento descrito por el método de la superclase es válido para la subclase, ésta no necesita sobreescribirlo, independientemente que lo hagan otras subclases.

De modo formal, el polimorfismo consiste en la posibilidad de que toda referencia a un objeto de una clase determinada (padre) puede tomar la forma de una referencia a un objeto de una clase heredada de la anterior (hijo) [Tar00].

1.1.5. Persistencia

Es la propiedad por la cual la existencia de un objeto trasciende en el tiempo (el objeto sigue existiendo después de que su creador deja de existir) o en el espacio (la localización del objeto cambia respecto a la dirección en que fue creado). Esto se consigue, por ejemplo, escribiendo el objeto en un fichero binario, el cual puede ser enviado a otra máquina que lo puede volver a crear.

1.1.6. Ocultación

Para este concepto, considérese el siguiente ejemplo. Cuando se emplea un televisor, el usuario no está interesado en su electrónica interna, simplemente en que al pulsar los diversos botones del mando a distancia consiga encenderlo, subir el volumen o cambiar de canal. Del mismo modo, cuando un usuario utilice un objeto (ya sea mediante las funciones que se pueden aplicar sobre él o mediante la información que suministra) únicamente está interesado en su funcionamiento y no en los datos o algoritmos que son empleados en la implementación de los métodos. Así, un objeto bien diseñado a través de la OOP interactuaría con otros objetos mediante la invocación de sus métodos, jamás mediante acceso directo a sus atributos.

Mediante la ocultación, una clase puede cambiar la implementación interna de sus métodos sin afectar a las clases que los invocan.

1.1.7. Concurrencia

Es la propiedad que distingue un objeto activo de uno no activo. Permite que diferentes objetos actúen al mismo tiempo, usando hilos (*thread*) de control. Por hilo o thread se entiende una secuencia de pasos que se ejecutan de uno en uno.

1.1.8. Clases abstractas

Las clases *abstractas* son aquéllas necesarias en la estructura de la jerarquía de clases pero de las cuales no se instancian objetos. Por ejemplo una superclase *Comida*[1] de la cual heredan las subclases *Yogur* y *Lenteja.* No tiene sentido instanciar directamente de la superclase ya que no existe ningún objeto del tipo *Comida*, por lo que esta clase se define como *abstracta*. En esta clase *Comida* pueden estar definidas propiedades comunes tanto a *Yogur* como a *Lenteja*, por ejemplo el atributo *fechaDeCaducidad*[2]. No obstante, existen métodos que no tiene sentido su implementación en *Comida* (por ejemplo *modoDePreparacion* no puede ser implementada puesto que cada subclase lo implementará de modo radicalmente diferente) pero que deben ser definidos obligatoriamente en las subclases. A estos métodos no implementados, sólo declarados, se les llama *métodos abstractos.*

1.1.9. Java

1.1.9.1. Orígenes de Java

En 1990 *Sun Microsystems* se plantea el problema de mantener la compatibilidad entre los diversos APIs[3] que empleaban sus compiladores de C++. Para la ocasión se creó un equipo liderado por James Gosling e inicialmente formado por Patrick Naughton, Chris Warth, Ed Frank, Mike Sheridan y Billy Joy. Con posterioridad se integraron al grupo Arthur van Hoff, Jonathan Payne, Frank Yellin y Tim Lindholm. A través de un proyecto denominado *Green Project* se pretendía crear un lenguaje cómodo de programar, potente, de propósito general, robusto y fiable. Este lenguaje debería permitir tanto crear un enorme sistema de gestión como programar un microondas.

Durante su investigación, Gosling y su equipo llegaron a la conclusión de que algunos lenguajes de programación como C y C++ no permiten realizar la tarea de hacer un software independiente de la arquitectura en la que se ejecuta. Así, un programa escrito en C o C++ debe ser compilado para ejecutarse en una determinada plataforma. Cuando se cambia la plataforma, el programa debe ser recompilado y seguramente modificado.

El *Green Team* empezó a trabajar sobre el C++ con fines de diseñar un nuevo lenguaje de programación más adecuado para los dispositivos de consumo que emplean software electrónico. Como resultado de esta investigación surgió *Oak*, un

[1] por convenio, los nombres de las clases empiezan por mayúsculas

[2] Por convenio los nombres de atributos y métodos empiezan con minúscula. Si este nombre está compuesto por varias palabras, cada una de ellas empezará con mayúscula.

[3] API, *Application Programming Interface*: Serie de definiciones sobre la forma en que un elemento sotware se comunica con otro elemento software.

lenguaje fiable e independiente de la arquitectura, simplificado al máximo para ser compatible con el espacio limitado en los chips.

Con el nacimiento de la *World Wide Web*, Gosling y sus colaboradores se dieron cuenta de que Java (el nuevo nombre del programa)[4], un lenguaje cuyos programas pueden ejecutarse en cualquier plataforma, sería ideal para programar en Internet ya que hay mucha diversidad de tipos de ordenadores conectados a la WWW. De esta forma Java se convirtió en un asunto prioritario para Sun.

El equipo escribió un navegador llamado *HotJava*, el primero en soportar *applets*[5] de Java. Este navegador demostró el poder de Java y lo puso de moda entre los programadores y usuarios de Interner en general. Al mismo tiempo que Sun publicaba la versión beta del lenguaje, *Netscape* anunciaba que la versión 2.0 del *Netscape Navigator* soportaría applets de Java. Esto sirvió para incrementar el fuerte interés en la tecnología Java, que aún no hace más que crecer, como lo indican las herramientas analizadas en la sección 3.8.

1.1.9.2. Características de Java

Java es un lenguaje de programación de propósito general orientado a objetos desarrollado por *Sun Microsystems.* De este modo, al depender de una única compañía, no existe necesidad de plantear una estandarización, tal como sucede en otros lenguajes. Pero también se puede afirmar que Java es una nueva tecnología que no sólo se reduce al lenguaje. Proporciona una máquina virtual Java (*JVM*) que permite ejecutar código compilado Java, sea cual sea la plataforma que exista por debajo: tanto hardware como software (sistema operativo que soporta ese hardware). La fuerza de esta tecnología viene dado por la gran cantidad de fabricantes que prestan apoyo a esta especificación de máquina virtual.

Además, los applets han dotado de una gran interactividad a las páginas Web que antes no era posible. Java también aumenta el contenido multimedia de estas páginas, ofreciendo animaciones fluidas, gráficos mejorados, sonido y vídeo, independientemente de la plataforma y sin necesidad de utilizar aplicaciones dentro de sus navegadores. Esta idoneidad para el empleo de Internet se ve reforzada por las potentes librerías que permiten emplear protocolos *http* y *ftp*.

Sun describe al lenguaje de la siguiente manera:

> Simple, orientado a objetos, tipado estáticamente, distribuido, interpretado, robusto, seguro, de arquitectura neutral, portable, de alto rendimiento (sobre todo con la aparición de hardware especializado y mejor software), multitarea y dinámico

Todas estas características son importantes, sin embargo cabe destacar tres que son las que han generado el creciente interés por el lenguaje: la portabilidad (por ejemplo, un entero siempre son 32 bits en complemento a 2 independientemente de la plataforma en que se ejecuta), el hecho de que sea de arquitectura neutral (habilidad para ser ejecutado en múltiples plataformas) y su capacidad multihilo, que permite ejecutar varios *threads* de modo simultáneo.

[4] *Oak* se encontraba ya registrado. El origen del nombre de Java es incierto, aunque parece que es un homenaje a las incontables tazas de café consumidas por los programadores de todo el mundo.

[5] pequeños programas que pueden ser incrustados en otra aplicación

Java ofrece toda la funcionalidad de los lenguajes potentes pero sin las características menos usadas y más confusas de éstos. Java elimina muchas de las características de C++, por ejemplo, ha eliminado, entre otras, los ficheros de cabecera, aritmética de punteros, estructuras, la conversión implícita de tipos y la sobrecarga de operadores. Además ha añadido un reciclador de memoria dinámica (*garbage collector*), por lo que no es necesario preocuparse por la liberación de memoria.

La robustez y seguridad de Java se manifiestan en las sucesivas comprobaciones realizadas tanto en tiempo de compilación como de ejecución. Java obliga a la declaración explícita de métodos y, al eliminar los punteros y el *casting* implícito, previene un posible acceso ilegal a la memoria. Antes de ejecutarse en una máquina, se comprueba en los *bytecodes*[6] que el código no intenta falsear punteros, violar derechos de acceso sobre objetos o cambiar el tipo o clase de un objeto.

Se suele considerar que Java es interpretado, aunque en realidad es tanto interpretado como compilado. De hecho, solamente cerca del 20 % del código es interpretado por la JVM, pero es un 20 % muy importante. Tanto la seguridad de Java como su habilidad para ser ejecutado en múltiples plataformas se deben a que los pasos finales de la compilación se manejan localmente. Existen ciertos entornos que, para mejorar el rendimiento, emplean un pequeño compilador para la última fase de compilación, generando código nativo para esa plataforma. Estos compiladores se conocen como JIT (*Just in Time*).

Java es un lenguaje basado en OOP, pero con algunas diferencias respecto a otros lenguajes del mismo tipo. La mayoría de los lenguajes OOP permiten la herencia múltiple, lo que puede llevar a confusiones y/o complicaciones innecesarias. Java solamente soporta la herencia simple en una estructura en las que todas las clases derivan jerárquicamente de la clase *Object*. Si un objeto necesita heredar de dos o más entidades, se emplean las *interfaces*, conjuntos de métodos no implementados y que obligatoriamente deben declararse en los objetos que implementan estas interfaces [AG97, Eck98, Gon00].

1.1.9.3. Tratamiento de Excepciones

Una *excepción* es una condición anormal (problemas de rango, errores hardware, accesos a posiciones fuera de límite de un array...) que surge durante la ejecución de una secuencia de código. De modo tradicional, los lenguajes de programación forzaban al programador al empleo de *códigos de retorno* desde las llamadas a funciones para señalar los errores producidos.

En Java el tratamiento de estas excepciones se produce mediante un objeto, llamado Exception. Este objeto se envía al método que ha provocado la excepción cuando surge una condición anormal. Los métodos pueden protegerse frente a una salida prematura y hacer que se ejecute un bloque de código antes de la salida del método [Nau96].

[6] Código resultante de la compilación de un programa en Java. Se trata de código máquina virtual, entendible por el intérprete Java.

1.2. Sistemas de Objetos Distribuidos: RMI y CORBA

Los *sistemas distribuidos* son sistemas cuyas partes se encuentran separadas geográficamente dentro de una red, ya sea porque estén distribuidos los datos, la computación o los usuarios. El uso de este tipo de sistemas permite una mejor gestión de los recursos ya que ofrece entre otras ventajas:

- Englobar en el mismo sistema dispositivos heterogéneos.
- El procesamiento global no se ve bloqueado ante el fallo de una de sus partes.
- Los datos empleados únicamente por una parte del sistema, no cargan innecesariamente el procesamiento en otras partes del sistema.
- El coste de un servidor central muy potente con una serie de "clientes tontos" que se aprovechen de los recursos ofrecidos por el servidor (*mainframe),* es mayor que el de varios ordenadores no demasiado potentes.
- Cubre las necesidades derivadas de distancias geográficas y requerimientos de cálculo por parte de un gran número de organizaciones.

Una de las ideas que primero surgen es el de la comunicación entre las diversas partes del sistema. El lenguaje de programación Java proporciona soporte para *sockets*, que en general son suficiente para el establecimiento de una comunicación. Sin embargo, los *sockets* exigen que tanto el cliente como el servidor se sumerjan en protocolos de bajo nivel para la codificación y decodificación de mensajes para el intercambio, y el diseño de esos protocolos suele ser engorroso y fuente de errores.

Una alternativa es la Llamada a Procedimiento Remoto (*Remote Procedure Call*, RPC). En esta técnica, la comunicación se realiza a nivel de llamada a procedimiento. Mientras el programador tiene la sensación de que realiza una llamada a un procedimiento, la realidad es que los argumentos de la llamada son empaquetados y trasladados a la parte del sistema distribuido en la que se procesan. Los sistemas RPC codifican los argumentos y los valores que devuelve el procedimiento siguiendo una representación de datos externos, como por ejemplo XDR[7].

Sin embargo, la RPC no es válida en los *Sistemas de Objetos Distribuidos,* es decir, sistemas donde las diferentes partes distribuidas por la red son *objetos*, debido al factor de la semántica de la invocación de objetos. Los valores de retorno en RPC siempre es un tipo básico y nunca un objeto.

Como alternativa, actualmente existen cinco tecnologías principales para la comunicación en un sistema de objetos distribuidos: RMI, CORBA, DCOM[8], Agentes Voyager[9] y las EJBs[10]. Debido a su mayor extensión en el desarrollo de las aplicaciones y a su interacción en los campos objeto de estudio de este trabajo, dedicaremos un mayor detalle a las tecnologías RMI y CORBA.

[7] eXternal Data Representation.

[8] *Distributed Common Object Model.* Desarrollada por Microsoft basada en componentes ActiveX. Permite el acceso remoto de objetos por Visual Basic, C ó C++, solamente en el sistema operativo Windows. http://microsoft.com/com/tech/DCOM.asp

[9] Desarrollada por la empresa ObjectSpace. Permite la creación de sistemas de agentes en Java. http://www.recursionsw.com/products/voyager/voyager.asp

[10] *Enterprise Java Beans.* Estándar de Sun para hacer un servidor de aplicaciones en Java. http://java.sun.com/products/ejb/

1.2.1. Invocación Remota de Métodos (RMI)

La invocación remota de métodos (*Remote Invocation Method*, RMI) es una tecnología desarrollada por la empresa *Sun Microsystems* para el lenguaje de programación Java, lo que le dota de la ventaja de la interoperabilidad. Puede considerarse como una versión RPC pero que soporta la orientación a objetos y, por tanto, el paso de objetos.

Las aplicaciones RMI suelen componerse de dos programas separados: el *servidor* y el *cliente*. Una típica aplicación servidor crea un número de objetos remotos, haciendo accesibles referencias a estos objetos y esperando que los clientes invoquen métodos sobre esos objetos remotos. El típico cliente toma una referencia remota a uno o más de los objetos del servidor y entonces invoca métodos sobre él. RMI proporciona el mecanismo a través del cual se comunican el cliente y servidor.

Las aplicaciones de objetos distribuidos necesitan:

- Localizar objetos remotos (objetos que contienen métodos que pueden ser invocados por otras máquinas virtuales): Las aplicaciones pueden emplear uno de los dos mecanismos para obtener referencias a los objetos remotos. Una aplicación puede registrar sus objetos remotos con la utilidad de identificación RMI, el `rmiregistry`, o la aplicación puede pasar y devolver referencias a objetos remotos como parte de su operación normal.
- Comunicarse con objetos remotos: Los detalles de la comunicación entre los objetos remotos son manejados por el RMI. Para el programador, la comunicación remota aparece como una invocación a métodos estándar.
- Cargar los *bytecodes* de las clases para objetos que son pasados como parámetros o valores de retorno: RMI proporciona los mecanismos necesarios para cargar el código de un objeto y para transmitir sus datos.

La Figura 1.1 muestra una aplicación distribuida RMI, que emplea el registro `rmiregistry` para obtener las referencias al objeto remoto. El servidor llama al registro para asociar un nombre con un objeto remoto. El cliente localiza por su nombre al objeto remoto en el registro del servidor y entonces invoca un método sobre él. Esta figura también muestra que el sistema RMI emplean un servidor web existente para cargar los bytecodes de las clases escritas en Java, del servidor al cliente y viceversa, cuando se necesitan los objetos de la otra máquina virtual.

RMI puede cargar los *bytecodes* empleando cualquier protocolo URL soportado por la plataforma Java (por ejemplo, HTTP, FTP...). Una de las principales características de RMI consiste en la posibilidad de descargar código de una clase de un objeto si ésta no se encuentra definida en la máquina del cliente. Pasa los objetos por su verdadero tipo, por tanto el comportamiento del objeto no cambia al ser enviados a otra máquina virtual[11].

1.2.1.1. *Stubs* y *Skeletons*

El RMI trata de forma diferente a un objeto remoto que a aquellos objetos no remotos pasados de una máquina virtual a otra. En vez de hacer una copia de la

[11] dentro de una misma máquina física pueden coexistir diferentes máquinas virtuales

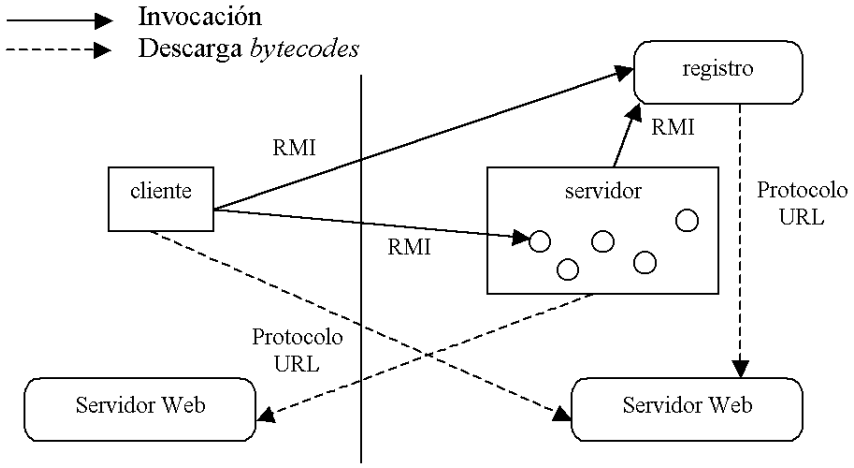

Figura 1.1.: Esquema RMI [WW].

implementación del objeto en la máquina virtual que invoca a uno de sus métodos, el RMI pasa un *Stub* para un objeto remoto, que actúa como la representación local (*proxy*) del objeto remoto que es su *referencia remota*. Lo que a vista del programador es la llamada a un método, es en realidad una invocación a un método de este *Stub*, el cual a su vez, es el responsable de llevar a cabo la llamada al objeto remoto. Solamente los métodos definidos a través de una interfaz remota pueden ser invocados en la máquina virtual que los recibe. Cuando se invoca un método de un *Stub* se producen los siguientes pasos:

1. Inicia una conexión con la máquina virtual que contiene el objeto remoto.
2. Escribe y transmite (*marshalling*) los parámetros a la máquina virtual remota.
3. Espera por el resultado de la invocación del método.
4. Lee (*unmarshalling*) el valor de retorno o la excepción producida.
5. Devuelve el valor al método/objeto que ha invocado el método remoto.

Del mismo modo, en el lado del servidor aparece un elemento llamado *Skeleton*[12], que funciona como *simulador* para recibir parámetros desde el cliente[13]. Cuando esto ocurre se producen los siguientes pasos:

1. Lee los parámetros para el método remoto.
2. Invoca el método correspondiente del objeto remoto, que está en su misma JVM.
3. Escribe y transmite el resultado (ya sea el valor de retorno o alguna excepción que se produzca) al *Stub* de la máquina virtual que invocó el método remoto.

[12] En entornos que implican solamente Java 2, no es necesario emplear los *Skeletons,* debido a que a partir de esa versión se añadió un protocolo adicional que elimina ese requisito.

[13] El empleo de *Stubs* y *Skeletons* es un mecanismo estándar de los RPC.

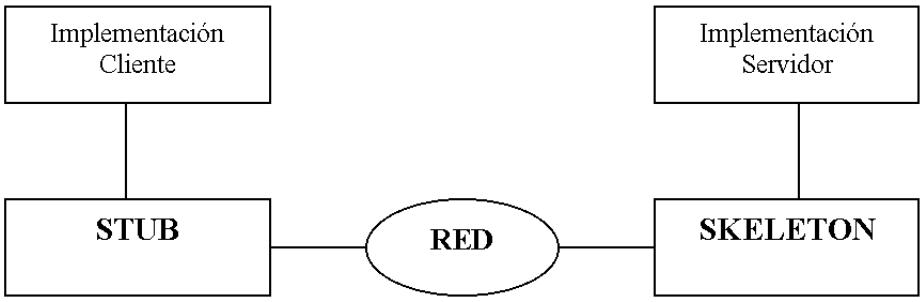

Figura 1.2.: *Stubs* y *Skeletons* en RMI [WW].

La Figura 1.2 ilustra la participación de los *Stubs* y *Skeletons* en la invocación de métodos remotos.

A nivel práctico, se emplea el compilador `rmic`[14] para la generación tanto los *Stubs* como los *Skeletons.* [WW]

1.2.2. Arquitectura de Intermediación de Petición de Objetos Comunes. (CORBA)

CORBA (*Common Object Request Broker Architecture*) es una especificación creada por el OMG (*Object Management Group)*[15] que establece un estándar para la comunicación de objetos a través de procedimientos/métodos remotos. A diferencia de RMI, CORBA no se ve restringido a la utilización del lenguaje de programación Java. Es más, CORBA permite la invocación de un método remoto de un objeto implementado en un lenguaje por otro objeto implementado en un lenguaje diferente. Por otro lado, se trata de un sistema con numerosas especificaciones, lo que lo hace más difícil de aprender que otros.

1.2.2.1. El lenguaje de Definición de Interfaz (IDL)

Para el desarrollo de una aplicación que emplee CORBA se utiliza un lenguaje llamado IDL (*Interface Definition Language*), que como su nombre indica, permite la definición de interfaces, o lo que es lo mismo, las diversas estructuras que se emplean en un ambiente CORBA. Un ejemplo de declaración IDL es la siguiente:

```
module ejemplo {

interface EjemploInterfaz {
        string recogerMensaje();
       };
```

[14] incluido en la distribución Java de *Sun Microsystems*
[15] Comprende más de 700 compañías y organizaciones.

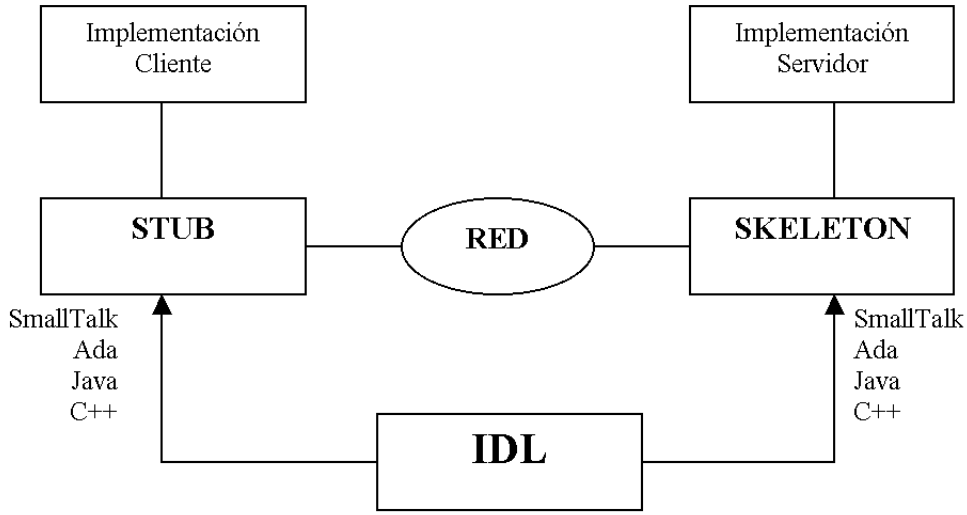

Figura 1.3.: *Stubs* y *Skeletons* en CORBA [Ins99].

```
};
```

Este ejemplo declara una interfaz llamada *EjemploInterfaz* que contiene un método llamado *recogerMensaje.*

1.2.2.2. *Stubs* y *Skeletons*

Una vez que se han definido las diversas interfaces a través del IDL, es necesario completar el proceso mediante el correspondiente lenguaje de implementación. Para ello se hacen uso, como en el caso de RMI de *Stubs* y *Skeletons*, del modo mostrado en la Figura 1.3.

De la misma interfaz IDL se pueden generar varios *Stubs* y *Skeletons* en diversos lenguajes, de modo, que aunque sean generados en diferentes lenguajes podrán comunicarse e invocar sus métodos respectivos.

En la distribución de *Sun Microsystems* del lenguaje Java, estos *Stubs* y *Skeletons* son generados a través de la herramienta `idltojava`.

1.2.2.3. El Intermediario de Petición de Objeto (ORB)

Los *Stubs* y *Skeletons* establecen sus comunicaciones a través del ORB (*Object Request Broker*). Por tanto, es el ORB el elemento clave en la conectividad de un sistema CORBA. Es el servicio distribuido que implementa la petición del objeto, permitiendo la comunicación transparente entre cliente y servidor. Localiza el objeto remoto en la red, comunica la petición al objeto, espera por los resultados y, cuando están disponibles, comunica los resultados al cliente, independientemente del lenguaje de implementación. El papel de este elemento en la arquitectura se muestra en la Figura 1.4.

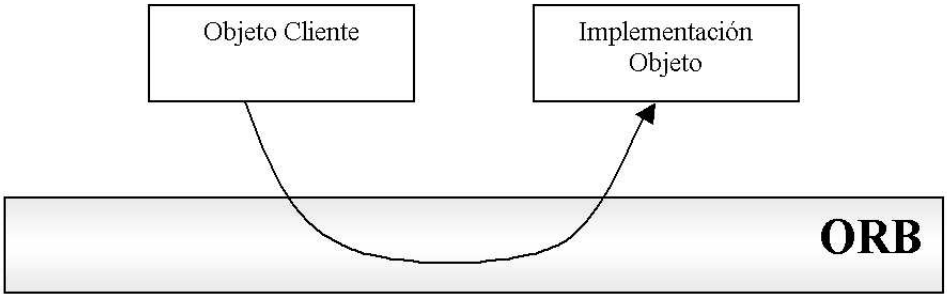

Figura 1.4.: El elemento ORB en CORBA[Ins99].

Este elemento, al igual que el resto, es producto de la estandarización OMG[16]. Esto se traduce en la aparición de diversos ORBs soportados por diferentes empresas[17] y proyectos de código libre, como por ejemplo, *ORBit2*[18]. En este sentido, la versión CORBA 2.0 define un protocolo de red llamado IIOP (*Internet Inter-ORB Protocol)*, que permite a los clientes emplear un producto CORBA de cualquier fabricante para comunicarse con objetos que emplean un producto CORBA de otro fabricante. El protocolo IIOP corre a través de cualquier implementación TCP/IP[19].

1.2.2.4. Servicios CORBA

Otra parte importante del estándar CORBA es la definición de una serie de servicios distribuidos para dar soporte a la integración e interoperación de los objetos distribuidos. Estos servicios son conocidos como *Servicios CORBA* (*CORBA Services*, COS) y están definidos por encima del ORB[20] como objetos CORBA con interfaces IDL.

Los servicios CORBA más significativos son:

Ciclo de vida del objeto Define cómo son creados, eliminados, copiados y se mueven los objetos CORBA. Debido a que los entornos basados en CORBA soportan objetos distribuidos, este servicio define convenciones que permite a los clientes llevar a cabo operaciones referidas al ciclo de vida sobre un objeto en localizaciones diferentes [OMG02a].

Naming Define cómo los objetos CORBA pueden tener unos nombres simbólicos amigables. Sirve para localizar un servicio por su nombre [OMG02b].

[16] OMG solamente define las especificaciones, por lo que a veces se presentan algunas incompatibilidades entre las diversas implementaciones.

[17] Java IDL/ORB de Sun, Visibroker de Borland, Component Broker de IBM. ..

[18] http://orbit-resource.sourceforge.net

[19] Existe una versión de Java RMI, llamada *RMI over IIOP* que emplea el protocolo IIOP.

[20] Actualmente existen definidos 16 servicios CORBA, que se encuentran detallados en http://www.omg.org/technology/documents/formal/corbaservices.htm

Eventos Separa la comunicación entre los objetos distribuidos. Este servicio define dos papeles a jugar por los objetos: suministradores (*supplier*) que producen datos referidos a eventos y consumidores (*consumer*) que procesan esos datos. La comunicación entre ambos tipos de objetos se realiza mediante CORBA [OMG01].

Transacciones Da soporte al concepto de *transacción*. Una transacción es una unidad de trabajo que tiene las características de ser *atómica* (si se interrumpe, se anulan todos sus efectos), producir resultados *consistentes* (los efectos de una transacción preservan las propiedades invariantes), estar *aislada* (sus estados intermedios no son accesibles para otras transacciones) y perdurar (los efectos de una transacción completa son persistentes)[21] [OMG02c].

Propiedad Proporciona soporte a la asociación de pares nombre-valor con objetos CORBA [OMG00b].

Consulta Proporciona soporte de consultas sobre objetos. Estas consultas son basadas en predicados y pueden dar como valor de retorno colecciones de objetos. Pueden especificarse mediante SQL u otros lenguajes de consultas.[Ins99, OMG00c].

[21] debido a las iniciales en inglés de estas propiedades (*atomic*, *consistent*, *isolated*, *durable*), se conocen como ACID

2. Ontologías

El presente capítulo se centra en el estudio del concepto de ontología. Dicho término es de gran importancia en las aplicaciones presentadas en este trabajo, puesto que, tal como se verá en capítulos posteriores, permite dotar a los agentes presentes en los MAS de una mayor racionalidad. Comenzaremos el capítulo presentando el concepto de Web Semántica, cuyo propósito es codificar la información de un modo entendible para las máquinas. Este concepto ha supuesto el desarrollo de nuevas formas, cada vez más completas, de expresión del conocimiento. A continuación reflejaremos las definiciones más habituales para el término *ontología*, así como sus componentes y su clasificación según diversos criterios. Seguiremos con la descripción, siempre de una forma breve y centrándonos en los aspectos más significativos, de diversas iniciativas para aumentar la información extraíble de un documento por una máquina. Debido a su empleo en este trabajo, distinguiremos entre las basadas en lenguajes de marcas y el resto, ya que muchos lenguajes de representación de ontologías emplean los lenguajes de marcas. Dentro de la sección 2.4, dedicada a los lenguajes de marcas, haremos un recorrido por este tipo de lenguajes, presentados en un orden creciente de riqueza semántica. Finalizaremos el capítulo analizando un amplio conjunto de herramientas destinadas a la edición y análisis de ontologías y al procesamiento de documentos codificados en lenguajes de marcas.

2.1. Web Semántica

La World Wide Web contiene una gran cantidad de información que crece día a día. La gran parte de esta información está representada en el lenguaje HTML, diseñado para permitir a los desarrolladores web mostrar la información (no solamente de documentos, sino imágenes o datos) de un modo accesible a las personas mediante la visualización a través de los navegadores de Internet. Mientras que HTML permite visualizar la información, no proporciona demasiada capacidad para describir la información de forma que un programa software pueda interpretarla. Esto provoca que el poder de su utilización se vea condicionado por la habilidad de los usuarios para navegar entre las diversas fuentes de información.

Una solución consiste en añadir a las páginas web descripciones de su contenido, permitiendo de este modo un cierto razonamiento sobre dicho contenido. En este sentido, el *World Wide Web Consortium* (W3C)[1] ha desarrollado el lenguaje de marcas XML que permite presentar la información de un modo más adecuado. Sin embargo, XML tiene una capacidad limitada para describir las relaciones semánticas. Los lenguajes DAML+OIL y OWL son unas extensiones del XML y RDF(un lenguaje más rico desde el punto de vista semántico) para construir ontologías y marcar la información de una forma entendible para las computadoras.

[1] http://www.w3.org/

Con estos lenguajes, en la próxima generación de la Web, denominada frecuentemente como la *web Semántica* (*Semantic Web*), la información no será interpretable únicamente por los lectores humanos, sino procesable por máquinas a través de poderosos motores de búsqueda, servicios de información inteligentes o sitios web personalizables. Esta web semántica requiere de interoperabilidad a nivel semántico, de ahí su nombre. En este sentido los lenguajes descritos intentan proporcionar unos estándares, no solamente a nivel de sintaxis, sino de semántica en el contenido [DMM00].

De un modo más formal, el W3C se refiere del siguiente modo a la web semántica:

> La Web Semántica es el representación de los datos en la World Wide Web. Es un esfuerzo colaborativo liderado por el W3C con participación de un gran número de investigadores y sociedades industriales. Se basa en el RDF, el cual integra una variedad de aplicaciones empleando XML para la sintaxis y URIs[2] para el nombrado.
>
> "La Web Semántica es una extensión de la web actual en la cual la información está dada mediante un significado bien definido, permitiendo una mejor cooperación en el trabajo de computadoras y personas."

> La Web Semántica es una visión: la idea de tener en la Web datos definidos y enlazados de tal forma que puedan ser usados por máquinas, no solamente para propósitos de visualización sino para automatización, integración y reutilización de los datos a través de varias aplicaciones.

2.2. Ontologías: Definición y clasificación

2.2.1. Ontologías y la Web Semántica

Para que los datos sean realmente entendibles por múltiples aplicaciones (ya se relacionen o no con el concepto de Web Semántica) es necesaria la interoperabilidad semántica. La actual interoperabilidad sintáctica consiste simplemente en el procesamiento de los datos de forma correcta. En cambio, la interoperabilidad semántica necesita de un análisis del contenido. Esto se traduce en la necesidad de especificaciones formales y explícitas de los modelos del dominio que define los términos empleados y sus relaciones. Es precisamente a esos modelos formales a los que se denominan *ontologías* [DMM00].

2.2.2. Ontologías y Conceptualización

En el sentido filosófico, nos podemos referir a una ontología como un sistema particular de categorías que explican una cierta visión del mundo. Como tal, este sistema no depende de un lenguaje particular. La ontología es siempre la misma, independientemente del lenguaje empleado. Por otro lado, en Inteligencia Artificial, una ontología se refiere a un artefacto de ingeniería, constituido por un vocabulario específico empleado para describir una cierta realidad, más un conjunto de supuestos explícitos relacionados con el significado planeado para el vocabulario.

[2]Identificador único de recursos. Ver sección 2.4.3.3

Este conjunto de supuestos tiene normalmente la forma de una lógica de primer orden, donde las palabras del vocabulario aparecen como nombres de predicados unitarios o binarios, llamados respectivamente conceptos y relaciones. En el caso más simple, una ontología describe una jerarquía de conceptos emparentados con relaciones subsumidas. En casos más sofisticados, se pueden añadir axiomas apropiados en orden a expresar otras relaciones entre conceptos y para restringir la interpretación deseada.

Las dos lecturas de ontología presentadas están relacionadas entre sí, pero, en orden a distinguirlas, se empleará en este trabajo la definición propia del campo de la Inteligencia Artificial. La otra interpretación será referida como *conceptualización.* Por tanto, según estas interpretaciones, una ontología será la especificación de una conceptualización. Dos ontologías pueden ser diferentes en el vocabulario empleado (por ejemplo mediante el empleo de idiomas diferentes), mientras que pueden compartir la misma conceptualización. La conceptualización se refiere a una visión simplificada del mundo que se desea representar para algún propósito.

Así llegamos a la definición más habitual de *ontología*.

> Una ontología es una especificación explícita de una conceptualización. Esto es, una ontología es una descripción de los conceptos y relaciones que pueden existir para un agente o comunidad de agentes. (Tom Gruber) [Gru93]

Otra definición muy empleada es la siguiente:

> Una ontología es un conjunto estructurado jerárquicamente de términos que describen un dominio, el cual puede ser empleado como un esqueleto inicial para una base de conocimiento. (Swartout, Patil, Knight y Russ) [SPKR96]

Una ontología, por tanto, es ante todo vocabulario. Sin embargo, una ontología únicamente compuesta de vocabulario tendría un uso muy limitado ya que su significado pretendido no sería explícito. Por tanto, además de especificar un vocabulario, una ontología debe especificar el significado pretendido a dicho vocabulario, es decir, su conceptualización subyacente [UG96, FIP01h].

2.2.3. Componentes de una Ontología

Una ontología consiste en varios componentes. Los más importantes son:

Conceptos Son términos abstractos organizados en taxonomías. Los conceptos jerarquizados se unen a través de una relación "es un/una". Por ejemplo, un Elefante *es un* Animal.

Relaciones. Enlazan conceptos no jerarquizados. Ejemplo, un Profesor *trabaja en* un Colegio.

Atributos Relaciones de tipos de datos predefinidos, como por ejemplo *string*, *entero*, *boolean*... Por ejemplo, una Persona tiene un *nombre* (string).

Instancias Existencias concretas de conceptos abstractos. Por ejemplo, *Juan* es instancia de Persona.

Axiomas Reglas que son válidas en el modelo del dominio. Por ejemplo, una *Persona* no puede ser a la vez *Hombre* y *Mujer*.

2.2.4. Clasificación de Ontologías

Las ontologías pueden clasificarse según los siguientes parámetros.

- El grado de dependencia de un dominio o tarea específica.
- El grado de detalle de su axiomatización.
- La naturaleza de su dominio.

2.2.4.1. Clasificación según su dependencia de un dominio o tarea.

Según este criterio, las ontologías se clasifican en:

- Ontologías de alto nivel (*top-level*). Estas ontologías describen conceptos muy generales como espacio, tiempo, materia, objeto, evento, acción... que son independientes de un problema o dominio particular. Parece, por tanto, razonable, al menos en teoría, tener ontologías de alto nivel unificadas para todos los usuarios, aunque esto no ha sido llevado a cabo.
- Ontologías de dominio o tarea. Describen, respectivamente, el vocabulario referido a un dominio genérico (como medicina, automoción...) o a una tarea o actividad (como por ejemplo una transacción económica...), mediante la especialización de los términos que aparecen en una ontología de alto nivel.
- Ontologías de aplicación. Describen conceptos dependientes asímismo de un dominio o tarea particular, que son frecuentemente especializaciones de estas ontologías. Estos conceptos suelen corresponder con papeles jugados por entidades de dominio mientras llevan a cabo una cierta actividad.

Puede ser importante distinguir claramente entre una ontología de aplicación y una base de conocimiento. La diferencia está relacionada con el propósito de una ontología, la cual es una base de conocimiento particular que describe hechos asumidos como ciertos por una comunidad de usuarios, en virtud del acuerdo sobre el significado del vocabulario empleado. Una base de conocimiento genérica, por otro lado, podría también describir hechos o afirmaciones relacionados con un estado particular.

2.2.4.2. Clasificación según su grado de axiomatización.

Una ontología se puede clasificar según su grado de detalle, es decir, el grado de caracterización de los modelos pretendidos. Una ontología de grano fino, muy rica en detalles, tiene un gran poder (y por tanto, suelen tomarse como *referencia*), pero suele ser complicada de diseñar y razonar. Una ontología poco detallada, por otro lado, consistiría en un conjunto mínimo de axiomas escritos en un lenguaje de mínima

expresividad, para dar soporte solamente a un conjunto de servicios específicos. En definitiva, pretende ser *compartida* entre usuarios que están ya de acuerdo en la conceptualización subyacente. Se puede clasificar una ontología, por tanto, según su grado de axiomatización entre *ontologías de referencia* (las de un mayor grado de detalle) y *ontologías compartibles* (las de un menor grado de detalle). Estas ontologías también reciben respectivamente los nombres de ontologías *off-line* (los usuarios solamente acceden de tiempo a tiempo) y *on-line* (dan soporte al núcleo de las funcionalidades del sistema).

2.2.4.3. Ontologías de meta-nivel

Un caso especialmente interesante de ontologías son las llamadas *ontologías de representación.* En realidad son ontologías de meta-nivel que describen una clasificación de las primitivas empleadas por un lenguaje de representación del conocimiento (es decir, conceptos, atributos, relaciones...). Un ejemplo de estas ontologías es la OKBC (ver sección 2.3.3), empleada para dar soporte a traducciones entre diferentes lenguajes de representación del conocimiento [FIP01h].

2.2.5. Principios Útiles para el Desarrollo de Ontologías

Claridad y objetividad La ontología debería contener un glosario del vocabulario empleado para proporcionar definiciones objetivas y significado preciso en forma de lenguaje natural.

Completitud Se prefiere una definición expresada como una condición necesaria y suficiente a una definición parcial.

Coherencia Debe permitir inferencias consistentes con las definiciones.

Máxima extensibilidad La inclusión y especialización de términos en la ontología no debe necesitar la redefinición de los términos existentes.

Principio de distinción ontológica monotóna Las clases con diferente criterio de indentidad deben ser disjuntas [FIP01h].

2.2.6. Diseño y Elaboración de Ontologías

Existen numerosos métodos para la elaboración de ontologías. Sin embargo, de una manera típica, los pasos a seguir se pueden resumir en los siguientes cinco:

1. Adquisición del conocimiento del dominio. Consiste en ensamblar los recursos de información apropiados que definirán, con consistencia y consenso, los términos empleados formalmente para la descripción del dominio de interés. Estas definiciones deben ser recogidas de tal modo que puedan ser expresadas en un lenguaje común elegido para la ontología.

2. Organización de la ontología. Consiste en el diseño de la estructura del dominio. Esto implicará identificar los principales conceptos y propiedades, las relaciones entre los conceptos, crear conceptos abstractos, referenciar o incluir otras ontologías, distinguir los conceptos de las instancias...

3. Desarrollar la ontología. Añadir conceptos, relaciones e individuos hasta el nivel necesario para satisfacer los propósitos de la ontología.

4. Comprobar la ontología. Eliminar las posibles inconsistencias entre los elementos de la ontología. Esta comprobación de la consistencia puede suponer cambios en clasificación de los elementos, añadiendo nuevos conceptos basados en propiedades individuales y relaciones entre clases.

5. Remitir la ontología a expertos en el dominio, como proceso de verificación final.

2.3. Propuestas no Basadas en Lenguajes de Marcas

En esta sección describiremos de forma breve cuatro propuestas significativas para el aumento de la interoperabilidad entre aplicaciones diferentes. Hemos realizado la distinción entre estas propuestas y las basadas en el empleo de lenguajes de marcas, puesto que son estas últimas las empleadas para el desarrollo de las ontologías empleadas en este trabajo, y por ello objeto de una mayor atención en la sección 2.4.

2.3.1. Dublin Core Metadata Initiative, *DCMI*

El DCMI (*Dublin Core Metadata Initiative*) es un fórum abierto dedicado al desarrollo de estándares de metadatos online con la característica de interoperabilidad que dan soporte a un amplio rango de propósitos y modelos. Es un esfuerzo multidisciplinar internacional para el desarrollo de mecanismos para la descripción orientada a descubrimiento de diversos recursos en un entorno electrónico, y por tanto para la Web Semántica.

El DCMI tiene sus orígenes en Chicago en la segunda *Conferencia Internacional sobre la World Wide Web* en Octubre de 1994. En dicha conferencia, Y. Rubinsky[3], S. Weibel y E. Miller[4] mantuvieron una conversación con T. Noreault[5] y J. Hardin[6] acerca de la semántica y la web. Dicha conversación derivó hacia la dificultad de la búsqueda de recursos en la Web (dificultad ya entonces cuando solamente existían alrededor de medio millón de objetos direccionables).

Esto llevó a NCSA y OCLC a organizar un taller en Dublin (Ohio) en marzo de 1995 para discutir acerca de la semántica de los metadatos. En este evento, llamado *OCLC/NCSA Metadata Workshop* más de 50 personas de diversos campos discutieron acerca de cómo un conjunto principal (*core set*) de semánticas para los recursos basados en Web podría ser extremadamente útil para clasificar la Web, haciendo de este modo más fácil la labor de búsqueda de recursos. Esto dio como resultado el *Dublin Core Metadata* en honor al lugar de celebración del taller. En la actualidad engloba más de 800 personas de más de 45 países.

El Dublin Core es un conjunto estándar propuesto en diciembre de 1996 de 15 elementos traducido a unos 25 idiomas. Se espera que los elementos de esa lista

[3] miembro de SoftQuad

[4] ambos miembros de OCLC (*Online Computer Library Center*) http://oclc.org

[5] director de la oficina de investigación de OCLC

[6] director de NCSA (*National Center for Supercomputing Applications*)

y sus nombres no cambien de modo significativo, aunque la aplicación de alguna de ellas es experimental y por tanto sujeta a variaciones en la interpretación de una implementación a otra. Este conjunto de elementos captura una representación de los aspectos esenciales relacionados con la descripción de recursos. En este sentido su objetivo se encuentra relacionados con los de RDF (tal como se verá en la sección 2.4.6). Sin embargo, en contraste RDF, el cual hace pocas suposiciones acerca de la semántica, los elementos de Dublin Core poseen nombres descriptivos que pretender transmitir un significado semántico a los mismos. Estos nombres descriptivos se refieren a:

- El contenido de un recurso[7]:

  Título El nombre dado a un recurso, usualmente por el autor.

  Claves Los tópicos del recurso. De forma típica, expresará las claves o frases que describen el título o contenido del recurso.

  Descripción Descripción textual del recurso.

  Lengua Lengua(s) del contenido intelectual del recurso.

  Relación Identificador de un segundo recurso y su relación con el recurso actual (*isVersionOf*, *isBasedOn*, *isPartOf*, *isFormatOf*)

  Fuente Secuencia de caracteres usados para identificar unívocamente un trabajo del cual proviene el recurso actual.

  Cobertura Característica de cobertura espacial y/o temporal del contenido intelectual del recurso.

- su propiedad intelectual[8]:

  Autor o creador La persona u organización responsable de la creación del contenido intelectual del recurso.

  Editor Entidad responsable de hacer que el recurso se encuentre disponible en la red en su formato actual.

  Colaborador Persona u organización que ha tenido una contribución intelectual significativa en la creación del recurso pero cuyas contribuciones son secundarias en comparación a las personas u organizaciones especificadas en el elemento autor/creador.

  Derechos Referencia para una nota sobre derechos de autor, para un servicio de gestión de derechos o para un servicio que dará información sobre términos y condiciones de acceso a un recurso.

- y de su instanciación[9]:

  Fecha Fecha en la que el recurso se puso a disposición del usuario en su forma actual.

[7] respectivamente se invocan mediante los términos *Title, Subject, Description, Language, Relation, Source, Coverage.*

[8] correspondientes a los términos *Creator, Publisher, Contributor, Rights*

[9] correspondientes a los términos *Date, Type, Format, Identifier*

Tipo Categoría del recurso (fotografía, página personal, diccionario...)

Formato Formato de datos de un recurso, usado para identificar el software y posiblemente el hardware necesario para mostrar el recurso.

Identificador Secuencia de caracteres usados para identificar unívocamente el recurso.

Cada elemento es opcional, pudiendo repetirse y aparecer en cualquier orden.

El espacio de nombres (ver sección 2.4.3.3) del Dublin Core es:

```
http://purl.org/dc/elements/1.0/
```

Actualmente está trabajando sobre una arquitectura de registro para dar soporte al mantenimiento de los documentos DCMI [DCM03, Bec02].

2.3.2. Cyc

La base de conocimiento Cyc, desarrollada por Cycorp[10], consiste en una representación formal de una extensa cantidad de conocimiento humano fundamental: hechos, reglas y heurística para el razonamiento sobre los objetos y eventos de la vida diaria. El medio de representación es un lenguaje formal llamado CycL, el cual es básicamente una ampliación del cálculo de predicados de primer orden. La base de conocimiento consiste en términos y declaraciones que relacionan dichos términos.

De la gran cantidad de términos existentes en esta gran base de conocimiento, se han seleccionado aproximadamente unos 3000 términos que capturan los conceptos más generales de la realidad humana. Estos términos forman la llamada *Upper Cyc Ontology*[11]. El motivo de agrupar estos términos estriba en que este núcleo de la ontología [CyC97] satisface los criterios de:

Universalidad Todo concepto imaginable puede ser enlazado correctamente con la *Upper Cyc Ontology*. Es decir, todo concepto nuevo tendrá algunos conceptos relacionados en esta ontología.

Articulación Las distinciones realizadas en la ontología son a la vez necesarias y suficientes para la mayoría de los propósitos. Por *necesarias* se quiere expresar que existe justificación tanto teórica como práctica para cada colección de objetos, cada predicado, cada función y cada individuo. Por *suficientes* se quiere expresar que se han incluido suficientes distinciones para permitir y dar soporte al compartimiento del conocimiento, reducción de la ambigüedad en lenguaje natural e integración de bases de datos.

Los conceptos incluidos en Cyc se distribuyen en 43 grupos estándar (Matemáticas, Agentes, Actores, Comida, Organizaciones, Geografía, Tiempo, Finanzas...) que se pueden consultar a través de su página web. Cada descripción de un concepto se compone de:

[10] http://www.cyc.com/

[11] http://www.cyc.com/cyc-2-1/cover.html

- Su nombre Cyc.
- Un comentario en Inglés sobre el significado pretendido y el uso del concepto.
- Una serie de enlaces empleados para ordenar e interconectar los conceptos.

Cada concepto se representa como una constante[12] Cyc. El nombre de toda constante empieza con los caracteres #$, por ejemplo #$Skin[13]. Cada constante puede representar una colección (por ejemplo el conjunto de todas las personas), un objeto individual (como una persona particular), una palabra en lenguaje natural, un cuantificador, una relación...

Como ejemplo, la entrada para #Skin, dentro de la página referida a Fisiología es:

#$Skin Una (un pedazo de) piel sirve como protector externo y sensor táctil para (una parte de) el cuerpo de un animal. Esta es la colección de todos los trozos de piel. Algunos ejemplos incluyen #$TheGoldenFleece[14] (referido a toda la piel de un animal) y (#$BodyPartFn #$YulBrynner #$Scalp)[15] (representando una pequeña porción de su piel).
isa: #$AnimalBodyPartType
genls: #$BiologicalLivingObject #$AnimalBodyPart #$SheetOfSomeStuff #$VibrationThroughAMediumSensor #$TactileSensor.

Tras el nombre y la definición, se incluye información de la jerarquía que ocupa el concepto. Toda constante Cyc es un elemento de una o más colecciones. Como consecuencia cada definición incluye una línea **isa**, que informa sobre algunas de las colecciones a las cuales pertenece el concepto. Para evitar confusiones, se incluye únicamente una lista no redundante de colecciones. En el caso de #$Skin, solamente es necesario incluir en esta línea #$AnimalBodyPartType (referido al tipo de parte del cuerpo de un animal).

Por otro lado, el término #$Skin es en sí mismo una colección de objetos, por lo que pueden existir otros términos referidos a sus subconjuntos, superconjuntos y elementos. La línea **gentl** proporciona una lista de los superconjuntos del concepto descrito. En el caso de #$Skin esto corresponde, entre otros términos a #$BiologicalLivingObject (objeto biológico vivo) y #$AnimalBodyPart (parte del cuerpo de un animal).

Un término importante es el de #$Collection. Cada #$Collection es un concepto que representa un conjunto o clase de cosas, con algunas propiedades en común.

A continuación, y como complemento de la descripción de la ontología Cyc, reproducimos a continuación sendos extractos de la definición de los términos #$Agent y #$Person, ambos contenidos en el grupo de definiciones reunidas bajo el epígrafe "Vocabulario de Agentes" (la jerarquía de términos se muestra en la Figura 2.1).

[12] también llamada término o unidad
[13] término referido a *piel*. Se ha preferido mantener la grafía inglesa para respetar el formato original de la ontología.
[14] correspondiente al Vellocino de Oro
[15] referido al cuero cabelludo de Yul Brynner

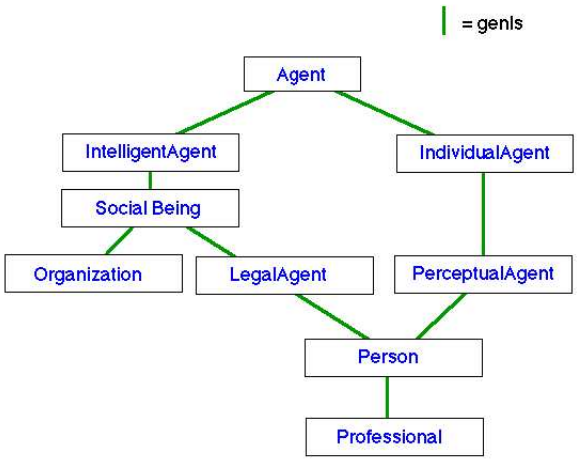

Figura 2.1.: Jerarquía de términos Cyc en Vocabulario de Agentes.

#$Agent Un agente es algo que puede mostrar acción independiente, ya sea consciente o no. #$Agent representa la colección de todos los agentes. La mayoría de los animales son considerados agentes, en la mayoría de los contextos, ya que son en su mayor parte organizaciones. La mayor parte de las plantas no son agentes, en la mayoría de los contextos. Los dispositivos inanimados a veces son considerados agentes, en ciertos contextos. Este es uno de esos conceptos en los que es importante, aunque muy duro, definir de modo preciso. (...)

#$Person La colección de todos los seres humanos. La colección #$Person constituye las especies Homo Sapiens; por tanto #$Person es una instancia de #$BiologicalSpecies en #$BiologyMt. (...) Las personas constituyen el subconjunto más inteligente de #$Primate, y es la única clase cuyos elementos son capaces de hablar un lenguaje de forma natural. Toda actividad cultural requiere de la participación de personas. #$Person excluye personas legales no humanas.

Relaciones Cyc: Predicados y Funciones

Mediante el término #$Relation se referencia el conjunto de todas las relaciones, incluyendo todos los predicados y funciones. Cada definición de una relación Cyc incluye el tipo de sus argumentos.

Un predicado puede ser visto como una función que devuelve los valores de verdadero o falso. Principalmente se emplean para la construcción de declaraciones Cyc. Por ejemplo, la siguiente definición corresponde al predicado *madre*.

#$mother : <Animal><FemaleAnimal>** (#$mother ANIM FEM) significa que el #$FemaleAnimal FEM es el progenitor hembra biológico del #$Animal ANIM. **isa: #$FamilyRelationSlot #$BinaryPredicate

Este predicado se puede emplear es expresiones del estilo

```
(#$mother #$Zeus #$Rea)
```

la cual indica que Rea es la madre de Zeus, o en relaciones de inferencia generales, por ejemplo

```
(#$forAll ?x
 (#$forAll ?y
  (#$implies
   (#$mother ?x ?y)
   (#$startsDuring ?y ?x))))
```

Esta relación indica que si *y* es la madre de *x*, entonces *y* nació antes que *x*, y que probablemente *y* estaba todavía viva cuando *x* nació.

Las funciones Cyc son similares sintácticamente a los predicados, pero con la diferencia de que las funciones pueden devolver cualquier clase de valor: números, palabras... Como ejemplo, la siguiente función devuelve el valor femenino del argumento de entrada.

**#$FemaleFn : <OrganismClassificationType>** (#$FemaleFn TYPE) devuelve una colección que es la intersección de TYPE y #$FemaleAnimal. Por ejemplo, (#$FemaleFn #$Person) es #$FemalePerson, y (#$FemaleFn #$Deer) es #$Doe[16]. Obsérvese que esta función opera sobre, y devuelve como valor, colecciones Cyc, no palabras inglesas (...).

2.3.3. Open Knowledge Base Connectivity (OKBC)

El *Open Knowledge Base Connectivity* (OKBC) es una API para acceder a bases de conocimiento almacenadas en sistemas de representación de conocimiento (*knowledge representation systems*, KRS). Está siendo desarrollado por el programa HPKB (*High Performance Knowledge Base*) del DARPA (*Defense Advanced Research Projects Agency*) y está siendo empleando como un protocolo inicial para la integración de varios componentes tecnológicos.

OKBC proporciona un modelo uniforme de los KRSs, llamado modelo de conocimiento OKBC, basado en una conceptualización común de clases, individuos, relaciones (*slots*), facetas (*facets*) y herencia. OKBC se define de un modo independiente del lenguaje de programación, existiendo implementaciones en Common Lisp, Java y C. El protocolo da soporte de un modo transparente a su integración en redes, tanto como acceso directo a KRSs y a bases de conocimiento.

OKBC consiste en un conjunto de operaciones que proporcionan una interfaz genérica para KRSs subyacentes. Esta interfaz aísla a una aplicación de muchas de las idiosincrasias de una KRS específica y habilita el desarrollo de herramientas (por ejemplo editores, navegadores gráficos, herramientas de análisis...) que operen en varias KRSs. Ha sido empleada en éxito en varios proyectos.

[16] El valor femenino del ciervo es la cierva

OKBC soporta una representación orientada a objetos del conocimiento y proporciona un conjunto de construcciones figurativas empleadas comúnmente en las KRSs. El conocimiento *obtenido del/proporcionado a* una KRS a través de OKBC se asume en la especificación de las operaciones OKBC para ser representado en el modelo de conocimiento. Por tanto, el modelo de conocimiento OKBC sirve de *interlingua* implícita para el conocimiento comunicado a través de OKBC y para sistemas que empleen OKBC, para traducir el conocimiento en esa interlingua.

2.3.3.1. Breve Resumen de Sintaxis OKBC

En este subapartado describiremos brevemente algunos aspectos de la sintaxis OKBC, base de algunos de los apartados siguientes. No entraremos en demasiados detalles sobre esta sintaxis, haciendo hincapié en aquellos aspectos que faciliten al lector la lectura de las posteriores secciones [FIP01h].

Un marco (*frame*) es un objeto primitivo que representa una entidad en el dominio del discurso.

Un marco tiene asociado un conjunto de slots (llamados *own slots).* Un *slot* es una relación binaria, de tal modo que cada valor `V` de un slot `S` de un marco F representa la afirmación de que la relación `S` es válida para las entidades representadas por `F` y `V`.

```
(S F V)
```

Cada slot tiene asociado un conjunto de facetas (*own facets*). Cada faceta de un slot de un marco tiene asociado a él un conjunto de valores (*facet values*). Una faceta es una relación ternaria. Cada valor `V` de una faceta `Fa` de un slot `S` de un marco `Fr` representa la afirmación de que la relación `Fa` es válida para la relación `S`, la entidad representada por `Fr` y la entidad representada por `V`.

```
(Fa S Fr V)
```

Una clase es un conjunto de entidades. Cada una de las entidades dentro de esa clase es una instancia de la clase. Una entidad puede ser una instancia de varias clases, que son sus tipos. Una clase que tenga instancias que a su vez sean clases se llama metaclase.

Las entidades que no son clases se denominan individuos. Por tanto, el dominio del discurso consiste en clases e individuos.

Las relaciones *class* e *individual* son ciertas si y sólo si el argumento es una clase o un individuo. Esto da lugar al siguiente axioma, que indica que una variable `X` (las variables se representan precedidas del signo de interrogación) es una clase si y sólo si (<=>) no son individuos.:

```
( <=>  (class ?X) (not (individual ?X)))
```

Se definen las relaciones de *instance-of* (instancia de, cierta cuando `?I` es una instancia de la clase `?C`) y su inversa *type-of* (`?C` es el tipo de la instancia `?I`).

```
( <=>  (type-of ?C ?I) (instance-of ?I ?C)))
```

Relación de subclase (la de superclase se define como su inversa). Para toda instancia `?I` de la subclase `?Csub`, `?I` también es instancia de la superclase `?Csup`.

```
(<=>(subclass-of ?Csub ?Csup)
   (forall ?I (=>(instance-of ?I ?Csub)
     (instance-of ?I ?Csub))))
```

A partir de estas relaciones, son definidas las correspondientes a los slots, facetas, clases primitivas, marcos así como elementos estándares. En este sentido, algunas clases estándares son:

`:THING` es la raíz de la jerarquía de clases para una base de conocimiento. Es decir `:THING` es la superclase de todas las clases de todas las bases de conocimiento.
`:CLASS` es la clase de todas las clases. Esto es, cada entidad que es una clase es una instancia de `:CLASS`.
`:INDIVIDUAL` es la clase de todas las instancias que no son clases.
`:NUMBER`, `:INTEGER`, `:STRING` son las clases, respectivamente de los números, enteros y cadenas de caracteres.

2.3.4. Formato de Intercambio de Conocimiento (KIF)

El Formato de Intercambio de Conocimiento (*Knowledge Interchange Format*, KIF) es un lenguaje formal para el *intercambio* de conocimiento entre programas dispares. KIF no pretende ser un lenguaje que sirva para la interacción con los usuarios humanos (esa interacción se realiza de cualquier otro modo adecuado, por ejemplo a través de interfaces gráficas). Tampoco pretende ser una representación interna del conocimiento en los programas, aunque también puede ser empleado para este propósito. De una forma habitual, cuando un programa lee una base de conocimiento en KIF, ésta se traduce a un formato característico de dicho programa. De un modo análogo, la representación interna se traduce a KIF cuando se desea comunicarse con otros programas.

KIF presenta las siguientes características esenciales:

- El lenguaje tiene semántica declarativa. Es posible comprender el significado de las expresiones en el lenguaje sin necesidad de recurrir a un intérprete para manipular las expresiones.
- Proporciona soporte para la expresión de sentencias en cálculo de predicados.
- El lenguaje proporciona soporte para la representación del conocimiento sobre la representación del conocimiento. Esto permite explicitar todas las decisiones de representación del conocimiento e introducir nuevas construcciones de representación de conocimiento sin necesidad de realizar cambios en el lenguaje.

Simultáneamente, KIF trata de maximizar las siguientes características:

- Capacidad de traducción entre KIF y bases de conocimiento.
- Legibilidad. Aunque ya se ha comentado que KIF no pretende ser un lenguaje que sirva para la interacción con los usuarios humanos, esta legibilidad facilitaría su empleo.
- Empleo como lenguaje de representación. Aunque tampoco pretende ser una representación interna del conocimiento puede ser empleado como tal.

KIF es un lenguaje con una alta expresividad. Sin embargo presenta las principales desventajas[17] de:

- La labor de construir sistemas que soporten todas las características KIF es complicada .
- Los sistemas resultantes tienden a ser más complejos y en algunos casos menos eficientes que otros sistemas que emplean lenguajes menos restrictivos.

En el resto de la subsección describiremos brevemente este formato de intercambio de conocimiento poniendo de manifiesto las características de KIF, sin entrar en demasiado detalle. El lector interesado en un análisis más profundo puede consultar [GF92].

2.3.4.1. KIF Lineal y KIF estructurado

KIF tiene dos variedades. En el KIF *lineal*, todas las expresiones son cadenas de caracteres ASCII, siendo adecuado su uso para el almacenamiento en dispositivos (como por ejemplo disquetes) y su transmisión serial. Por otra parte, en el KIF *estructurado*, las expresiones "legales" son objetos estructurados, siendo su uso adecuado en la comunicación entre programas que operan en el mismo espacio de direcciones.

KIF Lineal

El alfabeto del KIF lineal consiste en 128 caracteres ASCII. KIF se originó en una aplicación Lisp, del que hereda su sintaxis. Así, una cadena de caracteres forma una expresión legal KIF si y sólo si

1. Es aceptable para un lector de Common Lisp.
2. La estructura producida por el lector de Common Lisp es una expresión legal de KIF estructurado.

KIF Estructurado

En el KIF estructurado, la noción de palabra (*word*) se toma como primitiva. Una expresión (*expression*) puede ser una palabra o una secuencia de expresiones. En [GF92] se especifica el resto de términos de la sintaxis del KIF estructurado: variable (*variable*, una palabra cuyo primer caracter es ? ó @), operadores, constantes...

[17] Estas desventajas se han tratado de paliar mediante la aprobación por parte del comité KIF en otoño de 1997 de una extensión de la especificación básica del lenguaje.

2.3.4.2. Base

La base de la semántica KIF es una conceptualización del mundo en términos de objetos y relaciones entre esos objetos.

El universo del discurso es un conjunto de todos los objetos que supuestamente existen en el mundo. Estos objetos pueden ser concretos, abstractos, primitivos, compuestos, ficticios... Los objetos básicos que deben aparecer en todos los universos del discurso son:

- Todos los números reales y complejos.
- Todas los caracteres ASCII.
- Todos las cadenas de caracteres finitas ASCII.
- Palabras.
- Listas finitas de objetos.
- *bottom* - un objeto particular que aparece como el valor de retorno de una función, cuando dicha función es evaluada con argumentos que no tienen sentido.

En KIF, las interacciones entre objetos tienen lugar en forma de relaciones. Formalmente una relación se define como un conjunto arbitrario de listas finitas de objetos. Por ejemplo, la relación < con la lista <2,3> indica que el número 2 es menor que 3.

Una función es una clase especial de relación. Para cada secuencia finita de objetos (los *argumentos*), una función asocia un único objeto llamado *valor*. En cada lista, los elementos iniciales son los argumentos, mientras que el final es el valor. Ejemplo: la función +1 contiene la lista <2,3>, indicando que 2+1=3.

Términos funcionales

Para indicar cómo se calcula el valor de un término funcional, consideremos el siguiente ejemplo: `(+ 2 3)`. El valor se obtiene aplicando la función correspondiente (en este caso el de suma) a los argumentos 2 y 3. Por lo tanto, el valor de la expresión es 5.

2.3.4.3. Lógica

Términos lógicos

`if` El valor del término es el valor del primer término que siga a la primera expresión que sea verdadera en la lista de argumentos. Si ninguna de las expresiones de la lista es verdadera y existe un término aislado al final, el valor del término es el de éste. Si no existe ninguna expresión verdadera ni término aislado, el valor es `bottom`. Ejemplos:

- `(if (> 1 2) 1 (> 2 1) 2 0)` tiene como valor 2 (es el valor que sigue al término que indica que 2 es mayor que 1).

- `(if (> 1 2) 3 4)` tiene como valor 4
- `(if (> 1 2) 4)` tiene como valor `bottom`.

`cond` Análogo al `if` pero sin lo referente a un término aislado al final del término.

Definiciones

Los operadores de definición permiten declarar las expresiones que son ciertas por definición. Estas definiciones tienen *contenido*, expresiones que permiten derivar otras expresiones como conclusiones.

Para definir objetos se emplea el operador `defobject`. Sus formas legales KIF son las siguientes:

```
(defobject s := t)
(= s t)

(defobject s p1 ... pn)
(and p1 ... pn)

(defobject s :-> :=> p)
(=> (= s v) p)

(defobject s :-> :<= p)
(<= (= s v) p)
```

De un modo análogo se definen las funciones y las relaciones a través de los operadores `deffunction` y `defrelation`. Ejemplos de formas legales KIF son:

```
(deffunction f (v1 ... vn) := t)
(= (f v1 ... vn) t)

(defrelation r (v1 ... vn) := p)
(<=> (r v1 ... vn) p)
```

Así, por ejemplo, se puede expresar en KIF que los números naturales (expresados en KIF como `natural`) son aquellos enteros mayores o iguales que 0.

```
(defrelation natural (?x) :=
  (and (integer ?x) (>= ?x 0)))
```

2.3.4.4. Metaconocimiento

En la formalización del conocimiento del conocimiento, se emplea una conceptualización en la que las expresiones son tratadas como objetos en el universo del discurso y en los cuales hay funciones y relaciones adecuadas para esos objetos.

Referencias a expresiones

Para referirse a una expresión se emplea el operador `quote` ó `'`. Así, por ejemplo, para referirse que el individuo llamado `juan` cree que la luna está hecha de queso:

```
(believes juan '(material luna queso))
```

También se pueden incluir variables que aportan información. Por ejemplo, la siguiente expresión indica que el individuo llamado `francisco` cree todo lo que cree el individuo `juan`. Esto, unido con la expresión anterior, conduce a que el individuo `francisco` también cree que la luna está hecha de queso.

```
(=> (believes juan ?p) (believes francisco ?p))
```

Este tipo de expresiones se puede restringir empleando los símbolos `^` y `,` . Así, por ejemplo, podemos indicar que el individuo `francisco` cree únicamente lo mismo que el individuo `juan` solamente en aquello referido a las relaciones representadas por `material`.

```
(=> (believes juan ^(material ,?x ,?y))
    (believes francisco ^(material ,?x ,?y)))
```

Tipos de expresiones

KIF incluye una serie de relaciones de términos. Así, los siguientes axiomas indican que `v` es una variable individual, `s` es una variable de secuencia y `w` es una palabra.

```
(indvar (quote v))
(seqvar (quote s))
(word (quote w))
```

Para denotar al objeto referenciado por un nombre se emplea el término `denotation`.

```
(= (denotation 't) t)
```

De forma análoga, el término `name` devuelve el nombre con que se denota el objeto.

```
(= (name t) (quote t))
```

2.4. Lenguajes de marcas

Históricamente el término *marca* ha sido empleado para describir una anotación que intenta señalar a un tipógrafo cómo debe imprimirse o prepararse un texto determinado. Ejemplos de estas marcas pueden ser determinados símbolos especiales para indicar que una parte del texto debe omitirse o ser impresa en una fuente particular. Hoy en día, en plena era electrónica, el término se ha mantenido para

referirse a toda clase de códigos insertados en textos electrónicos que determinan el formato, el modo de impresión o cualquier otro proceso. Así, si por ejemplo, deseamos que se imprima el texto

La palabra **negrita** está en negrita

un posible modo de indicarlo a través de marcas sería el siguiente:

La palabra MARCA_INICIO_NEGRITA negrita MARCA_FINAL_NEGRITA está en negrita

De un modo más formal, se define *marca* como cualquier elemento que hace explícita una interpretación de un texto. En un sentido amplio, todo texto impreso se organiza de este modo. Los signos de puntuación, mayúsculas, los espacios entre palabras... pueden ser interpretados como marcas, ya que ayudan al lector a determinar aspectos como cuándo termina una palabra o frase, identificar nombres propios... De modo análogo, en el caso de un procesador de textos, se trata de explicitar lo implícito del texto, un proceso que indica al usuario cómo debe interpretarse el contenido del texto. En este sentido, los lenguajes de marcas constituyen un elemento clave para el desarrollo de aspectos tan diversos como la Web Semántica, la representación del conocimiento o el almacenamiento de datos.

Por *lenguaje de marcas*, se entiende una serie de convenciones sobre marcas empleadas de manera conjunta para la codificación de textos. Un lenguaje de marcas debe especificar [SMB99]

- qué marcas están permitidas,
- qué marcas son necesarias,
- cómo se distingue la marca del texto en sí, y
- el significado de la marca.

En los siguientes subapartados se pretende describir brevemente y de modo lógico los principales lenguajes de marcas, centrándonos en sus respectivas capacidades de representación del conocimiento y riqueza semántica. Nuestro recorrido comenzará en el SGML, el lenguaje de marcas más general. Este lenguaje dio origen a su vez al HTML, el lenguaje de la representación de la World Wide Web. La idea inicial del HTML era la de representación del contenido de un documento (título, contenido, enlaces...), pero derivó a un lenguaje de descripción del modo de visualización de documentos (negrita, cursiva...). Ante la necesidad de un verdadero lenguaje de marcas para la representación de datos, el W3C desarrolló el XML, ampliamente utilizado en la actualidad. Otro lenguaje, el SHOE intentó remediar la arbitrariedad en el etiquetado del XML.

Una vez que el XML (así como su forma alternativa a las DTDs, el XML Schema o XLMS) fue ampliamente aceptado como estándar de representación de datos, se pretendió aprovechar la fuerza de los lenguajes basados en marcas para ir un paso más allá: la representación del conocimiento al estilo de OKBC, es decir mediante

clases y relaciones. Esto supondría un gran paso en el desarrollo de la interoperabilidad entre aplicaciones y máquinas, y por tanto de la Web Semántica. Un intento de este desarrollo es el lenguaje XOL, cuya aceptación universal fue limitada. En esta línea, el W3C desarrolló el RDF (con su definición asociada de términos, el RDF Schema o RDFS) sobre la base del XML añadiéndole una mayor riqueza semántica. El DARPA, basándose en este nuevo lenguaje RDF y en OIL (no propiamente un lenguaje de marcas, pero que cuya descripción hemos incluido en esta sección por una cuestión de secuenciación lógica) elaboró la especificación de un lenguaje de marcas de una mayor expresividad: el DAML+OIL. Finalizaremos este recorrido por los lenguajes de marcas con el OWL, de mayor expresividad aún que los anteriores, pero que se enfrenta con la dificultad de la implementación en herramientas que ofrezcan un soporte completo de este lenguaje.

En la Figura 2.2 se muestra la evolución de la riqueza semántica de los principales lenguajes de marcas empleados para la representación de datos.

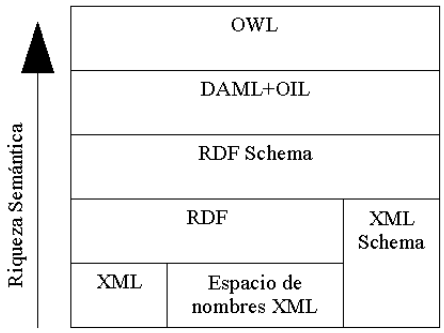

Figura 2.2.: Evolución de la riqueza semántica en los lenguajes de marcas.

2.4.1. Standard Generalized Markup Language, SGML

El *Standard Generalized Markup Language* (SGML) es un lenguaje de marcas definido en el estándar ISO 8879:1986. Las tres características principales que distinguen el SGML de otros lenguajes de marcas son:

1. Hace énfasis en marcas descriptivas más que en las procedurales. Un sistema de marcas descriptivas emplea códigos que simplemente proporcionan nombres para catalogar diversas partes del documento. Así, por ejemplo, una marca como `<parrafo>` puede servir para indicar "que la parte siguiente del texto es un párrafo". Por otro lado, un sistema de marcas procedurales define qué procedimiento será llevado a cabo en ciertos puntos particulares del documento, como por ejemplo, "llamada al procedimiento *parrafo* con parámetros 1 y b". Al escoger las marcas descriptivas, un mismo documento puede ser interpretado de forma diferente por programas diferentes, asignando instrucciones de procesamiento diferentes. Por ejemplo, un programa podría extraer de un

documento una serie de nombres y direcciones para crear una base de datos, mientras que otro programa podría dedicarse a imprimirlos en una tabla.

2. Su concepto de *tipo de documento*. El SGML introduce una *definición de tipo de documento* (*Document Type Definition,* DTD). Los documentos presentan tipos, como otros objetos procesados por un computador. El tipo de un documento se define formalmente por sus partes constituyentes y su estructura. La definición de un informe, por ejemplo, podría consistir en un título y posiblemente un autor, seguido de un resumen y una secuencia de uno o más párrafos. Por tanto, un documento sin título no puede formalmente ser un informe. Si el tipo del documento es conocido, un programa especial, llamado *parser,* puede ser empleado para procesarlo, para asegurarse de que aparezcan correctamente todos los elementos que definen su tipo.

3. Su independencia de cualquier sistema que represente la codificación en el que se escribe un texto. De este modo se pretende que un documento codificado en SGML pueda ser procesado por diferentes entornos hardware/software sin pérdida de información.

2.4.1.1. Estructuras SGML

Elementos

El término técnico empleado en el estándar SGML para una unidad de texto, vista como un componente estructural es el de *elemento*. Diferentes tipos de elementos reciben diferentes nombres. Sin embargo, SGML no proporciona ninguna forma particular para asignar esos nombres, pero sí su posible relación con otros elementos. Es decir, se puede definir un elemento llamado, por ejemplo, `<furgusto>` que puede aparecer (o no) con elementos del tipo `<suvincia>`, que a su vez, puede descomponerse (o no) en elementos del tipo `<leminoncio>`. Se debe resaltar que el SGML no se ocupa de la semántica de los elementos; este aspecto corre a cargo de la aplicación que lo procesa.

Cada elemento de un documento debe estar de alguna forma marcado o etiquetado explícitamente. El estándar proporciona una variedad de formas en las que se puede etiquetar o marcar, siendo la más empleada la de insertar una etiqueta al principio del elemento (marca inicial, *start-tag)* y otra al final (marca final, *end-tag*). El par formado por estas etiquetas sirve para enmarcar los elementos en el texto, de la misma forma que se emplean los paréntesis y las comillas en la escritura convencional. Un ejemplo de etiquetado es el siguiente:

César dijo aquello de `<comillas>` La suerte está echada `</comillas>`

Como se aprecia, la marca inicial es de la forma `<etiqueta>`, donde los símbolos < y > indican respectivamente el inicio y final de la etiqueta. La marca final presenta la misma estructura, salvo el símbolo / después de <. Éste es el modo más empleado para el etiquetado, pudiendo ser sustituido por guiones (-) o cualquier otro carácter adecuado.

Modelos de contenido.

Un elemento puede estar vacío, esto es, no tener ningún contenido, o por el contrario podría contener un texto. Sin embargo, lo más normal es que unos elementos (llamados elementos *hijo)* aparezcan embebidos, o sea, contenidos dentro de otro elemento (llamado elemento *padre*). Precisamente esta es una de las razones que justifican la introducción de las marcas.

Consideremos un ejemplo simple de modelo de estructura, adaptado de [SMB99]. Supongamos que estamos interesados en realizar una antología de un conjunto de poemas, sus títulos, estrofas y versos. En términos SGML, el tipo de documento puede ser llamado por ejemplo *antologia* y consistiría en una serie de elementos *poema.* Cada uno de estos poemas contiene embebidos un elemento *título* y varios elementos *estrofa,* cada uno de los cuales tiene embebidos a su vez varios elementos *verso.* Una posible codificación[18] para esta antología[19] sería la siguiente:

```
<antologia>
  <poema>
    <titulo>Soneto de Repente</titulo>
    <estrofa>
      <verso>Un soneto me manda hacer Violante,</verso>
      <verso>que en mi vida me he visto en tanto aprieto;</verso>
      <verso>catorce versos dicen que es soneto,</verso>
      <verso>burla burlando van los tres delante.</verso>
    </estrofa>
    <estrofa>
      <verso>Yo pensé que no hallara consonante</verso>
      <verso>y estoy a la mitad de otro cuarteto,</verso>
      <verso>mas si me veo en el primer terceto,</verso>
      <verso>no hay cosa en los cuartetos que me espante</verso>
    </estrofa>
    <!-- aquí va el resto de estrofas del poema -->
  </poema>
  <!-- aquí va el resto de poemas -->
</antologia>
```

Una línea del estilo

```
<!-- aquí va el resto de poemas -->
```

es un comentario SGML y no será tratado como parte de la estructura.

El ejemplo visto no hace suposiciones acerca de las reglas que determinan, por ejemplo, si un título puede aparecer en otro sitio que no sea antes de la primera estrofa o si un verso puede aparecer fuera de una estrofa. Este es el motivo por el que aparecen todas esas marcas. El principio y final de cada elemento debe ser marcado

[18] meramente para fines ilustrativos de la estructura, posteriormente se retomará este ejemplo para adaptarlo a las especificaciones SGML

[19] Esta antología presenta un poema de Lope de Vega.

explícitamente, al no existir reglas identificables sobre esos elementos y sobre dónde deben aparecer. En la práctica esas reglas son formuladas para reducir el número de etiquetas necesarias. En el caso que nos ocupa, podemos contemplar las siguientes reglas:

1. Una antología contiene un conjunto de poemas y nada más.
2. Un poema tiene siempre un único título que precede a la primera estrofa y no contiene otros elementos.
3. Aparte del título, un poema consiste únicamente en estrofas.
4. Las estrofas tienen una serie de versos y cada verso aparece únicamente en una estrofa.
5. Nada que no sea otra estrofa o el final de un poema puede aparecer tras una estrofa.
6. Nada puede seguir a un verso, salvo otro verso o el comienzo de otra estrofa.

De estas reglas, se deduce que no son necesarias de forma explícita las marcas finales de las estrofas y de los versos. De la regla 2 se deduce que tampoco necesitamos la marca de final del elemento *titulo* ya que está implícito en el comienzo de la primera estrofa. Tampoco se necesita la marca final del elemento *poema*, ya que está implícito en el comienzo de un nuevo poema o en el final de la antología. Con estas consideraciones el documento queda como sigue:

```
<antologia>
  <poema>
    <titulo>Soneto de Repente
    <estrofa>
      <verso>Un soneto me manda hacer Violante,
      <verso>que en mi vida me he visto en tanto aprieto;
      <verso>catorce versos dicen que es soneto,
      <verso>burla burlando van los tres delante.
    <estrofa>
      <verso>Yo pensé que no hallara consonante
      <verso>y estoy a la mitad de otro cuarteto,
      <verso>mas si me veo en el primer terceto,
      <verso>no hay cosa en los cuartetos que me espante
    <!-- aquí va el resto de estrofas del poema -->
  <poema>
  <!-- aquí va el resto de poemas -->
</antologia>
```

La posibilidad de definir estas reglas y su utilización en el procesamiento de documentos constituye uno de los elementos claves del SGML, tal como veremos en el siguiente apartado.

Definición de las Estructuras SGML: DTDs

Las DTDs constituyen las definiciones formales de las reglas que rigen los documentos SGML. La existencia de estas reglas se justifican por una parte en el propósito de homogeneizar los documentos de un mismo tipo y por otro lado en un procesamiento más efectivo.

Para el caso de la antología de poemas del apartado anterior, una posible DTD podría ser la siguiente:

```
<!ELEMENT antologia    - -  (poema+)>
<!ELEMENT poema        - O  (titulo?, estrofa+)>
<!ELEMENT titulo       - O  (#PCDATA)>
<!ELEMENT estrofa      - O  (verso+)>
<!ELEMENT verso        - O  (#PCDATA)>
```

Cada una de estas líneas constituye una *declaración* y aparece entre los símbolos <>. Cada declaración está formada por tres partes separadas por uno o varios espacios en blanco.

- La primera de ellas empieza con la palabra `ELEMENT` para indicar que se declara un elemento, seguida del nombre o conjunto de nombres de los elementos declarados.
- La segunda consiste en dos caracteres, relativos a las etiquetas inicial y final del elemento, separados por un espacio en blanco que especifican las reglas de minimización. El guión `-` indica que la etiqueta correspondiente debe aparecer siempre, mientras que la letra `O` indica que la etiqueta se puede omitir. En este ejemplo, todos los elementos deben tener una etiqueta inicial, mientras que solamente es obligatoria la etiqueta final para `antologia`.
- Por último, la tercera parte representa el modelo de contenido. Indica qué elementos y cuántas veces pueden aparecer embebidos en el elemento declarado. Existe la palabra reservada #PCDATA (abreviatura para *Parser Character Data*), que significa que el elemento definido puede contener cualquier dato en forma de caracter válido. En nuestro caso, este hecho se da en la declaración de `titulo` y `verso`. Además pueden aparecer los siguientes símbolos:

  + El elemento correspondiente debe aparecer una o más veces. Por ejemplo, una antología debe tener al menos un poema, un poema al menos una estrofa y una estrofa al menos un verso.

  ? El elemento en cuestión puede aparecer como mucho una vez, siendo posible que no aparezca ninguna. Por ejemplo, un poema podría no tener título.

  ***** El elemento puede aparecer, y si aparece, puede hacerlo una o varias veces. Por ejemplo `(poema*)` permitiría que una antología no contuviese ningún poema.

También es posible que en las declaraciones aparezca una lista de elementos unidos mediante otros símbolos especiales. Por ejemplo,

```
<!ELEMENT poema    - O  (titulo?, estrofa+)>
```

indica que un poema se compone de un título (opcional) seguido de al menos una estrofa.
Podemos considerar por ejemplo que un poema pudiese contener, aparte de estrofas, versos que no tienen por qué formar una estrofa. Esto se expresa de la siguiente forma:

```
<!ELEMENT poema    - O  (titulo?, (estrofa+ | verso+))>
```

Atributos

En SGML se emplea la palabra *atributo* para describir información que es de algún modo descriptiva de una instancia de un elemento específico independientemente de su contenido. Por ejemplo, se puede añadir un atributo `identificador` para poder referirse a una instancia particular del elemento desde cualquier parte del documento.

Aunque diferentes elementos podrían tener atributos con el mismo nombre, son tratados de modo diferentes y pueden tener asignados valores diferentes. Si un elemento se ha definido con atributos, los valores de estos atributos son suministrados como pares *atributo-valor* dentro de la etiqueta de inicio de la instancia particular del elemento. En cambio, una etiqueta final no puede contener una especificación *atributo-valor* ya que podría ser redundante. El siguiente ejemplo presenta un atributo llamado *identificador* con valor *Poema1* y un atributo *estado* con valor *revisado*:

```
<poema identificador=Poema1 estado= "revisado">...</poema>
```

El uso de atributos permite, por ejemplo, que un procesador SGML pudiese tratar de modo distinto los poemas en fase de revisión de los ya revisados.

Del mismo modo que los elementos, se pueden definir las reglas sobre los atributos en la DTD.

```
<!ATTLIST poema
          identificador ID                              #IMPLIED
          estado        (borrador| revisado| publicado)  borrador >
```

La palabra clave ATTLIST precede al elemento o elementos cuyos atributos van a definirse, en este caso, `poema`. Cada fila a continuación declara un atributo que especifica

- El nombre del atributo.
- El tipo de valores que puede tomar. Puede expresarse de dos modos.
  El primero emplea un número de palabras reservadas para declarar qué clase de valor puede tomar el atributo. En concreto, en el ejemplo dado, la palabra `ID` indica que el atributo `identificador` será empleado para suministrar un nombre único que sirve de identificación a `poema`. Otras palabras claves son:

  `CDATA` El valor del atributo puede contener cualquier dato de carácter válido, incluido una etiqueta.

IDREF El valor del atributo debe contener un puntero a algún otro elemento.

NMTOKEN El valor del atributo puede ser cualquier cadena de caracteres alfanuméricos.

NUMBER El valor del atributo se compone solamente de números.

La otra forma de declarar el tipo de los valores de los atributos consiste en una lista con los valores que puede tomar. En el ejemplo, `estado` puede tomar los valores *borrador*, *revisado* o *publicado*.

- El valor por defecto. En caso de ausencia del atributo en cuestión, un procesador SGML tomará este valor para el atributo. Puede aparecer una de estas tres palabras claves:

#REQUIRED Se debe especificar un valor.

#IMPLIED No se necesita especificar un valor.

#CURRENT Si no se suministra un valor para el atributo, entonces se toma el último valor especificado.

Entidades

Por *entidad* debemos entender una parte de un documento marcado sin tener en cuenta cualquier consideración estructural. Una entidad puede ser, por ejemplo, una cadena de caracteres o un fichero de texto. Esto permite al SGML nombrar partes arbitrarias del documento, que pueden ser referenciadas en otra parte. Un ejemplo de declaración de entidad en una DTD es la siguiente:

```
<!ENTITY cyc "Grupo Computadoras y Control">
```

Esta declaración define una entidad llamada `cyc` cuyo valor es la cadena de caracteres "Grupo Computadoras y Control". Una vez que la entidad ha sido declarada puede ser referenciada desde cualquier parte del documento. Esto se hace con el nombre de la entidad precedido por el símbolo `&` y seguido por `;` pudiéndose omitir este último si la referencia va seguida por un espacio en blanco. Cuando un *parser* encuentra una referencia a entidad, la sustituye inmediatamente por su valor declarado. Así el texto "El &cyc desarrolla su trabajo en la ULL" será interpretado exactamente igual que "El Grupo Computadoras y Control desarrolla su trabajo en la ULL".

En el caso particular de una declaración que incluya la palabra `SYSTEM`,

```
<!ENTITY fichero SYSTEM "documento.txt">
```

se define una entidad de sistema llamada `fichero` y cuyo valor es el nombre de un fichero del sistema, siendo el contenido de `documento.txt` el texto por el que se sustituye la referencia.

La definición de entidades presenta las ventajas inmediatas del ahorro de escritura y ayuda a mantener la coherencia en un conjunto de documentos.

Ficheros DTD

La DTD especifica la definición de tipo de documento respecto a la cual se valida una instancia de documento. Un ejemplo de DTD es el siguiente:

```
<!DOCTYPE my.dtd [
  <!-- las declaraciones de la DTD van aquí -->
  ...
]>
<my.dtd>
  Esto es una instancia del tipo de documento MY.DTD
</my.dtd>
```

En este ejemplo, la definición del tipo de documento consiste simplemente en una DTD que precede a una instancia de ese tipo de documento. Sin embargo, esto no es lo más frecuente, sino que la DTD suele estar en otro fichero invocado por referencia desde la instancia. El espacio entre `[]` puede estar vacío o incluir nuevas declaraciones no reflejadas en el fichero referenciado [Bry92, SMB99].

```
<!DOCTYPE mydtd system "my.dtd" [
]>
<my.dtd>
  Esto es una instancia del tipo de documento MY.DTD
</my.dtd>
```

2.4.1.2. Inconvenientes de SGML

El SGML es un lenguaje potente y versátil. Sin embargo es difícil y complicado de utilizar. Su especificación, de más 500 páginas con más de 100 de anexos, es muy complicada, diseñada para sistemas grandes y complejos, excesiva para los requerimientos de un usuario estándar, lo cual ha provocado que hayan surgido otros lenguajes de marcas [Stu00].

2.4.2. Lenguaje de Marcas de Hipertexto (HTML)

El lenguaje de marcas de hipertexto (*Hypertext Markup Language*, HTML) es un lenguaje de marcas, subconjunto del SGML, que se emplea en la construcción de páginas web. El hipertexto es el concepto fundamental de la World Wide Web. Una de las características más importantes del HTML, y que lo diferencia del SGML, es que el conjunto de etiquetas que se pueden emplear en un documento HTML es limitado y definido a priori.

En un principio el HTML era un lenguaje concebido para marcar los documentos de acuerdo con su significado, con etiquetas como `<TITLE>`, `<ADDRESS>`... dejando su interpretación al navegador de Internet (por ejemplo el Netscape Navigator o Internet Explorer).

Sin embargo, y por diferentes motivos, los programadores fueron añadiendo nuevas etiquetas, fuera del estándar, al HTML destinadas en su gran mayoría a aumentar la riqueza visual alejando al lenguaje de su función inicial de descriptor de contenido.

A modo de ejemplo del lenguaje, el par de marcas `<B></B>` sirve para indicar que el texto contenido entre ambas marcas debe ir en negrita[20].

De este modo, el siguiente documento:

```
<HTML>
Esto es un documento <B>HTML </B>
</HTML>
```

será interpretado por el navegador de Internet como

Esto es un documento **HTML**

2.4.2.1. Puntos débiles del HTML

Entre los puntos débiles de HTML encontramos:

- No existe ninguna etiqueta que permita identificar en qué idioma está escrito un documento HTML.
- Sistema de vínculos inadecuado. Al no existir un equivalente a las entidades SGML, si se modifica un vínculo, es necesario recorrer todo el documento HTML sustituyendo el antiguo vínculo por el nuevo. HTML no permite asociar vínculos a ningún elemento, ni tampoco realizar vínculos a ubicaciones múltiples.
- Estructura no rígida. Dentro de la etiqueta `<body>` es posible insertar cualquier etiqueta válida en el lugar que se desee. Es posible validar un documento HTML pero esta validación solamente confirma que las etiquetas se han empleado correctamente, no que se pueda mostrar el contenido de una manera coherente. Mas aún, si se dejan de poner las marcas de final, el navegador trata de encontrar el lugar donde irían y las agrega. Por tanto, es posible crear código HTML escrito de forma no correcta y que el navegador lo interprete como correcto.

2.4.3. Lenguaje de Marcas Extensible (XML)

En 1996, el World Wide Web Consortium (W3C) comenzó a desarrollar un lenguaje de marcas que fuese más fácil de aprender y usar que el SGML, pero que a su vez tuviese una estructura más rígida que el HTML. Este lenguaje es el lenguaje de marcas extensible (*eXtensible Markup Language*, XML). Los objetivos a cumplir por este lenguaje subconjunto de SGML son [Stu00]:

- Debe ser utilizable directamente en Internet.

[20] `<B>` del inglés *bold*, negrita

- Debe admitir un gran número de aplicaciones.
- Compatible con SGML.
- Facilidad para escribir programas de procesamiento del lenguaje.
- Eliminación lo más posible de características opcionales.
- Legible lo más posible para las personas.
- Diseño formal y conciso.
- Facilidad de creación de documentos.
- La concisión en el marcado de XML tiene una importancia mínima. Se valora más mantener conciso (y por tanto sencillo) el estándar XML.

2.4.3.1. Ventajas de XML

Con estos objetivos se ha conseguido un lenguaje considerablemente más fácil de aprender y usar. Sus especificaciones ocupan únicamente 26 páginas, lo cual contrasta con las más de 500 páginas de especificación SGML. Aparte de esta concisión en su especificación, XML ofrece las siguientes ventajas:

- XML se basa en Unicode, por lo que puede incluir caracteres de otros idiomas aparte del inglés. Esto contrasta con HTML y SGML que se basan en ASCII, lo cual no funciona bien en estos idiomas.
- XML puede estar estructurado. Se puede emplear una DTD para esta estructuración.
- Los documentos XML se pueden construir por composición de otros documentos, empleando para ello métodos de vinculación.
- Puede usarse para contener datos. No se trata de un lenguaje que solamente describa la forma de visualizar los datos del documento (como es el caso del HTML), sino que describe su contenido.
- Proporciona flexibilidad a la hora de definir el grado de detalle del documento XML mediante una DTD.
- Simple de utilizar.

2.4.3.2. Características para un documento XML *bien formado*

Se dice que un documento XML esté bien formado (y por lo tanto será correctamente procesado por un *parser*) si cumple los siguientes requisitos.

1. Debe contener un único elemento raíz (un elemento que abre y cierra la parte de los datos del documento). De este modo se aclara al analizador que el documento ya está completo. Por tanto, el siguiente documento no está bien formado.

```
<?xml version=”1.0” ?>
<LIBRO TITULO=“La tempestad”>
</LIBRO>
<LIBRO TITULO=“Rojo y negro”>
</LIBRO>
```

2. Todos los elementos deben estar correctamente anidados, al igual que ocurre con SGML y HTML. El siguiente código corresponde a un contraejemplo. Las marcas `<LIBRO>` y `<PRECIO>` no están bien anidadas.

```
<?xml version=“1.0” ?>
<COLECCION>
<LIBRO TITULO=“La tempestad”>
<PRECIO>18.00
</LIBRO></PRECIO>
</COLECCION>
```

3. Cada atributo solamente puede tener un valor. No estaría bien formado, por tanto, un documento con la siguiente línea:

```
<LIBRO TITULO=“La tempestad” TITULO=“Rojo y Negro”/>
```

4. Todos los valores de los atributos deben aparecer entre comillas dobles o sencillas.

5. Los elementos deben tener etiquetas de inicio y finalización, a no ser que se trate de elementos vacíos. En este sentido, debemos aclarar que XML distingue entre mayúsculas y minúsculas. Por lo tanto una etiqueta `<LIBRO>` no puede cerrarse con una etiqueta final `</libro>`.

6. Los elementos vacíos se pueden abreviar mediante una única etiqueta que termina con la barra inclinada hacia la derecha `/`. Este punto, junto al anterior, constituyen dos de las principales diferencias del XML respecto al SGML y el HTML.

7. No se permite la existencia de caracteres de marcado aislados en el contenido. Los caracteres especiales `<`, `&` y `>` se representan como `&gt;`, `&amp` y `&lt` respectivamente en las secciones de contenido.

8. Las comillas dobles se representan como `&quot`. Las comillas sencillas por `&apos`. Ambas representaciones en la sección de contenido.

9. No se puede emplear las secuencias `<[[` ni `]]>`

10. Si un documento no tiene una DTD asociada, los valores de todos sus atributos deben ser de tipo CDATA de modo predeterminado.

2.4.3.3. Espacios de nombres XML

Una de las motivaciones para el empleo de XML es su modularidad. Si existen documentos con vocabularios de formato bien entendido y para el cual hay programas disponibles, es mejor reutilizar este vocabulario en vez de reinventarlo.

El empleo de estos documentos presenta problemas de reconocimiento y colisión. El software de procesamiento debe ser capaz de reconocer las etiquetas, incluso cuando se entra en colisión con otros elementos que emplean la misma etiqueta.

En este sentido, se definen los espacios de nombres (*namespaces*) XML como unos conjuntos de nombres, identificados por una referencia URI[21] que se utilizan en documentos XML como tipos de elemento y nombres de atributo. Los nombres de los espacios de nombres pueden aparecer como nombres cualificados, que contienen un símbolo : que divide al nombre en un prefijo, el cual corresponde a la referencia URI y es el encargado de seleccionar el espacio de nombres, y una parte local. La combinación del espacio de nombres URI gestionado universalmente y del espacio de nombres propio del documento produce identificadores que son únicos a nivel universal [W3C99].

Un espacio de nombres se declara usando una familia de atributos reservados. El nombre de tales atributos debe o bien ser `xmlns`, o bien tener `xmlns:` como prefijo. Por ejemplo, la de

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

sirve para asociar al prefijo `xs` la URI `http://www.w3.org/2001/XMLSchema`. Esto permitirá distinguir entre los elementos de mismo nombre definidos con dos espacios de nombres diferentes. Así se podrá distinguir un supuesto elemento `element` en un documento referenciado por un prefijo `xs` (`xs:element`) de otro elemento `element` definido en el mismo u otro documento (probablemente definido además de un modo diferente) con otra URI y referenciado por otro prefijo, por ejemplo, `xsl` (`xsl:element`).

2.4.3.4. La declaración XML

Tal como se ha visto en los ejemplos anteriores, todo documento XML comienza con una línea llamada *declaración.* La declaración XML proporciona información sobre la versión del estándar XML con la que cumple el documento (atributo `version`), el conjunto de caracteres Unicode empleado en el documento (atributo opcional `encoding`) y si el documento depende o no de otros documentos (atributo opcional `standalone`, valor "yes" para documentos XML que no dependen de otros). La estructura de esta declaración es:

```
<?xml version="número_de_versión" encoding="codificación"
standalone="documento_aislado"?>
```

[21] Se considera que las referencias URI que identifican espacios de nombres son idénticas cuando son exactamente las mismas caracter por caracter.

Un ejemplo de declaración XML es:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
```

2.4.3.5. XSL y XSLT

Un documento XML presenta una facilidad de lectura y comprensión para las personas. No obstante, un documento con marcas no es la mejor forma de visualización. Por ejemplo, supongamos el siguiente documento XML referidos a las notas de los alumnos de una clase.

```
<?xml version="1.0" ?>
<CLASE CURSO="4">
  <ALUMNO>
    <NOMBRE>Eladio Ayoze</NOMBRE>
    <APELLIDOS>Gutierrez Bencomo</APELLIDOS>
    <NOTA>7.80</NOTA>
  </ALUMNO>
 ...
</CLASE>
```

Si un profesor quisiese publicar esta lista de notas a través de un navegador, la lista quedaría, dependiendo del navegador, en forma de árbol extensible (ver Figura 2.3), algo engorrosa. XML está pensado para el almacenamiento y transmisión de los datos, no para su visualización. Como es de esperar, existe un método para poder visualizar de una manera más funcional los datos de un fichero XML. Además, este método, permite elegir qué datos se muestran y cómo mostrarlos. Éste es el cometido de las hojas de estilo XSL (*eXtensible Stylesheet Language*).

Figura 2.3.: Visualización por defecto de un documento XML en un navegador de Internet: árbol extensible.

Estas hojas de estilo XSL cumplen una función similar a las CSS (hojas de estilo en cascada, *Cascading Style Sheets*). En CSS se define un estilo para cada elemento HTML. Por ejemplo, en una CSS, se puede definir el estilo para el elemento `h1` de

color verde y tamaño de fuente 10. Con esta definición, todos los elementos `h1` de una página web, aparecerán con esos atributos cuando se carga en el navegador. Si en un momento posterior el desarrollador quiere cambiar el color, no tendrá que recorrer todo el documento cambiando el formato. Solamente será necesario cambiar el atributo en la CSS.

Esta misma filosofía es aplicable a las XSL para documentos XML. Se puede definir un estilo para cada elemento del documento. En el documento que nos ocupa, el nombre de un alumno (correspondiente al elemento `<NOMBRE>`) puede aparecer siempre, por ejemplo, en negrita y cursiva. De este modo, cambiando únicamente la hoja de estilo, se controla la apariencia con que se visualiza el documento.

Para poder realizar esta visualización, lo primero es asignar al documento XML qué hoja de estilo se va aplicar. Por ejemplo, si se desea emplear una hoja de estilo llamada *tablasNotas.xsl*, se debe añadir una línea más al encabezado del documento XML indicando esta referencia.

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="tablasNotas.xsl"?>
<CLASE CURSO="4">
```

Por su parte, el documento XSL es un documento XML bien formado cuya estructura es la siguiente:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
  <!-- elementos de procesamiento -->
...
  </xsl:template>
</xsl:stylesheet>
```

En el presente trabajo no vamos a entrar en más detalle sobre el XSL y la forma de los elementos de procesamiento. No obstante, indicaremos que unos elementos de procesamiento adecuados pueden hacer que al cargar el documento con un navegador, la apariencia sea, por ejemplo, como la mostrada en la Figura 2.4.

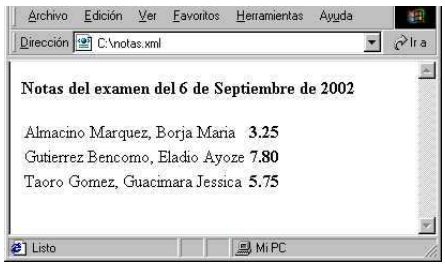

Figura 2.4.: Visualización alternativa de un documento XML mediante el empleo de una hoja de estilo XSL.

Obsérvese que la hoja de estilo XSL solamente proporciona una herramienta de visualización de documentos XML. En ningún momento se transforma el documento en un formato más legible. Para este cometido se emplea el XSLT[22], un lenguaje para transformar un documento XML en cualquier otro documento basado en texto.

2.4.3.6. XML Schemas

Al igual que los documentos SGML, se pueden fijar una serie de reglas que describan el contenido de un documento XML y que ayuden a jerarquizar y estructurar la información. Estas reglas pueden expresarse según una DTD (lo cual es de esperar, puesto que el XML es un subconjunto con SGML) o de un modo más recomendado en la actualidad a través del llamado XML Schema (XMLS, fichero con un extensión `xsd`[23]). Cuando un documento XML está ligado a uno de estos ficheros y cumple con sus reglas, se dice que el documento es *válido* o está *validado*, y el proceso de comprobar si un documento es válido respecto a una DTD o XML Schema recibe el nombre de *validación*.

Un XML Schema define:

- los elementos y atributos que pueden aparecer en un documento
- qué elementos pueden ser elementos hijo de un elemento padre, su orden y número
- si un elemento aparece como vacío o puede incluir texto
- tipos de datos para elementos y atributos
- valores por defecto, así como invariables, para los elementos y atributos

Un XML Schema ofrece las siguientes ventajas respecto a una DTD:

- Los XML Schema son fácilmente extensibles. Las DTDs ofrecen una extensibilidad complicada de ejercitar.
- Los XML Schema son más ricos y más útiles que las DTDs. Soporta los tipos de *string*, *date*, *integer*, *boolean*... De este modo, pueden definirse el tipo y rango de los elementos y atributos. Las DTDs solamente pueden emplear las palabras claves CDATA, NUMBER...
- Los XML Schema están escritos en XML, por tanto, no es necesario aprender un nuevo lenguaje. Las DTDs tienen un lenguaje propio de escritura.
- Los XML Schema soporta la definición por parte del usuario de nuevos tipos de dato, llamados *arquetipos*, los cuales pueden ser extendidos a su vez mediante una relación de herencia.
- Los XML Schema soportan los espacios de nombres. Por tanto, permite definir elementos de igual nombre. Para ello debe ser precedido por prefijos diferentes.

[22] *XSL Transformations*, http://www.w3.org/TR/xslt

[23] de XML Schema Definition, con el que también suele referirse al XML Schema.

Elementos y Atributos

Como hemos comentado, el XML Schema está escrito en XML. El elemento raíz de todo fichero XMLS es el elemento `<schema>`. Este elemento puede contener una serie de atributos. Un posible esquema sería el siguiente:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.cyc.ull.es"
  xmlns="http://www.evenet.cyc.ull.es">
...
</xs:schema>
```

Este esquema indica que los elementos y tipos de datos que comiencen con el prefijo `xs:` provienen del espacio de nombres `http://www.w3.org/2001/XMLSchema` (estos elementos van a ser *schema*, *elemento*, *secuencia*, *boolean*...). Los elementos definidos en el esquema a definir corresponderán al espacio de nombres `http://www.cyc.ull.es` (definidos por el *targetNameSpace*) *y* el espacio de nombres por defecto (aquellos elementos que no llevan prefijo) es `http://www.cyc.ull.es`.

A este esquema básico le añadiremos una serie de elementos que reflejan la estructura de un correo electrónico.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.cyc.ull.es"
xmlns="http://www.evenet.cyc.ull.es">
 <xs:element name="correo_electronico">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="de" type="xs:string"/>
    <xs:element name="para" type="xs:string"/>
    <xs:element name="fecha" type="xs:date"/>
    <xs:element name="asunto" type="xs:string"/>
    <xs:element name="contenido" type="xs:string"/>
   </xs:sequence>
   <xs:attribute name="respondido" type="xs:boolean"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

En este esquema se define un elemento `correo_electronico` como de tipo complejo (complexType) porque contiene otros elementos. En este caso se compone de cinco elementos: `de`, `para`, `fecha`, `asunto` y `contenido`, que al no contener ningún elemento embebido se denominan simples. Un elemento simple se define del siguiente modo general, donde `xxx` es el nombre del elemento e `yyy` es el tipo de dato

del elemento (por ejemplo *string*, *decimal*, *boolean*, *integer*, *date*, *time*).

```
<xs:element name="xxx" type="yyy"/>
```

El XMLS también permite declarar un valor por defecto (atributo *default*) y fijar un valor que no puede ser modificado (atributo *fixed*).

Los atributos definidos en XMLS se definen de modo equivalente al de los elementos:

```
<xs:attribute name="xxx" type="yyy"/>
```

En el ejemplo anterior, se define el atributo `respondido` de tipo *boolean.* Los atributos pueden, a través del atributo *use*, declararse opcionales (*optional*) u obligatorios (*required*).

Indicadores

A la hora de definir el elemento complejo `correo_electronico` se ha hecho uso del elemento `<xs:sequence>`. Este elemento es un indicador y se refiere a cómo pueden aparecer los elementos hijo en un documento XML validado respecto al correspondiente esquema. En XMLS existen siete indicadores:

`all` Los elementos hijo pueden aparecer en cualquier orden y cada elemento debe aparecer una y sólo una vez.

`choice` Puede aparecer uno u otro de los elementos hijo.

`sequence` Los elementos hijo deben aparecer en el orden especificado.

`maxOccurs` Número máximo de veces que puede aparecer un elemento. Aparece como atributo del elemento afectado.

`minOccurs` Número mínimo de veces que puede aparecer un elemento. Aparece como atributo del elemento afectado.

`group` Empleado para definir un conjunto de elementos relacionados. Se emplea con la estructura

```
<xs:group name="nombre_del_grupo">
  ...
</xs:group>
```

`attributeGroup` Análogo al indicador *group* pero referido a atributos relacionados.

Restricciones

Terminaremos este breve recorrido por el XML Schema tratando el tema de las restricciones que controlan los valores aceptables que pueden tomar los elementos o atributos de un documento XML.

Las restricciones más significativas en XMLS son:

enumeration Define una lista de valores aceptables.

fractionDigits Especifica el número máximo de decimales permitidos.

length Especifica el número exacto de caracteres o elementos permitidos.

maxExclusive Especifica el límite superior para valores numéricos. El valor del elemento debe ser estrictamente menor que el valor especificado.

maxInclusive Especifica el límite superior para valores numéricos. El valor del elemento debe ser menor o igual que el valor especificado.

minExclusive Especifica el límite inferior para valores numéricos. El valor del elemento debe ser estrictamente mayor que el valor especificado.

minInclusive Especifica el límite inferior para valores numéricos. El valor del elemento debe ser mayor o igual que el valor especificado.

maxLength Especifica el número máximo de caracteres o elementos permitidos.

minLength Especifica el número mínimo de caracteres o elementos permitidos.

pattern Define la secuencia exacta de caracteres que son aceptables.

totalDigits Especifica el número exacto de decimales permitidos.

whiteSpace Especifica cuánto espacio en blanco está permitido (incluyendo espacios, tabulaciones, salto de línea y retorno de carro).

El siguiente ejemplo indica que el elemento edad es un entero entre 0 y 100. Para ello se hace uso de los elementos *minInclusive* y *maxInclusive*.

```
<xs:element name="edad">
 <xs:simpleType>
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0"/>
   <xs:maxInclusive value="100"/>
  </xs:restriction>
 </xs:simpleType>
</xs:element>
```

2.4.4. Simple HTML Ontology Extensions, SHOE

SHOE[24] es un lenguaje de representación del conocimiento basado en HTML, compatible con SGML que permite añadir a las páginas web conocimiento legible por las máquinas. Es un superconjunto de HTML que añade las etiquetas necesarias para embeber datos de semántica arbitrarios en páginas web, datos que serán procesados por un *parser* que analice dicha página[25]. Intenta superar los problemas

[24] http://www.cs.umd.edu/projects/plus/SHOE

[25] SHOE también puede ser empleado en documentos XML.

de interoperabilidad que presenta la arbitrariedad en el etiquetado de XML[26]. Este lenguaje ha sido superado por lenguajes más expresivos como DAML+OIL y OWL.

SHOE fue diseñado con las necesidades de la web en mente. Tiene capacidades semánticas limitadas para hacer posible manejar grandes cantidades de datos. Sin embargo, una simple semántica de base de datos no es suficiente. SHOE tiene una variedad de mecanismos que tratan de manejar el hecho de que los datos en la Web se encuentran distribuidos y sin un control total.

SHOE puede emplearse para embeber datos procedentes de una gran variedad de fuentes y para una gran variedad de propósitos.

SHOE no es solamente un lenguaje de meta-contenido, no pretende ser un sistema de representación de conocimiento (SHOE trata de proporcionar toda la expresividad posible teniendo en cuenta la gran cantidad de información existente en la Web), ni proporciona ningún tipo de ontología, relaciones o inferencias predefinidas, dejando a los diseñadores estas definiciones [LH00].

Las etiquetas SHOE se dividen en dos categorías. Primero, existen etiquetas para construir ontologías. Las ontologías SHOE son unos conjuntos de reglas que definen qué tipo de afirmaciones puede hacer un documento SHOE así como el significado de esas afirmaciones. Por ejemplo, podría declarar que una determinada entidad es un "perro" y que puede tener un "nombre". El segundo conjunto de etiquetas se emplean para realizar anotaciones en los documentos web (que podrían ser legibles para una máquina) para subscribir una o más ontologías. Por ejemplo, un documento SHOE que suscriba la anterior ontología SHOE podría declarar que existe un perro llamado "Fido".

2.4.4.1. Breve descripción de la Sintaxis SHOE

En el resto de la subsección, pasaremos a describir brevemente la sintaxis SHOE.

Etiquetas añadidas a HTML

Para poder dotar de mayor expresividad a las páginas web, SHOE añade las siguientes etiquetas al HTML:
`ONTOLOGY`, `USE-ONTOLOGY`, `DEF-CATEGORY`, `DEF-RELATION`, `DEF-ARG`, `DEF-RENAME`, `DEF-CONSTANT`, `DEF-TYPE`, `DEF-INFERENCE`, `INF-IF`, `INF-THEN`, `COMPARISON`, `CATEGORY`, `RELATION`, `ARG`.

Declaración de la ontología

Una ontología puede aparecer en el nivel más alto del cuerpo de un documento HTML, esto es, solamente puede ir embebida entre las etiquetas `<BODY>` y `</BODY>`. El comienzo de la definición de una ontología responde al siguiente esquema, donde también pueden aparecer atributos relativos a las ontologías con la que es compatible, una descripción del propósito de la ontología y la lista de instancias que se van a declarar en la ontología:

[26] Se puede emplear una DTD o XML Schema, pero es complicado encontrar uno de estos documentos que describa cualquier elemento.

```
<ONTOLOGY ID="identificador" VERSION="version">
```

El final de la ontología, como es de esperar viene marcado por la etiqueta `</ONTOLOGY>`.

Extensiones de Ontologías Existentes

Una ontología puede emplear elementos de otras ontologías existentes. Estos elementos pueden ser empleados de otro modo por la ontología. Entonces, se dice que la ontología *extiende* las ontologías previas. Para distinguir estos términos prestados de los propios de la ontología, se debe proporcionar un prefijo único para cada ontología extendida (con una filosofía similar a los espacios de nombre XML). Esta extensión sigue el siguiente esquema, donde se puede añadir un atributo URL que apunte al documento que contiene la ontología extendida:

```
<USE-ONTOLOGY ID="identificador" VERSION="version"
    PREFIX="prefijo">
```

Definición de Reglas de Clasificación (Clases)

Para definir las clases o categorías, se emplea el siguiente esquema. Los atributos opcionales son `ISA` (lista de superclases), `DESCRIPTION` (descripción de la clase) y `SHORT` (una frase que un agente podría emplear de un modo más entendible que el nombre de la categoría). Por ejemplo:

```
<DEF-CATEGORY NAME="Hombre" ISA="Persona">
```

Definición de Relaciones

Las relaciones son definidas en SHOE del siguiente modo (pueden aparecer los atributos `DESCRIPTION` y `SHORT`, análogos a los definidos con anterioridad). Embebida entre las etiquetas de inicio y final, aparece una lista de argumentos de la relación, indicando en cada argumento la posición que ocupa y su tipo (una categoría definida o elementos definidos en la ontología base SHOE, como por ejemplo `.STRING`). Ejemplo:

```
<DEF-RELATION NAME="edad">
  <DEF-ARG POS="1" TYPE="Persona">
  <DEF-ARG POS="2" TYPE=".NUMBER">
</DEF-RELATION>
```

Definición de Reglas de Inferencia

Una ontología puede definir una serie de inferencias que podrían llevarse a cabo, basándose en las declaraciones encontradas en el texto. Estas definiciones se componen de una serie de proposiciones que se agrupan en un esquema del tipo *if-then*. En el siguiente ejemplo se expresa la propiedad transitiva de suborganizaciones, esto es, si x es una suborganización de y e y es una suborganización de z, entonces x

es una suborganización de *z*:

```
  <DEF-INFERENCE DESCRIPTION="Propiedad Transitiva de
Suborganizaciones">
    <INF-IF>
      <RELATION NAME="subOrganizacion">
        <ARG POS="1" VALUE="x" VAR>
        <ARG POS="2" VALUE="y" VAR>
      </RELATION>
      <RELATION NAME="subOrganizacion">
        <ARG POS="1" VALUE="y" VAR>
        <ARG POS="2" VALUE="z" VAR>
      </RELATION>
    </INF-IF>
    <INF-THEN>
      <RELATION NAME="subOrganization">
        <ARG POS="1" VALUE="x" VAR>
        <ARG POS="2" VALUE="z" VAR>
      </RELATION>
    </INF-THEN>
  </DEF-INFERENCE>
```

Las definiciones de las proposiciones son similares a las declaraciones de las categorías y las relaciones (salvo el prefijo `DEF` y el argumento `VALUE` - *valor* - para `RELATION`)[27], pudiendo definir entre las proposiciones del *if* la declaración de una comparación. El operador es una de las palabras claves `equal`, `notEqual`, `greaterThan`, `greaterThanOrEqual`, `lessThanOrEqual`, `lessThan`. Con estas palabras se indica respectivamente que el primer argumento es igual, distinto, mayor, mayor o igual, menor o igual o menor que el segundo argumento.

```
  <COMPARISON OP="greaterThan">
    <ARG POS="1" VALUE="a" VAR>
    <ARG POS="2" VALUE="18">
  </COMPARISON>
```

Declaraciones de instancias

Las declaraciones de instancias siguen el siguiente esquema, donde el argumento `KEY` hace referencia a una URL definida de forma única, la cual es significativa de la instancia. Ejemplo:

```
  <INSTANCE KEY="http://masplan.cyc.ull.es/#Evelio">
    <RELATION NAME= "edad">
      <ARG POS="2" VALUE="28">
    </RELATION>
```

[27] Se remite al lector interesado en un mayor detalle a [LH00].

```
</INSTANCE>
```

2.4.5. Lenguaje de intercambio de ontologías basado en XML (XOL)

Como aplicación de lenguaje de ontología basado en marcas, describiremos en primer lugar brevemente un lenguaje de marcas basado en XML, llamado Lenguaje de Intercambio de Ontologías basado en XML (*XML-based ontology-exchange language,* XOL)[28] [29]. Este lenguaje está diseñado para proporcionar un formato para el intercambio de definiciones de ontología entre un conjunto de partes interesadas.

El lenguaje es previo en su definición al DAML+OIL y ha obtenido una repercusión mucho menor. Sin embargo nos servirá para introducir en este trabajo el poder de los lenguajes basados en marcas en la definición de ontologías.

La sintaxis de XOL se basa en un enfoque según el cual basta un conjunto sencillo de etiquetas XML (definidas mediante una DTD XML) para describir cualquier ontología. Por lo tanto, su principal aportación es la sencillez. Sin embargo, el enfoque descrito presenta el gran inconveniente que los motores de procesamiento XML pueden llevar a cabo solamente unos pocos tipos de comprobaciones sobre las especificaciones XML. Por ejemplo, la siguiente definición XOL

```
<slot>
  <name>edad</name>
  <domain>persona</domain>
  <value-type>entero</value-type>
  <numeric-max>150</numeric-max>
</slot>
```

especifica, entre otras cosas, que la propiedad edad tiene un valor entero. Esta información no está incluida en la DTD (se ha preferido esta representación en detrimento del XML Schema) general de XOL, por lo que un parser XML no puede comprobar si la edad definida en un documento referida a un individuo es en realidad un entero o si es realmente menor que el valor máximo de 150. Esto hace su empleo algo ineficiente respecto a otros lenguajes detallados en posteriores apartados, como RDF, DAML+OIL u OWL.

2.4.5.1. Sintaxis XOL

Esta subsección describe la forma general de un documento XOL. El documento comienza con una sección de módulo, que identifica la ontología definida en el fichero. Embebidas en este módulo se encuentran las definiciones de clases de la ontología, a través de la etiqueta `<class>`, las propiedades `<slots>`, y los individuos `<individual>`. Con esto, un esquema general de un documento XOL sería el siguiente:

```
<?xml version="1.0" ?>
```

[28] http://ai.sri.com/~pkarp/xol

[29] existen otros lenguajes de marcas de ontologías como el OML

```
<!DOCTYPE module SYSTEM "module.dtd">
<module>
  ...
  <class>
   ...
  </class>
  <slot>
   ...
  </slot>
  <individual>
   ...
  </individual>
...
</module>
```

Sección de Modulo

El primer elemento del documento puede corresponder a uno de los siguientes cinco: `module`, `ontology`, `kb`, `database` y `dataset`. En realidad estos términos son sinónimos. XOL proporciona estos sinónimos de forma intencionada porque los diferentes desarrolladores están acostumbrados a emplear diferentes términos que son en esencia los mismos.

Esta sección contiene una descripción de la ontología mediante los siguientes elementos:

<name> Único elemento obligatorio de la sección que especifica el nombre de la ontología.

<kb-type> <db-type> Especifica el sistema de gestión de la base de datos o de la base de conocimiento en el que la ontología fue inicialmente construida.

<package> Especifica el paquete en el que por defecto están definidos todos los símbolos empleados en la base de conocimiento.

<version> Versión de la ontología

<documentation> Documentación de la ontología.

Sección de clases

Lista de la definición de las clases contenidas en la ontología. Embebidas dentro de la etiqueta `<class>` pueden ir los elementos:

<name> Elemento obligatorio que especifica el nombre de la clase.

<documentation> Elemento opcional que proporciona documentación sobre la clase.

<subclass-of> Contiene la clase de la cual la clase correspondiente es subclase.

<instance-of> Contiene la clase, llamada *metaclase,* de la cual la clase correspondiente es una instancia.

<slot-values> Especifica valores de propiedades de la clase.

Como ejemplo de esta sección, el siguiente código define la clase Hombre como una subclase de la clase Persona.

```
<class>
  <name>Hombre</name>
  <documentation>Clase Hombre</documentation>
  <subclass-of>Persona</subclass-of>
</class>
```

Sección de Slots

Definiciones de los slots que definen atributos de una clase o relaciones entre clases. Las posibles etiquetas embebidas entre las etiquetas `<slot></slot>` son:

<name> Elemento obligatorio que especifica el nombre del slot.

<documentation> Elemento opcional que proporciona documentación sobre el slot.

<slot-value-type> El tipo de dato del slot, que puede ser tanto el nombre de una primitiva de tipo de dato XOL (por ejemplo un entero, *integer*), el nombre de una clase definida en la ontología, una serie de palabras claves definidos mediante el empleo de un constructor set o una combinación de ambos.

<slot-inverse> Codifica la relación semántica inversa de este slot.

<slot-maximum-cardinality> Límite superior del número de valores que puede tener el slot.

<slot-minimum-cardinality> Límite inferior del número de valores que puede tener el slot.

<slot-cardinality> Especifica el número exacto de valores que puede tener el slot.

<slot-numeric-maximum> Límite superior del valor que puede tener el slot cuando sus valores son enteros o reales.

<slot-numeric-minimum> Límite inferior del valor que puede tener el slot cuando sus valores son enteros o reales.

<slot-collection-type> Para los slots que pueden tener más que un valor, este elemento especifica si esos valores deben ser interpretados como un conjunto (*set*), una lista (*list)* o una bolsa (*bag*[30]*).*

[30] una colección en la que el orden de los elementos no es importante, pudiendo tener recursos duplicados

Como ejemplo de esta sección, el siguiente código define un slot referente al año de nacimiento, indicando que su dominio se refiere a la clase Persona, a cada persona le corresponde exactamente un año de nacimiento, que ninguna persona ha nacido antes del año 1800 y que el valor del slot debe ser un entero:

```
<slot>
  <name>año_de_nacimiento</name>
  <domain>Persona</domain>
  <slot-cardinality>1</slot-cardinality>
  <slot-numeric-minimum>1800</slot-numeric-minimum>
  <slot-value-type>integer</slot-value-type>
</slot>
```

Sección de Individuos

Definiciones de los individuos de la ontología. Sigue la misma sintaxis que `<class>` salvo que no puede aparecer la etiqueta `<subclass-of>`.

Como ejemplo de esta sección, el siguiente código define un individuo de la clase Persona, identificado como Francisco, nacido en 1962 y con dos hermanos llamados Silvestre y Alejandro [KCT99].

```
<individual>
  <name>Francisco</name>
  <instance-of>Persona</instance-of>
  <slot-values>
    <name>año_de_nacimiento</name>
    <value>1962</value>
  </slot-values>
  <slot-values>
    <name>hermano</name>
    <value>Silvestre</value>
    <value>Alejandro</value>
  </slot-values>
</individual>
```

2.4.6. Marco de Descripción de Recursos (RDF)

El estándar XML proporciona una estructura sintáctica para la descripción de datos. Desgraciadamente, el XML puede ser empleado de muchas formas para describir los mismos datos. Esto lo convierte en demasiado arbitrario para la integración prevista en la Web Semántica.

El Marco de Descripción de Recursos (*Resource Description Framework,* RDF) es una recomendación del W3C para la representación de metadatos en la web. El propósito es dar una forma estándar de especificar los datos sobre algo. Los datos RDF representan recursos junto a parejas de valores atributo/valor. Un *recurso* es cualquier elemento representable mediante un identificador de recursos uniformes (URI). El empleo de URIs está justificado en que si una aplicación es capaz de acceder y emplear

los datos de cualquier otra aplicación, cada objeto y modelo de datos debe tener un medio de identificación único y universal. Por tanto, mediante el uso de URIs, RDF permite identificar a un recurso de modo unívoco. Un aspecto a resaltar es que RDF considera a cada URI como un recurso en sí mismo.

Realmente RDF emplea lo que se denomina URI cualificado (*qualified URI*), es decir, un URI con un identificador de fragmento opcional (un texto añadido al URI separado mediante un carácter "#", el cual es simplemente un mecanismo para separar el espacio de nombres del nombre del tipo y el nombre de la propiedad).

El elemento básico del RDF es el *triple.* Un recurso (el *sujeto*) está unido a otro recurso (el *objeto,* que puede ser una entidad atómica -textos, números...- u otro recurso representado por una URI) mediante un arco etiquetado con otro recurso (el *predicado*). Se dice entonces que el `<sujeto>` tiene la propiedad `<predicado>` con valor `<objeto>`. Un sujeto puede tener más de un recurso como objeto para el mismo predicado.

La relación de *triple* se muestra en el grafo de la Figura 2.5. En este triple, el recurso Documento1 con URI `http://masplan.cyc.ull.es/#Documento1` tiene como creador (propiedad de URI `http://purl.org/DC#Creator`) al recurso Autor1 (de URI `http://masplan.cyc.ull.es/#Autor1`).

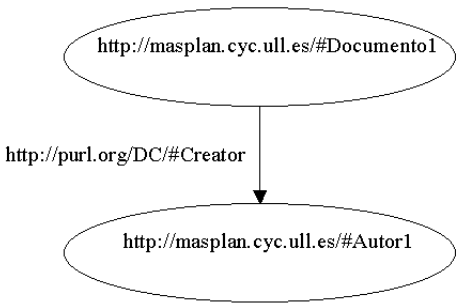

Figura 2.5.: Triple RDF simple.

Cuando el recurso es un literal (una cadena de caracteres), éste se representa entrecomillado en un rectángulo. Así, si en el ejemplo anterior se reemplaza el recurso referido a Autor1 por la cadena "Juan Méndez", la representación sería la mostrada en la Figura 2.6.

Este grafo permite las siguientes lecturas:

- El creador del Documento1 es Juan Méndez
- Juan Méndez es el creador del Documento1

Para los lectores humanos, ambas lecturas ofrecen el mismo significado. Sin embargo, para una máquina corresponden a dos cadenas de caracteres distintas. Es precisamente

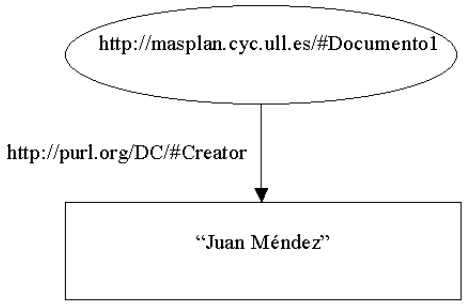

Figura 2.6.: Uso de rdf:Literal en la representación de triples RDF.

el RDF mediante su representación de *triples* quien proporciona un método sin ambigüedad para expresar la semántica en una codificación legible para las máquinas.

Los *triples* por convenio se escribe en el orden sujeto, predicado, objeto o hablando en términos RDF en el orden recurso, predicado, valor.

Un triple se encuentra íntimamente relacionado con la Web Semántica. Las partes de un *triple* RDF, por diseño, corresponde con un enlace Web:

- El sujeto del triple es la web inicial.
- El predicado conecta el sujeto con el objeto.
- El objeto corresponde al destino del enlace.

RDF necesita de una sintaxis de seriación para hacer disponible los datos en un sistema basado en web. Para ello se escogió el XML. Por tanto, RDF y XML son complementarios, en el sentido de que RDF representa el modelo abstracto mientras que XML proporciona la representación textual concreta del modelo. Existen varias maneras de representar el mismo modelo de datos RDF en XML. De un modo análogo, un documento XML puede ser modificado para hacerlo compatible con RDF, mediante unas transformaciones no demasiado complicadas [DMM00, W3C03c, W3C03d, Cha00, Mil98, CJ03a, Ekl01].

Como ejemplo, tenemos el siguiente ejemplo de documento XML, referido al río Yangtzé.

```
<? xml version="1.0" ?>
<Rio id="Yangtze" xmlns="http://www.geodesy.org/river">
 <longitud>6300 kilómetros</longitud>
 <lugarNacimiento>Meseta del Tibet</lugarNacimiento>
 <lugarDesembocadura>Mar de la China Oriental</lugarDesembocadura>
</Rio>
```

Puede transformarse en el siguiente documento RDF (la sintaxis se describirá con más detalle en el apartado 2.4.6.2). Como se puede observar, las diferencias consisten básicamente en 1) definir el recurso Yangtzé y las propiedades relativas a la longitud

y los lugares de nacimiento y desembocadura, y 2) establecer adecuadamente los espacios de nombres (obsérvese el empleo del símbolo #, al final del espacio de nombres).

```
<? xml version="1.0" ?>
<Rio rdf:ID="Yangtze" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.geodesy.org/river#">
 <longitud>6300 kilómetros</longitud>
 <lugarNacimiento>Meseta del Tibet</lugarNacimiento>
 <lugarDesembocadura>Mar de la China Oriental</lugarDesembocadura>
</Rio>
```

Como hemos indicado, el fundamento para el surgimiento del RDF es dotar de una forma estándar de estructura de datos. Esta estructura, basada en triples sujeto/propiedad/objeto, se muestra en la Figura 2.7. Obsérvese que el patrón de diseño es una secuencia alternada de recursos y propiedades.

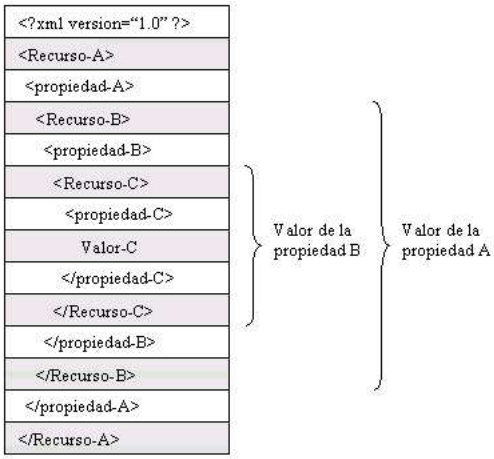

Figura 2.7.: Patrón de diseño RDF.

2.4.6.1. Ventajas y desventajas del uso del RDF respecto al XML

El RDF presenta las siguientes ventajas de utilización para la representación de datos respecto al XML (aunque debemos insistir en que el RDF se expresa en XML)

- El formato RDF ayuda a dotar al XML de más interoperabilidad.
  - Las herramientas de procesamiento pueden caracterizar de manera inmediata la estructura "este elemento es de un tipo (clase) y éstas son sus propiedades"
  - RDF promociona el uso de vocabularios, clases y propiedades estandarizados.

- El formato RDF proporciona un enfoque estructural para el diseño de los documentos XML.
- Ayuda a detectar rápidamente inconsistencias en los diseños XML, al forzarle a estar estructurado. De este modo ayuda a comprender mejor los datos.
- Puede disfrutarse de los beneficios de ambos lenguajes:
  - Se pueden emplear editores estándar XML para crear, editar y validar.
  - Se pueden emplear las herramientas RDF para dotar de inferencia a los datos.
- Prepara los datos estructurados para la Web Semántica.

Entre las desventajas del uso de RDF resaltamos:

- El formato RDF restringe al usuario en el diseño XML. Ya no se puede diseñar un documento XML de cualquier modo arbitrario.
- Requiere un conocimiento sólido sobre los espacios de nombres.
- Requiere aprender el vocabulario RDF.

2.4.6.2. Sintaxis RDF

El siguiente ejemplo de documento RDF se refiere a el triple mostrado en la Figura 2.6.

```
<? xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/02/22-rdf-syntax-ns#"
  xmlns:DC="http://purl.org/dc/elements/1.0/" >
  <rdf:Description about =
   "http://masplan.cyc.ull.es/#Documento1">
    <DC:Creator>Juan Méndez</DC:Creator>
  </rdf:Description>
</rdf:RDF>
```

Como se puede ver en este ejemplo, un documento RDF consiste en documento XML (obsérvese la primera línea) con una lista de descripciones de recursos a través de etiquetas `<rdf:Description>` embebidas en un elemento raíz nombrado como `<rdf:RDF>`. Los elementos `rdf:Description` pueden tener uno (no ambos) de los siguientes atributos:

`rdf:about` Se emplea para describir cualquier recurso. Su valor es un URI, ya sea absoluto o relativo. En el ejemplo anterior se emplea para referirse a la URI del Documento1. Se recomienda su uso cuando se extienda la información acerca de un recurso.

`rdf:ID` Se emplea para definir un recurso. Su valor es un fragmento identificador (sin el caracter #) que se añadirá a la URI del documento XML. Por tanto, se recomienda su uso cuando se desee introducir un recurso y proporcionar un conjunto inicial de informaciones acerca del mismo.

En caso de que una descripción no tenga ninguno de estos atributos, se dice que está describiendo un recurso anónimo[31]. Un elemento `rdf:Description` contiene una secuencia de elementos XML. Estos elementos son tomados como propiedades cuyos predicados tienen como URI el nombre expandido del elemento en cuestión, siendo la URI del sujeto del triple la correspondiente al elemento raíz. Si el elemento embebido está vacío, debe tener un atributo `rdf:resource` cuyo valor es la URI del objeto. En caso contrario, puede contener texto (entonces interpretado como un literal) o un elemento embebido simple del tipo `rdf:Description`. En el siguiente ejemplo se manifiestan estos aspectos, así como el empleo de recursos definidos en otra parte del documento. El grafo correspondiente se muestra en la Figura 2.8.

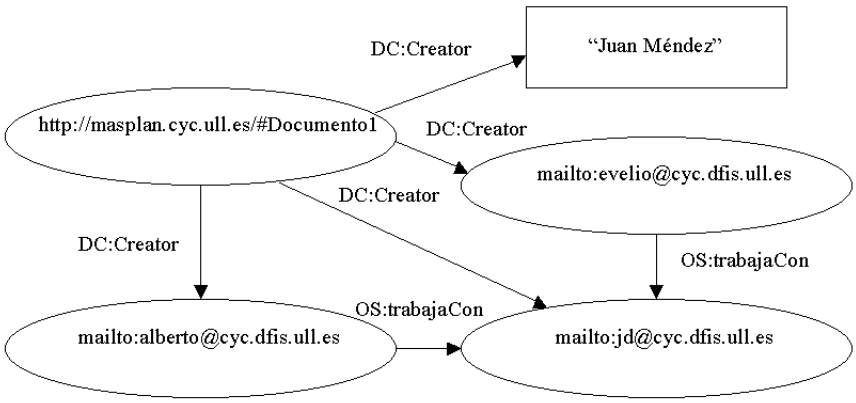

Figura 2.8.: Ejemplo de múltiples triples RDF.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/02/22-rdf-syntax-ns#"
  xmlns:DC="http://purl.org/dc/elements/1.0/"
  xmlns:OS="http://namespace.org/#" >
  <rdf:Description about =
   "http://masplan.cyc.ull.es/#Documento1">
    <DC:Creator>Juan Méndez</DC:Creator>
    <DC:Creator rdf:resource="mailto:jd@cyc.dfis.ull.es"/>
    <DC:Creator>
      <rdf:Description about="mailto:evelio@cyc.dfis.ull.es">
        <OS:trabajaCon rdf:resource="mailto:jd@cyc.dfis.ull.es">
```

[31] En realidad los *parsers* habitualmente generan un identificador para los recursos anónimos para evitar confusiones y poderlos distinguir entre ellos.

```
      </rdf:Description>
    </DC:Creator>
    <DC:Creator rdf:resource="mailto:alberto@cyc.dfis.ull.es"/>
  </rdf:Description>
  <rdf:Description about="mailto:alberto@cyc.ull.es">
    <OS:trabajaCon rdf:resource="mailto:jd@cyc.ull.es">
  </rdf:Description>
</rdf:RDF>
```

En caso de ambigüedad, se puede emplear el atributo `rdf:parseType` en los elementos de propiedad. Los valores permitidos para este atributo son *Resource* y *Literal*, empleado este último cuando un literal contiene etiquetas XML. De este modo se evita que sea procesado por un *parser* como una descripción.

Containers

Los contenedores (*Containers*, definidos por `rdfs:Container`[32]) hacen referencia a una colección de recursos. En RDF existen tres tipos de Container: `rdf:Bag` (una colección en la que el orden de los recursos no es importante, pudiendo contener recursos duplicados), `rdf:Seq` (una colección con un orden establecido y significativo, pudiendo contener recursos duplicados) y `rdf:Alt` (una serie de elementos que representan una elección entre ellos, pudiendo ser un elemento sustituido arbitrariamente por otro). Los elementos involucrados suelen etiquetarse con `<rdf:li>`[33] o ser numerados de la forma rdf:_1, rdf:_2...[34] El siguiente ejemplo representa un `<rdf:Bag>`.

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://w3.org/TR/1999/02/22-rdf-syntax-ns#">
  <rdf:Bag ID="mybag">
    <rdf:li resource="mailto:evelio@cyc.ull.es"/>
    <rdf:li resource="mailto:alberto@cyc.ull.es"/>
    <rdf:li resource="mailto:jose@cyc.ull.es"/>
  </rdf:Bag>
</rdf:RDF>
```

rdf:aboutEach, rdf:aboutEachPrefix

En algunas ocasiones, y para permitir la distribución de una descripción a través de una serie de recursos, una etiqueta `<rdf:Description>` puede, en vez del atributo `rdf:about`, tener un atributo `rdf:aboutEach` (su valor debe ser el URI de un `rdfs:Container`, aplicándose la descripción a todos los miembros del rdfs:Container) ó `rdf:aboutEachPrefix` (su valor debe ser una cadena de caracteres, aplicándose la descripción a todos los recursos cuya URI comienza con esa cadena de caracteres). Por ejemplo, si consideramos el `<rdf:Bag>` del ejemplo anterior, el

[32] El prefijo rdfs hace referencia a la definición del RDF Schema, como se verá en 2.4.7
[33] del inglés *list item* (elemento de una lista)
[34] que es lo que hará un parser RDF: sustituir cada `rdf:li` por las etiquetas numeradas.

siguiente extracto RDF se aplicará a los tres elementos definidos.

```
<rdf:Description aboutEach="#mybag">
  <OS:trabajaCon rdf:resource="mailto:jesus@cyc.ull.es">
</rdf:Description>
```

rdf:type

El elemento `<rdf:type>` especifica que un elemento `<rdf:Description>` se ajusta a una especificación dada. Por ejemplo, el siguiente extracto indica que el elemento referenciado por la URI relativa `#Macbeth` se ajusta a una definición de `Libro` (la cual se produce en otro lado del documento).

```
<rdf:Description about="#Macbeth">
<rdf:type resource="#Libro"/>
</rdf:Description>
```

2.4.6.3. Cosificación

La sintaxis XML del RDF proporciona un método de cosificación[35] (*reification*[36]), o de paso de lo abstracto a lo concreto, de las declaraciones de elementos (*statements*). Un arco simple puede ser cosificado añadiendo un atributo `rdf:ID` al elemento correspondiente a la propiedad, el cual define el URI de la declaración cosificada. De esta forma, las declaraciones cosificadas pueden accederse como si fuesen otros recursos.

Por ejemplo, consideremos la declaración vista en las subsecciones anteriores correspondiente a "El creador del Documento1 es Juan Méndez". Supongamos que ahora deseamos indicar que esa declaración ha sido creada a su vez por "Evelio González". Para poder hacerlo, se introduce un nuevo recurso (nuevo nodo en el grafo) referido a la declaración, indicando su tipo `rdf:Statement` y su sujeto (propiedad `rdf:subject`, en este caso con valor Documento1), su propiedad (`rdf:predicate` con valor `DC:Creator`) y su objeto (`rdf:object` con valor "Juan Méndez"). Una vez incluido, esta declaración puede emplearse como un recurso más, tal como se muestra en la Figura 2.9 [DMM00].

```
<rdf:Description ID ="EjemploStatement">
  <rdf:subject resource="http://masplan.cyc.ull.es/#Documento1"/>
  <rdf:predicate
   resource="http://purl.org/dc/elements/1.0/Creator"/>
  <rdf:object>Juan Méndez</rdf:object>
  <rdf:type resource=
   "http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement" />
  <DC:Creator>Evelio González</DC:Creator>
</rdf:Description>
```

[35] Según el diccionario RAE: Convertir algo en cosa. Considerar como cosa algo que no lo es, por ejemplo, una persona.

[36] Del latín res (cosa), facere (hacer)

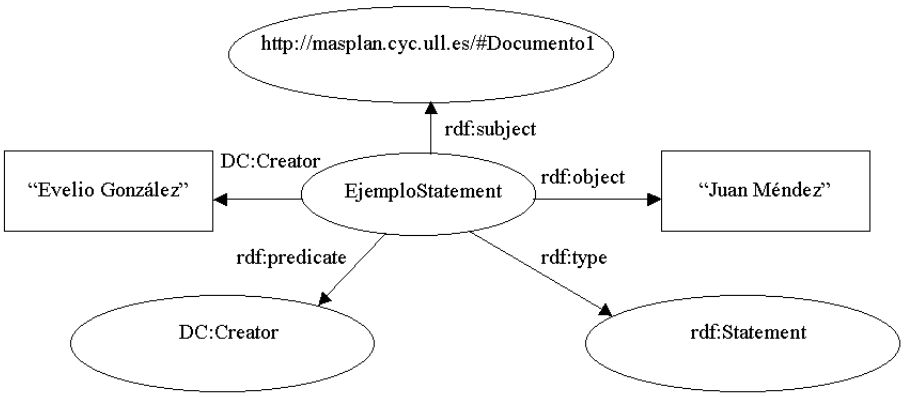

Figura 2.9.: Ejemplo de cosificación.

2.4.7. RDFS

RDF no define ningún vocabulario específico para regir las estructuras de datos. Por tanto, a semejanza de las DTDs y XML Schemas, es necesario definir un lenguaje de esquema, llamado RDF-Schema (RDFS), con las primitivas apropiadas para la definición de los términos empleado por un modelo de datos RDF: clases, propiedades... Éste es un lenguaje de ontologías que define el vocabulario básico de RDF que debe ser empleado en los triples RDF, en este sentido, es una ampliación del RDF. Se emplea el prefijo `rdfs` para abreviar el espacio de nombres del RDF Schema [NWC00]:

```
http://www.w3.org/2000/01/RDFSchema#
```

Con esta especificación se define una serie de primitivas de modelo adicionales. Entre las más destacadas se encuentran

- Para añadir a un documento RDF textos legibles para los usuarios humanos y que permitan una mejor interpretación del documento, RDFS proporciona las etiquetas `<rdfs:label>` (proporciona un nombre del recurso legible para las personas) y `<rdfs:comment>` (para dar una descripción más larga).
- `<rdfs:subPropertyOf>` indica que una propiedad está embebida en otra. Por ejemplo *tiene-madre* sería una subpropiedad de *tiene-progenitor*.
- `<rdfs:seeAlso>` y `<rdfs:isDefinedBy>` indican páginas web relacionadas que contienen información RDF adicional.
- `<rdfs:ConstrainResource>` y `<rdfs:ConstrainProperty>` definen mecanismos de restricción avanzados que no son cubiertos por el RDF Schema.
- `<rdfs:range>` y `<rdfs:domain>` definen el rango (valores posibles) y dominio (recursos que pueden ser sujeto de esa propiedad) de una propiedad.

Clases

Los recursos, siguiendo los conceptos introducidos en la OOP y en las ontologías, se dividen en grupos llamados clases (*classes*). Los miembros de una clase se conocen como instancias de una clase. Las clases a su vez son tratadas como recursos. RDF distingue entre una clase y el conjunto de sus instancias. La clase de los recursos que son clases RDF es referenciada como `<rdfs:Class>`. Para indicar que una clase es subclase de otra, se emplea la etiqueta `<rdfs:subClassOf>`.

```
<rdfs:Class rdf:ID="Hombre">
  <rdfs:label>Hombre</rdfs:label>
  <rdfs:comment>Subclase de Persona</rdfs:comment>
  <rdfs:subClassOf rdf:resource="#Persona"/>
</rdfs:Class>
```

Representación del RDF-Schema en RDF

El RDF-Schema se define él mismo empleando RDF (o sea un documento RDF que define el propio RDF), y a su vez existe un RDF-Schema que define el lenguaje RDF-Schema. Por tanto, y como el caso de XMLS, no hace falta aprender un nuevo lenguaje para su empleo. Como ejemplo, el siguiente extracto de código describe a la propiedad `<rdfs:subClassOf>`[37]

```
<rdf:Property rdf:about=
"http://www.w3.org/2000/01/rdf-schema#subClassOf">
  <rdfs:label>subClassOf</rdfs:label>
  <rdfs:comment>El concepto de subclase de una
clase</rdfs:comment>
  <rdfs:range rdf:resource=
"http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:domain rdf:resource=
"http://www.w3.org/2000/01/rdf-schema#Class"/>
</rdfs:Class>
```

2.4.8. Capa de Inferencia de Ontologías (OIL)

Capa de Inferencia de Ontologías (*Ontology Inference Layer*, OIL) es una propuesta para una representación basada en web y para dotar de capa de inferencia a las ontologías, que combina las primitivas de modelado empleadas ampliamente en los lenguajes basados en marcos con los servicios de razonamiento y semántica proporcionados por la lógica descriptiva (*description logic*, DL). Aunque no es un lenguaje de marcas, lo hemos incluido en este punto por cuestiones de incremento de la riqueza semántica en los lenguajes de representación de conocimiento. Es compatible en alto grado con la estructura de RDF Schema e incluye una semántica precisa para la descripción de los significados de los términos involucrados [DMM00].

[37] http://w3.org/TR/2003/WD-rdf-schema-20030123

OIL presenta un enfoque por capas para un lenguaje de ontología estándar. Cada capa adicional añade funcionalidad y complejidad a la capa anterior. De este modo, agentes (ya sean humanos o máquinas) que solamente puedan procesar una capa inferior, podrán entender, aunque sea parcialmente una ontología expresada en una capa superior [OIL]. Esta relación de capas se ve en la Figura 2.10.

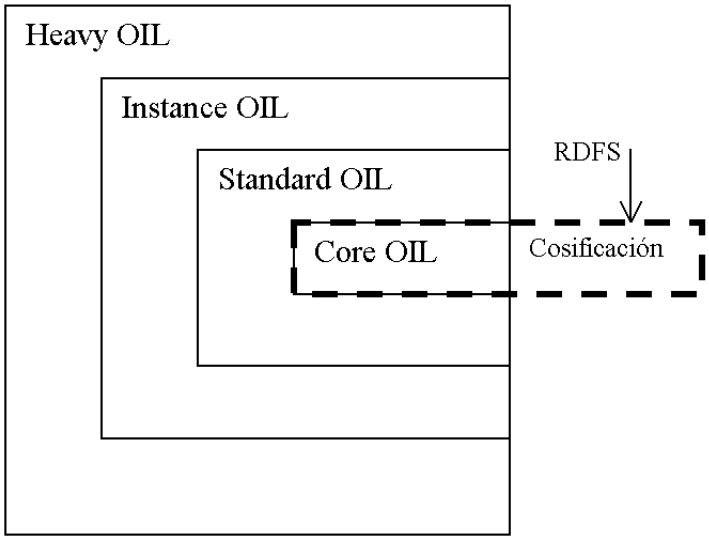

Figura 2.10.: Capas OIL [OIL].

Core Oil Constituye el núcleo de OIL. Coincide ampliamente con el RDF Schema (con la excepción de la cualidad de cosificación del RDFS - ver apartado 2.4.6.3). Esto significa que incluso los agentes RDFS son capaces de procesar hasta cierto grado las ontologías OIL.

Standard OIL Lenguaje que intenta capturar las primitivas de significado necesarias que sean entendibles y a la vez proporcionen un adecuado poder expresivo, permitiendo de este modo que la semántica sea especificada de un modo preciso.

Instance OIL Incluye una meticulosa integración de individuos. Mientras que la capa inferior (Standard OIL) incluía construcciones de modelado que permitían especificar valores de relleno (*filler)* referidos a individuos en las definiciones de términos, *Instance OIL* incluye una capacidad de base de datos consolidada.

Heavy OIL Podría incluir capacidades de representación (y de razonamiento) adicionales. Su sintaxis y correspondiente RDFS todavía no han sido definidos.

En los siguientes sub-apartados se describen brevemente los aspectos relativos al Standard OIL e Instance OIL [BBD+00].

2.4.8.1. Standard OIL

Una ontología en Standard OIL contiene descripciones de clases, slots e individuos. Las clases son colecciones de objetos. Las clases pueden estar relacionadas con otras clases declarando que una es subclase de otra, por ejemplo, una persona es una subclase de la clase mamífero. Así, cualquier instancia de persona debe ser también una instancia de mamífero. Expresado en OIL:

```
class-def persona
  subclass-of mamifero
```

Las clases de forma habitual contiene información sobre cómo se relacionan sus miembros con otros objetos. Estas relaciones binarias, los slots, pueden relacionarse con otros slots mediante el concepto de *subslot*. Por ejemplo, el slot *tiene-madre* puede ser definido del siguiente modo:

```
slot-def tiene-madre
  subslot-of tiene-progenitor
```

De este modo, se define el slot *tiene-madre* como un sub-slot de *tiene-progenitor*. Así, cualquier par *(x,y)* que se encuentra relacionado a través de *tiene-madre*, también lo estará mediante la relación *tiene-progenitor*.

Se puede limitar el valor de relleno del slot forzando a que sea de un tipo particular. Por ejemplo, uno puede imponer que el valor de relleno del slot *tiene-madre* debe ser de género femenino.

```
slot-constraint tiene-madre
  value-type genero-femenino
```

Cardinalidad

Entre las restricciones posibles está la de cardinalidad. Por ejemplo, se puede definir una clase con todas las instancias que tengan como máximo 2 hijos y como mínimo un sobrino (relaciones de máximo y mínimo a través de las etiquetas `max-cardinality` y `min-cardinality`)

```
slot-constraint tiene-familia
  max-cardinality 2 hijo
  min-cardinality 1 sobrino
```

Si se define la misma cardinalidad para el máximo y mínimo, el correspondiente código OIL (`cardinality`) es

```
slot-constraint tiene-familia
  cardinality 1 sobrino
```

Valores de relleno (*fillers*)

También se puede definir un conjunto posible de valores de relleno para un *slot*. Por ejemplo se definen los valores posibles para el color de un vino del siguiente modo:

```
slot-constraint tiene-color
  value-type (one-of Tinto Blanco Rosado)
```

El siguiente ejemplo referido a la definición de la clase `vino-blanco` muestra como fijar un valor de relleno determinado.

```
class-def vino-blanco
  subclass-of vino
  slot-constraint tiene-color
    has-filler blanco
```

Están permitidas las relaciones booleanas entre expresiones (`and`, `or`, `not`) incluso de forma recursiva.

```
  (vino-blanco and (vino-tinto or (not vino-dulce))
```

Standard OIL soporta dos tipos de datos: integer y string referidos respectivamente a enteros y cadenas de caracteres. Dentro de estos tipos, se puede definir un rango a través de las expresiones `(min x)`, `(max x)`, `(greater-than x)`, `(less-than x)`, `(equal x)` y `(range x y)`, donde `x` e `y` representan enteros o strings.

Tipo de definición de clases

El tipo de definición de clases puede ser *primitive* (primitivo) o *defined* (definido). El tipo será *primitive* (el tipo por defecto) si su definición debe tomarse como una condición necesaria, pero no suficiente para pertenecer a la clase. Por ejemplo, si definimos de forma *primitive* elefante como subclase de animal con el slot *color-piel* con un valor de relleno gris, se indica que todos los elefantes son grises, pero que un animal gris no necesariamente tiene que ser un elefante. Por el contrario, cuando una clase es defined, la definición correspondiente es suficiente y necesaria. Por ejemplo, un animal carnívoro es una subclase *defined* de animal que come carne. Todo carnívoro es un animal que come carne y todo animal que come carne es carnívoro.

Axiomas

Un axioma indica algún hecho adicional acerca de las clases de una ontología. Por ejemplo, las clases elefante y cocodrilo son disjuntas (o sea, que no existe ninguna instancia de elefante que a la vez sea una instancia de cocodrilo). Los axiomas permitidos son:

`disjoint` Las clases que aparecen a continuación (una lista de dos o más clases) son disjuntas dos a dos.

covered La forma es la de una expresión de una clase seguida por una lista de una o más expresiones de clases que cubren la expresión de clase inicial. Cada instancia de la primera clase es también una instancia de al menos una de las expresiones de clase de la lista. Por ejemplo,

```
covered animal by carnivoro herbivoro omnivoro mamifero
```

disjoint-covered La forma es la de una expresión de una clase seguida por una lista de una o más expresiones de clases que cubren la expresión de clase inicial y que son disjuntas entre sí. Cada instancia de la primera clase es también una instancia de exactamente una de las expresiones de clase de la lista.

```
disjoint-covered animal by carnivoro herbivoro omnivoro
```

equivalent Seguido por una lista de dos o más expresiones de clase. Las clases de la lista son equivalentes (es decir que tienen las mismas instancias).

```
equivalent leridano ilerdense
```

Rango y dominio de un *slot*

Respecto a los slots también pueden fijarse su dominio (*domain,* una lista de una o más clases que define el sujeto del slot), su rango (*range,* el objeto del slot) y su slot inverso (si el par (x,y) es una instancia del slot inicial, su inverso tendrá como instancia (y,x)). Por ejemplo,

```
slot-def come
  domain animal

slot-def edad
  range (min 0)

slot-def come
  inverse es-comido-por
```

deben ser interpretados respectivamente como cualquier individuo que come es un animal, el *slot* edad tiene como valores posibles los números enteros no negativos y, finalmente, si un par `(x,y)` está relacionados por el slot `x` come `y`, entonces también estarán relacionados mediante `y` es-comido-por `x`.

Propiedades de un *slot*

Finalmente, Standard OIL define las siguientes propiedades para un slot:

transitive Indica que el slot es transitivo. Si (x,y) e (y,z) son ambas instancias de un slot, entonces (x,z) también lo es. Por ejemplo

```
slot-def mas-grande-que
  properties transitive
```

`symmetric` Indica que el slot es simétrico. Si (x,y) es una instancia del slot, entonces (y,x) también lo es.

```
slot-def vive-con
  properties symmetric
```

`functional` Indica que el slot es funcional. Si (x,y) es una instancia del slot, entonces, no existe ningún z distinto de y tal que (x,z) sea una instancia de slot.

```
slot-def numero-DNI
  properties functional
```

2.4.8.2. Instance OIL

El Instance OIL es un superconjunto estricto de Standard OIL. Añade a él la posibilidad de definir instancias de clases y roles usando las siguientes dos construcciones:

`instance-of` Asegura que un individuo es una instancia de una clase o clases. Consiste en un nombre individual (un string) seguido por una o más expresiones de clase. El individuo debe ser una instancia de cada una de las expresiones de clase de la lista. Por ejemplo,

```
instance-of Evelio usuarioCYC
```

indica que el individuo *Evelio* es una instancia de la clase *usuarioCYC*.

`related` Asegura que un individuo se encuentra relacionado con otro individuo o valor de dato mediante una relación de slot. Consiste en el nombre de slot y un nombre de individuo seguido por un un segundo nombre de individuo o valor de dato. El primer individuo se relaciona con el segundo individuo o dato mediante la relación de slot. Por ejemplo,

```
related es-hijo-de George Zoe
```

indica que el individuo *George* es hijo del individuo *Zoe*.

2.4.8.3. Limitaciones de OIL

Las limitaciones actuales de OIL son:

- Aunque OIL proporciona un mecanismo para la herencia de valores desde las superclases, esos valores no pueden ser sobreescritos. Por tanto, esos valores no pueden ser empleados como valores por defecto.
- Solamente se permite un número fijo de tipos de axiomas (disjoint, covered, disjoint-covered y equivalent) y de propiedades algebraicas de slots, no cubriendo otros tipos de reglas/axiomas.
- OIL no proporciona soporte a dominios concretos (enteros, strings).
- El punto anterior también se aplica a las propiedades de los slots (inversa, funcional, transitiva, simétrica). Sería deseable la inclusión de otras propiedades como la reflexiva, irreflexiva, antisimétrica..., así como de relaciones compuestas.

2.4.9. DAML+OIL

DAML+OIL (DARPA Agent Markup Language + OIL)[38], a veces designado como simplemente DAML, es un lenguaje de marcas semántico para recursos Web que permite la definición de ontologías. Se basa en lenguajes de marcas estandarizados por el W3C como el RDF y RDFS, extendiendo estos lenguajes con primitivas más ricas, influenciadas directamente por el OIL, como pueden ser la de equivalencia y la de unicidad de una propiedad. A diferencia de otros lenguajes detallados en esta sección, no es un esfuerzo del W3C, sino de DARPA (*Defense Advanced Research Projects Agency*), que añade sus propios triples a semejanza de RDF, por lo que un desarrollador en XML o RDF no encuentra grandes dificultades para el uso de DAML. A pesar de no ser un desarrollo propio del W3C, existen numerosos enlaces en su página web referidos al DAML[39].

El lanzamiento de DAML tuvo lugar en agosto de 2000 bajo el nombre de DAML-ONT, el cual simplemente expresaba unas definiciones RDF (lenguaje al que se considera como no suficientemente estructurado) de clases más sofisticadas que las permitidas por el RDFS. Pronto los esfuerzos de DARPA se centraron en asimilar las relaciones soportadas por OIL.

DAML+OIL es un lenguaje fuertemente inspirado por la lógica descriptiva. Este tipo de lenguajes suele incluir las siguientes características adicionales a los lenguajes basados en marcos:

- Se puede especificar no solamente las condiciones necesarias, sino también las suficientes para la determinación de pertenencia a una clase. Por ejemplo, un vino producido en una bodega de la región de Burdeos es un vino Burdeos.
- Se puede emplear expresiones booleanas arbitrarias en las definiciones de clases y slots para especificar las superclases de una clase, el rango y dominio de un slot...
- Se puede especificar propiedades globales. Por ejemplo, *mayor que* es una propiedad *transitiva*, por tanto, si A es *mayor que* B, y B es *mayor que* C, entonces A es *mayor que* C.
- Se puede definir axiomas globales que expresen propiedades adicionales de las clases. Por ejemplo, la clase *Elefante* es disjunta con la clase *Cocodrilo*.

2.4.9.1. Sintaxis

Espacio de nombres

DAML está escrito en RDF, por lo que comparten estructura. Por tanto, el elemento raíz de un documento es `<rdf:RDF>` con las posibles declaraciones de los espacios de nombre. Éstos incluirán los espacios de nombres relativos al RDF, RDFS (vistas en secciones anteriores) y a la declaración de los triples DAML (http://www.daml.org/2001/03/daml+oil#).

[38] http://www.daml.org
[39] http://www.w3.org/TR/daml+oil-reference/

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:daml= "http://www.daml.org/2001/03/daml+oil#"
...
>
```

Con esta definición del espacio de nombres, los elementos de DAML+OIL serán accedidos mediante el prefijo `daml`.

Declaración de ontología

Lo primero que se debe indicar es que el documento en sí es una ontología. El valor del atributo `about` aparece normalmente vacío, lo que significa que el sujeto de la propiedad es el presente documento.

```
<daml:Ontology rdf:about="">
```

A través de la propiedad `<daml:versionInfo>`, el usuario puede indicar la versión de la ontología. Por otro lado, la propiedad `<daml:imports>` permite importar ontologías. Esta propiedad es transitiva: si una ontología A importa una ontología B, que a su vez importa una ontología C, entonces la ontología A importa tanto a B como C.

Definición de clases

La definición de clases se realiza con la etiqueta `<daml:Class>` del siguiente modo:

```
<daml:Class rdf:ID="Animal"/>
```

Esta definición solamente define la clase y le asigna un identificador Animal. A partir de ahora, esta clase se puede referenciar, por ejemplo, para definir una subclase. Esto se realiza aprovechando la propiedad correspondiente definida en RDFS.

```
<daml:Class rdf:ID="Hembra">
  <rdfs:subClassOf rdf:resource="#Animal"/>
</daml:Class>
```

Disyunción y herencia múltiple

DAML implementa el axioma OIL acerca de disyunción entre clases.

```
<daml:Class rdf:ID="Macho">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <daml:disjointWith rdf:resource="#Hembra"/>
</daml:Class>
```

La herencia múltiple entre clases está permitida.

```
<daml:Class rdf:ID="Hombre">
  <rdfs:subClassOf rdf:resource="#Persona"/>
  <rdfs:subClassOf rdf:resource="#Macho"/>
</daml:Class>
```

Definición de propiedades

Las propiedades se definen a través de <daml:ObjectProperty>, pudiéndose complementar con las etiquetas <rdfs:domain> y <rdfs:range>.

```
<daml:ObjectProperty rdf:ID="tieneProgenitor">
  <rdfs:domain rdf:resource="#Animal"/>
  <rdfs:range rdf:resource="#Animal"/>
</daml:ObjectProperty>
```

Las subpropiedades se definen con la etiqueta RDFS <rdfs:subPropertyOf>.

Restricciones

Un aspecto importante de DAML+OIL y que añade expresividad al lenguaje, es el empleo de la etiqueta <daml:Restriction>, la cual define una clase anónima cuyas instancias son todas las cosas que satisfacen la restricción. En el siguiente ejemplo se emplea esta etiqueta para describir características del concepto Persona.

```
<daml:Class rdf:ID="Persona">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#tieneProgenitor"/>
      <daml:toClass rdf:resource="#Persona"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction daml:cardinality="1">
      <daml:onProperty rdf:resource="#tienePadre"/>
    </daml:Restriction>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#tieneHermano"/>
      <daml:minCardinality>0</daml:minCardinality>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Así, en este ejemplo, indicamos que una Persona es un Animal, su progenitor es una Persona, tiene un padre y que puede tener o no hermanos. Obsérvese que únicamente indicamos características de la clase.

Propiedades de cardinalidad

En el anterior ejemplo, también se han introducido las propiedades de cardinalidad `<daml:cardinality>` (de exactitud), `<daml:minCardinality>` (de mínimo) y de `<daml:maxCardinality>` (de máximo). Cuando la restricción de `<daml:minCardinality>` es 0, no es necesario incluirlo en la ontología, ya que es sobreentendida. Una versión más sofisticada de cardinalidad es la siguiente, a través de `<daml:hasClassQ>` donde además se especifica el tipo de valor de la propiedad.

```
<daml:Class rdf:about="#Persona">
  <rdfs:subClassOf>
    <daml:Restriction daml:maxCardinalityQ="1">
      <daml:onProperty rdf:resource="#tieneOcupacion">
      <daml:hasClassQ rdf:resource="#ocupacionTiempoCompleto">
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Respecto a las propiedades de cardinalidad 1, se puede emplear la etiqueta `<daml:UniqueProperty>` (un sujeto únicamente puede relacionarse con un único objeto a través de la propiedad) y `<daml:UnambiguousProperty>` (cada objeto únicamente puede relacionarse con un único sujeto a través de la propiedad).

Propiedades `inverseOf`, `TransitiveProperty`, `samePropertyAs` y `complementOf`

También existe una implementación de las características OIL para las propiedades inversa `<daml:inverseOf>`, transitiva `<daml:TransitiveProperty>` y equivalente `<daml:samePropertyAs>`.

La etiqueta `<daml:complementOf>` determina el conjunto de elementos que no pertenecen a una clase. Por ejemplo, si queremos indicar que la clase Coche es una subclase de la clase de aquellos elementos que no son personas.

```
<daml:Class rdf:ID="Coche">
  <rdfs:subClassOf>
    <daml:Class>
      <daml:complementOf rdf:resource="#Persona"/>
    </daml:Class>
  </rdfs:subClassOf>
</daml:Class>
```

Colecciones `daml:Collection`

El DAML+OIL puede definir la disyunción de clases. El valor `daml:collection` indica que las clases embebidas deben tratarse como una unidad.

```
<daml:Disjoint rdf:parseType="daml:collection">
  <daml:Class rdf:about="#Coche"/>
  <daml:Class rdf:about="#Persona"/>
  <daml:Class rdf:about="#Planta"/>
</daml:Disjoint>
```

De un modo equivalente se puede definir el equivalente al *container* `alt`.

Unión Disyunta e intersección

Mediante la etiqueta `<daml:oneOf>` DAML permite identificar una clase como la unión disyunta de un conjunto de clases. Por ejemplo, la clase Persona se puede declarar como la unión disyunta de las clases Hombre y Mujer.

```
<daml:Class rdf:about="#Persona">
  <daml:disjointUnionOf rdf:parseType="daml:collection">
    <daml:Class rdf:about="#Hombre">
    <daml:Class rdf:about="#Mujer">
  </daml:disjointUnionOf>
</daml:Class>
```

Del mismo modo, se puede definir una nueva clase mediante la intersección de otras clases, a través de la etiqueta `<daml:intersectionOf>`.

Tipos de dato

En cuanto a la definición de tipos de datos definidos por el usuario, DAML+OIL ofrece los recursos provenientes de otros lenguajes como el XML Schema, mientras que el RDF aporta la definición de individuos [vHPSH01, CvHH$^+$01, ZFP$^+$01].

2.4.9.2. Lenguajes basados en DAML+OIL

DAML+OIL ha servido de base a diversos lenguajes. Entre los más significativos destacamos:

- DQL (DAML Query Language) es un lenguaje y protocolo formal para ser usado por parte de un agente que realiza una petición (referido como cliente) y un agente que responde (referido como servidor) en un diálogo petición-respuesta, empleando conocimiento representado en DAML+OIL. Una petición DQL contiene un patrón de petición (*query pattern*) que es una colección de sentencias DAML+OIL en las que algunos literales y/o URIs se sustituyen por variables. Una respuesta DQL proporciona enlaces de términos a esas variables

tales que la conjunción de las respuestas se vincula a una base de conocimiento (KB) llamada KB respuesta (*answer KB*) [Com02].

- DAML-S ha sido diseñado para habilitar el descubrimiento automatizado de servicios Web, proporcionando un lenguaje de marcas para codificar las propiedades y capacidades de un servicio Web de modo que estos servicios pueden ser incluidos en un registro más grande, o indexado a través de un motor de búsqueda o un sistema de casamiento [ABH+02].

2.4.9.3. Herramientas DAML+OIL

A pesar de su relativa juventud (la especificación inicial data del año 2000, y la especificación actual DAML+OIL data de marzo de 2001), su empleo se ha extendido rápidamente, siendo numerosas las herramientas (más de 65) relacionadas con este lenguaje, ya sea como editores de ontologías, motores de razonamiento, procesadores de documentos... Algunos de ellos serán analizados en la sección 2.5. Una lista actualizada de estas herramientas puede encontrarse en la dirección `http://www.daml.org/tools`.

2.4.10. Lenguaje Web de Ontologías (OWL)

El OWL (*Ontology Web Language*), como el caso del DAML+OIL, es una extensión del RDF Schema, cuya definición de estructura data del mes de marzo de 2003 y que en verano del mismo año ha adquirido el status de recomendación del W3C [W3C03b, W3C03a].

El propósito de este lenguaje es similar al de RDFS: proporcionar un vocabulario XML para la definición de clases, sus propiedades y las relaciones entre las clases. Sin embargo, permite al usuario expresar relaciones de mayor riqueza semántica, dotanto por tanto de una mayor capacidad de inferencia al procesamiento de un documento. En este sentido, es el paso más avanzado hacia la web semántica. En un esquema de evolución de los lenguajes de marcas:

- XML proporciona una sintaxis superficial para documentos estructurados, pero no impone ninguna restricción de carácter semántico sobre el significado de estos documentos.
- XML Schema es un lenguaje para restringir la estructura de documentos XML.
- RDF es un modelo de datos para objetos (recursos) y relaciones entre ellos, proporcionando una semántica sencilla para este modelo y pudiendo representarse con la sintaxis XML.
- RDF Schema es un vocabulario para describir propiedades y clases de los recursos RDF, con una semántica para jerarquías de generalización de dichas propiedades y clases.
- DAML+OIL añade más vocabulario para la descripción de propiedades y clases: entre otras, relaciones entre clases, cardinalidad, igualdad, nuevas propiedades y características de propiedades.

- OWL supone un enriquecimiento aún mayor del vocabulario, incluyendo por ejemplo las definiciones de las propiedades simétricas, inversas funcionales... Cuenta con la ventaja de que es el producto de un proceso de revisión más riguroso que el DAML+OIL. No obstante, y tal como se comprobará en esta sección, OWL es muy similar (salvo un par de construcciones posibles) al DAML+OIL. Esto se explica porque el W3C Web Ontology Group (diseñador del OWL) tomó a DAML+OIL como modelo.

2.4.10.1. Sintaxis

Definición del espacio de nombres

El espacio de nombres OWL corresponde al URI

```
http://www.w3.org/2002/07/owl#
```

Por tanto, la definición del espacio de nombres de un documento OWL se ajusta al siguiente esquema:

```
<rdf:RDF xmlns:owl="http://www.w3.org/2002/07/owl#"
...
>
```

Definición de propiedades

Como se ha explicado anteriormente, OWL es una extensión del RDFS, por lo que para definir una propiedad, los documentos OWL pueden hacer uso de las etiquetas de RDFS de caracterización de propiedades (`rdfs:range`, `rdfs:domain`, `rdfs:subPropertyOf`).

Una de las diferencias entre RDFS y OWL estriba en que en RDFS se emplea la etiqueta `rdf:Property` tanto para relacionar un recurso con otro como para relacionarlo con un `rdfs:Literal` o un tipo de dato. OWL distingue entre estos dos casos de propiedades.

- Se emplea `owl:ObjectProperty` para relacionar un recurso con otro.
- Se emplea `owl:DatatypeProperty` para relacionar un recurso con un `rdfs:Literal` o un tipo de dato construido en un XML Schema.

Ambos elementos son subclases de rdf:Property. Así tenemos las siguientes definiciones de ejemplo:

```
<owl:ObjectProperty rdf:ID="autor">
  <rdfs:domain rdf:resource="#Documento"/>
  <rdfs:range rdf:resource="#Persona"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="altura">
```

```
    <rdfs:domain rdf:resource="#Persona"/>
    <rdfs:range rdf:resource=
     "http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
  </owl:DatatypeProperty>
```

Del mismo modo, OWL permite expresar que una propiedad es simétrica del siguiente modo:

```
  <owl:ObjectProperty rdf:ID="viveCon">
    <rdf:type rdf:resource=
     "http://www.w3.org/2002/07/owl#SymmetricProperty"/>
    <rdfs:domain rdf:resource="#Persona"/>
    <rdfs:range rdf:resource="#Persona"/>
  </owl:ObjectProperty>
```

lo cual, empleando la etiqueta `owl:SymmetricProperty`[40], es equivalente a

```
  <owl:SymmetricProperty rdf:ID="viveCon">
    <rdfs:domain rdf:resource="#Persona"/>
    <rdfs:range rdf:resource="#Persona"/>
  </owl:SymmetricProperty>
```

De un modo análogo, las etiquetas `owl:TransitiveProperty`[41], `owl:FunctionalProperty`[42], `owl:inverseOf`[43] y `owl:InverseFunctionalProperty`[44] permiten definir respectivamente una propiedad transitiva, funcional, inversa e inversa funcional[45].

El empleo de estas etiquetas dota de capacidad de inferencia a un procesador OWL. Por ejemplo, de un código como el siguiente y teniendo en cuenta la definición anterior de la propiedad *viveCon*:

```
  <Persona rdf:ID="Cristina">
    <viveCon rdf:resource="#Ayoze"/>
  </Persona>
```

puede deducirse que el individuo *Ayoze* es una *Persona* (por la definición del rango) y que además *vive con* la Persona *Cristina* (por la declaración de *viveCon* como propiedad simétrica).

[40] subclase de `owl:ObjectProperty`, por tanto el rango de una propiedad simétrica solamente puede ser un recurso

[41] subclase de `owl:ObjectProperty`, por tanto el rango de una propiedad transitiva solamente puede ser un recurso

[42] subclase de `rdf:Property`, por tanto el rango de una propiedad funcional puede ser tanto un recurso como un literal o un tipo de dato

[43] propiedad

[44] subclase de `rdf:Property`,por tanto el rango de una propiedad inversa funcional puede ser tanto un recurso como un literal o un tipo de dato

[45] para un valor de rango el dominio es único.

Definición de clases mediante restricciones

Las clases OWL están pensadas para poder proporcionar una mayor expresividad que las clases del RDF Schema. Por tanto, OWL define una clase nueva, subclase de `rdfs:Class`, llamada `owl:Class`. Esta nueva clase permite, por ejemplo, redefinir el rango de una propiedad en una subclase. Por ejemplo, la clase *Animal* puede tener una propiedad *seAlimentaDe* con rango *Comida*. Por otro lado, una subclase *AnimalCarnivoro* restringe el rango de *seAlimentaDe* a *Carne*. Esto no se puede expresar usando RDFS, ya que `rdfs:range` impone una restricción global sobre el valor del rango en la clase padre y en todas sus subclases. En cambio, en OWL puede expresarse del siguiente modo:

```
<owl:Class rdf:ID="AnimalCarnivoro">
  <rdfs:subClassOf rdf:resource="#Animal"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#seAlimentaDe"/>
      <owl:allValuesFrom rdf:resource="#Carne"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Es decir, la clase AnimalCarnivoro es a la vez una subclase de Animal y de una clase anónima formada por el conjunto de instancias cuyo valor de la propiedad seAlimentaDe está restringido a las instancias de la clase Carne. Es decir *todos* los valores (`owl:allValuesFrom`) de *seAlimentaDe* en *AnimalCarnivoro* deben ser instancias de clase *Carne*. Este ejemplo también ha permitido introducir la definición de subclases a través del elemento `owl:Restriction`.

El anterior ejemplo contrasta con el siguiente:

```
<owl:Class rdf:ID="Plantilla_Equipo_Futbol">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#miembro"/>
      <owl:someValuesFrom rdf:resource="#Portero"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

En este caso, lo que se desea expresar que al menos uno de los valores (elemento `owl:someValuesFrom`) de la propiedad Miembro de las instancias de la clase de plantilla de un equipo de fútbol debe ser una instancia de la clase Portero.

El elemento `owl:hasValue` permite definir el valor de una propiedad. Por ejemplo, la clase ProfesorDeUnivesidad es una subclase de aquellas instancias cuya propiedad *trabajaEn* tiene el valor Universidad.

```
<owl:Class rdf:ID="ProfesorDeUniversidad">
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#trabajaEn"/>
        <owl:hasValue rdf:resource="#Universidad"/>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

En resumen, OWL proporciona tres modos en los que una clase puede restringir una propiedad:

`allValuesFrom` Todos los valores deben pertenecer a una cierta clase.

`someValuesFrom` Al menos un valor debe ser una instancia de una cierta clase.

`hasValue` La propiedad debe tener un valor específico.

Restricciones de cardinalidad

En este subapartado describiremos las relaciones de cardinalidad de las propiedades. Por ejemplo, la propiedad profundidadMaxima de una clase Oceano solamente puede tener un único valor (un océano solamente puede tener una profundidad máxima).

```
<owl:Class rdf:about="Oceano">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#profundidadMaxima"/>
      <owl:cardinality rdf:dataType=
        "http://www.w3.org/2001/XMLSchema#nonNegativeInteger">1
      </owl:cardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Del mismo modo, se pueden emplear las etiquetas `owl:minCardinality` y `owl:maxCardinality` para indicar las restricciones de cardinalidad máxima y mínima de la propiedad.

Propiedades equivalentes

Se pueden definir equivalencias entre diferentes propiedades a través del elemento `owl:equivalentProperty`.

```
<owl:DatatypeProperty rdf:ID="nombre">
  <owl:equivalentProperty rdf:resource=
   "http://purl.org/metadata/dublin-core#Title"/>
</owl:DatatypeProperty>
```

Construcción de Clases empleando Operadores de Conjunto

OWL proporciona la habilidad para construir clases empleando los siguientes operadores de conjunto:

`intersectionOf` La clase resultante es la intersección de las clases/restricciones definidas en el conjunto. Un empleo múltiple de `subClassOf` constituye un subconjunto de una intersección.

`unionOf` La clase resultante es la unión de las clases/restricciones definidas en el conjunto.

`complementOf` La clase resultante es el complemento de las clases/restricciones definidas en el conjunto.

Como ejemplo, se muestra a continuación el código OWL que indica que un afluente es un río que desemboca en otro río, o lo que es lo mismo, la clase Afluente es la intersección de la clase Río y de la clase anónima que contiene todas las instancias que desembocan en un río.

```
<owl:Class rdf:ID="Afluente">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Rio"/>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#desembocaEn"/>
      <owl:allValuesFrom rdf:resource="#Rio"/>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Obsérvese que en este caso estamos *definiendo* el Afluente como aquel Río que a la vez desemboca en otro Rio, en contraste con el uso de `</owl:Restriction>`, el cual se emplea para las *descripciones* de las clases.

Enumeración, equivalencia, disyunción

OWL proporciona la habilidad de poder construir una clase a partir de la enumeración de sus instancias (elemento `owl:oneOf`), especificar que una clase es equivalente a otra (`owl:equivalentClass`) y que dos clases son disjuntas (`owl:disjointWith`).

Como ejemplo, mostramos la definición de la clase IslasDeEspaña a partir de la declaración de todos sus elementos:

```
<owl:Class rdf:ID="IslasDeEspaña">
  <owl:oneOf rdf:parseType="Collection">
    <Isla rdf:about="Tenerife"/>
    <Isla rdf:about="Mallorca"/>
    ...
```

```
    </owl:oneOf>
  </owl:Class>
```

Declaraciones OWL

El resto de declaraciones OWL permite definir que dos instancias son iguales (`owl:sameIndividualAs`) o diferentes (`owl:differentFrom`). Cuando se quiere expresar que un conjunto de instancias son diferentes entre sí se emplea el elemento `owl:AllDifferent`.

```
<owl:AllDifferent>
  <owl:distinctMembers rdf:parseType="Collection">
    <Isla rdf:about="Tenerife"/>
    <Isla rdf:about="Mallorca"/>
    ...
  </owl:distinctMembers>
</owl:AllDifferent>
```

OWL define la clase `owl:Thing` que es la superclase de todas las clases.

2.4.10.2. OWL Lite, OWL DL, OWL Full

Todas las aplicaciones no precisan de todas las capacidades que ofrece OWL. Por este motivo, existen tres versiones de OWL acumulativas entre sí. De mayor a menor capacidad,

OWL Full Ofrece todas las posibilidades expresadas en los subapartados anteriores. Además, se pueden mezclar RDFS con OWL. Se emplea por los usuarios que desean la mayor capacidad de expresión sin garantía computacional. Por ejemplo, en OWL Full una clase puede tratarse simultáneamente como una colección de individuos o como un individuo en sí mismo. Por tanto, se considera poco probable que ningún software de razonamiento sea capaz de dar soporte a todas las capacidades OWL.

OWL DL No se puede emplear `owl:cardinality` con `TransitiveProperty`. Una ontología DL no puede importar una ontología que emplee OWL Full. No se pueden emplear una clase como miembro de otra clase, es decir, no se pueden tener metaclases. `FunctionalProperty` e `InverseFunctionalProperty` no se pueden usar con tipos de datos. Se emplea por aquellos usuarios que desean la mayor expresividad de OWL permitida por una completitud (se asegura que todas las conclusiones serán computadas) y determinación (todas las computaciones terminarán en un tiempo finito). Su nombre deriva de la lógica descriptiva (*Description Logics*).

OWL Lite Presenta todas las restricciones de OWL DL más las siguientes. No se puede emplear `owl:minCardinality`, `owl:maxCardinality`, `owl:hasValue`, `owl:disjointWith`, `owl:oneOf`, `owl:complementOf` ni `owl:unionOf`. Los únicos valores permitidos para `owl:cardinality` son 0 y 1.

La ventaja de emplear OWL Full es obviamente que se dispone de todo el poder que ofrecen las declaraciones OWL. Sin embargo, resulta muy complicado construir una herramienta capaz de procesar completamente todas esas declaraciones. Las herramientas para procesar OWL DL y OWL Lite pueden construirse de una manera más rápida y fácil.

2.4.10.3. Ejemplo OWL

Para finalizar, reproducimos brevemente un ejemplo de un caso donde se manifiesta el poder de OWL [CJ03b].

En este ejemplo, se supone que se ha producido un robo. En la huida, el ladrón pierde su pistola. En una supuesta comisaría de policía se rellena el siguiente informe:

```
<Robo>
  <fecha>...</fecha>
  <descripcion>...</descripcion>
  <evidencia>
    <Pistola>
      <numeroSerie>2222</numeroSerie>
    </Pistola>
    <ladron>
      <Persona><!--desconocido></Persona>
    </ladron>
  </evidencia>
</Robo>
```

A continuación la policía detiene un coche por exceso de velocidad. El oficial de tráfico rellena el siguiente informe:

```
<IncidenteTrafico>
  <fecha>...</fecha>
  <descripcion>...</descripcion>
  <conductor>
    <Persona>
      <nombre>Fred Blogs</name>
      <numeroCarnet>333333</numeroCarnet>
    </Persona>
  </conductor>
</IncidenteTrafico>
```

En la comisaría, una computadora recibe cada informe por separado. Esta computadora analiza la información de ambos informes y descubre la existencia una licencia de armas relativa a una pistola con el mismo número de serie que la empleada en el robo. En este registro del arma aparece un número de carnet que coincide con el de la persona detenida por exceso de velocidad.

```
<LicenciaArmas>
```

```
  <arma>
    <Pistola>
      <numeroSerie>2222</numeroSerie>
    </Pistola>
  </arma>
  <propietario>
    <Persona>
      <numeroCarnet>333333</numeroCarnet>
    </Persona>
  </propietario>
</LicenciaArmas>
```

Sin embargo todavía hay una serie de preguntas que deben ser respondidas antes de poder detener a Fred Blogs como sospechoso del robo. Estas preguntas pueden responderse con una ontología (llamada por ejemplo Policia.owl).

1. ¿Pueden varias pistolas tener el mismo número de serie? En caso afirmativo, Fred Blogs puede tener una pistola con el mismo número de serie que otra posible arma empleada en el robo. Y por lo tanto, Fred Blogs no sería sospechoso. Sin embargo, la siguiente regla informa que cada pistola se identifica de modo único con su número de serie.

```
<owl:InverseFunctionalProperty rdf:ID="numeroSerie">
  <rdfs:domain rdf:resource="#Pistola"/>
  <rdfs:range rdf:resource=
   "http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:InverseFunctionalProperty>
```

2. ¿Pueden varias personas tener el mismo número de carnet de identidad? De ser así, la licencia de armas puede estar referida a otra persona. Del mismo modo que en la pregunta anterior, la ontología informa que cada carnet de identidad se identifica de modo único con una Persona.

```
<owl:InverseFunctionalProperty rdf:ID= "numeroCarnet">
  <rdfs:domain rdf:resource= "#Persona"/>
  <rdfs:range rdf:resource=
   "http://www.w3.org/2001/XMLSchema#nonNegativeInteger"/>
</owl:InverseFunctionalProperty>
```

3. ¿Puede una misma arma estar registrada en varias licencias de armas? En caso afirmativo, entonces las otras licencias pueden referirse al verdadero poseedor de la pistola. Para responder a esta pregunta, la siguiente regla indica que cada pistola se asocia a una única licencia de armas.

```
<owl:InverseFunctionalProperty rdf:ID="arma">
  <rdfs:domain rdf:resource="#LicenciaArmas"/>
  <rdfs:range rdf:resource="#Pistola"/>
</owl:InverseFunctionalProperty>
```

4. ¿Puede una licencia de armas tener múltiples dueños de una determinada arma registrada? Si fuese así, entonces podría haber otra parte del documento (no disponible para la comisaría) acerca de la misma arma, pero con un diferente propietario. Esta hipótesis es desechada con la siguiente regla que afirma que cada licencia de armas se refiere a una única pistola y a una única persona.

```
<owl:Class rdf:ID="LicenciaArmas">
  <owl:intersectionOf rdf:parseType="Collection">
    <owl:Restriction>
      <owl:onProperty rdf:resource= "#arma"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#propietario"/>
      <owl:cardinality>1</owl:cardinality>
    </owl:Restriction>
  </owl:intersectionOf>
</owl:Class>
```

Una vez respondidas estas respuestas, ya se puede establecer, sin lugar a dudas que Fred Blogs es el dueño de la pistola y por tanto debe ser detenido como sospechoso del robo.

2.5. Herramientas de Ontologías y Lenguajes de Marcas

Finalizaremos el presente capítulo describiendo un amplio conjunto de herramientas disponibles por una parte para el desarrollo y utilización de ontologías y por otra para el procesamiento de documentos basados en lenguajes de marcas. Comenzaremos con *OpenCyc* una herramienta para el empleo de las ontologías Cyc. Seguiremos con dos herramientas para el procesamiento de documentos basados en lenguajes de marcas: *Castor* y *Jena*. A continuación describiremos tres herramientas para la edición de ontologías: *OilEd*, *Protégé* y *OntoEdit*. Las siguientes herramientas se refieren a la visualización de documentos (*DAMLViewer* y *VisioDAML*) y anotación DAML+OIL (*AeroDAML*). RDF ha dado lugar a una herramienta que define su propio lenguaje en capas de consulta, inferencia y transformación para la Web Semántica, el *TRIPLE*, descrito en la siguiente subsección. Entre los analizadores disponibles de las ontologías DAML+OIL hemos seleccionado *DAML+OIL Ontology Checker* y

DAML Validator. La última herramienta analizada es *OWLConverter*, un traductor de ontologías DAML+OIL a OWL.

2.5.1. OpenCyc

OpenCyc[46] es la versión en código abierto de la tecnología Cyc, lo cual incluye su base de conocimiento y su motor de razonamiento. Cycorp, los constructores de Cyc, crearon otra organización llamada OpenCyc.org, encargada de divulgar y administrar OpenCyc.

Esta herramienta puede ser empleada como la base de una amplia variedad de aplicaciones inteligentes como, por citar solamente algunas, el reconocimiento del habla, la integración de bases de datos, el desarrollo de ontologías y la priorización del correo electrónico.

La versión actual de OpenCyc[47] es la 0.7 beta. Se pretende que la versión 1.0 incluya entre otras características unos 6000 conceptos, aproximadamente 60000 declaraciones acerca de esos conceptos y una versión compilada del motor de inferencia y del navegador de bases de conocimiento Cyc.

2.5.2. Castor

Castor[48] es una API XML que a diferencia de la mayoría de las otras APIs XML, no trata con la estructura de un documento XML, sino con los datos definidos en dicho documento a través de un modelo de objetos que representa esos datos. En resumen, es capaz de realizar la conversión entre XML y objetos Java. Para ello es capaz de realizar las acciones de *marshalling* (conversión de un objeto en una secuencia de bytes) y su inversa *unmarshalling* (conversión de una secuencia de bytes a objeto).

2.5.3. Jena

Jena[49] es una API de código abierto en Java[50] para manipular los modelos RDF. Comprende un número de módulos que entre ellos dan soporte a varios formatos para el procesamiento y salida de datos RDF, métodos adecuados para manipular y realizar consultas dichos datos así como facilidades para la persistencia de los datos en una variedad de bases de datos. La versión 1.6 incluye una API para la manipulación de ontologías expresadas en DAML. Sin embargo, a partir de la versión 2.0[51] esta API ha sido re-implementada mediante una API de ontología que permite tratar con ficheros RDFS, DAML y OWL.

Los módulos contenidos en Jena más significativos son:

[46] http://opencyc.org
[47] Existe una versión DAML de la versión 0.6.0b en http://www.cyc.com/2002/04/08/cyc.daml
[48] http://castor.exolab.org
[49] http://www-uk.hpl.hp.com/people/bwm/rdf/jena/
[50] necesita Java 2 para actuar. Compatible con JDK 1.4, pero que puede emplearse con una versión anterior.
[51] lanzada el 28 de agosto de 2003

2.5.3.1. RDF API

RDF API[52] es un API implementado en Java que define interfaces para el procesamiento y acceso a los modelos RDF vistos como conjuntos de declaraciones. Es el núcleo de Jena para manipular las colecciones de declaraciones RDF. Proporciona, entre otras, las siguientes facilidades:

- Métodos centrados en declaraciones para manipular un modelo RDF como un conjunto de triples RDF.
- Métodos centrados en recursos para manipular un modelo RDF como un conjunto de recursos con propiedades.
- Soporte para los contenedores RDF: bag, alt y seq.
- La aplicación puede extender el comportamiento de los recursos.
- Soporte de salida y procesamiento en formatos XML y RDF.
- Diseño modular que emplea interfaces Java para permitir a los desarrolladores su extensión.

Aparte del módulo principal, la distribución RDF API incluye otra serie de módulos adicionales:

- Soporte para UML sobre RDF.
- Herramienta para la generación de clases con "constantes" Java para la manipulación simplificada de esquemas.
- Procesador para una sintaxis simplificada de RDF.
- Utilidad para crear identificadores URI únicos.

Aparte de Jena, muchas otras herramientas (OntoEdit, Protégé 2000...) emplean este API para procesar documentos RDF.

2.5.3.2. ARP Parser

ARP es un procesador estándar completo para la sintaxis RDF/XML. Tiene un diseño modular basado en XML Infoset[53]. Ofrece entre otras las siguientes ventajas:

- Es adecuado para grandes ficheros.
- Procesamiento de errores altamente configurable.
- Procesamiento de RDF/XML tanto embebido como aislado.
- Procesamiento basado en Xerces[54].

[52] http://www-db.standford.edu/~melnik/rdf/api.html

[53] XML Information Set. Recomendación W3C que proporciona un conjunto de definiciones para el uso de otras especificaciones que necesitan referirse a la información en un documento XML. Define únicamente una lista de partes para documentos XML y proporciona un sistema por el cual los desarrolladores que emplean XML pueden describir con cuáles de estas partes de XML trabajan. http://www.w3.org/TR/2001/REC-xml-infoset-20011024.

[54] http://xml.apache.org/xerces-j/

2.5.3.3. RDQL query language

RDQL es una implementación de un lenguaje de consultas al estilo SQL para RDF. Trata el RDF como datos y proporciona consultas con patrones y restricciones en forma de triple sobre un modelo RDF sencillo. Este lenguaje se basa en SquishQL[55]. Algunas de sus ventajas son:

- Lenguaje parecido al SQL.
- Soporte en línea de comando para explorar los conjuntos de datos.

2.5.3.4. DAML API

Este módulo está construido sobre el RDF API y permite el procesamiento de documentos DAML+OIL. Entre sus características destacan:

- Métodos para acceder a propiedades de las clases y propiedades DAML.
- Reconocimiento de múltiples vocabularios (incluyendo las especificaciones DAML+OIL de diciembre de 2000 y marzo de 2001).
- Procesamiento automático de las declaraciones `import`.
- Reconocimiento de las propiedades transitivas e inversas.

2.5.3.5. Jena 2 Ontology API

A fecha de escritura de este trabajo, se ha lanzado una nueva versión de Jena. Dicha versión incluye esta API que permite tratar con ficheros RDFS, DAML+OIL y OWL. En la misma distribución se advierte al usuario de la posible presencia de algunos errores o inconsistencias. Para los usuarios que deseen migrar su código desde las versiones 1.x hasta esta nueva versión, se proporciona una re-implementación del DAML API, la cual no se asegura su presencia en futuras distribuciones de la herramienta.

2.5.4. OilEd

Figura 2.11.: Logotipo de OilEd.

[55] http://swordfish.rdfweb.org/rdfquery

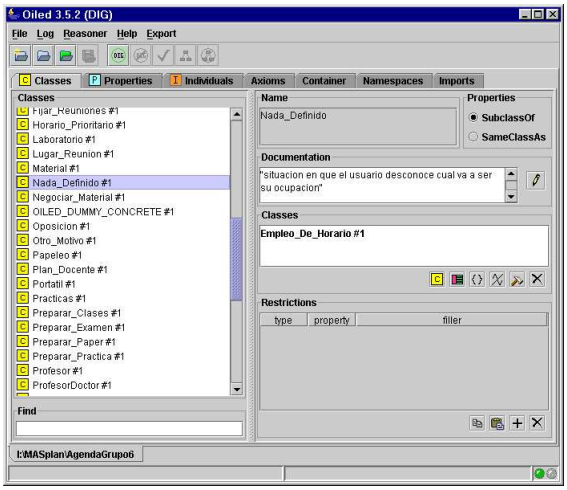

Figura 2.12.: Entorno de edición de ontologías OilEd.

OilEd[56] es un editor sencillo implementado principalmente en Java que permite al usuario crear y editar ontologías basadas en OIL. Su intención no es convertirse en un entorno completo de desarrollo de ontologías, sino proporcionar al usuario una demostración de su empleo. En este sentido, no es adecuado para la edición de ontologías de gran tamaño. La versión a fecha de escribir este trabajo es la 3.5.2 disponible para Windows y Linux bajo licencia GPL.

El diseño de esta herramienta (ver Figura 2.12) es similar al de otras herramientas como Protégé y OntoEdit, pero extendiendo sus características. Su funcionalidad permite la definición y descripción de las clases, slots, individuos y axiomas que constituyen una ontología. En general, las funciones de edición son accedidas a través de un modo gráfico (menú, botones...) en forma estándar.

El componente central de OilEd es la llamada descripción de marcos (*frame description*) . Consiste en una colección de superclases con una serie de slots de restricciones. Una característica propia de esta herramienta es que en cualquier lugar en que pueda aparecer el nombre de una clase se puede emplear la descripción de un marco anónimo definido recursivamente. Además pueden aparecer combinaciones booleanas arbitrarias de marcos o clases (empleando los operadores *and*, *or* y *not*). Este hecho contrasta con otros sistemas donde en general, las restricciones de slots y superclases deben ser nombres de clases.

OilEd permite la definición de las clases: nombre de la clase, una descripción opcional y una especificación sobre si la definición en cuestión es primitiva o definida. En la especificación OIL, como se vio en la sección 2.4.8, se permiten las definiciones múltiples de clases. En cambio, y por razones de implementación estas definiciones múltiples no son permitidas, lo cual no supone restricción alguna, ya que se puede hacer uso, en su caso, de lo axiomas de equivalencia y de la herencia múltiple.

En cuanto a la definición de los slots, ésta se realiza dando el nombre del mismo y algunas características adicionales como la dependencia de un superslot (denominado

[56] http://OilEd.man.ac.uk/

así por analogía con el término superclase), de un slot inverso, la definición del rango y dominio (de nuevo incidimos sobre el hecho de que puede aparecer marcos anónimos o combinaciones booleanas de clases o marcos) o la declaración del slot como transitivo, funcional o simétrico.

Respecto a los axiomas, OilEd proporciona la capacidad de expresar la disyunción, equivalencia y cubrimiento de clases (como en todos los casos, también se incluyen los marcos y las combinaciones booleanas detalladas anteriormente). OilEd permite la definición de individuos, aunque con algunas limitaciones en cuanto a la utilización de estas instancias en la definición de propiedades. Finalmente, se pueden emplear tipos de datos concretos (cadena de caracteres y enteros) así como expresiones relativas (cardinalidad, dominios y rangos) en las descripciones de clases.

Aunque OilEd se diseñó inicialmente como editor de ontologías en OIL, actualmente permite su exportación a otros formatos: Standard OIL, OIL-RDFS (una seriación RDFS de OIL), DAML+OIL y OWL. Además permite exportar las ontologías a HTML, facilitando de este modo su visualización a través de un navegador.

2.5.4.1. Elemento de Razonamiento FaCT

Aparte de lo comentado en el subapartado anterior, una de las principales aportaciones de la herramienta es su empleo de razonamiento para comprobar la consistencia de las clases e inferir relaciones subsumidas. Los servicios de razonamiento son proporcionados en la actualidad por el sistema FaCT (Clasificación Rápida de Terminologías*, Fast Classification of Terminologies*), pero en realidad se puede emplear cualquier elemento de razonamiento con una adecuada capacidad de conexión.

FaCT[57] es un clasificador DL escrito en Common Lisp que ofrece una completa capacidad de razonamiento para dos DLs llamadas $\mathcal{SHF}$ y $\mathcal{SHIQ}$[58]. Sus características más significativas son su lógica expresiva (especialmente a través del razonador $\mathcal{SHIQ}$), su implementación optimizada de descripción gráfica (que se ha convertido en el estándar en sistemas de DL) y su arquitectura cliente-servidor basada en CORBA. Con esta arquitectura[59], OilEd permite una conexión con un servidor razonador FaCT (preferentemente ejecutándose en una máquina potente) para poder llevar a cabo más rápidamente la comprobación de la consistencia de las clases de la ontología en concreto.

De este modo, se pretende procesar las ontologías basadas en OIL. Las optimizaciones FaCT están pensadas específicamente para la mejora de la actuación del sistema a la hora de clasificar ontologías realistas. Estas optimizaciones mejoran en varios órdenes de magnitud cuando se comparan con los antiguos razonadores basados en DLs y en lógica modal.

A nivel práctico, cuando se pide la verificación de la ontología, ésta es traducida a una base de conocimiento equivalente (en la lógica descriptiva correspondiente) y se envía al razonador para la clasificación. Los resultados son enviados al usuario

[57] http://www.cs.man.ac.uk/~horrocks/FaCT/

[58] El estudio de estas lógicas descriptivas excede los límites de este trabajo. El lector interesado tiene más información en la página web de FaCT.

[59] Para la interfaz ORB se necesita la versión 1.2 de Java

resaltando las clases inconsistentes y reorganizando la jerarquías de clases para mostrar los cambios descubiertos [BHGS01].

2.5.5. Protégé-2000

Figura 2.13.: Logotipo de Protégé-2000.

Protégé-2000[60] es una sencilla pero poderosa herramienta bajo la *Open Source Mozilla Public License* implementada en Java[61] para la construcción de modelos de dominio. Protégé-2000 fue desarrollada inicialmente por el *Medical Informatics Group* de la Universidad de Stanford para dar soporte a la adquisición de conocimiento para sistemas expertos médicos, aunque actualmente se ha convertido en muy popular para otros muchos propósitos. Protégé-2000 es de código abierto disponiendo a fecha de escritura de este trabajo de más de 7500 usuarios registrados. Se trata de una herramienta gráfica para la edición de ontologías y adquisición del conocimiento que se puede adaptar para habilitar el modelado conceptual con los nuevos lenguajes diseñados para la web Semántica. Por lo tanto, permite pensar acerca de los modelos de dominio sin necesidad de conocer la sintaxis del lenguaje empleado.

Se puede emplear Protégé-2000 para los siguientes propósitos:

- Modelado de clases. Protégé-2000 proporciona una GUI que modela clases (conceptos del dominio) con sus atributos y relaciones.
- Edición de instancias. De las clases anteriores, Protégé-2000 genera de forma automática formularios interactivos que permite la entrada de instancias válidas.
- Procesamiento de modelos. Protégé-2000 dispone de una librería de *plug-ins* que ayudan a definir la semántica, realizar consultas y definir un comportamiento lógico.
- Intercambio de modelos. Los modelos resultantes (clases e instancias) pueden ser cargadas y salvadas en varios formatos, incluyendo XML, UML y RDF.

Una de las características más interesantes, desde el punto de vista del programador, es que proporciona una API de código abierto para poder conectar a Protégé-2000

[60] http://protege.stanford.edu

[61] Para poder ejecutar la aplicación es necesaria la versión JDK 1.3 o superior

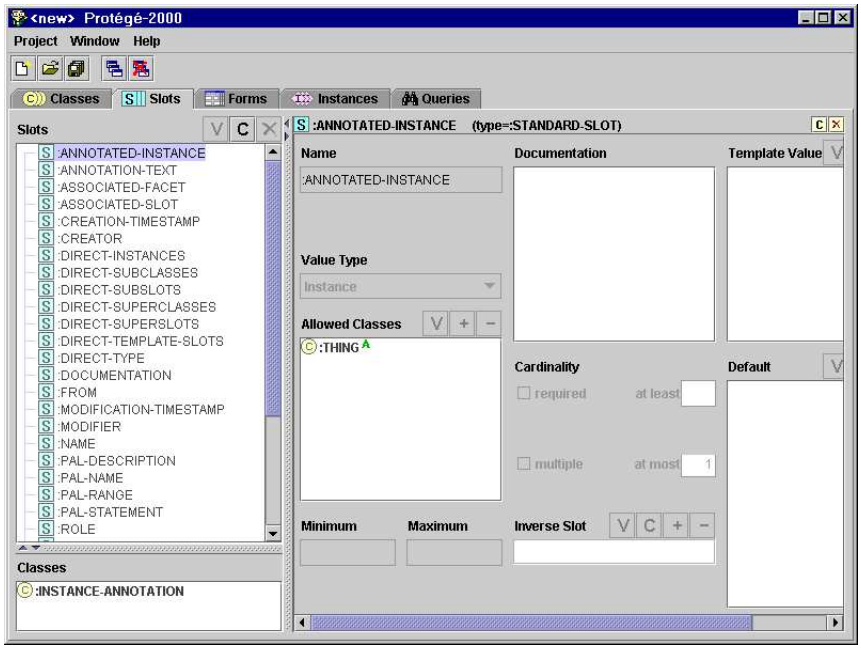

Figura 2.14.: Entorno de edición de ontologías Protégé-2000.

otros componentes Java y permitir el acceso a los modelos desde otras aplicaciones. Protégé-2000 también permite modificar la estructura gráfica para adaptarla mejor al lenguaje escogido de representación del conocimiento.

El modelo de conocimiento de Protégé-2000 es compatible completamente con OKBC. Protégé-2000 implementa todo lo que necesita el modelo de conocimiento OKBC y todo en el modelo de conocimiento Protégé-2000 es consistente desde el punto de vista lógico con OKBC. Para las características donde OKBC permite flexibilidad, Protégé-2000 preserva tanto como es posible la generalidad OKBC, restringiéndola únicamente debido a las necesidades de su interfaz de usuario. La tabla 2.1 resume aquellos puntos en que el modelo de conocimiento Protégé-2000 es menos general que el de OKBC. Protégé-2000 también extiende algunas de las características del modelo de conocimiento OKBC, en el sentido de que éste no las prohibe. Una de estas extensiones es la arquitectura de metaclases. Una metaclase se define como una plantilla que se emplea para definir nuevas clases en la ontología. De este modo se permite a los usuarios emplear Protégé-2000 como un editor para la representación del conocimiento con modelos diferentes del de la herramienta.

Protégé-2000 también puede adaptarse para poder ser empleado como editor RDF. La definición de una metaclase RDF no soluciona todos los problemas de compatibilidad entre los modelos de conocimiento Protégé-2000 y RDF. Es necesario incluir un componente que traduzca la base de conocimiento entre la representación interna de Protégé-2000 y los documentos RDF. Este componente debe resolver las diferencias entre ambos modelos que se resumen en la tabla 2.2 [GMF+03, NFM00, Knu03].

Para trabajar con ontologías DAML+OIL y OWL son necesarios respectivos *plug-*

Cuadro 2.1.: Diferencias entre los modelos de conocimiento de Protégé-2000 y OKBC

Protégé-2000	OKBC
Un marco solamente puede ser instancia de una sola clase	Un marco puede ser una instancia de múltiples clases
Un marco siempre es una instancia de una clase	Un marco no tiene que ser una instancia de alguna clase
Un slot propio adjuntado a un marco se deriva de la correspondiente plantilla de slot	Un slot propio puede ser adjuntado directamente a cualquier marco
Todas las clases, slots, facetas e individuos son siempre marcos	Clases, slots, facetas e individuos no tienen que ser obligatoriamente marcos
Un marco es una clase, un slot o una faceta	Un marco puede ser una clase, un slot y una faceta al mismo tiempo

ins[62].

2.5.6. OntoEdit

OntoEdit[63] (Figura 2.15) es un entorno de ontologías implementado en Java[64] y desarrollada en la Universidad de Karlsruhe que da soporte al desarrollo y mantenimiento de ontologías de forma gráfica.

La base de OntoEdit consiste en un poderoso modelo de ontologías interno. Este paradigma soporta un modelado neutral respecto a lenguaje de representación. Varias vistas gráficas de las estructuras contenidas en la ontología dan soporte al modelado de las diferentes fases de construcción de ontologías.

La herramienta está basada en un marco flexible. Primero, permite extender la funcionalidad de una forma modular. La interfaz está abierta a posibles extensiones de sus funcionalidades por parte de sus usuarios. Por otro lado, se encuentran disponibles una serie de *plug-ins* como por ejemplo de inferencia, de importación/exportación, algunos de ellos disponibles solamente en la versión profesional de la herramienta. De este modo, los usuarios pueden seleccionar solamente aquellos *plug-ins* necesarios.

Esta estructura de *plug-ins* permite su distribución en tres versiones diferentes: una gratuita OntoEdit Free, y dos comerciales: OntoEdit y OntoEdit Professional [Ont03].

[62] http://www.ai.sri.com/daml/DAML+OIL-plugin/
http://protege.stanford.edu/plugins/owl/

[63] la versión a fecha de escritura de este trabajo de OntoEdit es la 2.6. Página web: http://www.ontoprise.de

[64] Necesita al menos una CPU Pentium II (400 MHz), 128 MB de RAM, 20 Mb de espacio libre en el disco duro y tener instalado el Java Runtime Environment JRE 1.4

Cuadro 2.2.: Diferencias entre los modelos de conocimiento de Protégé-2000 y RDF, RDF-Schema

Protégé-2000	RDF, RDF-Schema
Un marco solamente puede ser instancia de una sola clase	Un recurso puede ser una instancia de múltiples clases
El valor de un slot puede ser un valor de un tipo primitivo o una instancia de una clase. Puede haber una o más clases que restrinjan el valor	El rango de una propiedad es una clase simple que restringe los objetos de la propiedad a instancias de esa clase
Un slot en una instancia individual y no puede ser una subclase de otro slot	Una propiedad puede ser una especialización de otra propiedad
Las colecciones son implementadas como listas	Hay tres tipos de objetos contenedores: bag, sequence y alternative

OntoEdit Free Contiene las características básicas necesarias para modelar una ontología. Permite importar, exportar y visualizar ontologías en múltiples lenguajes, como RDF(S), OXML[65], DAML+OIL... Permite también la gestión de múltiples ontologías e importar datos provenientes de Excel y estructuras de directorios. Incluye sendos *plug-ins* de editor de instancias y conceptos disjuntos. Está limitado a la definición de 50 conceptos, 50 relaciones y 50 instancias.

OntoEdit Incluye a OntoEdit Free más un *plug-in* de léxico de dominio. Además no tiene limitación en el número de conceptos y relaciones.

OntoEdit Professional Se basa en OntoEdit y permite al usuario además construir reglas, consultas e inferencia, importar desde y exportar a un esquema de base de datos.

2.5.7. DAML Viewer

DAML Viewer[66] es una herramienta implementada en Java que proporciona un modo de visualizar los recursos encontrados en un documento DAML+OIL. Se puede emplear en dos modalidades, a través de un Applet[67] en su página web o como aplicación tras descargar sus clases[68].

La herramienta inicialmente muestra una ventana donde se pide la URI del documento a analizar, ya sea introduciéndola por teclado o mediante la exploración de

[65] lenguaje de ontologías propio de la herramienta
[66] http://daml.org/viewer
[67] para ello necesita el *plug-in* 2 de Java
[68] necesita versión JDK 1.3 o superior, así como el RDF-API 2000-11-13 o superior (ver subsección 2.5.3.1)

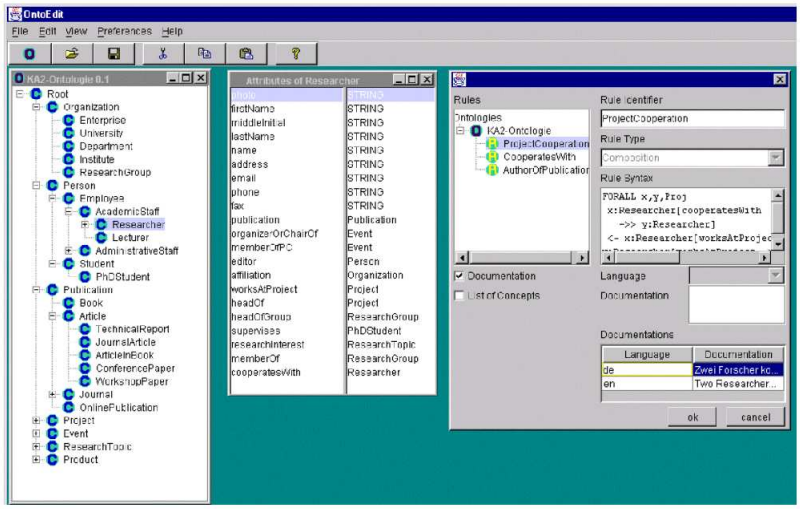

Figura 2.15.: Entorno de edición de ontologías OntoEdit.

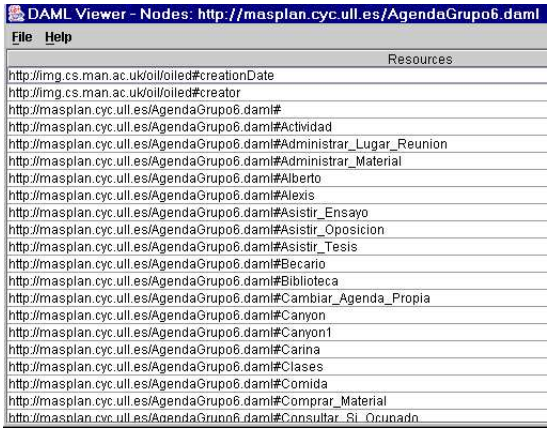

Figura 2.16.: DAML Viewer: lista de recursos.

Figura 2.17.: DAML Viewer: declaraciones que involucran a un recurso seleccionado.

los discos locales. Una vez cargado el documento, se muestra una ventana (ver Figura 2.16) con todas las URIs de los recursos referenciados en el documento. Esta ventana permite entre otras acciones, a través de las opciones de menú, mostrar el número de recursos y declaraciones encontradas en el código del documento.

Al pulsar sobre uno de los recursos, aparece una tercera ventana (ver Figura 2.17), que muestra las declaraciones en las que se ve envuelto el recurso seleccionado.

- Las declaraciones marcadas con −−> indican que la URI en cuestión es el sujeto de la declaración y que la URI que aparece en la columna Valor es el objeto.
- Las declaraciones marcadas con <−− indican que la URI en cuestión es el objeto de la declaración y que la URI que aparece en la columna Valor es el sujeto.

Los recursos y propiedades que aparecen en esta ventana pueden ser analizados del mismo modo de forma recursiva. Si se pulsa sobre la flecha, se muestran las declaraciones cosificadas de la declaración correspondiente.

2.5.8. VisioDAML

VisioDAML[69] es una aplicación para Microsoft Visio[70] que permite crear representaciones gráficas de ontologías DAML+OIL. En realidad consiste en una plantilla que contiene las formas que representan las construcciones del lenguaje DAML+OIL.

2.5.9. AeroDAML

AeroDAML[71] es una herramienta para la Web Semántica que permite realizar fácilmente anotaciones DAML en páginas web, a semejanza del lenguaje SHOE, y que de otro modo son tediosas y consumen excesivo tiempo. Para ello aplica técnicas de extracción de información en lenguaje natural (*natural language processing*, NLP) para generar de forma automática las anotaciones DAML. Esta herramienta enlaza la mayoría de los nombres propios y relaciones comunes con las clases y propiedades que aparecen en las ontologías DAML.

[69] http://www.daml.org/visiodaml
[70] http://www.microsoft.com/office/visio/default.asp
[71] http://ubot.lockheedmartin.com/ubot/hotdaml/aerodaml.html

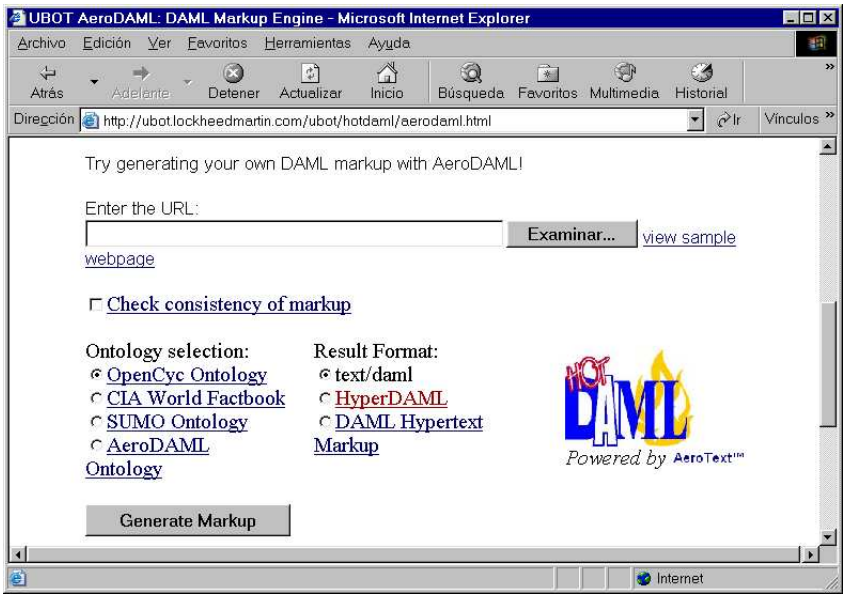

Figura 2.18.: Interfaz Web AeroDAML.

La versión AeroDAML a través de página web (ver Figura 2.18) permite al usuario introducir la URI del documento en el cual se desea realizar anotaciones DAML. Esta versión devuelve la anotación para la página web especificada. Las anotaciones consisten en palabras (*entidades*) enlazadas con ontologías como instancias de clases y relaciones que son enlazadas como instancias de propiedades. Una lista de ejemplos típicos es la siguiente [KH01]:

- Nombres propios. *Austria* instanceOf *nacion.*
- Nombres comunes. *pistola* instanceOf *arma.*
- Co-referencias. *Clinton* equivalent *Bill Clinton* instanceOf *persona.*
- Medidas. *22 metros* instanceOf *medida.*
- Dinero. *200$* instanceOf *dinero.*
- Fecha absoluta. *6 de Marzo de 1998* instanceOf *fecha absoluta.*
- Fecha relativa. *ayer* instanceOf *fecha relativa.*
- Co-referencias. La fecha actual es *2 de Agosto de 2003*, por lo tanto *ayer* fue *1 de Agosto de 2003.*
- Organización a lugar. *Universidad de La Laguna* a *España.*
- Persona a organización. *Henri Dunant* a *Cruz Roja Internacional*.

2.5.10. TRIPLE

RDF ha dado lugar a un lenguaje en capas de consulta, inferencia y transformación para la Web Semántica, llamado TRIPLE[72]. En vez de tener una semántica construida para RDFS, tal como lo tienen otros lenguajes de consulta, TRIPLE permite definir con reglas la semántica de lenguajes construidos sobre RDF. Para lenguajes donde esto no es posible, como el caso de DAML+OIL, se proporciona acceso para programas externos. Como resultado, TRIPLE permite el razonamiento y transformación RDF bajo varias semánticas diferentes. Como por ejemplo, esta es una codificación basada en el Dublin Core Metadata.

```
rdf := "http://w3.org/...rdf-syntax-ns#".
dc := "http://purl.org/dc/elements/1.0/".
dfki := "http://www.dfki.de/".

@dfki:documents {

  dfki:d_01_01[
    dc:title → "TRIPLE";
    dc:creator → "Michael Sintek";
    dc:creator → "Stefan Decker";
    dc:subject → RDF;
    dc:subject →triples;...].

  ∀ S,D search(S,D) ←
    D[dc:subject →S].
}
```

La herramienta para TRIPLE, implementada en Java, se distribuye bajo la licencia *Semantic Web Foundation for Open Source Software* (SFO), derivada de la licencia *Apache Software* [SD02, SD01].

2.5.11. DAML+OIL Ontology Checker

DAML+OIL Ontology Checker[73] es una interfaz web (Figura 2.19) para comprobar la consistencia de las ontologías DAML+OIL. Para ello emplea Jena. El usuario introduce en un formulario la URI de la ontología DAML+OIL a analizar, devolviendo el correspondiente informe HTML.

2.5.12. DAML Validator

DAML Validator[74] es una herramienta para comprobar el marcado DAML+OIL más allá de simples errores de sintaxis. El validador lee el documento DAML a analizar

[72] http://triple.semanticweb.org/
[73] http://potato.cs.man.ac.uk/oil/checker
[74] http://daml.org/validator

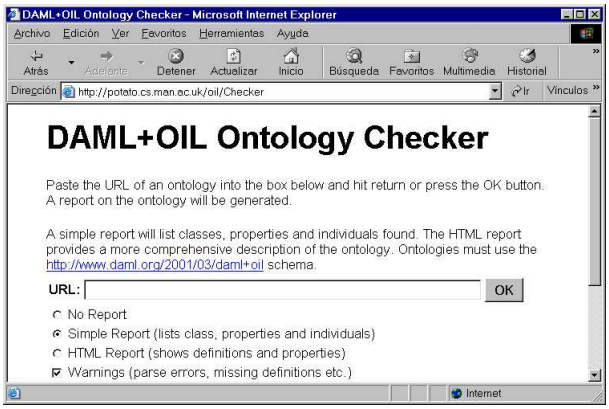

Figura 2.19.: Interfaz Web DAML+OIL Ontology Checker.

warning	In Unknown URI	
	line	**statement**
		http://masplan.cyc.ull.es/AgendaGrupo6.daml#Evelio http://masplan.cyc.ull.es/AgendaGrupo6.daml#Su_Email_Es http://masplan.cyc.ull.es/AgendaGrupo6.daml#evelio
Use of this property implies that object is of type http://masplan.cyc.ull.es/AgendaGrupo6.daml#DireccionEmail. (Object is declared type [http://www.daml.org/2001/03/daml+oil#Thing])		

Figura 2.20.: Ejemplo de salida DAML Validator (Fragmento).

y lo examina buscando una gran variedad de errores potenciales. Tras este análisis, proporciona al usuario una lista de errores, un puntero al error en el fichero y una indicación sobre la naturaleza del problema.

Se encuentra disponible a través de una interfaz en su página web, donde el usuario introduce la URI del documento a analizar, y a través de una distribución gratuita descargable en disco duro. Esta distribución se encuentra disponible en un validador basado en Jena (concretamente en su procesador ARP, subsección 2.5.3) o en RDF API (subsección 2.5.3.1).

Tras el proceso de validación se genera como salida una lista de indicaciones (un fragmento de un ejemplo se muestra en la Figura 2.20). Cada indicación tiene un nivel (error, advertencia o indicación), tipo (por ejemplo *recurso indefinido*), mensaje (un texto que describe el problema), identificador, número y localización. Cuando es posible, se indica un número de línea del documento donde se ha detectado un posible error. Debido a las limitaciones en el procesador, estos números se refieren a la línea donde se encuentra la etiqueta final del elemento que contiene la declaración correspondiente.

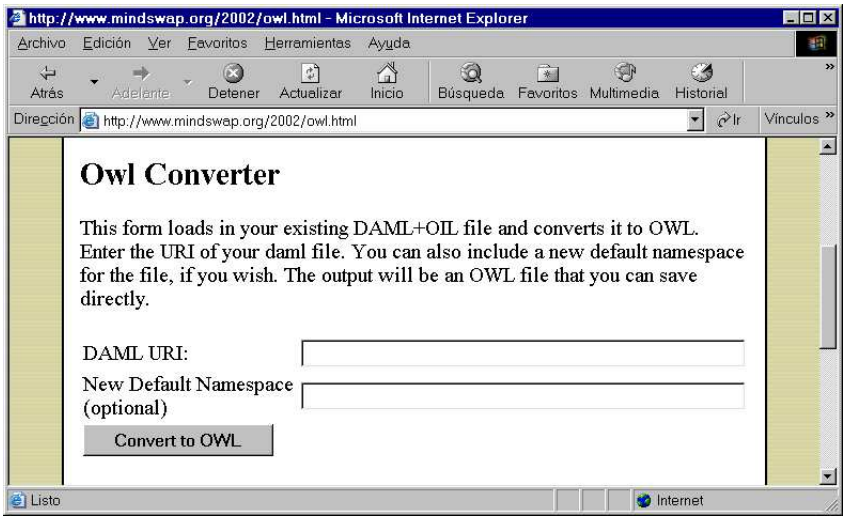

Figura 2.21.: Interfaz Web OWL Converter.

2.5.13. OWL Converter

OWL Converter[75] consiste en un formulario web (ver Figura 2.21) donde el usuario puede introducir la URI de un documento DAML+OIL existente. La salida del programa es la conversión de este documento en OWL mediante un fichero que el usuario puede grabar directamente. El usuario también puede especificar un nuevo espacio de nombres por defecto.

[75] http://www.mindswap.org/2002/owl.html

3. Agentes y Sistemas Multiagente

Este capítulo está dedicado a la presentación de los aspectos relativos al concepto de agente y sistema multiagente. En primer lugar, se presenta una noción de lo que significa un agente. Como se verá, no existe una definición universalmente aceptada, por lo que esta noción se basará en diversas definiciones y características que debiera tener un agente, propuestas por desarrolladores de sistemas multiagente. Estas definiciones y características permiten a continuación una clasificación de agentes según sus características dominantes: móviles, colaborativos... Una vez tratados los temas referentes a los agentes, pasaremos a los relativos a los sistemas multiagente. En primer lugar, justificaremos su utilización para a posteriori centrarnos en dos aspectos fundamentales de estos sistemas: la comunicación y la arquitectura. Debido a su importancia y su repercusión en el trabajo presentado en este proyecto de tesis dedicaremos una sección al estándar FIPA, prestando especial atención a las especificaciones sobre el asistente personal y el servicio de ontologías. Finalizaremos este capítulo analizando las diferentes herramientas existentes para la implementación de sistemas multiagente.

3.1. El concepto de agente

El concepto de *agente* ha cobrado una enorme importancia en nuestros días dentro de numerosos campos. Así se emplean en problemas muy diversos como el manejo y control de la información en redes de computadoras [SFJ97], la planificación de la agenda personal, la compra de billetes de avión o el transporte militar [MHVB00].

Su origen se sitúa en el año 1977, en el que Carl Hewitt propone la idea de un objeto auto-contenido, interactivo y ejecutado de modo concurrente llamado *actor*. Este objeto tenía un estado interno con cierto grado de encapsulado y podía responder a mensajes de otros actores [Nwa95, GB99].

> es un agente computacional que tiene una dirección de correo y un comportamiento. Los actores se comunican mediante el paso de mensajes y llevan a cabo sus acciones de forma concurrente [Hew77].

Sin embargo, a pesar de lo extendido de su aplicación, su definición es todavía objeto de discusión y análisis. De hecho no existe una definición académica aceptada por todo el mundo. Nwana [Nwa95] sitúa las razones para esto en al menos dos factores:

- Los investigadores que trabajan en el campo de los agentes *no poseen en exclusiva* el término, tal como ocurre en otras áreas como puede ser la *lógica fuzzy*. El término “agente” se emplea con frecuencia en el lenguaje corriente en expresiones como *agente* de bolsa o *agente* inmobiliario.

- La palabra "agente" se ha convertido en un paraguas donde se cobijan investigaciones muy heterogéneas, y que se sienten cómodas bajo ella. De este modo los desarrolladores no se ven ligados a una definición restrictiva y pueden calificar de *agente* a un amplio espectro de entidades.

Esto explica la gran cantidad de autores que han aportado su propia definición. La mayoría de estas definiciones, tal como se verá en las subsecciones siguientes, suelen incidir en aspectos como la autonomía del sistema o su capacidad de reacción ante cambios producidos en su entorno.

3.1.1. Definiciones de agente

En esta subsección reflejaremos las definiciones más significativas que varios autores y organizaciones han aportado relativas al término agente. No es objetivo de este trabajo decantarse por una en detrimento de las otras. Por el contrario, consideramos que el conjunto de todas ellas proporcionará al lector un cuadro bastante ajustado sobre qué se debe entender por agente.

- Los agentes autónomos son sistemas computacionales que habitan algún entorno dinámico complejo, perciben y actúan autónomamente en ese entorno y, haciendo esto, llevan a cabo un conjunto de metas o tareas para los que han sido diseñados (P. Maes) [Mae94, Mae95, Mae97].

- Los agentes inteligentes son entidades software que ayudan a la gente y actúan en su nombre:
  - Todos los agentes son *autónomos*. Es decir, tienen control sobre sus propias acciones.
  - Todos los agentes son *orientados a metas*. Los agentes tienen un propósito y actúan de acuerdo con él.
  - Un agente podría también ser *dirigido mediante reglas*, que es un modo más general de definir las metas de los agentes.
  - Todos los agentes *reaccionan* ante los cambios detectados en su entorno.
  - Algunos agentes son *sociables*. Esto es, interactúan o se comunican con otros agentes.
  - Algunos agentes *se adaptan*. Aprenden o cambian su comportamiento basándose en experiencias previas.
  - Algunos agentes son *móviles*. Se mueven de máquina en máquina.
  - Algunos agentes *se esfuerzan en ser creíbles*. Aparecen ante el usuario como una entidad visible o audible e incluso pueden tener aspectos emotivos o de personalidad (D. Gilbert) [Gil97].

- Un agente autónomo es un sistema situado en y parte de un entorno, que siente ese entorno y actúa sobre él, a través del tiempo, buscando cumplir su propia agenda y, de este modo, afectando a lo que sentirá en el futuro (Franklin, Graesser) [FG96].

- Un agente es cualquier entidad que percibe su entorno a través de sensores y actúa en ese entorno a través de actuadores (Russel, Norvig) [RN95].

- Un agente es un software persistente dedicado a un propósito específico (Smith, Cypher, Spohrer) [SCS94].
- Los agentes inteligentes llevan a cabo continuamente tres funciones: percibir las condiciones dinámicas en el entorno, actuar para afectar a las condiciones del entorno y razonar para interpretar lo que percibe, resolver problemas, inferir y determinar acciones (Hayes-Roth) [HR95].
- Un agente es un programa de ordenador que actúa autónomamente en nombre de una persona u organización (OMG) [OMG98].
- Los agentes software son programas que se comunican a través de diálogos, son autónomos e inteligentes, deben ser robustos, no son invariantes en el tiempo y son distribuidos en redes (Coen) [Coe95].
- Los agentes autónomos son sistemas capaces de actuar autónomamente y con determinación en el mundo real (Brustolini) [Bru91].
- El término agente se emplea para denotar un sistema hardware y/o software que disfruta de las siguientes propiedades: es autónomo, sociable, reacciona y toma la iniciativa para provocar que las cosas ocurran (Wooldridge) [WJ94].
- Un agente es una entidad que reside en entornos donde interpreta datos que reflejan eventos y ejecuta comandos que producen efectos en ese entorno (FIPA) [FIP96].
- Un agente software es un programa que se comunica con otros a través de un lenguaje de comunicación de agentes (Genesereth) [GK97].

3.1.2. Características de un agente

Las definiciones mostradas en la subsección anterior forman un conjunto heterogéneo. No obstante, de ellas se deducen una serie de características básicas que debe o puede tener un agente. En esta subsección analizaremos estas características con un poco más de detalle [Mae95, Nwa95, WJ94, OMG98, KSCK94, Fon01, Igl97, Edw97].

Autonomía. Es la capacidad del agente de existir independientemente de un usuario o de otra entidad, teniendo el control de sus propias acciones.

Reacción ante cambios de su entorno. Los agentes perciben los cambios producidos en su entorno y responde a esos cambios mientras se adapta a ellos para conseguir su objetivo.

Sociabilidad. Los agentes suelen mantener comunicaciones complejas, ya sean con otros agentes o con humanos, empleando para ello un *lenguaje de comunicación entre agentes* (*Agent Communication Language, ACL*).

Pro-actividad. Un agente no se conforma con responder a los cambios de su entorno o a las acciones ejecutadas directamente sobre él, sino que toma la iniciativa por sí mismo para alcanzar su objetivo.

Movilidad. Es la habilidad del agente para transportarse a sí mismo de una máquina a otra, manteniendo su estado actual.

Continuidad temporal. Los agentes deben ser vistos más como procesos que corren continuamente que como meras funciones que responden con una salida fija ante un mismo valor de la entrada. Mientras dure su ciclo de vida, los agentes continúan ejecutando su código siempre que se produzca el evento adecuado.

Credibilidad. Ningún agente debe proporcionar información falsa de un modo intencionado.

Benevolencia. Los agentes están dispuestos a colaborar mientras esta cooperación no entre en conflicto con sus objetivos a cumplir.

Multi-plataforma. Los agentes deben ser capaces de comunicarse entre diferentes arquitecturas y sistemas de computadores.

Robustez. Los agentes deben ser diseñados para tratar con los cambios inesperados de su entorno. Deben incluir mecanismos de recuperación ante errores del sistema o humanos.

Responsabilidad. Los agentes deben manejar la información privada de una forma responsable y segura.

Inteligencia. Capacidad de aprender y razonar.

Capacidad de representación. Un agente puede actuar en nombre de alguien o algo, esto es, actuar en interés de, en representación de, o en beneficio de alguna entidad.

Cooperación. Capaz de coordinarse con otros agentes para alcanzar un objetivo común.

3.1.3. Actitudes mentales de los agentes: el modelo BDI

La Inteligencia Artificial ha estudiado lo que denomina "actitudes mentales" en los agentes inteligentes, y que definen su capacidad de razonamiento. Estas actitudes están relacionadas con la observación y detección de capacidades ligadas con estados mentales humanos (conocimiento, creencias, deseos y metas). Estos estados pueden representarse por medio de actitudes, que pueden ser informativas, de motivación o sociales.

- Las actitudes informativas consideran lo que se debe hacer con la información a la cual tienen acceso los agentes, incluyendo la relativa a su entorno. Entre estas actitudes destacan el *conocimiento* (por definición verdadero) y las *creencias.*
- Las actitudes de motivación son las directamente relacionadas con las posibles acciones que un agente pueda llegar a tomar. Como ejemplo tenemos las actitudes de *elecciones, planes*, *metas, deseos, preferencias...*
- Por último, las actitudes sociales son externas y están relacionadas con los permisos y obligaciones, ya sean de carácter político, moral o racional.

El modelo BDI [RG95] limita estas actitudes mentales de los agentes a tres: creencias (*believes),* deseos (*desires*) e intenciones (*intentions*)[1].

Creencias: Conjunto de proposiciones que el agente acepta como verdaderas. Dicho de otra forma, lo que el agente conoce de su entorno (su modelo del mundo y del resto de los agentes). Obsérvese que aunque el agente lo tome como verdadera, esa creencia puede resultar falsa, a diferencia del conocimiento, que por definición es siempre verdadero.

Deseos: También llamados frecuentemente *metas* u *objetivos*. Denotan una propiedad o conjunto de propiedades del entorno que el agente quiere que sean verdaderas, pero que no se encuentran actualmente entre sus creencias. Es decir, representa algún estado final deseado.

Intenciones: Conjunto de acciones planificadas por el agente que le permiten llegar a un estado deseado. Estos planes pueden ser interrumpidos en un momento dado y son realimentados de los posibles cambios que se podrían haber producido en el entorno.

En este modelo, las entradas al sistema son *eventos* recibidos mediante una *cola de eventos.* El sistema puede reconocer tanto eventos internos como externos. Se asume que los eventos son indivisibles y se reconocen cuando se completan, no cuando ocurren. Por otro lado, la salida del sistema son *acciones* (también indivisibles) realizadas por una *función de ejecución*. Basándose en el estado actual del sistema y de los eventos presentes en la cola, se seleccionan las opciones a ejecutar.

La implementación de este modelo lleva los siguientes pasos, repetidos de forma continua:

1. Al inicio de cada ciclo, un generador de opciones lee la cola de eventos y genera una lista de opciones.
2. Un liberador de opciones selecciona un subconjunto de opciones y las agrega a la estructura de intenciones.
3. Si existe una intención que realiza una acción en este punto, el agente la ejecuta.
4. Si algún evento externo ha ocurrido durante el ciclo, es agregado a la cola de eventos.
5. Los eventos internos son agregados.
6. El agente modifica la estructura de deseos e intenciones eliminando aquéllos alcanzados con éxito así como aquellos deseos imposibles y acciones no realizables.

Algunas de las limitaciones de este modelo son:

- Algunas conductas no han sido bien establecidas. En particular, el modelo parece inapropiado para construir sistemas que deben aprender y adaptar su conducta.

[1] El acrónimo BDI proviene del nombre anglosajón de las tres actitudes mentales que contempla.

- No refleja consideraciones de arquitectura para aspectos explícitos de la conducta de sistemas multiagente.
- El modelo BDI no deja de ser un modelo de especulación filosófica, no sujeto a demostración.

3.1.3.1. Noción débil y noción fuerte de agente

Ya hemos comentado anteriormente que no existe una definición universalmente aceptada del concepto de agente. Sin embargo, y a partir de lo visto en los anteriores apartados, existen dos *nociones* de lo qué define un agente [Alo02].

- La *noción débil* afirma que para que un sistema sea considerado un agente basta que cumpla con las características de autonomía, sociabilidad, reacción ante cambios en su entorno y pro-actividad.
- Mientras, la *noción fuerte* exige además actitudes mentales (BDI), movilidad, capacidad de razonar, credibilidad y benevolencia.

3.2. Taxonomía de los agentes

Las numerosas definiciones y características relatadas en la sección 3.1 sobre el concepto de agente provocan que la labor de clasificación de los diversos tipos de agentes no resulte fácil.

Una primera clasificación puede basarse en la característica que se desea resaltar. Así hablamos de agentes *autónomos*, *reactivos*, *proactivos*, *inteligente*s... Sin embargo, a nuestro juicio, esta clasificación adolece de cierta vaguedad, decantándonos por la clasificación dada por Nwana en [Nwa95]. Esta clasificación se basa en varias dimensiones.

- Respecto a su movilidad los agentes se dividen en *móviles* o *estáticos,* según su capacidad o no de desplazarse en una red. Los agentes estáticos son aquéllos que solamente pueden ejecutarse en la máquina donde fueron iniciados. En contraste, un agente móvil es aquél que no se ve limitado al sistema donde comenzó su ejecución, sino que es capaz de transportarse de una máquina a otra a través de la red. Los agentes móviles se emplean, por ejemplo, para situaciones en las que no se puede tener una alta velocidad de comunicación a través de una red y el agente necesita acceder a datos que se encuentran en otro punto de esa red. En ese caso es rentable que el agente se desplace a ese punto y una vez tenga los datos (o en su caso, ejecutado el proceso), vuelva al sistema original. De este modo, el tránsito necesario a través de la red disminuye.
- En segundo lugar, un agente puede clasificarse como *deliberativo* o *reactivo.* Los primeros derivan del paradigma de pensamiento deliberativo: los agentes poseen un modelo simbólico interno de razonamiento y se ocupan de la planificación y la negociación con el objetivo de conseguir coordinarse con otros agentes. Los agentes reactivos, por el contrario, no poseen ningún modelo simbólico interno de su entorno y actúan empleando un comportamiento del tipo estímulo-respuesta, respondiendo al estado actual del entorno en que están embebidos.

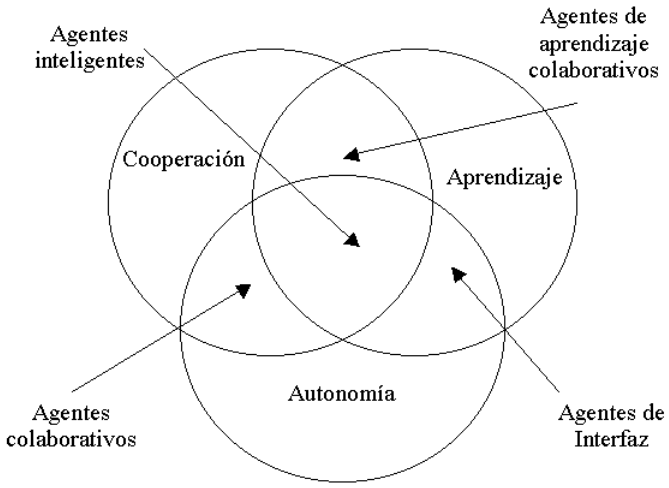

Figura 3.1.: Clasificación de los agentes según su característica primaria [Nwa95].

- En tercer lugar, los agentes pueden clasificarse según los atributos primarios que deberían mostrar. Nwana distingue tres principales: autonomía, aprendizaje y cooperación. De este modo existen los agentes *autónomos*, *de aprendizaje* y *cooperativos.* Las combinaciones dos a dos de estas características originan los agentes *colaborativos* (cooperación y autonomía), *colaborativos de aprendizaje* (aprendizaje y cooperación) y *de interfaz* (aprendizaje y autonomía). Finalmente, la combinación de los tres elementos da como resultado los agentes *inteligentes.* Esta distribución se muestra en la Figura 3.1.
- En algunas ocasiones, los agentes pueden ser clasificados por el papel que desempeñan. Por ejemplo, los agentes *de información* o los agentes *de Internet*. Estos tipos de agentes se dedican a la búsqueda y procesamiento de información en una red, como en el caso de Internet.
- Por último, Nwana habla de agentes *híbridos* para referirse a los que combinan dos o más de las categorías anteriores.

Debemos advertir que esta clasificación debe ser considerada como meramente ilustrativa. Los desarrolladores de agentes pueden ajustarse a uno de estos conceptos, o crear un a nueva categoría según sus necesidades.

3.3. Sistemas Multiagente

Los sistemas multiagente (*MultiAgent Systems*, MAS) constituyen el siguiente paso lógico en la tecnología de agentes. Consideremos el caso de un agente de información que es activado en una red de computadores. Este único agente se desplazará a lo largo de la red tratando de conseguir la información que le interesa a su usuario. En este caso, existe muy poca interacción con otros posibles agentes del entorno.

Sin embargo, son muy pocos los campos en que un único agente basta para llevar a cabo una tarea compleja. Lo normal es crear una *sociedad* de agentes que se comuniquen entre ellos, colaboren y se coordinen en la realización de la tarea. Estas sociedades son los sistemas multiagente. Por tanto se deduce que los agentes que van a aparecer en estos sistemas son los colaborativos.

Las principales razones que justifican los sistemas multiagente son [OMG98, Alo02, RL94, Hon99]:

- Se podría diseñar un único agente que implementase toda la tarea. No obstante, este tipo de *macroagente* representa un cuello de botella para la velocidad, confianza, robustez y mantenimiento del sistema. Dividir la funcionalidad entre varios agentes proporciona modularidad, flexibilidad, facilidad para ser modificado y extensibilidad. Por tanto, son ideales para la idea del *divide y vencerás*. Es decir, el problema se subdivide y se diseña un agente o conjunto de agentes especializados en la resolución de cada uno de los subproblemas, y así recursivamente.

- La totalidad del conocimiento no se encuentra generalmente disponible para un único agente. El conocimiento que está distribuido entre varias fuentes (agentes) puede ser recombinado en una visión más general cuando sea necesario.

- Las aplicaciones que necesitan computación distribuida son tratadas de una mejor forma por los MAS. En esta situación, los agentes pueden ser diseñados como componentes autónomos especializados que actúan en paralelo. Por tanto, los agentes constituyen el nivel más avanzado en cuanto a tecnología de componentes distribuidos. Además estos sistemas pueden estar *abiertos*, es decir, los diversos agentes pueden añadirse o destruirse en el sistema de una forma dinámica.

- Los sistemas multiagente son una plataforma atractiva para la convergencia de varias tecnologías de la Inteligencia Artificial. Ejemplo de esta convergencia es la *Robocup*[2].

- Se evita que un sistema quede bloqueado por un único punto de fallo.

Estas sociedades son más complicadas de diseñar que un sistema de un único agente, puesto que aparte del código necesario para el tratamiento de la tarea-problema, el diseñador del sistema tiene que hacer frente a aspectos relacionados con la comunicación y la negociación entre los agentes, así como de la organización de los agentes dentro del sistema. Sin embargo, a pesar de esta complejidad inicial, la atención de los desarrolladores se centran precisamente en el poder de los MAS, como lo demuestra la gran cantidad de herramientas para la implementación de MAS (ver sección 3.8) y el esfuerzo de organizaciones como OMG[3] y FIPA[4] para crear estándares para la interoperabilidad de agentes.

Estos estándares persiguen que sistemas heterogéneos, implementados con diferentes herramientas, diferentes filosofías, con diferentes fines e incluso por

[2]http://www.robocup.org
[3]Object Management Group. http://www.omg.org/
[4]Foundation for Intelligent Physical Agents. http://www.fipa.org

diferentes desarrolladores, sean capaces de interactuar, comunicándose en una estructura global compatible. Para ello, y en primer lugar, se debe establecer tanto un lenguaje común (que a partir de ahora llamaremos *lenguaje de comunicación de agentes*, *ACL*) como una organización compatible de los agentes dentro de los MAS que sean ampliamente aceptados.

En este sentido, en las secciones siguientes, analizaremos en primer lugar los ACL que cuentan con una mayor difusión (KQML y FIPA-ACL), para a continuación profundizar en las arquitecturas más extendidas de los sistemas multiagente (Facilitators, estándar OMG y estándar FIPA).

3.4. Lenguajes de comunicación de agentes

Debido a las particularidades de los sistemas multiagente, los aspectos de la comunicación entre agentes cobran un papel de enorme importancia, ya que es casi imposible que exista cooperación sin comunicación. Para ser realmente efectiva y provechosa, esta comunicación debe estribar en algo más que una mera invocación recíproca de métodos. En este sentido, lo que se ha demostrado como más provechoso es la comunicación a través de *mensajes* ya que aparte de una transmisión de datos, tiene asociado un contenido semántico accesible por ambas partes y sobre el que deben estar de acuerdo. Esto permite pasar de un modelo de objetos a un modelo de agentes inteligentes basados en actos de comunicación.

Los dos lenguajes de comunicación de agentes más extendidos en el desarrollo de agentes son el KQML y el FIPA-ACL, ambos basados en la teoría del habla (*Speech Act Theory).* Describiremos brevemente esta teoría antes de pasar al análisis de ambos lenguajes.

3.4.1. La teoría del habla (*Speech Act Theory)*

La teoría del habla *(Speech Act Theory)* es un marco teórico de alto nivel desarrollado por filósofos y lingüistas para explicar la comunicación humana. Esta teoría ha sido ampliamente usada, formalizada y extendida con campos de la Lingüística Computacional y la Inteligencia Artificial como un modelo general entre agentes arbitrarios [LFP99]. Se refiere sobre todo al papel del lenguaje como acto comunicativo (*speech act*), de modo que los interlocutores no se limitan solamente a enunciar oraciones que sean verdaderas o falsas. De hecho, un acto comunicativo se compone a su vez de tres actos:

locución El acto físico de la emisión, pronunciación o expresión del acto comunicativo.

ilocución El acto de transmisión de las intenciones del hablante hacia el oyente que se realiza por medio de la emisión.

perlocución Acciones que ocurren como resultado de la ilocución.

Por ejemplo, la locución “Cierra la puerta” que realiza un individuo A hacia otro B, lleva consigo la ilocución de un mandato hacia B de cerrar la puerta. Si todo va bien, se producirá la perlocución de que B cierre la puerta.

La ilocución suele dividirse a su vez en dos partes: *la fuerza ilocutiva* (*illucutionary force) y* una *proposición.* La fuerza ilocutiva es el parámetro que permite dividir los actos de comunicación en *afirmativos*, *directivos*, *declarativos, expresivos* y *de compromiso* (o *comisivos*).

Lo que nos interesa de esta teoría es que constituye la base de los lenguajes de comunicación de agentes. Actualmente existen unos 4600 actos de comunicación catalogados [Vas98]. Como es lógico pensar, y tal como se verá en esta sección, un lenguaje de comunicación de agentes no logra cubrirlos todos.

3.4.2. KQML

El KQML (*Knowledge Query Manipulation Laguange*) ha sido un lenguaje de comunicación entre agentes tomado como estándar de facto durante mucho tiempo. Sus dos especificaciones (una primera versión de 1993 [FWW+93, FFMM93] fue revisada por Labrou y Finin en 1997 [LF97]) persiguen crear un lenguaje que permite a los agentes software autónomos y asíncronos compartir su conocimiento y trabajar de modo cooperativo para la resolución de problemas. Fue desarrollado como parte del *ARPA Knowledge Sharing Effort (KSE)*, responsable a su vez de otros lenguajes como el KIF[5] y Ontolingua[6]. Actualmente su uso se encuentra extendido, existiendo numerosos paquetes y herramientas que implementan la comunicación a través de este lenguaje (ver sección 3.8)[7].

El KQML se trata de un lenguaje de alto nivel, orientado a mensajes. Asimismo se define como un protocolo para el intercambio de información *independiente* de la sintaxis, ontología y lenguaje *del contenido* del mensaje (este aspecto es importante, tal como se verá en secciones posteriores). También es independiente del mecanismo de transporte del mensaje (RMI, email...) y de los protocolos de alto nivel (subastas, reclutamiento de agentes...) involucrados. Cada uno de los agentes posee y gestiona una base de conocimiento virtual (*Virtual Knowledge Base, VKB*), permitiéndose que un agente pueda manipular y realizar búsquedas en la VKB de los otros agentes del sistema.

Se basa en una lista de paréntesis balanceados [FFMM94], cuyo elemento inicial es el acto de comunicación implicado (llamado *performativa*[8]). O sea, dota de fuerza ilocutiva al mensaje.

3.4.2.1. Performativas KQML

El conjunto de performativas KQML forman el verdadero núcleo del lenguaje, puesto que determinan el tipo de interacciones permitidas en una conversación basada en KQML (por tanto, en términos de la Teoría del Habla, constituye su fuerza ilocutiva). Su objetivo consiste en dotar de un contenido semántico al contenido del mensaje que ayude al receptor del mensaje a la hora de interpretarlo. Así, por ejemplo,

[5] Knowledge Interchange Format

[6] Lenguaje para la definición de ontologías compartidas

[7] Existen numerosos dialectos y versiones extendidas de KQML. Por ejemplo, la ampliación presentada en [FMT97]

[8] traducción del inglés *performative.* Este término no es aceptado de modo general, hablándose de *actos de comunicación.* En este trabajo, emplearemos ambos términos de forma indistinta.

el contenido de un mensaje (el cual constituye su proposición) no tiene el mismo significado cuando se trata de una afirmación, una consulta o una orden.

La lista de performativas permitidas en KQML, según la especificación de 1997, son:

> `ask-if`, `ask-one`, `ask-all`, `stream-all`, `eos`, `tell`, `untell`, `deny`, `insert`, `uninsert`, `delete-one`, `delete-all`, `undelete`, `achieve`, `unachieve`, `advertise`, `unadvertise`, `subscribe`, `error`, `sorry`, `standby`, `ready`, `next`, `rest`, `discard`, `register`, `unregister`, `forward`, `broadcast`, `transport-address`, `broker-one`, `broker-all`, `recommend-one`, `recommend-all`, `recruit-one` y `recruit-all`[9].

Dejamos para el apéndice A, una breve descripción del significado de estas performativas. Este conjunto de actos de comunicación es extensible, pudiendo un investigador incluir nuevas performativas, según sus necesidades (por ejemplo la presentada en [FMT97]).

No está claro el motivo por el que ciertos actos de comunicación en KQML tienen asociada una performativa que anula el acto (`tell`/`untell`) y otros no.

Las definiciones de estas performativas adolecen de cierta ambigüedad, puesto que éstas consisten en enunciados tales como:

> `achieve`: El emisor pide al receptor que ejecute una acción sobre su entorno físico

Los mismos Labrou y Finin, conscientes de esta falta de contenido semántico en la definición de la performativa, proponen una descripción complementaria compuesta de los siguientes seis puntos [LF94]:

1. Una descripción en lenguaje natural del significado intuitivo de la performativa.
2. Una expresión en lógica definida en [LF94] que describe el acto. Consiste en una representación formal de la descripción dada según el punto 1.
3. Las condiciones previas que describen los estados de los agentes en orden a mandar la performativa y para que el receptor la acepte y la procese.
4. Las postcondiciones que describen los estados de los agentes después de el envío del mensaje (el emisor) y la recepción del mismo.
5. Condiciones de completitud para el emisor que indican el estado final del agente, posiblemente después de que haya tenido lugar una conversación y tras que se haya cumplido la intención sugerida en la performativa que comenzó la conversación.
6. Cualquier comentario en lenguaje natural que pueda ayudar al entendimiento del significado de la performativa.

[9] La especificación de 1993 incluía las performativas `ask-about`, `stream-about`, `monitor`, `pipe`, `break`, y `generator`

A modo de ejemplo[10],

- ask-if (A,B,X)

  1. A desea saber lo que B cree sobre la veracidad o no del contenido.
  2. want(A,know(A,Y)),
     donde Y puede ser uno de los siguientes
     bel(B,X), bel(B,NOT(X)), NOT(bel(B,X))
  3. Pre(A): want(A, know(A,Y))
     Pre(B): ninguna
  4. Post(A): intend(A,know(A,Y))
     Post(B): know(B,want(A,know(A,Y)))
  5. Completion(A): know(A,Y)
  6. No creer algo no significa necesariamente lo mismo que creer lo contrario, aunque pueda ser el caso de ciertos sistemas

Es importante destacar que este análisis semántico es incompleto ya que en el citado trabajo solamente se llega a definir un subconjunto de todas las performativas: las relativas a `ask-if`, `ask-all`, `stream-all`, `tell`, `deny`, `error`, `sorry` y `eos`.

3.4.2.2. Parámetros de un mensaje KQML

Un mensaje KQML se completa con una serie de parámetros que facilitan el envío e interpretación en los procesos de comunicación entre agentes. Los parámetros KQML definidos en [LF97, CFF+92] son:

- `sender`: el emisor actual de la performativa.
- `receiver`: el receptor actual de la performativa.
- `from`: el origen de la performativa en el campo `:content` cuando se emplea la performativa `forward`.
- `to`: el destino final de la performativa en el campo `:content` cuando se emplea la performativa `forward`.
- `in-reply-to`: la etiqueta esperada en una respuesta a un mensaje previo.
- `reply-with`: la etiqueta esperada en una posible respuesta al mensaje actual.
- `language`: nombre del lenguaje de representación del campo `:content`.
- `ontology`: el nombre de la ontología asumida en el campo `:content`.
- `content`: la información acerca de la cual la performativa expresa una actitud.

[10] El apéndice B explica los términos que aparecen en esta definición.

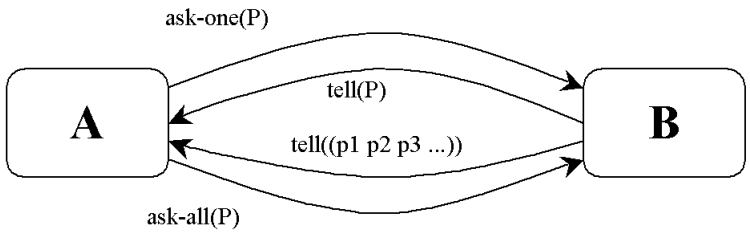

Figura 3.2.: Ejemplo de Conversación KQML [FL99].

Con estas definiciones, presentamos el siguiente ejemplo de mensaje KQML, donde un agente A pide un agente B fije un nuevo valor para el giro de un motor:

```
(achieve  :sender       A
          :receiver     B
          :in-reply-to  id1
          :reply-with   id2
          :language     Prolog
          :ontology     motores
          :content      "giro(motor1,5)")
```

Este ejemplo pone de manifiesto la sintaxis del KQML, inicialmente basada en el lenguaje LISP. Actualmente se han empleado sintaxis alternativas, basadas por ejemplo en SMTP o HTTP.

3.4.2.3. Conversaciones KQML

Se define el concepto de *conversación* como el conjunto de mensajes relativos al mismo hilo. Este concepto es interesante porque permite introducir un cierto control lógico sobre el flujo de mensajes. Por ejemplo, si un agente recibe un mensaje KQML con la performativa `ask-one`, lo lógico es que responda con un mensaje con performativa `tell`. Este protocolo, en concreto, se refleja en la Figura 3.2, junto al de los mensajes con performativa `ask-all`.

3.4.2.4. Capas KQML

Basándose en lo anterior, un mensaje KQML puede representarse dividido en tres capas: de contenido, de mensaje y de comunicación [LFP99]. (Figura 3.3)

- La capa de contenido contiene una expresión en algún lenguaje que codifica el conocimiento que se desea transmitir.
- Como el contenido es *opaco* al KQML, la capa de mensaje añade una serie de características que describen este contenido, como por ejemplo el lenguaje, la

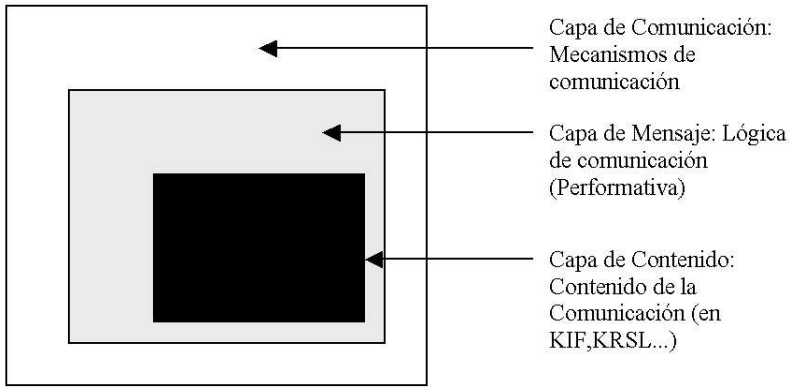

Figura 3.3.: Capas de un mensaje KQML [FFMM94]

ontología...

- Finalmente, la capa de comunicación añade una segunda capa de características del mensaje que describen los parámetros de comunicación a un nivel más bajo, como los relativos al emisor, receptor,...

Una vez más, obsérvese que el KQML *no* impone nada sobre el contenido en sí.

3.4.2.5. Aspectos a mejorar en KQML

Según Finin y Labrou [LFP99], existen tres aspectos en que los que el KQML debe mejorar:

- Inclusión de performativas que abarquen otros aspectos posibles de una comunicación. Por ejemplo, `offer`, `accept`,...
- Empleo de nuevos parámetros, como, por ejemplo, `protocol`, `version`, `reply-by`, `signature`,...
- Convenciones para el "contexto conversacional". Por ejemplo, una forma dependiente del contexto para determinar los valores por defecto de los parámetros de la conversación.

Sin embargo, los mismos autores hacen hincapié en su extensibilidad. No existe ningún impedimento para extender el lenguaje añadiendo nuevas performativas, nuevos parámetros o creando nuevas ontologías.

A estas desventajas debemos incluir la vaguedad en las definiciones de las performativas [CL95].

3.4.3. FIPA-ACL

La FIPA (*Foundation for Intelligent Physical Agents*) es una organización internacional sin ánimo de lucro creada en 1995 y con sede en Ginebra (Suiza). FIPA se dedica a promover la industria de los agentes inteligentes desarrollando especificaciones que soporten la interoperabilidad entre agentes y aplicaciones basadas en ellos. Esto se lleva a cabo mediante la colaboración de sus miembros: compañías[11] y universidades que investigan y desarrollan dentro del campo de los agentes. Los resultados provenientes de estas actividades, son puestas a disposición de todos los desarrolladores mediante una serie de documentos, disponibles a través su página web[12] [FIP01a].

Dentro de las numerosas especificaciones FIPA figura la relativa al lenguaje de comunicación de agentes. Este lenguaje, llamado FIPA-ACL (*FIPA Agent Communication Language*), consiste en una evolución del KQML.

Un mensaje FIPA ACL se compone de un conjunto de uno o más elementos. De un modo preciso, según la situación varía qué elementos se necesitan para una comunicación efectiva entre agentes. El único elemento obligatorio en todos los mensajes FIPA ACL es el de `performative`, aunque sería lógico esperar que la mayoría de los mensajes incluyesen los campos `sender`, `receiver` y `content`.

Podemos, por tanto, distinguir entre el *tipo de mensaje* (determinado por el valor del campo `performative`) y el de los parámetros del mensaje [FIP01a].

3.4.3.1. Tipo de mensaje en FIPA-ACL: el campo *performative*

Con este campo, se define el significado principal del mensaje enviado o, lo que es lo mismo, lo que el agente emisor del mensaje pretende con él. Los posibles valores para este campo son los siguientes:

```
accept-proposal, agree, cancel, cfp, confirm, disconfirm,
failure, inform, inform-if, inform-ref, not-understood,
propagate, propose, proxy, query-if, query-ref, refuse,
reject-proposal, request, request-when, request-whenever y
subscribe
```

Así, por ejemplo, un agente mandaría un mensaje con un valor de la performativa `propose` si lo que desea es enviar a otro agente una propuesta para llevar a cabo una determinada acción dada unas ciertas condiciones previas. En el apéndice C se detallan con un mayor detalle el significado de cada uno de estos valores.

Obsérvese que el conjunto de performativas es similar al de KQML. No obstante se han eliminado algunas como `register` o `unregister` que han sido reemplazadas por sendos actos de comunicación `request` con parámetros especiales en su campo `content`[13]. Este procedimiento es imposible en KQML, puesto que una de las mayores restricciones del KQML es que no interviene en el contenido del mensaje.

[11] Alcatel, British Telecom, Deutsche Telecom, France Telecom, Hitatchi, Hewlett Packard, IBM, Intel, Lucent, NEC, NHK, NTT, Nortel, Siemens, Telia...

[12] http://www.fipa.org

[13] Como se verá en posteriores apartados, esto deriva de la arquitectura de agentes FIPA

Aparte de las diferencias en el conjunto definido de actos de comunicación, un aspecto que distingue la especificación FIPA-ACL de la del lenguaje KQML es su definición formal mediante una lógica modal, mucho más detallada, que elimina cualquier tipo de ambigüedad sobre el verdadero significado del acto de comunicación. A modo de ejemplo, la siguiente definición formal corresponde al acto de comunicación `propose` [FIP02c]. Para una descripción de la lógica, se remite al lector al apéndice B.

$$
\begin{array}{ll}
< i, propose\,(j, < i, act >, \phi) > \equiv & \\
< i, inform\,(j, I_j\, Done\,(< i, act >, \phi) \Rightarrow & I_i\, Done\,(< i, act >, \phi)) > \\
FP: B_i\,\alpha \wedge \neg B_i\,(Bif_j\,\alpha \vee Uif_j\,\alpha) & RE: B_j\,\alpha
\end{array}
$$

donde

$$\alpha = I_j\, Done\,(< i, act >, \phi) \Rightarrow I_i\, Done\,(< i, act >, \phi)$$

Según esta definición formal con un acto de comunicación del tipo `propose`, un agente *i* informa a un agente *j* que, una vez que *j* informe al agente *i* de que *j* ha adoptado la intención de que *i* lleve a cabo la acción *action*, y de que las condiciones previas para que *i* ejecute la acción hayan sido establecidas, *i* adoptará la intención de llevar a cabo la acción *act*.

Los campos FP y RE hacen referencia respectivamente a las *Condiciones Previas de Viabilidad* (*Feasibility Preconditions*) y a los *Efectos Racionales* (*Rational Effects).* Las FPs de un acto comunicativo son las condiciones que deben cumplirse antes de que un agente pueda llevar a cabo una acción. Por otro lado, un RE es el efecto que un agente espera producir con la ejecución de la acción. En este sentido, conviene aclarar que el agente no se ve atado en ningún momento al cumplimiento de los REs asociados al acto de comunicación. Es decir, puede emplear los REs junto a su conocimiento para poder cumplir con sus objetivos, pero no preocuparse de si dichos REs se han producido.

3.4.3.2. Parámetros de un mensaje FIPA-ACL

Los parámetros básicos de un mensaje FIPA-ACL se pueden dividir en las siguientes categorías:

- Relativos a los participantes en la comunicación.
  - `sender`: Indica la identidad del emisor del mensaje. Tiene asociada la dirección de un agente. Es un elemento que aparece en la gran mayoría de los mensajes FIPA-ACL, aunque es posible omitirlo, por ejemplo, cuando el emisor desea permanecer en el anonimato.
  - `receiver`: Indica la identidad del receptor del mensaje. Tiene asociada la dirección de un agente. De modo general, este campo debe aparecer en todos los mensajes. Solamente está permitida su omisión si el receptor del mensaje puede ser deducido del contexto o, en casos especiales, a partir de un mensaje embebido en el contenido del mensaje.

  - `reply-to`: Este elemento indica que los mensajes posteriores del presente flujo de conversación tienen que ser dirigidos al agente que aparece en este campo, en vez del agente que aparece en el campo `sender`.

- Relativos al contenido del mensaje.

  - `content`: Indica el contenido del mensaje. De modo equivalente denota al objeto de la acción. La mayoría de los mensajes FIPA-ACL necesitan de este parámetro, aunque algunos mensajes como los del acto de comunicación `cancel`, tienen un contenido implícito.

- Relativos a la descripción del contenido del mensaje.

  - `language`: Indica el lenguaje en que se expresa el contenido del mensaje. Este campo puede omitirse si se asume que el agente receptor, conoce el lenguaje en que está expresado el contenido. El estándar en la versión de 1999 define un lenguaje de contenidos de carácter general llamado SL así como tres subconjuntos del mismo denotados como SL0, SL1 y SL2.
  - `encoding`: Indica la codificación específica del lenguaje en que está expresado el contenido del mensaje.
  - `ontology`: Indica la(s) ontología(s) empleadas para dar significado a los símbolos en la expresión del contenido del mensaje. Se emplean las ontologías conjuntamente con el elemento de lenguaje para dar soporte a la interpretación del contenido por parte del agente receptor. Suele omitirse si se da por entendido que los agentes conocen previamente el valor de este campo.

- Relativos al control de la conversación.

  - `protocol`: Indican el protocolo de interacción (ver sección 3.4.3.3) que el emisor del mensaje está empleando en el mensaje actual. Este elemento es opcional, aunque se recomienda su uso para evitar problemas de ambigüedad al emplear directamente la semántica ACL para el control de la generación e interpretación de los mensajes.
  - `conversation-id`: Introduce un identificador de conversación que se emplea para identificar la secuencia actual de actos de comunicación que forman dicha conversación. El empleo de este campo permite el seguimiento y análisis de las conversaciones por parte de los agentes.
  - `reply-with`: Introduce una expresión que será empleada por el agente que responda al mensaje actual para hacer referencia al mismo. Así un agente que reciba un mensaje con la expresión

    `reply-with` <expr>

    responderá con

    `in-reply-to` <expr>

- `in-reply-to`: Denota una expresión que referencia una acción anterior del que el presente mensaje es una respuesta.
- `reply-by`: Denota una expresión relativa a un tiempo y/o fecha que indica el límite temporal en el que el emisor del mensaje quisiese haber recibido una respuesta. En otras palabras, introduce un tiempo máximo de espera del mensaje de respuesta.

■ Reservado

- `envelope`: No se emplea, quedando reservado para llevar la información del transporte del mensaje (tiempos, rutas,...)

Con estas definiciones, el siguiente ejemplo de mensaje FIPA-ACL

```
(request
  :sender (agent-identifier :name i)
  :receiver (set (agent-identifier :name j))
  :content
    ((action (agent-identifier :name j)
      (deliver box017 (loc 12 19))))
  :protocol fipa-request
  :language FIPA-SL
  :reply-with order567)
```

serviría para que un agente *i* pidiese a un agente *j* que trasladase una caja identificada como box017 a la localización (12,19), mediante un protocolo fipa-request (ver sección 3.4.3.3), con un contenido en lenguaje FIPA-SL, e indicando que las respuestas a este mensaje deben contener el valor order567 en su parámetro `in-reply-to`.

3.4.3.3. Protocolos de interacción FIPA-ACL

Las conversaciones entre agentes suelen seguir unos ciertos patrones en la secuencia de mensajes intercambiados. A cada una de estas secuencias típicas se les llama *protocolo de conversación* y su explicitud constituye una diferencia sustancial con respecto al KQML[14].

Tal como se ha indicado en la sección 3.4.3.2, el protocolo de interacción aparece en el campo `protocol`. Esto permite identificar el protocolo de comunicación empleado por ambas partes en la conversación actual y de este modo controlar el flujo lógico de mensajes. Así, por ejemplo, se evita que un agente responda con un `cfp` (acto de pedir propuestas para ejecutar una acción) ante un mensaje correspondiente a un acto de comunicación `propose`.

FIPA define actualmente los siguientes 11 protocolos de interacción, definidos con un mayor grado de detalle y de modo más formal que las conversaciones KQML (sección 3.4.2.3):

[14] El KQML no permite el seguimiento inmediato de un protocolo, puesto que no define el parámetro `:protocol`

- FIPA Propose
- FIPA Request
- FIPA Request When
- FIPA Query
- FIPA Contract Net
- FIPA Iterated Contract Net
- FIPA English Auction
- FIPA Dutch Auction
- FIPA Brokering
- FIPA Recruiting
- FIPA Subscribe

Estos protocolos son descritos en detalle en el apéndice E. Sin embargo, a modo de ejemplo, analizaremos en este punto y de manera breve el protocolo *FIPA Request*, resumido de modo gráfico en la Figura 3.4. Este protocolo permite a un agente pedir a otro agente que lleve a cabo una acción [FIP02g].

El *iniciador* (agente que comienza el protocolo de comunicación) ejecuta la petición mediante el acto comunicativo `request`. El *participante* (agente que responde al acto de comunicación del iniciador) procesa el mensaje correspondiente y decide si aceptar o rechazar la petición. La respuesta tiene lugar mediante los actos comunicativos `agree` (opcional) y `refuse` respectivamente.

Una vez que el participante ha aceptado el `request` inicial, puede mandar los siguientes actos de comunicación al iniciador

- `failure` si el intento de llevar a cabo la acción ha fracasado
- `inform-done` si ha conseguido llevar a cabo la acción y solamente quiere informar que se ha completado
- `inform-result` si ha completado la tarea y además quiere comunicar los resultados al iniciador

3.4.4. Comparativa entre KQML y FIPA-ACL

A modo de resumen, podemos establecer las siguientes comparaciones entre los lenguajes de comunicación de agentes analizados en esta sección [Vas98, LFP99, FIP01a, MM98]:

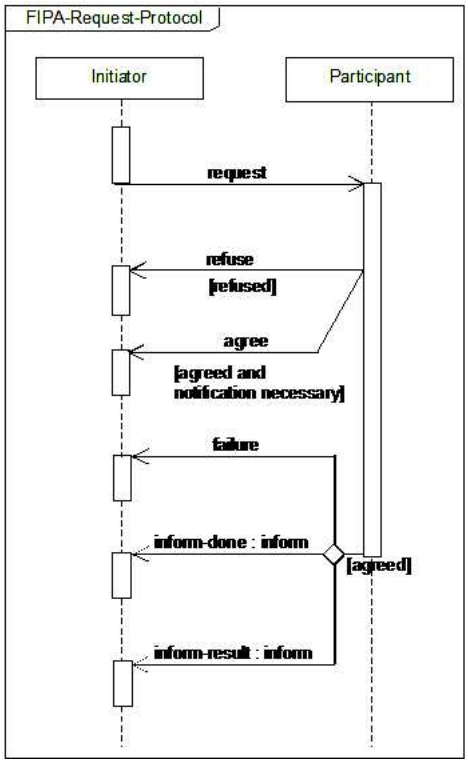

Figura 3.4.: Protocolo de Interacción *FIPA Request* [FIP02g]

- Ambos lenguajes presentan diferencias sintácticas menores. En cambio, disfrutan de una semántica diferente. Esto se debe, entre otros factores, al diferente grado de formalismo en la definición de sus actos de comunicación. Es inútil intentar establecer una correspondencia completa entre las performativas KQML y las de FIPA-ACL, y viceversa. No obstante, es posible establecer ciertas equivalencias, reflejadas en la tabla 3.1.

Cuadro 3.1.: Equivalencias entre algunas performativas KQML y actos de comunicación FIPA-ACL.

Performativa KQML	Acto de comunicación FIPA ACL
ask-if	query-if
tell, untell	inform
deny	inform, disconfirm
insert, uninsert	inform, disconfirm
subscribe	subscribe
error	not-understood
sorry	refuse, failure

- En KQML se permite que un agente pueda inferir directamente en la VKB de otro agente. En FIPA-ACL dicha inferencia se encuentra prohibida.
- Diferencias en el tratamiento de los actos de comunicación relativos a la gestión y administración de los MAS. Así, las performativas `register`, `unregister`,... son sustituidas en FIPA-ACL por actos de comunicación `request` con un contenido reservado.
- FIPA ACL muestra un mayor poder en la composición de nuevas primitivas en cuanto a la descripción del estado mental del agente se refiere (esto deriva del uso de los lenguajes de contenido SL para la descripción de los estados internos de los agentes, como se comprueba en el apéndice D). La debilidad del KQML está en su *religiosa falta de restricción sobre el lenguaje del contenido* [LFP99].

3.5. Arquitecturas de MAS

Aparte de la comunicación, otros de los elementos claves en los sistemas multiagente es su arquitectura, es decir, aspectos relativos, entre otros, a su distribución, organización, gestión y estructura de comunicación. De todos los modelos existentes, elaborados por diversos grupos de investigación, en esta sección detallaremos tres modelos significativos, a nuestro juicio, de arquitectura de agentes. El primero de ellos se refiere a una arquitectura de agentes validada por la experimentación que emplea el lenguaje de comunicación KQML. Sin embargo, a pesar de esta validación, en búsqueda de que dos sistemas multiagente con

diferentes implementaciones puedan interaccionar, se hace necesario un proceso de estandarización. En esta línea, los otros dos modelos analizados corresponden a los estándares FIPA y OMG MASIF, cuyo uso se encuentra extendido de manera amplia entre los desarrolladores de MAS.

3.5.1. Ejemplo de arquitectura de MAS con KQML

La especificación KQML se refiere únicamente a un lenguaje de comunicación de agentes. No obstante, a nuestro juicio, es ilustrativo mostrar un ejemplo de cómo integrar la comunicación a través de mensajes KQML en un sistema multiagente. El ejemplo escogido es el propuesto por Finin, Fritzon, McKay y McEntire en [FFMM94]. Los motivos para escoger este modelo concreto residen en una parte por provenir de autores implicados en la especificación inicial de KQML [FWW+93] y por otra al incluir el concepto de *Facilitator*, ampliamente utilizado en la organización de MAS.

En esta propuesta, conviven dos programas especializados: el *router* y el *facilitator,* y una librería de rutinas de interfaz, llamada *KRIL*. La estructura de esta propuesta se muestra en la Figura 3.5.

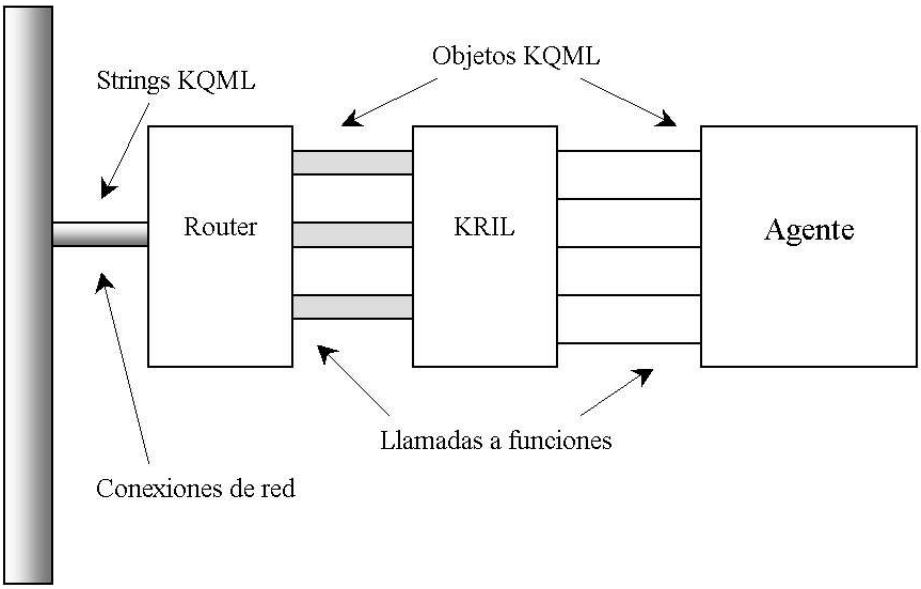

Figura 3.5.: Arquitectura basada en KQML: *Router* y *KRIL* [FFMM94].

3.5.1.1. KQML Router

Cada agente software está asociado a su correspondiente (e independiente) *router*, el cual es independiente del contenido del mensaje. El *router* es el encargado de manejar los mensajes con origen y destino en su agente asociado. Todos los *routers* son idénticos, siendo copias del mismo programa. Como cada programa tiene su

router asociado, no es necesario realizar grandes cambios en la organización interna de cada programa para permitirle de forma asíncrona recibir mensajes provenientes de fuentes diferentes. El *router* proporciona este servicio al agente además de un punto de contacto con el resto de la red. Proporciona, por tanto, funciones de servidor y cliente para la aplicación y gestiona múltiples conexiones simultáneas con otros agentes.

El *router* nunca accede al campo `content` del mensaje, solamente a las performativas KQML y sus argumentos. Si un mensaje saliente KQML especifica una dirección de Internet particular, el *router* direcciona el mensaje a esa dirección. Si el mensaje específica un servicio particular, el *router* intentará encontrar una dirección de Internet para ese servicio y direccionar el mensaje a esa dirección. Si el mensaje solamente proporciona una descripción del contenido, el *router* trataría de encontrar un servidor capaz de responder a ese mensaje, o reenviaría ese mensaje a un agente de comunicación más inteligente que pudiese distribuir ese mensaje a un destino adecuado.

Los *routers* pueden ser implementados con diferentes grados de sofisticación.

3.5.1.2. KQML Facilitator

Para repartir los mensajes que tienen una dirección incompleta, los *routers* recurren a los *facilitators*. Un *facilitator* es una aplicación que proporciona servicios de red útiles para el direccionamiento de estos mensajes en búsqueda de otros servicios. Mantiene un registro del servicio de nombres, reenviando los mensajes de petición de esos servicios.

El *Facilitator* son agentes software que tienen su propio *router* para manejar el tráfico de mensajes dirigidos hacia ellos. Generalmente hay un único *facilitator* por cada grupo de agentes. Cuando se inicia una aplicación, el *router* correspondiente se anuncia al *facilitator* del grupo para registrarse en su base de datos. De modo análogo, cuando la aplicación finaliza, su *router* manda otro mensaje al *facilitator* para darse de baja en esa base de datos.

De esta descripción, se deduce que las performativas KQML de especial interés para un *facilitator* son las relacionadas con la gestión de servicios y reenvío de mensajes, es decir, `advertise`, `broker`, `recruit`...

3.5.1.3. KQML KRIL

Ya que el *router* es un proceso independiente de la aplicación, es necesario que exista una interfaz software entre ambos elementos. Esta API se denomina KRIL (*KQML Router Interface Library*) e intenta facilitar el acceso al *router* por parte del programador de la aplicación. Como el *router* es un proceso separado de la aplicación, y no entiende el campo contenido de un mensaje KQML, esta API se encuentra embebida en la aplicación y tiene acceso a las herramientas de la aplicación para procesar el campo `content` de un mensaje.

A diferencia de los *routers* que consistían en copias del mismo código, puede haber varios KRILs diferentes para cada aplicación y lenguaje.

Para iniciar una transacción al *router* se define una función llamada *send-kqml-message*, que acepta como argumento de entrada el contenido de un mensaje, así como información acerca del mismo y su destinatario. Esta función devuelve la respuesta

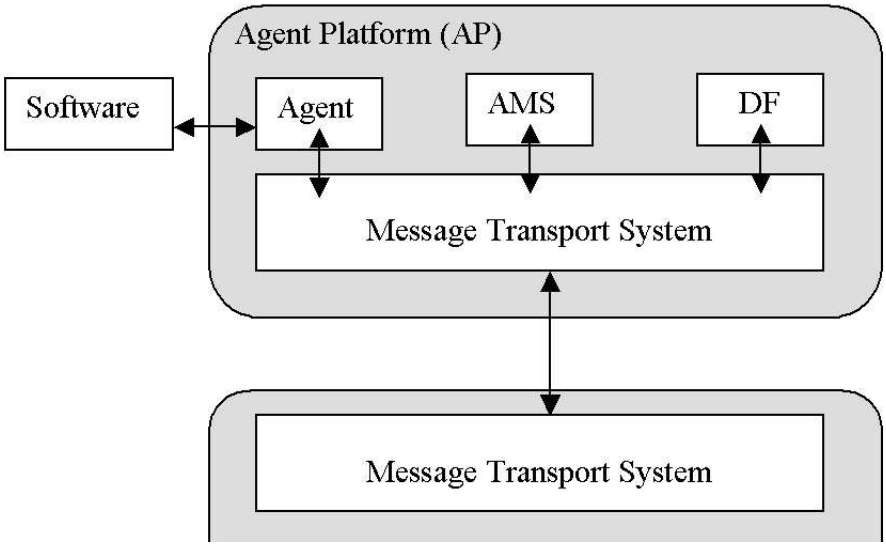

Figura 3.6.: Modelo Referencial FIPA [FIP01b]

al mensaje (si el proceso de comunicación es síncrono y permanece bloqueado hasta que se recibe la respuesta) o una cadena de caracteres indicando que el mensaje fue enviado.

De modo análogo existe generalmente una función llamada *declare-message-handler*, que permite declarar qué funciones deben ser invocadas cuando se recibe un mensaje.

Aparte de estas funciones, los KRILs pueden registrar, a través del *router*, su aplicación correspondiente en el *facilitator* local y contactar con agentes remotos para advertirles que están interesados en recibir datos provenientes de esos agentes remotos.

3.5.2. Arquitectura de una plataforma FIPA: Gestión de Agentes

FIPA establece una arquitectura de gestión una comunidad de agentes, imponiendo en su estándar una serie de elementos que facilitan dicha gestión. En este contexto, define la plataforma de agentes (*Agent-Platform, AP*) describiendo únicamente el comportamiento externo, dejando las decisiones de diseño a cada desarrollador. De esta forma, establece el marco en que los agentes FIPA existen y operan, así como el modelo de referencia para la creación, registro, localización, comunicación, migración y destrucción de los agentes [FIP01b, AM00, FIP02a].

Las entidades contenidas en el modelo de referencia mostrado en la Figura 3.6 representan únicamente servicios, no implicando ninguna configuración física concreta.

En este modelo, se definen los siguientes componentes lógicos: agente, facilitador de directorios, sistema gestor de agentes, servicio de transporte de mensajes y software. En los siguientes subapartados se detallan estos componentes.

3.5.2.1. Agente

Es el actor fundamental de la plataforma, que combina uno o más servicios en un modelo de ejecución integrado y unificado que podría incluir accesos a un software externo, usuarios humanos y diversas facilidades de comunicación. Un agente debe tener al menos un *propietario* (ya sea por parte de una organización al que agente está afiliado o por parte de un usuario humano) y debería prestar soporte a varias nociones de identidad. En este sentido, se define el identificador de agente (*Agent Identifier*, *AID*) como la etiqueta que permite distinguir al agente sin ambigüedad alguna dentro del universo de agentes.

3.5.2.2. Facilitador de Directorios (DF)

Es un componente obligatorio en la definición FIPA de la AP. El DF (*Directory Facilitator*) proporciona servicios de *páginas amarillas*[15] (nombre de agente - servicio) para el resto de los agentes. En palabras de FIPA, es el *guardián de confianza y benigno del directorio de los agentes.* Es *de confianza* en el sentido de que se esfuerza por mantener una lista de los agentes completa, actualizada y exacta. Es *benigno* porque proporciona la información más reciente acerca de los agentes de forma no discriminatoria a todos los agentes autorizados. Pueden existir múltiples DFs en la misma AP. Además, los DFs pueden estar organizados según una arquitectura federativa.

Los agentes pueden registrar sus servicios con el DF o pedirle información sobre los servicios ofrecidos por otros agentes. Cada agente que desee anunciar sus servicios a otros agentes, debería en primer lugar encontrar el DF adecuado para poder solicitar su registro. Dentro del lenguaje FIPA ACL el registro lleva a cabo mediante un *register* embebido dentro de un acto de comunicación `request`. No obstante este registro inicial no obliga al agente a prestar el servicio indicado, pudiéndose negar a llevarlo a cabo (mediante el `refuse` correspondiente). El DF no garantiza la validez o exactitud de la información proporcionada por el agente que se registra, ni controla el ciclo de vida de ningún agente.

Un acto del desregistro tiene como consecuencia que el DF se libera del compromiso de facilitar información acerca de los servicios suministrados por el agente en cuestión. Del mismo modo que es posible el desregistro, un agente puede modificar la información de sus servicios en el DF en cualquier instante. Para ambas acciones, un DF debe ser capaz de procesar mensajes con las performativas *register*, *deregister*, *modify* y *search*.

El DF de la AP presenta un AID reservado correspondiente al nombre local `df`.

3.5.2.3. Sistema Gestor de Agentes (AMS)

Es otro de los componentes obligatorios de la AP, existiendo solamente un AMS (*Agent Management System*) por cada AP.

El AMS ejerce la supervisión sobre el acceso y el uso de la AP, como la creación y destrucción de agentes, el registro de nuevos agentes, aspectos de movilidad y de

[15] a semejanza de las guías telefónicas, permite buscar a los agentes por sus servicios suministrados y no por su nombre

control del canal de comunicación y los recursos compartidos.

El AMS mantiene un directorio de los AIDs que contiene las direcciones[16] de los agentes residentes[17] en la AP en un momento dado. De este modo, el AMs ofrece un servicio de *páginas blancas*[18] (nombre de agente - dirección), pudiendo de este modo ser consultado sobre las capacidades de los agentes de su AP. Para ello, cada agente debe registrarse en el AMS para poder obtener un AID válido. El registro en el AMS implica la autorización de acceso al MTS de la AP para poder recibir y transmitir mensajes. El AMS comprueba la validez de la descripción suministrada por el agente, en particular, la unicidad local del campo nombre del agente en el AID.

Del mismo modo que en el caso del DF, las descripciones de los agentes en el AMS pueden ser modificados en cualquier momento, aunque se encuentra limitada por la autorización del AMS. El ciclo de vida de un agente (ver subapartado 3.5.2.7) termina con su desregistro en el AMS. Tras este desregistro, el AID del agente en cuestión puede ser eliminado del directorio, estando disponible para otros agentes que lo soliciten.

Un AMS representa la máxima autoridad en la gestión de la AP, de modo que puede solicitar a un agente que lleve a cabo una función relacionada con la gestión de la AP, por ejemplo, que abandone la plataforma (a través de la función *quit*). En caso de que su petición sea ignorada, dispone de autoridad para hacer cumplir esta petición.

Las funciones soportadas por el AMS son: *register*, *deregister*, *modify, search* y *get-description*. El siguiente código FIPA ACL es ilustrativo del registro de un agente en el AMS [FIP01b].

```
(request
 :sender
  (agent-identifier
    :name dummy@foo.com
    :addresses (sequence iiop://foo.com/acc))
 :receiver (set
  (agent-identifier
    :name ams@foo.com
    :addresses (sequence iiop://foo.com/acc))
 :language FIPA-SL0
 :protocol FIPA-Request
 :ontology FIPA-Agent-Management
 :content
  (action
    (agent-identifier
      :name ams@foo.com
      :addresses (sequence iiop://foo.com/acc))
  (register
    (ams-agent-description
      :name
```

[16] entre otros datos

[17] registrados con el AMS

[18] a semejanza de las guías telefónicas, permite buscar a los agentes por su nombre

```
        (agent-identifier
          :name dummy@foo.com
          :addresses (sequence iiop://foo.com/acc))
      :state active))))
```

3.5.2.4. Servicio de Transporte de Mensajes (MTS)

Es el método de comunicación por defecto entre los diversos agentes, ya sea entre los agentes que se encuentran dentro de la misma plataforma o con agentes residentes en otras APs. Todos los agentes FIPA deben tener acceso al menos a un MTS (*Message Transport Service*) y solamente pueden ser enviados a un MTS los mensajes dirigidos a un agente. Estos servicios de transporte son proporcionados por un Canal de Comunicación de Agentes *(Agent Communication Channel*, ACC), por lo que en la literatura se suelen emplearse indistintamente ambos términos.

Cuando un agente A desea mandar un mensaje a un agente B residente en otra AP existen tres opciones para poder hacerlo (ver Figura 3.7):

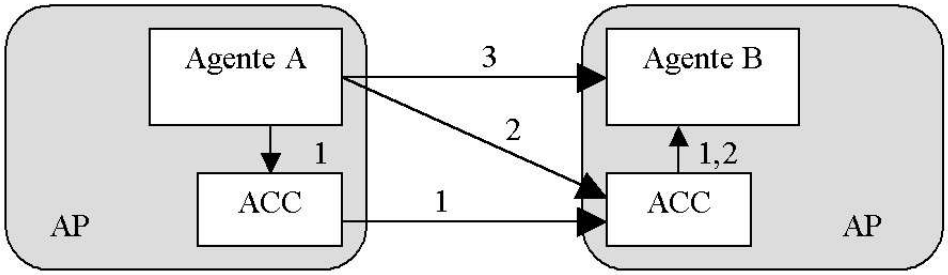

Figura 3.7.: Tres métodos de comunicación entre agentes de diferentes APs [FIP02b].

1. El agente A se lo manda primero al ACC de su AP. Este ACC se encarga de entregárselo al ACC de la AP donde reside el agente B, quien se encargará, a su vez, de enviárselo al agente destinatario. En esta gestión, los ACCs pueden acceder a otros servicios de la AP (como el AMS o el DF).

2. El agente A envía el mensaje directamente al ACC de la plataforma donde está registrado el agente B. Este ACC remoto es el encargado de reenviarlo al agente B.

3. El agente A manda directamente el mensaje al agente B, empleando un mecanismo de comunicación directa. Son los dos agentes, A y B, los encargados de manejar la transferencia de mensajes, direccionamiento, buffer de mensajes y cualquier posible mensaje de error [FIP02b].

El modelo de comunicación entre agentes es asíncrono. Por tanto,

- El ACC no se queda bloqueado ante el envío o recepción de mensajes.

- Existen colas de envío y recepción de mensajes.
- Existen políticas de gestión de colas de mensajes.

En este apartado debemos también citar al *Internal Platform Message Transport (IPMT)*, el cual representa toda la infraestructura de comunicaciones que permite la comunicación entre dos agentes de la misma AP.

3.5.2.5. Software

Describe todos los conjuntos de instrucciones no correspondientes a ningún agente, pero que pueden ser accedidos por los mismos. Los agentes pueden acceder al software, por ejemplo, para añadir nuevos servicios, adquirir nuevos protocolos de comunicación o de seguridad... [FIP01b]

3.5.2.6. Plataforma de Agentes (AP)

Proporciona la infraestructura física en que los agentes pueden ser desplegados. Consiste en la(s) máquina(s), el sistema operativo, el software de soporte de agentes, los componentes de gestión de agentes FIPA (AMS, DF y MTS) y los agentes. El diseño interno de la plataforma no se ve ligado a ninguna estandarización y depende de los desarrolladores. Obsérvese que los agentes de la AP (*Agent Platform*) pueden estar distribuidos en varias máquinas.

3.5.2.7. El ciclo de vida de un agente

De lo visto en los apartados anteriores se deduce que los agentes FIPA tiene una existencia física en una AP y emplean las facilidades ofrecidas por ella para desarrollar sus actividades. En este contexto, un agente tiene un ciclo de vida[19] físico que debe ser gestionado por la AP.

El ciclo de agente (ver Figura 3.8):

- Está ligado a la AP: Un agente es físicamente gestionado en el interior de una AP y por tanto el ciclo de vida de una agente estático está siempre ligado a una AP específica.
- Es independiente de la aplicación: El modelo de ciclo de vida de un agente es independiente del sistema de cualquier aplicación y define únicamente los estados y las transiciones del servicio del agente en su ciclo de vida.
- Es orientado a instancias: El agente descrito en el modelo de ciclo de vida se asume que es una instancia, esto es, un agente que tiene un nombre único y es ejecutado independientemente.
- Único: Cada agente tiene un único estado en cada instante dentro de su ciclo de vida y ligado a una única AP.

[19] como cualquier proceso software

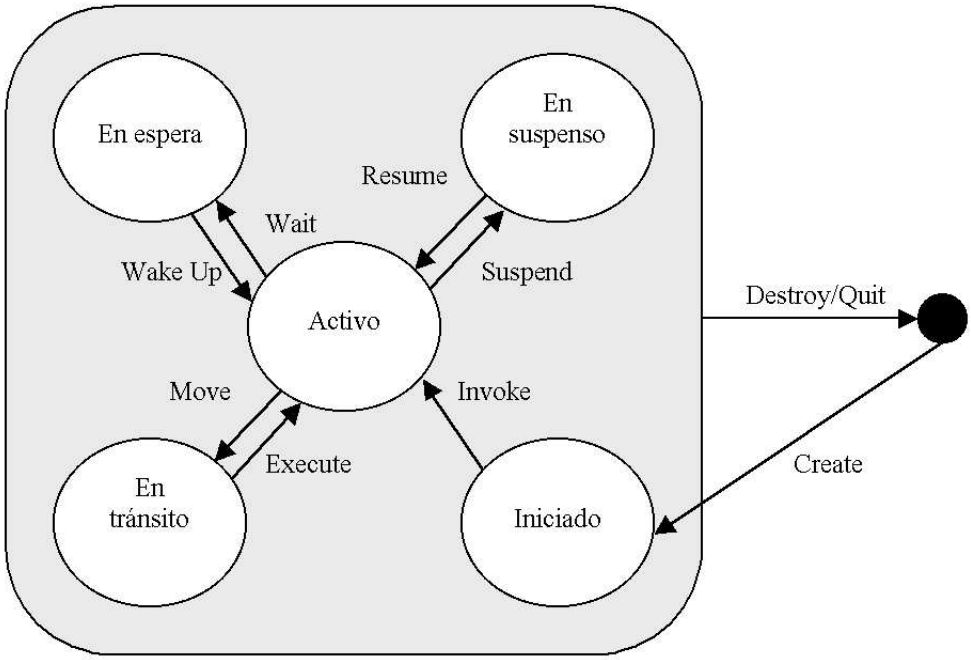

Figura 3.8.: Ciclo de vida de un agente FIPA dentro de una AP [FIP01b]

Los estados por los que puede pasar un agente en su ciclo de vida son *activo*, *iniciado*, *en espera*, *suspendido*, *en tránsito*[20] y *desconocido*[21]. Según el estado del agente, el modo de reparto de mensajes puede modificarse.

- Activo: el MTS reparte los mensajes al agente de modo normal.
- Iniciado/ En Espera/ Suspendido: El MTS almacena los mensajes en un buffer hasta que el agente alcanza un estado activo (o los reenvía a una nueva dirección si el sistema lo requiere)
- En tránsito: El MTS almacena los mensajes en un buffer hasta que el agente alcanza un estado activo (no se ha podido completar el traslado a otra AP, o este traslado ha tenido lugar) o los reenvía a una nueva dirección si el sistema lo requiere.
- Desconocido: El MTS almacena los mensajes en un buffer o los rechaza, dependiendo de la política implementada.

Las transiciones de estado de los agentes son las siguientes:

Create Creación o instalación de un nuevo agente.

Invoke Invocación de un nuevo agente.

[20] solamente alcanzable por los agentes móviles
[21] desde el punto de vista de la AP

Destroy Finalización forzada de un agente. Solamente puede ser iniciada por el AMS y no puede ser ignorada por el agente.

Quit Finalización elegante de un agente. Puede ser ignorada por el agente.

Suspend Coloca al agente en un estado de suspensión. Puede ser iniciado tanto por el AMS como por el agente.

Resume Activa al agente desde un estado de suspensión. Sólo puede ser iniciado por el AMS.

Wait Coloca al agente en un estado de espera. Solamente puede ser iniciado por el agente.

Wake Up Activa al agente desde un estado de espera. Sólo puede ser iniciado por el AMS.

Move Solamente para agentes móviles. Coloca al agente en un estado en tránsito. Solamente puede ser iniciado por el agente.

Execute Solamente para agentes móviles. Activa al agente desde un estado en tránsito. Sólo puede ser iniciado por el AMS.

3.5.3. El modelo OMG MASIF

Esta especificación, ampliamente relacionada con CORBA, tiene su origen en las Utilidades para Agentes Móviles (*Mobile Agent Facility*, MAF), una serie de reglas aprobadas por el OMG en 1997 que proporcionaban la base para la intercomunicación entre agentes heterogéneos. En Febrero del año siguiente se produce una actualización de estas reglas, aprobándose las denominadas Utilidades para la Interoperación entre Sistemas de Agentes Móviles (*Mobile Agent System Interoperability Facility*, MASIF)[22] [Cov98, MBB+98, OMG00a].

En esta estandarización para los agentes móviles se definen los siguientes términos:

Agente Entidad que actúa de manera autónoma en nombre de una persona u organización. Se necesita un identificador de agentes para la identificación, las operaciones de gestión y localización.

Sistema_de_Agentes Plataforma para la ejecución de agentes.

Lugar Contexto dentro del sistema de agentes donde se puede ejecutar un agente. Como consecuencia, un agente realizará sus desplazamientos de un lugar a otro lugar.

Localización Asociada a un lugar, indica su nombre y la dirección que ocupa en el sistema.

[22] El inicio del proceso data de Noviembre de 1995, siendo presentadas en el OMG Technical Meeting en Diciembre de 1997. Entre las organizaciones participantes, se encuentran IBM, GMD FOKUS y General Magic.

Localidad Propiedad de cercanía al destino, ya sea en el mismo ordenador o en la misma red.

Autoridad Persona o entidad en nombre de la cual un agente o sistema de agentes actúa.

Región Conjunto de agentes con un buscador (*finder*) y con la misma autoridad. Agrupación de los sistemas multiagente en general heterogéneos.

Buscador Registro dinámico para localizar agentes, lugares y sistemas de agentes. Constituye el servicio de nombres de una región, aunque varias regiones pueden compartir el mismo buscador.

Infraestructura de comunicaciones Provee al sistema de una serie de servicios de transporte que le permite interoperar con otros sistemas conectados a la red, ya sean multiagente o de cualquier otro tipo.

Estas relaciones son mostradas en las figuras 3.9 y 3.10.

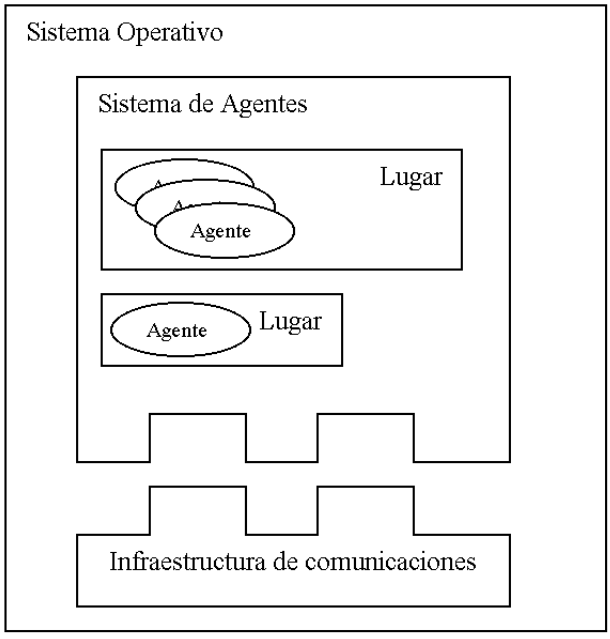

Figura 3.9.: Sistema de Agentes MASIF [Lab99]

Los aspectos sujetos a la estandarización MASIF son:

- La gestión de agentes. Aspectos relacionados con la creación, suspensión, reactivación y finalización de un agente.
- Transferencia de agentes. Es deseable que las aplicaciones de agentes puedan moverse entre sistemas de agentes de diferente tipo.

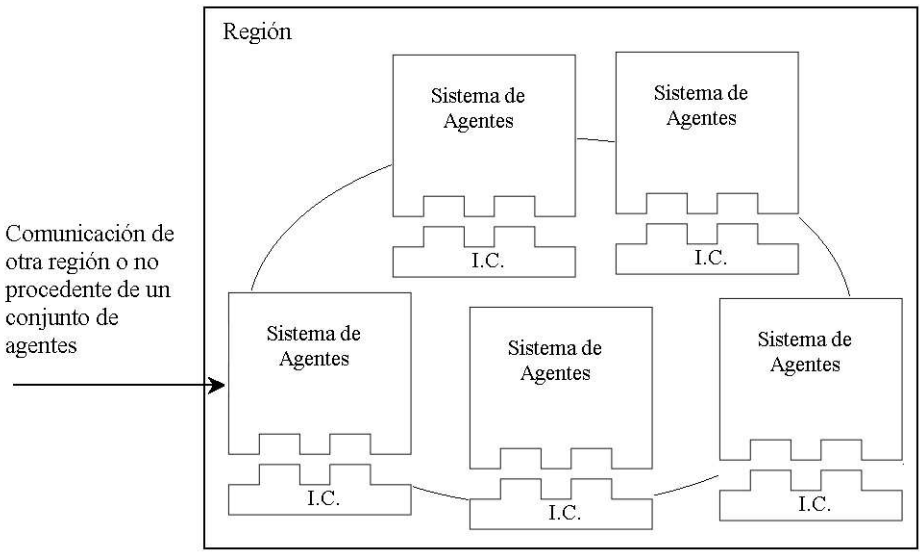

Figura 3.10.: Región MASIF [Lab99]

- Identificadores de agentes y sistemas de agentes. Pretende estandarizar la sintaxis y semántica de estos identificadores.
- Tipo de sistema y sintaxis de localización de agentes. La transferencia de agentes no puede ocurrir a menos de que el sistema soporte al agente. La sintaxis de localización permite que los sistemas de agentes puedan localizarse entre sí.

Obsérvese que MASIF no define ningún estándar sobre la comunicación agente-agente.

Las utilidades para los sistemas de agentes móviles forman parte de las utilidades definidas en CORBA. Los objetos definidos en las interfaces MASIF son objetos CORBA normales que pueden hacer uso de otros servicios y utilidades, pero los sistemas y agentes definidos en MASIF no tienen por qué ser objetos CORBA. Los servicios más ligados a las MASIF son:

- Servicio de nombres. Permite llevar un registro de los objetos CORBA y asociarles nombres. Los objetos que tengan que relacionarse con otro tipo de objetos CORBA, deben inscribirse en el servicio de nombres, además de en el registro de su región.
- Servicios para el ciclo de vida. Estandarizan las operaciones de creación, borrado, copia y traslado de objetos CORBA.
- Servicios de seriación[23]. Contiene los mecanismos para almacenar el estado de un objeto en una serie o flujo de datos para poder restaurarlo posteriormente.

[23] Proceso de almacenar un agente en una forma serial, suficiente para poder reconstruir el agente. El proceso inverso se denomina *deseriación.*

- Servicios de seguridad.

En esta estructura MASIF el buscador se implementa a través de la interfaz *MAFFinder*. Esta interfaz declara una serie de métodos relacionados con la búsqueda, registros y desregistros tanto de agentes, sistemas de agentes y lugares. Disponiendo de un objeto que implemente la interfaz, existen cuatro métodos básicos para la búsqueda de agentes.

- Búsqueda en bruto. Se manda a un agente a cada uno de los sistemas de la región, buscando al agente deseado.
- Registro de operaciones. Antes de cada traslado el agente deja una marca en su sistema de agentes indicando el destino, lo que facilita su posterior localización.
- Registro de agentes. Se registra la localización de cada agente en una base de datos actualizada.
- Anuncio del agente. Se registran solamente las localizaciones estáticas y se implementa una búsqueda en bruto para aquellos agentes que no se han anunciado.

MASIF también proporciona otra interfaz llamada *MAFAgentSystem*, que declara una serie de métodos relacionados con la gestión básica de los agentes móviles. Estos métodos reflejan aspectos como la creación de agentes, la búsqueda de conjuntos de agentes más cercana con un cierto perfil y la obtención del estado de un agente, entre otros [Lab99].

La Figura 3.11 muestra la arquitectura de un MAS que sigue el estándar MASIF, incluyendo la definición de estas interfaces.

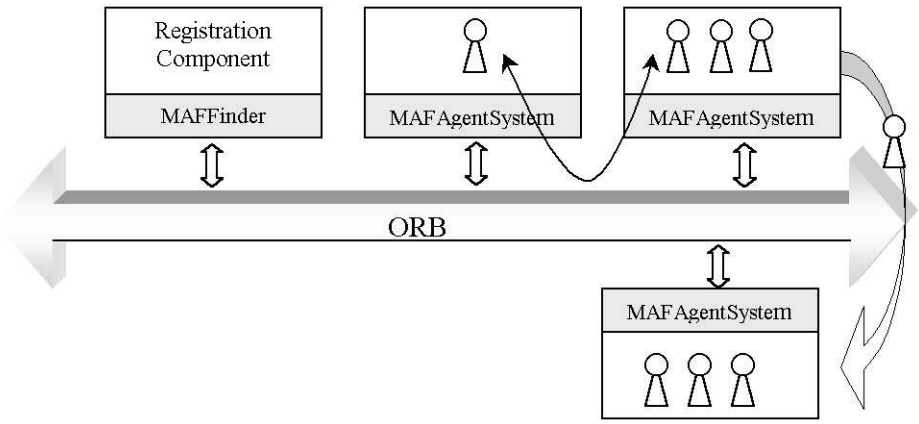

Figura 3.11.: Interfaces MASIF [Cov98]

3.5.3.1. Transferencia de agentes

MASIF define unos algoritmos de transferencia de agentes, que permite a éstos trasladarse de un lugar a otro. [MBB+98]

Por el lado del emisor, los pasos a cubrir son:

1. Suspender al agente.
2. Identificar el estado del agente transferible.
3. Seriar la clase del agente y su estado.
4. Codificarlos para el protocolo de transporte escogido.
5. Proporcionar información de autenticación al receptor del agente.
6. Transferir el agente.

Por el lado del receptor:

1. Autentificar al cliente.
2. Decodificar al agente.
3. Deseriar la clase del agente y su estado.
4. Instanciar el agente.
5. Restaurar el estado del agente.
6. Volver a activar la ejecución del agente.

3.6. Restantes Especificaciones del Estándar FIPA

En apartados anteriores hemos visto las especificaciones dadas por FIPA referidas a la comunicación y arquitectura de los sistemas multiagente. No obstante, existen otras especificaciones objeto de estandarización y con ello de compatibilidad de los MAS. Como en los anteriores casos, FIPA no especifica cómo debe ser implementados los elementos a estandarizar, sino que únicamente describe cómo debe ser su comportamiento externo.

La labor de elaborar los posibles estándares dentro de FIPA corre a través de una serie de comités técnicos. A fecha de escritura de este trabajo, se encuentran activos siete comités encargados de los aspectos relativos a:

- Protocolos de interacción: Trabaja en el desarrollo de una segunda generación de protocolos de interacción. Por ejemplo, ya existe un documento de propuesta de un nuevo protocolo de interacción para el mecanismo de recuento de Borda [FIP02e] de búsqueda de consenso entre varias partes.
- Metodologías: Trabaja en la identificación de una metodología para el desarrollo de MAS.

- Modelado: Intenta desarrollar una semántica común, meta-modelo y sintaxis para las metodologías basadas en agentes que sean independientes del fabricante.
- Ontologías: Se encarga de los métodos de estandarización para el conocimiento compartido y filtrado a través de una representación ontológica que permita a los sistemas automatizar el procesamiento de mensajes con respecto a una clasificación semántica de referencias internas.
- Seguridad: Su objetivo es liderar la investigación y desarrollo de la seguridad para sistemas multiagente.
- Semántica: Trabaja actualmente en la adopción de un nuevo marco semántico que dé cuenta de los actos comunicativos y protocolos de interacción FIPA, así como a un número de construcciones tales como contratos, políticas, descripciones de agentes...
- Servicios: Se encarga de proporcionar más estructura y soporte para los servicios del marco FIPA. Anima a los desarrolladores a proporcionar un metamodelo de servicios en el cual puedan expresarse los modelos de servicios disponibles actualmente (CORBA, servicios Web, DAML-S).

El trabajo de estos comités ha dado lugar a una serie de documentos e informes técnicos que reflejan los estándares FIPA que deben ser tenidos en cuenta en el desarrollo de una aplicación que quiera ser compatible con otras aplicaciones FIPA.

Las especificaciones aprobadas a fecha de escritura de este trabajo se refieren a[24]:

- Especificaciones relativas a aplicaciones (aplicación nómada, integración de agente software, asistente personal de viajes, difusión y entretenimiento audio-visual, provisión y gestión de red, asistente personal, buffer de mensajes, calidad de servicio)
- Especificaciones sobre la arquitectura abstracta (políticas y dominios).
- Especificaciones sobre la comunicación de agentes (estructura de mensajes ACL, servicio de ontologías, protocolos de interacción, actos de comunicación, lenguajes de contenido de mensajes).
- Especificaciones sobre la gestión de agentes (AMS, DF...)
- Especificaciones sobre el transporte de mensajes (interoperabilidad de mensajes, representaciones del mensaje como cadena de caracteres, en bits ó XML, protocolo de transporte...)

Como se puede apreciar, FIPA intenta cubrir la mayor parte posible de los aspectos de un sistema multiagente, buscando que un nuevo MAS interopere adecuadamente con otro MAS existente.

Algunas de las mencionadas especificaciones, como las referidas a la gestión de agentes y la comunicación han sido tratadas en secciones anteriores. Dedicaremos el resto de la sección a las dos más involucradas en este trabajo, las referidas al asistente personal y al servicio de ontologías.

[24] http://www.fipa.org/repository/bysubject.html

3.6.1. Especificaciones FIPA sobre un Aplicación Asistente Personal

Esta especificación trata de estandarizar el comportamiento externo que debe presentar una clase de agente de gran importancia, la de asistente personal (*Personal Assistant,* PA). Este PA es un agente software que actúa de forma semi-autónoma en nombre de un usuario, modelando los intereses del mismo y proporcionando servicios a su usuario o a otros usuarios y PAs. FIPA lo asemeja al papel de secretario/a que se encarga de las labores rutinarias y permite al usuario concentrarse en su trabajo real. No obstruye en ningún caso, pero está preparado cuando se le necesita. Es rico en conocimiento acerca del usuario y sus áreas de trabajo.

El concepto de PA es muy amplio. Existen numerosas funciones que pueden recaer sobre él: gestionar la agenda de su usuario, planificar sus viajes, encargar mercancías, recomendar espectáculos de entretenimiento, filtrar y ordenar el correo electrónico...[FIP01j]

3.6.1.1. Análisis General

En líneas generales un PA comprende:

- Inteligencia y habilidades asociadas como racionalidad y adaptabilidad/aprendizaje.
- Conocimiento incluyendo hechos, reglas y conocimiento adaptado/aprendido para y de su usuario.
- Habilidades y facilidades de interacción con el usuario, otros agentes y servicios software/hardware.
- Los servicios y funciones con los que el agente trabaja.

La Figura 3.12 muestra el modelo de referencia FIPA para el PA. En él se pueden apreciar los elementos que dotan al PA de las cualidades descritas.

3.6.1.2. Servicios de Directorio

Una de las funciones básicas para un PA es la gestión del directorio del usuario. Este directorio contiene los números de teléfono, direcciones e información personal y útil acerca del usuario, otros usuarios y empresas. Esta información facilita las respuestas que el PA puede proporcionar a las necesidades del usuario de una forma inteligente, en base al contexto de la petición. Por ejemplo, si el usuario pide llamar a una organización y el PA se da cuenta de que la llamada no puede ser hecha, entonces puede sugerir al usuario acciones alternativas, deduciendo las posibles intenciones del usuario y en los servicios proporcionados por la organización. En este caso, una llamada hecha a una agencia de viajes fuera de su horario de apertura implicaría, por ejemplo, que el PA sugiera conectarse con un servicio web de reserva de billetes de una compañía aérea.

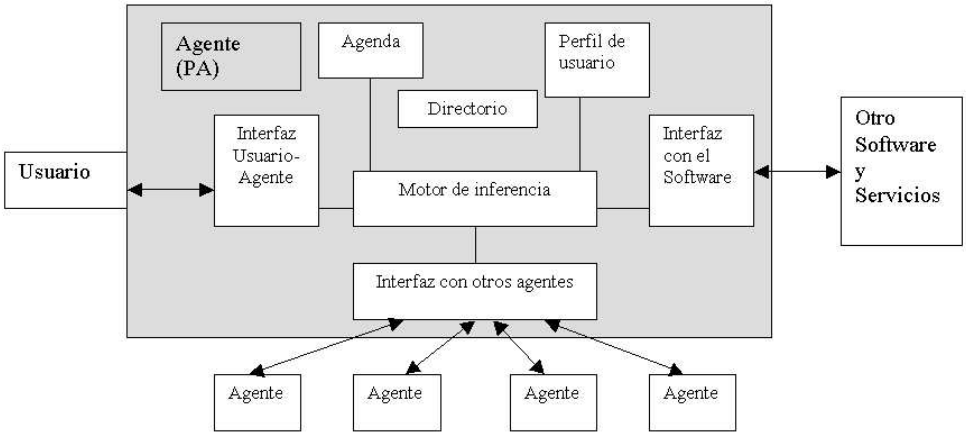

Figura 3.12.: Modelo de referencia FIPA para el PA [FIP01j].

3.6.1.3. Servicios de Programación de Reuniones

El PA disfruta de una habilidad de gestión de calendario que, entre otros propósitos, puede ser empleado para la programación de reuniones y en la negociación con otros usuarios y PAs. Esta funcionalidad incluye:

- Identificar una fecha y hora apropiada para todos los participantes de la reunión.
- Reservar un lugar apropiado para la reunión.
- Organizar los aspectos colaterales.
- Mandar recordatorios a todos los participantes.
- Manejar cualquier problema que pudiese surgir en una fecha posterior.
- Cancelar las reuniones.

3.6.1.4. Servicios de Gestión de la Información

Muchos de los profesionales se ven inundados cada día con información proveniente de muchas fuentes. Un PA podría de una forma semi-automática filtrar, ordenar o cualquier otra acción que contribuya a aliviar al usuario de una serie de labores rutinarias y que le ocupa un tiempo que puede ser empleado en otro trabajo. Estas tareas incluyen:

- Filtrar de correos electrónicos y *news*.
- Ordenar y priorizar la información recibida.
- Responder o reenviar información a otros usuarios de modo automático.

Un aspecto clave de la gestión de información consiste en la entrega de informaciones de alta prioridad. Esta entrega depende de la localización del usuario, de las limitaciones en la infraestructura y de las preferencias. Por ejemplo, puede mandar los mensajes importantes al teléfono móvil del usuario.

Un aspecto menos desarrollado consiste en la gestión y almacenamiento de la información de usuario. Incluso el almacenamiento en un ordenador personal a veces llega a ser difícil ya que, entre otros motivos, los ficheros suelen estar duplicados, la estructura de directorios suele hacerse de cualquier modo y los sistemas de ficheros no ofrecen un indexado ni facilidades de búsqueda de contenido adecuados.

3.6.1.5. Servicio de Planificación de Viajes

Un PA puede ayudar a su usuario a planificar un viaje mediante la interactuación con él, con otros agentes o con servicios de directorios externos. En este caso, un PA puede asumir mucha de la funcionalidad de un agente de viajes personal (*Personal Travel Assistant,* PTA), cuya especificación FIPA se encuentra en [FIP01k].

3.6.1.6. Ontología de PA

FIPA especifica una ontología para el PA, llamada FIPA-PA que incluye los siguientes términos:

`meeting-description` Representa la descripción de una reunión. Contiene tres parámetros: `identifier` (el identificador de la reunión), `user` (el usuario que organiza la reunión) y `details` (detalles de la reunión).

`meet` Función que tiene el efecto de pedir al PA que negocie una reunión entre el iniciador y los asistentes usando el protocolo de interacción *FIPA Contract-Net* (ver apéndice E).

`schedule` Función que tiene el efecto de pedir al PA que programe una reunión entre el iniciador y los asistentes.

`participate` Función que tiene el efecto de pedir al PA que participe en una reunión.

`travel` Función que tiene el efecto de pedir al PA que planifique un viaje para el iniciador.

3.6.2. Especificaciones FIPA sobre el Servicio de Ontologías en un MAS

El modelo de comunicación de agentes FIPA se basa en la suposición de que dos agentes que desean conversar entre sí, comparten una ontología[25] común en sus discursos. Este hecho asegura que los agentes atribuyen el mismo significado a los símbolos expresados en el mensaje. Para un dominio dado, los diseñadores podrían decidir emplear ontologías que son declaradas explícitamente o, de forma alternativa,

[25] Conviene recordar que las ontologías no contienen únicamente el vocabulario de términos empleados, sino además axiomas explícitos que intentan ajustar el significado de estos términos. Ver capítulo 2 para un mayor detalle.

Figura 3.13.: Modelo de comunicación basado en ontología [FIP01h]

las ontologías son codificadas implícitamente en la implementación software del agente y por tanto no existe formalmente un servicio de ontologías. Este segundo caso debería ser considerado en un menor grado de importancia por parte de los desarrolladores, puesto que en un entorno abierto, como debe ser un MAS, los agentes se diseñan alrededor de varias ontologías (ver Figura 3.13).

La declaración explícita de ontologías lleva consigo una serie de ventajas para las aplicaciones: permite la consulta sobre conceptos, la actualización y reutilización de ontologías, la interoperabilidad de MAS heterogéneos. Además, hace que no sea necesario que cada agente de forma individual contenga la ontología en su totalidad, con el consiguiente ahorro de recursos. En este contexto de definición explícita de ontologías, FIPA define un servicio de ontologías que será llevado a cabo por un agente llamado agente de ontología (*Ontology Agent*, OA*)*. El papel de este agente en la comunidad es proporcionar todos o algunos de los siguientes servicios:

- descubrimiento de ontologías públicas en orden de acceder a ellas.
- mantenimiento de un conjunto de ontologías públicas (por ejemplo, registrándolas en el DF, modificarlas...).
- traducir expresiones entre diferentes ontologías y/o diferentes lenguajes de contenido.
- responder a consultas sobre relaciones entre términos u ontologías.
- facilitar la identificación de una ontología común de comunicación entre dos agentes.

Como se ha comentado, el OA (de existir) no está obligado a implementar todos estos aspectos, pero sí debe estar involucrado en toda comunicación que trate sobre ellos.

3.6.2.1. Servicio de Ontologías

El OA proporciona acceso a una o más servidores de ontologías y podría proporcionar servicios de ontología a una comunidad de agentes. Como el resto de los agentes, el OA registra su servicio en el DF, así como la lista de ontologías mantenidas y sus capacidades de traducción. Cada ontología dispone de un nombre único por el

que son accedidas y gestionado por el OA. De este modo, los agentes pueden consultar al DF sobre el OA específico que gestiona una ontología en concreto. Los agentes del sistema pueden pedir al OA que defina, modifique o elimine términos y definiciones de la ontología; pueden pedir que traduzca expresiones entre dos ontologías diferentes cuando esta traducción sea posible, preguntar sobre relaciones entre los términos o sobre una ontología común que permita la comunicación con otros agentes. El OA se reserva el derecho a rechazar la petición.

Un tratamiento adecuado de este proceso lleva implícito un acuerdo sobre los términos en la comunicación sobre las ontologías, es decir, una *metaontología*. Esta metaontología, conocida como `FIPA-Meta-Ontology` se encuentra definida en [FIP01h], adoptando para su especificación el modelo de conocimiento de OKBC[26]. Pretende ofrecer un marco que permita a los agentes hablar sobre el conocimiento y trabajar con él, por ejemplo para consultar la definición de un concepto o definir uno nuevo. Para ello describe las primitivas empleadas por un lenguaje de representación del conocimiento, como conceptos, parámetros, relaciones...

FIPA también define una ontología llamada `FIPA-Ontol-Service-Ontology` para describir los aspectos relacionados con el servicio de ontologías. Los términos incluidos en ella son:

`ontology-description` Descripción de la ontología. Contiene cuatro parámetros: `ontology-name` (nombre simbólico de la ontología), `version` (versión de la ontología), `source-languages` (lista de los lenguajes en que se representa la ontología) y `domains` (la lista de dominios de aplicación donde se puede aplicar la ontología).

`translation-description` Descripción de la traducción. Contiene tres parámetros: `from` (representación de la ontología fuente), `to` (representación de la ontología destino) y `level` (relación de traducción entre las ontologías fuente y destino).

A modo de ejemplo de empleo de esta ontología, se reproduce el siguiente código de un mensaje FIPA-ACL (omitiendo por simplicidad los campos relativos al protocolo, lenguaje...) en el que un agente consulta al DF qué OAs dan soporte a la ontología `FIPA-VPN-Provisioning`.

```
(request
  :sender
     ...
  :receiver
     ...
  ...
  :content
    (action
      (agent-identifier
        :name df@bar.com
        :addresses (sequence iiop://bar.com/acc))
```

[26] Open Knowledge Base Connectivity. http://www.ai.sri.com/~okbc/

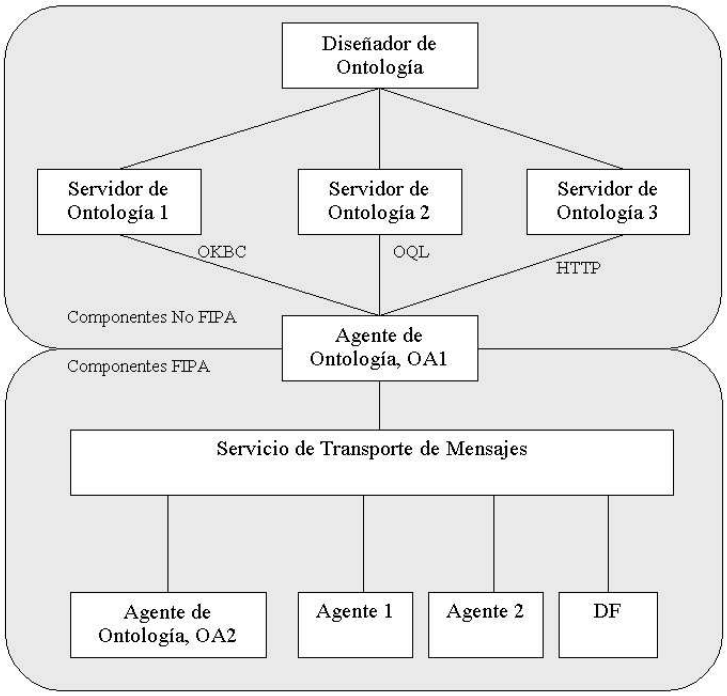

Figura 3.14.: Modelo de referencia FIPA para el Servicio de Ontología [FIP01h]

```
(search
  (df-agent-description
    :services (set
      (service-description
        :type fipa-oa
        :ontology (set FIPA-Ontol-Service-Ontology)
        :properties (set
          (property
            :name supported-ontologies
            :value (set
              (ontology-description
                :ontology-name FIPA-VPN-Provisioning)))))))))
```

En el modelo de referencia FIPA del servicio de ontologías (ver Figura 3.14), las ontologías se encuentran almacenadas en un servidor de ontologías. De modo general, en un MAS pueden coexistir varios servidores de ontologías con interfaces y capacidades diferentes. El OA permite a los agentes descubrir estas ontologías en los servidores y acceder a sus servicios de un modo único.

3.6.2.2. Relaciones entre Ontologías

Los agentes se pueden beneficiar en algunas aplicaciones de conocer la existencia de relaciones entre las diversas ontologías disponibles. Estas relaciones permitirían, por ejemplo, que un agente eligiese el modo de comunicarse con otro agente. La consulta debe realizarse a través del siguiente predicado:

```
(ontol-relationship ?O1 ?O2 ?level)
```

donde `level` indica la relación entre las ontologías `O1` y `O2`. Los posibles valores de este parámetro son:

Extension O1 extiende la ontología O2.

Identical Las dos ontologías son idénticas. Por idéntica se quiere decir que el vocabulario, los axiomas y el lenguaje de representación son físicamente iguales, como lo son dos copias del mismo fichero. Sin embargo, dos ontologías idénticas pueden ser accedidas por distintos nombres.

Equivalent Las dos ontologías son equivalentes. Por equivalente se quiere decir que el vocabulario y los axiomas son los mismos, pero posiblemente estén expresadas en lenguajes de representación diferentes (por ejemplo Ontolingua y XML).

Weakly-Translatable La ontología O1 es débilmente traducible a la ontología O2. Es decir, es posible traducir términos desde una ontología fuente a una ontología destino, incluso si existe una posible pérdida de información.

Strongly-Translatable La ontología O1 es fuertemente traducible a la ontología O2. Es decir, el vocabulario de la ontología fuente puede ser traducido totalmente a términos de la ontología destino, los axiomas de la ontología fuente se mantienen en la de destino, no existe pérdida de información entre ambas y no se introduce ningún elemento de inconsistencia.

Approx-Translatable La ontología O1 es aproximadamente traducible a la ontología O2. Es decir, si es débilmente traducible con la introducción de algunas posibles inconsistencias, como por ejemplo, algunas relaciones definidas en la ontología fuente que ya no son válidas o algunas restricciones que no se aplican.

Se pueden encontrar ejemplos de estas relaciones, así como propiedades entre ellas en [FIP01h].

3.6.2.3. Ejemplo de Utilidad del OA en un Sistema Multiagente

Terminaremos este apartado mostrando un ejemplo que muestra la utilidad del uso de ontologías en un sistema multiagente.

Considérese un agente B capaz de clasificar fotografías en base a lo que representan y mandarlas según las peticiones de otro agente.

1. Sea un agente A, por ejemplo, una agente de interfaz de usuario, que busca fotografías de cítricos infectados para su usuario, interesado en realizar un

diagnóstico de sus naranjos. El agente A pide al agente B que le envíe fotografías de *cítricos infectados* referidos a una ontología de dominio llamada, por ejemplo `agricultura`.

2. El agente B descubre que no existe ninguna fotografía bajo el epígrafe de *cítricos infectados*. Antes de enviar la respuesta al agente A, el agente B consulta a un OA con acceso a la ontología `agricultura` en búsqueda de sub-especies de cítricos.

3. El OA responde al agente B, informando que *naranja* y *limón* son sub-especies de *cítrico*.

4. El agente B encuentra fotografías de *naranja infectada* y *limón infectado* y las mandas al agente A.

5. Este ejemplo puede continuar con el usuario buscando varias de las fotos enviadas por el agente B, intentando encontrar parecidos con el problema de sus naranjos. Cuando encuentra el problema, puede pedir una solución al agente A. En este caso, el OA puede ser de nuevo de utilidad. Por ejemplo, el agente A podría acceder al OA para explicar el tipo de infección de la fotografía.

6. La existencia de una ontología explícita gestionada por un agente externo permite al agente B concentrarse en su tarea de clasificar y mandar fotografías, olvidándose por completo de la gestión y mantenimiento de la ontología.

3.7. Programación orientada a agentes, AOP

Los objetos y los agentes son similares en varios aspectos. De hecho los agentes de modo conceptual son objetos en el sentido de que poseen identidad (se puede distinguir un agente de otro agente), poseen su propio estado y comportamiento, y finalmente poseen interfaces mediante las cuales se comunican entre sí y con otros entes. Debido a estas analogías, la programación orientada a objetos se constituye como un método adecuado para la implementación de MAS.

En este sentido, Guessoum [GB99] afirma que

> La programación concurrente orientada a objetos es la tecnología más apropiada y prometedora para la implementación de agentes. El concepto de *objeto activo* podría ser considerada como la estructura básica para la construcción de agentes. (...) La uniformidad del mecanismo de comunicación de objetos proporciona facilidades para implementar agentes sociables y el concepto del encapsulado habilita la combinación de varios tipos de agentes. Más aún, el mecanismo de herencia permite la especialización y factorización del conocimiento.

Sin embargo la programación orientada a objetos suele evolucionar en la mayoría de las herramientas MAS, llegándose a hablar de *programación orientada a agentes* (*Agent-Oriented Programming*, AOP).

Mientras que en la OOP los sistemas son considerados como un conjunto de objetos (encapsulados con sus datos y sus métodos) comunicándose entre ellos para realizar

una serie de operaciones internas, la AOP toma a un *agente* como la unidad principal de encapsulación. El agente se programa con elementos de alto nivel como objetivos, tareas o creencias y de forma que el tipo de mensajes que un agente intercambia con otro hagan referencia a mecanismos, también de alto nivel, como datos, peticiones, ofertas, promesas o la aceptación o rechazo de las ofertas recibidas [Sho93].

Para Jennings et al. [JSW98] existen tres diferencias principales entre la OOP y la AOP:

> La primera es el grado en que los agentes y los objetos son autónomos. No pensamos en los agentes como invocadores de métodos entre ellos, sino pidiendo la realización de acciones. En el caso orientado a objetos, la decisión de la ejecución recae en el objeto que invoca al método. En el caso de los agentes, la decisión recae en el agente que recibe la petición[27].
>
> La segunda distinción importante... es respecto a la noción de un comportamiento autónomo y flexible (reactivo, pro-activo, social)
>
> La tercera distinción importante... es que se considera que cada agente tiene su propio flujo de control... en el modelo de objetos estándar hay un único flujo de control para todo el sistema

De modo gráfico, Van Parunak [vP97] resume lo anterior en:

$$\text{AOP} = \text{OOP} + \text{flujo de control independiente} + \text{iniciativa}$$

Basándose en este paradigma [Pet00], se han definido una serie de lenguajes orientados a agentes que además tienen en cuenta las actitudes mentales de los agentes, como el caso del modelo BDI, aunque ninguno de ellos ha sido adoptado de forma masiva en el desarrollo de MAS. Ejemplos de estos lenguajes [Alo02] son Agent0[28] [Sho93], Placa [Tho93] y Agent-K.

Finalizaremos esta sección, ilustrando el formalismo de este tipo de lenguajes, refiriéndonos como ejemplo al sistema Agent0. El componente lógico del sistema es una lógica multi-modal cuantificada (creencia, restricción y habilidad), permitiendo un acceso directo al referencias temporales. Así, una fórmula como la siguiente

$$CAN\hat{}5\_a\ open(door)\hat{}8\ B\hat{}5\_b\ CAN\hat{}5\_a\ open(door)\hat{}8$$

debe ser leída como "si en el tiempo 5 un agente *a* puede asegurar que la puerta está abierta en el tiempo 8, entonces en el tiempo 5, el agente *b* cree que en el tiempo 5 el agente *a* puede asegurar que la puerta estará abierta en el tiempo 8".

[27] Los agentes pueden decidir individualmente si responder a mensajes provenientes de otros agentes. En el caso de los objetos, hacen lo que les piden que hagan, a menos que esté prohibido por motivos de seguridad u otras restricciones.

[28] Primer intento de Shoham de lenguaje AOP, http://www.cs.caltech.edu/~bond/courses/cs101c/agents16/node3.html

3.8. Marcos de Trabajo de Agentes

Los marcos de trabajo de agentes son las herramientas de programación para la construcción de agentes, de modo que a partir de ahora emplearemos ambos términos indistintamente [FM99, Pis98]. En la actualidad existe un amplio abanico de estas herramientas, ya sean comerciales o de código libre. A pesar de la existencia de lenguajes basados en la AOP (ver sección 3.7), la inmensa mayoría de estas herramientas han sido desarrolladas en el lenguaje de programación Java, aprovechándose de su cualidad de lenguaje orientado a objetos multiplataforma y multihilo. De este modo, los agentes pueden residir en máquinas diferentes, incluso con diferentes sistemas operativos, ejecutándose con varios flujos de control. Sin embargo, a pesar de emplear un lenguaje orientado a objetos, estas herramientas emplean abundantes conceptos de la programación orientada a agentes, como la autonomía de ejecución de métodos internos o el flujo de control independiente.

De las herramientas MAS existentes, analizaremos a continuación aquéllas que a nuestro juicio son, a fecha de escribir el presente proyecto de tesis, representativas de las diferentes tendencias en cuanto a comunicación en MAS.

- ***KQML***: JATLite y Jackal
- ***OMG MASIF***: Grasshopper[29]
- ***FIPA***: FIPA-OS, Zeus y JADE

Aparte de estos marcos de trabajo, la subsección 3.8.7 analiza de forma más breve otras herramientas MAS.

Todas estas herramientas han sido estudiadas durante la realización de este trabajo, en busca del mejor marco de trabajo posible. Los criterios seguidos han sido:

- Facilidad de implementación de nuevos agentes
- Organización de forma lógica de las clases existentes en la distribución
- Distribución en régimen de código libre o abierto
- Robustez en su funcionamiento
- Cumplimiento de los diversos protocolos estándar (FIPA, KQML)
- Empleo de diferentes protocolos de comunicación (RMI, CORBA)
- Herramientas de depuración

3.8.1. FIPA-OS

Figura 3.15.: Logotipo FIPA-OS.

[29] Actualmente también cumple con las directrices FIPA

3.8.1.1. Breve Descripción

FIPA-OS[30] es un marco de trabajo de agentes desarrollado por Nortel Networks[31], que trata de suministrar al usuario una plataforma cuya arquitectura destaque por su facilidad de extensión y por su modularidad. Implementada en Java, es de código libre bajo los términos de una licencia EPL[32] que prohibe su explotación comercial. Cumple con numerosas especificaciones FIPA[33], como las relativas al manejo de los agentes, la transmisión y recepción de mensajes ACL, así como a los protocolos de conversación. Su diseño permite incluirlo en sistemas con la herramienta JESS[34].

Las comunicaciones entre los agentes de la misma plataforma se realizan mediante Java RMI y los mensajes entre plataformas se realizan mediante CORBA, RMI y TCP.

Al contrario de otras plataformas, FIPA-OS soporta perfiles de plataformas y de agentes, que son codificados en XML/RDF, con las ventajas que eso supone [Lau00a]. Como ejemplo de estos perfiles, incluimos a continuación el fichero *ACC.profile* de la distribución 2.1.0, donde se puede observar cómo se codifican las direcciones de la plataforma que se emplean en la comunicación de los agentes.

```
<?xml version="1.0" encoding="UTF-8"?>
<?enhydra-unmarshall package="fipaos.agent.profile"?>
<aCCProfile localAddressesLocation="platform.addresses">
    <databaseProfile databaseType="SerializationDatabase"
     databaseLocation="../databases/" />
    <internalAddress address="fipaos-rmi://benijo:3000" />
    <externalAddress address="corbaname://benijo:4000" />
</aCCProfile>
```

FIPA-OS incluye una plantilla denominada *GenericAgent.java* que sirve de base para la implementación de agentes más complicados. Esta plantilla incluye un *gestor de las tareas (Task Manager)* a desarrollar por el agente (entendiendo por *tarea* como la unidad primitiva de trabajo) y un *gestor de conversaciones* (*Conversation Manager*) encargado de observar el cumplimiento de los protocolos FIPA relativos a las mismas. Cada tarea es ejecutada en un *thread* diferente. De este modo se aprovecha la capacidad *multithread* de Java para ejecutar diversos flujos de control de manera simultánea.

En el paquete se incluyen clases relativas a los agentes de gestión del sistema definidos por FIPA: el Agent Management System (AMS), el Directory Facilitator (DF), el Agent Communication Channel (ACC) y el Internal Platform Message Transport (IPMT).

Esta herramienta se completa con diversas utilidades como un monitor de las tareas pendientes de ejecución, un visualizador de los *threads* activos en el sistema, un generador de tareas (generador automático de código Java de todas las subclases necesarias para la ejecución de una tarea) y una interfaz gráfica que permite al usuario enviar directamente mensajes ACL de modo manual a otros agentes del sistema.

[30] http://www.emorphia.com/research/about.htm
[31] http://nortelnetworks.com
[32] http://kxmlrpc.enhydra.org/software/license/epl.html
[33] De hecho fue la primera herramienta de código abierto en implementar el estándar FIPA
[34] Java Expert System Shell. http://herzberg.ca.sandia.gov/jess

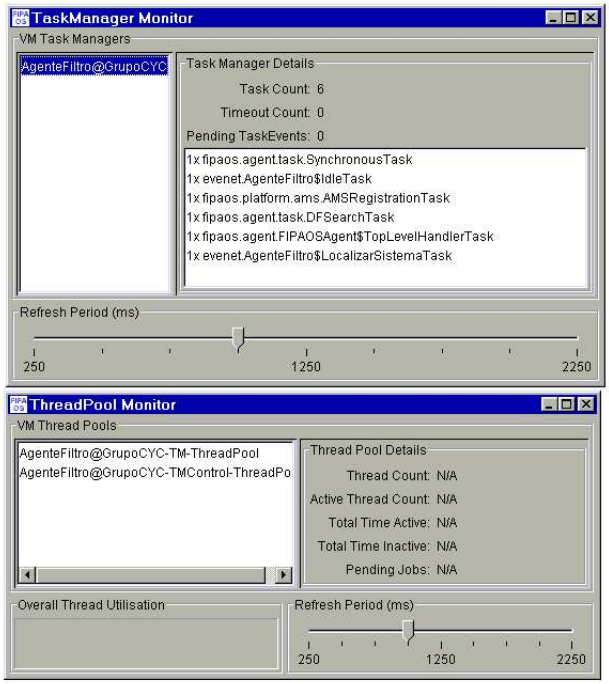

Figura 3.16.: Utilidades de Visualización de Threads y Monitor de Tareas Pendientes FIPA-OS.

3.8.1.2. Modelo de ejecución de un agente FIPA-OS

Como se ha indicado antes, el modelo de ejecución de un agente FIPA-OS viene determinada por dos objetos principales: el gestor de conversaciones y el gestor de tareas [PBH00, FIP01i].

Cuando se manda un mensaje a un agente, el Servicio de Transporte de Mensajes (MTS), es el encargado de enviar el mensaje al gestor de conversaciones del agente destinatario. Este último se encarga de comprobar el identificador de conversación del mensaje recibido (el campo `conversation-id`) y determina si el mensaje corresponde a una conversación ya existente. En caso afirmativo, comprueba si el mensaje cumple con el protocolo de la conversación, y si es así, una copia del mensaje se deposita en el objeto *Conversation* correspondiente. Por el contrario, si no existe todavía el objeto, se crea uno nuevo tras comprobar que el protocolo está soportado por el sistema.

Por otra parte, el gestor de tareas también comprueba el identificador de conversación del mensaje recibido, pero en este caso para comprobar si la conversación correspondiente está asociada a una tarea. Si lo está, se crea un objeto *Task Event* que se añade a la tarea correspondiente. Si no lo está, este objeto será añadido a una tarea por defecto denominada *Idle Task*. Esta tarea por defecto es la encargada de recibir mensajes relativos a nuevas conversaciones.

Una vez que el evento se almacena en la tarea adecuada, el gestor de tareas informa al *Task Manager Listener* de que se ha generado un nuevo evento. Este *Listener* se encarga a su vez de avisar al gestor de tareas de dónde colocar el evento en la cola de tareas pendientes de ejecución. De este modo, el gestor de tareas puede ejecutar las tareas en el orden adecuado.

En la Figura 3.17 se puede ver la interacción entre estos elementos al recibir un mensaje. El usuario debe colocar el código encargado de procesar el mensaje con performativa X en el método *handleX* de la tarea correspondiente.

3.8.1.3. Proyectos de sistemas multiagente que emplean FIPA-OS

FIPA-OS ha sido empleado en numerosos proyectos europeos, entre los que destacan FACTS[35], Cameleon[36] y MAPPA[37]. Actualmente es usado en los proyectos SHUFFLE[38], CRUMPET[39], SMONET[40] y PATTERNS [41]. Asimismo un creciente número de universidades como el Imperial College y EPFL[42] han empleado FIPA-OS.

[35] FIPA Agent Communication Technologies and Services. http://more.btexact.com/projects/facts/

[36] Communication Agents for Mobility Enhancements in a Logical Environment of Open Networks. http://www.comnets.rwth-aachen.de/project/cameleon/cameleon.html

[37] Multimedia Access through Persistent Personal Agents. http://www.sics.se/mappa

[38] An agent based approach to controlling resources in UMTS networks. http://www.aramis-research.ch/e/13441.html

[39] Creation for User-friendly Mobile services Personalised for Tourism. http://www.ist-crumpet.org

[40] Services MObil NETworks. http://www.decoit.de/smonet/

[41] Patterns to adopt knowledge based solutions for software management problems. http://www.esi.es/en/ojects/Patterns/patterns.html

[42] Swiss Federal Institute of Technology in Laussanne

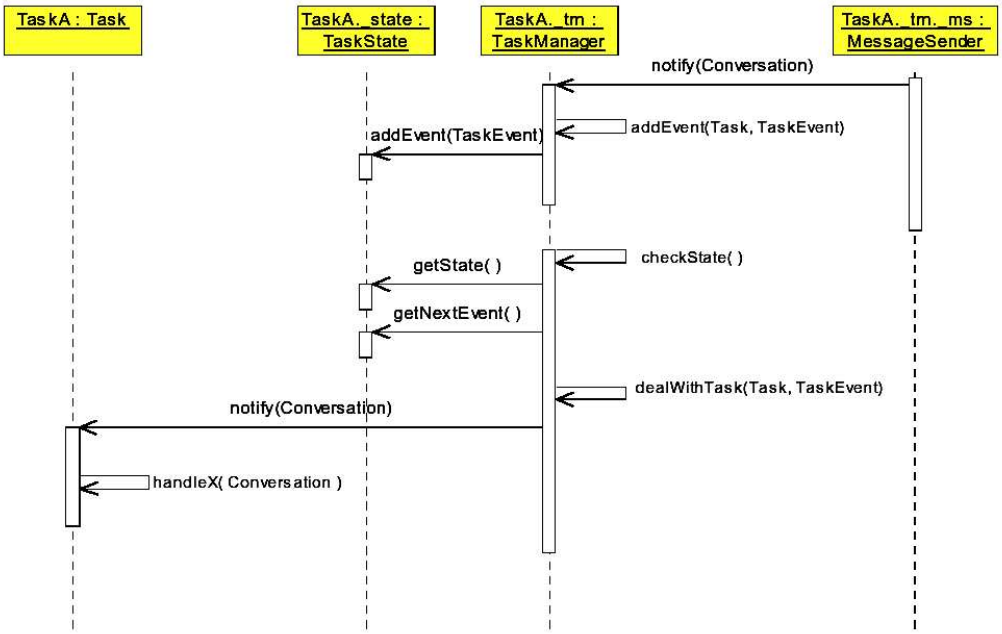

Figura 3.17.: Diagrama de la recepción de un mensaje en FIPA-OS[FIP01i].

3.8.2. JADE

Figura 3.18.: Logotipo de JADE.

3.8.2.1. Breve Descripción

JADE[43] es una herramienta de código libre bajo licencia LGPL (aunque suministra una versión no oficial no testeada), implementada en lenguaje Java que ha sido desarrollada en el CSELT[44] con el objetivo de facilitar la implementación de sistemas multiagente cuyos agentes interactúan según las normas FIPA. JADE proporciona la misma infraestructura de agentes que FIPA-OS[45], aunque incluye el concepto de *AgentContainer* un paradigma propio para la asociación de JVMs, agentes y hosts. Así, una plataforma de agentes implementada en JADE está formada por varios *AgentContainer* [BPR99, BCTR03] (ver la figura 3.19)

[43] Java Agent DEvelopment framework, http://jade.cselt.it
[44] Centro Studi e Laboratori Telecomunicazioni
[45] AMS, DF, ACC, IPMT, interfaz para JESS

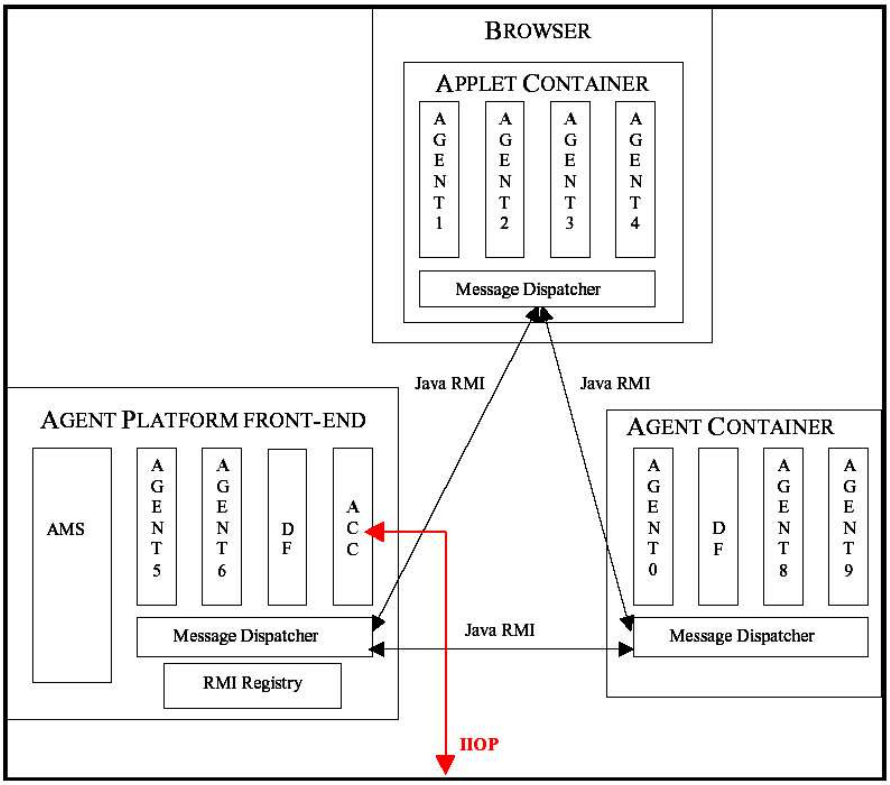

Figura 3.19.: Esquema de arquitectura JADE [BPR99].

Cada *AgentContainer* es un objeto servidor RMI que gestiona de manera local a un conjunto de agentes, controlando el ciclo de vida de estos agentes mediante la creación, suspensión o eliminación de los mismos. Además, maneja los aspectos de comunicación relativos a los mensajes ACL entrantes, distribuyéndolos de acuerdo con el campo destino del mensaje (*:receiver*) y depositándolos en las colas de mensajes de cada agente. Para los mensajes salientes, en cambio, el *AgentContainer* mantiene la suficiente información para determinar la localización del agente receptor y escoge la forma de transporte más adecuada para mandar el mismo. De este modo, las comunicaciones se agilizan ya que entre agentes del mismo *AgentContainer* la comunicación se realiza sin ningún tipo de invocación remota. Las invocaciones RMI quedan reservadas para mensajes entre los agentes de distintos *AgentContainer* de la misma plataforma y las comunicaciones a través de CORBA para los mensajes entre agentes de diferentes plataformas.

Cada AP cuenta con una interfaz gráfica que permite al usuario un mayor control de la plataforma y de los agentes, pudiendo crearlos, suspenderlos o eliminarlos de forma manual. Lo interesante de esta interfaz es que ha sido implementada como si se tratase de un agente más llamado RMA (*Remote Monitoring Agent*), de modo que la interacción de la interfaz con el resto de agentes de la plataforma se realiza mediante mensajes ACL.

Como unidades de depuración JADE destacan el *Agente Sniffer*, que intercepta los mensajes ACL que llegan a un agente y los representa gráficamente en una notación similar a los diagramas de secuencias UML y el *Agente Introspector* que informa sobre el ciclo vital del agente seleccionado (activo, dormido), sus mensajes ACL intercambiados y sus comportamientos implementados que se encuentran ejecutándose en la actualidad (ver Figura 3.20).

3.8.2.2. Modelo de ejecución de un agente JADE

JADE emplea el concepto de *comportamiento* (*Behaviour*) para modelar las tareas que un agente es capaz de llevar a cabo. Los agentes instancian estos comportamientos de acuerdo con sus necesidades y sus capacidades.

Desde el punto de vista de programación concurrente, en el caso de JADE se ha optado por la filosofía de emplear un único *thread* por agente en vez de la opción de un *thread* por tarea como en el caso de FIPA-OS. De este modo se inicializan un menor número de threads en una AP.

Cuando llega un nuevo mensaje, un gestor (implementado por la clase *Agent* y oculto para el programador) implementa una política *round-robin* entre todos los comportamientos disponibles en el agente hasta encontrar un comportamiento dispuesto a procesar el mensaje recibido. Esto se determina mediante una combinación lógica de análisis de las cadenas de caracteres (*string matching*) de los mensajes. En caso de encontrar un comportamiento interesado en procesar el mensaje, se ejecuta su método *action*. Este comportamiento que acepta el mensaje controla el flujo del agente hasta que él mismo lo libera. Por el contrario, en caso de no poder procesar el mensaje recibido, el comportamiento analizado es mandado a dormir.

La clase *Behaviour* es la superclase de la cual derivan dos subclases: *SimpleBehaviour* (que modela comportamientos que son ejecutados sin interrupción) y *ComplexBehaviour* (que modela comportamientos que sí pueden ser interrumpidos

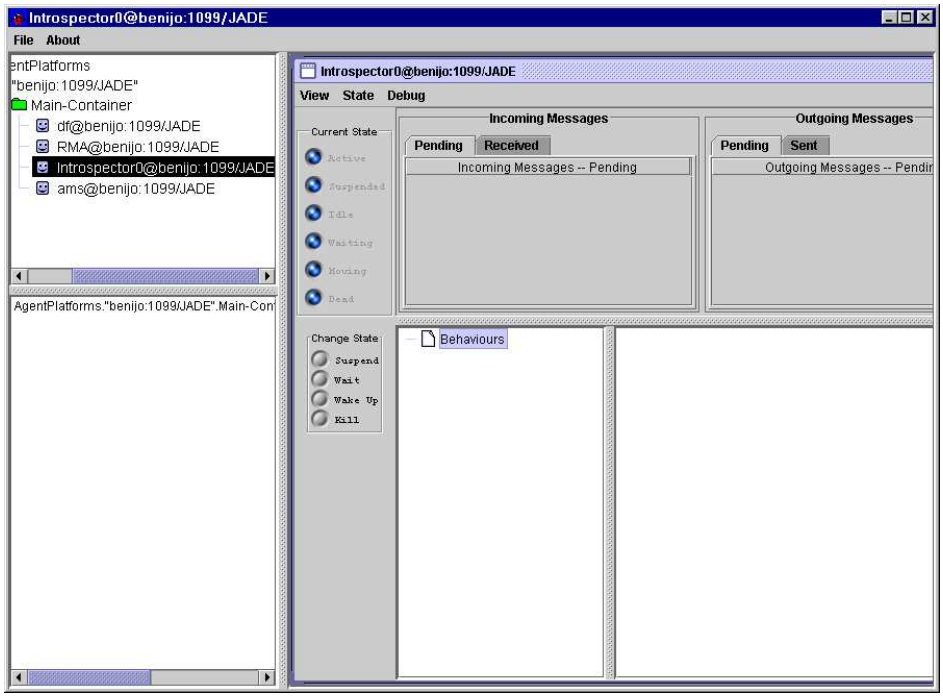

Figura 3.20.: Herramienta *Introspector* de JADE.

en su ejecución). Un *ComplexBehaviour* contiene una serie de comportamientos que pueden interrumpirse al pasar de uno a otro.

Dentro de las subclases de ComplexBehaviour destacan dos: *FSMBehaviour* y *JessBehaviour*. Respecto a *FSMBehaviour*, la política correspondiente es implementada definiendo una máquina de estados de comportamientos. Esta clase tiene la responsabilidad de mantener las relaciones (transiciones) entre los estados y seleccionar el próximo comportamiento a ejecutar. Por otro lado, *JessBehaviour* permite la completa integración con JESS. En este caso JADE proporciona la estructura FIPA, mientras que la herramienta JESS ofrece su potente motor de razonamiento.

3.8.2.3. Proyectos de sistemas multiagente que emplean JADE

JADE es empleado en numerosos proyectos y por varias universidades[46] como el Grupo de Sistemas Multiagente (GruSMA) de la Universidad Rovira y Virgili[47]. Destacaremos el proyecto MONADS[48] de la Universidad de Helsinki (Finlandia), los proyectos del CSELT para las implementaciones de un Agente Planificador de Reuniones y un Agente Organizador de Cines, y finalmente los proyectos CoMMA[49],

[46] http://jade.cselt.it/currently.htm

[47] http://etse.urv.es/reserca/banzai/toni/MAS/

[48] Adaptation Agents for Nomadic Users. http://www.cs.helsinki.fi/research/monads/

[49] Consistente en desarrollar un sistema de información multiagente que soporte la consulta de una memoria colectiva basada en tecnología XML

LEAP[50], KoD[51] y Pellucid[52].

3.8.3. Zeus

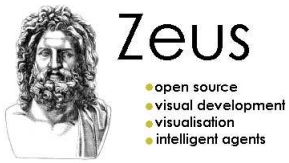

Figura 3.21.: Logotipo de Zeus

3.8.3.1. Breve Descripción

Zeus[53] ha sido desarrollado por British Telecom Labs. para proporcionar una herramienta de construcción de agentes colaborativos genérica a nivel industrial. Para ello adopta una configuración de agentes por capas, de tal modo que los agentes son capaces de seguir un protocolo de comunicación con otros agentes de la comunidad. Los agentes implementados en Zeus tienen la habilidad de planear la secuencia de pasos necesarios para llegar a un objetivo, monitorizar la ejecución de los planes y retroceder cuando sea necesario. Son orientados a metas mediante la representación de máquina de estados.

Zeus fue en principio desarrollado para cumplir con el estándar de comunicación KQML. Sin embargo, ha sido reescrito para adaptarlo al FIPA-ACL, soportando actualmente la especificación FIPA para la gestión de agentes.

Las clases de la distribución Zeus se pueden dividir en tres grupos funcionales (ver figura 3.22):

- Una librería de componentes de agentes. Es un paquete de clases Java que constituye los "bloques de construcción" de los agentes. Comprende aspectos tales como la comunicación, la coordinación o la ontología.
- Una herramienta de visualización de agentes con 5 módulos:
  - un visualizador de sociedades que muestra todos los agentes y sus relaciones (Figura 3.23)
  - un visualizador de la distribución/descomposición de las tareas activas y los estados de ejecución de las sub-tareas
  - un visualizador de los estados internos de los agentes
  - una herramienta de control que permite visualizar y modificar los estados internos de los agentes, por ejemplo redefiniendo el comportamiento del agente

[50] Lightweight Extensible Agent Platform. http://leap.crm-paris.com/
[51] Knowledge on Demand. Your personal e-learning platform. http://kod.iti.gr/
[52] A platform for organisationally mobile public employees. http://www.sadiel.es/EUROPA/pellucid
[53] http://193.113.209.147/projects/agents/zeus/index.htm

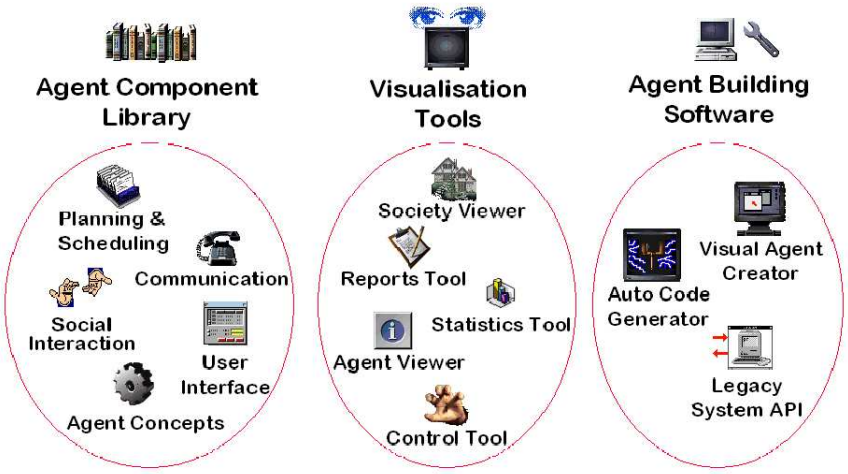

Figura 3.22.: Grupos funcionales de las clases de la distribución Zeus [NNLC].

  - un visualizador de las estadísticas de la sociedad de agentes

- Una herramienta de construcción de agentes a alto nivel. Proporciona una serie de editores que permiten crear agentes, especificando de forma manual sus atributos, ontologías o tareas.

3.8.3.2. Modelo de ejecución de un agente Zeus

Cada agente Zeus consta de tres capas.

- Capa de definición. Representa las habilidades de aprendizaje y razonamiento del agente, sus objetivos, creencias, recursos...
- Capa de organización. Describe las relaciones del agente con otros agentes.
- Capa de coordinación. Describe las técnicas de negociación y coordinación que posee el agente.

La plataforma proporciona un motor de coordinación que ejecuta máquinas de estados, denominadas *grafos.* Un grafo define los *nodos* de procesamiento de una parte del comportamiento del agente y los *arcos* que unen los diferentes nodos.

La parte principal del código del comportamiento del agente se divide en dos nodos llamados *exec* y *continue_exec*. El código de condición previa se define en el método *exec* del arco que determina qué nodo se va a ejecutar a continuación. Cada nodo cuenta con un método *reset* que permite devolver el estado al inmediatamente anterior de entrar en el nodo.

El motor de coordinación gestiona la ejecución de los nodos. Los nodos son almacenados en una cola mientras se espera a que se procese todo el código requerido. Cuando este procesamiento finaliza, los nodos son colocados en otra cola para su ejecución.

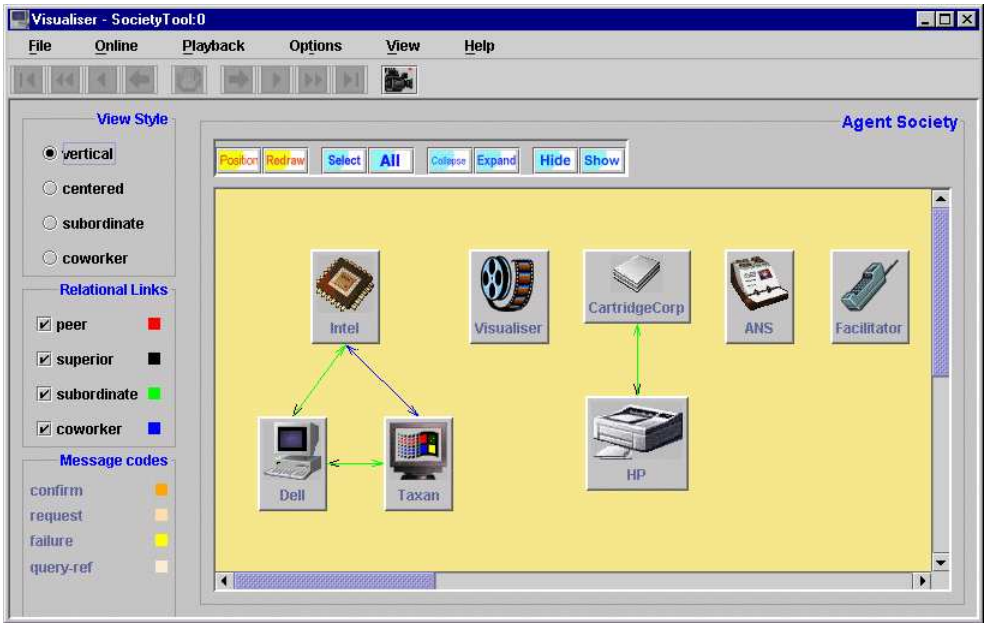

Figura 3.23.: Herramienta visualizadora de sociedades de agentes en Zeus.

Los mensajes ACL son distribuidos por una clase gestora de mensajes. Esta clase permite registrar las reglas para determinar qué objeto debe procesar un nuevo mensaje. Una vez que se dispara una determinada regla, se ejecuta el método *registered* del objeto correspondiente. Tras la ejecución de este método, el gestor de mensajes chequea la siguiente regla registrada para ver si también es satisfecha por el mensaje recibido. Este proceso continúa secuencialmente hasta que se da una oportunidad a todas las reglas activas para procesar el mensaje [NNL99, NNLC, DLL3].

3.8.3.3. Proyectos de sistemas multiagente que emplean Zeus

Actualmente Zeus se emplea por parte, entre otros, del Intelligent Business System Group (para flujo de trabajo basado en agentes), el Electronic Commerce Group (para la implementación de mercados virtuales basados en agentes) y BT North America (para gestión de redes basada en agentes) [ND02].

3.8.4. JAT*Lite*

3.8.4.1. Breve Descripción

JAT*Lite* (Java Agent Template, Lite) deriva de JAT, la primera herramienta de desarrollo para agentes escrita íntegramente en Java. El motivo de desarrollar JAT*Lite* fue que JAT incorporaba teorías de investigación específicas que no eran necesarias en una infraestructura general de agentes. Además no soportaba la comunicación entre los agentes a través de applets Java.

JAT*Lite* es un paquete de clases y programas Java desarrollado por la Universidad de Standford que permiten a sus usuarios crear rápidamente nuevos sistemas de agentes

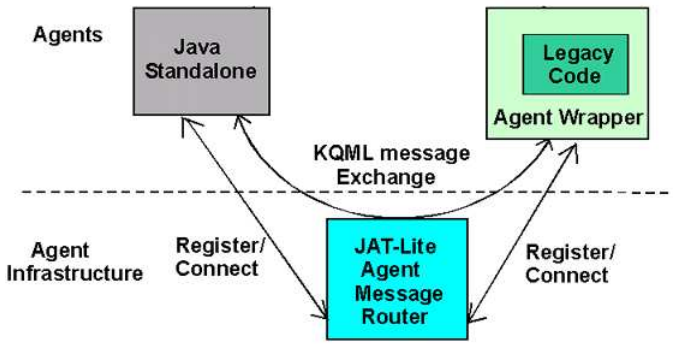

Figura 3.24.: Esquema de comunicaciones JAT*Lite* [JPC00].

software que se comunican a través de Internet, buscando de este modo implementar un sistema distribuido. Los agentes JAT*Lite* pueden ser creados desde cero o "envueltos" (*wrapped*) con software capaz de generar y recibir mensajes. JAT*Lite* proporciona una infraestructura robusta en la que los agentes se registran en un *Agent Message Router* (AMR) empleando un nombre y una contraseña para poder intercambiar mensajes y transferir ficheros con otros agentes a través de Internet (incluyendo applets Java), soportando los protocolos TCP/IP, SMTP y FTP. Este esquema se refleja en la Figura 3.24.

JAT*Lite* recomienda el KQML como estándar de mensaje aunque se pueden enviar mensajes con cualquier formato. No obstante, y dado de la fuerza que han adquirido las especificaciones FIPA, se ha implementado el paquete JAT*Lite*-ACL[54] que permite el intercambio de mensajes en FIPA-ACL, aunque no cumple con el resto de especificaciones FIPA[55].

3.8.4.2. Arquitectura JAT*Lite*

La arquitectura JAT*Lite* está organizada en capas en un orden de especialización creciente, estando cada capa basada en la anterior. De este modo, los desarrolladores de agentes solamente deben quedarse con aquellas capas que les interese para crear sus agentes. En la Figura 3.25 se observa esta jerarquía de capas.

- La capa *Abstract* proporciona una colección de clases abstractas necesarias para la implementación JAT*Lite*.
- La capa *Base* proporciona comunicación básica basada en TCP/IP y en la capa *Abstract*.
- La capa *KQML* proporciona infraestructura para el almacenamiento y tratamiento de mensajes KQML.

[54] http://liawww.epfl.ch/~calisti/ACL-LITE
[55] Para ello emplea el paquete de comunicaciones integrado en FIPA-OS.

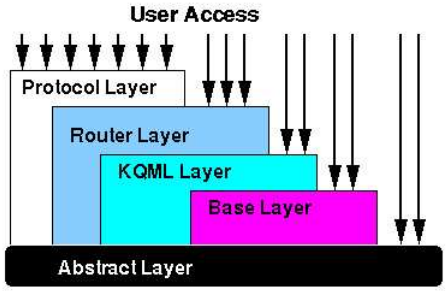

Figura 3.25.: Jerarquía de capas JAT*Lite* [JPC00].

- La capa *Router* proporciona los aspectos relacionados con el registro y direccionamiento de mensajes de los agentes. Por ejemplo si un agente si se desconecta del sistema, el AMR almacena los mensajes destinados a él hasta que el agente se vuelva a conectar.

- Finalmente la capa *Protocol* proporciona la infraestructura para la comunicación a través de los protocolos de Internet, como FTP o SMTP [JPC00].

Por tanto, un desarrollador que quiera utilizar las comunicaciones TCP/IP pero no quiera emplear KQML solamente debería emplear las capas *Abstract* y *Base*. El no colocar la capa KQML en el nivel más alto de la jerarquía constituye un fallo de diseño, puesto que dificulta el empleo de otros estándares de comunicación inter-agentes como el FIPA-ACL. Así, en este caso, habría que reformar no sólo la capa KQML (como sería lógico en un principio) sino las capas superiores, es decir la *Router* y la *Protocol*. Esta defecto se ha intentado mitigar en el paquete JAT*Lite*-ACL.

3.8.4.3. Proyectos de sistemas multiagente que emplean JAT*Lite*

JAT*Lite* ha sido la base de los proyectos *NextLink*[56] y *ProcessLink*[57].

3.8.5. Jackal

Figura 3.26.: Logotipo/mascota de Jackal.

[56] http://www-cdr.stanford.edu/NextLink/Expert.html
[57] http://www-cdr.stanford.edu/ProcessLink/

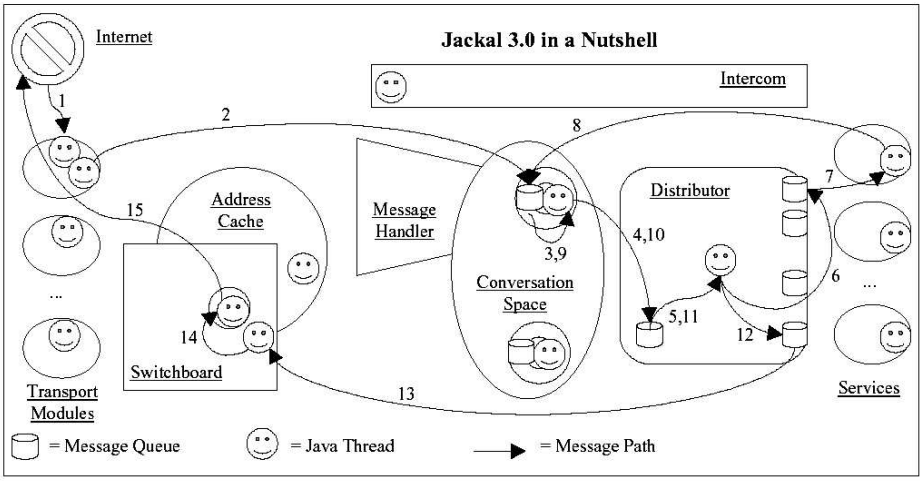

Figura 3.27.: Arquitectura Jackal y Flujo de Mensajes [CFL+98].

3.8.5.1. Breve Descripción

El motivo principal del estudio de esta herramienta es que ha sido desarrollada, entre otros, por Finin y Labrou, partícipes en las especificaciones del lenguaje KQML [FWW+93, LF97]. Jackal es una herramienta basada en Java desarrollada por el CIIMPLEX[58]para la comunicación a través de KQML, presentando una interfaz sencilla. Consiste en unas setenta clases que ocultan los detalles de la implementación al usuario.

La Figura 3.27 representa los principales componentes de la herramienta y el proceso de paso de mensajes.

Estos elementos principales son:

- Intercom. Es el puente entre la aplicación donde corren los agentes y Jackal. Controla la puesta en marcha y desactivación de Jackal, proporciona a la aplicación acceso a los métodos internos, almacena algunas estructuras de datos comunes y juega un papel de supervisor en la infraestructura de las comunicaciones.

- Interfaz de Transporte (*Transport Interface*). Jackal ejecuta un módulo de transporte para cada protocolo empleado en la comunicación. Es el responsable de recibir los mensajes y de transmitir según un protocolo dado.

- Tratante de Mensajes (*Message Handler*). Los mensajes recibidos por el *Switchboard* (una especie de pizarra donde se apuntan los mensajes recibidos a la espera de que alguien los recoja) deben ser llevados al lugar apropiado en el Espacio de Conversaciones (*Conversation Space*). Es el Tratante de Mensajes el encargado de esta tarea. Los mensajes son asociados con los *threads* actuales basándose en su identificador (campo `reply with`). Si no se puede asociar el mensaje a ninguna conversación existente, se inicia una nueva.

[58]Consortium for Intelligent Integrated Manufacturing Planning-Execution

- Espacio de Conversaciones (*Conversation Space*). Consiste en una colección de las conversaciones iniciadas por el Tratante de Mensajes. Estas conversaciones ejecutan intérpretes individuales de protocolo. De este modo y con el Distribuidor, los mensajes pueden ser direccionados directamente a los *threads* correspondientes.

- Distribuidor (*Distributor*). Es una pizarra al estilo Linda [CG89], que sirve para casar los mensajes con las conversaciones.

- Servicios (*Services*). Es cualquier thread, ya sea de Jackal o perteneciente a cualquier agente [CFL+98].

3.8.6. Grasshopper

Figura 3.28.: Logotipo de Grasshopper.

3.8.6.1. Breve Descripción

Grasshopper[59] es un marco de agentes freeware (por tanto, no proporciona código fuente) desarrollado en Java por IKV++ Technologies AG[60]. Es la primera plataforma de agentes que cumple con las especificaciones OMG MASIF (ver sección 3.5.3) y FIPA. En principio fue diseñado como una plataforma de agentes móviles que cumpliese el estándar MASIF, pero ha sido mejorado para incluir los mandatos FIPA mediante un paquete adicional, dotándole de este modo de mayor flexibilidad. Por tanto, integra el paradigma cliente/servidor y la tecnología de los agentes móviles.

Los agentes implementados en Grasshopper se mueven en un Entorno de Agentes Distribuido (*Distributed Agent Environment*, DAE). Este entorno se compone de regiones (*regions*), lugares (*places*), conjuntos de agentes (*agencies*) y diferentes tipos de agentes. Esta configuración puede visualizarse en la Figura 3.29.

Un grupo de agentes consta de dos partes: el núcleo (*core agency*) y al menos un lugar. El núcleo del grupo de agentes representa la mínima funcionalidad necesaria en orden a soportar la ejecución de los agentes. Para ello soporta los siguientes servicios:

- Servicio de Comunicación. Este servicio es el responsable de todas las interacciones remotas que tienen lugar entre los componentes distribuidos de Grasshopper, tales como comunicación inter-agente transparente a la localización, el transporte de agentes y la localización de agentes a través del registro de la región. Todas las interacciones se llevan a cabo a través de CORBA IIOP, Java RMI o sockets.

[59] http://213.160.69.23/grasshopper-website/
[60] http://ikv.de

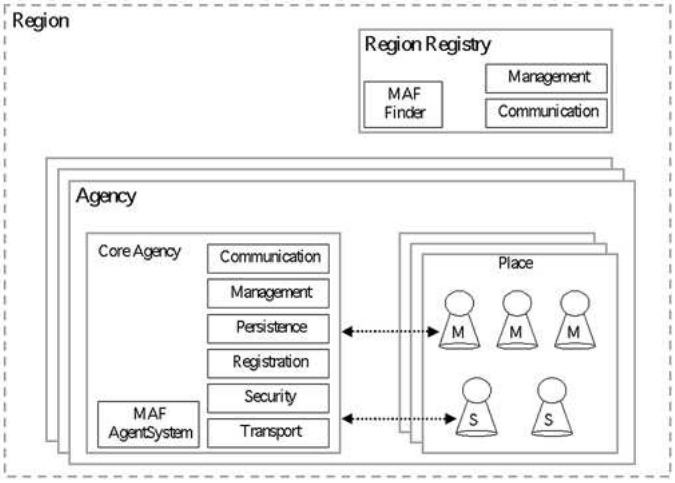

Figura 3.29.: Entorno de Agentes Distribuido (DAE) en Grasshopper [BM99].

- Servicio de registro. Cada agente debe ser capaz de conocer todos los agentes y lugares registrados en un momento determinado. Para ello, el servicio de registro de cada grupo de agentes se conecta al registro de la región.

- Servicio de gestión. Permite la monitorización y control por parte del usuario de los agentes y lugares de un grupo de agentes.

- Servicio de seguridad. Grasshopper muestra dos mecanismos de seguridad, uno externo y otro interno.

  - El mecanismo de seguridad externo protege las interacciones remotas entre los diferentes elementos del DAE mediante el empleo de SSL[61].

  - El mecanismo de seguridad interno salvaguarda los recursos del grupo de agentes de un posible acceso no autorizado, así como a los agentes de otros agentes. Para ello se basa en los propios mecanismos de seguridad de Java.

- Servicio de persistencia. Permite el almacenamiento tanto de agentes como de lugares de modo persistente. De este modo, es posible recuperar un agente o lugar cuando es necesario, por ejemplo tras una caída del sistema.

El paquete de distribución Grasshopper proporciona al usuario una serie de herramientas de control y visualización de los elementos de la sociedad de agentes (ver Figura 3.30), como por ejemplo un monitor de los threads que se están ejecutando.

[61] Secure Sockets Layer.

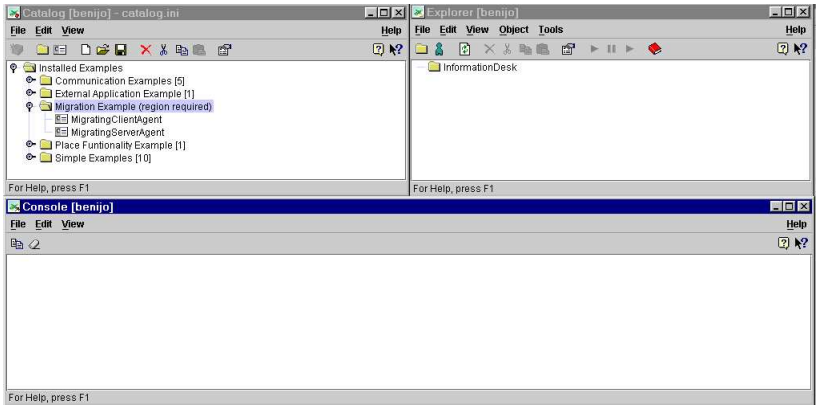

Figura 3.30.: Herramientas de visualización y control Grasshopper.

3.8.6.2. Proyectos de sistemas multiagente que emplean Grasshopper

Entre los proyectos que han empleado Grasshopper en su desarrollo destacan ANIMA[62], Cameleon, FACTS, MARINE[63], MARINER[64] y MIAMI[65] [ND02].

3.8.7. Otras herramientas MAS

No queremos terminar esta sección de análisis de herramientas MAS sin citar, aunque de forma más breve, otros marcos de trabajo que también consideramos interesantes.

- RETSINA[66] (Reusable Environment for Task Structured Intelligent Network Agents), desarrollado por el Software Agents Group de la Carnegie Mellon University. Más que de una herramienta MAS deberíamos hablar de una propuesta de organización de agentes. RETSINA tiene cuatro tipos de agentes básicos reusables para la comunicación, planificación, organización y monitorización de la ejecución de tareas y peticiones por parte de otros agentes. La idea subyacente en el proyecto es que los agentes del sistema deberían formar una comunidad de agentes en igualdad y con relaciones de igual a igual. Por tanto, RETSINA, a diferencia de FIPA, no implementa ningún control central en el MAS. De este modo, se consiguen conjuntar en el mismo sistema agentes heterogéneos implementados en Java, C, C++, Python, LISP y Pearl, y que se comunican en KQML [SPVG01].

- MAST[67] (Multi-Agent Systems Tool), anteriormente llamada MIX. Desarrollada en C++ por el Grupo de Sistemas Inteligentes de la Universidad

[62] Architecture Neutral Intelligent Mobile Agents. http://anima.ibermatica.com/anima
[63] Mobile Agent Environment for Intelligent Networks. http://www.itatel.it/marine/marine.htm
[64] Multi-Agent Architecture for Distributed-IN Load Control and Overload Protection. http://teltec.dcu.ie/mariner
[65] Mobile Intelligent Agents for Managing the Information Infrastructure. http://fokus.gmd.de/research/cc/ima/miami
[66] http://www-2.cs.cmu.edu/~softagents/retsina/retsina.html
[67] http://www.gsi.dit.upm.es/~mast/mast.html

Politécnica de Madrid. Es un marco de propósito general para la cooperación de múltiple agentes heterogéneos. La arquitectura MAST consta de dos entidades básicas: los agentes y la red a través de la cual interactúan [IGV96].

- dMARS (distributed Multi-Agent Reasoning System), desarrollado por el Australian Artificial Intelligence Institute (AAII)[68]. Se basa en el modelo de agente BDI (ver sección 3.1.3) y proporciona un entorno para el desarrollo de sistemas multiagente distribuidos. El mecanismo de razonamiento primario es la planificación. Como hay menos énfasis en la coordinación del sistema, no hay motor de coordinación alguno y no está claro cómo se pueden compartir protocolos de coordinación entre agentes dMARS. Los desarrolladores dMARS se ven obligados a trabajar en un nivel de abstracción bajo [dKLW98, NNLC].

- JAFMAS[69] (Java-based Agent Framework for Multi-Agent Systems), desarrollada por la División de Investigación y Estudios de la Universidad de Cincinnati, proporciona una metodología para el desarrollo de MAS basados en servicios de comunicación, de coordinación y lingüísticos. La coordinación es soportada a través del uso de clases de conversación que los agentes emplean para gestionar sus interacciones con otros agentes. Estas clases implementan modelos de autómatas basados en reglas. Emplea KQML como estándar de comunicación y carece de interfaz gráfica propia, aunque existe un entorno complementario llamado *Jive* [CB98, Fon01].

- Aglets[70]. Herramientas para el desarrollo en Java de agentes móviles en Internet. Es de software libre por parte del Tokyo Research Laboratory de IBM pero actualmente esta empresa no sigue desarrollándolo. Los *Aglets* son objetos Java que se pueden mover de un host a otro a través de Internet. Es decir, un *aglet* que se esté ejecutando en un host puede de repente detener su ejecución, trasladarse a otro host y continuar la ejecución allí. Actualmente se emplea en un sistema de mercado virtual de billetes de avión en Japón llamado TabiCan[71] [ND02].

- JKQML[72] es un marco de trabajo en proceso de desarrollo por IBM para construir agentes software que se comunican en KQML a través de Internet. JKQML se basa en la propuesta de especificación KQML [FWW+93, LF97]. Soporta los protocolos KTP (*KQML Transfer Protocol*, un transporte basado en sockets para un mensaje KQML representado en ASCII), ATP (*Agent Transfer Protocol*, un protocolo para mensajes KQML transferidos por un agente móvil que es implementado por Aglets) y OTP (*Object Transfer Protocol*, un protocolo de transferencia para objetos Java contenidos en un mensaje KQML).

- JACK[73] es una herramienta comercial orientada a agentes desarrollada por la compañía australiana *Agent Oriented Software Pty. Ltd*. Actualmente puede usarse una versión de evaluación de 60 días. Es un entorno para construir, correr

[68] http://www.aaii.com.au/
[69] http://www.ececs.uc.edu/~abaker/JAFMAS/JAFMAS.html
[70] http://www.tri.ibm.com/aglets
[71] http://www.tabican.ne.jp/
[72] http://www.alphaworks.ibm.com/formula/jkqml
[73] http://www.agent-software.com.au/shared/products/index.html

e integrar sistemas multiagente basados en Java usando una implementación basada en componentes. Se basa en el modelo BDI[74] de dMARS. Básicamente proporciona una extensión del lenguaje Java llamada JAL (Jack Agent Language) para aquellos aspectos específicos de los agentes. Los programas son traducidos a un Java puro mediante un compilador, antes de poder ser interpretados por la JVM [Mai03].

- AgentBuilder[75]. Conjunto de herramientas comerciales integradas para la construcción de agentes software inteligentes Fue desarrollada en Java por *Reticular Systems Inc*. Se basa en los modelos Agent0 y BDI[76] Placa. El diseño de los agentes se basa en la descomposición del problema a tratar en funciones que los agentes pueden llevar a cabo. A continuación se identifican los agentes, se definen sus características y finalmente se diseñan los protocolos de interacción. Por tanto, el desarrollo de los agentes consiste en definir el comportamiento del agente, sus reglas, restricciones, capacidades... Emplea KQML como lenguaje de comunicación.

- KAoS intenta suministrar un arquitectura de agentes abierta. Esta arquitectura describe las implementaciones de los agentes y las interacciones dinámicas entre ellos a través de la comunicación. Uno de los aspectos característicos de esta arquitectura es el relativo a los mecanismos de seguridad. De modo que dos agentes podrían incluso negociar acerca de los modelos de seguridad a emplear en una negociación de servicios. La herramienta de diseño de agentes KAoS soporta representaciones de alto nivel para la descripción de conversaciones y planes [BDBW97, NNLC].

- JAFIMA (Java Framework for Intelligent and Mobile Agents) es un desarrollo por capas para el diseño de sistemas de agentes. Está destinado a aquellos desarrolladores que quieren implementar sistemas multiagente partiendo de cero, ya que proporciona una metodología completa para el desarrollo pero ningún tipo de componente prefabricado a nivel de agentes [KKSP00].

- Agentes Voyager. Es una herramienta implementada en Java robusta para el desarrollo de agentes móviles, extensible, escalable basada en la tecnología ORB. Su arquitectura modular en capas soporta transparentemente múltiples protocolos de paso de mensajes (IIOP, RMI, DCOM), de comunicación (TCP/IP, SSL, SOCKS)... incluyendo las ventajas, entre otras, de agregación dinámica, movilidad de objetos, consola de gestión y carga de clases remotas. Actualmente su página web ofrece una versión de evaluación de 30 días.

[74] ver sección 3.1.3
[75] http://www.agentbuilder.com
[76] ver sección 3.1.3.

4. Sistema MASplan

En este capítulo se presenta una aplicación de MAS, denominada MASplan, para la planificación de agendas en el escenario de un grupo de investigación universitario, concretamente el Grupo de Computadoras y Control (CyC) de la Universidad de La Laguna. Este sistema se encarga de determinar el mejor horario para la organización de reuniones y un mecanismo de reserva de los recursos comunes dentro del grupo CyC. Se ha escogido este tipo de problema por ser un campo clásico en el campo de las aplicaciones MAS así como por el gran número y variedad de agentes involucrados y tareas asumidas. Por tanto, permite establecer comparaciones con otros sistemas existentes.

Empezaremos el capítulo presentando una serie de conceptos necesarios. En primer lugar, abordaremos el tema de los mecanismos de negociación entre agentes: votaciones, subastas, regateo, redes de contrato, coaliciones y argumentación. Dedicaremos una especial atención a las tácticas implementadas en MASplan, concretamente los mecanismos de votación y el regateo. Para completar la base teórica necesaria, se describe la técnica de los árboles de identificación para la generación de reglas a partir de un conjunto de datos observados. Esta herramienta permite dotar de cierta inteligencia a los agentes en el MAS. Centrándonos en el problema concreto, en primer lugar describiremos tres antecedentes significativos en este tipo de aplicaciones: el *Visitorbot* de Kautz et al. en el *AI Principles Research Department* en los *AT&T Bell Laboratories*, el propuesto por Sandip Sen en la Universidad de Tulsa y el sistema denominado *Electric Elves* de la División de Sistemas Inteligentes de la USC/ISI. A partir de ese punto, el capítulo se dedica a la implementación del MAS. Para la implementación, se ha empleado principalmente una herramienta de construcción basada en el estándar FIPA (FIPA-OS), una ontología desarrollada en un lenguaje de marcas de alta riqueza semántica (DAML+OIL) y un *parser* para dicho lenguaje (*Jena*).

4.1. Consideraciones Previas

4.1.1. Negociación en sistemas multiagente

El hecho de que en los sistemas multiagentes cada agente carezca de control directo sobre el resto de los agentes provoca que se introduzca el concepto de negociación, el cual puede definirse como [MSJ98]

> un proceso mediante el cual se toma una decisión común por dos o más partes. Las partes involucradas primero enuncian demandas contradictorias y entonces se desplazan hacia un acuerdo mediante un proceso de concesiones o de búsqueda de nuevas alternativas.

El proceso de negociación [Lee99] dentro de esta definición puede dar lugar a multitud de variantes, siempre implicando la resolución distribuida de posibles conflictos. En la coordinación pueden estar involucrados principalmente dos tipos de agentes [Flo03]:

- Agentes motivados colectivamente. Este tipo de agentes tiene metas comunes y por lo tanto cooperan para llevar a cabo dichas metas.
- Agentes auto-interesados. Este tipo de agente posee objetivos propios, siendo necesaria la negociación para un comportamiento coherente del sistema de agentes. La existencia de este tipo de agentes lleva implícita una búsqueda distribuida dentro del espacio de posibles soluciones.

La negociación entre agentes cobra cada vez más importancia principalmente por dos factores:

- La aparición de estándares de infraestructura de comunicación (Java, FIPA, KQML...) mediante los cuales agentes diseñados de forma diferente por programadores diferentes en entidades diferentes pueden interactuar de forma segura en un entorno abierto en tiempo real.
- El auge de las aplicaciones que dan soporte a la negociación a nivel de toma de decisiones.

En la Resolución Distribuida de Problemas (RDP) el diseñador del sistema impone un protocolo de interacción y una estrategia para cada integrante. Por otro lado, en los sistemas multiagentes, los agentes son provistos con un protocolo de interacción, pero cada agente escoge su propia estrategia [BdlCT97, SL97].

Las características deseables para un negociación entre los agentes de un sistema son [HS99]:

Eficiencia Los agentes no deberían desperdiciar recursos mientras llegan a un acuerdo.

Estabilidad Ningún agente debería mostrar iniciativa para desviarse de una estrategia acordada.

Simplicidad El mecanismo de negociación debería imponer a los agentes bajas demandas computacionales y de ancho de banda.

Distribución El sistema no debería necesitar de un mecanismo de decisión central. Se prefieren los sistemas distribuidos por su mayor robustez.

Simetría El mecanismo no debería estar predispuesto contra otro agente por razones arbitrarias o inapropiadas.

Se pueden distinguir tres partes fundamentales dentro de la negociación:

- Un lenguaje de comunicación.
- Un protocolo de negociación.

- Un proceso de decisión en el cual el agente decide sobre su posición, qué concesiones realizar...

Los aspectos relativos al lenguaje de negociación han sido ya tratados en capítulos anteriores de este trabajo, por lo que estudiaremos en este capítulo los otros dos puntos.

Definiremos *la negociación orientada a servicios* como aquélla en la que un agente (el cliente) solicita un servicio a otro agente (el servidor) para que éste lo lleve a cabo [FSJ98, MSJ98, FNJS00]. El proceso de negociación implica determinar el contrato entre estos agentes que fija bajo qué términos y en qué condiciones se realiza el servicio. Esto puede darse de una forma iterativa a través de ofertas sucesivas.

Entre las técnicas de negociación más empleadas, en orden creciente de complejidad, se encuentran:

- votaciones
- subastas
- regateo
- redes de contrato
- coaliciones
- argumentación

En los próximos apartados se analizan estas técnicas de negociación, en especial las empleadas en la implementación del sistema multiagente presentado en este capítulo.

4.1.2. Mecanismos de votación

Muchos de los mecanismos de negociación precisan que los agentes del sistema se comuniquen unos con otros directamente, por lo que son apropiados para solamente un número pequeño de agentes. Para un número mayor o desconocido de agentes, se necesita otro tipo de mecanismo de coordinación, como por ejemplo el de votación.

Un mecanismo de votación consiste en un método para la obtención de una elección resultado de las preferencias manifestadas por un conjunto de votantes. Este mecanismo es simple, equitativo y distribuido, pero necesita cantidades importantes de comunicación y organización, por lo que solamente deberían ser empleados para un número pequeño de cuestiones.

En una votación, todos los agentes suministran entrada al mecanismo de votación y el resultado es una solución aceptada por todos los agentes. Si se dispone de un número A de agentes y R posibles resultados, cada agente i tiene una relación de preferencia (orden de la preferencia del agente sobre cada resultado posible) $>_i$ sobre R. La votación consiste en obtener una $>_*$ que represente la preferencia social y que satisfaga una serie de criterios.

En el caso de la existencia de dos únicas opciones posibles, basta con elegir la que reciba un mayor número de votos, quedando reservados los problemas a situaciones donde se produce un empate. Por el contrario, en el caso de más de dos opciones, aparecen numerosos problemas para encontrar un método adecuado de votación. En la literatura al respecto [Str80], se suelen citar los siguientes criterios que debería cumplir un mecanismo de votación:

Criterio de Pareto Si cada votante prefiere una opción *a* a otra opción *b*, entonces *b* no debería ser declarada como la opción ganadora en el proceso de votación. Dicho de otro modo, una solución *a* es Pareto-eficiente si no existe otra solución *b* tal que haya al menos un agente que prefiera *b* a *a* y ninguno de los otros agentes se encuentra peor en *b* que en *a* [Str80, Flo03].

Criterio de ganador de Condorcet Si existe una opción *a* que ganaría todas las votaciones uno a uno con el resto de las opciones existentes, entonces el mecanismo de votación debería declarar ganadora del proceso a la opción *a*, recibiendo ésta el nombre de *ganadora de Condorcet*[1]. Obsérvese que no siempre existirá dicha opción *a*.

Criterio de perdedor de Condorcet Si existe una opción *a* que perdería todas las votaciones uno a uno con resto de las opciones existentes, entonces el mecanismo de votación no debería declarar ganadora del proceso a la opción *a*, la cual recibe el nombre de *perdedora de Condorcet*.

Criterio de monotonía Si una opción *a* es declarada ganadora según un mecanismo de votación, y uno o más votantes cambian sus preferencias a favor de la opción *a*, sin que se produzcan otros cambios, entonces la opción *a* debería seguir ganando la votación.

Ausencia de dictadores No debe haber un agente *i* en el sistema tal que una relación de preferencia $>_*$ sea la misma que $>_i$, independientemente de los otros agentes.

Existen varios métodos de votación, que se suelen clasificar en *preferenciales* y *no preferenciales*, según se pida o no al votante que manifieste su orden de preferencia entre las opciones dadas. Los métodos de votación más significativos se desglosan en los siguientes apartados.

4.1.2.1. Métodos preferenciales de votación

Método de pluralidad

Cada votante manifiesta su preferencia por una opción, siendo elegida ganadora la que obtenga un mayor número de votos. Este método satisface el Criterio de Pareto, pero no necesariamente los de ganador ni perdedor de Condorcet. Este método puede ser visto como una votación en que cada votante manifiesta el orden de preferencia entre las diversas opciones y donde se contabilizan únicamente el número de veces que cada opción aparece al principio de la lista.

En el ejemplo de la Tabla 4.1, la opción ganadora es la *c*, ya que aparece más veces en primer lugar (4 votos). Sin embargo, es la perdedora de Condorcet, ya que la mayoría de los votantes prefieren tanto la opción *a* como la *b* a la opción *c* (en ambos casos con un resultado de 5 a 4).

[1] debido al Marqués de Condorcet, matemático francés del siglo XVIII

Cuadro 4.1.: Ejemplo de método de pluralidad con 9 votantes.

	3 votantes	2 votantes	4 votantes
Primera opción	a	b	c
Segunda opción	b	a	b
Tercera opción	c	c	a

Cuadro 4.2.: Ejemplo de método de pluralidad con segunda vuelta.

	6 votantes	5 votantes	4 votantes	2 votantes
Primera opción	a	c	b	b
Segunda opción	b	a	c	a
Tercera opción	c	b	a	c

Método de pluralidad con segunda vuelta

Similar al de pluralidad, pero a continuación se escogen las dos opciones con un mayor número de votos. La opción ganadora es aquélla que gana la votación uno a uno entre estas dos opciones. En este caso se logra cumplir el Criterio de ganador de Condorcet, incumpliendo el de perdedor. En general, los mecanismos de votación con una segunda vuelta no cumplen el Criterio de Monotonía.

En el ejemplo de la Tabla 4.2, las dos opciones con un mayor número de primeros puestos son las opciones *a* y *b* (cada una con 6 votos). La opción *a* es declarada como ganadora, puesto que derrota a *b* con un resultado de 11 a 6 (11 votantes prefieren a la opción *a*). Se puede comprobar que si los dos votantes de la última cambian sus preferencias por *a*, no se cumpliría el Criterio de Monotonía, ya que la opción *c* sería la escogida (9 votos por 8 votos logrados por *a*).

Método de pluralidad con eliminación[2]

Se realiza una votación con el método de pluralidad, eliminando una opción no deseada. El proceso se repite hasta que queda únicamente una opción, declarada como ganadora. La opción a eliminar puede ser la que tenga el menor número de primeros puestos en la clasificación (método de Hare) o el de más últimos puestos (método de Coombs). Ninguno de los métodos satisface los criterios de ganador de Condorcet ni el de monotonía.

Método de votación secuencial por parejas

Se vota entre una pareja de opciones, eliminando la perdedora. La opción ganadora se somete a votación con otra alternativa, repitiéndose el proceso hasta que solamente

[2]Método que se emplea por el Comité Olímpico Internacional para la elección de la ciudad sede de los JJ.OO. Sin embargo, en este caso concreto el método se ve contaminado por el conocimiento por parte de los votantes del resultado de las anteriores votaciones.

Cuadro 4.3.: Ejemplo de método de votación secuencial por parejas.

	1 votante	1 votante	1 votante
Primera opción	a	c	b
Segunda opción	b	a	d
Tercera opción	d	b	c
Cuarta opción	c	d	a

Cuadro 4.4.: Método de recuento de Borda.

	3 votos	2 votos	Puntos
Primera opción	a	b	2
Segunda opción	b	c	1
Tercera opción	c	a	0

queda una opción. Presenta el problema de que, en ausencia de una opción ganadora de Condorcet[3], el sistema se vuelve dependiente del orden de las votaciones. Así en el ejemplo de la Tabla 4.3, se puede comprobar que según el orden en que se realizan las votaciones, cualquiera de las cuatro opciones puede resultar ganadora.

Método del recuento de Borda

Para escoger entre *N* alternativas, se asignan *N-1* puntos para la opción que ocupa el primer puesto en las preferencias de cada votante, *N-2* puntos para cada segundo puesto, y así hasta la opción en último lugar, la cual no recibe ningún punto. Se suman todos los puntos de cada votante y la ganadora es la alternativa con más puntuación. Este sistema de votación satisface los criterios de Pareto, perdedor de Condorcet y monotonía. Sin embargo, no cumple el de ganador de Condorcet, como se comprueba en el ejemplo de la Tabla 4.3. No obstante, la probabilidad de quedarse con una opción ganadora de Condorcet aumenta con el número de votantes. En el ejemplo citado, el recuento de Borda resulta 6 puntos para la opción *a* (3 votos · 2 puntos de primera opción más 2 votos · 0 puntos de última opción), 7 puntos para *b* y 2 puntos para *c*. La opción *b* es la escogida como ganadora, aunque *a* es la ganadora de Condorcet.

Método de Black[4]

Si existe una opción ganadora de Condorcet, elegirla. En caso contrario tomar la opción ganadora según el método de recuento de Borda. Este método, obviamente cumple con el criterio de ganador de Condorcet, sin embargo no lo hace con un enunciado más general dado por John Smith en 1973 de este principio.

[3] Como se puede comprobar, una opción ganadora de Condorcet resultaría la escogida según este método de votación.

[4] propuesto por Duncan Black en 1958

Cuadro 4.5.: Ejemplo de método de Copeland.

	1 votante	4 votantes	1 votantes	3 votantes
Primera opción	a	c	e	e
Segunda opción	b	d	a	a
Tercera opción	c	b	d	b
Cuarta opción	d	e	b	d
Quinta opción	e	a	c	c

a	b	c	d	e
+2	0	0	0	-2

Forma generalizada de Smith del criterio de ganador de Condorcet Si el conjunto de opciones puede ser dividido en dos conjuntos *A* y *B*, de modo que cada opción incluida en *A* ganase a cada opción incluida en *B*, entonces el método de votación no debería escoger como ganadora a una opción incluida en *B*.

Método de Nanson[5]

Este método consiste en la eliminación recursiva de la opción con una menor puntuación en un recuento de Borda, hasta que queda una única opción, o en su defecto un conjunto de opciones con igual puntuación. En este caso se cumple los criterios de Condorcet, incluso el generalizado de Smith, pero a semejanza de otros criterios que incluyen una segunda vuelta, no cumple con el de monotonía.

Método de Copeland[6]

Este método realiza una votación entre cada pareja de opciones. Se asigna 1 punto a cada alternativa ganadora y -1 a la opción perdedora. La opción ganadora es que obtiene una mayor puntuación. Este método cumple con todos los criterios anteriores, aunque en algunas ocasiones genera una solución en apariencia incorrecta desde el punto de vista intuitivo. Como ejemplo considérese la tabla de preferencias mostrada en la Tabla 4.5, que da como ganadora a la opción *a*, cuando 8 de los 9 votantes prefieren la opción *e* a la opción *a*.

Por tanto, se necesitan introducir más criterios a la hora de evaluar la bondad de un método de votación y que eliminen estos casos no intuitivos.

Criterio de Independencia de Opciones Irrelevantes Si una elección declara ganadora una opción *a*, entonces *a* debería también ganar una elección con las mismas preferencias de los votantes, pero cuando se eliminan una o más opciones perdedoras.

[5] propuesto por E.J. Nanson en 1907
[6] debido a A.H. Copeland en 1950

Cuadro 4.6.: Cuadro Resumen de los Mecanismos de Votación [Str80].

	Pareto	Monotonía	Perdedor de Condorcet	Ganador de Condorcet	Smith
Pluralidad	X	X			
Pluralidad con segunda vuelta	X		X		
Hare	X		X		
Coombs	X		X		
Secuencial por parejas		X	X	X	X
Recuento de Borda	X	X	X		
Black	X	X	X	X	
Nanson	X		X	X	X
Copeland	X	X	X	X	X

No obstante, la inclusión de nuevos criterios añade otra serie de problemas. De hecho, Kenneth Arrow publicó su famoso *Teorema de Imposibilidad*. Según este teorema, para un conjunto dado de criterios, es imposible para un mecanismo de votación satisfacer todos ellos de forma simultánea. A nivel práctico, implica que ningún método de votación puede satisfacer todas las condiciones vistas en este apartado sin violar Criterio de de Independencia de Opciones Irrelevantes.

Debido al teorema de imposibilidad de Arrow, los mecanismos de votación tienden a relajar algunos de los criterios vistos en este apartado. El cuadro 4.6 muestra un resumen de los diversos métodos preferenciales de votación.

4.1.2.2. Métodos no preferenciales de votación

Muchos de los problemas vistos en la sección 4.1.2.1 desaparecen con la implementación de métodos de votación no preferenciales. Así, por ejemplo, los criterios de Condorcet carecen de sentido en ausencia de un método de votación preferencial.

Votación por aprobación[7]

En este mecanismo, los votantes pueden dar ningún o un punto para cada opción. La opción ganadora es la que recibe el mayor número de votos. Cuenta con la ventaja de cumplir el criterio de independencia de opciones irrelevantes, pero también da lugar a situaciones problemáticas.

4.1.3. Mecanismos de negociación basados en subastas

En este tipo de mecanismo, un agente llamado subastador desea vender un artículo al mayor precio posible, mientras que otros agentes (los postores) desean adquirir dicho

[7]Método tomado para la elección del Secretario General de las Naciones Unidas.

artículo al precio más bajo posible. El protocolo implica un sistema centralizado en el que es el subastador quien determina el ganador en la subasta [Flo03].

Entre los métodos de subastas más conocidos se encuentran:

Subasta inglesa Cada postor anuncia abiertamente su oferta. Cuando ningún agente sube la oferta actual, la subasta finaliza. El agente de la oferta más alta gana la subasta y paga el precio de la oferta realizada.

Subasta holandesa El subastador va bajando continuamente el precio del artículo (suele partir de un precio artificialmente alto) hasta que uno de los postores acepta el precio actual.

Subasta de sobre cerrado Cada postor realiza una única puja desconociendo las de los otros agentes. La mayor oferta recibida gana la subasta y paga por el artículo el precio anunciado en su oferta.

Subasta Vickery Equivalente a la subasta de sobre cerrado pero en esta ocasión el precio a pagar por el que realiza la oferta más alta es el anunciado por la segunda oferta realizada más alta.

4.1.4. Mecanismos basados en regateo

4.1.4.1. Modelo de negociación bilateral

Sea i la representación de los agentes a y b implicados en la negociación y j la representación de los objetos bajo negociación ($j \in \{1,\cdots,n\}$). Sea $x_j \in \left[min_j^i, max_j^i\right]$ un valor para el objeto de negociación j aceptable por el agente i. Esto es, los valores posibles para el objeto de negociación se mueven en un rango predeterminado para cada agente.

Cada agente tiene un función de puntuación (*scoring function*) V_j^i: $[min_j^i, max_j^i] \rightarrow [0,1]$ que proporciona la conveniencia que el agente i asigna al valor x_j de su escala de valores aceptables. Por conveniencia estos valores se mueven en el intervalo [0,1].

Otro factor a considerar es la importancia relativa (es decir, el peso) que cada agente da a cada objeto de negociación j. Este factor se va a representar como w_j^i. Estos pesos se suponen normalizados, es decir

$$\sum_{j=1}^{n} w_i^j = 1 \quad \forall i \tag{4.1}$$

Con esto, se puede definir una función de puntuación global $V^i(x)$ para cada agente i con respecto a un contrato, siendo $x=(x_1,x_2,\cdots,x_n)$ un punto en el espacio multidimensional formado por los n objetos de negociación.

$$V^i(x) = \sum_{j=1}^{n} w_j^i V_j^i(x_j) \tag{4.2}$$

Raiffa [Rai82] demostró que, si ambos agentes negociadores emplean este tipo de función, es posible calcular el valor óptimo de x como un elemento de la frontera eficiente de negociación.

Los pesos que determinan la importancia relativa de las funciones de puntuación pueden evolucionar en el tiempo mediante *estrategias de negociación* que reflejen posibles cambios en el entorno u otras circunstancias del proceso de negociación [MSJ98, FOSG97].

Por ejemplo: supongamos un agente a cuyo conjunto de objetos de negociación sea {*precio, volumen*}. Los rangos aceptables por a son [min^a_{precio}, max^a_{precio}]=[10,20] y [$min^a_{volumen}$, $max^a_{volumen}$]=[1,5]. Consideremos que a considere el objeto de negociación *precio* más importante que el objeto de negociación *volumen*. Esto se refleja en los pesos dados por a a cada objeto: 0.8 para el precio y 0.2 para el volumen. En este caso, el modelo de función de puntuación para cada objeto de negociación es lineal en todo el rango:

$$V^a_{precio}(x_{precio}) = \frac{x_{precio} - min^a_{precio}}{max^a_{precio} - min^a_{precio}} \tag{4.3}$$

$$V^a_{volumen}(x_{volumen}) = 1 - \frac{x_{volumen} - min^a_{volumen}}{max^a_{volumen} - min^a_{volumen}} \tag{4.4}$$

Con todas estas consideraciones, si un agente servidor b realiza una oferta, por ejemplo, [11,5], el valor de $V^a(x)$ será

$$V^a(x) = 0{,}8\frac{11-10}{20-10} + 0{,}2\left(1 - \frac{5-1}{5-1}\right) = 0{,}08 \tag{4.5}$$

Esta función servirá al agente a para evaluar las ofertas recibidas, comparándolas y quedándose con la de mayor valor de su función de puntuación.

En general (aunque se pueden encontrar excepciones) un cliente c y un servidor s en una negociación tienen intereses contrapuestos. Por ejemplo, el cliente normalmente desea conseguir un precio del servicio lo más bajo posible, mientras que el servidor quiere obtener el mayor precio posible por el servicio a prestar. Es decir, si x_j,$y_j \in$ [min_j, max_j] con $x_j \geq y_j$, entonces lo más normal es que $V^c_j(x_j) \geq V^c_j(y_j)$ y que $V^s_j(x_j) \leq V^s_j(y_j)$.

El primer paso de una negociación entre agentes consiste en que éstos se pongan de acuerdo en los objetos de negociación. A partir de ahí la negociación basada en regateo entre dos agentes consiste en una sucesión alternada de ofertas entre ambos. Este proceso continúa hasta que una oferta de un agente es aceptada por el otro agente o uno de estos agentes da por finalizada la negociación (por ejemplo finaliza el plazo de tiempo fijado inicialmente para la negociación).

Representando por $x^t_{a\to b}$ el vector de valores propuesto por el agente a al agente b en un instante t, y por $x^t_{a\to b}[j]$ el valor del objeto de negociación j propuesto por a a b en el instante t, se define *un hilo de negociación* $X^t_{a\leftrightarrow b}$ entre dos agentes a y b como cualquier secuencia finita ordenada de longitud n de la forma

$$\left(x^{t_1}_{a\to b}, x^{t_2}_{b\to a}, x^{t_3}_{a\to b}, \cdots\right) \tag{4.6}$$

con $t_1 \leq t_2 \cdots \leq t_n$. Además para cada j, $x^i_{a\to b}[j] \in$ [min^a_j, max^a_j], $x^{i+1}_{b\to a}[j] \in$ [min^b_j,

max_j^b] con *i*=1,3,5... siendo el último elemento de la secuencia una de las partículas {*aceptar*,*rechazar*}.

Se dice que la negociación se encuentra activa en un momento dado si el último elemento del hilo de negociación no es ninguna de estas dos partículas.

Con esta definición, cuando un agente *a* recibe una oferta $x^t_{b\to a}$, la interpretación del agente *a* en un instante *t'*>*t* se define como

$$I^a(t', x^t_{b\to a}) = \begin{cases} rechazar & si\, t' > t^a_{max} \\ aceptar & si\, V^a(x^t_{b\to a}) \geq V^a(x^{t'}_{a\to b}) \\ x^{t'}_{a\to b} & en\, otro\, caso \end{cases} \tag{4.7}$$

donde t^a_{max} es el tiempo límite para que el agente *a* alcance un acuerdo. Es decir, si la puntuación de la oferta recibida es mayor que la que enviaría en ese momento el agente *a*, entonces la oferta es aceptada. En caso de superar el plazo para la negociación, la oferta es rechazada.

El aspecto que nos falta a tratar es cómo genera el agente *a* la nueva oferta $x^{t'}_{a\to b}$. Esto se implementa a través de las llamadas tácticas de negociación. Estas tácticas consisten en un conjunto de funciones que determinan cómo calcular el valor de un objeto de negociación, considerando un único criterio (tiempo, recursos...). En caso de querer negociar basándose en varios criterios se puede generar una nueva función que sea la suma ponderada de varias tácticas basadas en un único criterio. El rango de la función son los valores posibles del objeto de negociación mientras que el dominio consiste en el criterio de la táctica. Diversos trabajos [FSJ98, MSJ98] proponen tres grupos principales de tácticas de negociación:

4.1.4.2. Tácticas dependientes del tiempo

Si un agente *a* tiene un tiempo límite t^a_{max} para alcanzar un acuerdo, la táctica a emplear debe ser tal que las concesiones a efectuar aumenten según se acerca el final del plazo. En este caso, la función de táctica propuesta para la oferta que el agente *a* manda a un agente *b* es:

$$x^t_{a\to b}[j] = \begin{cases} min^a_j + \alpha^a_j(t)(max^a_j - min^a_j) & si\, V^a_j\, es\, decreciente \\ min^a_j + (1-\alpha^a_j(t))(max^a_j - min^a_j) & si\, V^a_j\, es\, creciente \end{cases} \tag{4.8}$$

donde $\alpha^a_j(t)$ es una función dependiente del tiempo cuyos valores se encuentran entre 0 y 1. Además $\alpha^a_j(0) = \kappa^a_j$ (su valor inicial) y $\alpha^a_j(t^a_{max}) = 1$. Ejemplos de esta función son:

- Polinomios:

$$\alpha^a_j(t) = \kappa^a_j + (1-\kappa^a_j)\left(\frac{\text{mín}(t, t^a_{max})}{t^a_{max}}\right)^{\frac{1}{\beta}} \tag{4.9}$$

- Términos exponenciales:

$$\alpha^a_j(t) = \exp\left(\left(1 - \frac{\text{mín}(t, t^a_{max})}{t^a_{max}}\right)^{\beta} \ln \kappa^a_j\right) \tag{4.10}$$

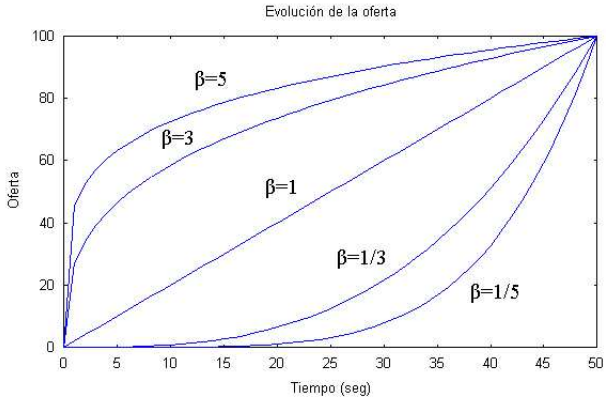

Figura 4.1.: Evolución de la oferta en técnicas dependientes del tiempo según el valor de β (expresión polinomial).

donde el parámetro $\beta \in \mathbb{R}^+$ determina la convexidad de la curva de comportamiento de las ofertas. Así, un valor de $\beta<1$ implica el mantenimiento del valor de la oferta hasta que el tiempo está casi agotado que es cuando concede el valor de reserva max_j^a (táctica de Boulware). Por el contrario, un valor de $\beta>1$ provoca que el agente alcance rápidamente su valor de reserva (táctica de concesión). Estos comportamientos para las expresiones polimoniales se ilustran en la Figura 4.1

4.1.4.3. Tácticas dependientes de los recursos

Este tipo de tácticas reflejan el curso de una negociación donde existe una limitación en los recursos (por ejemplo el ancho de banda que queda por ser asignado en una red de comunicaciones). Se puede considerar esta familia de tácticas como un caso particular de las tácticas dependientes del tiempo salvo que el dominio de la función es la cantidad de los recursos a negociar en vez del tiempo restante.

El modelado de estas tácticas se puede realizar de dos modos:

- Haciendo dinámico el valor de t_{max}^a en la ecuación 4.8, de modo que refleje la cantidad de recursos restantes en el entorno de negociación. Por ejemplo, consideremos como dominio el número de agentes que negocian con a. Un número elevado de agentes supone que la presión sobre a para alcanzar un acuerdo disminuya, presión que aumenta a medida que se alarga la negociación. Así, un posible modelado de esta situación es el siguiente:

$$t_{max}^a(t) = \mu^a \frac{|N^a(t)|^2}{\sum_i |X_{i \leftrightarrow a}^t|} \tag{4.11}$$

donde

  - μ^a representa el tiempo que el agente a considera como razonable para alcanzar un acuerdo

- $N^a(t)$ representa el número de agentes que mantienen un hilo de negociación activo con el agente a en un instante t.
- $|X^t_{i\leftrightarrow a}|$ representa la longitud del hilo de negociación entre los agentes a e i.

- Mediante tácticas de estimación de los recursos. En este caso, las ofertas sucesivas dependen del ritmo de consumo del objeto de negociación, de modo que el agente cliente se muestre más dispuesto al acuerdo según disminuya el número de recursos disponibles. Llamando *recurso*^{a}*(t)* a la función que representa la cantidad de recurso disponible para el agente a en un instante t, un comportamiento adecuado vendría determinado por ejemplo por la siguiente función

$$\alpha^a_j(t) = \kappa^a_j + (1-\kappa^a_j)e^{-recurso^a(t)} \tag{4.12}$$

4.1.4.4. Tácticas dependientes del comportamiento

Con esta familia de tácticas, un agente calcula la próxima oferta a realizar basándose en el comportamiento previo de su oponente, protegiéndose de este modo de ser explotado por otros agentes. Se ha demostrado la importancia de este tipo de tácticas en la negociación en la resolución de problemas de forma cooperativa.

Dado un hilo de negociación $\{\cdots, x^{t_n-2\delta}_{b\to a}, x^{t_n-2\delta+1}_{a\to b}, x^{t_n-2\delta+2}_{b\to a}, \cdots, x^{t_n-2}_{b\to a}, x^{t_n+1}_{a\to b}, x^{t_n}_{b\to a}\}$ con $\delta \geq 1$, se pueden distinguir las siguientes tácticas:

- Tit-For-Tat relativo (*Relative* TFT). El agente reproduce, en términos de porcentaje el comportamiento de su componente llevado a cabo δ etapas antes. Puede aplicarse cuando $n>2\delta$.

$$x^{t_{n+1}}_{a\to b}[j] = \text{mín}\left(\text{máx}\left(\frac{x^{t_{n-2\delta}}_{b\to a}[j]}{x^{t_{n-2\delta+2}}_{b\to a}[j]} x^{t_{n-1}}_{a\to b}[j], min^a_j\right), max^a_j\right) \tag{4.13}$$

- Tit-For-Tat absoluto aleatorio (*Random* TFT). Es idéntico al anterior pero esta vez en términos absolutos. Se añade o disminuye (dependiendo de si la función de puntuación es creciente o decreciente) un factor aleatorio, lo cual facilita al agente su salida de puntos de mínimo local. Si M es la máxima variación que un agente puede introducir en una oferta, entonces

$$\begin{aligned} x^{t_{n+1}}_{a\to b}[j] &= \text{mín}\left(\text{máx}\left(x^{t_{n-1}}_{a\to b}[j] + \left(x^{t_{n-2\delta}}_{b\to a}[j] - x^{t_{n-2\delta+2}}_{b\to a}[j]\right)\right.\right. \\ &+ \left.\left.(-1)^s R(M), min^a_j\right), max^a_j\right) \end{aligned} \tag{4.14}$$

donde

$$s = \begin{cases} 0 & si V^a_j \, es decreciente \\ 1 & si V^a_j \, es creciente \end{cases} \tag{4.15}$$

y $R(M)$ es una función que genera un número entero aleatorio en el intervalo [0,M]. La táctica puede aplicarse cuando $n>2\delta$.

- Tit-For-Tat promediado (*Average* TFT). El agente se basa en los cambios en las ofertas de su adversario en las últimas $\gamma \geq 1$ etapas. En el caso $\gamma = 1$ se reduce al caso de la táctica de Tit-For-Tat relativo con $\delta = 1$. La táctica puede aplicarse cuando $n > 2\gamma$.

$$x_{a\to b}^{t_{n+1}}[j] = \text{mín}\left(\text{máx}\left(\frac{x_{b\to a}^{t_{n-2\gamma}}[j]}{x_{b\to a}^{t_n}[j]} x_{a\to b}^{t_{n-1}}[j], min_j^a\right), max_j^a\right) \tag{4.16}$$

4.1.5. Redes de Contrato

El protocolo de redes de contrato (*contract nets*) consiste en un protocolo de interacción para la resolución de problemas cooperativos entre agentes. Para ello se basa en el modelo del mecanismo de contrato empleado por las empresas y negocios para gobernar el intercambio de bienes y servicios. La red de contrato proporciona una solución para el llamado problema de conexión (*connection problem*), consistente en encontrar un agente apropiado para una tarea dada.

Un agente llamado *manager* desea que una tarea dada sea llevada a cabo y para ello negocia con los posibles agentes que pueden realizarla (los contratistas, *contractors*) [HS99]. Desde el punto de vista del manager, el proceso consiste en

- Anunciar la tarea a realizar
- Recibir y evaluar las ofertas de los potenciales contratistas
- Realizar un contrato con el contratista adecuado
- Recibir los resultados y sintetizarlos

Desde el punto de vista de los contratistas,

- Recibir el anuncio de la tarea a desarrollar
- Evaluar la capacidad propia para responder
- Responder al *manager*
- Desarrollar la tarea si la oferta es aceptada
- Informar al *manager* de los resultados de la tarea

FIPA define al respecto dos protocolos: *FIPA Contract Net* y *FIPA Iterated Contract Net*.

4.1.6. Coaliciones

Una coalición consiste en un subconjunto de agentes que se agrupan para conseguir un beneficio común, ya que pueden ahorrar costes y aumentar sus beneficios de dicho modo, y que se disgrega una vez que el objetivo se ha cumplido [BdlCT97]. La formación de una coalición consta de tres puntos [SL97]:

1. Generación de la estructura de la coalición. La estructura debe ser estable, es decir, ninguno de los agentes involucrados en la negociación debe tener incentivos para separarse de su coalición.

2. Resolver el problema de optimización de cada coalición.

3. Dividir el valor de la solución generada entre los agentes de la coalición.

4.1.7. Negociación basada en argumentación

En este tipo de negociación, se emplean argumentos para convencer a un agente de que acepte una propuesta de negociación. Cada argumento define una serie de condiciones previas para su empleo. Si estas condiciones previas se cumplen, entonces el agente podría emplear el argumento en su negociación. Los argumentos que se pueden emplear son variados, como por ejemplo, recordar una promesa anterior en el tiempo, la promesa de una recompensa futura, convencer al otro agente de que también se beneficiará de la acción a realizar o amenazar con represalias en caso de negarse al acuerdo. El agente debe dotarse de una estrategia para decidir qué argumento emplear. Para ello, la mayoría de las ocasiones se asume un modelo BDI (ver la sección 3.1.3).

4.1.8. Teoría de Juegos

La Teoría de Juegos consiste en un enfoque interdisciplinario para estudiar el comportamiento humano. Mediante el empleo de ramas como las matemáticas, la economía y las ciencias sociales, permite, por ejemplo, proporcionar una serie de criterios para evaluar los protocolos de negociación entre agentes auto-interesados. Los agentes se suponen que deben comportarse de un modo racional [Flo03].

Comportamiento racional Un agente prefiere una gran utilidad (recompensa) frente a una pequeña. La racionalidad consiste en maximizar los beneficios bajo unas condiciones dadas.

La maximización de la recompensa puede seguir algún criterio, por ejemplo, la recompensa individual, recompensas de grupo o bienestar social.

Bienestar social Es la suma de las utilidades de todos los agentes para una solución dada. De este modo se realiza una medición del bien global de los agentes. El problema surge al comparar estas utilidades.

Un protocolo es estable si una vez que los agentes han llegado a una solución no se desvían de él.

Estrategia dominante El agente consigue un mejor resultado empleando una estrategia específica independientemente de qué estrategias emplean los otros agentes.

Equilibrio de Nash Dos estrategias S_1 de un agente A y S_2 de un agente B se encuentran en equilibrio de Nash si se cumplen las siguientes dos condiciones

- En el caso de que el agente *A* siga S_1, el agente *B* no puede tener una mejor recompensa que la obtenida usando S_2.
- En el caso de que el agente *B* siga S_2, el agente *A* no puede tener una mejor recompensa que la obtenida usando S_1.

Generalizado a sistemas con varios agentes, el equilibrio de Nash implica que la estrategia de cada agente es la mejor respuesta a la estrategia del resto de los agentes.

4.1.9. Aprendizaje mediante construcción de árboles de identificación

Se define un árbol de identificación como un árbol de decisión en el que cada conjunto de posibles conclusiones se establece implícitamente mediante una lista de muestras de clase conocida.

Para clarificar este concepto, considérese el siguiente ejemplo tomado de [Win94]. Considérese que se desea investigar los factores que determinan que una persona se queme la piel tras un periodo de exposición al sol. Para ello se toma una serie de posibles factores como por ejemplo la estatura, el color de pelo, el peso y el empleo de loción, investigándose los valores de estos factores en una población de individuos.

Cuadro 4.7.: Población de ejemplo para árbol de identificación

Nombre	Color de Pelo	Estatura	Peso	Loción	Quemado
Sarah	rubio	promedio	ligero	no	sí
Dana	rubio	alta	promedio	sí	no
Alex	castaño	baja	promedio	sí	no
Annie	rubio	baja	promedio	no	sí
Emily	pelirrojo	promedio	pesado	no	sí
Pete	castaño	alta	pesado	no	no
John	castaño	promedio	pesado	no	no
Katie	rubio	baja	ligero	sí	no

Ante una nueva persona, y a partir de los valores observados (ver Tabla 4.7) se quiere determinar si se va a quemar la piel o no. Un posible método consistiría en buscar en la tabla un patrón de variables idéntico al de la nueva persona. Sin embargo con una cantidad elevada de variables este método es impracticable, ya que la probabilidad de encontrar un dato con los mismos valores de sus variables es pequeño. Por ejemplo, en una situación de 12 variables con 5 valores posibles para cada variable, aún con una tabla de un millón de datos, la posibilidad de encontrar un dato con los mismos valores es del 0,4 %. En este sentido el empleo de árboles de identificación es más práctico, permitiendo clasificar las propiedades de los datos observados.

El primer problema a la hora de elaborar un árbol de identificación es saber cuál de las posibles clasificaciones posibles es la más adecuada y por tanto con más visos de veracidad. Por ejemplo, a partir los datos anteriores se pueden generar, entre otros, a los árboles de identificación mostrados en las Figuras 4.2 y 4.3. El primero de ellos parece

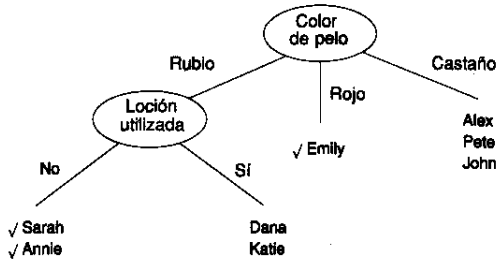

Figura 4.2.: Ejemplo de árbol de identificación [Win94].

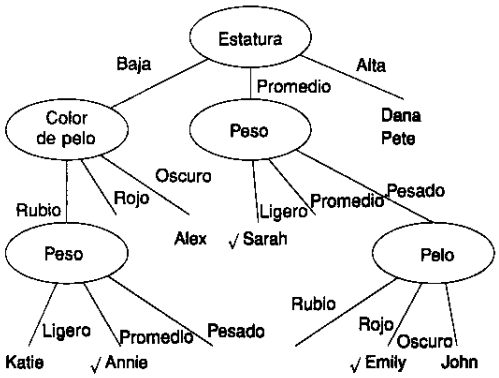

Figura 4.3.: Ejemplo de árbol de identificación [Win94].

coincidir en un mayor grado con lo esperado de forma intuitiva, pero en principio el segundo árbol de identificación parece igualmente válido.

En este punto, se toma una variación del conocido principio de la navaja de Occam.

> El mundo es inherentemente simple. Por tanto, el árbol de identificación más pequeño consistente con la muestra es el que tiene más probabilidades de identificar objetos desconocidos de manera correcta.

Para proporcionar un método cuantitativo para la estimación de la sencillez y consistencia de un árbol de identificación se habla de desorden del árbol, en el sentido de la aparición de grupos de datos de carácter heterogéneo en la clasificación. Por tanto, se considerará como el mejor de los árboles de identificación aquél que clasifique los datos con un menor desorden. De modo matemático, este desorden se calcula del siguiente modo:

$$\textit{Desorden promedio} = \sum_b \left[\left(\frac{n_b}{n_t} \right) \left(\sum_c -\frac{n_{bc}}{n_b} \log_2 \frac{n_{bc}}{n_b} \right) \right] \tag{4.17}$$

donde

- n_b es el número de muestras en la rama b.

- n_t es el número total de muestras en todas las ramas.
- n_{bc} es el número de muestras en la rama b de la clase c.

De un modo práctico, las muestras iniciales se ordenan según cada una de las variables consideradas, seleccionando aquella clasificación que presente un menor desorden. Este proceso se repite de forma recursiva hasta que los datos presentan la mayor homogeneidad posible.

Aplicando este proceso a los datos de la Tabla 4.7 se puede comprobar que la clasificación de la Figura 4.2 es la adecuada, ya que presenta un menor desorden.

Tras lograr un árbol de identificación consistente para los datos observados, a partir de él se pueden enunciar la siguiente serie de reglas:

1. Si el color de pelo de la persona es rubio y la persona usa loción, entonces no se quema.
2. Si el color de pelo de la persona es rubio y la persona no usa loción, entonces se quema.
3. Si el color de pelo de la persona es pelirrojo, entonces se quema.
4. Si el color de pelo de la persona es castaño, entonces no se quema.

Estas reglas permiten una clasificación de una nueva muestra. No obstante, el conjunto de reglas puede simplificarse mediante la eliminación tanto de antecedentes como de reglas innecesarias. Para el primer caso, se suelen emplear las denominadas tablas de contingencia, las cuales muestran el grado en que el resultado es aleatorio sobre una propiedad.

Por ejemplo, para el antecedente de persona de pelo rubio en la regla 1, se toma el número de muestras que verifiquen el segundo antecedente, estudiando si una persona se va a quemar según sea rubia o no. De esta tabla (ver Tabla 4.8), se concluye que el antecedente *rubio* es innecesario en la regla, puesto que no influye en el resultado.

Cuadro 4.8.: Ejemplo de tabla de contingencia para los casos de “uso de loción”.

	Se quema	No se quema
Persona de pelo rubio	2	0
Persona de pelo no rubio	1	0

En cambio, si se elabora una tabla de contingencia (ver Tabla 4.9) para el segundo antecedente (*uso de loción*) se comprueba que en este caso el antecedente es importante.

De estas tablas de contingencia la regla 1 queda de la siguiente forma:

1. Si la persona usa loción, entonces no se quema.

De una forma matemática, la determinación de la importancia o no de los antecedentes de las reglas puede efectuarse mediante la prueba exacta de Fisher.

Cuadro 4.9.: Tabla de contingencia para el antecedente de uso de loción en la regla 1 para los casos de “ser rubio”.

	Se quema	No se quema
Uso de loción	2	0
No uso de loción	0	2

Supóngase un estudio de la influencia de la propiedad P sobre la presencia o ausencia de un resultado R. Se denota por R_+ la presencia de R y por R_- su ausencia. Del mismo modo, se denota por P_+ y P_- respectivamente la presencia y ausencia de la propiedad P. La forma general de la tabla de contingencia es la mostrada en la Tabla 4.10.

Cuadro 4.10.: Forma general de tabla de contingencia.

	R_+	R_-
P_+	V_{++}	V_{+-}
P_-	V_{-+}	V_{--}

Los valores de V_{++}, V_{+-}, V_{-+} y V_{--} son los que determinan la relevancia de P en R. Por ejemplo, los valores reflejados en las tablas 4.11, 4.12 y 4.13 se refieren a situaciones diferentes. Si nos basamos en 4.11, se debe mantener el antecedente P, puesto que su eliminación implicaría la clasificación incorrecta de una muestra. La tabla 4.12 en principio se vería afectada por la misma situación, pero en este caso, se podría plantear la conveniencia de la simplificación ya que solamente se clasificaría mal uno de cada mil datos, es decir un error ocasional. Además se podría considerar que ese dato es producido por una medición con ruido. Del mismo modo, las tablas 4.11 y 4.13 reflejan la misma proporción en los datos, pero de modo intuitivo, los de la última parecen reflejar una mayor consistencia en las reglas.

Cuadro 4.11.:

	R_+	R_-
P_+	1	0
P_-	0	1

El primer modo de establecer una estrategia para la simplificación de la reglas consistiría en determinar la existencia de una dependencia estadística entre P y R. Desgraciadamente, de existir, es extremadamente complicado determinar a priori la forma de la misma. Sin embargo, resulta mucho más fácil determinar la independencia estadística, la cual solamente tiene una forma. De este modo, en vez de demostrar que R depende de P, se trata de mostrar que es improbable que R no dependa de P.

Una vez determinado este primer objetivo, el segundo consiste en calcular la

Cuadro 4.12.:

	R_+	R_-
P_+	999	0
P_-	0	1

Cuadro 4.13.:

	R_+	R_-
P_+	1000	0
P_-	0	1000

posibilidad de observar una combinación particular V_{++}, V_{+-}, V_{-+}, V_{--}. El método consiste en suponer que existe cierto número fijo de muestras correspondientes a P_+ ($SP_+ = V_{++} + V_{+-}$), P_- ($SP_- = V_{-+} + V_{--}$), R_+ ($SR_+ = V_{++} + V_{-+}$), R_- ($SR_- = V_{+-} + V_{--}$). A estas cantidades se les denomina *suma marginales* (ver Tabla 4.14).

Cuadro 4.14.: Tabla de contingencias con sumas marginales.

	R_+	R_-	Suma marginal
P_+	V_{++}	V_{+-}	$SP_+ = V_{++} + V_{+-}$
P_-	V_{-+}	V_{--}	$SP_- = V_{-+} + V_{--}$
Suma marginal	$SR_+ = V_{++} + V_{-+}$	$SR_- = V_{+-} + V_{--}$	$SP_+ + SP_- = SR_+ + SR_-$

Una vez fijada las sumas marginales, se escoge un valor para uno de los parámetros, el cual determina el resto mediante las sumas marginales. En el caso de escoger un valor para el parámetro V_{++} (para demostrar así la independencia o no de P y R), se toma la siguiente fórmula que determina la probabilidad del factor V_{++} dadas las sumas marginales.

$$p(V_{++}|SP_+, SP_-, SR_+, SR_-) = \frac{\frac{SP_+!}{V_{++}!(SP_+ - V_{++})!}\frac{SP_-!}{(SR_+ - V_{++})!(SP_- - (SR_+ - V_{++}))!}}{\frac{(SP_+ + SP_-)!}{SR_+!(SP_+ + SP_- - SR_+))!}} \quad (4.18)$$

Esta fórmula genera una curva característica como la mostrada para en la Figura 4.4 (la simetría en la figura se pierde cuando los valores de los parámetros son distintos entre sí). Se comprueba que la probabilidad de los valores de los extremos de V_{++} es muy baja. Por tanto, si el valor observado de V_{++} se encuentra fuera del intervalo central, se puede concluir que la probabilidad de independencia es muy baja (concretamente menos de un 5 %), y por lo tanto el antecedente debe mantenerse, pues la propiedad y el resultado muestran dependencia entre sí [Win94].

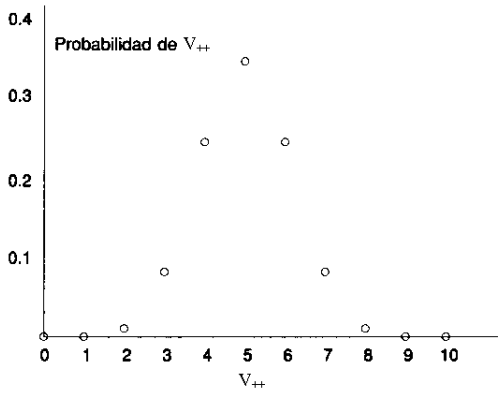

Figura 4.4.: Curva de probabilidad de l con $SP_+ = SP_- = SR_+ = SR_- = 10$ [Win94].

4.2. Aplicaciones MAS similares

La planificación automática de reuniones es uno de los campos clásicos de la aplicación de sistemas multiagentes. Con fines ilustrativos, tomaremos tres aplicaciones significativas para este cometido, el *Visitorbot* de Kautz et al. en el *AI Principles Research Department* en los *AT&T Bell Laboratories*[8] [KSCK94], el propuesto por Sandip Sen en la Universidad de Tulsa[9] [Sen97a] y el sistema denominado *Electric Elves*[10] [STLP00, PTAC00, CGK+01, HGB] de la División de Sistemas Inteligentes de la USC/ISI[11].

4.2.1. Visitorbot

Kautz et al. [KSCK94] presentan un sistema para el problema de planificar una visita externa a un laboratorio. Este trabajo es bastante rutinario, pero consume una gran cantidad de tiempo. La secuencia de tareas, en este modelo, consiste en los siguientes puntos:

- Anunciar la visita por correo electrónico.
- Recolectar las respuestas de los usuarios que deseen encontrarse con el visitante, con sus respectivos horarios deseados.
- Planificar un horario global que satisfaga, lo más posible, los deseos de los miembros del laboratorio.
- Enviar esta planificación-resultado a los participantes.
- Reorganizar ante cambios imprevistos.

[8] http://www.research.att.com/
[9] http://www.mcs.utulsa.edu/~sandip/sandip.html
[10] Término que se puede traducir como *Elfos Eléctricos*
[11] http://www.isi.edu/divisions/div3/

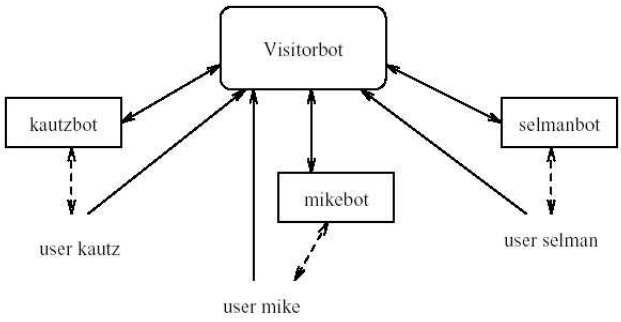

Figura 4.5.: Sistema Visitorbot.

Con estas premisas, se ha diseñado un agente software denominado *visitorbot*, encargado de calcular la mejor planificación posible. Inicialmente se diseñó un modelo de un único agente monolítico, en el cual se observaron los siguientes tres puntos:

1. La comunicación por correo electrónico entre el *visitorbot* y los usuarios humanos se manifestó como engorrosa y propensa a errores. Aparte de esto, muchos usuarios se mostraron disgustados con el sistema porque consideraban que suponía una complicación adicional. Para esto, se hace necesaria la inclusión de una interfaz gráfica.

2. Se necesita una redundancia en el manejo de los errores. El sistema podría mostrarse confuso ante correos electrónicos rebotados o mandados por otros programas.

3. La tarea de crear la planificación a partir de una serie de restricciones no necesitaba en principio técnicas avanzadas de planificación.

Estos puntos llevó al diseño de la Figura 4.5. La interacción entre el *visitorbot* y los diferentes usuarios queda como sigue. El anuncio inicial de la visita se envía a través de correo electrónico a los usuarios. El agente software de cada usuario (llamado *userbot*) determina entonces el mejor método de comunicación con su propietario, ya sea a través de una interfaz gráfica o un correo electrónico. Entonces, el usuario puede mandar sus preferencias del mismo modo al *visitorbot*. En dicha figura las líneas continuas representan comunicaciones a través de correo electrónico, mientras que las líneas de puntos representan comunicaciones a través de interfaces gráficas y correo electrónico.

4.2.2. El modelo de Sandip Sen

En diversos artículos, Sandip Sen [Sen97a, SHA97, Sen97b] estudia el problema de la automatización de la planificación de reuniones entre los miembros de una organización. Su enfoque a la hora de tratar este problema consiste en un sistema distribuido donde cada empleado posee su propio agente. Cuando un usuario desea planificar una reunión con otros usuarios, se introduce una petición en su

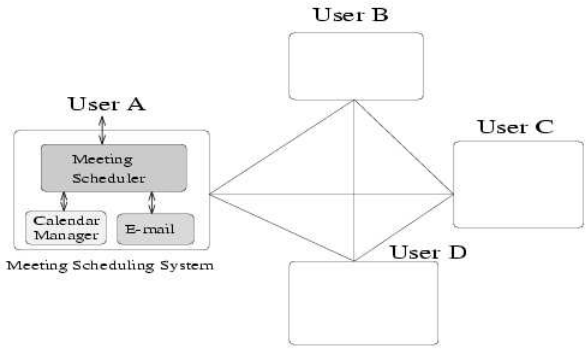

Figura 4.6.: Sistema multiagente de Sandip Sen [Sen97a, SHA97, Sen97b].

agente correspondiente. Este agente negocia con los agentes relativos a los usuarios involucrados. Como las respuestas y las agendas de los usuarios son accedidas a través del agente, se preserva la privacidad del usuario. El agente tiene acceso a la agenda del usuario y a un software de correo electrónico para las comunicaciones con los otros agentes. Este modelo se muestra en la Figura 4.6.

El usuario que pide la fijación de un reunión con otros usuarios se denomina huésped (*host*), mientras que los usuarios requeridos para la reunión se denominan invitados (*invitee*). El protocolo simplificado de la planificación consta de cuatro puntos:

1. Cuando se necesita planificar una reunión, el agente huésped trata de encontrar los intervalos de tiempo más adecuados que respeten lo más posible las restricciones de fecha y horario deseados por su usuario. Si no puede encontrar ningún intervalo adecuado, se produce un fallo y se abandona la planificación de la reunión. De otro modo, si es el único participante de la reunión, planifica una reunión para el mejor horario posible. Si existen otros participantes, el huésped anuncia un contrato para la reunión a los invitados, proponiendo uno o más de los intervalos encontrados. La reunión se especifica por el conjunto de asistentes, la duración propuesta, la prioridad, un conjunto de posibles horas de inicio de la reunión, un límite para la negociación de la reunión y otras posibles restricciones.

2. Cada invitado recibe las propuestas de contrato y trata de encontrar soluciones locales para satisfacer esos contratos y enviar sus propuestas al huésped. Las ofertas consisten en uno o más intervalos de tiempo (incluidos o no en la propuesta inicial del huésped) para los cuales los invitados pueden asistir a la reunión.

3. El huésped recolecta y evalúa estas propuestas. En el caso de que las ofertas sugieran un horario común que se encuentra libre para todos los asistentes, la reunión puede celebrarse y el huésped envía el horario-resultado a los invitados. En caso contrario, el huésped genera nuevas propuestas dependiendo de las ofertas recibidas y de su propia agenda.

4. Cuando los invitados reciben nuevas propuestas, se repiten de forma iterativa los puntos 2 y 3, hasta que se llega a un acuerdo o se cumple el plazo fijado.

Para el mecanismo de negociación se ha optado por una combinación de estrategias heurísticas, con una arquitectura interna para el planificador de reuniones mostrada en la Figura 4.7. Esta arquitectura se compone de los siguientes módulos:

- El módulo de negociación: Es el cerebro del sistema. Consulta las preferencias del usuario a la hora de fijar las reuniones con los planificadores del resto de usuarios involucrados, almacenando los resultados en la memoria de trabajo.
- El manipulador de agenda. Permite a la interfaz de usuario y al módulo de negociación acceder y modificar el estado de la planificación de la agenda del usuario a través de un programa de gestión de agendas.
- El constructor/decodificador de mensajes. Sirve de interfaz con el sistema de correo electrónico a través de cuyos mensajes el planificador se comunica con el resto de planificadores.
- La memoria de trabajo. Simplemente almacena todas las comunicaciones sobre todas las reuniones que se están negociando en un momento dado.
- Preferencias de usuario. Codifica las preferencias y prioridades del usuario que se emplearán como restricciones a cumplir en el proceso de negociación.

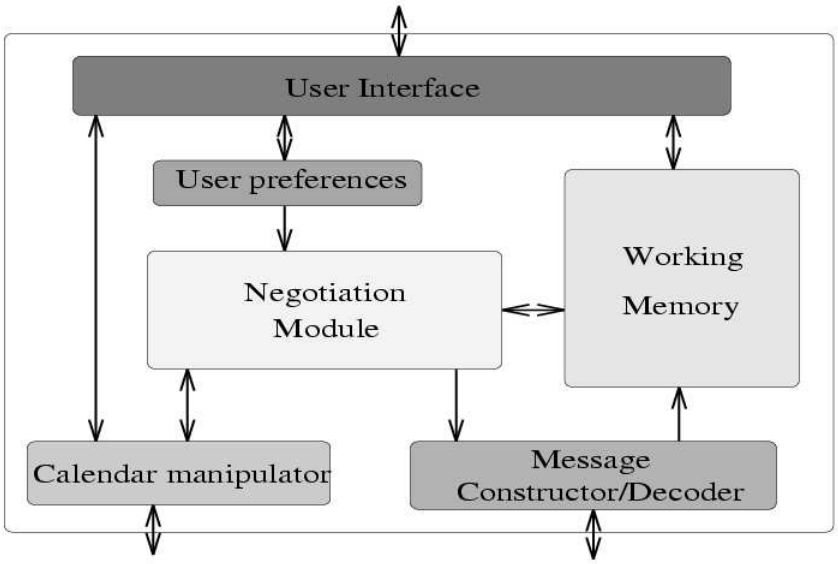

Figura 4.7.: Arquitectura del planificador de agenda propuesto por Sen et al. [Sen97a, SHA97, Sen97b].

Una propuesta para la codificación de las preferencias de usuario consiste en fijar los factores que van a determinar la conveniencia o no de una reunión para el conjunto

de usuarios. Por ejemplo, se puede considerar el día de la semana, el tramo horario (mañana, tarde), el número de asistentes... A continuación los usuarios asignan una cantidad entre 0 y 1 a cada posible valor de todos los factores involucrados, así como un criterio de mínimo, debajo del cual el usuario manifestaría su preferencia por no acudir a la reunión. La suma de las cantidades de cada factor puede adquirir cualquier valor.

El siguiente paso consiste en asignar unos pesos relativos que reflejen la importancia de cada factor en la disponibilidad del usuario para acceder a la reunión.

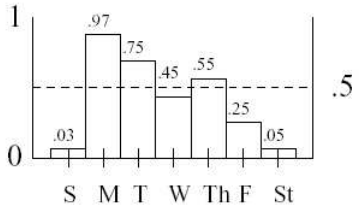

Figura 4.8.: Ejemplo de asignación de prioridades a cada valor posible de la dimensión *día de la semana*. El valor de mínimo es este caso 0.5 [Sen97a, SHA97, Sen97b].

A partir de estos factores, se pueden implementar diferentes esquemas de decisión, como por ejemplo basados en mecanismos de votación, que permiten seleccionar la más adecuada entre las propuestas recibidas.

4.2.3. *Electric Elves*

Los *Electric Elves* [STLP00, PTAC00, CGK+01] es un completo proyecto de la División de Agentes Inteligentes de la USC/ISI. Su objetivo consiste en que los agentes del sistema gestionen de modo automático tareas como:

- Seleccionar equipos de investigadores para llevar a cabo una exposición fuera de la ciudad donde se encuentra el laboratorio, planificando los detalles del viaje y alojamiento y tratando los imprevistos que pudiesen surgir (por ejemplo, que uno de los miembros del equipo cayese enfermo el día antes de la exposición).
- Determinar aquellos investigadores que pudiesen estar interesados en encontrarse con un visitante al laboratorio, planificando las reuniones entre ellos.
- Volver a planificar las reuniones si uno o más usuarios se encuentran ausentes o no pueden llegar a tiempo a ellas.
- Monitorizar la localización de los usuarios y mantener a los otros informados (imponiendo algún tipo de política de privacidad) acerca de su paradero.

Como ejemplo del funcionamiento del sistema, referenciamos dos casos diferenciados. El primero se refiere a la coordinación de las actividades de los componentes de un

proyecto, mientras que el segundo se refiere a la organización de una visita de personas externas al laboratorio.

En cuanto a la coordinación entre los miembros de un proyecto, se asigna a cada persona involucrada un agente personal (denominado *Friday*[12]) que la representa en el sistema. Este agente mantiene información actualizada sobre el paradero actual de su usuario, empleando para ello los datos que tiene sobre su agenda, un dispositivo GPS cuando el usuario se encuentra fuera del edificio, un sistema de comunicaciones por infrarrojos en el interior del edificio y la actividad de los diferentes ordenadores del laboratorio. Cuando uno de estos agentes se da cuenta de que alguien se encuentra ausente de una reunión o que se encuentra demasiado lejos para poder llegar a tiempo, entonces su agente le envía una petición a un dispositivo personal (como por ejemplo un teléfono móvil) preguntándole si desea cancelar la reunión, retrasarla o que la reunión empiece sin él. Si se produce respuesta, ésta se reenvía al resto de usuarios afectados. En caso contrario, es el agente el encargado de tomar una decisión. El sistema también se encarga de reservar mesa en restaurantes cercanos a la reunión, teniendo en cuenta las especificaciones previas sobre la dieta de los asistentes.

Por otro lado, en la organización de visitas externas, el sistema de agentes trata de identificar a aquellas personas que podrían estar interesadas en asistir a reuniones con el visitante. Para ello, los agentes acceden a un sistema de bibliografía on-line que proporciona la lista de artículos escritos por una persona. Entonces, trata de deducir las áreas de interés de cada asistente confirmado en la reunión. Con esta información selecciona los participantes en la reunión y envía una invitación a todos los participantes potenciales. Tras esto, selecciona el lugar adecuado para llevar a cabo la reunión y gestiona la reserva de restaurantes para las comidas.

Uno de los aspectos claves de este sistema es el grado de autonomía con que se dota a un agente. Un agente tiene la opción de actuar de manera completamente autónoma o, en el otro extremo, de actuar sin ninguna clase de autonomía, limitándose únicamente a preguntar al usuario qué hacer. No es un problema sencillo, puesto que una mayor autonomía conlleva un mayor ahorro de tiempo para el usuario, pero por otro lado puede llevar potencialmente a errores graves: por ejemplo, pedir una reserva en un restaurante de lujo cuando el usuario no se encuentra con apetito. En este sentido, el sistema de *Electric Elves*, emplea árboles de decisión bajo C4.5[13] que aprenden las decisiones del usuario. En este proceso, involucra un proceso en tres fases:

1. Antes de la transferencia del control en la toma de decisiones, un agente pesa explícitamente el coste de esperar la respuesta del usuario así como la probabilidad y coste de un posible error producto de una acción autónoma.

2. En la transferencia de control, un agente no se compromete de forma rígida en su decisión, sino que de forma flexible vuelve a evaluar su decisión, tomando una decisión contraria en algunas ocasiones.

3. Antes de tomar una decisión autónoma con un riesgo elevado, un agente cambia su agenda de negociación, posponiendo la toma de decisión o reordenando las actividades que pudiesen potencialmente hacerle ganar tiempo.

[12] por analogía al personaje de Viernes en *Robinson Crusoe*

[13] Extensión del algoritmo de árboles de identificación realizada por Ross Quinlan. http://www.cse.unsw.edu.au/~quinlan

La arquitectura de los *Electric Elves* se muestra en la Figura 4.9. La comunicación entre los diversos agentes se realiza mediante KQML. Los agentes se coordinan empleando *TeamCore* una arquitectura de integración independiente del dominio, descentralizada y basada en la colaboración [PTCC99].

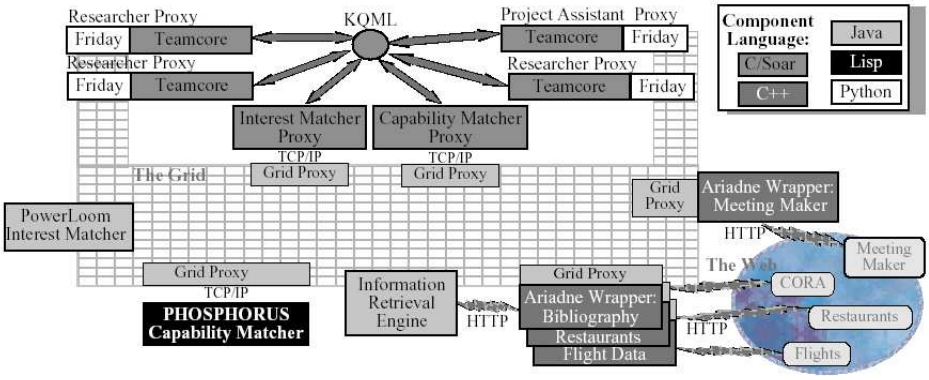

Figura 4.9.: Arquitectura del Sistema *Electric Elves* [STLP00, PTAC00, CGK+01].

4.3. Introducción a MASplan

Figura 4.10.: Logotipo MASplan.

En las siguientes secciones presentamos un sistema multiagente, elaborado durante el presente trabajo, denominado MASplan [GHM+03c] para la planificación automática de las agendas de los miembros de un grupo de investigación universitario. Este problema de planificación es, como se ha visto en la sección 4.2, clásico dentro del campo de los sistemas multiagente. Sin embargo, lo novedoso del planteamiento de este trabajo respecto a los anteriores (aparte del campo de aplicación y funcionalidad) es el empleo de la arquitectura FIPA y de un agente de ontologías basado en un

lenguaje de marcas de alta expresividad semántica, lo que está en la línea de las últimas novedades en el desarrollo de MAS.

De un modo más concreto, este sistema multiagente ha sido desarrollado para el caso del Grupo de Computadoras y Control (CyC), el cual desarrolla su actividad en la Universidad de La Laguna (Tenerife, España). En la actualidad dicho grupo cuenta con 2 catedráticos, 6 profesores titulares, 12 profesores asociados, 3 becarios y 4 miembros del servicio técnico, localizados físicamente en diferentes despachos ubicados incluso en edificios distintos y con acceso a computadores con distintos sistemas operativos.

Los miembros de este grupo llevan a cabo actividades tanto de carácter académico como de investigación, por lo que son frecuentes las reuniones internas para tratar una gran variedad de temas. Desgraciadamente, a veces estas reuniones internas son extremadamente complicadas de organizar puesto que es casi imposible encontrar una fecha y hora adecuadas para todos los asistentes deseados. De este modo, casi nunca se puede encontrar la solución deseable de que ningún miembro del grupo vea interrumpidos sus planes por la organización de una reunión, teniendo que llegar al compromiso de una solución que cause el menor daño al bienestar social del grupo.

Otra fuente de conflicto consiste en la disponibilidad de los recursos comunes del grupo, concretamente unos portátiles y un cañón de proyección. En este sentido, cuando dos o más miembros del grupo solicitan uno de estos recursos, no es inmediato determinar cuál de ellos goza de una mayor prioridad, y por tanto disfrutará del recurso.

La intención inicial de esta parte del trabajo consiste por tanto en dotar a este grupo de investigación de un sistema multiagente, fácilmente ampliable, que ayude a sus miembros a encontrar la fecha y hora más adecuada para una reunión interna, así como regular el empleo de los recursos comunes. Consideramos este problema de alto interés como aplicación para un sistema multiagente por las siguientes razones:

1. El gran número y variedad de agentes involucrados en la arquitectura del sistema, en especial un agente de ontología.

2. Las tareas de planificación de reuniones internas y de negociar acerca de los recursos comunes son significativamente diferentes.

3. La posibilidad de que cada usuario dote a su agente correspondiente de una estrategia distinta enriquece al sistema, alejándole de una mera resolución distribuida de problemas.

4. Los agentes relativos a varios miembros del grupo pueden encontrarse simultáneamente en una negociación. Del mismo modo, un agente concreto puede estar involucrado al mismo tiempo en varias negociaciones diferentes.

5. El empleo de un lenguaje de marcas de alta riqueza semántica, (en este caso se va a emplear DAML+OIL), dota al sistema de un mayor poder.

Como en el caso de los antecedentes analizados en la sección 4.2, el MAS implementado tiene un ámbito de aplicación concreto[14], no tan ambicioso si lo comparamos con el de los *Electric Elves*. No obstante, pretende ser un paso de

[14] El flujo de visitas externas al grupo CyC no es suficientemente significativo como para considerar su inclusión en el diseño de MASplan.

evolución mayor dentro de los sistemas multiagentes al emplear las herramientas de última generación para la representación de la información.

En los siguientes apartados describiremos en detalle la implementación realizada para MASplan. No obstante, debemos hacer notar al lector que la verdadera importancia no consiste tanto en los algoritmos diseñados (aunque en este sentido se han realizado aportaciones en el tratamiento de escenarios similares) sino en mostrar el inmenso poder que los sistemas multiagentes, en combinación con los lenguajes basados en marcas de representación de ontologías, pueden aportar a éste y a otros tipos de problemas. En este sentido, el texto está repleto de referencias a futuras implementaciones del MAS que doten de una mayor riqueza aún al sistema. En el diseño se ha perseguido que todas estas ampliaciones sean lo más fácilmente integrables en el código, mediante una estructura de clases lo más general posible.

4.4. Elección de Herramientas

Tras un análisis del problema y del escenario del Grupo de Investigación CyC, pasamos en esta sección a justificar la elección de herramientas empleadas para la implementación del sistema de agentes.

4.4.1. Marcos de trabajo MAS

Tal como se ha descrito en la sección 3.8, existen numerosas herramientas para el diseño e implementación de sistemas multiagentes. Esto hace que no sea preciso empezar el sistema desde cero y permitiéndonos olvidar, por ejemplo, la implementación de los detalles concretos de comunicación. El primer paso consiste pues, en, a partir de las especificaciones del problema, decidir el tipo de herramienta de plataforma de agentes (basada en lenguaje KQML, estructura FIPA, arquitectura OMG MASIF, agentes móviles...). Se apostó desde el principio por un sistema multiagente sometido a algún estándar existente, lo cual favorecería la inclusión de otros agentes diseñados incluso por otros programadores. La alta calidad de las comunicaciones dentro de la Universidad (ámbito donde se van a mover los usuarios, y por tanto el sistema) nos hizo considerar innecesario el empleo de agentes móviles, puesto que no haría falta el desplazamiento físico del agente en la red. No obstante, el sistema diseñado debería poder aceptar la inclusión de dichos agentes, lo cual sería necesario en caso de un agente inicializado externamente a la red universitaria (por ejemplo, algún miembro de grupo en estancia en otra universidad) o ante un colapso de la red (el agente se desplazaría hacia una localización que le permitiese una mejor comunicación).

De los tres estándares (KQML, FIPA, OMG MASIF), se abandonó en primer lugar la opción de un sistema multiagente basado en KQML, puesto que el estándar se restringe únicamente al lenguaje de comunicación entre los agentes. La no idoneidad del KQML se refleja en las diversas estructuras-arquitecturas de las herramientas analizadas en la sección 3.8 (JATLite, Jackal), lo que siempre dificulta la compatibilidad. En aras de una mayor estandarización, la herramienta para la implementación de agentes debería cumplir con los estándares FIPA, ampliamente aceptados y perfectamente documentados, relativos no únicamente al lenguaje, sino

a casi todos los aspectos de un sistema multiagente: protocolos de comunicación, arquitectura de plataforma, sistema de transporte de mensajes... Este primer punto ya supone una evolución en cuanto a lenguajes de agentes respecto a los antecedentes analizados (correo electrónico en *Visitorbot*, KQML entre los *Teamcore* de los *Electric Elves*). Del mismo modo supone un paso más allá, al emplear una arquitectura más estándar, lo que proporciona a nuestro sistema una mayor robustez (avalado por una organización como FIPA) y facilidad de inclusión de nuevos agentes. Aparte de la extensa implantación y completitud de sus estándares, la opción FIPA parece en el problema en cuestión más adecuada que la OMG MASIF, ya que no se van a emplear, tal como se ha aclarado con anterioridad, agentes móviles de modo principal.

Una vez decidida la arquitectura de gestión del sistema, nos resta elegir la herramienta adecuada para su implementación. Las herramientas multiagente FIPA posibles son FIPA-OS, JADE, Zeus y Grasshopper (conviene recordar que esta última, aunque diseñada inicialmente para OMG MASIF se ha adaptado para el estándar FIPA). Todas estas herramientas son multiplataforma, al encontrarse implementadas en el lenguaje de programación Java. Por tanto, cualquiera de ellas permite tratar el hecho de que los usuarios del grupo de investigación accedan a la red interna a través de diferentes sistemas operativos.

De estas herramientas, hemos seleccionado FIPA-OS. Aparte de cumplir satisfactoriamente con las especificaciones FIPA [Lau00a], este marco de desarrollo presenta una estructuración de código por tareas. Esta estructura implica una depuración de código sencilla, puesto que los métodos relativos a la realización de una misma tarea se incluyen bajo el mismo objeto de la clase *Task*. De este modo, la programación se basa en el tratamiento de los actos de comunicación de los mensajes recibidos en cada conversación, facilitado por la herramienta de generación de la estructura del código, incluida en la distribución FIPA-OS.

Un ejemplo de estructura de código FIPA-OS es el siguiente:

```
private class IdleTask extends Task
{
/* constructor*/
 public IdleTask()
 {...
 }
/* método que se invoca al inicio de la tarea */
 public startTask()
 {...
 }
/* método de manejo de los mensajes request */
 public void handleRequest(Conversation id)
 {...
 }
/* método de manejo del resto de mensajes en la tarea*/
 public void handleOther(Conversation id)
 {...
 }
```

```
}
```

Como ventajas adicionales, FIPA-OS proporciona el código fuente en una organización de clases por paquetes estructurada de forma lógica.

En el resto de herramientas, la codificación se vuelve más complicada. JADE proporciona una codificación menos estructurada. Así, por ejemplo, bajo el método de inicialización del agente `setup()` se podrían agrupar todas las tareas del agente. Además, JADE no proporciona una versión completamente fiable del código fuente, por lo que no podrían realizarse con garantías suficientes las modificaciones necesarias.

Zeus proporciona el código fuente. No obstante, la programación en Zeus, orientada a metas y estructurada a través de nodos y arcos, se lleva a cabo normalmente mediante interfaces gráficas, las cuales encorsetan al usuario a una estructura de relleno de campos que en ocasiones no se adapta a las necesidades del problema a tratar.

Por último, Grasshopper no proporciona el código fuente. Solamente proporciona dos plantillas para la codificación de agentes (una básica llamada *SimpleAgent* y otra *SmartAgent* con métodos adicionales referidos al traslado del agente a otra plataforma y espera para realizar una tarea). Los métodos y sus invocaciones se basan en el ciclo de vida del agente (ver sección 3.5.2.7). De este modo tenemos métodos `live`, `beforeMove`... no tan útiles desde el punto de vista práctico como la estructura de FIPA-OS.

Aparte de esta sencillez de codificación con FIPA-OS, la razón principal para elegir FIPA-OS es la dedicación de un *thread* por tarea en vez de por agente, como en el resto de herramientas. Esto dota de una mayor robustez al sistema, puesto que evita que un posible bloqueo del *thread* inutilice completamente al agente. No obstante, en contraposición, esto supone un aumento de los *threads* del sistema. Esto no debe suponer un gran coste en cuanto a recursos dentro del sistema, puesto que no es probable que coexistan simultáneamente varias negociaciones inicializadas por usuarios distintos, por lo que el número de *threads* se debe mover en límites razonables. En este sentido debemos recordar además que en la implementación escogida, estos *threads* se podrán repartir entre varios procesadores.

En cuanto a la capacidad de depuración del flujo de mensajes, todas las herramientas cuentan con utilidades equivalentes. En el caso de FIPA-OS, el usuario puede emplear la clase DIAGNOSTICS, que permite almacenar en un fichero de texto los mensajes producidos en el sistema. Como última ventaja, destacamos la definición de perfiles XML, lo cual permite, de una forma sencilla, cambiar la configuración del sistema (por ejemplo la lista de los agentes a inicializar) sin necesidad de recompilar código.

La herramienta FIPA-OS también cuenta con una interfaz para la herramienta de razonamiento *Jess*, por lo que se valoró su posible empleo en este MAS. No obstante, para esta versión inicial, se consideró que era innecesario su empleo, debido a la sencillez de los algoritmos a implementar. Esto no quiere decir que se descarte el empleo de esta interfaz para futuras implementaciones.

El punto débil de esta herramienta es el tratamiento del ciclo de vida de los agentes del sistema, puesto que no proporciona utilidad alguna al respecto. Sin embargo, esto no supone restricciones en el sistema diseñado, puesto que se supone, debido a la naturaleza del sistema, que todos los agentes se encuentran activos (dispuestos a recibir nuevos mensajes provenientes de otros agentes) o se han destruido, circunstancia

asumible perfectamente mediante la herramienta.

4.4.2. Herramientas de Ontologías

De la sección 3.6.2 se deducen las ventajas de la inclusión de un agente de ontologías dentro del sistema multiagente. De hecho, el propósito perseguido en todo nuestro trabajo de mostrar la versatilidad y poder de los sistemas multiagentes en general, y de la arquitectura seleccionada en particular, justifica la inclusión de este tipo de agentes. En el caso de la planificación de agendas, nos permite la definición jerarquizada de los conceptos involucrados en el sistema: los usuarios, instancias de material y demás elementos del sistema. Asimismo pueden definirse axiomas y otras relaciones entre los conceptos que doten de capacidad de inferencia a los agentes. Este punto supone otra diferencia clara respecto a los otros sistemas analizados, puesto que no se aprovechan de modo alguno de la fuerza de los lenguajes de marcas ni del empleo de ontologías (lo más parecido en este sentido es el sistema *PHOSPHORUS Capability Matcher*[15] en los *Electric Elves*).

La primera decisión a tomar sobre la inclusión de ontologías estriba en el lenguaje en que se representan éstas. Sería deseable que este lenguaje ofreciese la mayor riqueza semántica posible, y en particular, el poder de los lenguajes basados en marcas. En este sentido, el candidato más adecuado en la actualidad es el OWL. Sin embargo, tal como se ha indicado en el capítulo relativo a las ontologías, a fecha de redacción de este trabajo no existen herramientas (especialmente un razonador) capaces de procesar adecuadamente las características de riqueza semántica del OWL (concretamente el OWL Full), aunque actualmente se trabaja en ello[16].

No obstante, el presente trabajo fue iniciado con anterioridad al lanzamiento de la especificación del OWL, por lo que se eligió como lenguaje de ontología el DAML+OIL. Esta elección cuenta, aún a fecha de escritura de este trabajo, con la ventaja de las numerosas ontologías escritas en DAML+OIL existentes en Internet.

La riqueza semántica de DAML+OIL es suficiente (de hecho se considera equivalente a la de OWL [HP 03]) para demostrar el poder de este tipo de lenguajes de marcas en su integración en sistemas multiagentes mediante el empleo de ontologías. Por este motivo, se ha optado por no migrar las ontologías y las rutinas de procesamiento a OWL. No obstante, una de las líneas abiertas en nuestra investigación consiste en la conversión de las ontologías DAML+OIL a OWL, empleando para ello la herramienta *OWL Converter* y migrando el código de las clases implementadas mediante las nuevas herramientas que puedan surgir (por ejemplo, la nueva versión de Jena, lanzada el 28 de agosto de 2003).

La decisión de tomar un lenguaje de ontologías basados en marcas, lleva implícita la búsqueda de tres herramientas:

- Una herramienta de edición de la ontología de una forma gráfica (aunque el resultado de la edición es un fichero de texto, y por tanto, editable en un procesador tan simple como el bloc de notas). De las tres opciones analizadas: *OntoEdit*, *OilEd* y *Protégé-2000*, la primera de ellas se desechó por ser un programa comercial y tener la versión gratuita limitaciones en cuanto al número

[15] un casador de capacidades

[16] http://www.w3.org/2001/sw/WebOnt/impls

de elementos en la ontología. Las otras dos (ambas implementadas en Java) presentan características de edición similares. Protégé-2000 es de código libre, pero se ha escogido en este caso la herramienta OilEd por dos motivos. En primer lugar, la capacidad de editar DAML+OIL viene en la distribución estándar (por tanto no es necesario incluir ningún *plug-in*). Además incluye la posibilidad de procesar directamente la ontología mediante el razonador FaCT.

- Un razonador que verifique la estructura de la ontología editada. En este punto, se han aplicado todos los razonadores vistos en la sección 2.5, esto es, *FaCT* (incluido en la distribución de OilEd), *DAML+OIL Ontology Checker* y *DAMLValidator*. De este modo nos aseguramos, aunque de forma redundante en principio, de detectar completamente las posibles inconsistencias de la ontología.

- Un *parser* que permita extraer información de la ontología resultante. En este sentido es conveniente que esta herramienta estuviese implementada en Java, ya que de este modo la integración con la herramienta FIPA-OS (implementada en ese lenguaje) será más fácil. Con estas premisas, nos hemos decantado por Jena. Además, de este modo, prevemos que la transición a las nuevas versiones de esta herramienta, que incluyen soporte para el lenguaje OWL, sea asimismo no excesivamente compleja.

Conviene aclarar que en el sistema, el DAML+OIL no se emplea únicamente con el propósito de ontología, sino que se utilizará para almacenamiento de datos (guardando información necesaria para el funcionamiento del sistema) pero con la misma estructura de la ontología. De este modo se almacena, por ejemplo, la agenda de cada usuario.

4.4.3. Lenguaje de Programación

Una vez decidido que el desarrollo del sistema de agentes se va a realizar en el lenguaje de programación Java, nos queda seleccionar el entorno de compilación de las clases implementadas. En este sentido, a pesar de la existencia de entornos integrados de desarrollo de código Java (por ejemplo el *JBuilder* de *Borland*), se ha empleado el kit para desarrolladores distribuido por *Sun Microsystems*. Este kit es el JDK (*Java Development Kit*), que a fecha de escritura de este trabajo se encuentra en la versión 1.4.2. Los motivos de emplear este kit radican, por una parte, en nuestra política de evitar, en la medida de lo posible, herramientas comerciales, y por otro lado, por ser *Sun Microsystems* la empresa desarrolladora del lenguaje.

El JDK contiene todas las clases básicas que permiten un desarrollo de una aplicación en Java. No obstante, *Sun* complementa este kit con una serie de librerías (APIs) adicionales. En el desarrollo de este sistema multiagente ha sido necesario incluir las librerías relativas al correo electrónico (JavaMail) y a la encriptación (Java Cryptography Extension, JCE).

4.5. Arquitectura de agentes

Aparte de los elementos de la arquitectura derivados de la utilización de la Plataforma de Agentes FIPA, esto es, el Agent Management System (AMS), el Directory Facilitator (DF) y el Message Transport Service (MTS), MASplan se compone de los siguientes ocho tipos de agentes, los cuales pasamos a describir de forma breve a continuación.

Agente de Usuario (*UA*) Este agente sirve de interfaz con usuario mediante interacción gráfica, mostrando la agenda relativa al usuario y permitiendo que éste pueda realizar todas las interacciones con el sistema (por ejemplo, solicitar una reunión con otros miembros del grupo o la reserva de un recurso común). Cuando esto ocurre, el UA trata de localizar al agente de negociación correspondiente al mismo usuario y transmitirle las especificaciones deseadas por el usuario. Una vez que el negociador ha finalizado su trabajo, el agente de usuario recibe el resultado y lo muestra al usuario, avisándole de que se ha producido un cambio en su agenda. Este aviso también se produce cuando el cambio se produce fruto de la negociación iniciada por otro usuario CyC. Como es lógico pensar, cada miembro de CyC cuenta con su propio UA en el sistema global de agentes.

Otra función es ofrecer al usuario un módulo de entrenamiento para permitir a su agente de negociación aprender (como se verá posteriormente a través del Agente de Reglas) si el usuario se mostraría o no propicio a un futuro cambio de su agenda.

Agente Negociador (*NA*) Este agente y el anterior abarcan el concepto de agente asistente personal FIPA visto en la sección *3.6.1*. Se ha preferido separarlo en dos agentes para repartir la carga entre ellos (uno negocia y el otro muestra los resultados al usuario), ya que de forma general se ejecutarán en máquinas diferentes. En concreto, el NA es el encargado de implementar los algoritmos seleccionados para la negociación de reuniones internas y petición de recursos comunes. Cuando el UA correspondiente al mismo usuario solicita la negociación para fijar una reunión, el NA realiza una búsqueda mediante el DF de la AP[17] para localizar los NAs del resto de usuarios involucrados, comenzando entonces el proceso de negociación. De un modo análogo, en el caso de la petición de un recurso común, se solicita al DF la localización del agente encargado de gestionar los recursos comunes del grupo. En ambos casos, una vez finalizada la negociación, solicita al agente de Mail (en caso de encontrarse activo) el envío de un correo electrónico al usuario, guarda los cambios en el fichero-agenda del usuario (implementado, como el caso de la ontología, en DAML+OIL) y comunica el resultado al UA. Cada miembro de CyC cuenta con su propio NA. Estos ficheros-agenda se encuentran en el sistema de ficheros del grupo. Como línea abierta de mejora del MAS, puede incluirse un nuevo agente AgenteFichero encargado de leer/escribir estos ficheros. De este modo, sería este último agente el que debería inicializarse en el sistema

[17] De un modo más formal, se pregunta al DF por agentes registrados capaces de negociar en nombre de los usuarios involucrados.

de ficheros del grupo y, como consecuencia, los NAs podrían inicializarse en computadores ajenos a dicho sistema de ficheros.

Agente de Ontología (*OA*) La función de este agente es la reflejada en la sección 3.6.2, y es uno de los puntos clave, que lo diferencia de las otras aplicaciones vistas en este capítulo. Al registrarse en el DF, descubre al resto de los agentes la ontología global. Asimismo, suministra información sobre la ontología de grupo cuando así lo solicitan el resto de agentes. Esta ontología proporciona, por ejemplo, la lista completa de usuarios y de recursos disponibles.

Como se explicó en la misma sección, la declaración explícita de ontologías lleva consigo una serie de ventajas para las aplicaciones: permite la consulta sobre conceptos, la actualización y reutilización de ontologías, la interoperabilidad de MAS heterogéneos. La existencia de OA hace innecesario que cada agente de forma individual contenga la ontología en su totalidad, con el consiguiente ahorro de recursos. Además, el uso de la ontología permite acciones como la inclusión de un nuevo usuario CyC o recurso común sin tener que compilar de nuevo el código. Se ha optado por un único OA en la estructura debido al empleo de una única ontología de una complejidad no demasiado elevada (tal como se verá posteriormente en la sección 4.6) que será accedida de forma puntual. Por lo tanto un único OA debería ser capaz de soportar el sistema. El OA es el único agente con permiso de acceso a la ontología del grupo CyC. De este modo, se evitan posibles problemas derivados de accesos y actualizaciones simultáneas. Además se prepara al sistema a situaciones donde el resto de los agentes no pueden, por ejemplo por restricciones en el servidor, acceder a la ontología.

Agente de Recursos Materiales (*RMA*) Este agente es invocado cuando se produce la petición de un recurso común del grupo (a fecha de escritura de este trabajo se resumen en un cañón de proyección y dos portátiles). El RMA también se encarga de almacenar los cambios producidos en el fichero-agenda del recurso correspondiente, codificados en DAML+OIL, de modo equivalente al de un usuario. Solamente existe un único RMA en la arquitectura, puesto que éste debería ser capaz de tratar con los tres recursos comunes al grupo. No obstante, nada impide que el sistema implique a más de un RMA.

Agente de Mail (*MA*) Cuando se confirma un cambio en la agenda de algún usuario, el NA correspondiente solicita al MA que envíe un correo electrónico con el cambio producido a través del servidor de correo del grupo CyC. Esta comunicación se realiza en condiciones de seguridad mediante el protocolo SSL. A nivel práctico se ha creado una nueva cuenta de correo, perteneciente a un usuario ficticio denominado *MASplan*. Esta cuenta, como se verá más adelante, también jugará un papel importante en el entrenamiento de los NAs. Se ha optado por la inclusión de un único MA debido a que la estimación del flujo de mensajes en el sistema, no demasiado elevado, así lo recomienda. La comunicación a través de correo electrónico se emplea en los antecedentes vistos en la sección 4.2, pero MASplan ofrece la novedad de la existencia de un agente específico para mandar los correos. Este hecho supone una mayor especialización en los papeles a jugar por los agentes.

En este punto, comentaremos que durante la implementación del sistema se pensó en, aparte del correo electrónico, enviar un mensaje de texto SMS a los teléfonos móviles de los usuarios involucrados. Esta idea fue abandonada por dos motivos. En primer lugar, los problemas de instalación del paquete correspondiente en el servidor del grupo CyC. La otra razón, quizás de más peso, las reticencias de los miembros CyC ante lo que se consideraba una disminución en su privacidad, a pesar de la existencia de otros trabajos en esta línea como el sistema GPS en los *Electric Elves*. De todos modos, y aunque no se ha implementado esta acción, el sistema multiagente diseñado permite su futura inclusión.

Agente de Reglas (*RA*) Este agente es consultado (en caso de existir en el sistema) cada vez que se plantea un cambio de agenda para una reunión. El NA consulta a este agente sobre si el usuario se ha mostrado con anterioridad poco propicio al cambio de agenda. Para ello emplea árboles de identificación con la simplificación de Fisher sobre los datos (la implementación realizada se muestra en la sección 4.10), ya sean reales o provenientes de un entrenamiento. En ambos casos, el dato es almacenado inicialmente como correo electrónico en la cuenta ficticia *MASplan*. El RA lee esa cuenta de correo, actualiza los árboles de identificación, borra los correos correspondientes y graba el nuevo árbol seriándolo en un fichero. De esta forma, en caso de reinicialización del sistema multiagente, los datos, y por tanto las reglas deducidas no se pierden. Se incluye un único RA en el sistema, puesto que se considera que será capaz de procesar la información de todos los agentes del sistema. Una vez más, la versatilidad del sistema hace que este patrón de diseño sea fácilmente modificable.

Agente Cargador (*CA*) Este agente tiene una vida corta en relación con el resto de los agentes, puesto que su función consiste únicamente en localizar al OA (mediante el DF), consultarle los nombres de los usuarios existentes en la ontología de grupo y crear los NAs de esta lista de usuarios. Su labor, por tanto, tiene lugar en la inicialización del sistema (esto se declara en el fichero de perfil de la AP). Se puede optar por un único CA o por varios en el caso en que se desee distribuir los NAs entre varias máquinas virtuales.

Agente de Comprobación (*CoA*) La vida de este agente en el sistema, como en el caso del CA, también es de corta duración. Su cometido es el gestionar la inicialización de un UA. Al solicitar un usuario CyC la creación de su UA, realmente lanza un CoA que pregunta al usuario por su *login* y *contraseña*. El CoA comprueba esta contraseña (empleado el API de encriptación de *Sun*) y en caso positivo, inicia el UA, invocando el método de destrucción de un UA del mismo usuario registrado de forma previa en el sistema, en caso de existir.

En la Figura 4.11 se muestran las relaciones existentes entre los agentes principales del sistema (los seis primeros). Debemos reiterar que aunque se han implementado los tipos descritos, el sistema se encuentra abierto a la inclusión de otros agentes, por ejemplo, agentes negociadores implementados con otra filosofía por otros programadores.

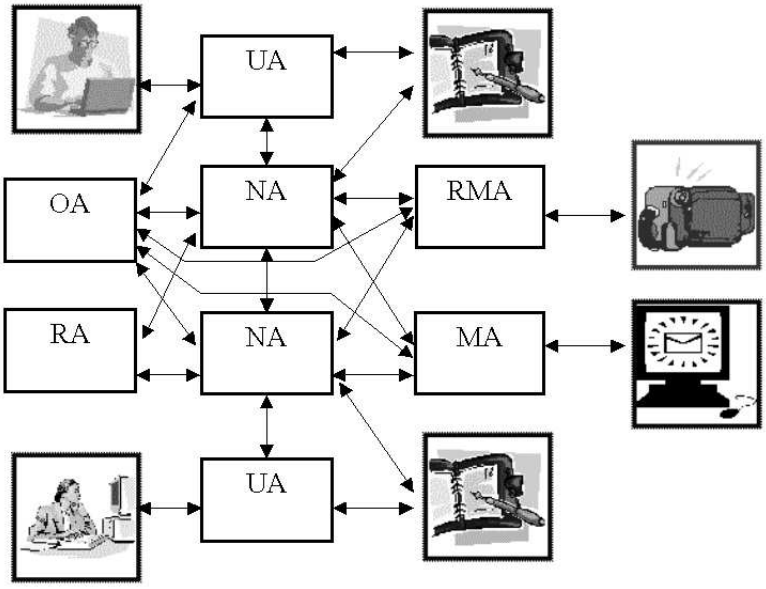

Figura 4.11.: Arquitectura MASplan para la planificación de agendas.

4.6. Diseño de Ontología

En el diseño de la ontología a emplear por el sistema multiagente se han seguido, en el mayor grado posible, los pasos descritos en la sección 2.2.6: adquisición del conocimiento del dominio, organización de la ontología, desarrollo, comprobación y remisión a expertos en el dominio. Este último punto ha sido llevado a cabo por el doctorado y su director, los cuales poseen un alto conocimiento en el dominio de la aplicación, esto es, el Grupo CyC y su capacidad de negociar sobre la organización de reuniones y gestión de los recursos comunes.

En primer lugar, y antes de los puntos descritos en el párrafo anterior, se centró el problema en determinar el objetivo que debería tener la ontología, así como el grado de detalle de la misma. Debido a la naturaleza del problema (empleo de la ontología para un problema concreto dentro del sistema multiagente), la ontología será del tipo de aplicación. Del mismo modo, y debido a que su empleo se fundamenta en mostrar las capacidades de los OAs en un sistema multiagente, tampoco es necesario un grado de axiomatización elevado (la cuestiones básicas como, por ejemplo, la definición estricta de *Usuario* se encuentra codificada en cada agente, pero sí se incluye en la ontología que un *Profesor* es un *Usuario*[18]). Por tanto, nuestra ontología será de grano grueso, o compartible (ver sección 2.2.4).

En un momento dado, se pensó en aumentar la compatibilidad de la ontología, haciendo derivar sus términos de una ontología estándar como la representación de la OpenCyc en DAML+OIL. Sin embargo, debido al tamaño del fichero resultante (el sistema debería cargar esta ontología al iniciarse), esta opción fuese desechada. La inclusión de esta ontología hubiese supuesto una carga innecesaria en el sistema al consumir una gran cantidad recursos en el almacenamiento de términos que no se van

[18] concretamente que *Profesor* es una subclase de *Usuario*

a emplear. Conviene en este punto recordar la premisa relativa a la eficiencia de la negociación (ver sección 4.1.1).

Otro motivo para la no inclusión de esta ontología es que introduce un grado de detalle excesivo e innecesario en la jerarquía de conceptos para el cometido del sistema multiagente. Como ejemplo de este grado, obsérvese el siguiente código relativo a una hora[19]:

```
<daml:Class rdf:ID="CalendarHour">
  <rdfs:label xml:lang="en">calendar hours</rdfs:label>
  <rdfs:comment>An instance of #$CalendarCoveringType, and a
   specialization of #$Date. Each instance of #$CalendarHour
   is an hour in some particular calendar. Instances of
   #$CalendarHour include (#$HourFn 12 (#$DayFn 20 (#$MonthFn
   #$January (#$YearFn 1965)))) and (#$HourFn 13 (#$DayFn 13
   (#$MonthFn #$July (#$YearFn 2000)))).</rdfs:comment>
  <guid>bd58933b-9c29-11b1-9dad-c379636f7270</guid>
  <rdf:type rdf:resource="#PublicConstant-DefinitionalGAFsOK"/>
  <rdf:type rdf:resource="#PublicConstant-CommentOK"/>
  <rdf:type rdf:resource="#ConventionalClassificationType"/>
  <rdf:type rdf:resource="#PublicConstant"/>
  <rdf:type rdf:resource="#CalendarCoveringType"/>
  <rdfs:subClassOf rdf:resource="#TimeOfDay"/>
  <rdfs:subClassOf rdf:resource="#Date"/>
  <daml:disjointWith rdf:resource="#TimePoint"/>
</daml:Class>
```

Esto no debe suponer una pérdida de compatibilidad con agentes externos, siempre que estos últimos hagan referencia a la misma ontología. Otra posible solución para asegurar la compatibilidad consistiría en la implementación de otro OA encargado de traducir los términos entre las ontologías involucradas.

4.6.1. Definición de usuarios CyC

Una vez anotadas estas consideraciones previas, se considera en primer lugar la definición de los distintos usuarios del grupo CyC. Para ello se define la clase *Usuario*, de la cual derivan una serie de subclases que reflejan los diferentes papeles principales existentes en el grupo: *Profesor*, *Becario* y *Técnico*[20], así como la posibilidad de la inclusión en el sistema multiagente de un usuario externo a CyC (clase *UsuarioExterno*). Tal como se puede comprobar a la vista de la Figura 4.12, se han definido una serie de subclases dentro de *Profesor*, buscando definir con mayor rigor a los usuarios. En esta ontología, la clase *Catedratico* es una subclase de *Titular* (clase referida a un Profesor Titular de Universidad), la cual a su vez es una subclase de *ProfesorDoctor*. Esta clasificación no impide, debido a la capacidad de definición de la herencia múltiple en DAML+OIL, que, por ejemplo, una instancia

[19] Obsérvese la gran cantidad de código involucrado en la definición del concepto

[20] perteneciente al servicio técnico

UsuarioExterno, sea a la vez instancia de la clase *ProfesorDoctor*. El siguiente código corresponde a la definición de *Titular*.

```
<daml:Class rdf:about= "#Titular">
  <rdfs:label>Titular</rdfs:label>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#ProfesorDoctor"/>
  </rdfs:subClassOf>
</daml:Class>
```

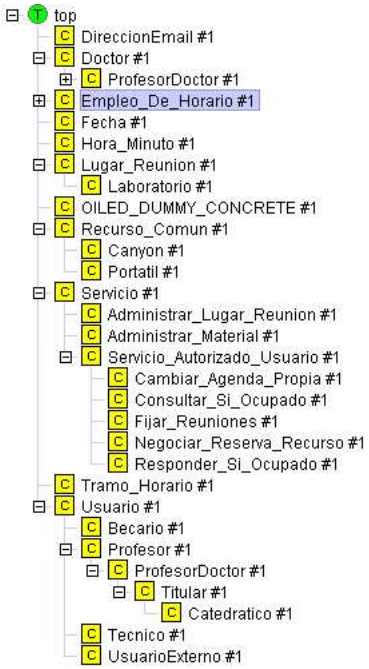

Figura 4.12.: Jerarquía de clases.

Por tanto, esta especificación permite calificar, en principio, a cada uno de los usuarios CyC como miembro de una o varias de estas clases. Para refinar la ontología, a partir de estas definiciones, se ha aprovechado la riqueza semántica del DAML+OIL para incluir una serie de axiomas (ver Figura 4.13) relativos a estas clases, por ejemplo, que un individuo de la clase *Becario* no puede ser instancia a la vez de la clase *Profesor*.

```
<daml:Class rdf:about= "#Becario">
  <daml:disjointWith>
    <daml:Class rdf:about= "#Profesor"/>
  </daml:disjointWith>
</daml:Class>
```

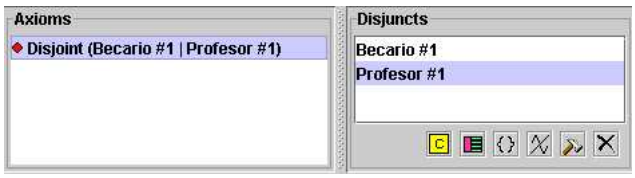

Figura 4.13.: Declaración de axiomas con OilEd.

La utilidad de esta clasificación se manifiesta en la posibilidad, por ejemplo, de localizar en el MAS todos aquellos individuos que sean profesores (para ello se haría uso de la propiedad de subclase), o de organizar una reunión en la que debe asistir al menos un miembro del servicio técnico.

Haciendo uso del editor de ontologías OilEd, se implementaron las instancias relativas a los miembros del grupo CyC. Como nombre de estas instancias se ha tomado el empleado normalmente dentro del grupo para referirse a los usuarios, por ser éste el ámbito donde se pone en marcha el MAS. Así, la instancia del profesor Evelio González es simplemente *Evelio*. En futuras versiones de la ontología, una vez migrada al lenguaje OWL, se puede emplear la propiedad `owl:sameIndividualAs` para poder referenciar a un usuario por todos sus referentes conocidos, enriqueciendo de este modo la ontología. En este sentido, por ejemplo, los usuarios *Alberto* y *Hamilton* se entenderán en el sistema como un único usuario.

La definición de los usuarios es uno de los puntos donde se comprueba la ventaja de emplear una ontología en un escenario como el analizado. Es normal que el conjunto de usuarios sufra cambios. Ejemplos de estos cambios son nuevas incorporaciones o cambios en la condición de cada usuario (de *Profesor* a *ProfesorDoctor*...). Para que el sistema admita estos cambios, solamente es necesario redefinir la ontología, no el código de la implementación del sistema.

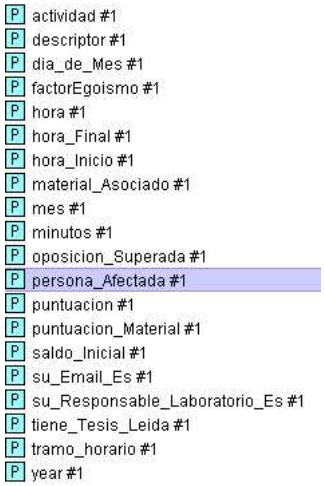

Figura 4.14.: Jerarquía de propiedades.

La información de cada usuario se completa mediante la definición de una serie

de propiedades (la lista de las propiedades definidas en la ontología se muestra en la Figura 4.14). Una de las propiedades definidas es *su_Email_Es*, la cual se refiere al alias de cada usuario dentro del dominio de correo electrónico del grupo CyC. Para definir el rango de esta propiedad se incluye una nueva clase denominada *DireccionEmail*[21]. Esta definición permite definir instancias de dicha clase que registran los alias de cada usuario.

```
<rdf:RDF
...
xmlns:ns0="http://masplan.cyc.ull.es/AgendaGrupo6.daml#">
<rdf:Description rdf:about= "#Evelio">
  <rdf:type>
    <daml:Class rdf:about= "#Profesor"/>
  </rdf:type>
  <ns0:su_Email_Es rdf:resource= "#evelio"/>
</rdf:Description>
<rdf:Description rdf:about= "#evelio">
  <rdf:type>
    <daml:Class rdf:about= "#DireccionEmail"/>
  </rdf:type>
</rdf:Description>
```

La propiedad *tiene_Tesis_Leida* (de dominio *Usuario* y rango *boolean*) permite definir la clase *Doctor*, como la subclase formada por todas las instancias individuales que presenten un valor *true* para esta propiedad. Estas definiciones permiten perfilar la definición de *ProfesorDoctor* por herencia múltiple de *Profesor* y *Doctor*.

```
<daml:Class rdf:about= "#ProfesorDoctor">
  <rdfs:label>ProfesorDoctor</rdfs:label>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#Profesor"/>
  </rdfs:subClassOf>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#Doctor"/>
  </rdfs:subClassOf>
</daml:Class>
```

De un modo análogo, la propiedad *oposicion_Superada* permite refinar el concepto de *Titular*.

Estas clases llevan implícitas la definición de una serie de axiomas (por ejemplo, *Catedrático* lleva implícito ser *Doctor*), cuya consistencia ha sido comprobada mediante los razonadores seleccionados.

[21] Podría haberse definido simplemente como *Literal*, pero de este modo ayuda a aumentar la consistencia de la ontología global.

4.6.2. Definición de recursos comunes

Un segundo grupo de definiciones se refieren a los recursos comunes del grupo CyC. Se define la clase *Recurso_Comun* de la cual derivan las clases relativas a los dos tipos de recursos considerados hasta el momento: *Cañón*[22] y *Portatil*. Como en el caso de los usuarios, se definen a continuación las instancias relativas a cada uno de los conceptos.

```
<daml:Class rdf:about= "#Recurso_Comun">
  <rdfs:label>Recurso_Comun</rdfs:label>
</daml:Class>
<daml:Class rdf:about= "#Portatil">
  <rdfs:label>Portatil</rdfs:label>
  <rdfs:comment>Un ordenador portátil</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#Recurso_Comun"/>
  </rdfs:subClassOf>
</daml:Class>
<rdf:Description rdf:about= "#Portatil_1">
  <rdf:type>
    <daml:Class rdf:about= "#Portatil"/>
  </rdf:type>
</rdf:Description>
```

Aunque no se ha empleado en la implementación del sistema multiagente, se ha incluido en la ontología para futuras versiones un tercer tipo de recurso referido a los posibles lugares de reunión gestionados por los usuarios CyC, con la clase *Lugar_Reunion*. Las instancias de esta clase (referidas a tres laboratorios y una sala-biblioteca) podrían ser seleccionadas como un recurso más a negociar (para ello debe modificarse la ontología, haciendo a *Lugar_Reunion* una subclase de *Recurso_Comun*) o emplearse, en una versión más refinada, como un parámetro más en la planificación de reuniones. Para el caso particular de los laboratorios, se incluye la definición de la propiedad *su_Responsable_Laboratorio_Es*, la cual indica el usuario CyC encargado de la gestión y organización de dicho laboratorio[23].

```
<rdf:Description rdf:about= "#Laboratorio_B">
  <rdf:type>
    <daml:Class rdf:about= "#Laboratorio"/>
  </rdf:type>
  <ns0:su_Responsable_Laboratorio_Es rdf:resource= "#Jose_Luis"/>
</rdf:Description>
<daml:ObjectProperty rdf:about= "#su_Responsable_Laboratorio_Es">
  <rdfs:label>su_Responsable_Laboratorio_Es</rdfs:label>
```

[22] Por limitaciones de la herramienta, no se puede emplear la 'ñ'. Por esto, la clase se define como *Canyon*.

[23] La inclusión de este concepto permitiría, en su caso, emplear al MAS para mandar un correo electrónico a este responsable, avisándole que se va a emplear el laboratorio a su cargo para llevar a cabo una reunión.

```
    <rdfs:domain>
      <daml:Class rdf:about= "#Laboratorio"/>
    </rdfs:domain>
    <rdfs:range>
      <daml:Class rdf:about= "#Profesor"/>
    </rdfs:range>
  </daml:ObjectProperty>
```

4.6.3. Definición de ocupaciones de horario

Otra serie de definiciones se refieren a las ocupaciones posibles para los usuarios CyC en los diferentes tramos horarios de su agenda (la jerarquía implementada relativa a esta parte se muestra en la Figura 4.15). La superclase *Empleo_De_Horario* presenta una subclase *Horario_Prioritario* referida a aquellas ocupaciones difícilmente aplazables como pueden ser clases, exámenes, oposiciones, viajes... Otras clases referidas a ocupaciones son *ReunionProyecto*, *PrepararClases, NadaDefinido*, *OtroMotivo*...

```
<daml:Class rdf:about= "#Preparar_Paper">
  <rdfs:label>Preparar_Paper</rdfs:label>
  <rdfs:comment>Preparar un articulo</rdfs:comment>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#Empleo_De_Horario"/>
  </rdfs:subClassOf>
</daml:Class>
```

Se puede discutir que las referencias a estas ocupaciones hayan sido implementadas como clases y no como instancias individuales de las dos superclases. En este sentido, la justificación consiste en que, al definirla como clases (un concepto más general), podemos, en un futuro, establecer una jerarquía de superclases entre ellas, así como definir instancias (un concepto más particular) de ellas. Por ejemplo, *PrepararPaperJornadasAutomatica2003* como una instancia de *Preparar_Paper*.

4.6.4. Definición de tramos horarios

Otro grupo de definiciones se deriva del modo diseñado para la gestión de las agendas de usuario. La clase *Fecha* representa a un día de la agenda de usuario. Esta clase se perfila, a su vez, mediante los valores de sus propiedades *dia_de_Mes*, *mes* y *year*[24].

Con la estructura descrita, las instancias de la clase *Fecha* presenta varios valores para la propiedad *tramo_horario* que son instancias de la clase *Tramo_Horario* (referidos a cada uno de los tramos de media hora de duración). El tramo horario vendrá definido unívocamente por los valores de sus propiedades de *hora_Inicio* y *hora_Final*. El rango de ambas propiedades se define como de la clase *Hora_Minuto*,

[24] De nuevo nos hemos topado con el problema de OilEd con la 'ñ', por lo que no se ha definido esta propiedad como año.

Figura 4.15.: Jerarquía de clases (Detalle de la superclase Empleo_Horario).

que aglutina la *hora* y los *minutos* de un horario[25]. Esta estructura de definiciones deriva de la necesidad impuesta por el esquema de un fichero RDF (recuérdese Figura 2.7).

```
<daml:ObjectProperty rdf:about= "#tramo_horario">
  <rdfs:label>tramo_horario</rdfs:label>
  <rdfs:range>
    <daml:Class rdf:about= "#Tramo_Horario"/>
  </rdfs:range>
</daml:ObjectProperty>
```

Con el fin de aclarar la estructura descrita, se muestra a continuación el siguiente código, almacenado en un fichero con formato DAML+OIL. Dicho código describe una reunión, proyectada para el 6 de marzo de 2003, entre las 10:00h y las 10:30h, entre los usuarios Roberto Marichal y Evelio con el objetivo de planificar la petición de un proyecto de investigación. La importancia asignada a la reunión es de 100 puntos, sin asignación de material.

```
<RDFNsId1:Fecha rdf:about= "A1044469811110">
  <RDFNsId1:dia_de_Mes>6</RDFNsId1:dia_de_Mes>
  <RDFNsId1:mes>3</RDFNsId1:mes>
  <RDFNsId1:year>2003</RDFNsId1:year>
```

[25] En esta implementación, los minutos solamente pueden adquirir los valores de 0 y 30. Sin embargo, esta jerarquía es adaptable a situaciones donde cada tramo tenga una duración distinta.

```
    <RDFNsId1:tramo_horario>
      <RDFNsId1:Tramo_Horario rdf:about='0D1044469811110'>
        <RDFNsId1:descriptor>"Reunión para preparar la petición un
proyecto"</RDFNsId1:descriptor>
        <RDFNsId1:puntuacion_Material>0</RDFNsId1:puntuacion_Material>
        <RDFNsId1:puntuacion>100</RDFNsId1:puntuacion>
        <RDFNsId1:hora_Inicio>
          <RDFNsId1:Hora_Minuto rdf:resource="0E1044469811110">
            <RDFNsId1:hora>10</RDFNsId1:hora>
            <RDFNsId1:minutos>0</RDFNsId1:minutos>
          </RDFNsId1:Hora_Minuto>
        <RDFNsId1:hora_Inicio>
        <RDFNsId1:hora_Final>
          <RDFNsId1:Hora_Minuto rdf:resource="0F1044469811110">
            <RDFNsId1:hora>10</RDFNsId1:hora>
            <RDFNsId1:minutos>30</RDFNsId1:minutos>
          </RDFNsId1:Hora_Minuto>
        <RDFNsId1:hora_Final>
       <RDFNsId1:actividad rdf:resource="B1044469811110"
  rdf:type="#Solicitar_Proyecto"/>
       <RDFNsId1:persona_Afectada rdf:resource="#Roberto_Marichal"/>
       <RDFNsId1:persona_Afectada rdf:resource="#Evelio"/>
      </RDFNsId1:Tramo_Horario>
    </RDFNsId1:tramo_horario>
  </RDFNsId1:Fecha>
```

4.6.5. Otras definiciones

Como se deduce a partir del código, los tramos horarios, al ser empleados para describir la agenda de usuario, son el dominio de las siguientes propiedades

descriptor una breve descripción, definida por el usuario, del tramo horario), *actividad* (la actividad a la que se pretende dedicar el tramo horario

puntuacion la puntuación asignada al tramo[26]

material_Asociado recurso(s) asociado(s) al tramo horario

puntuacion_Material puntuación asociada al material asignado al tramo horario[27]

persona_Afectada personas que están involucradas en el tramo horario.

La inclusión de otras propiedades, particulares de cada usuario, denominadas *factorEgoísmo* y *saldoInicial*, quedarán entendidas convenientemente en los apartados 4.9.1 y 4.11.1.

[26] Este concepto será explicado en el apartado 4.9.1.

[27] Este concepto se explica adecuadamente en el apartado 4.11.1.

Las últimas clases definidas en la ontología se refieren a los servicios proporcionados por los agentes, lo cual también convierte al MAS en novedoso respecto a las otras aplicaciones citadas. La superclase *Servicio* define las subclases *Administrar_Lugar_Reunion* y *Administrar_Material*, que prestarían, respectivamente, los agentes encargados de gestionar los lugares de reunión y de los recursos comunes. En este punto, se ha definido la subclase *Servicio_Autorizado_Usuario* para definir, como indica su nombre, a los servicios autorizados por el usuario, por ejemplo, cambiar la agenda propia, fijar reuniones, negociar el material reservado previamente...

```
<daml:Class rdf:about= "#Cambiar_Agenda_Propia">
  <rdfs:label>Cambiar_Agenda_Propia</rdfs:label>
  <rdfs:subClassOf>
    <daml:Class rdf:about= "#Servicio_Autorizado_Usuario"/>
  </rdfs:subClassOf>
</daml:Class>
```

Reiteramos una vez más que el objetivo de nuestro trabajo no implica una ontología completa y detallada. En este sentido, las definiciones descritas son suficientes para la implementación del sistema. No obstante las posibilidades de ampliación y de utilización de la ontología son innumerables. Así por ejemplo, puede, en un futuro, definirse la propiedad *AsignaturaImpartida*, con lo que se podrían planificar reuniones entre profesores que comparten una misma asignatura.

4.7. Inicialización del Sistema

El sistema MASplan se inicializa normalmente en el servidor del grupo CyC (Pentium III a 1 GHz con 1 Gb de memoria), por ofrecer una mayor potencia computacional que el resto de equipos, así como por ser este sistema el que suministra al grupo algunos servicios como la gestión de correo electrónico, invocado por el MA. En este servidor, en primer lugar, van a residir los agentes básicos de la plataforma FIPA, esto es, el AMS y el DF. A continuación también van a residir el OA (la ontología es accedida a través de un protocolo HTTP), el MA, el RMA y el RA (el fichero de las reglas del usuario reside en el sistema de archivos del servidor). Estos cuatro agentes son comunes a todos los usuarios, por lo que lo más lógico, desde el punto de vista operativo, es que se inicialicen en el servidor.

Por último, también van a correr en el servidor los agentes negociadores de todos los usuarios (el Agente Cargador es inicializado en el servidor). De este modo se facilitan las comunicaciones entre los agentes negociadores o entre un NA y el RMA y, por tanto, se reduce el tiempo que precisa una negociación, al evitarse el acceso a red. Asimismo se logra que todos los NAs se encuentren activos durante todo el tiempo, con lo que un usuario siempre encontrará el agente adecuado con quien negociar, independientemente de que el usuario se encuentre o no conectado a MASplan a través de su UA. Los agentes a inicializar por el CA se indican a FIPA-OS mediante un fichero de perfiles XML, donde se codifica el nombre del agente, la clase que lo implementa, el nombre de la plataforma que lo contiene y si se inicializa por

defecto al cargar el sistema.

```
  <?xml version="1.0" encoding="UTF-8"?>
  <?enhydra-unmarshall package="fipaos.agent.profile"?>
  <loaderProfile>
    <agentDescription agentName="ams"
      className="fipaos.platform.AgentManagementSystem"
      owner="fipaos" start="true" />
    <agentDescription agentName="df"
      className="fipaos.platform.DirectoryFacilitator"
      owner="fipaos" start="true" />
    <agentDescription owner="fipaos"
      className="cyc.agendaGrupo.AgenteOntologia"
      agentName="AgenteOnt1" start="true" />
    <agentDescription owner="fipaos"
      className="cyc.agendaGrupo.AgenteMail"
agentName="AgenteMail1"
      start="true" />
    <agentDescription owner="fipaos"
      className="cyc.agendaGrupo.AgenteRecursosMateriales"
      agentName="AgenteMaterial1" start="true" />
    <agentDescription owner="fipaos"
      className="cyc.agendaGrupo.AgenteCargador"
      agentName="AgenteCargador1" start="true" />
  </loaderProfile>
```

A nivel de implementación de código Java, consideramos ilustrativo de la sencillez de empleo de las librerías FIPA-OS mostrar las siguientes líneas, referidas al registro de un NA en el DF. Este código permite manejar estructuras en el lenguaje SL como las señaladas en el apéndice E.

```
  String AGENT_TYPE="Agente_Negociador";
  try
  {/* crear la descripción para el registro con el DF */
   DFAgentDescription descripcion = new DFAgentDescription ();
   ServiceDescription[] serv = new ServiceDescription[1];
   serv[0] = new ServiceDescription();
   serv[0].setServiceName("Serv_1");
  /* indicar el tipo de servicio (tipo del agente) */
   serv[0].setServiceType(AGENT_TYPE);
  /* indicar la ontología */
   String[] ontol = new String[1];
   ontol[0] = "AgendaGrupo.daml";
   serv[0].setOntologies(ontol);
  /* especificar el usuario específico y la acción de negociar la
agenda */
   PropertyTemplate[] u = new PropertyTemplate[2];
```

```
 u[0] = new PropertyTemplate("Usuario",usuario);
 u[1] = new PropertyTemplate("Servicio","NegociarAgenda");
 serv[0].setProperties(u);
 descripcion.setAgentServices(serv);
/* especificar los lenguajes soportados */
 String[] languages = new String[2];
 languages[0]= "FIPA-SL";
 languages[1] = "DAML";
 descripcion.setLanguages(languages);
/* registro con el DF */
 registerWithDF(null, descripcion);
} catch (Exception e) {}
```

También aprovecharemos para destacar la detección de un error en la implementación de FIPA-OS. Concretamente, cuando se manda un mensaje en cuyo contenido se incluye un objeto de la clase *PropertyTemplate*, el sistema lanza una excepción, ya que no puede transportar ese mensaje. La explicación estriba en que en la distribución del programa, este objeto no implementa la interfaz *Serializable*, por lo que no puede seriarse y mandarse empleando un protocolo como CORBA. De hecho, recompilando el código con esta corrección, el mensaje puede ser enviado. Se constata de este modo el acierto en la elección de una herramienta de código libre, que permite detectar y corregir este tipo de errores.

La Figura 4.16 muestra un diagrama general UML que indica el flujo de mensajes entre los agentes. Dicho diagrama se encuentra simplificado ya que, por sencillez, no se han incluido los registros con el DF ni con el AMS, ni los flujos de control en las líneas de tiempo de vida de los agentes implicados. Asimismo, no se han detallado las respuestas del sistema ante situaciones adversas (por ejemplo, la reacción del CA si no recibe respuesta del OA antes de cumplirse el plazo de tiempo prefijado).

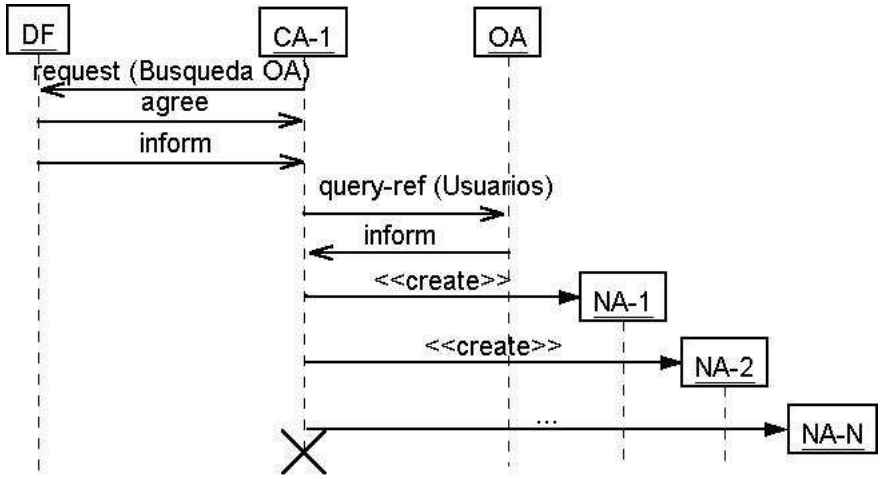

Figura 4.16.: Inicialización del sistema MASplan.

Figura 4.17.: FIPA-OS Agent Loader.

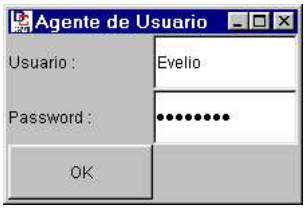

Figura 4.18.: Verificación de contraseña de usuario.

4.7.1. Inicialización de un Agente de Usuario

En esta sección se explica cómo un usuario inicializa a su agente de usuario. El usuario carga[28] a un Agente de Comprobación, ya sea mediante el *Agent Loader*, mostrado en la Figura 4.17, de la distribución FIPA-OS o mediante línea de comando. Este agente CoA, como el resto de los agentes de la plataforma, intenta registrarse con AMS y el DF.

Una vez logrado este objetivo muestra al usuario una ventana (Figura 4.18), solicitándole el nombre de usuario y su contraseña en el sistema. Esta contraseña, por seguridad, se ha tomado como diferente de la del sistema operativo del servidor CyC, y se ha generado mediante la ayuda de la librería de encriptación proporcionada por *Sun Microsystems*. Esta API también es empleada por el CoA para comprobar la contraseña introducida por el usuario. De esta forma se garantiza la autenticación del usuario en las gestiones iniciadas por su UA. Conviene aclarar que las contraseñas, en esta primera versión del MAS, no son almacenadas en ningún fichero, sino que son verificadas internamente por el código de encriptación incluido en el CoA. Como consecuencia de esta política de diseño, en esta implementación, un usuario CyC no puede cambiar su contraseña de acceso al sistema.

Tras verificar la contraseña, el CoA solicita al DF que le informe sobre la existencia o no de un UA[29] ya iniciado (más correcto es decir registrado) en el sistema, lo cual corresponde al símil de tener otra *terminal abierta*. En caso de existir dicho UA, el sistema le manda la orden de morir. El motivo es evitar problemas de registro del UA inicializado tanto en el AMS como el DF, debido a que se registrarían con el mismo nombre. Hasta que el CoA compruebe que no existe ningún UA referido al mismo usuario, no creará al agente de usuario. Este esquema proporciona seguridad

[28] Se puede cargar el agente desde un ordenador distinto del servidor del grupo.
[29] de modo formal, preguntar al DF si existe un agente capaz de mostrar la agenda al usuario indicado.

al MAS (el usuario controla sus *terminales*) al mismo tiempo que reduce el coste computacional.

El agente de usuario, tras registrarse en el sistema, realiza una serie de tareas que le permiten servir de interfaz de usuario. En concreto, necesita saber la lista de usuarios en el MAS, las tareas posibles a desarrollar por el usuario y las clases de recursos materiales. Para ello, solicita información al DF sobre la existencia de un OA en el sistema. Cuando recibe esta información, el UA establece con el OA tres conversaciones sucesivas *FIPA Query* realizando consultas sobre los usuarios, tareas y recursos materiales. Conviene aclarar que las búsquedas se realizan mediante el lenguaje FIPA-SL. Por ejemplo, el siguiente código en el contenido del mensaje solicita la lista de usuarios.

```
((iota ?x (instance-of ?x Usuario)))
```

Con esta información, el UA ya se encuentra preparado para establecer una comunicación con el NA encargado de negociar la agenda del usuario correspondiente.

La primera comunicación entre ambos agentes consiste en la petición por parte del UA de la agenda actual del usuario (el NA es el único agente del sistema con capacidad para modificar el fichero-agenda del usuario) mediante un nueva conversación *FIPA Query*. El NA devuelve en el campo *content* del mensaje *inform* el fichero-agenda del usuario.

Al recibir el fichero, el UA puede mostrar ya toda la información, mediante la interfaz de usuario (la estructura de esta interfaz se muestra en la Figura 4.20). Para ello procesa los valores recibidos mediante las librerías de la herramienta *Jena*.

La Figura 4.19 muestra un diagrama general de tiempo UML del proceso descrito. Como en el caso de la inicialización del sistema, este diagrama se encuentra simplificado, no mostrando las posibles respuestas ante un comportamiento no deseado (por ejemplo, actos de comunicación *failure* o *non-understood* en la respuesta del DF).

4.8. Estructura de la Interfaz de Usuario

La estructura de la interfaz de usuario se fundamenta en cinco clases principales, que determinan las diferentes zonas gráficas de dicha interfaz, tal como se muestra en la Figura 4.20.

Calendar Esta subclase de la clase *JPanel* sirve al usuario para mostrar de forma gráfica un calendario de un mes. Para ello emplea las librerías estándar Java e implementa el algoritmo de Butcher[30] para determinar las fechas correspondientes a Semana Santa. Este calendario, pulsando sobre la casilla

[30] a partir de un artículo anónimo publicado en *Nature* en 1876. (Nature, 1876 April 20, vol. 13, p. 487)
Algoritmo de Butcher:

```
a=year %19              k=c %4
b=year/100              l=(32+2*e+2*i-h-k) %7
c=year %100             m=(a+11*h+22*l)/451
d=b/4                   Mes =(h+l-7*m+114)/31
e=b %4                  [3=Marzo, 4=Abril]
f=(b+8)/25              p=(h+l-7*m+114) %31
g=(b-f+1)/3             Día=p+1
```

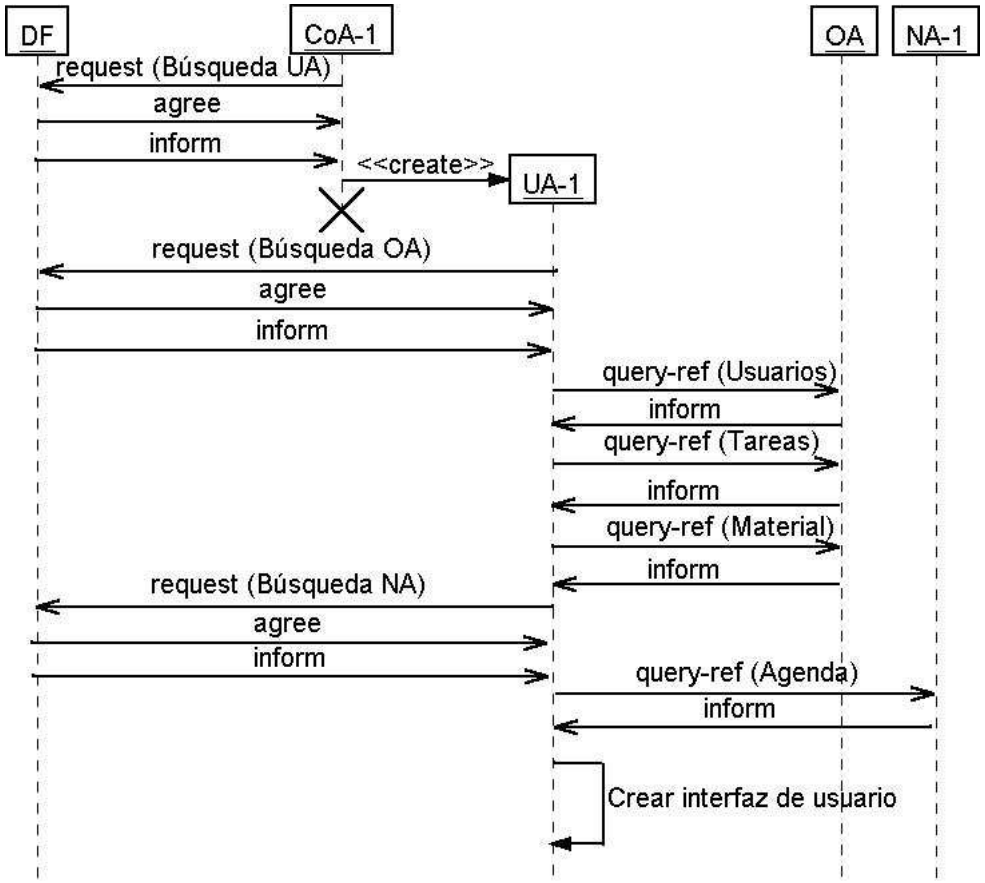

Figura 4.19.: Inicialización de un Agente de Usuario UA-1.

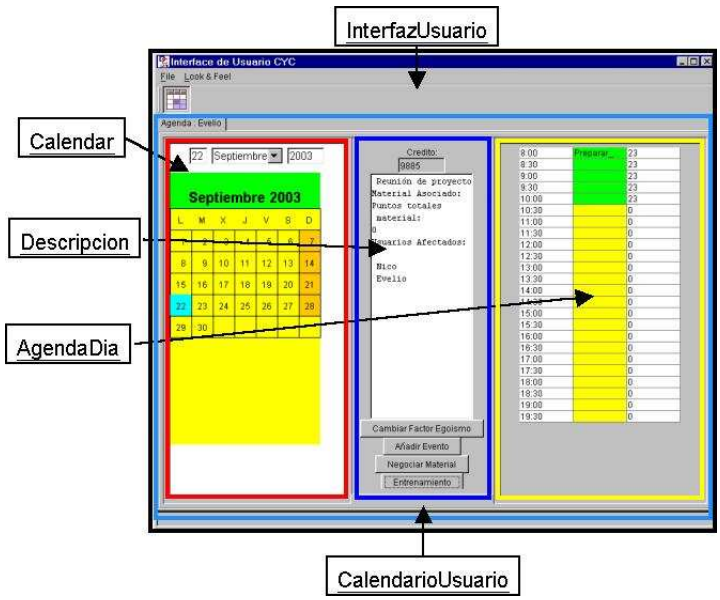

Figura 4.20.: Estructura de la interfaz de usuario.

respectiva, permite seleccionar los diferentes días del mes. Al seleccionar un día, se muestra en la zona de *AgendaDia* los datos relativos a la agenda del usuario en ese día. Para fijar una reunión, o solicitar un recurso, el usuario debe primero seleccionar el día deseado en este calendario.

AgendaDia Esta clase también hereda de la clase *JPanel*. Es la encargada de albergar una tabla que muestra, a modo de horario, las actividades previstas para el usuario durante el día seleccionado en el calendario de la clase *Calendar*. Dicha tabla está formada por tres columnas que muestran la hora de comienzo de cada tramo horario, el tipo de actividad prevista (el nombre de la subclase de *Empleo_De_Horario*) y un número (posteriormente denominado *puntuación*) relacionado con la importancia del evento.

Descripcion Subclase de *JPanel* que muestra la descripción detallada del evento seleccionado en la tabla incluida en el área de *AgendaDia*. Asimismo detalla la puntuación restante que le queda al usuario (ver sección 4.9.1) y permite acceder mediante botones a diferentes servicios: fijar los parámetros deseados de una reunión o efectuar la reserva de un recurso común, cambiar el factor de egoísmo o entrenamiento de reglas.

CalendarioUsuario Clase que, como las anteriores, también hereda de la clase *JPanel* y que sirve de contenedor y nexo de comunicación entre las tres clases anteriores.

InterfazUsuario Subclase de la clase *JFrame* que representa la ventana donde se

```
h=(19*a+b-d-g+15) %30
i=c/4
```

incluyen los *JPanel* descritos con anterioridad, y unos menús que permiten terminar el UA así como cambiar la apariencia de la interfaz.

4.9. Planificación de una reunión en MASplan

Como se ha mencionado en la introducción al problema, uno de los objetivos para el diseño del sistema multiagente es la planificación de reuniones internas dentro del Grupo CyC en un escenario donde resulta prácticamente imposible el encontrar un horario donde todos los asistentes deseados se encuentren libres. Por tanto la solución a encontrar es la que produzca un menor daño a los usuarios. Para ello, se ha diseñado un algoritmo de negociación, explicado en el subapartado 4.9.1. Esta descripción se complementa con el subapartado 4.9.2 donde se muestra el flujo de mensajes y protocolos FIPA establecidos entre los agentes del sistema para la implementación del algoritmo.

4.9.1. Determinación del horario más adecuado

En el sistema MASplan, el proceso de negociación entre los agentes para la planificación de una reunión tiene lugar mediante un mecanismo diseñado a medida consistente en una variación del método de votación del recuento de Borda.

Por propósitos de operabilidad, se divide la agenda diaria de los usuarios en tramos de media hora. Cuando un usuario desea negociar una reunión que comience un tramo horario de índice i y L tramos de duración, su NA calcula el mejor horario posible. Este cálculo se realiza mediante la minimización de una función de coste que representa el grado de incomodidad producida entre los asistentes. Dicha función se construye de la siguiente forma.

En primer lugar, el agente determina un factor de distancia D_n para todos los tramos horarios del día seleccionado. Este factor representa la distancia existente de un tramo horario de índice n a los tramos de horario en los cuales se desea convocar la reunión. Denominando $f = i + L - 1$ al último tramo deseado involucrado en la reunión, este factor D_n es igual a

$$D_n = \begin{cases} 0 & si\, i \leq n \leq f \\ n - f & si\, n > f \\ i - n & si\, n < i \end{cases} \tag{4.19}$$

Una vez que el agente calcula este factor para todos los tramos horarios, el NA del usuario iniciador pregunta a los NAs de los invitados pretendidos por la puntuación dada por sus usuarios a los tramos de tiempo deseados por el iniciador para la reunión. De este modo, el NA iniciador puede asignar un valor de coste (daño producido) J_n a cada tramo del día deseado para la reunión según la ecuación

$$J_n = k\, D_n + \frac{1}{P} \sum_{i=1}^{N} P_{i,n} \tag{4.20}$$

donde

- N es el número de asistentes deseados, excluyendo al iniciador

- P es la puntuación por tramo de tiempo que el agente iniciador da a la reunión
- $P_{i,n}$ es la puntuación dado por el agente i al tramo de tiempo de índice n
- k es un factor que pesa la importancia del factor de alejamiento. Un valor aconsejado (aunque puede adquirir cualquier valor) para esta constante podría ser el siguiente:

$$k = \frac{N}{tramos\,de\,tiempo\;diarios} \tag{4.21}$$

que pesa el número de agentes involucrados (como es el caso del segundo sumando, relativo a las puntuaciones) y de algún modo normaliza el alejamiento entre los tramos de tiempo.

Obsérvese que el segundo sumando de la ecuación 4.20 introduce un mecanismo de votación entre los agentes similar al de recuento de Borda, pero en esta ocasión la opción favorita no es aquélla con una mayor puntuación, sino la que representa un menor daño a los usuarios involucrados (menor puntuación). Con este sumando, por tanto, se incluye en la negociación los criterios de Pareto, monotonía y perdedor de Condorcet. No obstante, estos criterios pueden verse corrompidos por la inclusión del factor de alejamiento de la ecuación 4.19 y un factor que involucra la duración de la reunión, que será incluido en la ecuación 4.22. Precisamente esta distorsión provoca que no sea productivo la implementación de algoritmos basados en los mecanismos de votación de Black, Nanson y Copeland en comparación al aumento de complejidad computacional que supondría su inclusión en la función de coste (recordar que la simplicidad computacional es uno de los parámetros deseables en una negociación entre agentes). Un mecanismo de votación es adecuado para este caso, puesto que la negociación involucra a un número a priori desconocido de agentes.

Aparte de la inclusión en el mecanismo de negociación de los criterios mencionados, este segundo término de la ecuación 4.20 implica el cumplimiento del objetivo de la distribución (en cuanto al proceso de negociación se refiere) dentro del sistema multiagente, ya que la determinación del resultado no va a depender de un único agente. Del mismo modo, también aparece la premisa de simetría, puesto que todos los agentes tienen el mismo peso dentro de la decisión final.

Como se ha indicado, para concluir con el cálculo de la función de coste, el agente calcula la función de coste teniendo en cuenta la diferencia entre la duración deseada y la posible. Así se puede definir la función de coste $J_{n,d}$ a cada reunión posible que empezase en el tramo de tiempo n y tuviese duración d de la siguiente manera.

$$J_{n,d} = \sum_{i=n}^{n+d-1} J_i + \frac{L-d}{L} \tag{4.22}$$

En este punto, obsérvese que en el contexto descrito no tiene demasiado sentido calcular estos términos para una duración de la reunión más larga que la deseada. En todo caso, se puede considerar, sin necesidad de cambiar la definición de la función de coste, que un valor de duración d mayor que la propuesta L puede tener una contribución negativa a la función de coste. Es decir, que, por ejemplo, se pueda sacrificar los factores de alejamiento y puntuación en aras de una mayor duración de la

reunión. Sin embargo en MASplan, de forma general, se calcula el valor de $J_{n,d}$ para los valores de d tales que $d = 1, 2, \ldots, n$.

Este método global puede recordar ligeramente al sistema de votación ponderada del modelo de Sandip Sen con tres parámetros: la importancia de cada tramo horario, el alejamiento al horario deseado y la duración de la reunión. No obstante debemos señalar que, mientras que en el modelo de Sandip Sen, la importancia de un tramo horario concreto es la intersección de las preferencias dadas por el usuario a cada día de la semana y a cada fase del día (mañana, tarde), el modelo MASplan permite asignar un valor a cada tramo en sí de forma independiente. Aparte de que en MASplan se busca minimizar una función mientras que en el de *Sandip Sen* se compara con un valor de referencia. Otra diferencia es su filosofía de que la reunión propuesta ha de celebrarse en todo caso, puesto que se supone que repercutirá favorablemente a la marcha del grupo CyC , aunque trata de minimizar el *daño* causado a las agendas de los usuarios.

Tras los cálculos anteriores, el NA iniciador selecciona los parámetros que minimizan esta función de coste, comunicando el resultado a los otros agentes negociadores. Si no ocurre nada anómalo, todos los agentes cambiarán sus respectivos ficheros-agenda, puesto que se supone la estabilidad del proceso de negociación.

Obsérvese que la función de coste global definida de este modo penaliza

- El alejamiento de la reunión al horario propuesto por el usuario iniciador. Por tanto, el sistema tenderá a encontrar el horario más cercano a los deseos del usuario.
- Fijar la reunión en un horario donde sus tramos hayan sido valorados con una alta puntuación por el resto de usuarios respecto a la puntuación del usuario iniciador[31].
- La desviación de la duración de la solución propuesta respecto a la deseada. Es decir, el sistema tenderá a fijar la reunión con la duración más larga posible.

La tabla 4.15 muestra en detalle este proceso con un ejemplo. El agente iniciador desea organizar una reunión con tres miembros de CyC: L. Acosta, G.N. Marichal y C. González. Esta reunión debería empezar a las 10:00h de la mañana (correspondiente al tramo de tiempo de índice 5) y duración L=5 (es decir 2h30min). Se supone unos valores de k=0.23 y P=100. Las filas relativas a cada uno de los usuarios reflejan los valores asignados a cada de uno de los tramos horarios (numerados de 1 a 11). Obsérvese que ninguno de estos tramos se encuentra libre para todos los usuarios afectados.

Tras los cálculos descritos, el NA iniciador encuentra dos configuraciones que minimizan la función de coste (concretamente $J_{5,2}$ y $J_{6,1}$). Cuando esto sucede, el NA selecciona la de duración más larga. En el caso analizado, propone una reunión a celebrar a las 10:30h (tramo de horario de índice 5) y de duración igual a 2 tramos (es decir, una hora).

La estructura del NA permite que sean implementadas fácilmente otras técnicas de negociación adicionales. Entre éstas cabe citar el abandono de la negociación si el valor

[31] o lo que es lo mismo, se entrometa en actividades consideradas importantes por el resto de usuarios, respecto a la importancia asignada por el iniciador.

Cuadro 4.15.: Ejemplo de proceso de negociación

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
G.N. Marichal	100	100	20	20	10	0	0	30	10	20	25
C. González	10	50	30	10	0	10	10	35	10	50	0
L. Acosta	0	0	10	10	10	0	50	0	20	35	10
$k * D_n$	0.69	0.46	0.23	0	0	0	0	0	0.23	0.46	0.69
J_n	1.79	1.69	0.83	0.40	0.20	0.10	0.60	0.65	0.63	1.51	1.04
$J_{n,5}$	5.18	3.49	2.13	1.95	2.18	3.49	4.43	X	X	X	X
$J_{n,4}$	5.18	3.59	1.73	1.50	1.75	2.18	3.59	4.03	X	X	X
$J_{n,3}$	4.98	3.59	1.83	1.10	1.30	1.75	2.28	3.19	3.85	X	X
$J_{n,2}$	4.35	3.39	1.83	1.20	**0.90**	1.30	1.85	1.88	2.74	3.15	X
$J_{n,1}$	2.59	2.76	1.63	1.20	1.00	**0.90**	1.40	1.45	1.43	2.31	1.84

mínimo de la función de coste es mayor que un valor de referencia o la negociación de la reunión para otro día.

En este sentido, debemos señalar que inicialmente nada impide que cada usuario pueda escoger su propia función de coste o los valores de los parámetros incluidos en ella, con lo que se podrían diseñar otras estrategias. Un usuario puede codificar su propia estrategia en una clase aparte e indicar, en el correspondiente fichero de perfil la clase de su estrategia seleccionada. Esto supone un enriquecimiento del sistema, alejándolo de la mera resolución distribuida de problemas. No obstante, y de modo práctico en la versión inicial del MAS, se ha consensuado dentro del grupo CyC la fórmula descrita de determinación del mejor horario, común a todos los NAs. Esta fórmula proporciona aparentemente un alto nivel de bienestar social (las reuniones que favorecen al grupo tienen lugar, intentando minimizar el daño que puede causar su convocatoria) dificultando la aparición de estrategias dominantes basadas en comportamientos anti-sociales.

En esta línea, un usuario podría convertirse en dictador del sistema mediante la estrategia dominante de asignación de un valor artificialmente alto a todos los tramos horarios de su agenda, lo cual no sería aceptable desde el punto de vista del bienestar social. Este efecto se mitiga asignando a cada usuario un valor mensual de puntos, de modo que la suma global de los puntos no pueda superar en ningún momento esa asignación mensual. En el momento de escritura de este trabajo este valor mensual es de 10000 puntos. No obstante, esta cantidad puede ampliarse en situaciones especiales. Así por ejemplo, al haber incluido en la ontología la ocupación de horario *Tesis*, un usuario en situación de lectura de tesis (un control de veracidad de esta situación puede ser llevada a cabo empleando la clasificación de usuarios dentro de la ontología) puede ver aumentada su asignación mensual. A fecha de escritura de este trabajo, esta contabilidad del crédito restante es llevada a cabo por el mismo NA. Se puede plantear la creación de un nuevo agente AgenteCajero, encargado de verificar que la operación a realizar cuenta con crédito suficiente.

Los usuarios CyC pueden configurar su valor deseado para el parámetro k de la ecuación 4.20 en el ficheros XML de perfil de su NA, lo que vendría a ser equivalente a la definición de una estrategia propia. Un valor de k alto pesaría más el factor de alejamiento, por lo que el sistema tendería a fijar una reunión lo más cercana posible al horario deseado por el usuario, sin coste adicional para el usuario.

En un caso extremo, esto supondría ignorar las votaciones proporcionadas por el resto de los NAs, lo que conduce de nuevo a la aparición de una estrategia dominante indeseable. Por ello se ha limitado el valor posible del parámetro k (entre 0 y 2).

Esta posibilidad ha supuesto un cambio de diseño en el proceso de asignación de la puntuación resultante para los tramos horarios. En un principio, una vez concretada la reunión, se asignaba a los tramos horarios el valor indicado por el iniciador a través de la interfaz gráfica del UA. Sin embargo, tras analizar el caso indicado, se llegó a esta propuesta

$$P_{asignada} = \text{máx}\left(P, \frac{1}{N}\sum_{i=1}^{N} P_i\right) \tag{4.23}$$

es decir, en caso de que la media de la incomodidad causada sea mayor que la puntuación propuesta, el iniciador se verá "penalizado" con dicha media. De este

modo, se asegura el mínimo impuesto por el usuario iniciador, lo cual es conveniente para que el grado de resistencia a la hora de fijar otras reuniones en los mismos tramos horarios sea al menos el deseado por el usuario. Pero, del mismo modo, favorece la definición de un valor bajo del parámetro k, puesto que la misma definición de la función coste en la ecuación 4.20 se encargaría de buscar situaciones donde se minimice la media de las puntuaciones (*daños provocados*) correspondientes al resto de usuarios[32]. El código necesario se implementa en el NA del iniciador, donde además se verifica que la puntuación resultante se encuentra dentro del crédito sobrante del iniciador. En caso contrario, se pasa a la siguiente opción calculada, hasta encontrar una que se sitúe dentro del crédito restante. Una versión más refinada del MAS podría consistir en que sea el propio NA quien, respetando el esquema general de negociación, suba o baje a conveniencia el valor del parámetro k siguiendo una estrategia definida por el usuario. En este caso, quedaría justificada la integración de la herramienta de razonamiento *Jess*.

Otro caso en este tipo de negociación, a analizar en futuras versiones del sistema multiagente, es la posible formación de coaliciones de agentes para la organización de reuniones, ante situaciones de crédito insuficiente. Este caso puede permitir la inclusión en el sistema de técnicas más sofisticadas de negociación.

Durante todo el diseño del sistema multiagente hemos estando suponiendo un comportamiento honesto por parte del usuario, pero dicha suposición no debe implicar que el sistema permita comportamientos anti-sociales. Así, la inclusión del mecanismo basado en un RA no supone la aparición de una estrategia dominante consistente en que un usuario se muestre disconforme con todos los cambios que se produzcan, ya que provocaría un elevado gasto de puntuación propia. El factor de aumento de puntuación se realiza en términos de porcentaje, con un valor seleccionado por el usuario. Tampoco sería una estrategia acertada, por la misma razón, el fijar un porcentaje elevado para dificultar la organización de las reuniones.

4.9.2. Flujo de mensajes entre los agentes de MASplan

De un modo básico, el proceso de negociación dentro del sistema multiagente se fundamenta principalmente en la acción de los UAs y NAs de los usuarios involucrados. No obstante, en el estudio del flujo de mensajes entre los agentes del sistema, incluiremos los agentes MA y RA. De no encontrarse en activo estos dos últimos agentes (o el UA de alguno de los usuarios afectados) el proceso de negociación prescinde del flujo de mensajes involucrado, continuando el proceso de forma normal. También reflejamos la intervención del DF en este proceso.

Una vez que el usuario fija, mediante la ventana mostrada en la Figura 4.21, los parámetros deseados en la reunión a organizar (personas afectadas, importancia, asunto, fecha y hora) el UA realiza una comprobación de los parámetros suministrados por el usuario (se ha seleccionado una fecha posterior a la actual, se dispone de crédito suficiente[33], el formato de los campos del formulario son los adecuados...). A continuación establece una conversación según el protocolo de comunicación *FIPA*

[32] Obsérvese que la limitación en el crédito de puntuación disuade a un usuario malintencionado de provocar mermas de puntuación en los otros usuarios.

[33] en futuras implementaciones, esta verificación corresponderá a un AgenteCajero

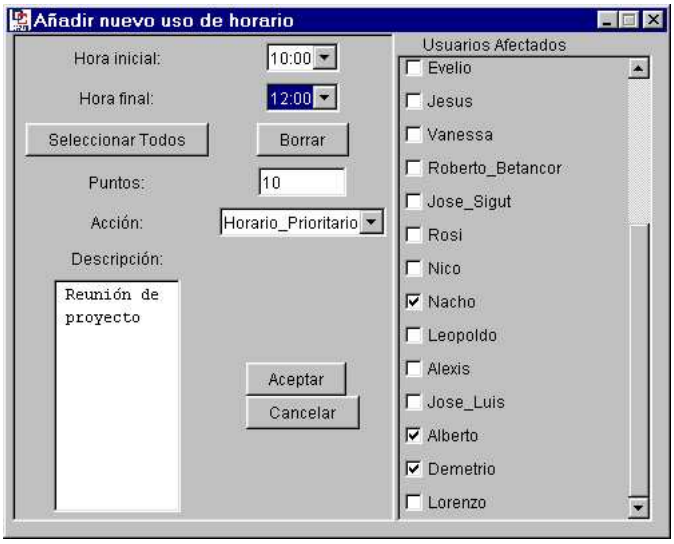

Figura 4.21.: Petición de reunión empleando la interfaz de usuario.

Request con su NA correspondiente, solicitándole que negocie los parámetros de la reunión.

El primer punto de la planificación de la reunión consiste en comprobar si existe algún otro usuario involucrado en la actividad a realizar. En caso negativo, es decir, cuando el usuario quiere marcar parte de su agenda con una actividad a realizar en solitario, el NA cambia su fichero-agenda e informa del resultado a su UA. En el caso más general de que haya otros usuarios involucrados, el NA debe localizar (mediante las gestiones ante el DF) los NAs encargados de negociar en su nombre. Cuando recibe la localización de estos agentes, el NA iniciador establece diferentes conversaciones *FIPA Query* simultáneos con el resto de NAs involucrados solicitándoles las puntuaciones asociadas a cada tramo horario del día seleccionado.

En este caso, el NA de cada invitado analiza los datos de la reunión y consulta al agente de reglas RA sobre si del comportamiento registrado anteriormente de su usuario se deduce si éste se encuentra o no propenso a facilitar la reunión. En caso negativo, el NA aumenta en un cierto factor proporcional sus puntuaciones asociadas (siempre, claro está, que su crédito se lo permita). Al ser el aumento proporcional, se dificulta que la reunión tenga lugar en los tramos horarios considerados más importantes para el usuario.

Con las respuestas recibidas, el NA determina la mejor hora posible de la reunión siguiendo el protocolo descrito en la sección 4.9.1. El NA entonces realiza la propuesta-resultado a través de actos *FIPA Propose* al resto de NAs, que si todo va de forma adecuada, es respondido de forma positiva.

Una vez recibidas todas las aceptaciones, el NA iniciador avisa al UA de que la reunión ya se ha fijado, mientras que los NAs de los invitados localizan a sus respectivos UAs comunicándoles que se ha producido un cambio en sus agendas (ver Figura 4.22). Los usuarios también son advertidos de estos cambios a través de un correo electrónico mandado por el MA (un ejemplo de correo electrónico de este tipo

Figura 4.22.: Aviso de nuevos cambios en la agenda de usuario.

se muestra en la Figura 4.23).

Todos este flujo de mensajes aparece sintetizado en el diagrama UML simplificado de la Figura 4.24 [OPB01].

4.10. Estructura de Implementación de los Árboles de Identificación

Como se ha indicado anteriormente, se ha intentado dotar de una mayor inteligencia al MAS mediante la inclusión de un agente que extraiga reglas de comportamiento deseado por el usuario. Debido al alto número de variables involucradas en el sistema (los 27 usuarios, la puntuación, cada tipo de empleo de horario), así como la presencia de variables no numéricas, se ha implementado un algoritmo de árboles de identificación como el mostrado en el apartado 4.1.9, incluyendo la simplificación mediante Fisher. Esta implementación se ha realizado en Java para su integración con el resto del MAS[34].

A nivel de MAS, se intenta dotar de una mayor inteligencia al sistema mediante la inclusión del agente de reglas RA. Este agente lee la cuenta de correo ficticia *MASplan*, buscando correos electrónicos enviados por cada usuario (ya sea en modo de entrenamiento, cuya ventana se muestra en la Figura 4.25, o a partir de las respuestas producidas ante cambios reales en la agenda). Tras la lectura de estos correos, más un fichero que almacena datos anteriores (empleando la interfaz *Serializable* de Java), se forman árboles de identificación que intentan reflejar las posibles reglas sobre el comportamiento del usuario acerca de la idoneidad o no de un nuevo cambio de agenda. En caso de que el RA detecte que el nuevo cambio de agenda probablemente no sea del gusto del usuario, se lo indica al NA. La consecuencia de esta respuesta es que el NA dificulta la organización de la reunión, aumentando la puntuación asignada a los tramos de horario involucrados.

El esquema de esta implementación puede verse en la Figura 4.26. El RA crea un objeto *Algoritmo*, al que se le pasan como argumentos las variables y los datos a analizar. La información de cada variable se almacena en objetos de la clase *Variable*. Cada objeto contiene el nombre de la variable así como el conjunto de valores posibles de dicha variable.

Las variables escogidas se pueden agrupar en tres grupos:

- Variables de los usuarios. Cada variable recibe el nombre del usuario correspondiente (*Evelio*, *Alberto...*) pudiendo adquirir los valores de *Sí* y *No*, según se encuentre o no involucrado en la reunión a planificar.

[34] recordar que el sistema *Electric Elves* también emplea árboles de identificación.

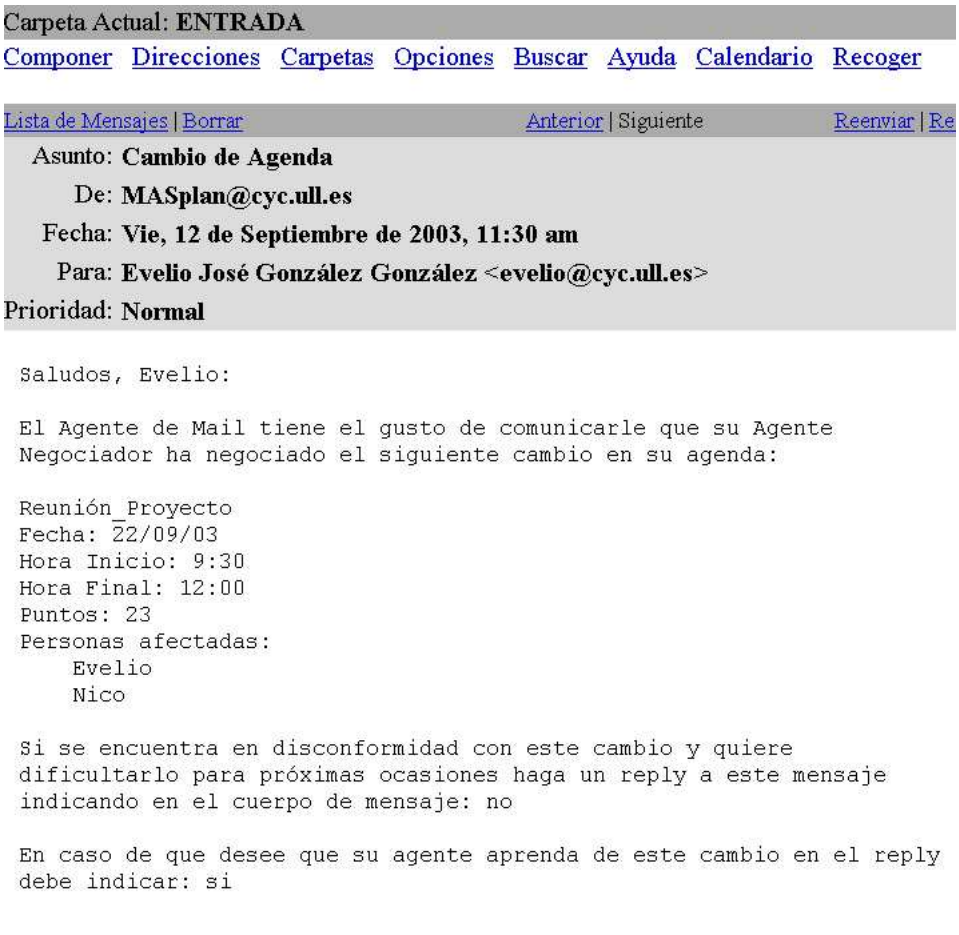

Figura 4.23.: Ejemplo de correo electrónico generado por MASplan.

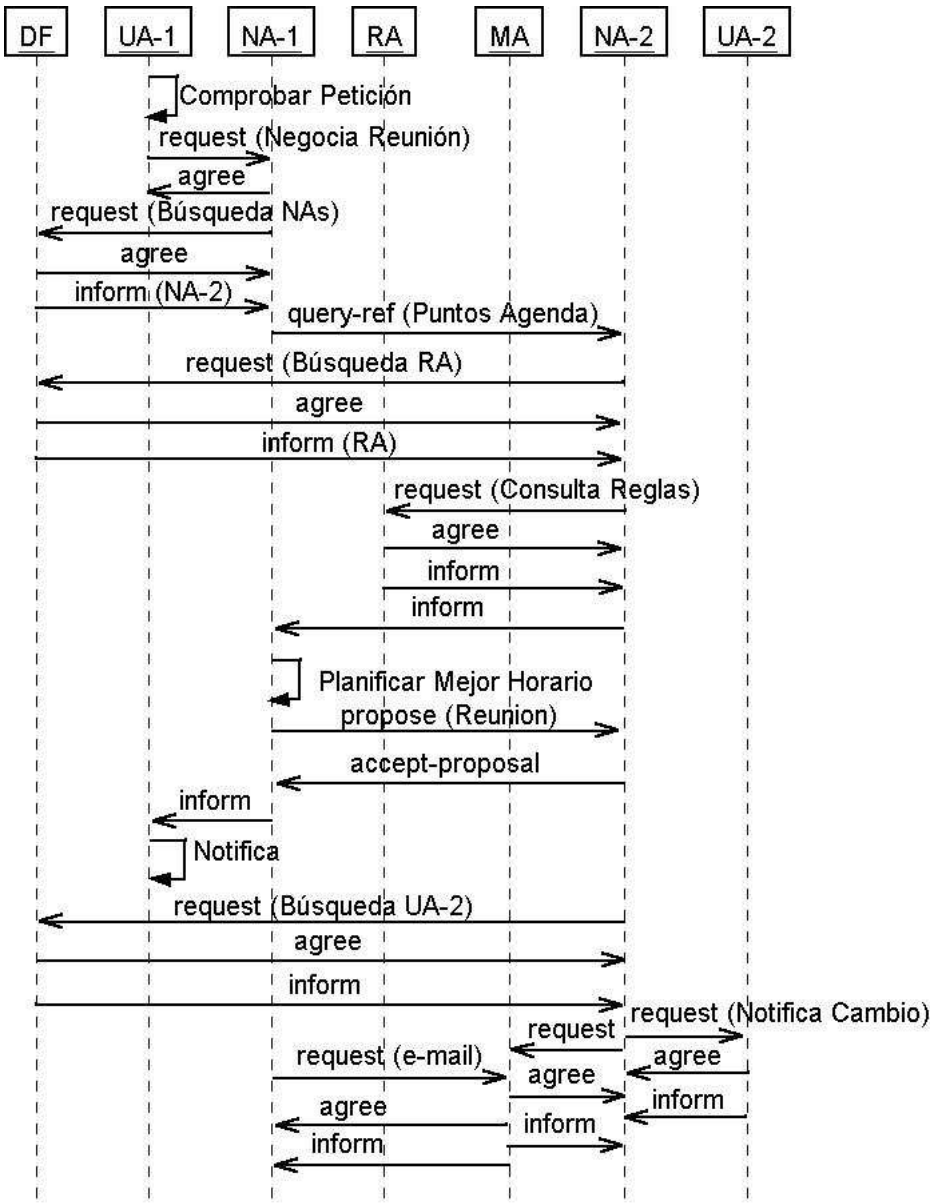

Figura 4.24.: Flujo de mensajes MASplan en el proceso de planificación de reuniones.

Figura 4.25.: Módulo de entrenamiento de reglas.

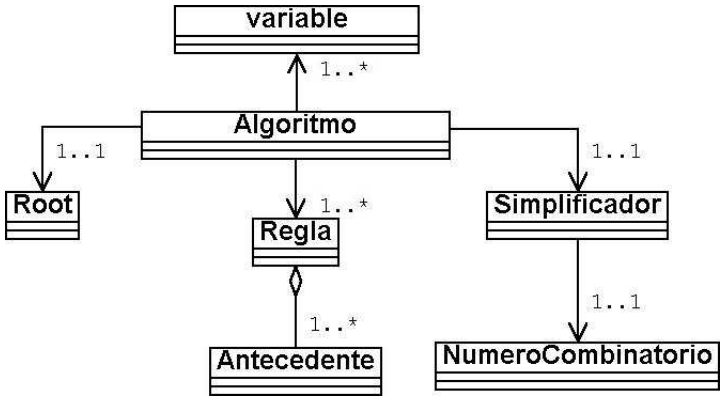

Figura 4.26.: Estructura de implementación de árboles de identificación.

- Variable de actividad. Esta variable *Actividad* tiene como valores posibles los nombres de las diferentes subclases de *Empleo_De_Horario.*

- Variable de puntuación. Como los valores de la puntuación asignada puede adquirir cualquier valor entero, no resulta operativo asignar un valor de variable a cada número entero. Además valores de puntuación similares deben ser tratados por el algoritmo como correspondientes la misma tendencia. Con este fin, los valores de la variable corresponden a grupos de puntuación. Concretamente, se han escogido tres grupos de puntuación: de 0 a 10 puntos (poco importante), de 10 a 100 puntos (importancia media) y superior a 100 puntos (gran importancia). Esta división no resuelve completamente el problema, puesto que un valor de 99 es tratado de forma diferente a 101 a pesar de su cercanía, mientras que es agrupado con valores lejanos como 11. Por tanto, debe ser considerada como un tratamiento preliminar del problema.

El objeto *Algoritmo* empieza el proceso de obtención de las reglas. Para ello analiza el desorden en la distribución de los datos producido por cada variable, seleccionando la variable de menor desorden (que será el nodo de comienzo del árbol de identificación, llamado *Root*). Se establece un proceso recursivo con el resto de variables en cada rama del árbol (una para cada valor posible de la variable *Root*).

Una vez implementado el árbol, se procede a la formación de las reglas. El árbol se recorre formando los antecedentes (clase *Antecedente*) que forman las reglas (clase *Regla*). Finalmente se simplifican estas reglas mediante la fórmula de la prueba exacta de Fisher (un objeto de la clase *Fisher*, que admite como argumentos los valores de V_{++}, V_{+-}, V_{-+} y V_{--}). Se ha implementado un método optimizado para el cálculo de los números combinatorios involucrados en la ecuación 4.18 (se emplea un objeto de la clase *NumeroCombinatorio*).

Se ha simplificado, dentro de lo posible, la generación de los árboles de identificación, omitiendo las variables referidas a duración y horario concreto de la reunión. Si se hubiese introducido estas variables en el árbol de identificación, el número de datos necesarios para la extracción de reglas de alta probabilidad hubiese crecido hasta el punto de hacer inútil la inclusión del RA. En esta línea de favorecer la aparición de reglas, se ha fijado un valor de probabilidad en el tramo ligeramente superior (0.08) de lo recomendado en la bibliografía (0.05).

4.11. Negociación de Recursos

Tal como se indicó en la presentación de este sistema multiagente, otra de las fuentes de conflicto existente en el escenario del grupo de investigación CyC consiste en la gestión de los recursos comunes, concretamente de los portátiles y el cañón de proyección. Este objetivo diferencia al MAS implementado respecto a los antecedentes analizados, más centrados en la planificación de reuniones.

En este caso, el proceso escogido para la negociación es un mecanismo de regateo[35], analizado en el apartado siguiente. Tal como en el caso de la planificación de reuniones,

[35] En realidad, y tal como se ha implementado se trata de un mecanismo iterativo de redes de contrato. No obstante, y debido a la sencillez del mismo (ya que existe un único argumento negociado, el precio) hemos preferido definirlo como perteneciente a un mecanismo de regateo.

analizaremos el flujo de mensajes entre los agentes (apartado 4.11.2).

4.11.1. Mecanismo de Regateo

Cuando dos o más miembros de CyC solicitan el mismo recurso, no es sencillo determinar cuál de ellos goza de una mayor prioridad. MASplan presenta un esquema de negociación para este caso, basado en una combinación de las tácticas de negociación dependientes del tiempo e imitativa del comportamiento como las mostradas en la sección 4.1.4.1.

Ilustraremos este mecanismo a través del siguiente ejemplo. Un miembro B del grupo CyC indica mediante su UA su deseo de emplear el cañón de proyección para una conferencia dentro de 5 días. Sin embargo, cuando su NA correspondiente trata de reservarlo, se da cuenta, tras consultar al agente de recursos materiales RMA (encargado de extraer la información de los ficheros[36] correspondientes), de que el recurso se encuentra actualmente reservado por otro miembro S de CyC (un caso que se analizará más adelante es cuando existen varias instancias del mismo tipo de recurso).

En ese momento, el NA correspondiente a B comienza un proceso de negociación bilateral orientada a servicios tratando de *comprar* el uso del cañón de proyección al *vendedor* S, empleando un mecanismo de regateo, basándose para ello en el crédito mensual asociado.

Tal como se indica en 4.1.4.1, el primer paso consiste en determinar la región [min^i_j, max^i_j] (j=S,B) aceptable para cada uno de los agentes involucrados. Este proceso no es siempre sencillo. En el caso del comprador B, los límites de la región deben basarse en la oferta (el precio) realizada por su usuario a través de la interfaz gráfica de su UA y en el capital restante del crédito mensual. Por el contrario, en el caso del vendedor S el principal factor consistiría en el precio pagado previamente cuando reservó el cañón de proyección. Además, en el contexto de las negociaciones dentro de MASplan, el vendedor S normalmente no se encuentra muy interesado en vender el recurso, ya que S también necesita el proyector para el mismo día y hora. Por otra parte se muestra receptivo a que el empleo del cañón de proyección por parte del usuario B pueda tener una mayor prioridad dentro del grupo CyC. Por tanto, es de esperar que los límites de la región de S se muevan artificialmente por encima del valor pagado previamente. Por ejemplo, si el precio pagado por S fue de 100, unos límites lógicos en la negociación podrían situarse en [125,175], lo que garantiza una ganancia de al menos un 25 %, en caso de perder el uso del recurso común.

Una vez que los agentes han determinado sus respectivas regiones, pueden comenzar la negociación, informando al otro de sus ofertas sucesivas en el proceso de regateo. La implementación de dicho proceso en MASplan involucra una suma ponderada de términos relativos a tácticas dependientes del tiempo e imitativas del comportamiento contrario.

Así el término dependiente del tiempo para el comprador B adquiere la siguiente forma donde se ha elegido el término exponencial frente al polinomial. La ecuación es una variación derivada de las ecuaciones 4.8 y 4.10, con un valor equivalente de $\beta = 1$, lo que supone un término medio entre las tácticas de concesión y las de Boulware.

[36] implementados en DAML+OIL

$$OT_B(t) = min_B + (max_B - min_B)\left(1 - e^{-\frac{t}{t^B_{max}}}\right) \tag{4.24}$$

En el caso del vendedor S se asume que, debido a las características particulares del escenario, la convergencia a una solución aceptable debiera ser más lenta, ya que el agente no se encuentra demasiado interesado en desprenderse del recurso común. Por este motivo se ha incluido en la oferta dependiente del tiempo OT_S un factor que se ha denominado como *factor de egoísmo* E_S (cuyo cometido es similar al del exponente β en la ecuación 4.10) que representa la resistencia del vendedor en llegar a un acuerdo. De un modo experimental, se ha comprobado que un valor de E_S=3 reproduce un comportamiento adecuado para el bienestar social.

$$OT_S(t) = min_S + (max_S - min_S)e^{-\frac{t}{E_S t^B_{max}}} \tag{4.25}$$

El factor de egoísmo es modificable por el usuario mediante la opción correspondiente que aparece en la interfaz gráfica del UA. En aras de evitar la aparición de comportamientos anti-sociales, dicho factor se puede mover entre unos límites prefijados (a fecha de escritura de este trabajo estos límites son [1,4]). Una medida adoptable para favorecer que los usuarios no aumenten hasta el máximo este factor, tal como indicaría su comportamiento racional, consiste en el aumento del crédito mensual de puntos para aquellos usuarios que se muestren más *generosos*.

En el ejemplo que estamos tratando, en principio $t^B_{max} = 5$ días. De manera obvia, no es una buena idea que los agentes se mantuviesen negociando durante todo ese tiempo. Por este motivo, MASplan considera en realidad $t^B_{max} = 5$ etapas en vez de días. Esta suposición implica que *t=t+1* cada vez que el comprador B (el iniciador) realiza una nueva oferta. Obsérvese que de este modo se penalizan las peticiones de material para un tramo horario próximo en el tiempo al empleo del recurso. Así se implementa en el MAS la noción intuitiva de que un cambio de planes es más dañino para el vendedor cuanto más cerca se produce del tramo horario afectado.

Por otro lado, MASplan incluye en la oferta global de B un término correspondiente a una oferta OB_B derivada de una táctica dependiente del comportamiento global del vendedor $O_S(t)$. Para esto, se ha implementado una variación del Tit-for-Tat absoluto aleatorio (ecuación 4.14).

$$OB_B(t+1) = OB_B(t) + \text{mín}\left(R(1)\left(O_s(t-1) - O_s(t)\right), K_B\, O_B(t)\right) \tag{4.26}$$

donde el producto $K_B O_B(t)$ representa la máxima variación que se puede producir en la oferta dependiente del comportamiento OB_B. Por ejemplo, un valor K_B=0.2 representa una variación máxima del 20 %.

De modo recíproco,

$$OB_S(t+1) = OB_S(t) + \text{mín}\left(R(1)\left(O_B(t-1) - O_B(t)\right), K_S\, O_S(t)\right) \tag{4.27}$$

Finalmente la oferta global es una suma ponderada de ambos términos.

$$O_j(t) = w_{T,j}(t) OT_j(t) + (1 - w_{T,j}(t))\, OB_j(t) \quad j = B, S \tag{4.28}$$

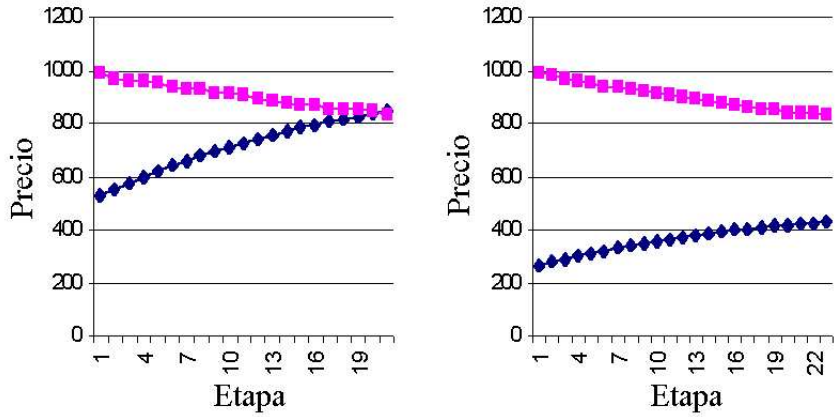

Figura 4.27.: Ejemplos de negociación de recursos.

donde

$$w_{T,j}(t) = w_{min,j} + (1 - w_{min,j})\,R(1) \tag{4.29}$$

En esta ecuación $w_{min,j}$ es el mínimo valor del peso relativo a la oferta dependiente del tiempo. Un valor recomendado para este peso es 0.8. Como consecuencia, el factor dependiente del tiempo es pesado al menos 4 veces más que el dependiente del comportamiento.

Dos ejemplos de negociación se muestran en la Figura 4.27. En el primero de ellos, ambos NAs alcanzan un acuerdo satisfactorio para ambas partes, es decir, la oferta del comprador B es mayor que la última oferta de S. En cambio, la segunda negociación acaba en fracaso, ya que el comprador no alcanza la oferta del vendedor antes de que se acabe el tiempo de negociación. Inicialmente esta negociación acabaría en un tiempo máximo t^B_{max} (común para comprador y vendedor), pero se ha incluido, teniendo en cuenta las consideraciones anteriores, un factor corrector que aumente el número de etapas límite para la negociación. De este modo, la negociación se produce mientras $t < 1{,}25 * t^B_{max}$.

Independientemente del éxito o fracaso de la negociación, ambos NAs remiten el resultado a los respectivos UAs (en caso de encontrarse activos). No obstante, y en caso de que la negociación suponga un cambio de propietario, se debe producir antes un protocolo de cesión/adquisición del recurso a través del RMA. Como cabría pensar, la puntuación asignada al material no es la propuesta inicial del usuario solicitante, sino la resultante de la negociación. El vendedor S recibe la bonificación correspondiente en su crédito mensual.

Finalmente, y del mismo modo, se remite una petición al MA (igualmente si éste se encuentra activo) para que envíe un correo electrónico (similar al de la Figura 4.23) tanto al comprador B como al vendedor S, avisando de los detalles (puntuación, horario...).

De nuevo se está suponiendo un comportamiento honesto por parte de los usuarios. Con respecto a este comportamiento, la modularidad del sistema multiagente permite la inclusión de un nuevo agente (denominado por ejemplo AgenteJuez) encargado de

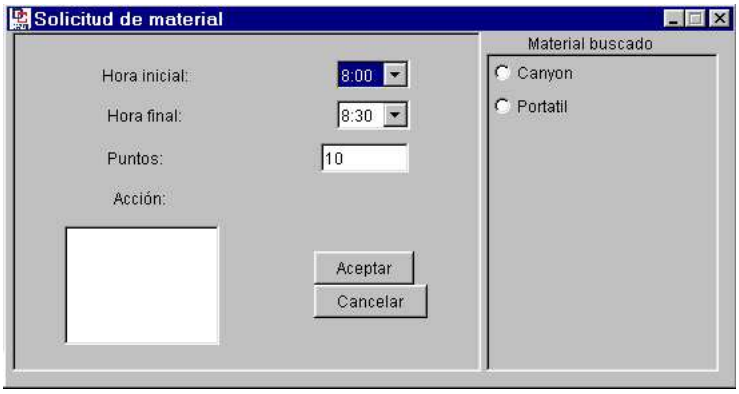

Figura 4.28.: Petición de reserva de material a través de la interfaz de usuario.

imponer sanciones a aquellos usuarios que manifiesten un comportamiento anti-social (por ejemplo, no ceder el recurso o apropiarse indebidamente de él).

4.11.2. Flujo de mensajes en la Negociación de Recursos

Como en el caso de la planificación de agendas, la negociación se fundamenta en los NAs y UAs, pero del mismo modo se puede incluir de forma general el MA en este proceso. La principal diferencia con el anterior es la inclusión del agente de recursos materiales RMA. También reflejamos en este flujo el papel del DF.

En primer lugar, el UA realiza, de modo análogo al señalado en la planificación de reuniones, una comprobación de los parámetros de la petición de material suministrados por el usuario (la ventana en que lo pide se muestra en la Figura 4.28). En caso de ser una petición válida, establece una conversación según el protocolo *FIPA Request* con el NA[37], solicitando que negocie este cambio en la agenda y reduciéndose el número de mensajes necesarios.

Una vez que el NA acepta hacerse cargo de la negociación, debe, en primer lugar, comprobar si el recurso se encuentra reservado por otro usuario. Para ello, el agente a consultar es el RMA. Tras el correspondiente proceso de búsqueda en el DF, el NA consulta al RMA sobre quién o quiénes tienen reservado el tipo de recurso común solicitado, así como qué instancias del recurso aparecen en el sistema.

Para este cometido, en primer lugar, el RMA consulta al OA sobre las instancias del tipo de recurso. Con esta información, puede consultar los ficheros-agenda relativos a estas instancias. Debido a que el usuario puede, en el caso más general, querer reservar el recurso para varios tramos horarios, las instancias pueden estar reservadas por varios usuarios CyC. En resumen, de forma general, la respuesta del RMA consiste en un vector (clase *seriable* en Java) con la lista de las instancias y los usuarios involucrados en cada tramo horario.

Con esta información, el NA comprueba si existe una instancia del tipo de recurso que se encuentra disponible en todos los tramos horarios solicitados. En caso

[37] En realidad realiza con anterioridad una búsqueda en el DF del NA. Se ha omitido este protocolo en la Figura 4.29 debido a razones de espacio.

afirmativo, el NA solicita la reserva directamente al RMA mediante un protocolo de comunicación *FIPA Request*.

En caso de que ninguna instancia del recurso se encuentre totalmente disponible establece varios protocolos *FIPA Iterated Contract Net* simultáneos, intentando obtener propuestas asumibles por el comprador y en todo caso, con el mecanismo y las limitaciones en el número de iteraciones permisibles descritos en el apartado 4.11.1. El conjunto global de propuestas es evaluado por el NA, seleccionando la que más se adecúa a los deseos del usuario iniciador (menor precio, posibilidad de reserva del material en todos los tramos horarios...). En este caso, se envían respectivos mensajes *accept-proposal* a los NAs seleccionados. Como es preceptivo en este protocolo, al resto de NAs que continúan esperando respuesta a sus ofertas se les envía un mensaje de rechazo *reject-proposal*.

La materialización de la venta se produce cuando el vendedor se lo indica al RMA (tras una nueva búsqueda en el DF), para que cambie el usuario poseedor del recurso en los tramos horarios deseados. Tras recibir confirmación, los cambios son notificados a los usuarios mediante sus respectivos UAs y por correo electrónico mandados por el MA.

El flujo descrito se muestra, de forma simplificada en la Figura 4.29.

Se reproduce a continuación el código Java encargado de comenzar la primera conversación según el protocolo FIPA Request por parte del UA para solicitar al NA que negocie acerca de reservar un portátil.

```
  try{
  /* comenzar una nueva conversación FIPA Request */
  ACL aclneg = getNewConversation("fipa-request");
  String contenido = "((action (agent-identifier :name
AgenteNegociador"
  + usuario + ")(NegociarMaterial Portatil)))";
  /* fijar los campos del mensaje */
  aclneg.setPerformative("request");
  aclneg.setSenderAID(_owner.getAID());
  aclneg.setReceiverAID(agenteNegociador);
  aclneg.setContentObject(contenido);
  /* enviar el mensaje */
  forward(aclneg);
  } catch (Exception e) { e.printStackTrace();}
```

4.11.3. Flujo de mensajes en el Cambio de Factor de Egoísmo

Los agentes involucrados en el proceso de cambio del factor de egoísmo son el DF, UA, NA y OA. El usuario CyC solicita mediante la interfaz de usuario el cambio de su factor. Tras comprobar la legitimidad del cambio (un número entre 0 y 2), el agente de usuario UA busca establecer un protocolo *FIPA Proxy* con el NA relativo a su mismo usuario, para que propague un mensaje con la petición *request* al OA, que es el único capaz de acceder al fichero de ontologías. Para ello, antes debe localizar a dicho agente,

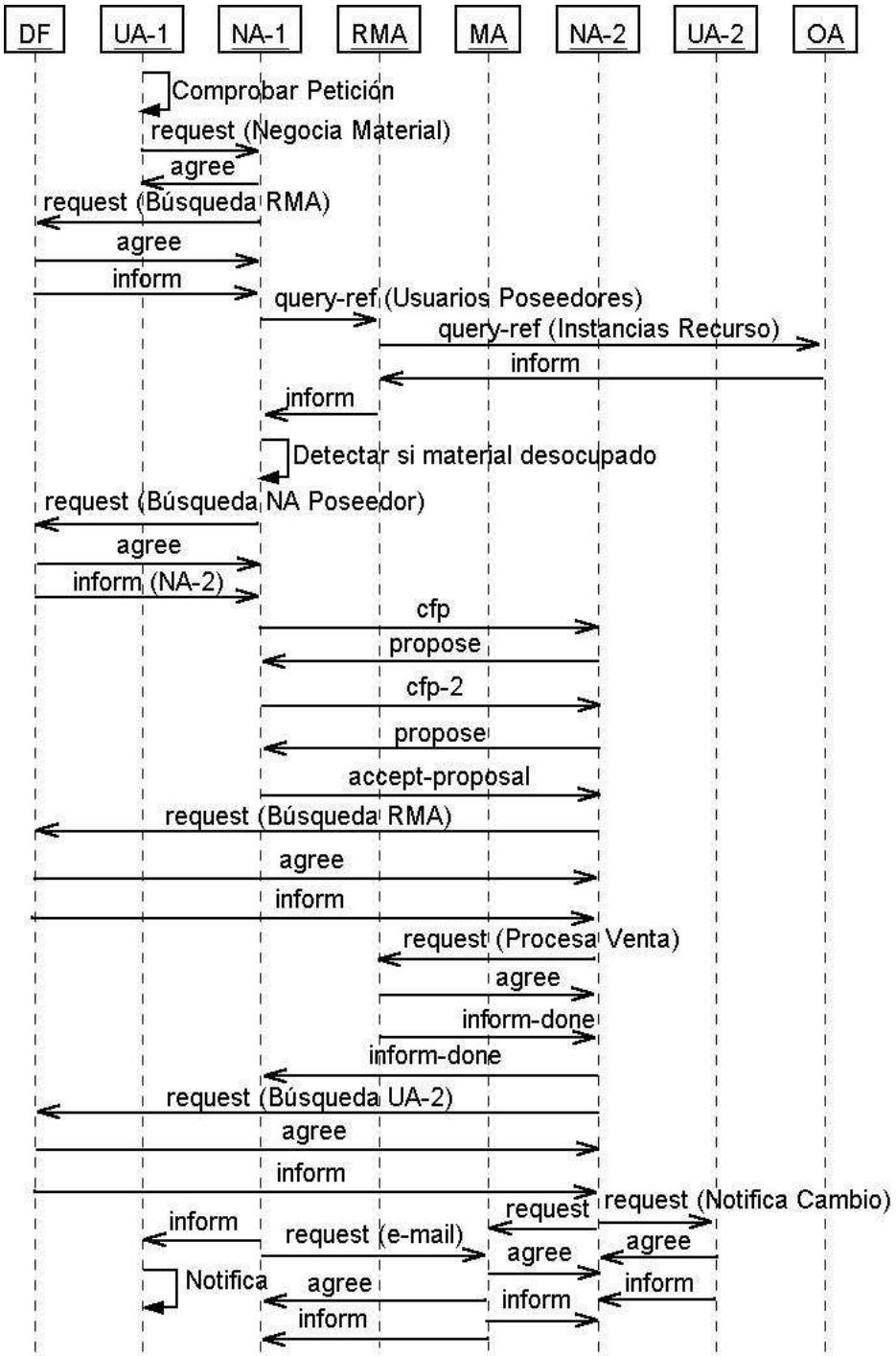

Figura 4.29.: Esquema general de la negociación de recursos.

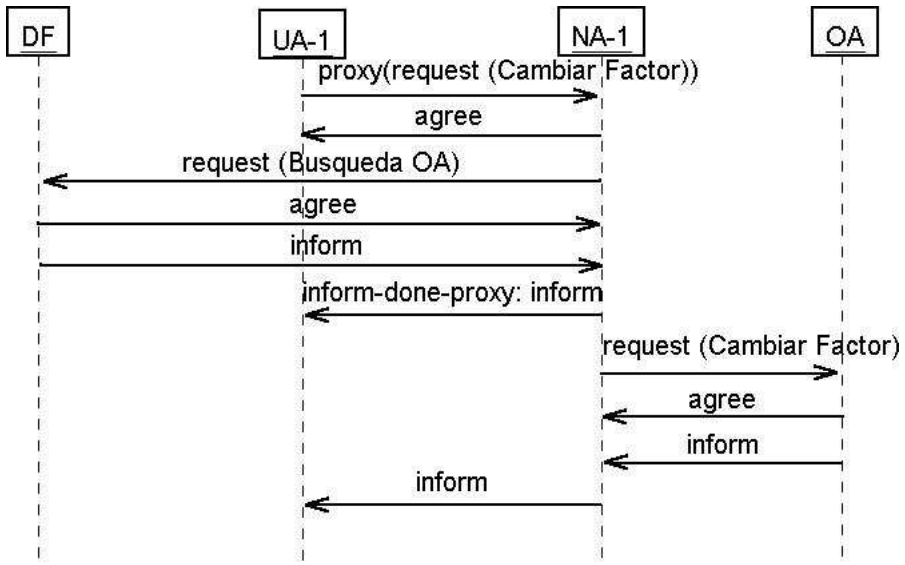

Figura 4.30.: Cambio del Factor de Egoísmo.

invocando para ello al DF[38]. El agente NA también realiza a un proceso de búsqueda en el DF, esta vez para localizar al OA. Por tanto, el NA solicita, mediante un protocolo de conversación *FIPA Request* (el mensaje embebido del *proxy* inicial), que el OA modifique el factor de egoísmo del usuario. Al recibir la confirmación mediante el *inform*, comunica el resultado de la gestión al UA iniciador. El protocolo global en una versión simplificada se muestra en la Figura 4.30. Este protocolo ha sido diseñado de esta manera para dotar de una mayor variedad al MAS en cuanto a comunicaciones (podría haberse implementado mediante un acto de comunicación *request*).

El siguiente código reproduce un mensaje enviado por un NA al OA para que modifique el correspondiente factor de egoísmo.

```
(request
  :sender (agent-identifier :name AgenteNegociadorEvelio)
  :receiver (set (agent-identifier :name OA))
  :content
    ((action (agent-identifier :name OA)
      (CambiarFactorEgoismo 2.5)))
  :protocol fipa-request
  :language FIPA-SL
  :reply-with orden11)
```

Como se ha podido comprobar en los protocolos vistos en esta sección, cada vez que un agente necesita comunicarse con otro agente, busca su localización en el DF mediante la descripción de servicios del agente buscado. De este modo, el sistema global está preparado para la movilidad de los agentes o coexistencia de varios agentes

[38] formalmente, intenta localizar a un agente registrado capaz de prestar el servicio de cambio de la agenda propia del usuario implicado.

que ofrezcan los mismos servicios, en caso de implementarse en futuras versiones del MAS.

4.12. Evaluación del Sistema

El sistema MASplan ha sido evaluado dentro del grupo CyC. La evaluación de la herramienta se centra en dos aspectos principales: la robustez y la generación de situaciones en concordancia con el comportamiento esperado.

4.12.1. Evaluación de la robustez

En cuanto al primer punto, se ha sometido al sistema a una serie de pruebas durante un periodo de evaluación de varias semanas. Mediante dichas pruebas, se ha comprobando la robustez del sistema multiagente ante varias pruebas de esfuerzo, consistentes en cargar el sistema con un flujo alto de mensajes. En este sentido, cabe destacar el comportamiento del sistema con los 27 UAs conectados, cada uno de ellos iniciando un proceso de negociación independiente (obsérvese que es una situación extremadamente improbable en el comportamiento real en el sistema). Esto supuso una gran carga para el servidor (del orden de 10 *threads* por NA), y que mientras tanto ha de seguir ofreciendo el resto de servicios al grupo. No obstante, en un periodo entorno a 10 minutos fue capaz de resolver satisfactoriamente las negociaciones.

Otra posible limitación del sistema puede estribar en el tamaño de los ficheros involucrados. En este sentido, el manejo del fichero de la ontología no supone problema, puesto que su tamaño es inferior a los 70 Kb. El resto de los ficheros crecen a medida de que los usuarios realizan cambios en su agenda, pudiendo en algún instante ser conflictivo su uso. Al respecto se ha adoptado una política de eliminación periódico de aquellos *triples* referidos a tramos horarios anteriores a tres meses a la fecha del sistema. Un usuario puede desear en un momento dado analizar sus datos referidos a una reunión pasada en el tiempo, por lo que se ha dejado dicho margen de tres meses. Como medida complementaria, los ficheros-agenda son almacenados siguiendo una escritura RDF más compacta (RDF/XML-ABBREV). Podemos concluir, por tanto, que las necesidades computacionales del sistema son asumibles tanto por el servidor como por la red de comunicaciones CyC.

4.12.2. Evaluación del comportamiento del sistema

Respecto al segundo punto citado, el referido a la generación de situaciones en concordancia con el comportamiento esperado, éste fue comprobado forzando en el sistema situaciones lo suficientemente diversas y estudiando posibles modificaciones en el código. De este modo se produjo una realimentación por parte de los usuarios de casos no contemplados en el diseño inicial y que se han incluido en el sistema, o se contemplarán en futuras versiones de MASplan.

La reacción de los usuarios CyC respecto al sistema multiagente planificador de agendas puede calificarse de acogedora y receptiva, aunque debemos señalar que la mayoría de los usuarios han seguido optando por el método *tradicional* a la hora de organizar sus reuniones y reservando los recursos comunes. Conviene recordar que no está dentro de la filosofía del sistema tomar alguna medida contra aquellos usuarios que

no emplean el sistema (por ejemplo negarles el recurso común si no ha sido reservado mediante *MASplan*). Esto puede, en un futuro dar lugar a conflictos, por ejemplo, ante coincidencias en la reserva de un recurso común. Afortunadamente este tipo de problemas no se detectó durante el periodo de funcionamiento de la herramienta.

A la fecha de escritura de este trabajo, la herramienta se encuentra inactiva debido a las pruebas del sistema multiagente destinado al control de plantas descrito en el capítulo 5 y por la aparición de inestabilidades (ajenas al empleo de MASplan) en el servidor. Una vez solucionadas estas inestabilidades, se pretende ofertar un proyecto a alumnos de la Escuela Técnica Superior de Ingeniería en Informática de la Universidad de La Laguna para la implementación de técnicas de negociación más complicadas, así como su extensión a situaciones no contempladas hasta ahora (por ejemplo, la organización de los laboratorios, o reserva de materiales para el mismo día de la semana durante una temporada). Del mismo modo, este proyecto persigue la mejora de la interfaz del UA para dar un servicio de agenda competitivo con las aplicaciones similares (por ejemplo, generación de alarmas). Con esta mejora se facilitaría, a nuestro juicio, la aceptación de la herramienta por parte de los miembros del grupo.

Otra de las líneas abiertas de este MAS consiste en definir una política de privacidad de información de los datos de usuario según los propietarios de los NAs involucrados, esto es, el usuario podría restringir en distintos niveles la información comunicada al resto de agentes.

4.13. Inclusión de un nuevo agente en MASplan

Dedicaremos una sección a ilustrar el mecanismo de inclusión de un nuevo agente de tipo conocido en el sistema multiagente propuesto. Para ello tomaremos, sin pérdida de generalidad, el ejemplo de los pasos a seguir por un usuario que desee crear un agente negociador que implemente una política de negociación distinta a la descrita en las secciones anteriores. Como hemos indicado, en el caso de la planificación de reuniones, el usuario puede especificar la forma de la función de coste y el valor de sus parámetros en una clase separada. A partir de las respuestas de los otros agentes de negociación involucrados sobre la puntuación asignada a los tramos horarios. En este apartado, vamos un paso más allá al definir el proceso de un agente que puede basar su negociación en otros parámetros, como por ejemplo a partir de un fichero de afinidades con el resto de usuarios.

Dividiremos la sección en dos partes: la relativa al código Java del nuevo agente y la relativa a las modificaciones del resto de elementos (ontología, fichero de perfiles...).

4.13.1. Código Java de implementación del nuevo agente

Como en cualquier clase Java, el primer punto en el código (aparte de la definición del paquete donde se incluirá la clase) consiste en la importación de los paquetes necesarios. Respecto a este aspecto, hacemos hincapié en dos clases de paquetes: los paquetes incluidos en la distribución FIPA-OS y referentes a la implementación de las conversaciones, tareas, transporte de mensajes...

```
import fipaos.ont.fipa.*;
```

```
import fipaos.agent.*;
import fipaos.platform.*;
import fipaos.util.DIAGNOSTICS;
```

y por otra las relativas a los paquetes de la herramienta Jena, que permitirá al agente manejar los modelos de las agendas en DAML+OIL.

```
import com.hp.hpl.mesa.rdf.jena.mem.ModelMem;
import com.hp.hpl.mesa.rdf.jena.model.*;
import com.hp.hpl.mesa.rdf.jena.vocabulary.*;
import com.hp.hpl.jena.daml.*;
```

Para que el agente pueda emplear los métodos de envío de mensajes y registro con el AMS y DF, hereda de la clase *FIPAOSAgent* incluida en la distribución de la herramienta FIPA-OS.

```
public class NuevoAgenteNegociador extends FIPAOSAgent
```

El código debe reflejar asimismo el servicio que el agente implementa, en este caso un nuevo servicio de negociación respecto a un usuario definido. Dicha definición tiene lugar (como en el resto de agentes) en el registro con el DF.

```
try
{DFAgentDescription descripcion = new DFAgentDescription ();
...
PropertyTemplate[] u = new PropertyTemplate[2];
u[0] = new PropertyTemplate("Usuario",usuario);
// fijar el nuevo servicio asumido, que tiene el mismo
// nombre que el del resto de agentes NA
u[1] = new PropertyTemplate("AccionSoportada",
"Fijar_Reuniones");
...
//registrar el agente con el DF
registerWithDF( null, descripcion);
} catch (Exception e) { e.printStackTrace();}
```

Independientemente de la política de negociación a implementar, la inclusión del nuevo agente debe ser llevada a cabo sin necesidad de modificación del código del resto de agentes como indica el paradigma de la programación orientada a agentes. Dicho de otro modo, el agente debe ser compatible con lo diseñado hasta el momento. Esto se traduce en que el nuevo agente debe ser capaz de responder de modo adecuado y robusto a los flujos de mensajes vistos en las secciones 4.9.2, 4.11.2 y 4.11.3. Así, como ejemplo, respecto a la negociación de reuniones, el agente debe procesar las peticiones de los UAs (según el protocolo que se desea llevar a cabo) y de otro NA iniciador que solicite la lista de puntos de los tramos de horario. Para este último caso, consistiría en la mera duplicación del código de la tarea incluida en la clase *AgenteNegociador*. Por el contrario, para la implementación de la nueva política de

negociación la estructura del código sería la siguiente:

```
  // tarea que procesa por defecto los mensajes recibidos por el
agente
  private class IdleTask extends Task
  {

  // procesa los mensajes con acto de comunicación request
    public void handleRequest(Conversation id)
    {
     ...
  // si en el contenido aparece la orden de "Fijar_Reuniones"
     if (contenido==“Fijar_Reuniones”){
  // aquí debe ir el código que fijará
  //la nueva política de negociación
     _tm.newTask(new nuevaPoliticaTask(id), id);
     }
     ...
    }

  }

  public class nuevaPoliticaTask extends Task
  {...
  }
```

A la vista de estas modificaciones, la inclusión de nuevos agentes codificados por otros usuarios puede ser llevada a cabo de un modo sencillo. Es importante recalcar que el código del agente puede estar escrito en otro lenguaje, por ejemplo C++ o Perl, siempre que se registre correctamente con el DF y responda adecuadamente a las conversaciones definidas. Durante esta sección, nos hemos fijado en el lenguaje Java por ser en el que está implementado la herramienta FIPA-OS.

4.13.2. Resto de modificaciones

Aparte de las evidentes modificaciones en el código Java, pueden ser necesarias algunas modificaciones. Así, respecto a la ontología, puede definirse, por ejemplo, un nuevo servicio de negociación que sea una subclase del servicio de negociación implementado por el resto de NAs. Esta modificación puede ser llevada a cabo de modo inmediato a través de la herramienta OilEd, o mediante modificación directa del fichero de texto.

```
  <daml:Class rdf:about="#NuevoServicioFijarReuniones">
    <rdfs:subClassOf>
      <daml:Class rdf:about="#Fijar_Reuniones"/>
    </rdfs:subClassOf>
```

```
</daml:Class>
```

Con esta información, un agente que se registre en el DF anunciando este nuevo servicio, debería aparecer en la respuesta del DF ante una consulta sobre los agentes que implementan el servicio de Fijar_Reuniones. Para este caso, el DF estándar de la distribución FIPA-OS debe ser modificado para dotarlo de cierta capacidad de inferencia a partir del lenguaje DAML+OIL.

5. Sistema MASCONTROL

En este capítulo se detalla el sistema MAS propuesto y realizado en este trabajo para la identificación y control de procesos. Con este campo de actuación, considerablemente distinto al de negociación de agendas visto en el capítulo anterior, se pretende demostrar la versatilidad del empleo de sistemas multiagentes (en particular la arquitectura FIPA y los agentes de ontologías) y los lenguajes de marcas de alta riqueza semántica. Asimismo se entronca con el trabajo desarrollada por el Grupo de Computadoras y Control de la Universidad de la Laguna, en cuyo seno se ha desarrollado este proyecto de tesis.

Comenzaremos definiendo brevemente una serie de conceptos básicos del campo del control y que van a ser empleados a lo largo de todo el capítulo: sistema, proceso, planta, variables de estado... A continuación pasaremos a describir, siempre de modo breve las formas de representación de sistemas (función de transferencia y variables de estado) tanto en sistemas continuos como en discreto, y los controladores básicos empleados en este trabajo (PID y realimentación por variables de estado). Todos estos términos son muy conocidos dentro del control de procesos, pero esta sección pretende ser una ayuda para aquél lector ajeno a dicho campo.

Las siguientes secciones ya se refieren al otro problema involucrado en el planteamiento del MAS: la identificación de sistemas. En este sentido se tocará (de nuevo de forma breve) el problema de la identificación, los criterios de estimación de parámetros, los diversos modelos de función de transferencia racional (ARX, FIR, ARMAX...), los predictores de salida y la identificación en tiempo real. Este último concepto será de enorme importancia en nuestro trabajo, puesto que el MAS planteado busca realizar simultáneamente los procesos de identificación y control de una planta. Esto se consigue adoptando una configuración basada en los reguladores autoajustables (STR).

Tras estos conceptos teóricos, dedicaremos un apartado al empleo de la herramienta de entrenamiento de redes neuronales *Evenet2000* a problemas de control (simulación de sistemas discretos, controladores basados en redes neuronales...). Esta herramienta ha sido desarrollada en trabajos previos en el seno del Grupo de Computadoras y Control de la Universidad de la Laguna por parte del autor de este proyecto de tesis. El motivo de la inclusión de este apartado estriba en que sus módulos han sido empleados en el MAS con la configuración detallada en esta sección.

Antes de pasar a la descripción de la arquitectura de MAS implementada, enumeraremos una serie de trabajos, que consideramos importantes dentro de la aplicación de los MAS al control de procesos.

Terminadas estas secciones previas, pasaremos a detallar el trabajo realizado en el presente proyecto de tesis. Comenzaremos por un prototipo inicial de MAS destinado a controlar la altura de una plataforma incluida en un robot móvil. Este prototipo ha servido de toma de contacto con los sistemas multiagentes y de comparación de su arquitectura con los estándares estudiados. A partir de esta comparativa, se ha

desarrollado un sistema multiagente, bautizado como MASCONTROL, basado en arquitectura FIPA y que implementa un esquema STR de identificación y control de una planta. Todos los detalles del MAS y los resultados obtenidos se detallan a partir de la sección 5.5.

5.1. Consideraciones Previas

5.1.1. Definiciones previas

Antes de entrar de lleno en el sistema multiagente para el control de procesos, en primer lugar definiremos los conceptos empleados a lo largo de este capítulo [Oga96, Oga98, Kuo82]. Estos términos son muy conocidos en el control de procesos, pero son incluidos en este trabajo por completitud y como apoyo al lector ajeno a dicho campo.

Sistema Combinación de componentes que actúan de forma conjunta cumpliendo un determinado objetivo.

Proceso Operación natural o artificial caracterizada por una serie de cambios graduales y progresivamente continuos, que consisten en una secuencia de acciones controladas o movimientos dirigidos sistemáticamente hacia un determinado resultado o fin. Por extensión, denominaremos por este nombre cualquier proceso que se desee controlar.

Planta Equipo que funciona de manera conjunta y cuyo objetivo es realizar una operación determinada. Por extensión, consideraremos una planta como cualquier objeto físico sobre el cual se desea implementar una acción de control.

Variable controlada Cantidad o condición que se mide y controla.

Variable manipulada Cantidad o condición modificada por el controlador, a fin de afectar la variable controlada.

Variables de estado Conjunto mínimo de variables tales que su conocimiento en un instante de tiempo y el conocimiento de la entrada aplicada a partir de ese instante son suficientes para determinar la evolución de estas variables en el futuro.

Variables de salida Variables controladas, o función de ellas, que se desean limitar dentro de unos márgenes preestablecidos.

Comando Valor que toma la acción de control suministrada a la planta.

Ganancia Estática Cociente entre la variación de la salida una vez que el sistema alcanza un régimen estacionario y la variación sufrida por la entrada.

Control Medición del valor de la variable controlada y aplicación al sistema de la variable manipulada para corregir o limitar la desviación del valor medido respecto al valor deseado.

Controlador Sistema que realiza el control sobre la variable controlada.

Consigna Valor que se desea que presente la variable controlada.

Dinámica Conjunto de leyes que establecen cómo cambian las variables de estado de un sistema.

Perturbaciones Cantidades que afectan adversamente a la variable controlada y que no pueden ser manipuladas directamente.

Control automático o retroalimentado Operación que, en presencia de perturbaciones, tiende a reducir la diferencia entre una entrada de referencia y la salida de un sistema empleando esta diferencia, denominada error, como medio de control (Ver Figura 5.1).

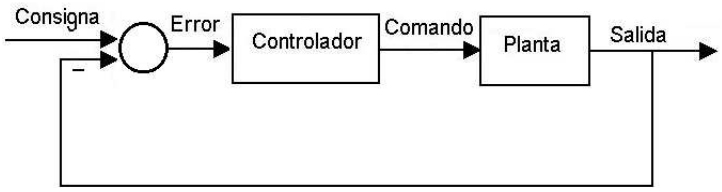

Figura 5.1.: Esquema de control realimentado.

Sistema de control en lazo abierto Sistema en que la salida no tiene efecto sobre la acción de control. Por lo tanto no se produce medición de la salida ni retroalimentación. Como consecuencia, no responden adecuadamente ante la presencia de perturbaciones.

Sistema lineal Sistema cuya dinámica se puede describir mediante ecuaciones diferenciales o en diferencias lineales. A estos sistemas se les puede aplicar el principio de superposición. Este principio indica que la respuesta de un sistema producida por la aplicación simultánea de varias fuentes excitadoras es la suma de las respectivas respuestas individuales.

Sistema no lineal Sistema cuya dinámica se puede describir mediante ecuaciones diferenciales o en diferencias no lineales[1]. Los procedimientos para hallar soluciones a problemas que involucren sistemas no lineales suelen presentar una complejidad alta, por lo que se suelen aproximar a sistemas lineales equivalentes, válidos únicamente en un rango restringido.

Sistema continuo Sistema cuyas señales varían de forma continua en el tiempo. El análisis de estos sistemas se implementa en la disciplina de Control mediante la transformada de Laplace $\mathcal{L}$.

Sistema de control digital o discreto Sistema cuyas señales son muestreadas en el tiempo. El muestreo se realiza regularmente cada T segundos (el periodo de muestreo). El análisis se realiza mediante la transformada *Z*.

[1] De una forma estricta, todos los sistemas reales son no lineales.

Orden de un sistema Orden de la ecuación diferencial (en el caso de sistemas continuos) o en diferencias (en el caso de sistemas discretos) que modela el comportamiento del sistema.

Sistema SISO Sistema de una única entrada y una única salida (*Single Input, Single Output*)

Sistema MIMO Sistema de varias entradas y varias salidas (*Multiple Inputs, Multiple Outputs*)

Sistema estable Sistema cuya salida se encuentra acotada para cualquier entrada acotada.

5.1.2. Modelado de Sistemas

Por simplicidad, en los siguientes apartados nos centraremos en sistemas lineales invariantes en el tiempo cuyo comportamiento viene descrito por una ecuación de coeficientes constantes. Esto no supone restricción alguna en la definición de los conceptos. Por ejemplo la definición de *función de transferencia* es la misma, solamente que las expresiones resultantes en el resto de sistemas se tornan más complicadas.

El modelo matemático de un sistema continuo es una ecuación diferencial, mientras que en sistemas discretos corresponde a una ecuación en diferencias. No obstante, dentro de la teoría de control se prefieren otras representaciones como la función de transferencia y las variables de estado.

5.1.2.1. Función de Transferencia en Sistemas Continuos

La función de transferencia en un sistema continuo SISO *G(s)* se define como el cociente entre las transformadas de Laplace de la salida (*Y(s)*) y la entrada (*U(s)*), suponiendo condiciones iniciales nulas. Así, un sistema gobernado por la ecuación diferencial de orden n

$$\begin{aligned} &\frac{d^n y}{dt^n}(t) + a_{n-1}\frac{d^{n-1}y}{dt^{n-1}}(t) + \cdots + a_0 y(t) \quad = \\ &b_m\frac{d^m u}{dt^m} + b_{m-1}\frac{d^{m-1}u}{dt^{m-1}}(t) + \cdots + b_0 u(t) \end{aligned} \tag{5.1}$$

da lugar a la siguiente forma general de una función de transferencia:

$$G(s) = \frac{Y(s)}{U(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_0} \tag{5.2}$$

con $n \geq m$ para que el sistema sea causal y donde s es una variable compleja.

A partir de la función de transferencia se pueden realizar las siguientes definiciones

Polos del sistema Valores de la variable s que anulan el denominador de la función de transferencia. La importancia de este concepto estriba en que determinan la dinámica del sistema. El número de polos coincide con el orden del sistema, ya que $n \geq m$.

Ceros del sistema Valores que anulan el numerador de la función de transferencia.

Tipo de un sistema Número de polos que el sistema tiene en el origen.

Para un sistema de MIMO se habla de matriz de transferencia, cuyos elementos expresan la función de transferencia entre las distintas salidas y entradas del sistema. Así, para un sistema de r entradas y p salidas

$$G_{ij}(s) = \frac{Y_i(s)}{U_j(s)} \qquad \begin{matrix} i = 1,\ldots,p \\ j = 1,\ldots,r \end{matrix} \tag{5.3}$$

5.1.2.2. Representación en Variables de Estado en Sistemas Continuos

La representación de estado de un sistema de n variables de estado $\bar{X}(t)$ con r entradas $\bar{U}(t)$ y p salidas $\bar{Y}(t)$ consiste en un sistema de ecuaciones llamado ecuación de estado de la forma

$$\dot{\bar{X}}(t) = A\bar{X}(t) + B\bar{U}(t) \tag{5.4}$$

donde las dimensiones de las matrices involucradas son:

- $\bar{X}(t)$ es de dimensión n x 1
- A es de dimensión n x n
- B es de dimensión n x r
- $\bar{U}(t)$ es de dimensión r x 1

Con esta ecuación, la salida del sistema se puede expresar a través de la siguiente relación

$$\bar{Y}(t) = C\bar{X}(t) + D\bar{U}(t) \tag{5.5}$$

donde:

- $\bar{Y}(t)$ es de dimensión p x 1
- C es de dimensión p x n
- D es de dimensión p x r

A partir de la representación de estado se pueden obtener los polos del sistema como los autovalores de la matriz A. La representación en variables de estado no es única.

Un sistema se determina unívocamente a través de su función de transferencia, su representación de estados o el conjunto polos, ceros, ganancia.

5.1.2.3. Función de Transferencia en Sistemas Discretos

La función de transferencia en un sistema SISO discreto *G(z)* se define como el cociente entre las transformadas *Z* de la salida *Y(z)* y la entrada *U(z)*, suponiendo condiciones iniciales nulas. De modo análogo al caso continuo, un sistema gobernado[2] por la ecuación en diferencias de orden *n*

$$\begin{aligned} y_{k+n} + a_{n-1}y_{k+n-1} + \cdots + a_0 y_k \quad &= \\ b_m u_{k+m} + b_{m-1}u_{k+m-1} + \cdots + b_o u_k \end{aligned} \tag{5.6}$$

da lugar a la forma general de una función de transferencia discreta:

$$G(z) = \frac{Y(z)}{U(z)} = \frac{b_m z^m + b_{m-1}z^{m-1} + \cdots + b_0}{z^n + a_{n-1}z^{n-1} + \cdots + a_0} \tag{5.7}$$

con $n \geq m$ para que el sistema sea causal.

En un sistema discreto siguen siendo válidos los conceptos de polos, ceros y matriz de transferencia para sistemas MIMO.

5.1.2.4. Representación en Variables de Estado en Sistemas Discretos

La representación de estado de un sistema discreto de *n* variables de estado proporciona la siguiente ecuación de estado, análoga a la ecuación 5.4

$$\bar{X}_{k+1} = A\bar{X}_k + B\bar{U}_k \tag{5.8}$$

con un análisis dimensional equivalente al de los sistemas continuos (sección 5.1.2.2).

La ecuación de salida del sistema en función de las variables de estado es

$$\bar{Y}_k = C\bar{X}_k + D\bar{U}_k \tag{5.9}$$

5.1.2.5. Forma Canónica Controlable

De las diversas formas que puede tener la representación interna de variables de estado, en este trabajo se ha empleado la de la Forma Canónica Controlable (FCC).

Partiendo de un sistema continuo (el caso discreto es equivalente) con una función de transferencia general G(s)

$$G(s) = \frac{Y(s)}{X(s)} = \frac{b_m s^m + b_{m-1}s^{m-1} + \cdots + b_0}{s^n + a_{n-1}s^{n-1} + \cdots + a_o} \tag{5.10}$$

las matrices de estado adquieren la siguiente forma:

[2] Por cuestiones de simplicidad, en la notación y_k representa *y(kT)* siendo T el periodo de muestreo

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 0 & 1 & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 1 \\ -a_0 & -a_1 & -a_n & \dots & -a_{n-2} & -a_{n-1} \end{bmatrix}$$

$$B = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \\ 1 \end{bmatrix} \tag{5.11}$$

$$C = \begin{bmatrix} b_0 - a_0 b_n & b_1 - a_1 b_n & \dots & b_{n-2} - a_{n-2} b_n & b_{n-1} - a_{n-1} b_n \end{bmatrix}$$

$$D = [b_n]$$

5.1.3. Controladores Implementados

En esta sección describiremos brevemente los dos controladores básicos implementados en el sistema multiagente para el control de procesos presentado en este trabajo: el PID y la realimentación por variables de estado.

5.1.3.1. Controlador PID

Este tipo de controlador es el más utilizado en los procesos industriales. Se compone de la suma de tres acciones: proporcional, integral y derivativa. Son las iniciales de estas acciones las que dan nombre al controlador PID, aunque según sus diferentes combinaciones también dan origen a otros controladores como por ejemplo el P (solamente acción proporcional) o PI (acción proporcional e integral).

$$u(t) = k_p e(t) + k_i \int_0^t e(t)dt + k_d \frac{de(t)}{dt} \tag{5.12}$$

donde *e(t)* representa al error en un lazo realimentado (diferencia entre la entrada de referencia y la salida del sistema).

Estas tres acciones dotan al controlador de una gran versatilidad, permitiendo controlar la forma de la salida del sistema controlado e incluso evitar que esta salida sobrepase la consigna en ningún momento así como eliminar el error en régimen permanente. La forma general de la función de transferencia continua de un controlador PID es

$$PID(s) = k_p + \frac{k_i}{s} + k_d s \tag{5.13}$$

siendo los valores k_p, k_i y k_s conocidos como los *parámetros* del controlador.

- La acción proporcional permite aplicar un mayor comando cuanto mayor sea el error.
- La acción integral permite eliminar los errores estacionarios en régimen permanente.
- La acción derivativa detecta anticipadamente una elevada velocidad de reducción del error, tendiendo a corregirla, y evitando de este modo que se rebase en exceso el valor de referencia deseado. Esta acción presenta el inconveniente de producir inestabilidad en presencia de ruidos en el sistema, ya que los amplifica.

La función de transferencia discreta correspondiente a un PID [Jac81] con periodo de muestreo T es

$$PID(z) = \frac{U(z)}{E(z)} = \frac{\alpha + \beta z^{-1} + \gamma z^{-2}}{1 - z^{-1}} \tag{5.14}$$

donde

$$\begin{aligned} \alpha &= k_p + \frac{k_i T}{2} + \frac{k_d}{T} \\ \beta &= \frac{k_i T}{2} - k_p - \frac{2k_d}{T} \\ \gamma &= \frac{k_d}{T} \end{aligned} \tag{5.15}$$

5.1.3.2. Realimentación por variables de estado. Asignación de polos.

Es un sistema de control que se basa en tomar las variables de estado del sistema a controlar y, a partir de ellas, generar el comando del sistema. Es decir, lo que se realimenta en el controlador no es la salida del sistema sino su vector de variables de estado *X(k)*.

Por tanto, en caso de consigna nula el comando *U(k)* enviado a la planta viene dado por

$$\bar{U}_k = -K\bar{X}_k \tag{5.16}$$

donde K es una matriz de ganancias que debe garantizar el estado final $\overline{X} = 0$ (sistema estable), existiendo diferentes técnicas para su diseño.

Una de ellas es la técnica de asignación de polos. Sean p_1, p_2... p_n los polos que determinan el comportamiento deseado del sistema realimentado. El valor de K viene dado por la denominada fórmula de Ackermann:

$$K = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} B & AB & \cdots & A^{n-1}B \end{bmatrix} \alpha_c(A) \tag{5.17}$$

siendo $\alpha_c(z)$ el denominado *polinomio característico*, el cual sitúa los polos del sistema realimentado en los valores deseados.

$$\alpha_c(z) = (z - p_1)(z - p_2)\cdots(z - p_n) \tag{5.18}$$

Para consignas $\bar{U}_k$ no nulas la ley de control es:

$$\bar{U}_k = -K\bar{X}_k + Q\bar{R}_k \tag{5.19}$$

donde Q es un factor de ajuste de la ganancia del sistema.

$$Q = \left((C - DK)(I - A + BK)^{-1} B + D \right)^{-1} \tag{5.20}$$

5.1.4. Identificación de Sistemas

5.1.4.1. Introducción

Básicamente existen dos enfoques para el desarrollo de modelos matemáticos que describan el comportamiento de un sistema:

- *Modelo Teórico*. Empleo de relaciones de carácter físico, químico... que permiten derivar un modelo. Por ejemplo, el uso de la segunda ley de Newton para el modelo de la dinámica de un péndulo.
- *Modelo Empírico*. Identificación basada únicamente en los datos de entrada-salida observados.

El primero de ellos adolece de dos grandes inconvenientes. En primer lugar, se necesita un conocimiento profundo del sistema que se desea modelar, lo cual no es siempre posible. En segundo lugar, se suele caer en un modelo idealizado que no tiene en consideración las perturbaciones aleatorias del sistema.

En contraste, el segundo enfoque requiere muy poca información acerca del sistema y toma en cuenta las perturbaciones, pero necesita de una evaluación continua. Es precisamente esta técnica la empleada en este trabajo en cuanto a identificación de sistemas se refiere.

El modelo empleado en la identificación puede ser básicamente de dos tipos:

Paramétricos Son aquéllos ligados a una estructura matemática en términos de una ecuación diferencial o en diferencias, matrices de transferencia... donde uno o varios parámetros sirven para caracterizar el sistema.

No paramétricos Estos modelos expresan dependencias racionales como un conjunto de valores calculados.

El empleo de uno u otro tipo de modelo depende tanto de la aplicación que vaya a usar el modelo como del conocimiento a priori del sistema. Así, por ejemplo, para propósitos de control, como lo es el caso de este trabajo, suele bastar con un modelo paramétrico [RS02].

5.1.4.2. Consideraciones sobre la entrada en un proceso de identificación

Para que un proceso de identificación sea efectivo, la señal de entrada debe contener el mayor número de frecuencias posibles. Así, una señal senoidal pura no es adecuada en un experimento de identificación, puesto que de este modo sólo se obtendrá la

respuesta del sistema para la frecuencia de dicha señal. Por el contrario, las señales escalonadas son adecuadas, puesto que contienen un espectro suficientemente amplio de frecuencias.

5.1.4.3. Nomenclatura

Dentro del enfoque de identificación de sistemas basada únicamente en los datos de entrada-salida observados (utilizado en esta parte de nuestro trabajo), introduciremos la siguiente nomenclatura:

- u: Datos de entrada
- y: Datos de salida
- e: Ruido blanco
- v: Posible ruido de color

La presencia de los dos últimos términos se justifica en que en la mayoría de sistemas: la salida no se encuentra determinada únicamente por los valores de entrada sino, además, por la presencia de perturbaciones en el sistema.

5.1.4.4. Criterios de Estimación de Parámetros

En un modelo paramétrico se necesita establecer el criterio mediante el cual se calculan los parámetros del sistema. Existen tres métodos básicos para esto:

- Predicción del error: Los parámetros se calculan de forma que la diferencia entre la salida del modelo y la salida real del sistema es mínima. Este método es el que se va a emplear en nuestro trabajo.
- Método de subespacio: Se basa en que el estado contiene, en cierto sentido, toda la información pasada que es relevante para el futuro.
- Correlación: Se calculan los parámetros de tal forma que el error de predicción sea independiente de los datos anteriores de entrada-salida.

5.1.4.5. Modelo de Función de Transferencia Racional

En este modelo, la relación entre los datos de entrada y salida del sistema se relacionan mediante una ecuación lineal de la forma

$$y_k = G(q^{-1})u_k + H(q^{-1})e_k \tag{5.21}$$

donde q^{-1} representa un operador de retraso temporal, es decir

$$q^{-1}u_k = u_{k-1} \tag{5.22}$$

$$q^{-l}u_k = u_{k-l} \tag{5.23}$$

y donde $G(q^{-1})$ y $H(q^{-1})$ son las funciones de transferencia de la dinámica del sistema y de las perturbaciones respectivamente. Podemos representar estas funciones como

$$G(q^{-1}) = \sum_{j=0}^{\infty} g(j)q^{-j} \tag{5.24}$$

$$H(q^{-1}) = \sum_{j=0}^{\infty} h(j)q^{-j} \tag{5.25}$$

Para la estimación de los respectivos modelos, supondremos un vector θ que contenga los parámetros de los mismos. De esta forma la ecuación 5.21 se puede expresar de la forma:

$$y_k = q^{-d}G(q^{-1},\theta)u_k + H(q^{-1},\theta)e_k \tag{5.26}$$

donde el término q^{-d} refleja un posible retardo de d periodos en la entrada y donde $G(q^{-1},\theta)$ y $H(q^{-1},\theta)$ se pueden expresar como un cociente de polinomios del siguiente modo:

$$G(q^{-1},\theta) = \frac{B(q^{-1},\theta)}{A(q^{-1},\theta)} \tag{5.27}$$

$$H(q^{-1},\theta) = \frac{C(q^{-1},\theta)}{D(q^{-1},\theta)} \tag{5.28}$$

Los numeradores y denominadores de estas funciones son de la forma:

$$B(q^{-1},\theta) = b_1q^{-1} + \ldots + b_{n_b}q^{-n_b} \tag{5.29}$$

$$A(q^{-1},\theta) = 1 + a_1q^{-1} + \ldots + a_{n_a}q^{-n_a} \tag{5.30}$$

$$C(q^{-1},\theta) = 1 + c_1q^{-1} + \ldots + c_{n_c}q^{-n_c} \tag{5.31}$$

$$D(q^{-1},\theta) = 1 + d_1q^{-d} + \ldots + d_{n_d}q^{-n_d} \tag{5.32}$$

Con $n_b \leq n_a$ y $n_c \leq n_d$ para que el sistema cumpla la condición de causalidad. Para resultados comentados en secciones posteriores se impondrá que $H(q^{-1})$ sea invertible.

Con estas definiciones, el vector θ contiene los coeficientes de los polinomios de las ecuaciones 5.29-5.32:

$$\theta = [a_1,\cdots,a_{n_a},b_1,\cdots,b_{n_b},c_1,\cdots,c_{n_c},d_1,\cdots,d_{n_d}] \tag{5.33}$$

La ecuación 5.26 da lugar a una serie de estructuras de modelos, que describiremos brevemente en los siguientes subapartados.

Autoregresivo con entrada exógena (*Auto Regressive with eXogeneous input*, ARX)

Este modelo presenta la siguiente estructura, ampliamente usada para propósitos de control y simulación, con $C(q^{-1},\theta)=1$ y $D(q^{-1},\theta)=A(q^{-1},\theta)$:

$$y_k = q^{-d}\frac{B(q^{-1},\theta)}{A(q^{-1},\theta)}u_k + \frac{1}{A(q^{-1},\theta)}e_k \tag{5.34}$$

Las ecuaciones 5.29-5.30 permiten escribir la ecuación 5.34 del siguiente modo:

$$y_k + a_1 y_{k-1} + \cdots + a_{n_a} y_{k-n_a} \quad = \quad b_1 u_{k-d-1} + \cdots + b_{n_b} u_{k-d-n_b} + e_k \tag{5.35}$$

De forma que, definiendo el vector de regresión φ_k como

$$\varphi_k = [-y_{k-1},\cdots,-y_{k-n_a},u_{k-d-1},\cdots,u_{k-d-n_b}]^T \tag{5.36}$$

obtenemos la expresión del modelo de estructura ARX de la forma de regresión lineal

$$y_k = \varphi_k^T\theta + e_k \tag{5.37}$$

Esta forma de regresión lineal se muestra particularmente útil en la estimación del modelo de estructura ARX.

Respuesta de impulso finita (*Finite Impulse Response*, FIR)

En esta estructura de modelo, $A(q^{-1},\theta)=$ $C(q^{-1},\theta)=$ $D(q^{-1},\theta)=$ 1. Por tanto, sustituyendo en 5.26, se obtiene

$$y_k = q^{-d}B(q^{-1},\theta)u_k + e_k \tag{5.38}$$

Es decir, el valor de la salida viene determinado por los valores de las entradas anteriores corrompidas por ruido blanco. Como se puede comprobar de modo sencillo, esta estructura FIR, al igual que la ARX, permite el empleo de la forma de regresión lineal.

Error de la salida (*Output Error*)

La expresión de esta estructura de modelo es la siguiente:

$$y_k = q^{-d}\frac{B(q^{-1},\theta)}{A(q^{-1},\theta)}u_k + e_k \tag{5.39}$$

A diferencia del ARX (ecuación 5.34) permite la estimación de un denominador en la dinámica del modelo. Sin embargo, difiere del mismo en que el modelo de la dinámica de las perturbaciones no posee polos comunes con la del sistema.

En esta estructura de modelo no se puede llegar a una expresión de regresión lineal, siendo por tanto algo más engorrosa la estimación de los parámetros θ.

Autoregresivo de media móvil con entrada exógena (*Auto Regressive Moving Average with eXogeneous input*, ARMAX)

La expresión de este modelo implica que $D(q^{-1},\theta)$=$A(q^{-1},\theta)$.

$$y_k = q^{-d}\frac{B(q^{-1},\theta)}{A(q^{-1},\theta)}u_k + \frac{C(q^{-1},\theta)}{A(q^{-1},\theta)}e_k \tag{5.40}$$

Es decir, ambas dinámicas, la correspondiente al modelo del sistema como la de sus perturbaciones presentan polos comunes. Esta suposición se muestra como importante en sistemas donde la perturbación se produce al inicio del proceso de obtención de la salida. De nuevo, esta forma no permite su expresión en forma de regresión lineal, en esta ocasión debido a por la inclusión del término $C(q^{-1},\theta)$. Por este motivo, es más complicado la obtención de los parámetros.

Box Jenkins

Este modelo corresponde a la forma más general de 5.26:

$$y_k = q^{-d}\frac{B(q^{-1},\theta)}{A(q^{-1},\theta)}u_k + \frac{C(q^{-1},\theta)}{D(q^{-1},\theta)}e_k \tag{5.41}$$

De los modelos presentados en esta sección, éste es el menos sensible a las perturbaciones no blancas ya que se puede modelar las dos dinámicas de forma independiente, pero a costa del aumento de la complejidad del cálculo [Bec].

Predicción de la Salida

En orden a estimar los parámetros θ en un modelo de función de transferencia racional (ecuación 5.26) debemos derivar un predictor para la salida. En este sentido, introduciremos el *predictor de un paso adelante* (*one step ahead predictor*)[3].

$$\widehat{y}_{k|k-1} = q^{-d}H^{-1}(q^{-1},\theta)G(q^{-1},\theta)u_k + \left[1 - H^{-1}(q^{-1},\theta)\right]y_k \tag{5.42}$$

Por cuestión de simplicidad en la notación denotaremos este predictor como $\widehat{y}_k$.
Insertando la ecuación 5.26 en 5.42 se obtiene

$$\begin{aligned}
\widehat{y}_k &= q^{-d}H^{-1}(q^{-1},\theta)G(q^{-1},\theta)u_k + \left[1 - H^{-1}(q^{-1},\theta)\right]y_k \\
&= q^{-d}H^{-1}(q^{-1},\theta)G(q^{-1},\theta)u_k \\
&+ \left[1 - H^{-1}(q^{-1},\theta)\right]\left[q^{-d}G(q^{-1},\theta)u_k + H(q^{-1},\theta)e_k\right] \\
&= q^{-d}G(q^{-1},\theta)u_k + H(q^{-1},\theta)e_k - e_k \\
&= y_k - e_k
\end{aligned} \tag{5.43}$$

Es decir, que la diferencia entre los valores de la salida predicha y la real consiste en la perturbación de ruido blanco en el sistema. Además al ser imposible por definición

[3] Se pueden considerar predictores $\widehat{y}_{t|t-k}$ de un número arbitrario de pasos k, pero éstos no serán tratados en este trabajo. Más adelante se demostrará la independencia respecto a y_k

la predicción $\hat{e}_{k|k-1}$ se puede concluir que la ecuación 5.42 corresponde a un predictor óptimo.

Para asegurar que efectivamente que la predicción $\widehat{y}_k$ depende únicamente de los datos de salida anteriores al periodo t, se impone que[4]

$$H^{-1}(q^{-1},\theta) = \sum_{j=0}^{\infty} \widetilde{h}(j)q^{-j}, \quad h(0) = 1 \tag{5.44}$$

Por tanto, sustituyendo 5.44 en 5.42

$$\widehat{y}_k = q^{-d}\left[\sum_{j=0}^{\infty} \widetilde{h}(j)q^{-j}\right] G(q^{-1},\theta)u_k + \left[\sum_{j=1}^{\infty} \widetilde{h}(j)q^{-j}\right] y_k \tag{5.45}$$

lo cual es claramente independiente de y_k.

La ecuación 5.45 indica que el predictor precisa de los datos de entrada-salida desde t=-∞, aunque desde un punto de vista práctico suele bastar considerar desde t=0.

Criterio de mínimos cuadrados

Una vez definido el modelo de estructura y el predictor a emplear, queda por determinar un criterio que permita calcular el vector de parámetros θ y con él la identificación del sistema. Entre los métodos existentes, el más popular es el de mínimos cuadrados. Este método, aplicado al problema que estamos analizando, define una función de coste a minimizar en θ de la forma

$$V_N(\theta) = \frac{1}{2N}\sum_{k=1}^{N}(y_k - \widehat{y}_k)^2 \tag{5.46}$$

expresión que, teniendo en cuenta la ecuación 5.42 puede ser expresada como sigue:

$$V_N(\theta) = \frac{1}{2N}\sum_{k=1}^{N}\left[H(q^{-1},\theta)\left[y_k - q^{-d}G(q^{-1},\theta)u_k\right]\right]^2 \tag{5.47}$$

5.1.5. Identificación recursiva en tiempo real

Debido al problema a tratar, de los diversos métodos de identificación prestaremos especial atención a los métodos de identificación recursiva en tiempo real, adecuado cuando las propiedades de la planta pueden cambiar (de modo lento) con el tiempo. Según Söderström y Stoica [SS89] existen dos enfoques principales para tratar este problema:

1. Empleando un factor de olvido
2. Empleando un filtro de Kalman como un estimador de parámetros

Desglosaremos el método del factor de olvido por ser el método empleado en la implementación del sistema multiagente.

[4] El primer coeficiente tanto de $C(q^{-1},\theta)$ como de $D(q^{-1},\theta)$ es 1.

5.1.5.1. Factor de olvido

En este caso, la función de coste a minimizar adquiere la siguiente forma:

$$V_N(\theta) = \frac{1}{2N}\sum_{k=1}^{N} \lambda^{N-t}(y_k - \widehat{y}_k)^2 \tag{5.48}$$

La cual corresponde a la ecuación 5.46 cuando $\lambda = 1$. A este factor se le denomina *factor de olvido*. Este número, menor o igual que la unidad, indica que a medida que se incrementa el tiempo, las medidas realizadas anteriormente pierden importancia (se *olvidan*). Un menor valor del factor de olvido implica una pérdida más rápida de la información.

La estimación de los parámetros θ puede llevarse a cabo mediante un proceso iterativo, a través de las siguientes ecuaciones [SS89], siempre suponiendo un modelo que permita la forma de regresión lineal indicada por la ecuación 5.37:

$$\begin{aligned}
\widehat{\theta}(k) &= \widehat{\theta}(k-1) + K(k)\varepsilon(k) \\
\varepsilon(k) &= y(k) - \varphi^T(k)\widehat{\theta}(k-1) \\
K(k) &= \frac{P(k-1)\varphi(k)}{\lambda + \varphi^T(k)P(k-1)\varphi(k)} \\
P(k) &= \frac{1}{\lambda}\left(P(k-1) - \frac{P(k-1)\varphi(k)\varphi^T(k)P(k-1)}{\lambda + \varphi^T(k)P(k-1)\varphi(k)}\right)
\end{aligned} \tag{5.49}$$

En cada etapa de identificación k, se calcula a partir de la matriz $\varphi(k)$, obtenida experimentalmente, sucesivamente los valores de $P(k)$, $K(k)$, $\varepsilon(k)$y finalmente el vector estimado de parámetros $\hat{\theta}(k)$.

5.1.6. Reguladores autoajustables (*Self-Tunning Regulators*)

Los reguladores autoajustables (*self-tunning regulators*, STR) consisten en un esquema de control en el que en tiempo real se estima el sistema a controlar y se determinan los mejores parámetros del controlador [BRK94].

El STR se compone de dos bucles:

- Un bucle interno que engloba el proceso y un sistema de control realimentado.
- Un bucle externo que se compone de un estimador de parámetros de la planta y un diseñador del controlador.

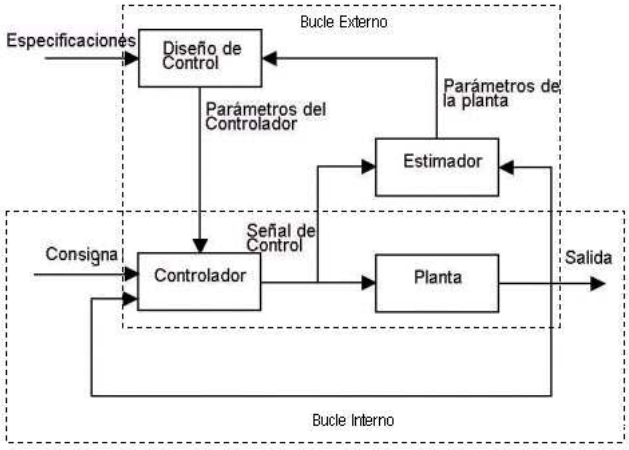

Figura 5.2.: Esquema STR.

El esquema del algoritmo del STR se compone de dos pasos (ver Figura 5.2):

1. Estimar los parámetros de la planta.

2. Estimar y actualizar los parámetros del controlador.

5.2. Empleo de *Evenet2000* en problemas de Control

Antes de describir el sistema basado de agentes presentado en este trabajo para la identificación y control de sistemas, consideramos altamente ilustrativo describir el empleo de la herramienta *Evenet2000*, detallada en el apéndice J, en problemas relacionados con el control de procesos. El motivo para ello radica en el empleo diversas clases de la herramienta en el sistema multiagente con las funcionalidades analizadas en esta sección [GHM$^+$03b, GHMM03].

Muchos problemas de identificación y control de sistemas se basan en la optimización de algunos parámetros. De la misma forma, el entrenamiento de redes neuronales consiste en encontrar los mejores valores para los pesos de la red. Debido a esta similitud, se ha planteado considerar la aplicación de los métodos de entrenamiento de redes neuronales a los problemas de control. Como en el caso de las redes neuronales, uno de los aspectos más complicados en esta aplicación radica en el establecimiento de las fórmulas del gradiente.

En esta sección, presentaremos un conjunto de problemas donde la utilización de *Evenet2000* se ha mostrado altamente útil.

5.2.1. Simulación de Sistemas

Evenet2000 permite simular la evolución de un sistema discreto. Como ejemplo, hemos considerado un sistema de tanques interconectados, tal como se muestra en la Figura 5.3. Esta planta consiste en un sistema SISO cuya entrada es el flujo de agua q

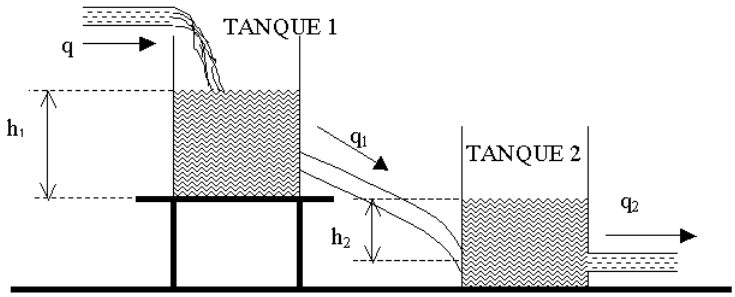

Figura 5.3.: Sistema de tanques interconectados.

que se introduce en el Tanque 1 y cuya salida es la altura h_2 que alcanza el agua en el Tanque 2.

La función de transferencia para este sistema es:

$$\frac{H_2(s)}{Q(s)} = \frac{R_2}{(1+R_1C_1s)(1+R_2C_2s)+C_1R_2s} \tag{5.50}$$

donde

- R_i representa la resistencia al paso de agua en el tanque *i.*
- C_i representa la sección del tanque *i.*

Tomando los valores de $R_1 = R_2 = 3$ sg·cm^{-2}, $C_1 = 10$ cm^2 y $C_2 = 7$ cm^2 y aplicando transformación bilineal con periodo de T=1 segundo, se obtiene la siguiente función de transferencia discreta.

$$\frac{H_2(z)}{Q(z)} = \frac{0{,}001118+0{,}002236z^{-1}+0{,}001118z^{-2}}{1-1{,}8777487z^{-1}+0{,}8792396z^{-2}} \tag{5.51}$$

Esta ecuación puede ser implementada a través de los módulos de *Evenet2000*, tal como se muestra en la Figura 5.4.

Con esta implementación, es posible la simulación del comportamiento del sistema, por ejemplo ante una entrada escalón. Para este caso, se ha tomado un fichero de patrones de 600 pares de entrenamiento con entrada 1 y una salida arbitraria, puesto que no influye en el aprendizaje. Se lleva a cabo un entrenamiento con 1 etapa y paso de entrenamiento 0, es decir no se realiza entrenamiento alguno. Los resultados obtenidos se muestran en la Figura 5.5. Estos resultados han sido testeados con los obtenidos a través de otros programas de simulación como MATLAB[5] u Octave[6].

[5] http://mathworks.com/
[6] http://octave.org

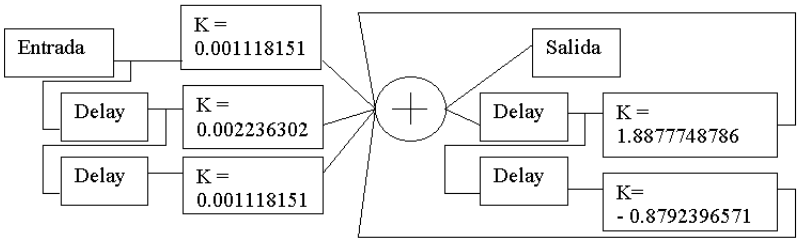

Figura 5.4.: Implementación del sistema de tanques interconectados a través de la herramienta Evenet2000.

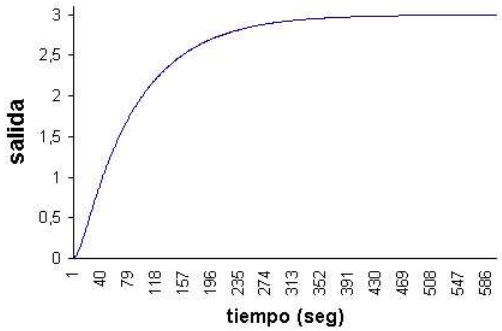

Figura 5.5.: Resultados obtenidos para el sistema de tanques simulado a través de la herramienta *Evenet2000*.

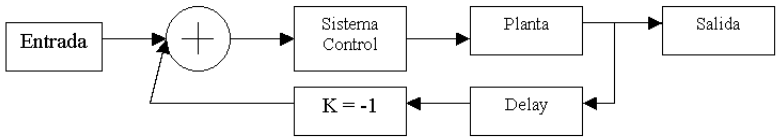

Figura 5.6.: Esquema de control en lazo cerrado.

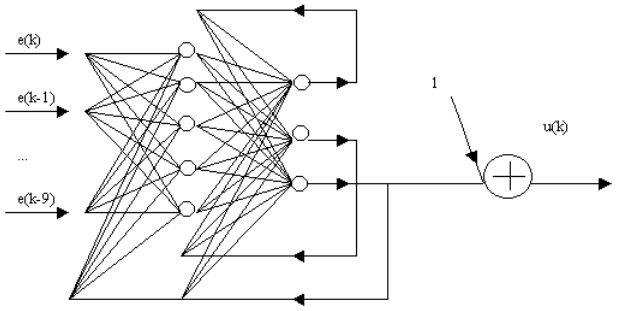

Figura 5.7.: Esquema de red neuronal empleada como controlador.

5.2.2. Controladores basados en redes neuronales

En este ejemplo, emplearemos *Evenet2000* para el diseño de un controlador basado en redes neuronales. Como planta seguiremos tomando el modelo de tanques interconectados visto en la sección 5.2.1. Una vez que la planta ha sido implementada mediante los módulos de *Evenet2000*, se salva este modelo en un fichero de texto para su posterior utilización en otros diseños, tal como indica la sección J.8. En concreto, el sistema grabado representará el papel de la planta en un esquema de control en lazo cerrado, tal como se muestra en la Figura 5.6.

Para el controlador se ha optado por una red neuronal cuyo esquema se muestra en la Figura 5.7. Se ha escogido una arquitectura de red neuronal alejada de los estándares para demostrar la capacidad de la herramienta para entrenar cualquier red de arquitectura arbitraria. La función de activación en sus neuronas es la tangente hiperbólica.

Asimismo, se ha dotado a la red neuronal de capacidad de memoria. La entrada al controlador, esto es, el error del proceso de control *e(k)*, se retrasa de modo sucesivo hasta 9 veces. Por lo tanto, en realidad, la red neuronal presenta 10 entradas correspondientes a *e(k)*, *e(k-1)*, $\cdots$, *e(k-9)*. Las diez entradas se conectan a una capa de cinco neuronas que se conecta a su vez totalmente a una capa de tres neuronas. La salida de la tercera neurona se incluye como entrada realimentada en la primera y segunda capas. Mientras, la salida de la primera y segunda neuronas también intervienen en la realimentación, pero únicamente en la segunda capa.

La salida de la tercera neurona está afectada por un peso y trasladada para generar la salida de la red neuronal, o lo que es lo mismo, el comando del sistema de control.

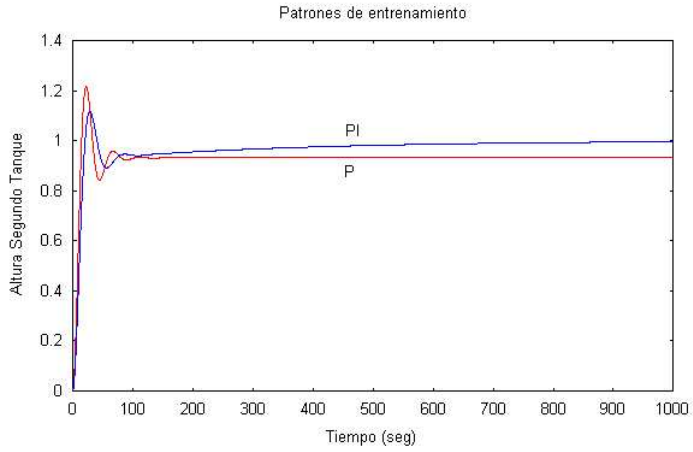

Figura 5.8.: Patrones provenientes del uso de un controlador proporcional y un controlador proporcional-integral.

Como se puede comprobar, con esta arquitectura el cálculo de las fórmulas para algoritmos basados en el gradiente no es sencillo. Sin embargo, se debe tener en cuenta que la herramienta *Evenet2000* libera a sus usuarios de este cálculo.

En la búsqueda de obtener mejores resultados para el entrenamiento, los pesos de la red neuronal se han optimizado para aprender el patrón obtenido del sistema global en dos casos:

- con un controlador proporcional con controlador k_p= 4.5.
- con un controlador PI con un valor para los parámetros de k_p= 3 y T_i= 0.003

en ambos casos con un valor de consigna[7] igual a 1.

Estos patrones de entrenamiento se muestran en la Figura 5.8.

En el entrenamiento de la red se han utilizado como algoritmos de aprendizaje el Gradiente Conjugado con Sección de Oro para la optimización del paso de entrenamiento. El criterio de parada consiste en 12000 iteraciones con un vector de pesos inicial aleatorio.

Tras el entrenamiento se presenta al sistema una consigna variable, buscando comprobar su bondad. Los valores de la consigna se escogen de forma que se someta al sistema consignas tanto mayores como menores que la consigna de entrenamiento. La respuesta del sistema ante estos valores se refleja en la Figura 5.9.

Como se puede comprobar, en el caso del patrón obtenido con un P, la salida se sitúa siempre por debajo de la consigna, de modo que el error en régimen permanente aumenta a medida que lo hace la consigna.

Por otro lado, en el caso del patrón obtenido con un PI, la salida estacionaria es cercana al valor de referencia (menos de un 5 %) en todos los casos. Sin embargo, la salida es mayor que la referencia cuando ésta es menor que la de entrenamiento, ocurriendo lo opuesto en el caso contrario.

[7]Los patrones a entrenar han sido obtenidos con la propia herramienta *Evenet2000* del modo visto en 5.2.1.

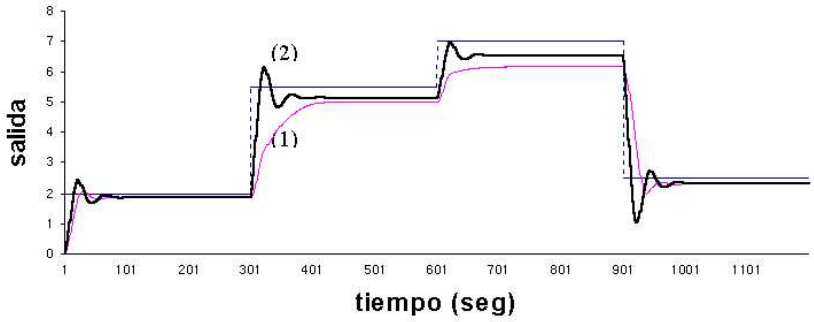

Figura 5.9.: Respuesta del sistema global entrenado ante una consigna variable con un patrón proveniente del uso de un controlador proporcional (1) y un controlador proporcional-integral (2).

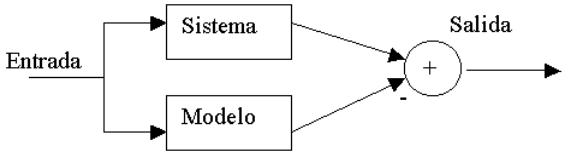

Figura 5.10.: Esquema para la identificación de sistemas con *Evenet2000*.

5.2.3. Identificación de Sistemas

Evenet2000 permite también llevar a cabo identificaciones de sistemas mediante una red neuronal (en realidad a través de cualquier sistema con unos parámetros a optimizar).

El problema se enfoca como un problema de optimización con el esquema mostrado en la Figura 5.10, implementado mediante módulos de *Evenet2000*. La salida deseada en el entrenamiento del sistema global debe ser 0, ya que de este modo se fuerza a que el comportamiento del sistema a modelar y el del modelo sea mínima. Para mejorar el resultado de la identificación, la entrada debe ser lo más rica posible, por ejemplo una suma de señales sinusoidales de diferentes frecuencias o una señal aleatoria. Así el entrenamiento incluye más información respecto al comportamiento del sistema a modelar.

El bloque correspondiente al sistema real puede ser implementado a través de un objeto Java que proporcione el comportamiento del sistema (por ejemplo la lectura on-line de una planta real). De hecho, se ha realizado una comprobación de la capacidad para identificar sistemas empleando la implementación de tanques interconectados visto en 5.2.1. Como modelo a optimizar se toma dicha implementación sustituyendo los coeficientes constantes por pesos. Como era de esperar, en el entrenamiento realizado los valores de los pesos convergen a los de las constantes.

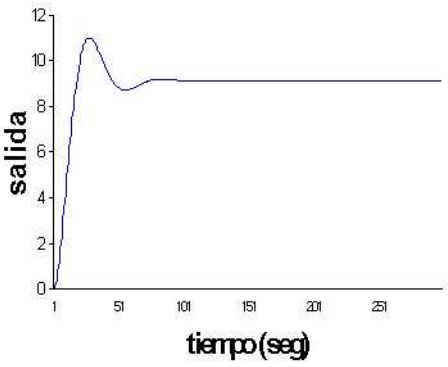

Figura 5.11.: Salida del sistema tras un entrenamiento del controlador imponiendo limitaciones en la función de coste.

5.2.4. Función de Coste

En los sistemas reales con frecuencia existen limitaciones físicas. Así por ejemplo, el comando suele encontrarse limitado entre ciertos valores. En el caso de los tanques interconectados sería deseable que la altura del segundo tanque no superase un cierto valor, evitando el consiguiente desbordamiento de agua.

Estas limitaciones pueden ser contempladas en *Evenet2000*. Para ello se define una función de coste que proporcione un valor alto cuando la variable se encuentre fuera de los límites deseados. Si se van a emplear algoritmos de entrenamiento que involucren al gradiente de la función de coste, ésta debe mantener la continuidad y derivabilidad en los puntos frontera. En cambio, para el resto de algoritmos como por ejemplo el Camino Aleatorio, el valor de la función de coste fuera de los límites deseados puede ser tan simple como una constante de alto valor.

En la Figura 5.11 muestra el comportamiento de la salida del sistema de tanques con consigna de valor 10 y tras el entrenamiento de los parámetros del controlador del apartado 5.2.2 imponiendo una limitación en el valor de la salida de 11.

5.2.5. Optimización en Problemas de Control

Como los módulos de *Evenet2000* han sido diseñados para los problemas de optimización, también pueden ser aplicados a problemas de control que involucren una optimización. Por ejemplo, encontrar los parámetros de un controlador PID que minimicen una determinada función de coste.

En este sentido, se han entrenado sistemas de control P, PI y PID con la estructura de lazo cerrado para minimizar tres funciones de coste diferentes: cuadrática en el error, valor absoluto del error y tiempo por valor absoluto del error. El fichero de entrenamiento consiste en 1000 pares cuya salida deseada es igual a la entrada (es decir, la consigna)[8] con valor 10. Con esta imposición se pretende que el sistema alcance la consigna lo más rápidamente posible pero penalizando los grandes sobrepasamientos (esta penalización varía según la forma de la función de coste escogida).

[8]esto es pares de la forma *consigna*, *consigna*

Cuadro 5.1.: Resultados obtenidos en la optimización de controladores P, PI y PID minimizando diferentes funciones de coste.

	$J = \sum_k (e(k))^2$	$J = \sum_k \lvert e(k)\rvert^2$	$J = \sum_k k\lvert e(k)\rvert^2$
P	k_p=9.359 Primer pico=14.855 Valor Final=9.687	k_p=14.571 Primer pico=16.585 Valor Final=9.797	k_p=19.650 Primer pico=17.965 Valor Final=9.848
PI	k_p=7.031 T_i=3.313·10^{-3} Primer pico=14.074 Valor Final=10	k_p=4.265 T_i=1.092·10^{-2} Primer pico=13.120 Valor Final=10	k_p=2.399 T_i=1.229·10^{-2} Primer pico=11.700 Valor Final=10
PID	k_p=10.034 T_i=1.793·10^{-2} T_d=12.817 Primer pico=12.455 Valor Final=10	k_p=11.014 T_i=1.114·10^{-2} T_d=7.488 Primer pico=11.205 Valor Final=10	k_p=9.395 T_i=1.117·10^{-2} T_d=7.294 Primer pico=10.565 Valor Final=10

Los resultados obtenidos de estos entrenamientos se muestran en la Tabla 5.1 y en las Figuras 5.12, 5.13 y 5.14.

5.3. Otras aplicaciones de los MAS al control de procesos

Varios modelos de MAS han sido propuestas en el campo de la industria. Sin embargo, las aplicaciones de la tecnología de agentes en la automatización de procesos no han sido tan numerosas. Las razones para ello se pueden resumir en las siguientes [SAV+02]:

- Las necesidades de tiempo real de las aplicaciones de automatización de procesos son díficilmente alcanzables por la tecnología actual sobre agentes.
- Las características de las tareas de control de procesos, con interrelaciones entre las diversas variables controladas. Esto dificulta descomponer el problema de un modo adecuado para la aplicación de los agentes.
- La dificultad de encontrar paralelismo en el control de procesos. Este paralelismo podría ser modelado mediante el empleo de agentes.

Aún a pesar de estos motivos, existen algunas aplicaciones que, por su relación con el sistema de control presentado en este capítulo, referenciamos a continuación.

- Un tipo significativo de MAS aplicado al control de procesos es aquél en el que se emplean las técnicas de comunicación entre agentes como un mecanismo de

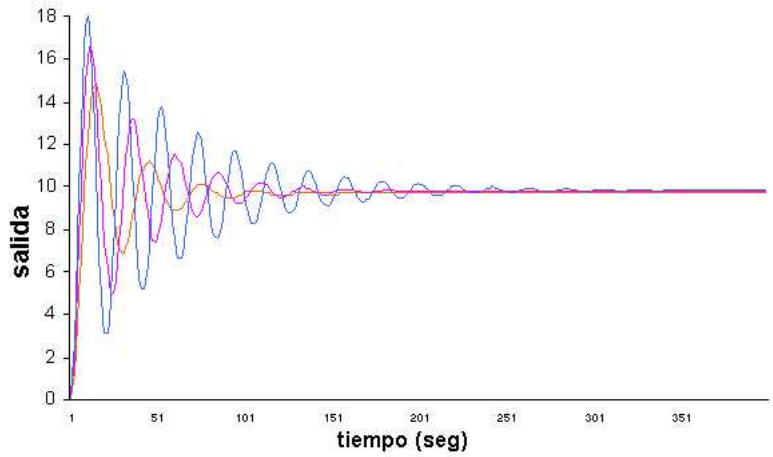

Figura 5.12.: Salida del sistema con controladores P, PI y PID minimizando una función de coste cuadrática en el error.

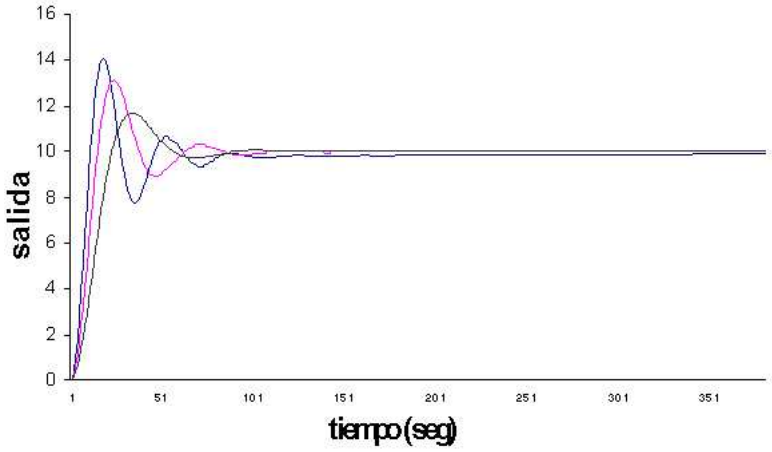

Figura 5.13.: Salida del sistema con controladores P, PI y PID minimizando una función de coste de valor absoluto del error.

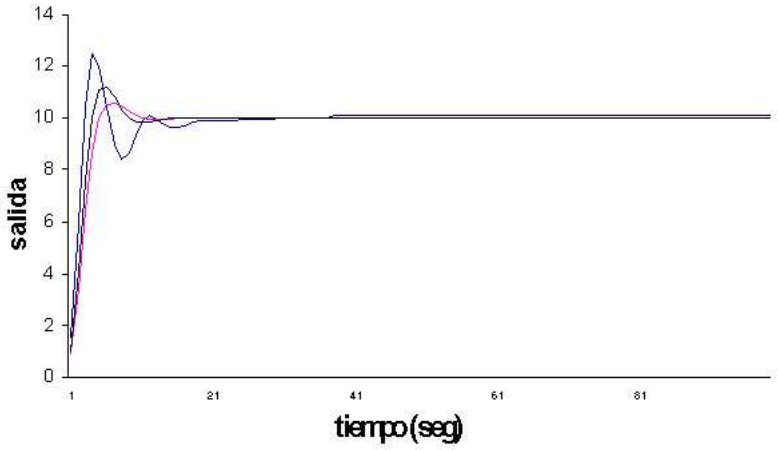

Figura 5.14.: Salida del sistema con controladores P, PI y PID minimizando una función de coste de tiempo por valor absoluto del error.

integración entre sistemas diseñados por separado. El ejemplo más conocido de este tipo es ARCHON [9] [CJ96]. En este sistema, se dota a cada programa de aplicación (conocido como *Intelligent System*, IS) de una capa (*Archon Layer*, AL) que permite las comunicaciones entre ellos. Un esquema de la arquitectura puede verse en la Figura 5.15.

- Otro tipo de aplicación de los MAS al control de procesos lo constituyen los sistemas supervisores. Un ejemplo de este tipo es APACS[10]. En este MAS, el agente de usuario emplea técnicas de comunicación de agentes con el fin de operar con otros agentes, los cuales proporcionan ciertas funciones de supervisión (adquisición de datos, monitorización...).

- Un tercer grupo de aplicaciones son las que implementan un control en lazo cerrado. En este sentido, citaremos el trabajo de Velasco et. al. [VGMI46], para el control de una central térmica. El objetivo de este sistema de control es reducir el consumo de combustible mientras que mantiene constante la potencia generada. Este sistema presenta dos problemas. El primero de ellos es la inexistencia de un modelo para la planta. El segundo es que la mezcla de combustible para la planta cambia cada cinco minutos, por lo que se modifica la relación de poder calorífico, que es la variable de control. Este sistema de control emplea lógica difusa, cuyas reglas se obtienen en un módulo de aprendizaje empleando árboles de identificación. Un esquema de este sistema se muestra en la Figura 5.16.

- Una propuesta diferente es la de Seilonen et al. [SAV+02, Pir02, SAHK02]. Estos autores proponen complementar un sistema de automatización de procesos existente con la tecnología de agentes. Es decir, se complementa, no se

[9] ARchitecture for Cooperative Heterogeneous ON-line systems, http://www.fe.up.pt/~eol/PROJECTS/archon.html

[10] Advanced Process Analysis and Control System Project

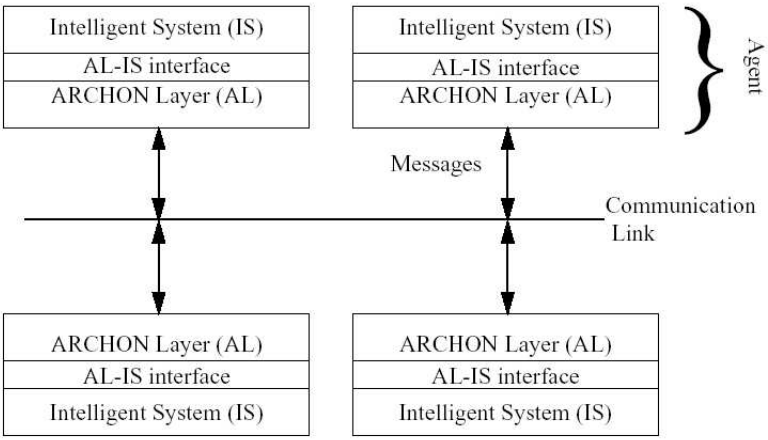

Figura 5.15.: Arquitectura ARCHON [CJ96].

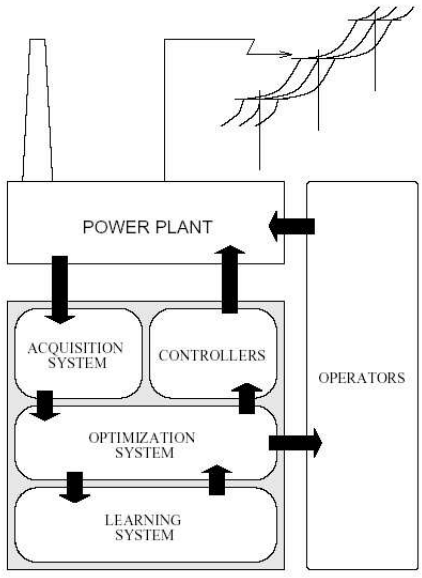

Figura 5.16.: Esquema del MAS propuesta por Velasco et al. [VGMI46].

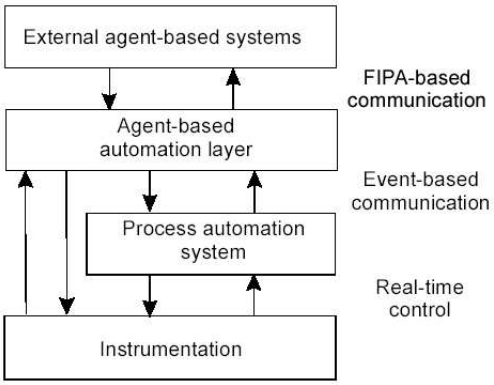

Figura 5.17.: Esquema en capas de la propuesta de Seilonen et al. [SAV+02, Pir02, SAHK02].

reemplaza. El sistema de agentes forma una capa adicional sobre el sistema de automatización cuya misión es supervisarlo y reconfigurarlo cuando sea necesario. La estructura en capas de esta propuesta se muestra en la Figura 5.17.

- Por último, citaremos el trabajo de V. Gyurjyan et al. [GAH+03]. Estos autores proponen un sistema de control, todavía en desarrollo[11], con la capacidad de combinar procesos o sistemas de control heterogéneos en un entorno homogéneo. Este sistema, también basado en la arquitectura FIPA, está desarrollado empleando al agente como nivel de abstracción. La Figura 5.18 muestra un esquema de esta arquitectura.

  El agente mediador a alto nivel *Hepatic* es el responsable de la creación, recuperación y eliminación de los agentes. El agente *Thalamus* se asegura de que las soluciones locales a problemas de control se integran al marco global. El agente *Cortex* es la interfaz entre un diseñador de un sistema de control específico y el sistema de control global. Habilita la integración de los controles propuestos por un software no basado en agentes dentro de la comunidad de control multiagente. El *Cortex* auto-genera un agente-envoltura (*wrapper*), así como las clases de ontología necesarias mediante el procesamiento de la descripción del sistema específico de control escrito en un lenguaje basado en RDFS llamado COOL (*Control Oriented Ontology Language*). Los agentes de la plataforma de control pueden transmitir los mensajes a este agente-envoltura, y a través de estos mensajes gestionar el hardware de control

5.4. Implementación de un MAS con KQML

Como paso inicial, en la búsqueda de una arquitectura de MAS aplicada a labores de control de procesos, se ha desarrollado un prototipo descrito en esta sección. Haremos notar su importancia dentro del trabajo desarrollado, puesto que por un lado supuso la toma de contacto con el campo de los MAS, y a posteriori permitió comprobar que los

[11] http://www.jlab.org/~gurjyan/currentprojects.htm

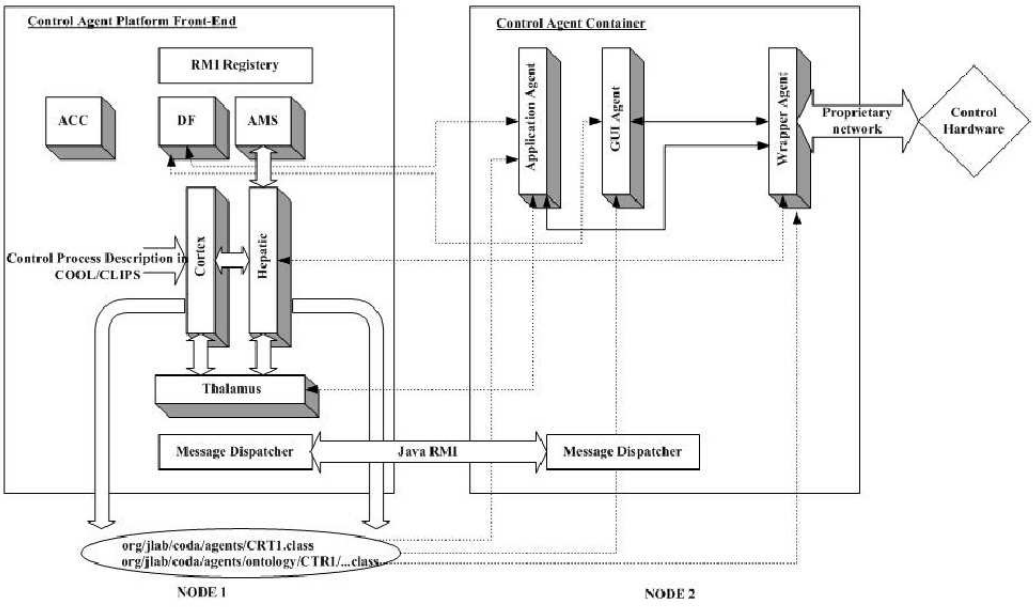

Figura 5.18.: Plataforma de agentes del sistema de control propuesto por Gyurjyan et al. [GAH+03].

conceptos introducidos por el autor sin el conocimiento del estándar de arquitectura FIPA se encuentran incluidos de una u otra forma en dicho estándar. El propósito inicial consistió en controlar en simulación la altura de una plataforma integrada en un robot móvil.

Esta arquitectura se basa en KQML para el intercambio de mensajes. Como este estándar se refiere únicamente al lenguaje empleado por los agentes, existe arbitrariedad a la hora de construir y organizar a los agentes del sistema.

Esta arquitectura agrupa a los agentes en *sistemas*. Un *sistema* se entiende como el conjunto de agentes que pueden invocar sus métodos respectivos de una forma directa, sin el empleo de protocolos remotos. Este criterio de agrupaciones recuerdan a las agencias del estándar OMG MASIF. La arquitectura interna propuesta para un sistema se muestra en la Figura 5.19.

Los agentes del mismo *sistema* pueden comunicarse mediante invocaciones de sus respectivos métodos, los cuales tienen nombres relativos a las performativas KQML. Así un agente dispone de los métodos *procesaTell* o *procesaAchieve*, encargados de procesar respectivamente los mensajes con performativa *tell* o *achieve*. Los argumentos de estos métodos se refieren a los diversos campos del mensaje. De este modo se acelaran las comunicaciones, ya que se evitan los pasos correspondientes a dar formato KQML al mensaje y extraer su información contenida.

Para enviar un mensaje a un agente en otro *sistema* los agentes envían en primera instancia estos mensajes a un tipo especial de agente denominado *Ruteador*. Este agente lo reenvía a un objeto *Formateador* encargado de convertir el mensaje a una cadena de caracteres con formato KQML, pasándolo a su vez a un objeto llamado *Comunicador*. Es el *Comunicador* quien, siguiendo el protocolo programado (por

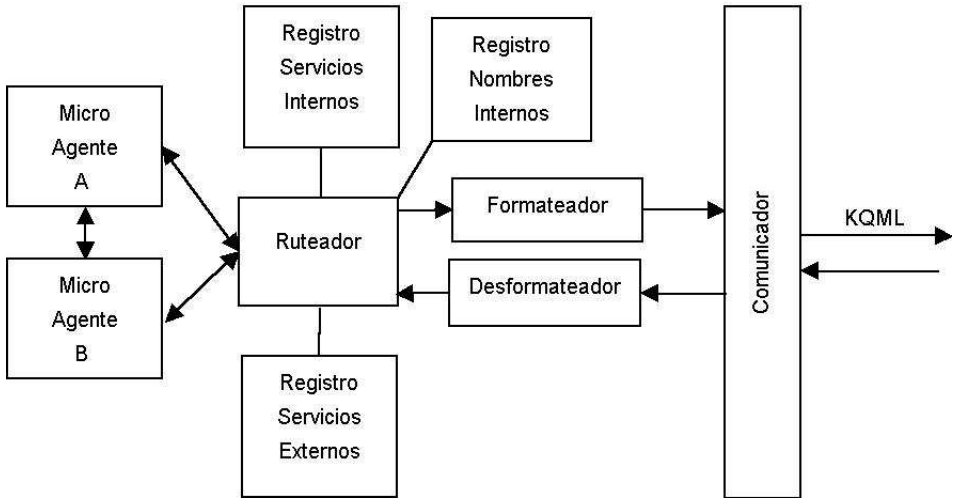

Figura 5.19.: Arquitectura del MAS basado en KQML.

ejemplo *sockets*) transmite el mensaje al otro *sistema*.

La apariencia externa de un *sistema* es la de un único agente (todos los agentes se encuentran en la misma dirección) que se comunica a través de KQML. La recepción del mensaje se realiza también mediante el *Comunicador*. Una vez completada la recepción traspasa el mensaje (en formato de cadenas de caracteres KQML) a un objeto denominado *Desformateador*. Este objeto convierte la cadena de caracteres a un objeto *Mensaje*, cuyos atributos se refieren a los campos de un mensaje KQML: *performativa*, *sender*, *receiver*... Este objeto es traspasado al *Ruteador*, encargado de localizar al destinatario e invocar el correspondiente método.

Un sistema se completa con sendos registros de servicios internos y externos. Cuando un agente necesita de un servicio, pide al *Ruteador* que localice en estos registros a un agente capaz de proporcionar ese servicio. Esta parte del *sistema* se comporta por tanto como un servicio de *páginas amarillas*. La búsqueda se realiza primero de forma interna en el *sistema*. De este modo, ante dos agentes (uno interno y otro externo) que ofrezcan el mismo servicio, el *Ruteador* elegirá en primer lugar el interno. El último servicio contemplado consiste en un registro de agentes externos (al modo de unas *páginas blancas*) que es actualizado por el *Ruteador* cada vez que aparece en los mensajes procesados una referencia a un agente situado en otro *sistema*. De este modo, no hace falta que el agente de otro sistema se registre, agilizando el flujo de mensajes.

Para facilitar el control y la depuración de los agentes del sistema se ha dotado a cada uno de los agentes de una interfaz gráfica donde se muestran sus correspondientes mensajes transmitidos y recibidos. Esta interfaz se comporta como un objeto independiente del agente y por lo tanto no interviene en el flujo de mensajes. En esta línea se ha codificado otra interfaz gráfica donde el usuario puede interactuar con el sistema, componiendo y mandando sus propios mensajes KQML.

Se ha implementado con éxito un MAS con tres *sistemas* donde un agente (el cual contiene un mecanismo de decisión que determina el movimiento de la plataforma del robot móvil) es capaz de encontrar a un agente en otro *sistema* (el encargado de subir

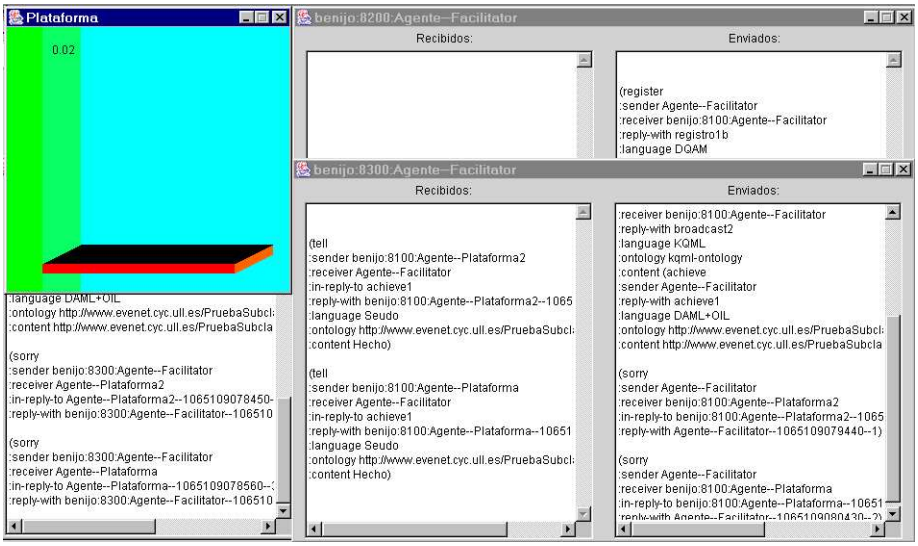

Figura 5.20.: Sistema Multiagente basado en KQML

o bajar la plataforma) consultando al *Ruteador* ubicado en un tercer *sistema* (el cual centraliza todos los registros de servicios).

Finalmente se ha incluido en el MAS una pequeña ontología DAML+OIL implícita en la codificación de los agentes. Esta ontología se reduce a dos conceptos opuestos *SubirPlataforma* y *BajarPlataforma*, así como la definición y relaciones de unidades de longitud (*metro* y *centímetro*).

Una visión del sistema completo en ejecución puede contemplarse en la Figura 5.20. En dicha figura se aprecian las interfaces gráficas correspondientes a tres agentes ubicados en tres *sistemas* diferentes, y la interfaz gráfica que simula la acción sobre la plataforma.

Esta aplicación adolece de la falta de estandarización propia de los sistemas KQML, siendo superada por el MAS presentado en la sección 5.5.

A pesar de esta falta de estandarización se puede comprobar el paralelismo de algunos aspectos de la arquitectura con el estándar FIPA. Así, el conjunto de Formateador, Desformateador y Comunicador puede identificarse con el MTS, los diversos registros de servicios implementan las acciones de un DF mientras que los registros de nombres implementan las del AMS. Esta circunstancia nos ha hecho alejarnos de este prototipo puesto que los aspectos diseñados ya están incluidos en su mayor parte en la arquitectura FIPA.

5.5. Introducción al Sistema MASCONTROL para el control de procesos

Figura 5.21.: Logotipo MASCONTROL.

Tras explorar la utilización del estándar KQML para la implementación de un MAS destinado al campo del control, se optó por cambiar al estándar FIPA, principalmente que representa una estandarización más elaborada.

El primer paso para realizar la aplicación consiste en determinar la función a cumplir por el MAS diseñado. En este sentido, se optó por centrarse en el control de procesos en diferentes plantas, siempre con el objetivo de demostrar la versatilidad de los MAS.

De una forma concreta, se pretende seguir el planteamiento de un STR. Consideramos que este tipo de problema se puede tratar de forma adecuada con MAS. Una primera razón es la coexistencia de módulos conceptualmente diferentes, como la lectura/actuación sobre la planta, identificación del sistema y estimación de los mejores parámetros del controlador. Otro motivo que sugiere la utilización de agentes consiste en lo adecuado de la actuación en paralelo de los módulos, optimizando de esta forma la capacidad de cálculo.

5.6. Elección de herramientas

A grandes rasgos, las herramientas a emplear en la implementación de MASCONTROL son las mismas que las utilizadas en MASplan. La decisión de emplear de nuevo un estándar FIPA nos remite al empleo de la herramienta FIPA-OS.

También buscamos con esta aplicación demostrar que se puede aprovechar el empleo de un agente de ontologías en un campo tan alejado, en apariencia, como es el control de procesos. Por las mismas razones vistas en la justificación de las herramientas para MASplan, se ha elegido el lenguaje de marcas DAML+OIL como lenguaje de implementación de la ontología. Esto nos permite emplear las mismas herramientas que las utilizadas en la implementación del sistema multiagente destinado a la planificación de un grupo universitario:

- OilEd como editor de ontologías.

- FaCT, DAML+OIL Ontology Checker y DAMLValidator como razonadores que verifiquen la estructura de la ontología diseñada.
- Jena como *parser* DAML+OIL.
- JDK 1.4.2 como modo de compilación e interpretación Java.

Aparte de estas herramientas, se utilizan en el sistema las librerías de la herramienta Evenet2000 (detallada en el apéndice J), desarrollada en el Grupo de Computadoras y Control de la Universidad de La Laguna por parte del autor de este trabajo. Esta librería fue diseñada inicialmente (y de forma independiente del desarrollo del MAS) para el entrenamiento de redes neuronales de arquitectura arbitraria. No obstante, esta herramienta se ha demostrado como muy versátil y útil en problemas de optimización en general, y relativos al control en particular, tal como se muestra en la sección 5.2. La inclusión de esta herramienta en el MAS se justifica por tres razones principales:

- Su capacidad de optimización de funciones dependientes de unos parámetros es útil en la estimación de los coeficientes en la identificación del sistema a controlar.
- Al haber sido diseñada de forma independiente al estudio de los MAS, permite demostrar la capacidad y facilidad de inclusión en los MAS de herramientas ya existentes.
- Al ser una herramienta de código libre, permite realizar modificaciones para adaptarla a las necesidades del MAS. Debemos aclarar que no fue necesario realizar ningún tipo de modificación durante el desarrollo del MAS, lo que reafirma el punto anterior.

En este aspecto de inclusión de herramientas diseñadas dentro del Grupo CyC, una de las líneas abiertas consiste en la inclusión de los módulos de la librería de simulación y análisis de sistemas lineales de control, ControlWeb[12] [PRS+03].

5.7. Diseño de Ontología

Siguiendo los mismos razonamientos que los contemplados en la sección referida al diseño de la ontología de MASplan, en este caso nos hemos decantado por una ontología de aplicación de grano grueso. Ahora queda mucho más justificada la sencillez de la ontología, puesto que el papel asumido por el OA es mucho menor en MASCONTROL, tal como se comprobará en la sección 5.8. Muchos de los conceptos y relaciones definidos no son empleados por el MAS. No obstante se han incluido en la ontología pensando en futuras ampliaciones del sistema.

Las figuras 5.22, 5.23 y 5.24 muestran respectivamente la jerarquía de clases, las propiedades y las instancias individuales definidas en la ontología.

En cuanto a las clases definidas, éstas hacen referencia, principalmente, a conceptos propios del control. En esta línea tenemos, por ejemplo, las clases *Planta*, *Sistema* (con las subclases *SistemaLineal* y *SistemaNoLineal*), *Comando*, *Error*, *Salida*, *FuncionTransferencia*, *NumeradorTF*, *DenominadorTF*[13], *Polo*, *Cero*,

[12] http://controlweb.cyc.ull.es

[13] relativos al numerador y denominador de la función de transferencia

AccionControl...

Otro conjunto de clases hacen referencia a los métodos de optimización de parámetros. Así tenemos *MetodoOptimizacion* (con sus subclases *MetodoOptimizacionLineal* y *MetodoOptimizacionNoLineal*), *Entrenamiento* (referido a los entrenamientos de optimización de los parámetros), *ConjuntoParametros* (el conjunto de parámetros a optimizar), *AlgoritmoOptimizacionPaso*...

Especial atención merece el concepto de *Red*. Este concepto se refiere al modelo a optimizar. Recuérdese que los módulos de optimización de la herramienta Evenet2000 tratan al modelo como una red y a los parámetros a optimizar como pesos en un entrenamiento.

Como ejemplo de definición de clases, el siguiente código indica que la clase *SistemaNoLineal* es una subclase de *Sistema*.

```
  <daml:Class rdf:about="#SistemaNoLineal">
    <rdfs:label>SistemaNoLineal</rdfs:label>
    <rdfs:comment>Sistema con ecuacion diferencial o en diferencias
no lineal</rdfs:comment>
    <rdfs:subClassOf>
      <daml:Class rdf:about="#Sistema"/>
    </rdfs:subClassOf>
  </daml:Class>
```

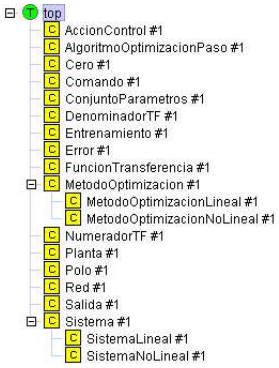

Figura 5.22.: Jerarquía de clases en la ontología de MASCONTROL.

Dentro de las propiedades definidas, el grueso de ellas se refieren a los valores de los conceptos definidos como clases: *valorPolo*, *valorCero*, *valorNumeradorTF*, *valorDenominadorTF*, *valorMetodoEntrenamiento*... El siguiente código, por ejemplo, define que los objetos de la clase *Polo* tienen exactamente un valor de tipo NumeroComplejo[14] para la propiedad *ValorPolo*.

```
  <daml:Class rdf:about="#Polo">
    <rdfs:label>Polo</rdfs:label>
```

[14]Definida la clase dentro de esta misma ontología

```
    <rdfs:subClassOf>
      <daml:Restriction daml:cardinalityQ="1">
        <daml:onProperty rdf:resource="#valorPolo"/>
        <daml:hasClassQ rdf:resource="#NumeroComplejo"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

Otra serie de propiedades se refieren a términos de la teoría de control como *orden* y *tipo* de un sistema. También se definen una serie de propiedades que facilitan la acción de MASCONTROL. En esta línea se han definido las siguientes propiedades

- *consignaAlcanzadaConControladorP*. Es una propiedad de dominio *Sistema* y rango *boolean*, que indica si la salida del sistema ha alcanzado la consigna mediante un controlador P.
- *accionControlAdecuada*. Esta propiedad tiene como dominio un *Sistema* y como rango una *AccionControl*. Indica la acción de control adecuada para que un sistema alcance la consigna deseada.
- *numeroVecesMejor*. El dominio de esta propiedad es *MetodoEntrenamiento* y de rango un *entero*. Indica el número de veces que el método de entrenamiento sujeto se ha mostrado como el mejor en una optimización. Los detalles concretos se comprenderán mejor tras la lectura de las secciones 5.8 y 5.11.

Estas definiciones permiten al MAS realizar algunas inferencias interesantes a partir de los axiomas de la ontología. Por ejemplo, si un sistema no alcanza la consigna deseada con una acción de control P pura se puede concluir que se trata de un sistema de tipo cero. La acción de control adecuada para un sistema de tipo cero es la PI. Por tanto, un agente MASCONTROL puede deducir que la acción de control más adecuada para un sistema que no alcance la consigna con un controlador P es la PI, sin que esta última definición aparezca explícitamente. El código correspondiente es el siguiente:

```
<daml:DatatypeProperty rdf:about="#tipo">
  <rdfs:domain>
    <daml:Class rdf:about="#Sistema"/>
  </rdfs:domain>
  <rdfs:range>
    <xsd:integer/>
  <rdfs:range>
</daml:DatatypeProperty>
<rdf:Description rdf:about="#SistemaTipo0">
  <rdf:type>
    <daml:Class rdf:about="#Sistema"/>
  </rdf:type>
  <ns0:tipo><xsd:integer xsd:value="0"/></ns0:tipo>
  <ns0:vieneControladoPor rdf:resource="#AccionPI"/>
```

```
    <ns0:consignaAlcanzadaConControladorP>
      <xsd:boolean xsd:value="false"/>
    </ns0:consignaAlcanzadaConControladorP>
  </rdf:Description>
  <rdf:Description rdf:about="#SistemaTipo1oMayor">
    <rdf:type rdf:resource="#Sistema"/>
    <ns0:vieneControladoPor rdf:resource="#AccionP"/>
    <ns0:consignaAlcanzadaConControladorP>
      <xsd:boolean xsd:value="true"/>
    </ns0:consignaAlcanzadaConControladorP>
  </rdf:Description>
  <daml:Class rdf:about="#Sistema">
    <daml:disjointUnionOf rdf:parseType="daml:collection">
      <daml:Class rdf:about="#SistemaTipo1oMayor">
      <daml:Class rdf:about="#SistemaTipo0">
    </daml:disjointUnionOf>
  </daml:Class>
```

A partir de estas definiciones, un *Sistema* puede ser un *SistemaTipo0* o un *SistemaTipo1oMayor* (no a la vez). Por tanto, un sistema cuya salida no alcanza la consigna cuando está controlado mediante una acción proporcional es unívocamente de tipo 0 (no existe otro sistema posible con esa característica). Por último, la acción de control adecuada es *AccionPI*.

En este punto, aclararemos que existe una concordancia entre los nombres de la clase de las acciones de control (*AccionP*, *AccionPI*) y las clases Java. De este modo, se facilita el cambio de la acción de control sobre la planta. En este sentido, una de las líneas de investigación abiertas consiste en el estudio sobre la posibilidad de incluir los *bytecodes* de las clases Java equivalentes directamente dentro de la ontología, lo cual supondría una cohesión todavía mayor de ésta con el sistema multiagente.

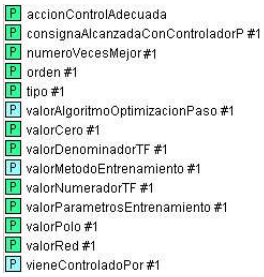

Figura 5.23.: Propiedades en la ontología de MASCONTROL.

Finalmente se declaran en la ontología una serie de instancias individuales. Entre estas declaraciones se encuentran las de los métodos de optimización disponibles en la herramienta Evenet2000: *GradienteDescendente*, *GradienteConjugado*, *AlgoritmoGenetico*, *EnfriamientoProgresivo* y *CaminoAleatorio*. Todos estos términos son instancias de la clase *MetodoOptimizacion*.

```
<rdf:Description rdf:about="#GradienteConjugado">
  <rdf:type>
    <daml:Class rdf:about="#MetodoOptimizacion"/>
  </rdf:type>
</rdf:Description>
```

De modo análogo, se definen las instancias disponibles de la clase *AlgoritmoOptimizacionPaso*, esto es *PasoFijo*, *AjusteParabolico* y *SeccionOro*.

Asimismo se declaran las instancias de las acciones de control implementadas (de la clase *AccionControl*) a saber, *AccionP*, *AccionPI* y *AccionRealimentacionEstado*. Las instancias de la ontología se complementan con unas instancias referidas a los polos en el origen (*PoloOrigen*, instancia de la clase *Polo*), y al tipo del sistema, distinguiendo entre los sistemas de tipo 0 (*SistemaTipo0*) y el resto de sistemas (*SistemaTipo1oMayor*).

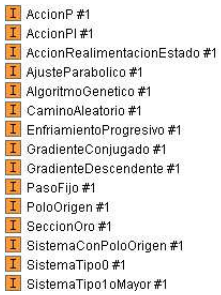

Figura 5.24.: Instancias individuales declaradas en la ontología de MASCONTROL.

Conviene aclarar, tras definir la ontología a emplear, que la filosofía de ésta es totalmente distinta a la de COOL, puesto que este lenguaje únicamente describe los sistemas de control (interfaces, software, hardware). En nuestro caso se trata de una ontología empleable por los agentes del MAS para el funcionamiento del mismo.

5.8. Arquitectura del sistema

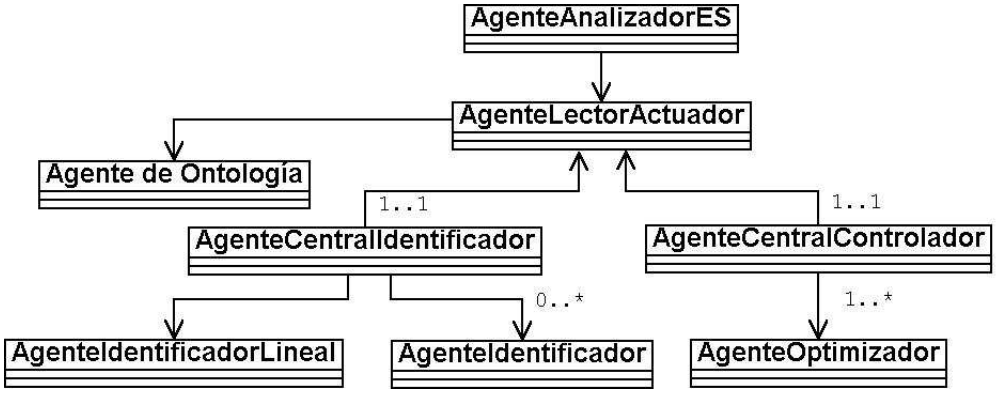

Figura 5.25.: Arquitectura de agentes MASCONTROL.

La Figura 5.25 muestra el esquema básico de agentes de MASCONTROL. Poniendo en práctica el concepto de MAS, se ha dividido entre los agentes las diversas funcionalidades del sistema. La arquitectura MASCONTROL se compone de los siguientes tipos de agentes.

AgenteLectorActuador (*LAA*) Implementa el controlador indicado. Para ello muestrea la salida de la planta con un periodo fijado en su correspondiente fichero de perfiles XML. Dicho de otro modo, actúa como interfaz del MAS respecto a la planta.

Almacena en su interior un vector con los pares de entrada-salida de la planta, lo que permitirá la identificación del sistema y determinar si la entrada es lo suficientemente rica para dicha identificación. También se encarga de mostrar las gráficas de evolución de la entrada y la salida.

Este tipo de agente accede a un objeto de una clase que hereda una clase abstracta Java denominada *AccionControl*, encargado de calcular el comando que hay que suministrar a la planta en la siguiente etapa. Hasta el momento, las subclases implementadas se refieren a acciones de control proporcional, proporcional-integral y realimentación por variables de estado. Esto muestra la versatilidad de la herramienta, al implementar estructuras de control tan diversas. En cualquier momento, el LAA es capaz de modificar esta acción de control (cambiando el objeto que realiza el cálculo) basándose, por ejemplo, en estudios sobre el tipo del sistema.

AgenteIdentificador (*IA*) Cada uno de estos agentes se encarga de llevar a cabo la identificación del sistema a partir del conjunto de pares entrada-salida provenientes del LAA. Para ello hace uso de las librerías de optimización de la herramienta Evenet2000. El usuario, mediante el fichero de perfiles, o el CaIA, a partir del historial de MASCONTROL, escoge un método de optimización de los parámetros del sistema. En ambos casos, se han de suministrar los

criterios de parada dependiendo del método escogido). El modelo del sistema es suministrado asímismo mediante el fichero de perfiles. Este modelo se encuentra en forma de red en un fichero generado en el editor gráfico del Evenet2000. Es decir, para un IA, el problema es equivalente al de optimización de los pesos de una red, cuyo patrón lo forman los pares de entrada/salida del sistema. Hemos incluido en el código del IA un predictor a un paso para la determinación del coste asignado, según la fórmula 5.42.

AgenteIdentificadorLineal (*ILA*) Idéntico al anterior, pero supone un modelo que permita la expresión en forma de regresión lineal (ecuación 5.37). En las pruebas realizadas, se ha supuesto un modelo ARX empleando un predictor de un paso adelante. Al ser de esta forma, el modelo queda determinado por el grado del numerador y denominador de la función de transferencia del sistema. En vez de emplear los módulos de optimización de Evenet2000, se emplea una instancia de una clase Java *IdentificacionRecursiva* que implementa el método descrito en la sección 5.1.5.1, relativo al factor de olvido. A fecha de escritura de este trabajo, este factor de olvido (por defecto con un valor 1) se indica, como el resto de factores, a través del correspondiente fichero de perfiles.

AgenteCentralIdentificador (*CIA*) Sirve de gestor de los diversos agentes identificadores (ya sean lineales o no). Inicialmente consulta al DF sobre aquellos agentes registrados que prestan el servicio de identificación de un sistema. De forma periódica pide a estos agentes los resultados de la optimización que están llevando a cabo: valor de error, parámetros calculados y fichero donde se encuentra el modelo empleado. Con estos datos selecciona aquella optimización con unos mejores resultados y comunica al resto de agentes con el mismo modelo el conjunto de parámetros de mínima función de coste obtenido para dicho modelo. De este modo, los agentes de identificación partirán de este valor en la próxima optimización. En en cualquier momento, este CIA puede solicitar a cada IA o ILA que cambie su modelo de sistema o cualquiera de los parámetros de la optimización. Para introducir un comportamiento más inteligente en el sistema, el CIA lleva una contabilidad de las veces que cada método de optimización ofrece el mejor resultado en la consulta periódica a los IAs. Esta información se almacena en un historial con el formato de DAML+OIL. De esta forma, aparte del uso de este fichero por parte del AgenteCargadorOptimización, en futuras versiones del MAS se puede añadir una mayor capacidad de inferencia a los agentes.

En caso de pérdida de comunicación con alguno de los agentes identificadores, el sistema no se ve afectado, puesto que el CIA fija un tiempo máximo de espera de respuesta de los IAs y ILAs (para ello se emplea el método `timeOut` implementado por la herramienta FIPA-OS), de modo que si se ha superado ese plazo, el agente realiza los cálculos a partir de los datos recibidos. De esta forma se consigue que el sistema sea, en este sentido, tolerante a fallos.

AgenteOptimizador (*OpA*) Este agente realiza una optimización de los parámetros del controlador seleccionado. Para ello toma los valores procedentes de la identificación del sistema y los incluye como constantes en un sistema tratado como una red neuronal por los módulos de Evenet2000. Este modelo refleja el

esquema de control deseado de tal modo que los parámetros del controlador son los pesos de la red. En las pruebas realizadas, el esquema de optimización de los OpA con la familia de controladores PID es similar al visto en la sección 5.2.5. Recordar en este punto que con este tipo de imposición se pretende que el sistema alcance la consigna lo más rápidamente posible pero penalizando los grandes sobrepasamientos. Para buscar un mejor comportamiento del sistema, los patrones de entrenamiento consisten en una serie por tramos de pares del tipo *consigna*, *consigna* con valores significativamente distintos para la consigna[15]. El modelo de controlador puede, a partir de la información proveniente del resto del MAS, ser cambiado de manera fácil en cualquier momento gracias a la modularidad de los elementos de Evenet2000.

AgenteCentralControlador (*CCoA*) Su cometido es similar al de CIA, esto es de gestión y supervisión, pero en este caso relativa a la optimización de los parámetros del controlador. Su labor se resume en los siguientes puntos:

1. De forma periódica solicita al CIA los detalles de la marcha del proceso de identificación: fichero donde se encuentra el modelo con mejores resultados en la identificación del sistema, pesos del sistema.
2. Tras analizar estos datos pide a cada OpA los valores de los parámetros del controlador que minimizan la función de coste, así como el valor de dicha función.
3. Entre los valores recibidos, determina el conjunto de parámetros de mínimo coste.
4. El CCoA almacena estos valores para su posterior envío al LAA.
5. Finalmente indica a todos los OpAs que detengan el entrenamiento actual y comiencen uno nuevo con los valores óptimos entre los de los entrenamientos anteriores con el modelo de planta pasado por el CIA. Para ello, los OpAs deben crear un fichero de texto que represente el esquema de control deseado y con el modelo de planta indicada por el CIA.

Del mismo modo que el CIA, este agente lleva una contabilidad de cuántas veces ofrece un mejor resultado cada OpA. Estos datos se almacenan en un fichero DAML+OIL, empleando términos definidos en la ontología diseñada. En cualquier momento, el CCoA puede solicitar a un OpA que modifique el modelo de su controlador.

AgenteAnalizadorES (*AESA*) Este agente opcional analiza los datos de entrada (comando enviado a la planta) y salida (lectura de la planta) proporcionado por el LAA. Este análisis se realiza en dos frentes. En primer lugar testea de forma aproximada si la entrada se ha estancado. Para ello determina el máximo y el mínimo en su valor en los últimos N pares (este valor es suministrado en un fichero de perfiles), y comprueba si esa cantidad es menor que un umbral (también fijado en el fichero de perfiles). En caso afirmativo, supone que la entrada no es lo suficientemente rica y sugiere al LAA que modifique el valor

[15] a semejanza del caso expuesto en el apartado 5.2.5.

de la consigna, con lo que mejorará la identificación. Un proceso similar se realiza para determinar la estabilización de la salida. Esta información puede ser empleada por el MAS (mediante el OA) para estudiar el tipo del sistema (en caso de entrada escalón) o para sugerir un cambio de consigna para enriquecer la identificación del sistema.

La opción de cambio de consigna puede deshabilitarse *on-line* a través de la ventana mostrada en la Figura 5.26.

Figura 5.26.: Visualizador de parámetros MASCONTROL.

Agente de Ontología (*OA*) Al igual que en el caso de MASplan, se ha incluido en el sistema un agente de ontología, volviendo a ser este un punto novedoso en esta parte del trabajo y que lo vuelve a diferenciar de sus antecedentes. En este caso el OA no interviene directamente en el control de la planta. La intervención de este agente se reduce a fecha de escritura de este trabajo a facilitar la información al LAA referente al tipo del sistema según su error en régimen permanente. Teniendo en cuenta la serie de definiciones incluidas en la ontología, son muchas las posibilidades de uso del OA en futuras implementaciones del MAS. En este sentido, por ejemplo, el CCoA puede consultar en un momento dado, las instancias de los métodos de optimización disponibles en el sistema. Con esa información puede crear un nuevo OpA que comience un entrenamiento empleando algunos de estos métodos.

Además de estos agentes, se han diseñado una serie de agentes cuyo periodo de vida en el sistema es corto y se centran en la inicialización de MASCONTROL de un modo más sencillo.

AgenteCargadorOptimización (*COA*) Este agente es el encargado de leer el perfil indicado por el usuario en el fichero correspondiente e inicializa el CCoA y los OpAs (con sus parámetros de aprendizaje) en la misma JVM donde el COA es inicializado. Opcionalmente, y si así se indica por el usuario, localiza en el fichero de historial, en caso de existir, aquél método de optimización, incluidos el modelo y los parámetros del entrenamiento, que ha conducido a un mejor resultado para la planta concreta. La planta se indica mediante un identificador en el fichero de perfiles. Si no encuentra registrado un OpA con esos parámetros óptimos del historial, crea uno extra con esos criterios para el entrenamiento. De este modo, y como ese método concreto ha conducido a unos buenos resultados con anterioridad, es de esperar que el comportamiento del sistema mejore con la inclusión de este OpA extra.

AgenteCargadorIdentificador (*CaIA*) Realiza las operaciones equivalentes al COA pero respecto al CIA y los IAs. Por tanto, este agente es el encargado de leer el fichero de perfiles correspondiente, inicializando en la misma JVM estos tipos de agente. Del mismo modo, si así lo indica el usuario, localiza en el historial aquél método de identificación que ha conducido a un mejor resultado con anterioridad con la planta empleada. Si encuentra que no se ha creado un IA con esos parámetros crea uno de estos agentes extra con esos criterios de identificación.

Debemos aclarar de nuevo que la existencia de estos dos tipos de agentes se justifica únicamente para simplicar la inicialización de MASCONTROL. Sin embargo, la existencia de un COA obliga a que la optimización de los parámetros del controlador se lleven a cabo en la misma JVM. Del mismo modo, el CaIA implica que la identificación se realice en una única JVM (aunque puede ser distinta a la correspondiente a la optimización). Esto puede ser modificado por el usuario creando otros IAs y/o OpAs en diferentes máquinas, aprovechando aún más la ventaja de los sistemas distribuidos. Otro modo de llevarlo a cabo es extendiendo la arquitectura incluyendo la posibilidad de varios CIAs (o CCoAs) con un tipo de agente superior que los gestionase solicitándole los datos optimizados periódicamente y trasladándolos al LAA. La implementación y un estudio en más profundidad de estas alternativas constituye una de las líneas abiertas del presente trabajo.

La arquitectura propuesta permite optimizar la implementación de un sistema STR optimizado, ya que permite distribuir los cálculos en paralelo, en varias máquinas potentes, alejadas incluso de la planta objeto del control.

5.9. Flujo de mensajes

Analizaremos en este apartado el flujo de mensajes de los diferentes procesos involucrados en MASCONTROL. Estos procesos son principalmente tres: identificación del sistema, optimización de los parámetros del controlador y estimación del estancamiento de la entrada y la salida de la planta. Por mótivos de sencillez, los diagramas mostrados en esta sección no reflejan las interacciones con el DF. En este sentido, por ejemplo, cada cierto número de etapas durante el control tanto el CCoA como el CIA solicitan al DF la lista de OpAs y IAs/ILAs activos respectivamente. De este modo, el sistema multiagente permite la inclusión on-line de nuevos agentes.

Esta estructura de flujo de mensajes puede verse modificada si se considera que los agentes pueden moverse a través de la red. En ese caso sería necesaria la inclusión del DF.

5.9.1. Flujo de mensajes en el proceso de identificación del sistema

Los tipos de agente participantes en el proceso de identificación del sistema son tres: el AgenteLectorActuador (encargado de recoger los datos del comportamiento de la planta), el AgenteCentralIdentificador (encargado de suministrar los datos y gestionar las optimizaciones de los parámetros) y finalmente los diversos agentes que implementan el servicio de tipo AgenteIdentificador[16], encargados en sí de la

[16] y/o AgenteIdentificadorLineal

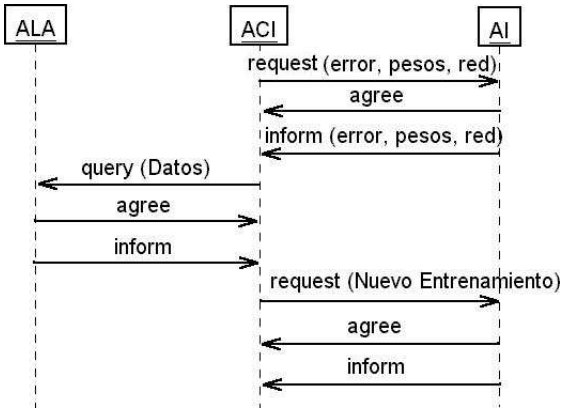

Figura 5.27.: Diagrama simplificado UML de flujo de mensajes en el proceso de identificación del sistema.

optimización de los parámetros.

El flujo de mensajes consiste en tres conversaciones establecidas entre estos agentes. El primero de ellos sigue el protocolo *FIPA Request* iniciado de manera periódica por el CIA (tiempo especificado por el usuario en el fichero de perfiles), solicitando a los IAs registrados que detengan los entrenamientos actuales. Estos IAs responden con mensajes, informando sobre el modelo, el conjunto de parámetros y el coste del entrenamiento. El modelo de red es necesario puesto que en implementaciones futuras del MAS los diferentes IAs podrían modificar de modo autónomo el modelo empleado. De forma simultánea, mientras recibe las respuestas de los IAs, solicita (protocolo *FIPA Query*) al LAA los nuevos pares de entrada-salida registrados del comportamiento de la planta.

El CIA une estos pares a los recibidos anteriormente y solicita a cada IA que comience un nuevo entrenamiento de optimización de los parámetros del sistema utilizando los nuevos datos. Ese nuevo entrenamiento tiene como conjunto de parámetros iniciales los de mínimo coste calculados en la última tanda para el modelo tratado por el IA, que se suponen un buen punto de partida para continuar la identificación por parte de un IA que utilice el mismo modelo. Esta petición se lleva a cabo mediante una conversación según el protocolo de comunicación *FIPA Request*.

El diagrama de tiempo UML simplificado relativo a la identificación del sistema se muestra en la Figura 5.27.

5.9.2. Flujo de mensajes en el proceso de optimización de los parámetros del controlador

Tal como se deduce de la descripción de la arquitectura MAS realizada en la sección 5.8, los agentes involucrados son el AgenteLectorActuador, el AgenteCentralIdentificador, el AgenteCentralControlador y los diversos agentes del servicio AgenteOptimizador.

El proceso es iniciado por el CCoA que solicita de forma periódica al CIA información sobre la marcha de la identificación (el muestreo se indica, como es de esperar, en el fichero de perfiles correspondiente). Para ello establece una conversación

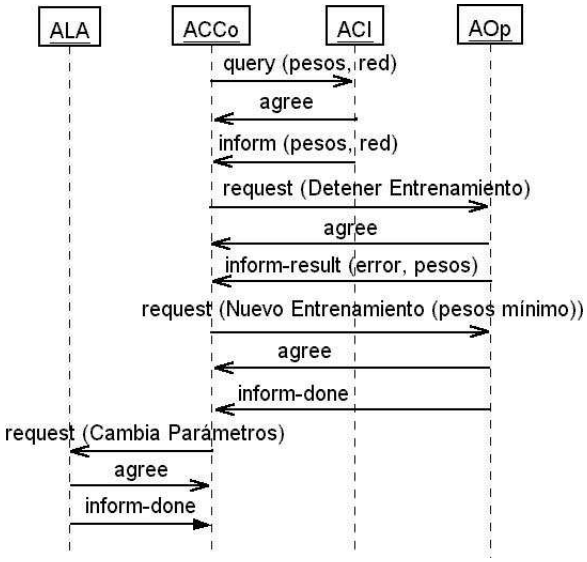

Figura 5.28.: Diagrama simplificado UML de flujo de mensajes en el proceso de optimización de los parámetros del controlador.

según el protocolo *FIPA Query*, solicitando el conjunto de pesos y el contenido del fichero de red (siempre desde el punto de vista de la herramienta *Evenet2000*) que ofrecen una mejor identificación de la planta. De forma inicial, este envío se realizaba en el contenido del mensaje en forma de cadena de caracteres. Sin embargo, a fecha de escritura de este trabajo, se ha optimizado empleando la ventaja de la seriación implementada por Java.

Al recibir los resultados, solicita a los OpAs (protocolo *FIPA Request*) que detengan los entrenamientos que se están llevando a cabo e informen de conjunto de parámetros de mínimo coste encontrado.

A continuación, el CCoA lleva a cabo dos acciones. Por una parte, pide (mediante un nuevo protocolo *FIPA Request*) a los OpAs que comiencen un nuevo entrenamiento cuyo conjunto de pesos inicial es el de mínimo coste de entre los entrenamientos realizados en la última tanda, y cuya planta es la del modelo comunicado por el CIA. Por el otro lado, el CCoA solicita al LAA (de nuevo un protocolo escogido es el *FIPA Request*) que modifique los parámetros de la acción de control sobre la planta. En caso de que el usuario tenga bloqueado el cambio de parámetros en la acción de control (recordar la Figura 5.26), la respuesta por parte del LAA es un `refuse`. En esta situación, el sistema aprovecha el tiempo de bloqueo para seguir calculando en búsqueda de unos parámetros mejores.

5.9.3. Flujo de mensajes en el proceso de estimación del estancamiento de la entrada y la salida de la planta

Este proceso permite estimar tanto el estancamiento de la entrada para inducir una falta de riqueza en la entrada para la identificación como de la salida de la planta, para el estudio del tipo del sistema y para aumentar la riqueza de la entrada.

En este flujo solamente se encuentran involucrados dos agentes de MASCONTROL,

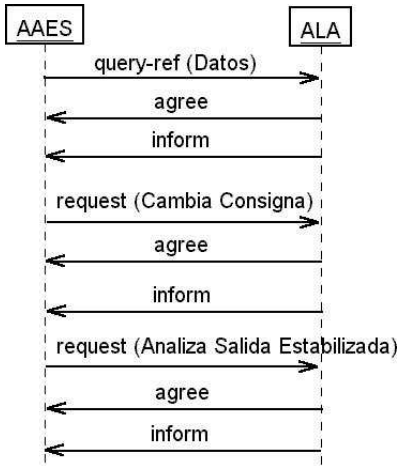

Figura 5.29.: Diagrama simplificado UML de flujo de mensajes en el proceso de estimación del estancamiento de la entrada y de la salida de la planta.

concretamente el AgenteLectorActuador y el AESA. Es este último agente quien solicita (protocolo *FIPA Query*) de forma periódica al LAA los datos de los nuevos pares de entrada-salida de la planta. En caso de que el AESA detecte una entrada estabilizada, inicia un protocolo *FIPA Request* solicitando que el LAA cambie la consigna. En caso de que la opción de cambio de consigna se encuentre bloqueada, la respuesta al protocolo será un acto de comunicación `refuse`. Este protocolo también se produce en caso de salida estabilizada, donde también se solicita (un nuevo *FIPA Request*) que el LAA estudie el tipo del sistema (el cual lo hará mediante consulta al OA), y sugiera un cambio en el controlador.

5.10. Fichero de perfiles

A continuación mostramos un ejemplo de fichero de perfiles, donde se demuestra la facilidad para definir y modificar los parámetros del sistema de control.

```
<?xml version="1.0" encoding="UTF-8"?>
<datosMAS>
  <planta descripcion="Tanques_interconectados"/>
  <accionControl = "AccionPI">
    <kp_inicial valor="1"/>
    <ki_inicial valor="0"/>
    <valorMaximoComando valor="5"/>
    <valorMinimoComando valor="0"/>
  </accionControl>
  <AgenteLectorActuador>
    <consigna valor="3"/>
    <muestreo tiempo ="1"/>
    <repintado_salida muestreos="10"/>
```

```
    </AgenteLectorActuador>
    <OpA ValorMetodoEntrenamiento="calculos.GradienteDescendente"
      ValorParametros="0,10000,0.01"
      ValorAlgoritmoOptimizacionPaso="calculos.AjusteCuadratico"
      ValorRed="redes/modelosistematanques.red"
      ValorModelo2entradas="redes/modelo2entradas.red"/>
    <OpA ValorMetodoEntrenamiento="calculos.AlgoritmoGenetico"
      ValorParametros="50,28,18,2,0,1000"
      ValorAlgoritmoOptimizacionPaso="calculos.SeccionOro"
      ValorRed = "redes/modelosistematanques.red"
      ValorModelo2entradas="redes/modelo2entradas.red"/>
    ....
</datosMAS>
```

5.11. Resultados

Las condiciones impuestas tanto por la velocidad en las comunicaciones en el interior de la red CyC pero sobre todo por el tiempo de procesamiento del sistema nos obliga a imponer ciertas consideraciones sobre el tipo de planta a controlar. En concreto, se deduce de forma inmediata que el MAS propuesto no es adecuado para controlar sistemas excesivamente rápidos (tal como se indica en [SAV+02]), o lo que es lo mismo, con una constante de tiempo demasiado pequeña, puesto que las diversas optimizaciones necesarias precisan un tiempo relativamente largo para alcanzar resultados fiables.

En este contexto, se ha escogido para las pruebas del MAS dos plantas existentes en los laboratorios gestionados por el grupo CyC de la ULL. Concretamente se ha empleado una planta de tanques interconectados y otra planta de control del nivel de agua en un depósito. Al comprobarse el funcionamiento del MAS con dos plantas distintas, se pretende que el modelo quede suficientemente validado.

5.11.1. Planta de Tanques Interconectados

5.11.1.1. Descripción de la planta

La primera planta con que se ha probado MASCONTROL ha consistido en una maqueta de tanques interconectados, concretamente el modelo CPI-100 (control de procesos industriales) de la empresa ALECOP [ALE]. Una fotografía de esta planta se muestra en la Figura 5.30.

El esquema de la planta (mostrado en la Figura 5.31) permite definir tres zonas bien diferentes:

- *Depósitos de trabajo*. La maqueta está compuesta por dos depósitos (tanques) exactamente iguales, sobre los cuales se pueden realizar mediciones simultáneas y estudiar las interacciones existentes cuando se conectan según diversas configuraciones. La única diferencia entre ambos tanques es que uno de ellos está montado sobre una plataforma de elevación, para poder así variar la altura relativa de uno de los depósitos respecto al otro.

Figura 5.30.: Planta de tanques interconectados.

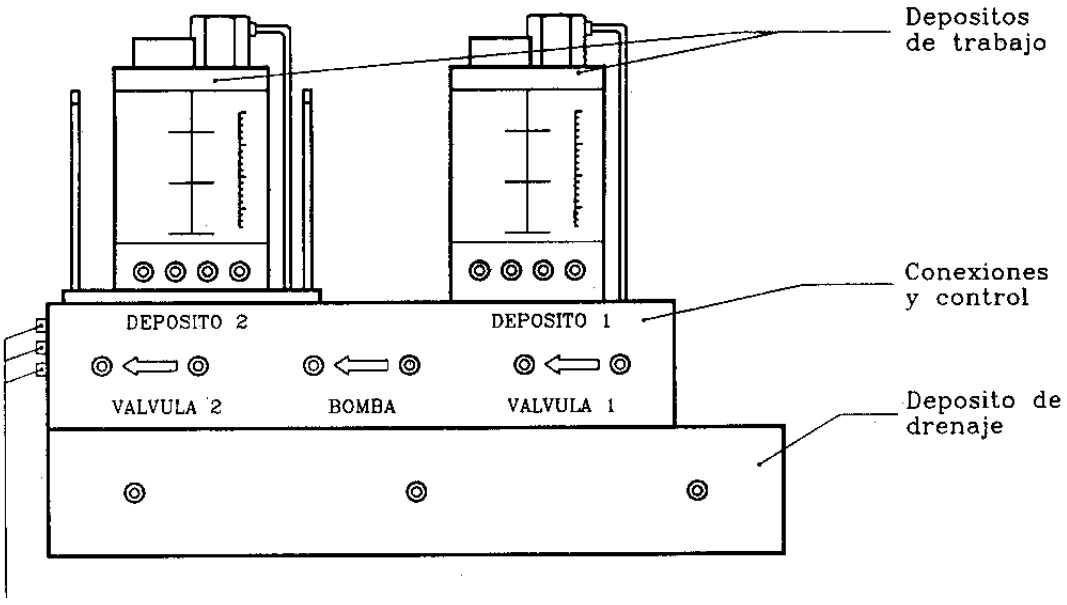

Figura 5.31.: Esquema de la planta de tanques interconectados.

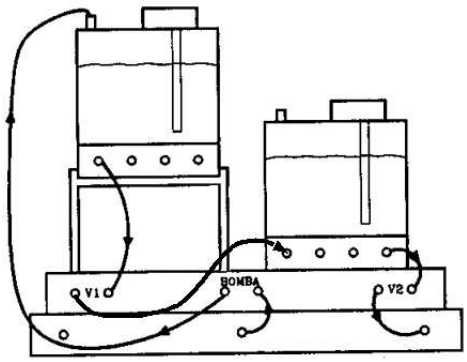

Figura 5.32.: Esquema de conexiones de la planta de tanques interconectados.

Cada uno de los depósitos está formado por los siguientes elementos:

- Captador de nivel: Este sensor es sobre el que tenemos más interés, puesto que la variable a controlar va a ser la altura del nivel de líquido en el tanque más bajo. Está constituido por una sonda capacitiva que aprovecha el fenómeno de la variación que se produce por la variación de nivel. Existe un condensador formado por la sonda de nivel y el volumen de líquido que hay en el depósito.
- Captador de temperatura
- Resistencia calefactora
- Agitador
- Termostato
- Tomas de entrada/salida de fluido
- Recipiente. Este recipiente está constituido por una base más una tapa de aluminio anodizado azul, unidas entre sí por cuatro vástagos roscados ejerciendo de elementos de fijación para un tercer elemento: un cuerpo cilíndrico de pírex, a través del cual se puede observar la evolución del líquido.

- *Conexiones y control*. Los depósitos van apoyados sobre una base de madera forrada. En su frontal aparecen tres grupos de dos tomas de entrada/salida, asociados a los elementos de control de flujo de líquido: 2 electroválvulas y una motobomba. Las electroválvulas son las encargadas de actuar de según el modo todo/nada para el paso del líquido de un tanque a otro. La motobomba es la encargada de generar el flujo de líquido a través de toda la maqueta. Esta motobomba es de corriente contínua y de 24 voltios nominales con la posibilidad de ser regulada desde 0 hasta la tensión nominal, pudiendo de esta manera variarse el caudal suministrado por la misma. Las conexiones para el montaje de los tanques interconectados, empleado por MASCONTROL, se muestra en la Figura 5.32.

- *Depósito de drenaje*. Mientras se está trabajando con la maqueta, se tiene una parte de líquido en los depósitos. El resto se encuentra en un depósito de drenaje que también almacena el total del líquido cuando no se trabaja con la maqueta. Este depósito, de 30 litros de capacidad, es de acero inoxidable y dispone de tres tomas de entrada/salida para su conexión con el resto de la maqueta.

Un módulo electrónico convierte el valor del captor de nivel en un voltaje que es la magnitud concreta que se pretende controlar mediante el MAS. La lectura de esta magnitud se realiza mediante una tarjeta conversora analógica-digital de 8 bits con una velocidad de lectura de 32000 adquisiciones por segundo. Del mismo modo, la actuación sobre la planta se realiza a través de un conversor digital-analógico. El comando se va a encontrar limitado entre los valores de 0-5 V debido a la tarjeta que se aplica a la entrada de la etapa de potencia de la bomba. Por tanto, la entrada a la planta es el voltaje suministrado por la tarjeta que fija el flujo de líquido bombeado al tanque, mientras que la salida es la lectura en voltaje que representa la altura que alcanza el líquido en el segundo tanque.

En este sentido, y basándonos por un lado en la velocidad de muestreo de la tarjeta conversora y de otro en la lentitud del sistema, no se ha tomado como valor de voltaje medido el de una única medición sino el promedio correspondiente a 1000 mediciones sucesivas. De este modo, se dispone de un dato con una mayor fiabilidad y que elimine en lo posible el ruido producido en el proceso de medición.

En cuanto a la vertiente software del sistema, el AgenteLectorActuador, implementado en Java como el resto de los agentes del MAS no puede acceder a la tarjeta conversora, al no estar implementados en este lenguaje el acceso a puertos de entrada/salida. Esta circunstancia se ha solucionado incluyendo métodos nativos escritos en lenguaje C.

5.11.1.2. Resultados

Con la arquitectura de agentes propuesta se realizaron varios experimentos con el objetivo de controlar la planta descrita. Los experimentos llevados a cabo se refieren a tres tipos de acciones de control: proporcional, proporcional-integral y realimentación por variables de estado. Conviene aclarar que la identificación se realizó, en todos los casos, a través de agentes simultáneos que optimizaban diversos modelos de la planta empleando diferentes métodos de optimización. Entre los tipos de modelos empleados destacamos los que implementan un sistema lineal. La Figura 5.33 muestra el esquema del modelo lineal de orden 2 construido mediante bloques de la herramienta Evenet2000. También destacaremos los que modifican este tipo de modelo incluyendo no linealidades. Un ejemplo de sistema no lineal se muestra en la Figura 5.34, el cual consiste en una modificación del modelo de la Figura 5.33, para incluir una no-linealidad mediante un bloque que representa una función tangente hiperbólica.

Los resultados obtenidos se muestran a continuación.

Acción de Control P

La acción de control proporcional se implementa en dos puntos concretos de la arquitectura. En primer lugar en el objeto *AccionControl* unido al LAA. En este caso

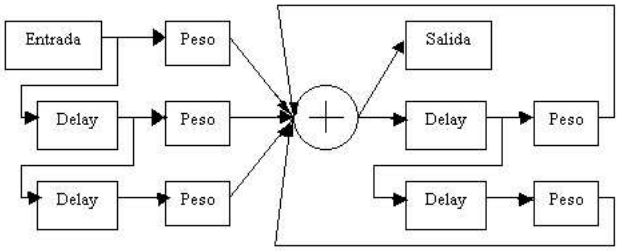

Figura 5.33.: Esquema de un modelo de un sistema lineal de orden 2.

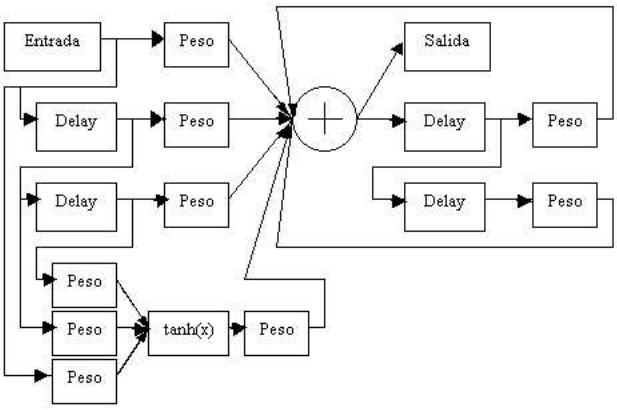

Figura 5.34.: Esquema de un modelo de un sistema lineal de orden 2 modificado para incluir una no-linealidad.

el objeto es de la subclase *AccionP*. El segundo sitio donde se indica es en el modelo de red cuyos parámetros se encarga de optimizar el conjunto de OpAs. En este caso el modelo es el de sistema de lazo cerrado con un único peso que multiplica al error. Respecto a este punto, la limitación en el valor del comando (entre 0 y 5 voltios) dificulta la optimización, introduciendo una no-derivabilidad en la función, modelada mediante un bloque cuya salida se encuentra acotada entre dichos límites. Debido a esta no-derivabilidad, no son adecuados los métodos basados en el gradiente. Por esta razón, en el proceso de entrenamiento se emplean sobre todo los algoritmos genéticos en los OpA. Puede definirse un enfoque alternativo empleando el método descrito en la sección 5.2.4, esto es, definiendo una función de coste (derivable) que penalice enormemente los valores de comando fuera de los límites deseados.

Inicialmente se parte de un valor de $K_p = 1$. Con este valor inicial se comienza una fase de aprendizaje donde continuamente se producen cambios de consigna en la planta, siempre evitando desbordamientos en los tanques e intentando dotar de suficiente riqueza a la entrada para mejorar la identificación. Se ha decidido que en esta etapa de aprendizaje la planta sea controlada por un PI, ya que se tiene la seguridad, a partir de la experiencia con este tipo de controladores, que es improbable que se produzcan efectos indeseados, por ejemplo, desbordamiento del sistema.

Cuando se estima que la identificación es conveniente, ya sea porque ha pasado suficiente tiempo, los parámetros se han estabilizado o se ha indicado de forma manual, se somete la entrada a la consigna deseada.

La gráfica de la Figura 5.35 muestra la optimización obtenida por los OpAs. Se intenta que el sistema siga los pares *consigna*, *consigna* (de modo que alcance la consigna pero penalizando los sobrepasamientos[17]). Para dotar de mayor información al patrón, éste se divide en cinco tramos con una secuencia creciente y decreciente. Los valores escogidos para la consigna son 3, 1, 2, 0.8 y 1.5. En esta figura se manifiesta que el sistema identificado es de tipo cero, puesto que el error en régimen permanente ante este esquema de realimentación no es nulo. Como es de esperar a partir de este dato, el sistema tiende a hacer crecer indefinidamente el valor de K_p, para disminuir lo más posible el coste producido.

La evolución de la salida de la planta ante una consigna de valor 3 voltios se muestra en la Figura 5.36. En esta figura se distinguen claramente las dos fases descritas: la de identificación (hasta un tiempo de 200 segundos) y la fase de consigna fija cambiando únicamente los parámetros del controlador. Como es de esperar, la salida de la planta no alcanza la consigna. Sin embargo, también se comprueba que a medida que aumenta el valor de K_p, disminuye el error en régimen permanente, tal como es de esperar y siempre dentro de las limitaciones de comando impuestas por el sistema.

Acción de Control PI

Tras la acción proporcional, se realizaron los mismos experimentos referidos a la acción proporcional-integral. Para ello se cambió el objeto de la clase *AccionP* por una instancia de la clase *AccionPI* y se modificó el modelo a optimizar por los OpAs. Estos son los dos únicos cambios que es necesario incluir en el sistema, debido a la modularidad con que está construída la arquitectura.

[17] Esta penalización depende de la forma de la función de coste.

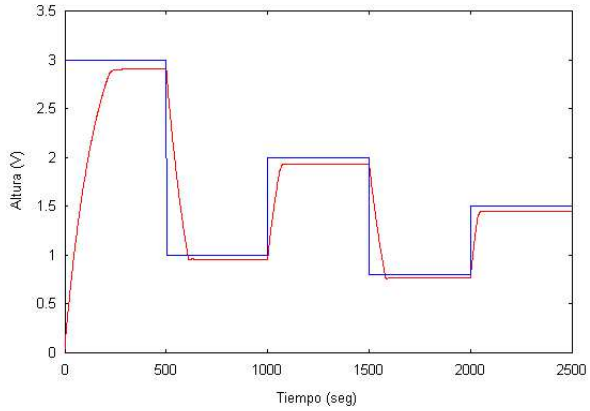

Figura 5.35.: Optimización de los parámetros del controlador P por parte de los AOps.

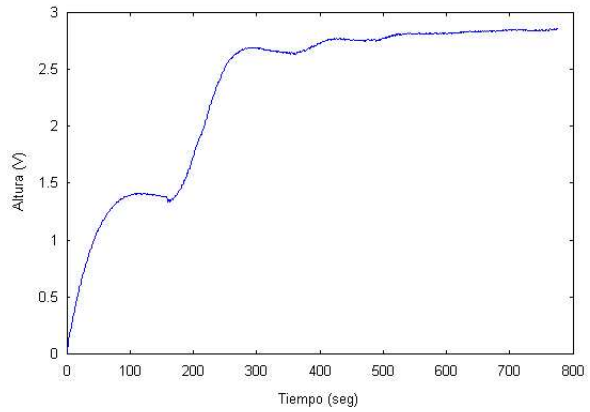

Figura 5.36.: Control de la planta mediante una acción proporcional.

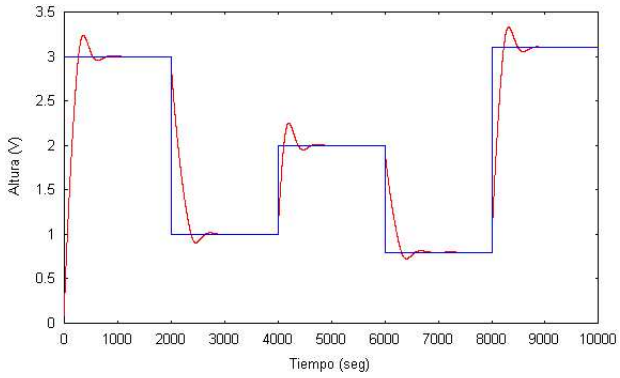

Figura 5.37.: Optimización de los parámetros del controlador PI por parte de los AOps.

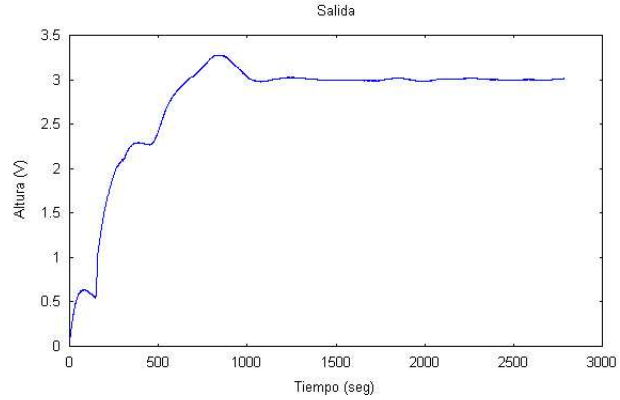

Figura 5.38.: Control de la planta mediante una acción proporcional-integral.

Si analizamos de nuevo la optimización realizada por los OpAs (en esta ocasión los valores de las consignas son distintos a los anteriores), nos encontramos con el comportamiento mostrado en la gráfica de la Figura 5.37. Como era de esperar, la salida del sistema optimizado alcanza el valor de la consigna en cada tramo, presentando un sobrepaso pequeño.

Esta mejora se refleja en la salida real de la planta (ver Figura 5.38). Como se comprueba, ésta alcanza el valor de consigna (en este caso 3), además de que el sobrepaso es pequeño.

El proceso, como en el caso de la acción proporcional, se ha realizado en dos fases, una de entrenamiento y otra de consigna fija.

Tal como se ha comentado anteriormente, el sistema de agentes crea de forma periódica una serie de ficheros (en DAML+OIL) con información de cuántas veces ha sido un entrenamiento y un modelo los mejores a la hora de optimizar los parámetros del sistema. El siguiente código es un extracto de uno de estos ficheros:

```
<RDFNsId2:Entrenamiento rdf:about="entr1"
  RDFNsId2:numeroVecesMejor="47">
```

```
    <RDFNsId2:ValorRed>redes/modelosistematanques2.red
    </RDFNsId2:ValorRed>
</RDFNsId2:Entrenamiento>
```

Como se indica en el apéndice J, en un futura implementación de la herramienta *Evenet2000*, se pretende que los ficheros de red sean estructurados en el lenguaje de marcas XML. Este hecho podría traducirse en que la información de la red pueda almacenarse directamente en DAML+OIL en vez de un fichero independiente.

El caso de la acción proporcional-integral nos ilustra sobre lo adecuado de emplear un agente de ontología en un MAS destinado a un problema tan aparentemente alejado del empleo de esta herramienta. En la figura 5.39, se muestra la evolución de la salida de la planta inicialmente con una acción proporcional pura. En un momento dado, entorno a los 650 segundos, el sistema multiagente (mediante el agente AESA) se da cuenta de que la salida se ha estabilizado en un valor alejado de la consigna. Entonces, el LAA pregunta al OA sobre qué acción de control le va bien a una planta cuando su salida no alcanza la consigna mediante una acción proporcional. El OA busca en su ontología y le responde que la acción buscada es una acción proporcional-integral. El sistema entonces realiza las modificaciones necesarias (acción de control y modelo de la red) para cambiar a una acción proporcional-integral. De nuevo resaltamos que este es un punto novedoso en el tratamiento de los problemas de control mediante sistemas multiagente. La salida de la planta, como era de esperar, alcanza, ahora sí, el valor de la consigna (3 voltios).

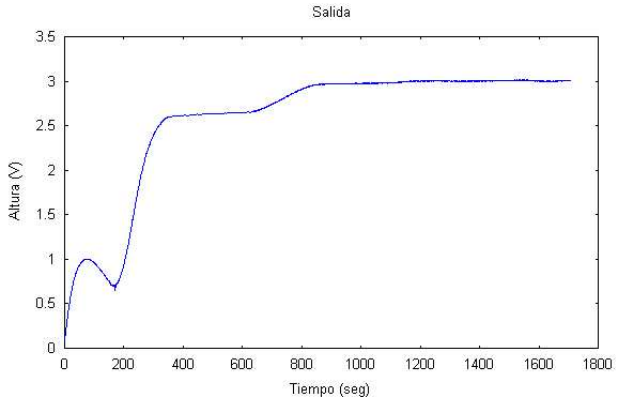

Figura 5.39.: Control de la planta mediante una acción proporcional modificada a una acción proporcional-integral.

Realimentación por Variables de Estado

Habiendo comprobado la bondad de la arquitectura de agentes MASCONTROL para entrenamientos del tipo proporcional y proporcional-integral, el próximo punto consiste en demostrar lo adecuado de la arquitectura para un sistema de control basado en otra acción completamente diferente: la realimentación por variables de estado. Como se deduce a partir del modelo de control, el controlador es más dependiente de la bondad de la identificación. En este sentido, por ejemplo, para un controlador PI, el comportamiento de la salida lleva a alcanzar la consigna independientemente

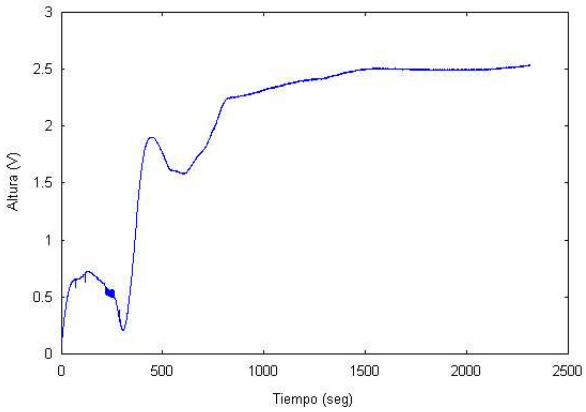

Figura 5.40.: Control de la planta mediante realimentación por variables de estado.

de que el sistema haya sido identificado convenientemente o no. En el caso de la realimentación por variables de estado, al menos la ganancia de la planta debe ser determinada adecuadamente para alcanzar la consigna.

A nivel práctico, se deben realizar algunas modificaciones menores en el sistema. La primera de ellas, claro está, se debe producir en el objeto *AccionControl*, que ahora debe implementar una realimentación por variables de estado. En concreto, suministra el comando sobre la planta a partir de la consigna y los valores de entrada y salida anteriores (se emplea la forma canónica controlable para la definición de las variables de estado). La otra modificación estriba en que ya carece de sentido la optimización de los parámetros del controlador. Simplemente se calcula ante una nueva identificación el valor de K y de Q. Desde un punto de vista práctico, esto provoca que, en este caso, el CCoA se limite a calcular estos valores.

Los resultados obtenidos se muestran en las figuras 5.40, 5.41, 5.42 y 5.43. En este caso, la fase de identificación se toma más larga que la realizada en los controladores P y PI, en búsqueda de que la identificación sea más efectiva. Por tanto, la salida de la planta en fase de identificación presenta picos, correspondientes a los tramos de diferente consigna.

Las Figuras 5.40 y 5.41 se refiere a un valor de $K = 0$ en el lazo de realimentación de las variables. Esto se corresponde a un modulado de la ganancia con el comportamiento global determinado únicamente por los polos originales de la planta. Como puede observarse, la salida del sistema tiende a la consigna fijada (un valor de 2.5 voltios en la Figura 5.40 y valores de 2.5 y 3 en la Figura 5.41).

En contraste, las Figuras 5.42 y 5.43, corresponden a situaciones de $K \neq 0$. En este caso, se han escogido valores para los polos deseados cercanos relativamente a los polos detectados en los experimentos anteriores. De este modo, la acción de realimentación no se ve afectada por la limitación en el comando entrante a la planta. En todos los casos se obtiene un comportamiento aceptable, puesto que la salida de la planta se estabiliza entorno a los valores de consigna (1 voltio en el caso de la Figura 5.42, y en los dos tramos de la 5.43, correspondiente a valores de 2 y 2.5 voltios), aunque presentando algunas oscilaciones.

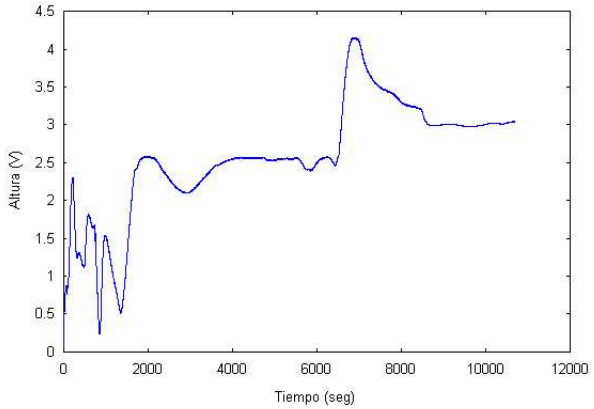

Figura 5.41.: Control de la planta mediante realimentación por variables de estado.

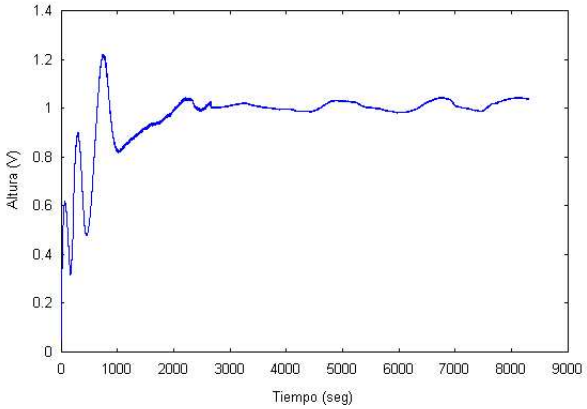

Figura 5.42.: Control de la planta mediante realimentación por variables de estado con una consigna final de 1 voltio.

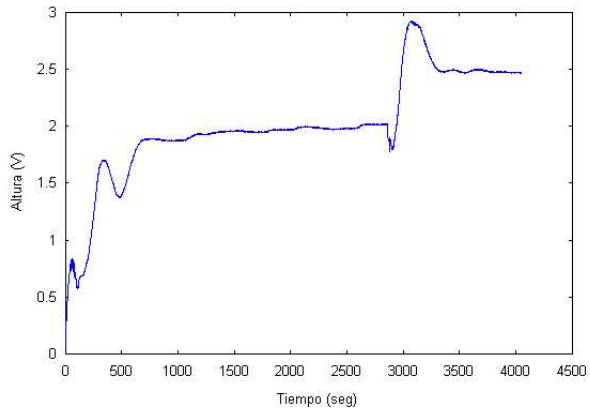

Figura 5.43.: Control de la planta mediante realimentación por variables de estado con consignas de 2 y 2.5 voltios.

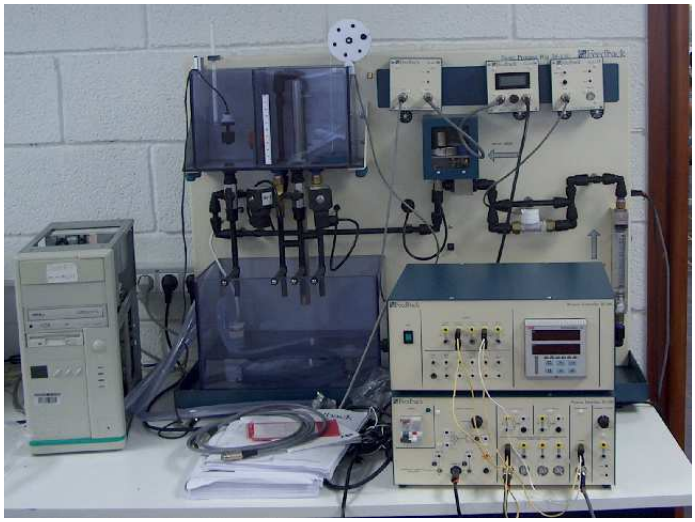

Figura 5.44.: Fotografía de la Planta de Control de Nivel de Agua en un Depósito.

5.11.2. Planta de Control de Nivel de Agua en un Depósito

5.11.2.1. Descripción de la planta

Esta planta consiste en un módulo educativo para el control de procesos de la empresa Feedback[18] de la serie PROCON (*Process Control Trainer*). Concretamente se trata del sistema de control sobre nivel y flujo 38-001[19] [Fee00].

La planta (una fotografía de la misma puede observarse en la Figura 5.44) consiste en un circuito cerrado de agua que permite el estudio de los principios del control de procesos empleando como variables a controlar el nivel y el flujo de líquido en el circuito. El circuito se encuentra soportado por un panel vertical que lo hace adecuado para las prácticas en laboratorio.

La planta comprende un tanque superior dual (donde se mide el nivel de agua) unido a un tanque depósito por medio de cinco válvulas manuales y tres electroválvulas. El agua es bombeada desde el tanque depósito hasta el tanque superior (hasta un máximo de 5 litros por minuto) a través de un medidor de flujo de área variable (rotámetro) y una servoválvula de apertura controlable. El flujo de líquido se mide mediante un medidor óptico por pulsos. El esquema de la planta puede observarse en la Figura 5.45).

Esta planta funciona mediante un comando de intensidad estándar de 4-20 mA. No obstante, la tarjeta conversora disponible funciona en tensión. Por tanto, si deseamos emplear el mismo equipo de adquisición/envío de datos debemos implementar sendos circuitos de conversión. La Figura 5.46 corresponde a un circuito de conversión de niveles de tensión de 0-5V a 0-20 mA. Este circuito es compatible con la entrada de 4-20 mA. mediante la inclusión de un offset. La conversión de la salida de la planta (en intensidad) en entrada a la tarjeta conversora (en tensión) se lleva a cabo fácilmente mediante el paso de la corriente a través de una resistencia de 330 ohmios. Con estos

[18] http://fbk.com

[19] http://www.fbk.com/full/products/pdfs/process/proconcts.pdf

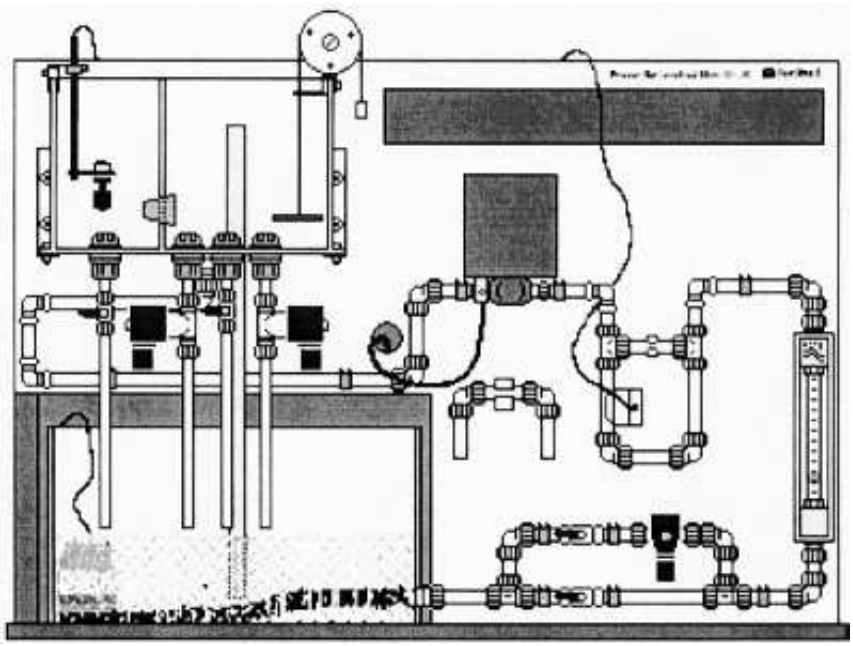

Figura 5.45.: Esquema de la Planta de Control de Nivel de Agua en un Depósito.

circuitos se logra, además, que el código nativo C de lectura/envío de datos permanezca inalterado.

Con estas premisas, se han realizado experimentos similares a los llevados a cabo con la primera planta analizada. La variable controlada en toda esta serie de experimentos ha sido la altura del fluido en el tanque (medido a partir del ángulo de giro de una polea de la que cuelga un flotador contrapesado por el otro lado de la polea) como voltaje leído por la tarjeta conversora. El periodo de muestreo empleado ha variado de 0.5 a 1 segundo.

5.11.2.2. Resultados

En esta planta se han realizado pruebas similares a las llevadas a cabo con la anterior planta. Sin embargo en este apartado de resultados nos hemos restringido a las acciones PI y a la realimentación por variables de estado.

Acción de Control PI

Del mismo modo que con la anterior planta, reproducimos aquí las figuras correspondientes a la optimización por parte de los OpAs (Figura 5.47) así como la evolución de la salida de la planta ante una serie de consignas sucesivas (Figura 5.48). El comportamiento es análogo al contemplado con la planta anterior (se alcanza la consigna en todos los tramos). No obstante, destacaremos que la salida de la planta alcanza la consigna en todos los casos sin sobrepasamiento, tal como predice la optimización de los OpA, y con el mismo tiempo (aproximadamente 150 segundos).

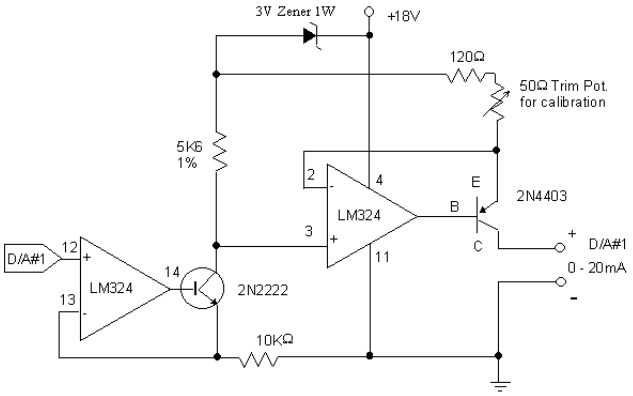

Figura 5.46.: Circuito de conversión 0-5 V a 0-20 mA.

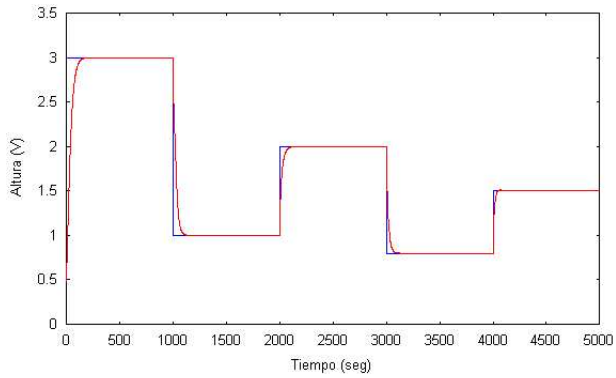

Figura 5.47.: Optimización de los parámetros del controlador PI por parte de los AOps.

Este hecho supone un respaldo al sistema.

Realimentación por Variables de Estado

Las Figuras 5.49, 5.50 y 5.51 reflejan distintas evoluciones de la salida de la planta en una realimentación por variables de estado. En todas estas figuras se pueden distinguir claramente las fases de entrenamiento para la identificación (los picos iniciales) donde como en el caso de los tanques interconectados, la planta se encuentra controlada por un controlador PI. De nuevo, el comportamiento es aceptable, puesto que, con algunas oscilaciones ocasionales, se alcanza la consigna prefijada.

5.12. Inclusión de un nuevo agente en MASCONTROL

Como en el caso del sistema multiagente MASplan, dedicaremos una sección a ilustrar el mecanismo de inclusión de un nuevo agente en el sistema multiagente MASCONTROL. Para ello tomaremos el ejemplo de los pasos a seguir por un usuario que desee implementar un agente identificador con un sistema de optimización

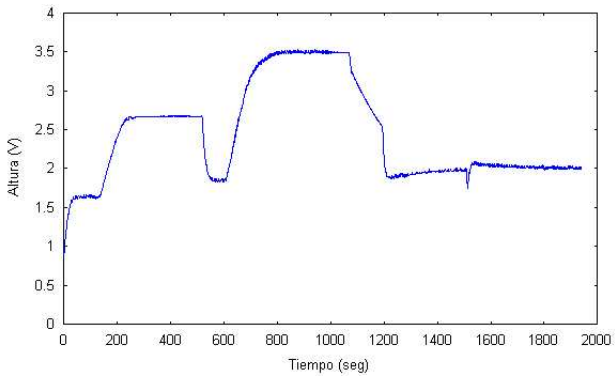

Figura 5.48.: Control de la planta mediante una acción proporcional-integral.

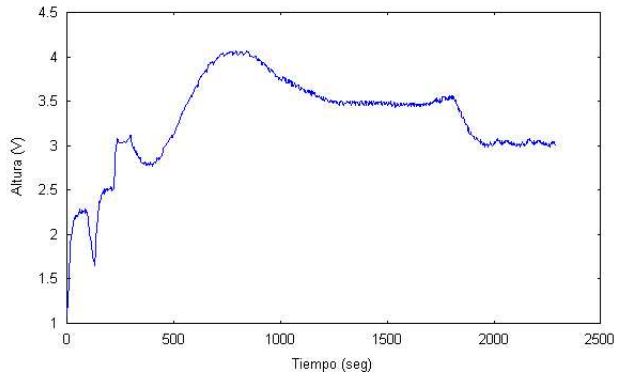

Figura 5.49.: Control de la planta mediante realimentación por variables de estado. Consignas sucesivas de 3.5 y 3 voltios.

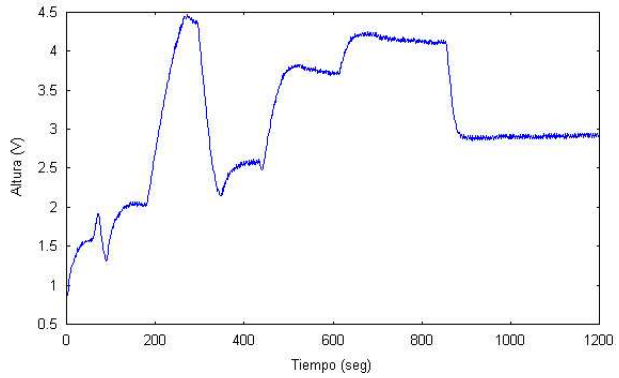

Figura 5.50.: Control de la planta mediante realimentación por variables de estado. Consignas sucesivas de 2.5, 3.7, 4 y 3 voltios.

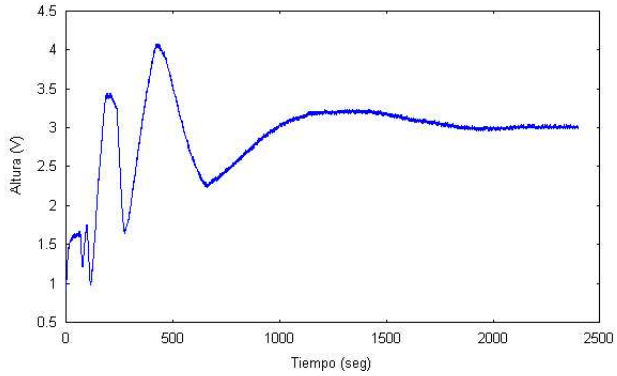

Figura 5.51.: Control de la planta mediante realimentación por variables de estado. Consigna de 3 voltios.

diferente.

Estructuraremos la sección en dos partes: la relativa al código Java (aunque volvemos a indicar que el código puede estar implementado en otro lenguaje) del nuevo agente y la del resto de los cambios necesarios.

5.12.1. Código Java de implementación del nuevo agente

Empezaremos este apartado haciendo hincapié en los paquetes Java que se precisan importar. De nuevo serán necesarios los referentes a la implementación de las conversaciones, tareas, transporte de mensajes... dentro de la distribución FIPA-OS.

```
import fipaos.ont.fipa.*;
import fipaos.agent.*;
import fipaos.platform.*;
import fipaos.util.DIAGNOSTICS;
```

Por otra parte, si se desean reutilizar los paquetes `calculos` (para la optimización de parámetros) y `editorGrafico` (para el manejo de los ficheros de redes) de la herramienta Evenet2000,

```
import cyc.evenet2000.calculos.*;
import cyc.evenet2000.editorGrafico.*;
```

Para que el agente pueda emplear los métodos de envío de mensajes, registro con el AMS y DF, hereda de la clase *FIPAOSAgent* incluido en la distribución de la herramienta FIPA-OS.

```
public class NuevoAgenteIdentificador extends FIPAOSAgent
```

En el código relativo al registro con el DF, se debe reflejar el nuevo método optimización que el agente implementa.

```
  try
  {DFAgentDescription descripcion = new DFAgentDescription ();
  ...
  PropertyTemplate[] u = new PropertyTemplate[2];
  // se definen las propiedades de AccionSoportada con valor
  // Identificacion (igual que el resto de AIs) y
  // el nuevo método de identificación
  u[0] = new PropertyTemplate("AccionSoportada",
"Identificacion");
  u[1] = new PropertyTemplate("MetodoEntrenamiento",
"NuevaIdentificacion");
  serv[0].setProperties(u);
  ...
  registerWithDF( null, descripcion);
  } catch (Exception e) { e.printStackTrace();}
```

Por último, el nuevo agente debe asimilar los flujos de mensajes correspondientes. (ver sección 5.9). En el caso de un nuevo IA, esto se reduce al flujo de mensajes con el CIA para la identificación del sistema.

```
  // tarea que procesa por defecto los mensajes recibidos por el
agente
  private class IdleTask extends Task
  {

  // procesa los mensajes con acto de comunicación request
    public void handleRequest(Conversation id)
    {
     ...
  // si en el contenido aparece la orden de "NuevoEntrenamiento"
     if (contenido==“NuevoEntrenamiento”){
     _tm.newTask(new nuevoEntrenamientoTask(id), id);
     }
     ...
    }

  }

  // aquí debe ir el código que implementará
  // el nuevo método de entrenamiento
  public class nuevoEntrenamientoTask extends Task
  {...
  }
```

A la vista de estas modificaciones, y tal como se demostró para el sistema MASplan, la inclusión de nuevos agentes codificados por otros usuarios puede ser llevada a cabo de un modo sencillo.

5.12.2. Resto de modificaciones

Estas modificaciones en el código Java, pueden verse complementadas con cambios en otros módulos del MAS, como la ontología y los ficheros de perfiles.

Con respecto a la ontología puede definirse, por ejemplo, un nuevo individuo de la clase MetodoOptimizacion relativo al método de optimización implementado. De este modo la información estará disponible para el resto del sistema (por ejemplo el CIA puede crear un IA según este nuevo método).

```
<rdf:Description rdf:about="#NuevaIdentificacion">
  <rdf:type>
    <daml:Class rdf:about="#MetodoOptimizacion"/>
  </rdf:type>
</rdf:Description>
```

Por otro lado, el nuevo agente puede suponer modificaciones en los ficheros de perfiles. Así, por ejemplo, si se desea que el CaIA inicialice uno de estos sistemas, el código XML a incluir es:

```
<nuevoAgenteIdentificador atributos="..." />
```

Conclusiones y Líneas Abiertas

El trabajo realizado en el presente proyecto de tesis, así como sus principales aportaciones y conclusiones se pueden resumir en los siguientes puntos:

- Se ha llevado a cabo un estudio detallado del concepto de agente y de los diversos lenguajes y arquitecturas de sistema multiagente existentes, dedicando especial atención a los estándares más extendidos: KQML, FIPA y OMG MASIF.
- Se ha realizado un análisis de los marcos de trabajo para la implementación de sistemas multiagente.
- Se ha implementado un primer prototipo de arquitectura MAS codificado en Java utilizando el intercambio de mensajes KQML, con el objetivo de controlar en simulación la altura de una plataforma instalada en un robot móvil.
- Se ha comparado la arquitectura diseñada en el prototipo inicial con los estándares estudiados, comprobando sus similitudes con el estándar FIPA.
- Se ha constatado el estándar FIPA como el más adecuado para nuestros propósitos de implementación, principalmente por referirse a aspectos más generales que el simple intercambio de mensajes.
- Se ha elegido, a partir de los estudios descritos, la herramienta FIPA-OS como marco de trabajo de MAS adecuado ya que ofrece las suficientes prestaciones para el trabajo planteado.
- Se ha realizado un estudio de los lenguajes de representación del conocimiento y de los lenguajes de marcas en particular.
- Se ha demostrado el poder de los lenguajes de marcas en los sistemas multiagentes, un aspecto en la vanguardia dentro de este campo.
- Se han diseñado e implementado una primera arquitectura de sistema multiagente para la planificación de reuniones y reserva de material en el escenario de un grupo de investigación universitario (MASplan).
- Estudio de las técnicas de negociación en general, y entre agentes en particular.
- Se han aportado variaciones sobre diferentes algoritmos de votación (como el recuento de Borda en el caso de la planificación de reuniones) y negociación bilateral (definición de un factor de egoísmo en la gestión de los recursos comunes).

- Estudio de la técnica de los árboles de identificación.
- Se ha aportado un cierto grado de autonomía (de momento reducido) a los agentes, mediante la implementación de un sistema de reglas obtenidas mediante el empleo de la técnica de los árboles de identificación. Estas reglas se deducen de las decisiones tomadas previamente por el usuario en situaciones análogas.
- Se han diseñado e implementado una segunda arquitectura de sistema multiagente para la identificación y control de sistemas (MASCONTROL).
- Se han estudiado los diferentes tipos de esquemas de control de sistemas y su enfoque a través de los sistemas multiagentes.
- Identificación y Control de dos plantas diferentes: tanques interconectados y planta de control de nivel de agua en un depósito, según acciones de control proporcional, proporcional-integral y asignación de polos.
- Integración de la herramienta *Evenet2000*, con lo que se ha demostrado la posibilidad de emplear programas o módulos de programas ya existentes en los MAS. Esta herramienta había sido implementada con anterioridad por el autor de este trabajo, empleando para ello, la técnica de Grafo de Flujo de Señal.
- A partir de la arquitectura propuesta para el sistema MASCONTROL, se ha demostrado lo adecuado del empleo de la herramienta *Evenet2000*, implementando diferentes controladores como el proporcional, proporcional-integral y la asignación de polos.
- Se han diseñado y codificado de sendas ontologías en DAML+OIL en los MAS implementados.
- Se ha implementado un agente de ontología para los MAS que cumpla con las especificaciones FIPA.
- Estudio y descripción de la facilidad de inclusión de otros agentes, incluso pudiendo estar codificados en otros lenguajes diferentes al Java y/o por parte de otros programadores, empleando el paradigma de la programación orientada a agentes.

Líneas Abiertas

A lo largo de todo este trabajo, se han indicado una serie de líneas abiertas. Podemos dividirlas en dos grupos bien diferenciados. En primer lugar listaremos las referidas a mejoras del funcionamiento de los sistemas implementados así como a la realización de nuevas pruebas.

- Empleo del sistema experto Jess en algoritmos más complejos para ambos MAS.
- Inclusión de agentes móviles en los MAS.
- Integración de la librería ControlWeb, desarrollada en el seno del Grupo de Computadoras y Control de la Universidad de la Laguna, en MASCONTROL.

- Mejora de la interfaz gráfica de los UAs en MASplan para hacer más atractivo su empleo a los usuarios. En este sentido, integrar un sistema de calendario preprogramado no comercial.
- Estudio de mejoras sobre la arquitectura de MASCONTROL. Cambios producidos por la inclusión de varios CIAs...
- Migración de los ficheros de Evenet2000 a una estructura XML que permita una mayor interactuación con las ontologías diseñadas.
- Identificación y control de más tipos de plantas mediante MASCONTROL.
- Migración de las ontologías de los sistemas MASplan y MASCONTROL del lenguaje DAML+OIL a OWL. De este modo se pueden explotar definiciones proporcionadas por este último como `owl:sameIndividualAs`.
- Modificaciones en las arquitecturas propuestas mediante la inclusión de otros tipos de agentes: AgenteCajero, AgenteFichero...
- Implementación en MASCONTROL de técnicas de control más complejas como por ejemplo de control óptimo y robusto.
- Estudio sobre nuevos usos de un agente de ontología en el sistema MASCONTROL.

Por otro lado, citaremos aquellas líneas abiertas que suponen unos cambios de mayor envergadura y que afectan a la estructura básica y las capacidades de los MAS.

- Implementación de un entorno amigable de creación y edición de agentes, lo que facilitaría a un usuario ajeno a los estándares FIPA desarrollar agentes que interactuasen con el resto del MAS.
- Implementación de negociaciones de tipo más complejo como coaliciones de agentes.
- Inclusión de los *bytecodes* de las clases Java de los agentes dentro de la ontología, pudiendo llegar a inicializarse estos agentes si el MAS lo precisase.
- Implementación de las modificaciones para dotar de inferencia al DF.
- Estudio y aplicación de los MAS a otros campos: robots móviles, sistemas multi-robot, tutoriales inteligentes...

A. Performativas o actos de comunicación KQML

En este apéndice presentamos el conjunto de performativas del lenguaje KQML [FWW+93, LF97].

ask-if El emisor desea conocer si el contenido del mensaje es cierto para el receptor (se encuentra en la base de conocimiento del receptor).

ask-all El emisor desea conocer todas las instancias del contenido del mensaje en la base de conocimiento del receptor.

ask-one El emisor desea conocer una de las instancias del contenido del mensaje en la base de conocimiento del receptor.

stream-all Versión de ask-all, pero implicando una respuesta múltiple.

eos Indica el final de una respuesta múltiple (provocada por un *stream-all*).

tell El contenido está en la base de conocimiento del emisor (es cierto para el emisor).

untell El contenido no está en la base de conocimiento del emisor (no es cierto para el emisor).

deny La negación del contenido se encuentra en la base de conocimiento del emisor.

insert El emisor pide al receptor que añada el contenido del mensaje a su base de conocimiento.

uninsert El emisor pide al receptor que anule un acto previo de *insert.*

delete-one El emisor quiere que el receptor elimine una instancia del contenido en su base de conocimiento.

delete-all El emisor quiere que el receptor elimine todas las instancias del contenido en su base de conocimiento.

undelete El emisor pide al receptor que anule un acto previo de *delete.*

achieve El emisor pide al receptor que ejecute una acción sobre su entorno físico.

unachieve El emisor pide al receptor que anule un acto previo de *achieve.*

advertise El emisor desea que el receptor conozca que el emisor procesará un mensaje como el que aparece en el contenido del mensaje.

unadvertise El emisor desea que el receptor conozca que anula un *advertise* previo y no procesará más mensajes como el que aparece en el contenido del mensaje.

subscribe El emisor desea que el receptor le envíe un mensaje cada vez que se actualiza el resultado de la acción indicada en la performativa.

error El emisor considera que el mensaje previo mandado por el receptor está mal formado (no cumple la sintaxis KQML).

sorry El emisor entiende el mensaje enviado el receptor pero es incapaz de proporcionar una respuesta más informativa.

standby El emisor desea que el receptor anuncie que está preparado para proporcionar una respuesta al mensaje que aparece en el contenido.

ready El emisor está preparado para responder a un mensaje previamente recibido desde el receptor.

next El emisor quiere la siguiente respuesta del receptor a un mensaje previamente enviado por el emisor.

rest El emisor desea el resto de las respuestas del receptor a un mensaje previamente enviado por el emisor.

discard El emisor no quiere el resto de respuestas del receptor a un mensaje previamente enviado por el emisor.

register El emisor anuncia al receptor su presencia y su nombre simbólico.

unregister El emisor pide al receptor que anule un acto previo de *register*.

forward El emisor quiere que el receptor reenvíe el mensaje al agente que aparece en el campo *:to* del mensaje. (El receptor podría también ser ese agente).

broadcast El emisor quiere que el receptor reenvíe el mensaje a todos los agentes cuya existencia conoce el receptor.

transport-address El emisor asocia su nombre simbólico con una nueva dirección.

broker-one El emisor quiere que el receptor encuentre una respuesta a una performativa (otro agente distinto al receptor va a proporcionar esa respuesta).

broker-all El emisor quiere que el receptor encuentre todas las respuestas a una performativa (otro agente distinto al receptor va a proporcionar esa respuesta).

recommend-one El emisor quiere saber de un agente que pueda responder a una performativa.

recommend-all El emisor quiere saber de todos los agentes que puedan responder a una performativa.

recruit-one El emisor quiere que el receptor reclute un agente adecuado para responder a una performativa.

recruit-all El emisor quiere que el receptor reclute a todos los agentes adecuados para responder a una performativa.

B. Marco para la semántica KQML

En este apéndice presentamos los operadores propuestos por Finin y Labrou [LF94] para la descripción del lenguaje de comunicación de agentes KQML.

Respecto a las actitudes mentales:

Bel(A,P) La proposición P es cierta para el agente A. Dicho de otro modo, P se encuentra en la base de conocimiento de A.

Know(A,P) El agente A conoce P, es decir, A tiene conciencia de P.

Want(A,P) El agente A desea que el evento o estado descrito por P ocurra.

Intend(A,P) El agente A tiene todas las intenciones de llevar a cabo P y por tanto se compromete a llevar a cabo P en el futuro.

Respecto a las acciones:

Proc(A,M) Se refiere a la acción por parte del agente A de procesar el mensaje M

sendMSG(A,B.M) Indica la acción del agente A de mandar el mensaje M al agente B

C. Actos de comunicación FIPA

En este apéndice presentamos el conjunto de actos de comunicación del estándar FIPA [FIP02c].

Accept Proposal Acto de aceptar una proposición remitida anteriormente para ejecutar una acción, generalmente a través de un *propose*

Agree Acto de manifestar estar de acuerdo en ejecutar una acción, posiblemente en el futuro

Cancel Acto en que un agente informa a otro agente que el primer agente no mantiene la intención de que el segundo agente ejecute una acción

Cfp (Call for Proposals) Acto de pedir propuestas para ejecutar una acción

Confirm El emisor informa al receptor de que una proposición dada es cierta, donde se entiende que el receptor presenta cierta incertidumbre acerca de dicha proposición

Disconfirm El emisor informa al receptor que una proposición dada es falsa, donde se entiende que el receptor presenta cierta incertidumbre acerca de dicha proposición

Failure Acto de comunicar a otro agente que se intentó llevar a cabo una determinada acción, pero dicho intento fracasó

Inform El emisor informa al receptor que una determinada proposición es verdadera

Inform If Macro para que el agente encargado de realizar una acción informe al receptor sobre si una determinada proposición es verdadera o no

Inform Ref Macro para que el emisor informe al receptor sobre el objeto que corresponde a un cierto descriptor

Not Understood El emisor informa al receptor que ha percibido que el receptor ha llevado a cabo alguna acción, pero que es incapaz de entender que es lo que ha realizado.

Propagate El emisor intenta que el receptor trate el mensaje embebido como si hubiese sido enviado directamente al receptor, y quiere que el receptor identifique a los agentes denotados por un descriptor dado y les envíe el mensaje a propagar

Propose Acto de presentar una propuesta para llevar a cabo cierta acción, dadas ciertas condiciones previas

Proxy El emisor quiere que el receptor seleccione unos agentes de destino determinados por una descripción dada y les envíe el mensaje embebido a ellos

Query If Acto de preguntar a otro agente si una proposición dada es cierta o no

Query Ref Acto de preguntar a otro agente por el objeto referenciado por una determinada expresión

Refuse Acto de rechazar llevar a cabo una acción dada, explicando la razón para ese rechazo

Reject Proposal Acto de rechazar una propuesta para ejecutar una acción dada durante una negociación

Request El emisor quiere que el receptor ejecute una acción determinada. Un uso importante de este acto consiste en pedir que el receptor lleve a cabo otro acto comunicativo

Request When El emisor quiere que el receptor lleve a cabo una determinada acción cuando se cumplan ciertas condiciones previas

Request Whenever El emisor quiere que el receptor lleve a cabo una cierta acción tan pronto como se cumpla una condición previa dada y a partir de entonces, cada vez que se vuelve a cumplir dicha proposición.

Subscribe Acto de pedir la intención persistente de notificar al emisor el valor de una referencia y notificarlo de nuevo si dicho objeto sufre algún cambio.

D. Marco para la semántica FIPA ACL

En este apéndice realizaremos una breve descripción del lenguaje formal (llamado *Semantic Language*, SL) empleado en la descripción de la semántica FIPA ACL [FIP02c].

Una definición más completa puede encontrarse en [FIP02d]

D.1. Definiciones básicas SL

En SL, las proposiciones lógicas se expresan en una lógica de actitudes mentales y acciones, formalizados en un lenguaje modal de primer orden. Los componentes del formalismo empleados son:

- p, p1,... son fórmulas cerradas que denotan proposiciones.
- ϕ y ψ son esquemas de fórmulas, los cuales aparecen en toda proposición cerrada.
- *i* , *j* denotan agentes.
- $\models \phi$ significa que ϕ es válida.

El modelo mental de un agente está basado en la representación de tres actitudes primitivas: creencia (*believe*), incertidumbre (*uncertainty*) y elección (*choice)*[1]. Estas actitudes se encuentran formalizadas respectivamente por los operadores *B*, *U* y *C*. De forma que las siguientes fórmulas deben interpretarse del modo indicado.

$B_i p$ El agente *i* cree *p*.

$U_i p$ El agente *i* tiene dudas acerca de *p*, pero piensa que *p* es más probable que $\neg p$.

$C_i p$ El agente *i* desea actualmente que se dé *p.*

Para dotar de razonamiento a propósito de acciones, el universo del discurso implica secuencias de eventos además de objetos y agentes individuales. Una secuencia puede estar compuesto por un único evento, incluso un evento vacío.

$a_1 ; a_2$ es una secuencia en la que el evento a_2 sigue al evento a_1.

$a_1 | a_2$ es una elección no determinista, en la que ocurre el evento a_1 o evento a_2, pero no ambos a la vez.

Los operadores *Feasible* (factible), *Done* (hecho), *Agent (*agente) y *Single* (sencillo) son introducidos para permitir el razonamiento acerca de acciones.

[1] En un sentido amplio: objetivo, meta (*goal)*

Feasible (a,p) La acción *a* puede tener lugar y si lo hace, *p* se hará cierta justo después de la acción.

Done (a,p) La acción *a* ha tenido lugar y *p* se ha hecho cierta justo después de la acción.

Agent (i,a) El agente *i* es el único que puede llevar a cabo la acción *a* (ya sea en el pasado, presente o futuro)

Single (a) La acción *a* indica una expresión de una acción que no es una secuencia. Cualquier acción individual es *Single*. El acto compuesto **a ; b** no es *Single*. Finalmente el acto compuesto **a | b** es *Single*, si tanto a como b lo son.

Se define el concepto de objetivo persistente (*persistent goal*) del siguiente modo: un agente *i* tiene un objetivo persistente, si *i* tiene a *p* como meta y se auto-compromete en ese objetivo hasta que *i* llega a creer que la meta se ha cumplido o cree que es inalcanzable. Se denota como $\mathbf{PG_i p}$.

El concepto de intención (*intention*) se define como un objetivo persistente que obliga al agente a actuar. En SL se denota como $\mathbf{I_i p}$.

Obsérvese que no existe restricción alguna sobre la posibilidad de ensamblar operadores de actitudes mentales o acciones. Así, por ejemplo, la fórmula

$$U_i B_j I_j \text{Done (a, } B_i p)$$

significa que el agente *i* cree que, probablemente, el agente *j* cree que *i* tiene la intención de que la acción *a* sea llevada a cabo antes de que el agente *i* tenga que creer *p*.

D.2. Abreviaturas

En SL, se emplean, entre otras, las siguientes abreviaturas

1. Feasible (a) $\equiv$ Feasible (a, True)
2. Done (a) $\equiv$ Done (a, True)
3. Possible (ϕ) $\equiv$ ($\exists$a) Feasible (a,ϕ)
4. $\text{Bif}_i\phi \equiv B_i\phi \vee B_i\neg\phi$
5. $\text{Uif}_i\phi \equiv U_i\phi \vee U_i\neg\phi$
6. $AB_{n,i,j}\phi = B_i B_j B_i ...\phi$
   Aquí se introduce el concepto de *creencias alternas*, *n* es un número natural que representa el número de operadores B que se alternan entre *i* y *j*.

D.3. Operadores Referenciales

SL ofrece tres operadores referenciales:

`iota` Introduce una expresión en la cual un identificador dado está definido, cuando en otras circunstancias no lo estaría. La expresión `(iota x (P x))` debe interpretarse como "la x tal que P [es verdadera] de x".

`any` Este operador se emplea para denotar cualquier objeto que satisfaga una proposición dada.

`all` Este operador se emplea para denotar el conjunto de todos los objetos que satisfacen la proposición dada.

D.4. Ejemplos

No es objetivo de este trabajo entrar en demasiados detalles más sobre la gramática del SL. Para ello, el lector interesado puede consultar [FIP02d]. Sin embargo, proporcionaremos el siguiente ejemplo, que refleja una interacción entre dos agentes A y B empleando el operador `iota`. Se supone que el agente A tiene la siguiente base de conocimiento KB={P(C), Q(1,A), Q(1,B)}.

El agente B hace una consulta a A sobre qué elemento x satisface la relación P(x)

```
(query-ref
:sender (agent-identifier :name B)
:receiver (set (agent-identifier :name A))
:content
"((iota ?x (p ?x)))"
:language fipa-sl
:reply-with query1)
```

El agente A responde, tras consultar su base de conocimiento, que el único elemento que cumple con la relación es C.

```
(inform
:sender (agent-identifier :name A)
:receiver (set (agent-identifier :name B))
:content
"((= (iota ?x (p ?x)) c))"
:language fipa-sl
:in-reply-to query1)
```

El agente B hace una nueva consulta a A sobre qué elemento x satisface la relación Q(x,y)

```
(query-ref
:sender (agent-identifier :name B)
```

```
:receiver (set (agent-identifier :name A))
:content
"((iota ?x (q ?x ?y)))"
:language fipa-sl
:reply-with query2)
```

El agente A responde, de nuevo, tras consultar su base de conocimiento, que el único elemento que cumple con la relación es 1, tanto en Q(1,A) como en Q(1,B).

```
(inform
:sender (agent-identifier :name A)
:receiver (set (agent-identifier :name B))
:content
"((= (iota ?x (q ?x ?y)) 1))"
:language fipa-sl
:in-reply-to query1)
```

E. Protocolos de Interacción FIPA

En este apéndice describiremos brevemente los protocolos de interacción definidos en el estándar FIPA, los cuales determinan las conversaciones que se pueden establecer entre los diversos agentes. Un protocolo de interacción está compuesto por varios actos de comunicación.

E.1. Protocolo de Interacción *FIPA Propose*

El iniciador (agente que comienza el protocolo) manda un mensaje `propose` al participante (agente receptor) indicando que ejecutará una acción si el participante acepta. El participante responde aceptando o rechazando la propuesta, a través de los actos comunicativos `accept-proposal` o `reject-proposal` respectivamente.

Este protocolo se muestra en la Figura E.1.

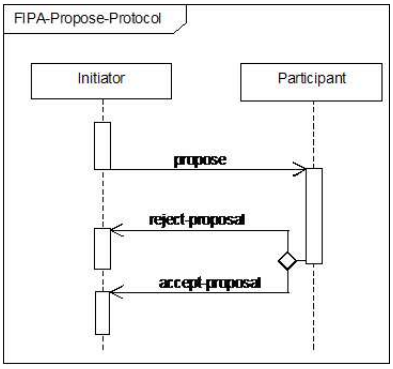

Figura E.1.: Protocolo de Interacción *FIPA Propose* [FIP02f].

E.2. Protocolo de Interacción *FIPA Request*

Este protocolo permite a un agente pedir a otro agente que lleve a cabo una acción [FIP02g].

El iniciador ejecuta la petición mediante el acto comunicativo `request`. El participante procesa el mensaje correspondiente y decide si aceptar o rechazar la petición. La respuesta tiene lugar mediante los actos comunicativos `agree` (opcional) y `refuse` respectivamente.

Una vez que el participante ha aceptado el `request` inicial, puede mandar los siguientes actos de comunicación al iniciador:

- `failure` si el intento de llevar a cabo la acción ha fracasado

- `inform-done` si ha conseguido llevar a cabo la acción y solamente quiere informar que se ha completado
- `inform-result` si ha completado la tarea y además quiere comunicar los resultados al iniciador

Este protocolo se muestra en detalle en la Figura E.2.

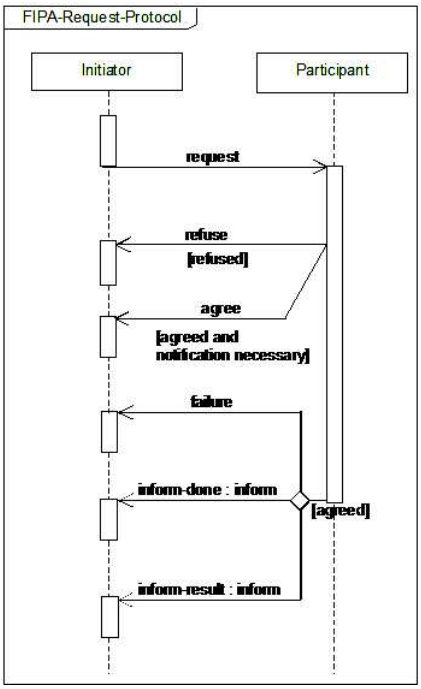

Figura E.2.: Protocolo de Interacción *FIPA Request* [FIP02g]

E.3. Protocolo de Interacción *FIPA Request When*

Este protocolo permite a un agente pedir a otro agente que lleve a cabo una acción, cuando se cumplan unas ciertas condiciones previas. Por tanto es una variación del protocolo de interacción FIPA Request.

El iniciador ejecuta la petición mediante el acto comunicativo `request when` (ver apéndice C). El participante procesa el mensaje correspondiente y decide si aceptar o rechazar la petición. La respuesta tiene lugar mediante los actos comunicativos `agree` (en este caso no es opcional) y `refuse` respectivamente.

La conversación termina cuando el participante manda un `failure` (no puede llevar a cabo la acción), `inform-done` (comunica que la acción ha sido completada) o `inform-result` (acción completada y resultados).

Este protocolo se muestra en detalle en la Figura E.3.

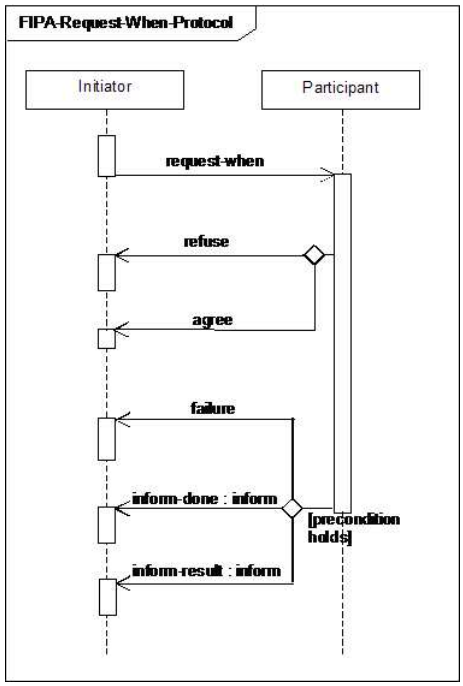

Figura E.3.: Protocolo de Interacción *Request When*

E.4. Protocolo de Interacción *FIPA Query*

Este protocolo permite a un agente llevar a cabo algún tipo de acción sobre otro agente [FIP01l].

El iniciador pide al participante que lleve a cabo alguna clase de acción *inform* usando para ello uno de estos actos de comunicación: `query-if` o `query-ref` (ver apéndice C). El acto `query-if` se emplea cuando el iniciador desea saber si una determinada proposición es cierta o no. Por otro lado, el `query-ref` se emplea cuando el iniciador desea consultar por algunos objetos.

En ambos casos, el participante procesa el acto de comunicación y decide si la acepta (`agree` opcional según las circunstancias) o la rechaza *(*`refuse`, en cuyo caso termina la interacción).

Si el participante es incapaz de realizar la consulta, devuelve al iniciador un acto de comunicación `failure`. En caso de que la consulta se haya realizado satisfactoriamente,

- En el caso de un `query-if`, el participante devuelve un `inform-t/f`, informando sobre la veracidad (*truth*) o falsedad (*falsehood*) de la proposición consultada.
- En el caso de un `query-ref`, el participante devuelve un `inform-result`, informando en su campo `content` de la expresión de los objetos cuya consulta fue especificada.

Este protocolo se muestra en detalle en la Figura E.4.

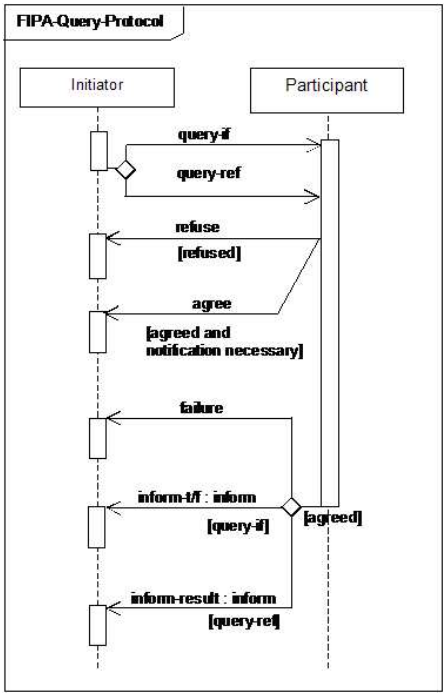

Figura E.4.: Protocolo de Interacción *FIPA Query* [FIP01l].

E.5. Protocolo de Interacción *FIPA Contract Net*

En este protocolo de interacción, un agente (iniciador) toma el papel de gestor que desea que otros agentes (los participantes) lleven a cabo una tarea y además desea optimizar una cierta función que caracteriza la tarea en cuestión (el precio asociado a la tarea). Para una tarea dada, cualquier número de participantes podrían responder con una propuesta, rechazando el resto la negociación. La negociación continúa con aquellos agentes que han realizado una propuesta [FIP01d].

El iniciador solicita *m* propuestas de otros tantos agentes mediante un acto de comunicación `cfp` (call for proposals), donde se especifica la tarea así como cualquier condición impuesta por el iniciador para llevar a cabo la negociación. Los participantes son vistos como potenciales contratistas y que son capaces de generar *n* respuestas. De éstas, *j* son propuestas para llevar a cabo la tarea a través de un acto de comunicación `propose`.

La propuesta del participante incluye las precondiciones de éste para llevar a cabo la tarea, como por ejemplo el precio, el tiempo en que estará lista,... De modo alternativo, los *i=n-j* participantes restantes deberían ejecutar un acto comunicativo `refuse`. Una vez que expire el plazo dado para obtener respuesta, el iniciador evalúa las ofertas recibidas y selecciona el agente o agentes a los que encargar la realización de la tarea. A los *l* agentes que son seleccionados se les envía un `accept-proposal`, mientras que a los *k=j-l* se les envía un `reject-proposal`. Las propuestas son de obligado

cumplimiento para el participante, por lo que una vez que el iniciador acepta la propuesta, el participante adquiere un compromiso para ejecutar la tarea. Una vez que el participante ha llevado a cabo la tarea, envía un mensaje al iniciador en la forma de un `inform-done` o en la versión más explicativa `inform-result`. Por el contrario, si no ha sido capaz de llevar a cabo la tarea, el participante envía un acto comunicativo `failure` al iniciador.

Obsérvese que este protocolo necesita que el iniciador sepa cuándo ha recibido todas las respuestas. En caso de que un participante no pueda responder al llamamiento inicial, el iniciador potencialmente podría quedar bloqueado esperando indefinidamente por las respuestas. Para evitarlo el `cfp` inicial incluye un plazo de tiempo para que los participantes puedan responder. Las propuestas recibidas después de este plazo son automáticamente rechazadas con el motivo de que ha llegado tarde. Este plazo es especificado por el parámetro `reply-by` en el mensaje ACL.

Este protocolo se muestra en la Figura E.5.

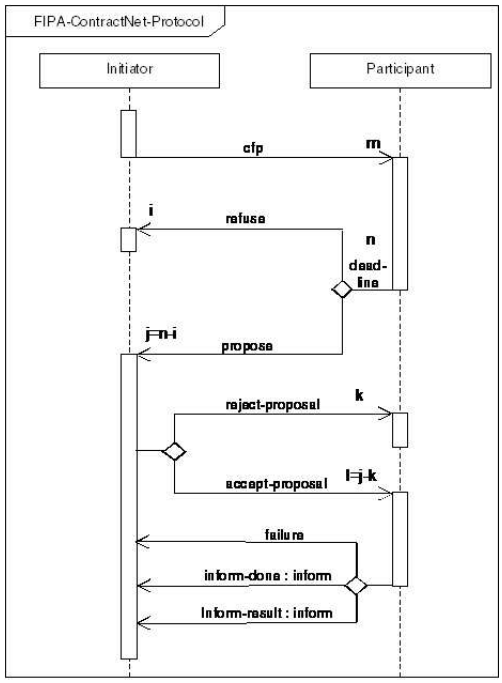

Figura E.5.: Protocolo de Interacción *FIPA Contract-Net* [FIP01d].

E.6. Protocolo de Interacción *FIPA Iterated Contract Net*

Este protocolo es una extensión del protocolo de iteracción FIPA Contract Net, que permite un ciclo de ofertas en varias rondas [FIP01g].

Como en el caso del protocolo de interacción *FIPA Contract Net*, el iniciador lanza *m* cfp iniciales. De los *n* participantes que responden, *k* son mensajes `propose` provenientes de los participantes que quieren y son capaces de llevar a cabo la tarea

en cuestión bajo las condiciones especificadas, mientras que las restantes *j* respuestas consisten en `refuse`.

De estas *k* propuestas recibidas, el iniciador podría decidir que se encuentra en la iteración final del proceso y aceptar *p* de las propuestas recibidas, rechazando el resto. De modo alternativo, el iniciador podría decidir iterar el proceso, lanzando un `cfp` revisado a *l* de los participantes y rechazando mediante un `reject-proposal` a los restantes. El motivo consiste en que el iniciador busca obtener mejores ofertas modificando el `cfp` inicial. El proceso termina cuando el iniciador rechaza todas las ofertas y no manda un nuevo `cfp`, el iniciador acepta una o más ofertas o todos los participantes rechazan la negociación.

Este protocolo se muestra en la Figura E.6.

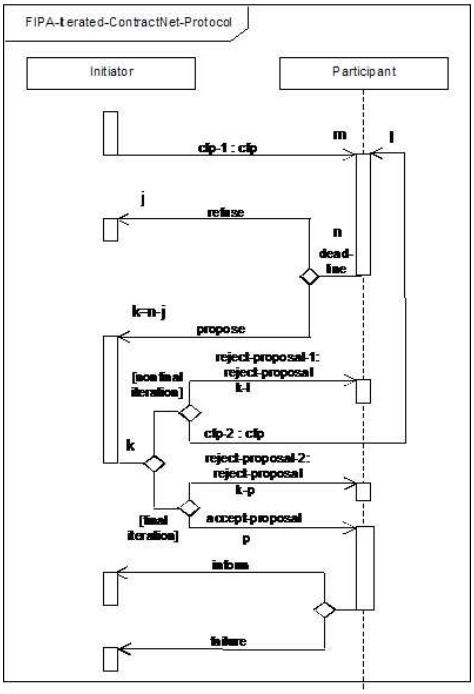

Figura E.6.: Protocolo de Interacción *FIPA Iterated Contract-Net* [FIP01g].

E.7. Protocolo de Interacción *FIPA English Auction*

Este protocolo trata de implementar un sistema de subasta a la inglesa. El iniciador trata de encontrar el precio de mercado de una mercancía proponiendo un precio inferior al que se supone de su valor de mercado, para después ir subiendo progresivamente ese precio. Cada vez que se anuncia el nuevo precio, el iniciador espera a ver si alguno de los posibles compradores (participantes) está dispuesto a pagar ese nuevo precio. Tan pronto como algún comprador anuncia esta disposición, el iniciador lanza una nueva oferta con un precio incrementado. Este proceso de subasta termina cuando ningún comprador acepta el nuevo precio. Si el último

precio aceptado por un comprador supera al precio esperado por el iniciador (este precio es desconocido para los participantes), la mercancía es vendida al participante correspondiente. En caso contrario, la mercancía no es vendida [FIP01f].

Inicialmente, el iniciador lanza su oferta inicial a todos los participantes mediante un `cfp`. Obsérvese que cuando se recibe una respuesta de uno de los participantes, el resto de participantes no tienen por qué enterarse de esa respuesta. Lo que si debe ocurrir es que cada participante que realiza una oferta tiene que conocer si su oferta ha sido aceptada o no, mediante la recepción respectivamente de los actos de comunicación `accept-proposal` y `reject-proposal`. Para finalizar el protocolo, de manera típica, el subastador introduce un protocolo FIPA Request con el postor ganador con el fin de completar la transacción objeto de la subasta.

El protocolo se puede ver de un modo más detallado en la Figura E.7.

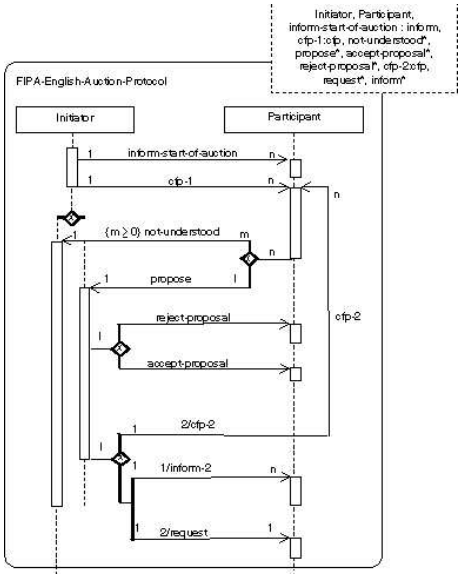

Figura E.7.: Protocolo de Interacción *FIPA English Auction* [FIP01e].

E.8. Protocolo de Interacción *FIPA Dutch Auction*

Este protocolo trata de implementar un sistema de subasta a la holandesa[1] [FIP01e]. En contraste con el protocolo de *FIPA English Auction* (sección E.7), el iniciador trata de encontrar el precio de mercado de una mercancía proponiendo un precio muy superior al que se supone su valor de mercado, para después ir reduciendo progresivamente ese precio, hasta que uno de los participantes en la subasta acepta el precio. Esa reducción progresiva depende del iniciador, el cual suele tener un precio mínimo de venta, oculto para los participantes en la subasta.

El protocolo se puede ver de un modo más detallado en la Figura E.8.

[1] El término deriva del proceso de venta en los mercados de flores en Holanda.

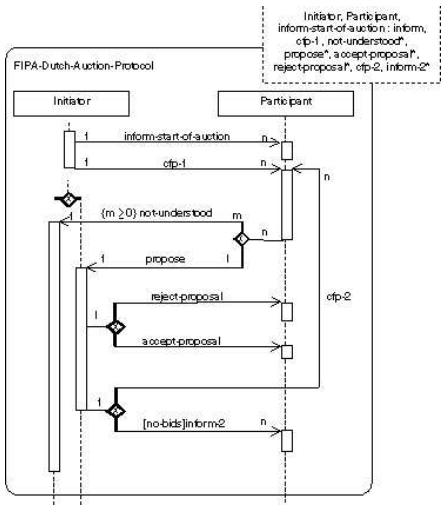

Figura E.8.: Protocolo de Interacción *FIPA Dutch Auction* [FIP01e].

E.9. Protocolo de Interacción *FIPA Brokering*

Este protocolo está diseñado para soportar las interacciones de intermediación (*brokerage*) en la mediación en sistemas multiagentes [FIP01c].

En general, un *broker* es un agente que ofrece un conjunto de servicios de comunicación a otros agentes empleando para ello algún tipo de conocimiento acerca de las peticiones y capacidades de esos agentes. Un ejemplo típico de intermediación es aquél en que un agente pide a un *broker* encontrar a uno o más agentes que le puedan responder a un acto de comunicación `query`. El broker determina un conjunto de agentes adecuados para la consulta, establece un protocolo de interacción `query` (ver sección E.4) y transmite las respuestas recibidas al iniciador. El uso del intermediación puede simplificar de modo significativo la interacción de los agentes en un MAS. Además, los *brokers* también permiten a un sistema ser robusto y adaptarse ante situaciones dinámicas, incluyendo los aspectos de seguridad en estos agentes.

El protocolo comienza cuando el iniciador manda un mensaje referente a un acto de comunicación `proxy` (ver apéndice C) al *broker*. El iniciador embebe dentro del `proxy` otro mensaje con el acto comunicativo deseado. El agente *broker* procesa la petición y decide si la acepta (`agree`) o la rechaza *(*`refuse`, en cuyo caso termina la interacción).

En caso de aceptarla, el reclutador intenta localizar los agentes cuya descripción concuerde con la expresión referencial del `proxy` inicial. Si no es posible encontrar a ningún agente, el reclutador enviará un `failure-no-match` al iniciador, lo que supone el fin de la interacción. En caso contrario, el reclutador podría modificar la lista de los agentes encontrados basándose en el parámetro de condición del `proxy`. Entonces comienza *m* interacciones con la lista resultante de *n* agentes, con cada interacción en sub-protocolo por separado. En este punto, el *broker* debería recordar algunos de los parámetros del `proxy` inicial, por ejemplo el `conversation-id`, `reply-with` y `sender`, con fines de enviar al iniciador las *r* respuestas obtenidas en el proceso.

Obsérvese que la naturaleza del sub-protocolo y de las respuestas dependen de los

protocolos de interacción especificados en el mensaje embebido en el `proxy` original. A medida que el sub-protocolo tiene lugar, el *broker* reenvía los mensajes recibidos de los participantes al iniciador.

El protocolo se puede ver de un modo más detallado en la Figura E.9.

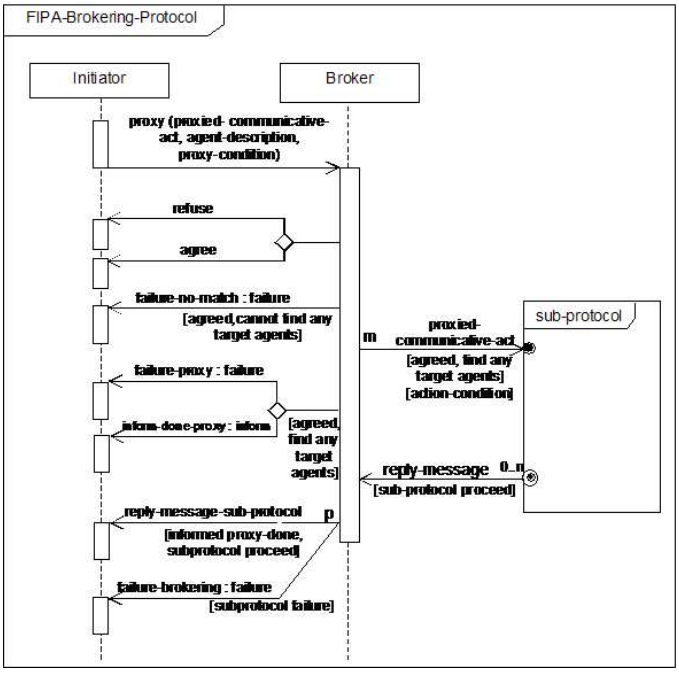

Figura E.9.: Protocolo de Interacción *FIPA Brokering* [FIP01c]

E.10. Protocolo de Interacción *FIPA Recruiting*

Este protocolo está diseñado para soportar las interacciones de reclutamiento (*recruiting*) en la mediación en sistemas multiagentes.

Un agente *reclutador* es una especie de *broker* (ver sección E.9), pero con la diferencia que los agentes seleccionados por el reclutador se dirigen directamente al agente iniciador o a algún receptor previamente designado [FIP01m].

Del mismo modo que con el *broker*, el empleo de agentes reclutadores puede simplificar de modo significativo la interacción de los agentes en un MAS, así como dotar al sistema de robustez y adaptabilidad ante situaciones dinámicas.

El protocolo comienza cuando el iniciador manda un mensaje referente a un acto de comunicación `proxy` (ver apéndice C) al reclutador. Este mensaje contiene una expresión referencial indicando los agentes a los que el reclutador debiera enviar el mensaje embebido, el acto comunicativo a enviar y una serie de parámetros como por ejemplo el número máximo de agentes a los que reenviar el mensaje. El reclutador procesa la petición y decide si la acepta (`agree`) o la rechaza (`refuse`, en cuyo caso termina la interacción).

En caso de aceptarla, el reclutador intenta localizar los agentes cuya descripción

concuerde con la expresión referencial del `proxy` inicial. Si no es posible encontrar a ningún agente, el reclutador enviará un `failure-no-match` al iniciador, lo que supone el fin de la interacción. En caso contrario, el reclutador podría modificar la lista de los agentes encontrados basándose en el parámetro de condición del `proxy`. Entonces comienzan *m* interacciones con la lista resultante de *n* agentes, con cada interacción en un sub-protocolo por separado. El inicio del protocolo debe realizarse con cuidado, usando los parámetros para manejar las respuestas de la petición. Así, si el mensaje que ha llegado al reclutador tiene un parámetro `designated-receiver`, necesita empezar cada sub-protocolo con un parámetro `reply-to` conteniendo al receptor designado, así como el identificador de conversación `conversation-id` de la conversación original. Es posible que sea necesario propagar otros parámetros.

Obsérvese que la naturaleza del sub-protocolo y de las respuestas dependen de los protocolos de interacción especificados en el mensaje embebido en el `proxy` original.

El protocolo se puede ver de un modo más detallado en la Figura E.10.

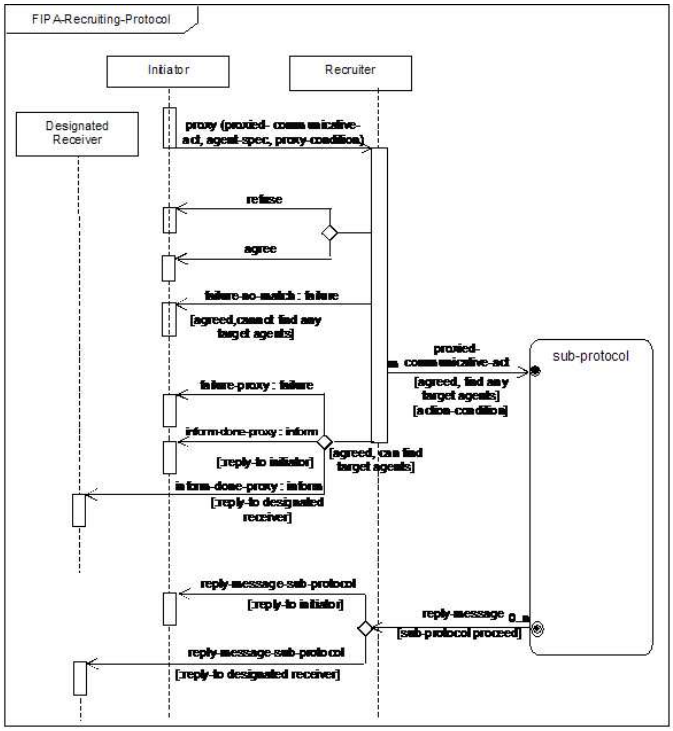

Figura E.10.: Protocolo de Interacción *FIPA Recruiting* [FIP01m].

E.11. Protocolo de Interacción *FIPA Subscribe*

Este protocolo permite a un agente pedir a otro agente llevar a cabo una acción de suscripción (*subscribe*) [FIP02h].

El iniciador comienza la interacción con acto de comunicación `subscribe` conteniendo la referencia de los objetos en los que está interesado. El participante

procesa la petición y decide si la acepta (`agree` opcional según las circunstancias) o la rechaza *(*`refuse`, en cuyo caso termina la interacción).

Tras aceptarla, el participante responde con un `inform-result` cuyo campo `content` corresponde a una expresión referencial de los objetos de la suscripción. El participante continúa enviando un `inform-result` cada vez que se produce algún cambio en dichos objetos. En caso de que en algún momento se produjese algún problema, se envía un `failure` al iniciador, finalizando el protocolo.

El protocolo se puede ver de un modo más detallado en la Figura E.11.

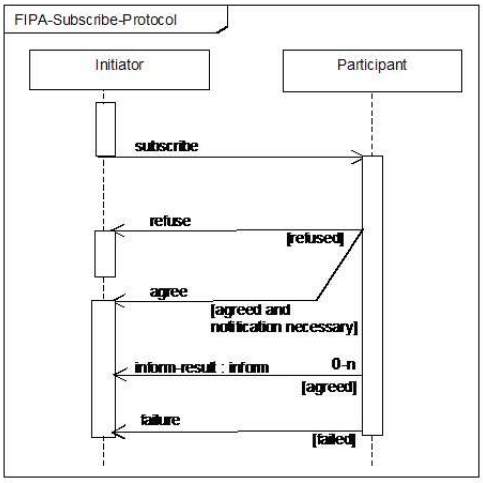

Figura E.11.: Protocolo de Interacción *Subscribe* [FIP02h].

F. Requisitos obligatorios FIPA para una plataforma de agentes

En este apéndice enunciamos los requisitos obligatorios impuestos por FIPA para que una plataforma de agentes sea compatible con su estándar [Lau00b, Lau00a].

F.1. Requisitos de Interoperabilidad

1. IIOP debe ser usado como protocolo base.
2. El protocolo base debe definir una interfaz IDL llamado FIPA_Agent_97 que contenga un método llamado "mensaje", que toma como parámetro de entrada una cadena de caracteres CORBA.
3. Se debe emplear ACL en la comunicación entre plataformas.
4. La plataforma debe soportar direcciones en el formato URL.

F.2. Requisitos del agente FIPA

1. Un agente FIPA debe ser capaz de comunicarse con otros agentes FIPA.
2. Un agente FIPA debe tener una identidad única.
3. Un agente FIPA tiene una o varias direcciones.
4. Un agente FIPA debe tener una plataforma principal, la cual es estática.
5. El nombre asociado al agente FIPA se asocia al mismo cuando el agente es creado o cuando se registra con un AMS la primera vez.
6. La dirección del agente FIPA no debe emplearse como nombre del agente.
7. Un agente FIPA debe tener uno o más propietarios.

F.3. Requisitos sobre ACL y mensajes

1. Un agente FIPA debe ser capaz de enviar un acto de comunicación `not-understood`, si el mensaje entrante está mal formado ya sea sintáctica o semánticamente, o no es soportado por el agente.

F.4. Requisitos sobre el AMS

1. Un AMS controla y administra los ciclos de vida de los agentes FIPA registrados en él.
2. Un agente FIPA debe estar registrado con un AMS, a fines de interactuar con los agentes de la misma plataforma o residentes en otra plataforma.
3. AMS mantiene un índice de los agentes registrados en él (a modo de *páginas blancas* de la plataforma)
4. Solamente el AMS administra la plataforma.

F.5. Requisitos sobre el ACC

1. Un ACC debe soportar IIOP.
2. Un ACC debe ser capaz de escoger el protocolo de transporte correcto de acuerdo con el formato de la dirección.
3. Un ACC debe tener cuidado de comprobar la sintaxis del mensaje entrante antes de redistribuirlo.
4. Los agentes no necesitan ser advertidos sobre los detalles del transporte de los mensajes.
5. Un ACC debe soportar transporte de mensajes hacia un único receptor.
6. Todos los mensajes deben ser enviados mediante un ACC local o remoto.
7. Un ACC debe soportar tanto la comunicación dentro de la misma plataforma como entre diferentes plataformas.
8. Solamente los mensajes dirigidos a agentes son enviados a través del ACC.
9. Un ACC no necesita ser un agente. En realidad, el ACC solamente proporciona servicios de transporte de mensajes,
10. En el direccionamiento de los mensajes, el ACC no necesita acceder al contenido de los mensajes.

F.6. Requisitos sobre el DF

1. Todas las plataformas deben incluir al menos un DF.
2. Si el DF es accedido de forma remota (por ejemplo desde otra plataforma,), entonces el DF debe proveer una interfaz ACL.

G. Requisitos opcionales FIPA para una plataforma de agentes

En este apéndice enunciamos los requisitos opcionales por parte de FIPA para una plataforma de agentes [Lau00b, Lau00a].

G.1. Requisitos de Interoperabilidad

1. Otros protocolos aparte del IIOP podrían estar soportados, aunque el contacto inicial debe realizarse a través de IIOP.

2. Un agente podría soportar IIOP por sí mismo.

3. Se podría dar soporte a direcciones en forma de IOR.

G.2. Requisitos del agente FIPA

1. Un agente podría estar registrado en una plataforma (es decir, ser residente) o ser no-residente.

2. Un agente podría tener una plataforma exterior, la cual cambia cuando el agente se desplaza.

G.3. Requisitos sobre ACL y mensajes

1. El SL podría estar soportado como lenguaje de contenido.

2. El XML podría estar soportado como lenguaje de contenido.

G.4. Requisitos sobre Movilidad

1. Un agente FIPA podría trasladarse de un lugar a otro,

G.5. Requisitos sobre el AMS

1. Una plataforma podría entenderse por varias máquinas.

2. Un AMS podría anunciarse a sí mismo al DF.

G.6. Requisitos sobre el ACC

1. Un ACC podría soportar otro protocolo aparte del IIOP.
2. Un ACC podría anunciar sus transportes al DF.

G.7. Requisitos sobre el DF

1. Un DF puede ser distribuible.
2. Un DF podría registrarse él mismo con otros DFs.
3. Los servicios ofrecidos por un DF podrían restringirse siguiendo algún criterio.

H. Introducción a las Redes Neuronales

La idea de las Redes Neuronales Artificiales surge del intento de imitar el funcionamiento de las redes neuronales biológicas. De hecho, el avance de la neurofisiología y la neuroanatomía en la década de los 40, así como el descubrimiento de los modelos de aprendizaje llevaron a Donald Hebb a proponer en 1949 el punto de partida de las leyes de entrenamiento de las Redes Neuronales Artificiales. Tras un periodo de descrédito durante los años 60 y 70, en las últimas tres décadas se ha registrado un resurgir de esta herramienta como técnica general de resolución de un amplio rango de problemas.

Las redes neuronales se pueden definir como sistemas computacionales, tanto hardware como software, que tratan de imitar las habilidades de los sistemas biológicos usando un gran número de neuronas artificiales interconectadas. Las neuronas artificiales, por su parte, son emulaciones sencillas de las neuronas biológicas. Así, toman información desde sensores o desde otras neuronas artificiales, realizan una sencilla operación con estos datos, pasando el resultado a otras neuronas artificiales. Como se puede observar, esta estructura presenta una clara analogía con la anatomía cerebral. Sin embargo, y más allá de esta semejanza superficial, el éxito de las redes neuronales consiste en la exhibición de un sorprendente número de características cerebrales. Entre otras, se pueden citar:

- Capacidad de Aprendizaje: Presentan la habilidad de ser adaptables frente a un conjunto de ejemplos (entradas y quizás sus correspondientes salidas deseadas) a través de un proceso de aprendizaje o entrenamiento. En este apartado se han desarrollado una gran cantidad de algoritmos de entrenamiento.
- Capacidad de Representación: Algunos tipos de redes, como el Perceptrón Multicapa (PMC, o en su nomenclatura inglesa, *multilayer perceptron,* MLP), son capaces de representar cualquier función con la precisión deseada.
- Capacidad de Generalización: Una vez entrenada, la respuesta de la red puede ser insensible a las variaciones de la entrada. Esta habilidad de ver los patrones subyacentes, a pesar de posibles ruidos y distorsiones, es de vital importancia para el reconocimiento de los mismos en un entorno real.
- Capacidad de Procesamiento en Paralelo: De manera análoga que en las redes neuronales biológicas, las artificiales se constituyen por elementos simples que pueden funcionar de manera paralela.
- Tolerancia a fallos: La información está distribuida en toda la red, por lo que al producirse un fallo en un elemento individual, el comportamiento global se degrada únicamente de forma ligera.

Con estas características, las redes neuronales ofrecen para ciertos problemas un aumento del rendimiento con respecto a las técnicas convencionales en áreas como:

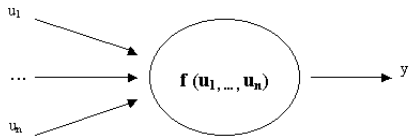

Figura H.1.: Representación de un perceptrón.

- Reconocimiento de patrones.
- Filtrado de señales.
- Clasificación de datos.
- Compresión de datos.
- Control Adaptativo.
- Optimización y Control Óptimo.

H.1. Representación de una neurona artificial

Tal como se ha indicado, una neurona artificial puede ser representada como un sistema con una serie de entradas (proveniente de estímulos externos a la red o de otras neuronas de la red) y que aplica a estas entradas una función que proporciona la salida de la neurona (que puede ir a otras neuronas o constituir la salida de la red) [Gon00].

Así se puede considerar la neurona artificial más común (*perceptrón*, ver Figura H.1) como una función con *n* entradas y una única salida. Esta función se puede dividir en dos operaciones consecutivas: una suma ponderada de las entradas a través de unos coeficientes (*pesos*), y la aplicación a esta suma de una función (*función de activación*). Suele existir asimismo una entrada cuyo valor se encuentra fijado a la unidad y cuyo peso recibe el nombre de *umbral*, *sesgo* o *bias*.

De acuerdo con lo visto, se puede resumir la acción de un perceptrón mediante las siguientes dos ecuaciones:

$$a = \sum_{i=1}^{n} w_i u_i + w_0 \tag{H.1}$$

$$y = f(a) \tag{H.2}$$

donde

- n es el número de entradas a la neurona.
- u_i es la entrada i-ésima.
- w_i es el peso i-ésimo.

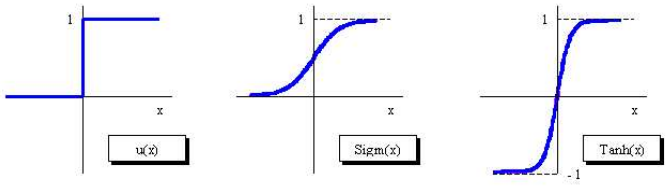

Figura H.2.: Funciones de activación más empleadas en neuronas artificiales: función umbral lógica (izq,), sigmoide (centro) y tangente hiperbólica (der.).

- w_0 es el peso umbral.
- *f(x)* es la función de activación.
- *y* es la salida de la neurona.

Entre las funciones de activación más utilizadas se encuentran:

- *Función Umbral Lógica*: La forma más sencilla de definir la activación de una neurona es considerarla binaria, es decir una salida que únicamente adquiere valores 0 ó 1. En el caso de que la suma ponderada de las entradas sea menor que el umbral de la neurona, la salida es 1; si es menor es 0 (Figura H.2, izq.) . Expresado de modo matemático:

$$f(x) = u(x) = \begin{cases} 1 & Six \geq -w_0 \\ 0 & Six < -w_0 \end{cases} \tag{H.3}$$

- *Función Sigmoide* (Figura H.2, centro):

$$Sigm(x) = \frac{1}{1+e^{-x}} \tag{H.4}$$

- *Tangente Hiperbólica* (Figura H.2, der.):

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{H.5}$$

Las ventajas de las dos últimas respecto a la primera estriban en que son continuas y además infinitamente derivables, con lo que se pueden aplicar métodos de entrenamiento que involucren derivadas. Además, en ambas existe un continuo en la salida, no solamente dos únicos valores, como en el caso de la función umbral lógica.

Aún cuando son las más utilizadas, estas funciones no son, ni mucho menos, las únicas. Aparte de las descritas, existen multitud de funciones de activación que intentan describir el comportamiento de una neurona biológica. Como ejemplo se citan las siguientes:

- *Función saturación*: Se definen límites inferior y superior. Si la suma ponderada es menor que el límite inferior, la salida es 0 (ó -1 según el caso). Si la suma es mayor que el límite superior, la salida (activación) de la neurona es 1.

Finalmente, si se encuentra entre ambos límites, la activación se comporta según una función lineal de la suma de las entradas. De modo analítico:

$$f(x) = \begin{cases} 0 & Si\, x \leq -\alpha \\ \frac{x}{2\alpha} & Si\, -\alpha < x < \alpha \\ 1 & Si\, x \geq -\alpha \end{cases} \tag{H.6}$$

donde -α y α corresponden respectivamente a los límites inferior y superior.

- *Funciones de base radial*: De entre ellas la más utilizada es la función Gaussiana. Es útil cuando la red neuronal se utiliza para reproducir funciones continuas. Se pueden variar la media y la desviación, lo que la hace más adaptable que la función sigmoide.

H.2. Topologías de las Redes Neuronales

Las formas en que se interconectan las neuronas artificiales definen los diferentes tipos de arquitecturas. Una primera distinción se basa en su número de capas. Acorde a este criterio, se tienen redes neuronales monocapas, bicapas o multicapas.

Según el tipo de conexiones, las redes monocapas se dividen a su vez en redes lateralmente conectadas (conexiones entre las neuronas de la capa), y aquéllas que presentan un único orden lógico. Las redes bicapa típicamente tienen conexiones tanto hacia delante como hacia atrás, pero usualmente carecen de conexiones laterales.

Finalmente, en las redes multicapas existe una gran variedad de tipos que poseen únicamente conexiones hacia delante, así como otro gran número con conexiones complejas (hacia delante, hacia atrás y laterales).

Con todo lo anterior, se pueden identificar cinco estructuras diferentes de redes neuronales:

Redes neuronales multicapas de alimentación hacia delante Las señales se propagan hacia delante a través de las capas. Por tanto, no existen autoconexiones, conexiones laterales o hacia atrás. Un ejemplo de este tipo de red es el perceptrón multicapa. Constituyen el grupo más utilizado y se suelen aplicarse en labores reconocimiento de patrones, imitación de funciones y en control de sistemas.

Redes monocapas lateralmente conectadas Esta red sólo tiene una capa, por lo que solamente se puede activar un patrón cada vez. No obstante, las conexiones laterales, y a veces recurrentes, causan que en cada ciclo de operación aparezcan diferentes patrones. Su uso principal son los patrones de asociación.

Redes monocapas topográficamente ordenadas Estas redes no tienen conexiones explícitas. Durante el entrenamiento, una medida de la distancia vectorial entre diferentes vectores de neuronas se usa para ajustar su posición relativa en el espacio vectorial. Entre estas redes se encuentran las LVQ y las TPM desarrolladas por Kohonen.

Redes asociativas/resonantes Redes bicapas en las cuales la información pasa tanto hacia delante como atrás. Se presentan patrones a la entrada y la red evoluciona hasta llegar al equilibrio. Este tipo es particularmente adecuado para la asociación de patrones en la primera capa con otros en la segunda capa (heteroasociación de patrones). Ejemplos de este tipo son las ART de Carpenter y las BAM de Kosko.

Redes multicapas cooperativas/competitivas Algunas redes, dentro de la categoría de aquellas redes con alimentación hacia delante y alimentación hacia atrás, tienen conexiones laterales. Las conexiones se diseñan específicamente de manera que las positivas (cooperativas) compensen a las negativas (competitivas). Este diseño se ha realizado para imitar ciertas redes biológicas.

Otra división posible de las redes neuronales es entre redes neuronales estáticas y dinámicas. Las redes neuronales estáticas son aquellas cuyas salidas dependen de las entradas actuales, sin dependencia alguna de las entradas y salidas previas. En contraposición, las redes dinámicas se caracterizan por una dependencia de las entradas y/o salidas previas. Este último tipo ha sido aplicado con cierto éxito en problemas de modelización de la dinámica directa e inversa de sistemas complejos (robots, naves espaciales...) [Mar99].

H.3. Entrenamiento de las Redes Neuronales

Por entrenamiento de una Red Neuronal se entiende el proceso de ajuste de los pesos según unos criterios determinados por el algoritmo de aprendizaje escogido.

Se pueden dividir los algoritmos de entrenamiento en dos grandes grupos: entrenamiento supervisado y entrenamiento no supervisado.

Entrenamiento Supervisado En los métodos supervisados, un maestro empareja cada vector de entradas (conjunto de entradas) con un vector de salida deseado, formando los denominados *pares de entrenamiento*. Para el entrenamiento se toma un cierto número de estos pares. El proceso consiste en presentar a la entrada unos vectores de entrada e intentar que la salida reproduzca los patrones deseados, mediante la modificación de los pesos según el algoritmo escogido. Este algoritmo tiende a minimizar el error (diferencia entre los valores deseados y los obtenidos), repitiendo el proceso hasta que el error, para todo el conjunto de entrenamiento, sea aceptablemente bajo.

Entrenamiento No Supervisado El entrenamiento no supervisado se caracteriza porque no se dispone de los valores deseados de la salida, por tanto el conjunto de entrenamiento consta solamente de los vectores de entrada. El algoritmo modifica los pesos de la red para que las salidas de la red sean consistentes, esto es, dos aplicaciones del mismo vector de entrenamiento, o de dos vectores lo suficientemente similares, debe producir el mismo vector de salida. El proceso de entrenamiento extrae propiedades estadísticas del conjunto de entrenamiento y agrupa vectores similares dentro de clases.

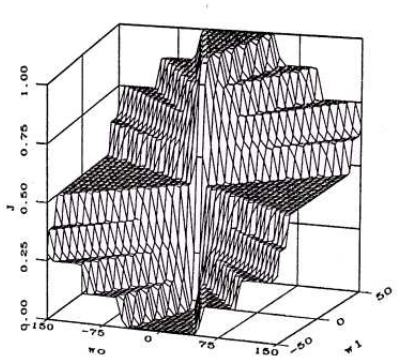

Figura H.3.: Ejemplo de superficie generada por la función de coste para una red neuronal con dos pesos.

H.4. Métodos de Entrenamiento de Redes Neuronales

En este apartado se mostrarán los diversos algoritmos de optimización incluidos en la herramienta de cálculo *Evenet2000*.

El aprendizaje de una red neuronal se resume en encontrar el conjunto de pesos w que hace mínima una función denominada *de coste*. Ésta indica lo alejado o lo cercano, cuantitativamente, que está la salida de la red respecto a la salida deseada. Normalmente se suele elegir del tipo cuadrática con el error:

$$J(w) = \frac{1}{2} \sum_{p=1}^{n_p} \sum_{j=1}^{N_L} (d_j(p) - y_j(p))^2 \tag{H.7}$$

donde:

n_p = Número de patrones a entrenar.

$d_j(p)$ = Salida deseada j correspondiente al patrón p.

$y_j(p)$ = Salida de la red j correspondiente al patrón p.

N_L = Número de salidas de la red.

Gráficamente, dicha función representará una hipersuperficie. Ésta puede llegar a ser muy compleja, incluso en el caso de tener tan sólo dos pesos tal como se puede observar en la Figura H.3, por lo que la determinación del mínimo puede no resultar sencilla.

Los algoritmos a utilizar son aquéllos de optimización multidimensional sin ligaduras. De modo formal, el significado de la optimización sin ligaduras consiste en buscar un vector $X=(x_1,x_2,\cdots,x_n)$ que cumpla lo siguiente:

$$\frac{\partial f}{\partial x_i} = 0 \ \forall i = 1,2,\cdots,n \tag{H.8}$$

siendo f(X) la función que queremos optimizar y X un punto sin ningún tipo de ligadura.

En el caso de buscar un mínimo, hay que garantizar que la matriz Hessiana esté definida positiva, o lo que es lo mismo.

$$J_x = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right] = definida\, positiva \tag{H.9}$$

Las dos ecuaciones H.8 y H.9 están condicionadas a que la función tenga definidas las derivadas parciales primeras y segundas, al menos en el mínimo. En caso de no estar definida la derivada segunda o primera, el objetivo será encontrar un punto X tal que el valor de f(X) sea mínimo/máximo.

De importancia es el llamado *problema del mínimo local* consistente en que el método converja a una solución que sea el mínimo en un dominio entorno a ese punto (*mínimo local*), pero que fuera de ese entorno existan otros puntos con un menor valor de la función. Por ello, que estos métodos alcancen la condición de parada no garantiza que se esté en el punto donde la función es la mínima posible.

Existen muchos métodos de optimización, aunque todos estos se pueden dividir en dos categorías generales que son los métodos de búsqueda directa y los métodos descendentes.

Los métodos directos se distinguen de los métodos descendentes porque sólo requieren la evaluación de la función, ya que no se utiliza el conocimiento de las derivadas. Estos métodos se suelen utilizar para problemas simples que requieren un número de dimensiones relativamente pequeños o que no tienen disponibles sus derivadas. Son menos eficientes que los métodos descendentes, aunque, por otro lado, no suelen presentar el problema del mínimo local. La búsqueda de Camino Aleatorio, el Enfriamiento Progresivo y los Algoritmos Genéticos (codificados en la herramienta de cálculo Evenet2000) se sitúan en este tipo.

En cuanto a los descendentes, éstos requieren de las derivadas de primer orden y en ocasiones de mayores órdenes, además de la evaluación de la función. Por tanto estos métodos utilizan más información que los de búsqueda directa, lo que justifica su mayor eficiencia. Los métodos descendentes son conocidos también como métodos de gradiente. En este grupo se engloban los algoritmos de gradiente descendente y gradiente conjugado.

Al haberse implementado algunos métodos que precisan del cálculo del gradiente, antes de detallar cada uno de ellos, se explicarán diversos modos de cálculo del gradiente de una red neuronal.

H.4.1. BackPropagation y Real Time Recurrent Learning

La BackPropagation (BP) es la forma de obtener las derivadas que constituyen el gradiente de la función de error con respecto a los pesos para el caso de una red multicapa de alimentación hacia adelante totalmente interconectada. Este tipo de red suele recibir el nombre de perceptrón multicapa (PMC ó MLP) [HH93].

La notación que se utilizará a partir de este momento, es la siguiente (Figura H.4):

- u_{lj} salida del nodo j en la capa l.
- w_{lj_i} peso que conecta el nodo i en la capa l-1 al nodo j de la capa l.
- w_{lj_0} bias del nodo j de la capa l.
- u_{0i} componente i del vector de entrada.

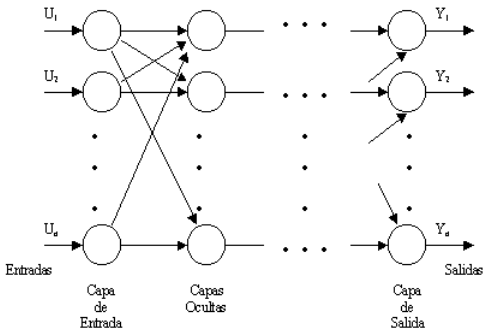

Figura H.4.: Perceptrón Multicapa.

- $d_j(x_p)$ la respuesta deseada del nodo j de salida para el patrón de entrenamiento.
- n_l número de nodos en la capa l.
- L número de capas.
- p el índice del patrón.
- n_p el número de patrones.

La capa 0 de la red representa a las entradas. Teniendo esto en cuenta, la salida de las neuronas de cada capa puede calcularse como:

$$u_{lj} = f\left(\sum_{i=0}^{n_{l-1}} w_{ljl} u_{l-1\,i}\right) \tag{H.10}$$

donde f es la función de activación, que suele ser una sigmoide (ecuación H.4), cuya derivada se puede expresar de la siguiente manera:

$$f'(\alpha) = \frac{df(\alpha)}{d\alpha} = f(\alpha)(1 - f(\alpha)) \tag{H.11}$$

Tal como se ha indicado, la optimización consiste en minimizar la función de coste:

$$J(w) = \sum_{p=1}^{n_p} J_p(w) \tag{H.12}$$

donde J_p es la función de coste de cada patrón. En el caso de ser cuadrática, viene definida de la siguiente manera:

$$J_p(w) = \frac{1}{2} \sum_{q=1}^{n_L} \left(u_{lq}(p) - d_q(p)\right)^2 \tag{H.13}$$

Como se puede observar, sólo se tiene en cuenta en la función la salida deseada en la última capa.

Haciendo uso de la regla de la cadena:

$$\frac{\partial J_p(w)}{\partial w_{ljji}} = \frac{\partial J_p(w)}{\partial u_{lj}} \frac{\partial u_{lj}}{\partial w_{ljji}} \tag{H.14}$$

donde el segundo factor del producto se determina mediante la ecuación H.10 como:

$$\frac{\partial u_{lj}}{\partial w_{ljji}} = f^{'}\left(\sum_{m=0}^{n_{l-1}} w_{ljm} u_{l-1m}\right) u_{l-1i} \tag{H.15}$$

El primer término del lado derecho de la ecuación H.14 se denomina sensibilidad, ya que da una idea cualitativa de la variación de la función de coste respecto a la variación de la salida en cada nodo. Utilizando de nuevo la regla de la cadena se obtiene para su cálculo:

$$\begin{aligned} \frac{\partial J_p(w)}{\partial u_{lj}} &= \sum_{m=1}^{n_{l+1}} \frac{\partial J_p(w)}{\partial u_{l+1m}} \frac{\partial u_{l+11}}{\partial u_{lj}} = \\ & \sum_{m=1}^{n_{l+1}} \frac{\partial J_p(w)}{\partial u_{l+1m}} f^{'}\left(\sum_{k=0}^{n_l} w_{l+1mk} u_{lk}\right) w_{l+1mj} \end{aligned} \tag{H.16}$$

expresión que se puede particularizar en el caso de la función sigmoide haciendo uso de la ecuación H.11.

$$\frac{\partial J_p(w)}{\partial u_{lj}} = \sum_{m=1}^{n_{l+1}} \frac{\partial J_p(w)}{\partial u_{l+11}} u_{l+1m}\left(1 - u_{l+1m}\right) w_{l+1mj} \tag{H.17}$$

En un análisis la ecuación se observa que existe un proceso recursivo hacia atrás, ya que si se conocen las salidas de la capa *l+1*, y la sensibilidad de los nodos de dicha capa, se conocerá la sensibilidad de la capa anterior. De ahí que se busquen expresiones que relacionen la derivada de la función de activación con la salida de la neurona, del mismo modo que la ecuación H.11. Por otro lado, para la última capa, se tiene:

$$\frac{\partial J_p(w)}{\partial u_{Lj}} = u_{Lj}(p) - d_j(p) \tag{H.18}$$

Con esto, se puede plantear el proceso de determinación del gradiente de la siguiente forma. Se calcula la salida de cada capa en un proceso hacia delante a partir de la entrada de la red. A continuación se determina la sensibilidad de la última capa a través de la ecuación H.18. Mediante la ecuación recurrente H.16 se calcula el resto de sensibilidades. Por otro lado se calcula la derivadas de las salidas de las capas con respecto de los pesos con la ecuación H.15 y finalmente se determina el gradiente mediante la ecuación H.14.

En sí, el algoritmo de Backpropagation es aquél en que el cálculo del gradiente viene dado de esta manera. Este es precisamente el origen de su nombre, ya que existe una propagación hacia atrás en la forma de calcular el gradiente debido a que, cuando se hallan las sensibilidades de las diferentes capas, la información va de la última hacia la primera.

Este método de cálculo del gradiente, no se limita al tipo de red MLP, es posible

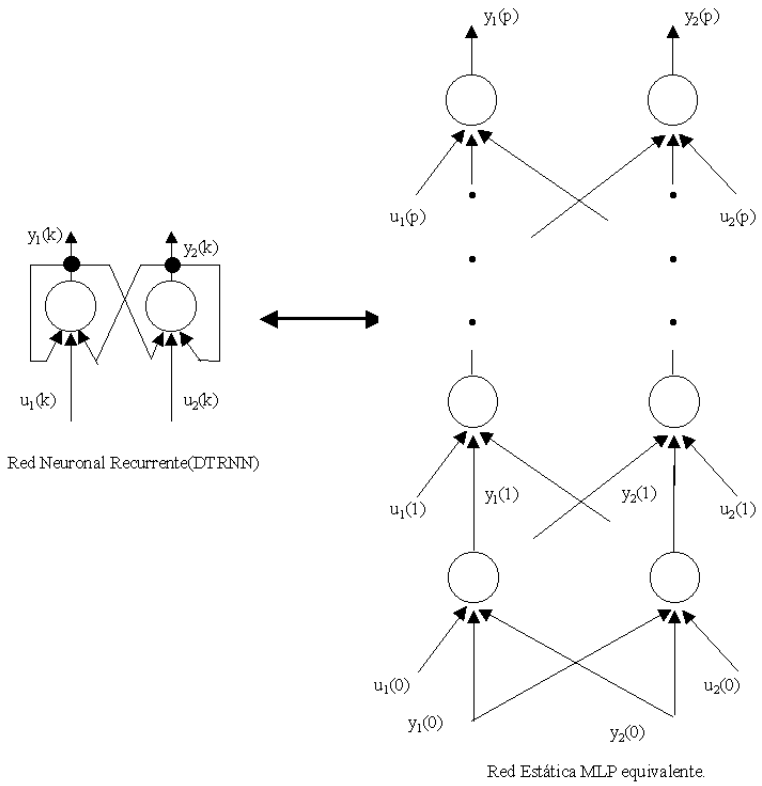

Figura H.5.: Equivalencia de una red DTRNN con un perceptrón multicapa.

hacer una generalización a las redes tipo DTRNN (redes neuronales recurrentes discretas). Un primer método es hacer una aplicación directa de la BP a la red estática equivalente (ver Figura H.5), en cuyo caso se tiene el método denominado *Backpropagation Through Time* (BPTT) [Wer90]. En este caso, el aprendizaje debe realizarse compartiendo los pesos, esto es, todos los pesos que en la red propagada ocupen posiciones equivalentes según la red original, deben tener simultáneamente el mismo valor.

Un segundo método surge de la aplicación a la red de la regla de la cadena de forma diferente a la del Backpropagation y se denomina aprendizaje recurrente en tiempo real (*Real Time Recurrent Learning*, RTRL).

Para su explicación y por conveniencia se empleará la notación indicada en la Figura H.6:

- X es el vector de entradas de la red.
- Y es el vector de salidas de la red.
- m es el número de entradas.
- n es el número de salidas.

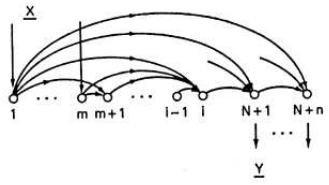

Figura H.6.: Red tipo DTRNN o Williams-Zipser.

- N es el número de entradas + número de neuronas descontando las de salida.
- w_{ji} es el peso que conecta el nodo i al nodo j.
- w_{j0} es el bias del nodo j.
- $u_j(k)$ es la salida de la neurona j en el instante k.

En la anterior figura hay que tener en cuenta que los primeros m nodos son simbólicos, es decir no le corresponde una neurona sino un punto de bifurcación. Por otro lado se puede observar que la salida de los últimos n nodos son las salidas de la red. Con todas estas consideraciones, la ecuación de salida de las neuronas se puede expresar como:

$$u_j(k) = f\left(\sum_{i=0}^{N+n} w_{ji} u_i(k-1)\right) \tag{H.19}$$

En este caso, la función de coste incluye el error cometido en cada salida para cada etapa de la evolución temporal y para todas las secuencias de entrada. Esta función de coste es:

$$J(w) = \sum_{p=1}^{n_p} J_p(w) \tag{H.20}$$

siendo J_p:

$$J_p(w) = \frac{1}{2} \sum_{k=1}^{K_p} \sum_{h=1}^{n} \left(d_h(k) - u_{h+N}(k)\right)^2 \tag{H.21}$$

donde K_p es la longitud de la secuencia de entrenamiento p-ésimo.
Las derivadas parciales se pueden poner de la siguiente forma:

$$\frac{\partial J_p(w)}{\partial w_{ji}} = -\sum_{k=1}^{K_p} \sum_{h=1}^{n} \left(d_h(k) - u_{h+N}(k)\right) p_{ji}^h(k) \tag{H.22}$$

donde:

$$p_{ji}^h(k) = \frac{\partial u_h(k)}{\partial w_{ji}} \tag{H.23}$$

Para hallar el gradiente es necesario determinar los coeficientes de esta ecuación.

Si se deriva u_h respecto a w_{ji} para determinar p^h_{ji}, se obtiene que:

$$\begin{aligned} p^h_{ji}(k) &= \frac{\partial u_h(k)}{\partial w_{ji}} \\ &= f'\left(\sum_{\beta=0}^{N+n} w_{h\beta}u_\beta(k-1)\right)\left[\sum_{\alpha=0}^{N+n}\left(\frac{\partial w_{h\alpha}}{\partial w_{ji}}u_\alpha(k-1)+w_{h\alpha}\frac{\partial u_\alpha(k-1)}{\partial w_{ji}}\right)\right] \\ &= f'\left(\sum_{\beta=0}^{N+n} w_{h\beta}u_\beta(k-1)\right)\left[\sum_{\alpha=0}^{N+n}\delta_{hj}\delta_{\alpha i}u_\alpha(k-1)+\sum_{\alpha=1}^{N+n} w_{h\alpha}\frac{\partial u_\alpha(k-1)}{\partial w_{ji}}\right] \end{aligned} \tag{H.24}$$

donde δ_{hj} es la función delta de Kronecker.

Si se tiene en cuenta que el conjunto de entradas (incluyendo la correspondiente al peso umbral) es constante respecto a los pesos, sus términos no contribuyen al último sumatorio, con lo que la ecuación H.25 queda:

$$\begin{aligned} p^h_{ji}(k) &= f'\left(\sum_{\beta=0}^{N+n} w_{h\beta}u_\beta(k-1)\right)\left[\delta_{hj}u_i(k-1)+\sum_{\alpha=1}^{N+n-m} w_{h\alpha+m}\frac{\partial u_{\alpha+m}(k-1)}{\partial w_{ji}}\right] \\ &= f'\left(\sum_{\beta=0}^{N+n} w_{h\beta}u_\beta(k-1)\right)\left[\delta_{hj}u_i(k-1)+\sum_{\alpha=1}^{N+n-m} w_{h\alpha+m}p^{\alpha+m}_{ji}(k-1)\right] \end{aligned} \tag{H.25}$$

Como se puede observar, se plantea una recurrencia hacia delante en la variable de etapa k. Esto hace que, a diferencia del backpropagation estático, deben fijarse condiciones iniciales arbitrarias para inicializar el proceso recursivo (generalmente se supone que los $p^h_{ji}(0) = 0$).

Existen varias diferencias entre los algoritmos de entrenamiento de las redes recurrentes BPTT y RTRL. El primero ofrece un coste computacional bastante mayor que el segundo pero este último presenta un requerimiento de espacio mayor. Otra diferencia fundamental es en la dirección de los procesos recursivos para determinar el gradiente de cada uno, hacia atrás en BPTT y hacia delante en RTRL. Esto confiere a este último método ventajas para su implementación en tiempo real.

H.4.2. Gradiente Descendente

Es conocido que la derivada en la dirección del gradiente es máxima con respecto al resto de derivadas direccionales en ese punto. Debido a esto, la dirección del gradiente se denomina *dirección de máxima variación*. Desgraciadamente, la propiedad de máxima variación es local en un entorno infinitesimal, debido a que el vector gradiente cambia en cada punto y con él la dirección de máxima variación. Esto se puede comprobar en la Figura H.7 en donde se dispone de una función de dos variables independientes, visualizándose las curvas de nivel y las direcciones del gradiente en unos puntos:

Se puede observar la dirección 11', correspondiente al gradiente en el punto 1, no coincide con la dirección de máxima variación dada por el gradiente en el punto 2

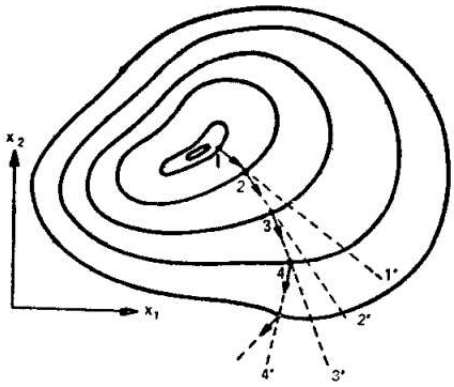

Figura H.7.: Dirección del paso descendente.

(dirección 22').

El Gradiente Descendente se basa en la característica principal del gradiente pero aplicada a la búsqueda de un mínimo, es decir en lugar de utilizar la dirección del gradiente se utiliza la dirección opuesta. El modo de operar es el siguiente: en primer lugar se parte de un punto inicial X_0 y se obtiene un punto nuevo según la siguiente relación iterativa:

$$X_{i+1} = X_i + \lambda_i^* S_i = X_i - \lambda_i^* \nabla f_i \tag{H.26}$$

donde el valor λ_i^* se denomina paso de entrenamiento. Este paso puede ser constante pero siendo aconsejable que sea relativamente pequeño, ya que, como se ha mencionado, la dirección del gradiente es característica de cada punto y un desplazamiento excesivo en el espacio multidimensional no sería beneficioso. En realidad este valor debería ser infinitesimal, pero en las implementaciones reales no puede serlo (puesto que se tardaría un tiempo infinito), con lo cual no se puede garantizar la convergencia independiente del paso inicial elegido. En un problema bidimensional, la aplicación de este algoritmo suele derivar en un camino en zigzag, lleno de segmentos perpendiculares y paralelos, con lo que el proceso se alarga considerablemente. En dimensiones mayores puede que aparezcan características diferentes. Para funciones con gran excentricidad el método se establece en un zigzag *n*-dimensional, con lo que el proceso se hace demasiado lento. Sin embargo, si el contorno de la función objetivo no está muy distorsionado, el método converge más rápido [Pic94].

H.4.3. Método del gradiente conjugado de Fletcher-Reeves

El método del gradiente conjugado supera en convergencia al método del gradiente descendente en el caso de que la función sea cuadrática, ya que garantiza la convergencia en *n* o menos iteraciones, siendo *n* el número de variables independientes de la función (dimensiones del problema de optimización). Esto se suele denominar *convergencia cuadrática*, existiendo otros métodos que presentan esta propiedad.

En realidad, este método parte de la idea de resolver problemas de optimización

cuando las funciones tienen una forma cuadrática, con lo cual es más adecuado si la función a optimizar es de esa forma.

Para explicar este método, es preciso definir previamente el concepto de conjugado y el teorema en que se basa.

Se dice que dos vectores S_i y S_j son mutuamente conjugados respecto a la matriz A si ocurre que:

$$S_i^{'} A S_j = 0 \tag{H.27}$$

Teorema

Suponiendo que un punto X_{i+1} es calculado después de i pasos para minimizar la función cuadrática $f(X) = \frac{1}{2} X^{'} A X + B^{'} X + C$, si las direcciones de búsqueda utilizadas en el proceso, S_1, S_2, S_3,$\cdots S_i$, son mutuamente conjugadas respecto a A, entonces:

$$S_k^{'} \nabla f_{i+1} = 0 \ \ \forall k = 1, \cdots, i \tag{H.28}$$

Basado en este teorema se pueden construir las direcciones de búsqueda para la resolución de un problema de optimización siendo la función de la forma cuadrática. La idea es comenzar por la dirección contraria a la del gradiente y plantear nuevas direcciones que sean conjugadas a las de iteraciones anteriores. Por tanto, la primera dirección propuesta es:

$$S_1 = -\nabla f(X_1) \tag{H.29}$$

y el nuevo punto generado es:

$$X_2 = X_1 + \lambda_1^* S_1 \tag{H.30}$$

donde λ_1^* es el paso de búsqueda lineal optimizado que tiene que cumplir $S_1 \nabla f(X_2)=0$. Considerando las ecuaciones anteriores, este parámetro viene dado por:

$$\lambda_1^* = -\frac{S_1^{'} \nabla f_1}{S_1^{'} A S_1} \tag{H.31}$$

A continuación se expresa S_2 de la siguiente manera:

$$S_2 = -\nabla f_2 + \beta_2 S_1 \tag{H.32}$$

Si se impone la condición $S_1^{'} A S_2 = 0$ entonces se puede deducir β_2:

$$\beta_2 = \frac{\nabla f_2^{'} \nabla f_2}{\nabla f_1^{'} \nabla f_1} = \frac{|\nabla f_2|^2}{|\nabla f_1|^2} \tag{H.33}$$

En el tercer paso la fórmula general para la nueva dirección es:

$$S_3 = -\nabla f_3 + \beta_3 S_2 + \delta_3 S_1 \tag{H.34}$$

Imponiendo que $S_3^{'} A S_1 = 0$ y $S_2^{'} A S_1 = 0$ se llega a que:

$$\delta_3 = 0 \quad \beta_3 = \frac{|\nabla f_3|^2}{|\nabla f_2|^2} \tag{H.35}$$

En general se puede plantear como nueva dirección de búsqueda la siguiente:

$$S_i = -\nabla f_i + \beta_i S_{i-1} \tag{H.36}$$

siendo

$$\beta_i = \frac{|\nabla f_i|^2}{|\nabla f_{i-1}|^2} \tag{H.37}$$

ya que el resto de índices que acompañan a las direcciones de las iteraciones *1* a la *i-2* son cero.

Inspirado en lo anterior, se pueden ver los pasos del algoritmo de una manera general, sin tener en cuenta el tipo de función aplicada:

1. Se escoge un punto arbitrario X_1.
2. Se plantea la primera búsqueda $S_1 = -\nabla f(X_1) = -\nabla f_1$
3. Se determina el punto X_2 de acuerdo con la relación $X_2 = X_1 + \lambda_1^* S_1$, donde λ_1^* es el paso óptimo en la dirección S_1.
4. Se calcula el gradiente $\nabla f_i = -\nabla f(X_i)$ y se determina la siguiente dirección:

$$S_i = -\nabla f_i + \frac{|\nabla f_i|^2}{|\nabla f_{i-1}|^2} S_{i-1} \tag{H.38}$$

5. Se evalúa el paso óptimo λ_i^* en la dirección S_i, y se encuentra un nuevo punto:

$$X_2 = X_1 + \lambda_1^* S_1 \tag{H.39}$$

6. Se verifican las condiciones de convergencia para X_{i+1}. Si X_{i+1} es óptimo, se para el proceso. De otro modo, se actualiza la variable *i=i+1*, y se repiten los pasos 4, 5 y 6 hasta que se alcance la convergencia según algunos criterios de parada.

Este método tiene un mejor comportamiento en el caso de funciones que se acerquen a una aproximación cuadrática, ya que se garantiza una convergencia cuadrática, circunstancia que no ocurre con el método del gradiente descendente. Por otra parte, en este método se tiene en cuenta una mayor información que la correspondiente a la del gradiente descendente, ya que el segundo término que aparece en la parte derecha de la igualdad H.38, contiene dos informaciones: en primer lugar la dirección anterior propuesta, con lo cual evita excesivas oscilaciones en la dirección de búsqueda que contribuyen a empobrecer la convergencia; y en segundo lugar se inyecta información de la curvatura de la hipersuperficie mediante la división del módulo del gradiente en la iteración actual respecto a la anterior.

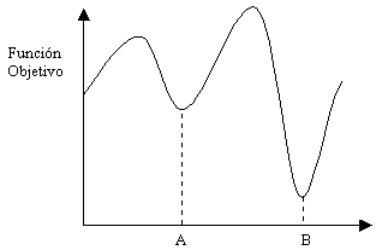

Figura H.8.: Problema del mínimo local.

H.4.4. Método del Camino Aleatorio

La principal ventaja de este método, del Enfriamiento Progresivo y de los Algoritmos Genéticos (detallados en los siguientes subapartados) es la posibilidad de evitar los mínimos locales. Ambos se engloban en los denominados métodos estadísticos, caracterizados por usar números aleatorios para determinar el mínimo.

Este método del Camino Aleatorio, en concreto, consiste en generar una secuencia de aproximaciones al mínimo, cada una basada en la aproximación anterior. Por ello, si X_i es la aproximación al mínimo obtenida en la etapa *i-1*, la nueva aproximación en la etapa i-ésima se obtiene de la relación

$$X_{i+1} = X_i + \lambda U \tag{H.40}$$

donde λ es la longitud escalar del paso preestablecida, y U es un vector unitario aleatorio generado en la etapa i-ésima.

El procedimiento de este método se compone de los siguientes pasos.

1. Se elige un punto inicial X_1 y el paso λ.
2. Se genera un vector aleatorio unitario U.
3. Se determina el nuevo punto según la relación $X_2 = X_1 + \lambda U$.
4. Si $f(X_2) < f(X_1)$, entonces se guarda ese valor haciendo $X_1 = X_2$. Los pasos 2, 3 y 4 se repiten hasta conseguir la condición de parada .

Este método tiende a minimizar la función objetivo pero puede quedar trabado en una solución pobre. En la Figura H.8 se muestra como esto puede ocurrir en un sistema monodimensional. Supóngase que el punto inicial tiene como abscisa A. Si el paso escalar es pequeño, todas las desviaciones desde este punto incrementarán la función objetivo y serán rechazados. Un mejor punto (el que tiene como abscisa B) nunca será encontrado y el sistema quedará atrapado en un mínimo local en vez de alcanzar el mínimo global. Por otro lado, si el paso de entrenamiento es demasiado grande los entornos de los dos puntos serán visitados frecuentemente, pero el sistema nunca se establecerá en el mínimo deseado.

Una estrategia útil para evitar este problema es empezar con grandes pasos e ir reduciendo progresivamente la longitud del paso. Esto permite que el algoritmo escape de los mínimos locales, a la vez que asegura su convergencia. Bastaría, en el algoritmo anterior, empezar con un paso escalar suficientemente grande con respecto a la precisión deseada al final.

1. Se elige un punto inicial X_1 y el paso λ grande.
2. Se genera un vector aleatorio unitario U.
3. Se determina el nuevo punto según la relación $X_2 = X_1 + \lambda U$.
4. Si $f(X_2) < f(X_1)$, entonces se guarda ese valor haciendo $X_1 = X_2$.
5. Los pasos 2, 3 y 4 se repiten un número predeterminado de iteraciones, tras las cuales se reduce el valor del paso λ.

En este nuevo algoritmo, además, se expresa la condición de parada. Ésta consiste en que el paso sea pequeño y que, a pesar de ello, no exista mejoría en un cierto número de intentos.

H.4.5. Método del Enfriamiento Progresivo

Es una refinación del método previo y cuya aplicación a las redes neuronales es factible.

Supóngase que se dispone de una bola en una caja de fondo rugoso, del que la figura anterior podría ser un corte, y se desea situarla en un pozo más profundo. Un posible procedimiento es comenzar agitando la caja de manera violenta. La bola se moverá de un lado a otro, sin asentarse en ningún punto, y en cada instante podrá estar en cualquier punto de la superficie con igual probabilidad. Si la violencia de la agitación se va reduciendo progresivamente, se alcanzará una situación en que la bola estará en los puntos A y B por cortos periodos de tiempo. Si la agitación continúa reduciéndose, se alcanzará un punto crítico en que la agitación es lo suficientemente fuerte para mover la bola de A a B, pero no lo suficiente para permitir a la bola subir la cuesta desde B hasta A. Por lo tanto, al final, la bola quedará en el mínimo global cuando la amplitud de la agitación se reduzca a cero.

Este método presenta una gran analogía con el proceso de temple de los metales. En un metal, calentado por encima de su punto de fusión, los átomos están en violento movimiento aleatorio. Como todos los sistemas físicos, los átomos tienden al estado de mínima energía (un cristal simple en este caso), pero a altas temperaturas, la fuerza del movimiento atómico lo evita. A medida que el metal es progresivamente enfriado se asumen estados de energía cada vez menores, hasta que finalmente se alcanza el menor de todos: el mínimo global. En este proceso la distribución de estados de energía está determinada por la relación siguiente:

$$P(E) \propto exp\left(-\frac{E}{kT}\right) \tag{H.41}$$

donde:

- *P(E)* es la probabilidad de que el sistema esté en un estado de energía *E*.
- k es la constante de Boltzman.
- T es la temperatura en grados Kelvin.

A altas temperaturas el denominador del exponente es muy grande con respecto a cualquier valor de la energía E, por ello el exponente completo estará próximo a cero en todos los casos y P(E) será similar para todos los estados de energía, esto es, los estados de alta energía son tan probables como los de baja. Cuando la temperatura disminuye, el cociente del exponente será mayor para los valores grandes de E. Como éste es negativo, la probabilidad de estados de alta energía se reduce comparada con la probabilidad de estados de baja energía. Dicho de otro modo, cuando la temperatura tiende a cero es muy difícil que el sistema permanezca en un estado de alta energía.

El método del Enfriamiento Progresivo consiste en la aplicación de esta idea a un problema de minimización multidimensional. Este procedimiento implica, no sólo una dirección de búsqueda aleatoria, sino una longitud de paso también aleatoria. Los pasos a seguir serán:

1. Escoger los valores iniciales del vector X_1 y la temperatura T.
2. Calcular un vector unitario U y un paso λ ambos aleatorios.
3. Se determina el nuevo punto según la relación $X_2 = X_1 + \lambda U$.
4. Si $f(X_2) < f(X_1)$, entonces se guarda ese valor haciendo $X_1 = X_2$.

   En caso contrario:

   a) Calcular la probabilidad P(c) de aceptar el cambio $c = f(X_2) - f(X_1)$.

   b) Elegir un valor aleatorio v de distribución uniforme entre 0 y 1. Si $v <$*P(c)*, entonces se guarda ese valor haciendo $X_1 = X_2$.
5. Reducir la temperatura T.
6. Se repiten los pasos 2 a 5 hasta que T sea lo suficientemente pequeño.

La probabilidad de aceptar ese cambio se calcula mediante la distribución de Boltzman con la fórmula

$$P(E) \propto exp\left(-\frac{c}{kT}\right) \tag{H.42}$$

donde

- *P(c)* es la probabilidad de un cambio c en la función objetivo.
- k es una constante, análoga a la de Boltzman, dependiente del problema.
- T es la variable temperatura.

La aplicación de esta probabilidad permite que el sistema dé, ocasionalmente, un paso que empeore la función objetivo. Esto le da la posibilidad para salir de un mínimo local donde los pasos pequeños siempre incrementan la función objetivo.

El paso escalar λ debe depender de la variable T. Esta dependencia puede ser de varias maneras. La empleada en la herramienta de cálculo es la del Algoritmo de Cauchy. Este algoritmo consiste en utilizar la distribución de Cauchy en el cálculo del paso escalar de búsqueda. Esta distribución tiene unas colas largas con lo que se favorece la posibilidad de pasos escalares largos. De hecho la distribución de Cauchy tiene una invarianza infinita (indefinida). La máxima velocidad de reducción de la temperatura se hace inversamente lineal, en vez de inversamente logarítmica como en el caso de utilizar la distribución de Boltzman [Ham95]. De este modo se reduce drásticamente el tiempo de convergencia. Esta relación se puede expresar como

$$T(t) = \frac{T_0}{1+t} \tag{H.43}$$

La distribución de Cauchy es

$$\rho(\lambda) = \frac{T}{T^2+\lambda^2} \tag{H.44}$$

donde $\rho(\lambda)$ es la densidad de probabilidad de un paso de longitud λ. Ésta puede ser integrada por los métodos usuales resultando la siguiente expresión:

$$\lambda_c = tan\,[P(\lambda)] \tag{H.45}$$

donde λ_C es la longitud de paso para la cual la probabilidad es *P(λ)*. Para encontrar λ se siguen estos pasos:

1. Seleccionar un numero aleatorio de una distribución uniforme sobre el intervalo abierto (-π/2,π/2) (necesario para limitar la función tangente).

2. Sustituir *P(λ)* por este valor en la fórmula y calcular el paso de búsqueda λ_C usando la temperatura actual.

H.4.6. Algoritmos Genéticos

La idea de los algoritmos genéticos puede ser entendida como una explotación inteligente del método del Camino Aleatorio. El nombre del algoritmo proviene de la analogía entre una estructura compleja, cuya solución puede ser representada por un vector de componentes, y la estructura genética de los cromosomas. Así, en la crianza selectiva de animales, por ejemplo, se busca que las siguientes generaciones presenten unas ciertas características deseadas. Éstas son determinadas por la combinación de los cromosomas de los padres, por lo que se suelen cruzar aquellos ejemplares con mejores propiedades. De un modo similar, en la búsqueda de mejores soluciones de problemas complejos, se combinan frecuentemente pedazos de soluciones existentes. Por otra parte, en la Naturaleza, se dan mutaciones (cambios aleatorios en los genes) que pueden originar individuos con mejores características. Análogamente, pequeñas variaciones en una solución existente pueden dar lugar a mejores resultados. Este

Figura H.9.: Ejemplo de mutación de un vector de diseño.

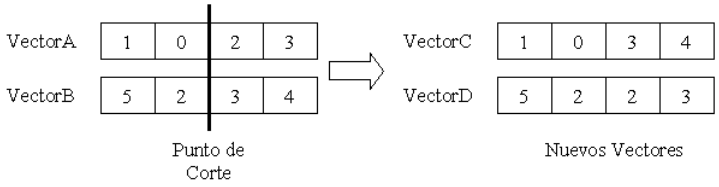

Figura H.10.: Ejemplo de *crossover* entre vectores de diseño.

paralelismo no es completo, pero lo suficientemente claro como para justificar su nombre [RS91].

En muchas aplicaciones, tal como es el caso de las redes neuronales, la solución viene en función de varios parámetros (por ejemplo, pesos a ajustar) que pueden ser agrupados en un vector. Éste puede ser asemejado con un cromosoma formado por una cadena de genes. Tratando de aprovechar esta similitud, varios operadores genéticos han sido definidos para la manipulación de estos cromosomas. Los más frecuentes son el *crossover* (una recombinación de fragmentos de los cromosomas padre) y la *mutación* (una modificación aleatoria de un cromosoma).

Debido a las numerosas versiones existentes de este tipo de algoritmo, únicamente se va a reflejar la variación implementada en la herramienta de cálculo.

Los individuos de la población son vectores de diseño, cada uno de los cuales representa una solución del problema y son los padres potenciales de nuevos cromosomas. Al principio del entrenamiento se crean tantos vectores de diseño de forma aleatoria como población inicial se desee. En cada generación se generan nuevos miembros de la población producidos por las mutaciones y crossover de los actuales.

El proceso de mutación es el siguiente: se toma un individuo al azar, se genera un número entero aleatorio que indica el número de componentes del vector que se van a mutar. También se escogen al azar los índices de las componentes del vector que se sustituyen así como el valor por el que son sustituidas (Figura H.9).

Mientras, los individuos generados mediante *crossover* se producen del siguiente modo: se toman dos individuos al azar, se escoge de manera aleatoria una componente del vector, que se tomará como punto de corte para la recombinación de los individuos. A partir de estos datos se generan dos nuevos individuos para la población. El primer vector generado está formado por la primera parte del primer vector y la segunda parte del segundo vector, y el otro vector se forma por la primera parte del segundo vector y la segunda parte del primer vector, tal como muestra el ejemplo de la Figura H.10:

Una vez generados los nuevos individuos, se eliminan los individuos de la población antigua que presenten peor comportamiento (mayor coste), de modo que

los supervivientes y los nuevos individuos no superen la población máxima fijada. La razón de eliminar los individuos con peor comportamiento de la población antigua y no de la global (antiguos más formados en esta generación) es para aumentar la diversidad de la población, ya que en general se obtienen mejores resultados.

Uno de los factores más analizados por los investigadores en este algoritmo es la determinación de una población máxima adecuada. Así, las poblaciones pequeñas pueden no cubrir adecuadamente el espacio de las soluciones, mientras que las grandes producen un elevado coste computacional.

Muchas otras variaciones de este algoritmo han sido aplicadas: *crossover* con más de un punto de sección, inversiones de vectores... Se cita las definiciones de genes *dominantes* y *recesivos*. Los genes recesivos únicamente permanecen en la generación siguiente si son emparejados con otro gen recesivo. Sin embargo, en la mayoría de los casos, la única aplicación de crossover simples y de mutaciones se ha demostrado suficiente.

H.5. Algoritmos de Optimización del Paso de Entrenamiento

En este subapartado se mostrarán los diversos algoritmos de optimización del paso de entrenamiento escogidos para la implementación de la herramienta de cálculo *Evenet2000*.

Se pueden dividir los métodos de optimización unidimensional en dos grandes grupos, los de carácter analítico y los numéricos. A su vez, en los numéricos se distinguen los métodos de eliminación y los de interpolación.

Los métodos de optimización provenientes del cálculo diferencial son de carácter analítico y aplicables a funciones continuas y dos veces diferenciables. En estos métodos, los cálculos de la función se hacen en el último paso del proceso, es decir, el valor óptimo de la función objetivo se calcula después de determinar el valor óptimo de la abscisa. En los métodos numéricos, por el contrario se calculan los valores de la función a optimizar a medida que se comparan las ordenadas con el objetivo de deducir el mínimo (o máximo) y, como consecuencia de esto, obtener la abscisa óptima.

El método analítico requiere un conocimiento de la forma analítica explícita de la función, con lo cual, si se desconoce, sólo se pueden utilizar métodos de optimización numéricos, en donde sólo se evalúa la función en una serie de puntos. Este es el caso del entrenamiento de Redes Neuronales.

Los métodos de eliminación se basan en acotar en un intervalo cada vez más pequeño el mínimo (o máximo). Estos métodos se pueden utilizar tanto para funciones continuas como para discontinuas. Los métodos de interpolación consisten en ajustar diversos puntos conocidos a una función sencilla de la cual se conozcan analíticamente el mínimo/máximo o las raíces, proponiendo estos últimos como nuevos candidatos a mínimo. Finalmente, existen otros métodos numéricos híbridos entre las dos clases anteriormente citadas. A modo de ejemplo, se pueden mencionar el método de Brent para optimización de funciones que consiste en una combinación adecuada de un método de interpolación y un método de eliminación.

En la herramienta de cálculo se han implementado un método de interpolación (Interpolación cuadrática o parabólica) y otro de eliminación (Sección de oro).

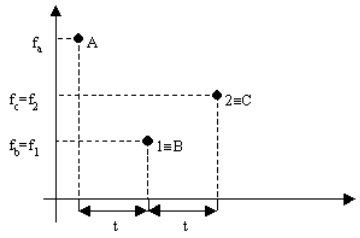

Figura H.11.: Situación 1 de Búsqueda Inicial.

H.5.1. Búsqueda Inicial

En todos los métodos de optimización unidimensionales siempre existe una primera fase que consiste en acotar el intervalo en donde se encuentra el mínimo. Más concretamente, el método utilizado se basa en buscar tres valores de la abscisa *a*, *b* y *c* tales que:

$$f(a) > f(b) \quad f(b) < f(c) \tag{H.46}$$

Con esta condición se garantiza que existe un mínimo en el intervalo abierto (*a*, *c*), siempre y cuando la función sea continua en el intervalo cerrado [*a*, *c*]. Se puede justificar esta afirmación de la siguiente manera: por ser continua en ese intervalo cerrado existen un mínimo y un máximo. El mínimo no puede estar en los extremos, debido a que existe un punto intermedio *b* con una abscisa menor que la de los extremos del intervalo, de modo que el mínimo debe estar en el intervalo abierto.

Para determinar el conjunto de estos tres puntos, se escoge un punto *a* y un paso *t*. Pueden ocurrir dos posibilidades:

1. Si $f(a+t)=f_1<f(a)$. En este caso se puede asignar el punto *1* a *b*, y avanzar un paso *t*, calculando un segundo punto *2*. Se comprueba que si el nuevo punto cumple $f(b+t)=f_2>f(b)$. En caso afirmativo este punto es el *c* buscado (como ocurre en la situación de la gráfica H.11) y en caso contrario el nuevo punto toma el papel de *b* y se repite el test anterior con un paso *2t*.

2. Si $f(t+a)=f_1>f(a)$. En este caso será el punto *1* el que se asigna como punto *c* y el objetivo es encontrar un punto *2* que esté situado en el punto medio entre *a* y *c* tal que su ordenada sea inferior a la de *a*. Si no ocurre, lo anteriormente mencionado el punto intermedio pasa a ser el nuevo *c* y se repite el test con un paso *t/2* hasta que se cumpla la condición. Cuando esto suceda, el punto intermedio será el *b* buscado (como ocurre en la situación de la gráfica H.12).

Una vez que se ha determinado el intervalo donde está el mínimo se emplean métodos de búsqueda más precisos para fijar su posición.

H.5.2. Método de interpolación parabólica

El método se resume del siguiente modo. Se obtienen tres puntos de la manera señalada en la sección anterior, x_0, x_1 y x_2 se ajustan a una parábola y se considera

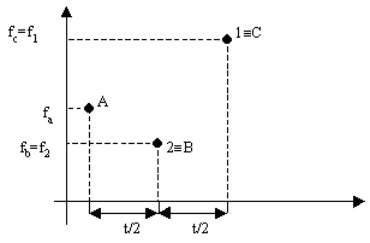

Figura H.12.: Situación 2 de Búsqueda Inicial.

como candidato el mínimo de la parábola así determinado.

De la expresión de la parábola:

$$p(x) = a_1(x-x_0)^2 + b_1(x-x_0) + c_1 \tag{H.47}$$

Siendo a_1, b_1 y c_1

$$\begin{aligned} a_1 &= \frac{q-r}{d} \\ b_1 &= \frac{(x_0-x_2)q-(x_0-x_1)r}{d} \\ c_1 &= f(x_0) \end{aligned} \tag{H.48}$$

donde:

$$\begin{aligned} q &= (x_0-x_2)\,(f(x_0)-f(x_1)) \\ r &= (x_0-x_1)\,(f(x_0)-f(x_2)) \\ d &= (x_0-x_1)(x_1-x_2)(x_2-x_0) \end{aligned} \tag{H.49}$$

entonces, el mínimo de la parábola viene dado por:

$$x = x_0 - \frac{b_1}{2a_1} \tag{H.50}$$

El paso de entrenamiento así determinado λ^* es una aproximación al mínimo buscado de la función.

El proceso se continúa eligiendo tres nuevos puntos, uno de los cuales es el mínimo aproximado, tales que cumplan la condición H.46. Los cálculos finalizarán cuando se crea estar lo suficientemente cerca del mínimo tal como indique un criterio de parada.

H.5.3. Método de la Sección de Oro

A semejanza del método anterior el objetivo es, a partir de tres puntos que cumplan la condición inicial H.46, determinar tres nuevos puntos a', b' y c' que también cumplan y que acoten mejor el mínimo, es decir:

$$\left|c' - a'\right| < |c - a| \tag{H.51}$$

En el caso del método descrito en este subapartado, se calcula previamente un punto intermedio x, con el que se determinan los nuevos puntos $a´$, $b´$ y $c´$. Este punto viene dado de la siguiente manera:

$$x = \begin{cases} b + \alpha(c - b) & Si\ |c - b| > |a - b| \\ b + \alpha(a - b) & Si\ |c - b| < |a - b| \end{cases} \tag{H.52}$$

La constante α definida en la expresión anterior se denomina Constante de Oro o Sección de Oro, y tiene el siguiente valor:

$$\alpha = \frac{3 - \sqrt{5}}{2} \tag{H.53}$$

En cuanto a la construcción del nuevo trío en función de x y a, b, c viene dada de la siguiente forma:

1. Si *f(b)* <*f(x)* y *x* >*b* entonces el trío viene dado por (*a* ,*b, x*).
2. Si *f(b)* >*f(x)* y *x* >*b* entonces el trío viene dado por (*b, x, c*).
3. Si *f(b)* <*f(x)* y *x* <*b* entonces el trío viene dado por (*x, b, c*).
4. Si *f(b)* >*f(x)* y *x* <*b* entonces el trío viene dado por (*a, x, b*).

Esta distinción de cuatro casos se hace para que los nuevos puntos *a'*, *b'* y *c'* cumplan la condición H.46.

La propiedad fundamental de este método es que el factor de disminución del tamaño del intervalo donde está situado el mínimo tiende rápidamente a ser 0.381966 (constante de Oro) [Mar98, Mar03].

I. Grafo de Flujo de Señal y su aplicación a redes neuronales

Cuando se trabaja con una nueva estructura de red neuronal, la deducción de la expresión del gradiente normalmente lleva consigo realizar cálculos complicados y engorrosos. Así, por ejemplo, el algoritmo de *backpropagation* para el entrenamiento de redes de alimentación hacia delante se obtiene mediante la aplicación, repetidas veces, de extensiones de la regla de la cadena hacia atrás en la red. Sin embargo, el cálculo de la derivada se puede ver como un sencillo cambio de sentido en el flujo tanto de la señal como en el tiempo. En ambos procedimientos, existe una reciprocidad entre la propagación de los valores hacia delante y la propagación hacia atrás de los términos del gradiente. Numerosos autores [WB94, WB97, Oso94, OH95, OC99, CMUP97] han demostrado que esta reciprocidad es característica de todas las arquitecturas y que su algoritmo puede ser deducido directamente, sin necesidad de extensiones de la regla de la cadena, mediante la técnica de Grafo de Flujo de Señal (*Signal Flow Graph, SFG*). Esta teoría tiene su origen en el campo de los circuitos eléctricos. Así, el teorema de Tellegen [BLS03], relacionando topológicamente redes idénticas, permite demostrar numerosas propiedades de los circuitos eléctricos.

Desde el punto de vista práctico, Wan y Beaufays [WB94] presentan un método para la obtención del gradiente para redes neuronales dependientes del tiempo, basado en un conjunto de reglas sencillas de manipulación de diagramas de bloque. Dicho método hace uso del Grafo de Flujo de Señal para construir y manipular diagramas de bloques que representan redes neuronales.

Con anterioridad al trabajo citado, estos métodos de reciprocidad ya se habían aplicado a redes neuronales, pero se limitaron a arquitecturas donde los sistemas recíprocos provenían de una técnica de optimización Euler-Lagrange. En cambio el enfoque de Wan y Beaufays no elabora un método gráfico que se limite a ilustrar la fórmula del gradiente obtenida mediante un método alternativo. Su propuesta consiste en un sencillo método de diagrama de bloques para representar la red neuronal y construir una red recíproca a partir de la cual se puede escribir la fórmula del gradiente.

Este apéndice está dedicado a la descripción general del Grafo de Flujo de Señal y del método citado de Wan y Beaufays para la determinación de los términos del gradiente.

I.1. Grafo de Flujo de Señal

El Grafo de Flujo de Señal es una representación gráfica para un sistema lineal de ecuaciones de la forma

$$\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0} \tag{I.1}$$

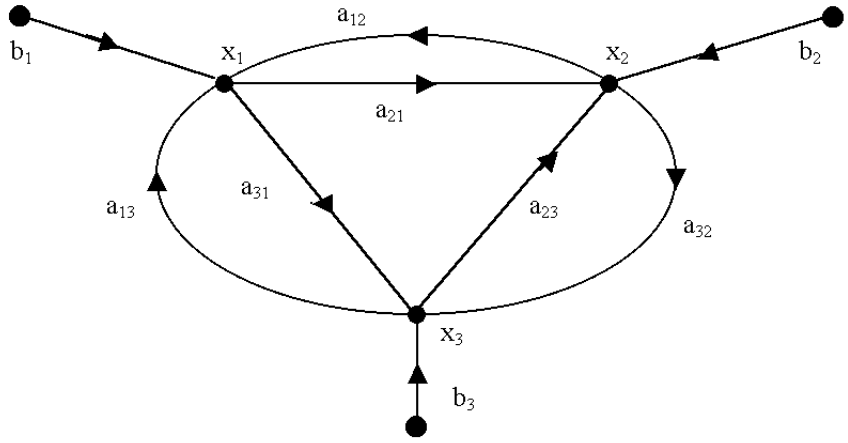

Figura I.1.: Ejemplo de Grafo de Flujo de Señal

donde el vector **b** corresponde a las excitaciones del sistema y la matriz **A** representa los pesos de conexión entre las variables que forman el vector **x**. Sin pérdida de generalidad se puede asumir que los términos de la diagonal de la matriz **A** son iguales a -1, esto es,

$$\mathbf{A} = \begin{bmatrix} -1 & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & -1 & a_{23} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & -1 \end{bmatrix} \tag{I.2}$$

El sistema en cuestión puede ser representado de modo gráfico a través de un flujo de señales entre nodos, donde cada nodo representa una variable x_i, y un conjunto de arcos orientados a los que se le asigna un coeficiente. Por definición el valor de la señal asociada con el nodo es la suma de las señales ponderadas por los coeficientes de los arcos que entran en ese nodo. Este tipo de representación es lo que se denomina Grafo de Flujo de Señal, esto es, un conjunto de nodos y arcos orientados. Aquellos nodos que solamente tienen un arco de salida y ninguno de entrada se les llama *nodos de entrada.* Del mismo modo, los *nodos de salida* son aquellos nodos que solamente tienen un arco de entrada y ninguno de salida. Los arcos que unen nodos que no son ni de entrada ni de salida reciben el nombre de *arcos-f.*

En el caso particular de un sistema de tres ecuaciones de la forma:

$$\begin{bmatrix} -1 & a_{12} & a_{13} \\ a_{21} & -1 & a_{22} \\ a_{31} & a_{32} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \mathbf{0} \tag{I.3}$$

se puede representar su Grafo de Flujo de Señal de la forma mostrada en la Figura I.1

Continuando con este ejemplo, la señal de cada nodo x_i viene dada por

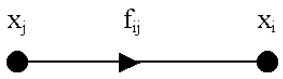

Figura I.2.: Arco complejo

$$x_i = \sum_{k=1,k\neq i}^{3} a_{ik}x_k + b_i \tag{I.4}$$

La relación entre los nodos origen y final de cada arco puede ser más compleja que la simple multiplicación por un escalar. Así en el caso más general, un arco del tipo mostrado en la Figura I.2, debe ser interpretado como

$$x_i = f_{ij}[x_j] \tag{I.5}$$

con f_{ij} una función arbitraria.

La representación del sistema en forma de grafo de flujo de señal encuentra numerosas aplicaciones en el análisis y síntesis de sistemas lineales. Como representación gráfica del flujo de señales, el grafo es un factor importante en la comprensión del comportamiento del sistema en diversas situaciones. Así la fórmula de Mason para la ganancia, aplicada a los grafos, simplifica el problema de su análisis y proporciona una solución en función explícita de los parámetros. Existen numerosos programas que resuelven estos grafos, convirtiendo este método de análisis es muy interesante desde el punto de vista práctico. Por otra parte, una gran cantidad de sistemas complejos, tales como redes neuronales recurrentes, circuitos eléctricos, filtros adaptativos [HSMP96] o reacciones químicas [DPD95], pueden ser representados mediante grafos.

Dado un grafo de flujo de señal G, se obtiene el grafo *recíproco* (también denominado en la literatura como traspuesto, adjunto o inverso) $\mathbf{G_r}$ al invertir la orientación de todos sus arcos. Este grafo recíproco presenta numerosas propiedades. Antes de explicarlas en detalle, introduciremos la notación para estos grafos que se puede ver en la Figura I.3.

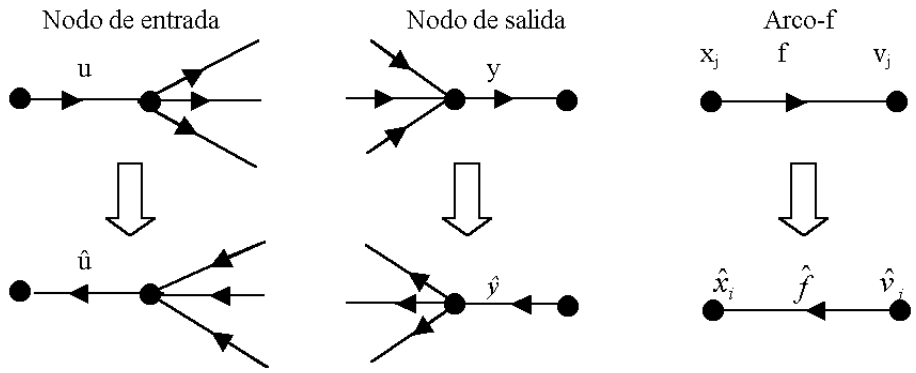

Figura I.3.: Notación para los grafos recíprocos

Con esta notación, si el grafo original G presenta r salidas, m entradas y q arcos-f, se puede enunciar el siguiente teorema, similar al teorema de Tellegen en la teoría de redes [CMUP97].

Teorema

Considérese un sistema causal dinámico. Sea G su correspondiente Grafo de Flujo de Señal. Sean u_i las ganancias de los arcos de los nodos de entrada, y_i las ganancias de los arcos de salida de G. Sean x_i y v_i los valores asociados respectivamente a los nodos origen y destino de los arcos-f de G. Sea G_r su grafo recíproco. Si $\widehat{u}_i$, $\widehat{y}_i$, $\widehat{x}_i$ y $\widehat{v}_i$ son las variables en G_r relativas respectivamente a u_i, y_i, x_i y v_i en G, entonces

$$\sum_{i=1}^{r} \widehat{y}_i(t) * y_i(t) + \sum_{i=1}^{q} \widehat{x}_i(t) * x_i(t) = \sum_{i=1}^{m} \widehat{u}_i(t) * u_i(t) + \sum_{i=1}^{q} \widehat{v}_i(t) * v_i(t) \tag{I.6}$$

donde * denota el operador de convolución. Para sistemas estáticos esta ecuación se convierte en

$$\sum_{i=1}^{r} \widehat{y}_i y_i + \sum_{i=1}^{q} \widehat{x}_i x_i = \sum_{i=1}^{m} \widehat{u}_i u_i + \sum_{i=1}^{q} \widehat{v}_i v_i \tag{I.7}$$

Esto permite otra importante aplicación de los grafos: el cálculo de la sensibilidad de un circuito. Se ha demostrado que este término definido como la derivada de cualquiera de sus señales respecto a la ganancia de un arco, puede ser calculada como el producto de la señal en el grafo original G y de una señal en el grafo recíproco G_r. Así, en un circuito, cuyo grafo original puede representarse del modo mostrado en la Figura I.4, la sensibilidad de cualquier nodo v_k del sistema con respecto a la ganancia f_{ij} es

$$\frac{dv_k}{df_{ij}} = v_j \hat{v}_i^{(k)} \tag{I.8}$$

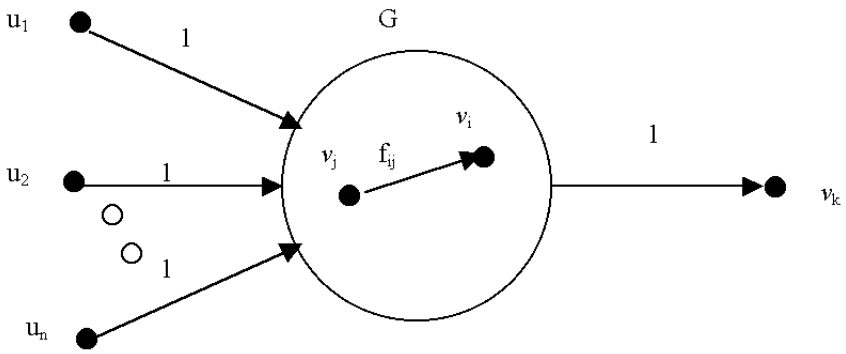

Figura I.4.: Esquema de un grafo de flujo de señal de un sistema

donde $\widehat{v}_i^{(k)}$ representa la señal del nodo i-ésimo en el grafo recíproco $\mathbf{G_r}$ cuando se aplica una excitación unidad en el nodo k-ésimo.

Este concepto de sensibilidad es de gran importancia, entre otros aspectos por su generalización en forma de gradiente para las estrategias de aprendizaje supervisado en redes neuronales.

En la mayor parte de las aplicaciones (incluidas las redes neuronales o las optimizaciones en el diseño de circuitos eléctricos) la función de energía (u objetivo, que en el caso de las redes neuronales se identifica con la función de coste) se define como la función cuadrática de las señales de los nodos escogidos, esto es:

$$E(\mathbf{W}) = \frac{1}{2}\sum_{k=1}^{M}(v_{out,k} - d_k)^2 \tag{I.9}$$

donde $v_{out,k}$ es la señal actual del nodo de salida k-ésimo, d_k es el valor deseado de este nodo y M es el número de nodos de salida que forman parte en la definición de la función de energía. Aunque esta ecuación es cuadrática respecto a las señales de los nodos, es en general no lineal respecto a los parámetros a optimizar (un conjunto de pesos w_{ij}) ya que $v_{out,k}$ es no lineal respecto a los pesos.

El gradiente ∇E se define como el vector de derivadas de la función de energía respecto a los parámetros (pesos) del sistema. Por diferenciación

$$\frac{\partial E}{\partial w_{ij}} = \sum_{k=1}^{M}(v_{out,k} - d_k)\frac{\partial v_{out,k}}{\partial w_{ij}} \tag{I.10}$$

con lo que para determinar el gradiente de la función de energía, se necesita calcular la sensibilidad de los nodos de la salida. La aplicación del grafo de flujo de señal a este problema puede simplificar notablemente los cálculos necesarios. En el grafo adjunto pueden aplicarse todas las excitaciones a la vez, aplicando la regla de la superposición lineal para reproducir la última expresión en forma de sencilla multiplicación de dos señales, de manera análoga a (I.8)

$$\frac{\partial E}{\partial w_{ij}} = v_j\widehat{v}_i \tag{I.11}$$

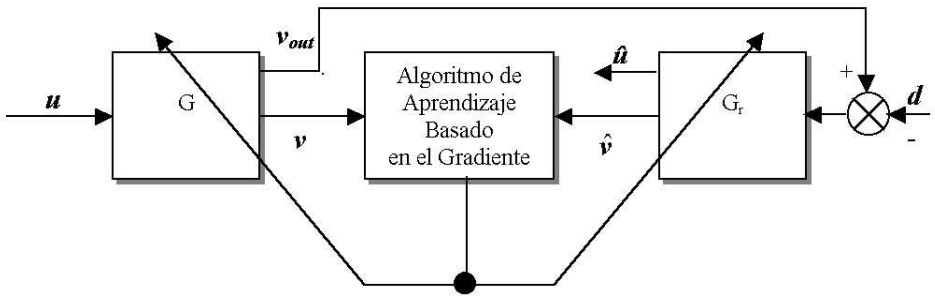

Figura I.5.: Esquema de uso de la red recíproca para el entrenamiento de redes neuronales

La única diferencia entre las expresiones de la sensibilidad y la del gradiente estriba en la excitación del sistema adjunto. En el caso de la sensibilidad es la señal unidad aplicada al nodo apropiado de G_r, mientras que en el cálculo del gradiente esta señal es igual a la diferencia entre el valor actual de $v_{out,k}$ y el valor deseado d_k, es decir igual al error producido. En el caso de que la función de coste usada no sea la cuadrática, el método sigue siendo válido, solamente cambia la expresión de la excitación.

La obtención del gradiente puede ser utilizada para adaptar los pesos del sistema de modo que el vector de salida de la red $\boldsymbol{v_{out}}$ se aproxime al de salidas deseadas $\boldsymbol{d}$. Así ambas redes G y G_r pueden incluirse en un sistema auto-adaptativo junto a un bloque que representa un algoritmo de aprendizaje basado en el gradiente.

En la Figura I.5 se realiza una representación de este sistema auto-adaptativo. Los vectores de pares de entrenamiento ($\boldsymbol{u}$,$\boldsymbol{d}$) son empleados en el proceso de auto-adaptación. El vector $\boldsymbol{u}$ es el vector de excitación del sistema original G, mientras que la diferencia entre el vector salida $\boldsymbol{v}$ y el vector de salidas deseado $\boldsymbol{d}$ constituye la excitación del sistema recíproco. Los vectores salida de ambos sistemas son aplicados como entradas al bloque que genera el algoritmo de aprendizaje basado en el gradiente, que adapta los pesos de G y G_r para hacer el vector de salida $\boldsymbol{v_{out}}$ de G igual al vector de salidas deseado $\boldsymbol{d}$. El sistema automáticamente ajusta sus pesos en la dirección de minimización de la función de energía [OH95].

I.2. Representación de Redes Neuronales en Diagramas de Bloque

De lo visto en la subsección I.1 se deduce que el primer paso a seguir es representar la red en forma de Grafo de Flujo de Señal. En el caso de una neurona típica (entradas ponderadas por pesos, evaluando su suma a través de una función de activación f) este grafo se corresponde al que se muestra en la Figura I.6.

Sin embargo, por razones prácticas, Wan y Beaufays proponen otra representación para las redes. En vez de nodos y arcos orientados, emplean diagramas de bloques [WB94, WB97]. Este método sigue siendo en esencia un Grafo de Flujo de Señal. No obstante, a la hora de desarrollar la herramienta *Evenet2000*, se ha preferido esta representación por ser más intuitivo su tratamiento mediante la OOP. Así, una red neuronal arbitraria puede ser representada como un diagrama constituido por

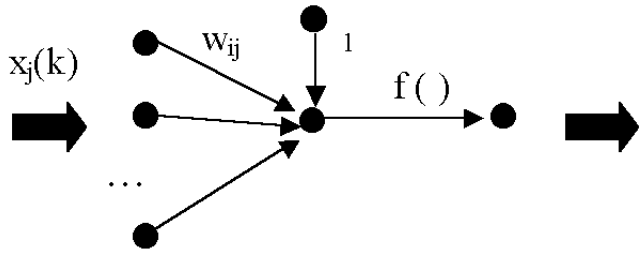

Figura I.6.: Grafo de Flujo de Señal de una neurona típica

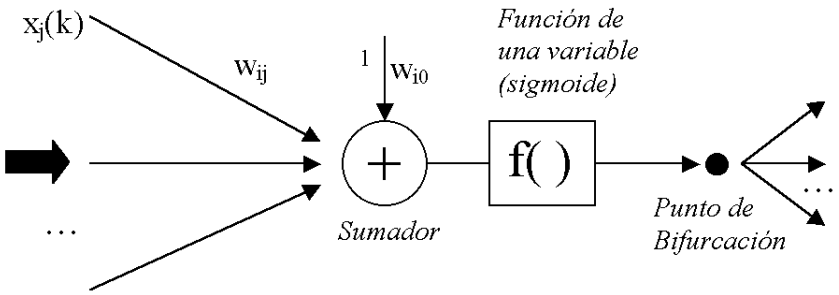

Figura I.7.: Neurona Típica en representación de Diagrama de Bloques

puntos de suma, puntos de ramificación, funciones de una variable, funciones de varias variables y operadores de retardo en el tiempo. Por simplicidad, consideraremos sistemas discretos en el tiempo.

Un peso en una sinapsis, por ejemplo, puede ser pensado como un caso especial de una función de una variable. La neurona típica es simplemente un bloque sumador seguido por un bloque representando una función de una variable (por ejemplo la función sigmoide). La neurona básica se completa con un punto de ramificación que establece las conexiones con los siguientes elementos de la red. Esta configuración de bloques se representa en la Figura I.7.

De este modo, las diversas redes pueden construirse a partir de neuronas individuales así como otros bloques adicionales de funciones y operadores de retardo.

I.3. Reglas de construcción de la Red Recíproca

Una vez representada la red objeto de problema a través de un diagrama de bloques, únicamente se deben aplicar un conjunto de sencillas reglas de transformación, invirtiendo el sentido de flujo de la red original. Las señales resultantes de este proceso se denominarán $\delta_i(k)$ por representar términos relativos a errores propagados. Las operaciones a realizar son las siguientes [WB94].

1. Los sumadores se convierten en puntos de ramificación

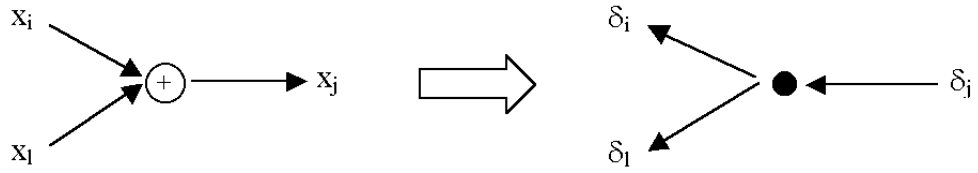

Figura I.8.: Conversión recíproca de sumadores

2. Los puntos de ramificación se convierten en sumadores

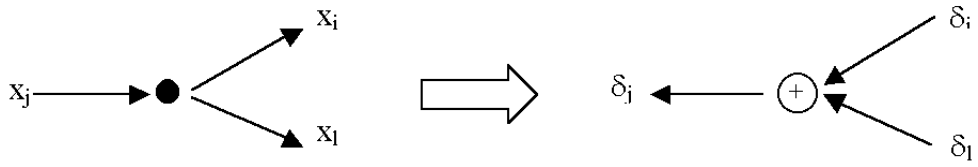

Figura I.9.: Conversión recíproca de puntos de ramificación

3. Las funciones de una variable se reemplazan por sus derivadas

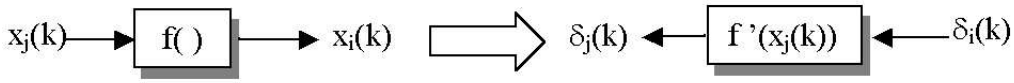

Figura I.10.: Conversión recíproca de función de una variable

Casos especiales son:

- Pesos: $x_i(k) = w_{ij}x_j(k)$, en cuyo caso $\delta_j(k) = w_{ij}\delta_i(k)$

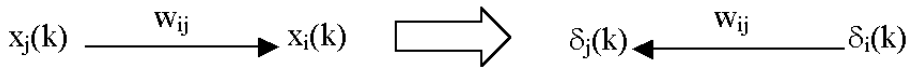

Figura I.11.: Conversión recíproca de pesos

- Funciones de activación: Por ejemplo, si $x_i(k) = tanh(x_j(k))$, entonces $\delta_j = 1 - x_i^2(k)$

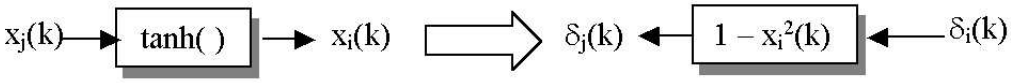

Figura I.12.: Conversión recíproca de función de activación

4. Las funciones de varias variables se reemplazan por sus Jacobianos

   En general, una función de varias variables proporciona un vector de salidas a partir de un vector de entradas: $\boldsymbol{x_{out}}(k)= F(\boldsymbol{x_{in}}(k))$. En la red recíproca se obtiene $\delta_{\boldsymbol{in}}(k)=G^F(\boldsymbol{x_{in}}(k))\delta_{\boldsymbol{out}}(k)$. Los bloques sumadores y las funciones de una variable son casos especiales de funciones de varias variables.

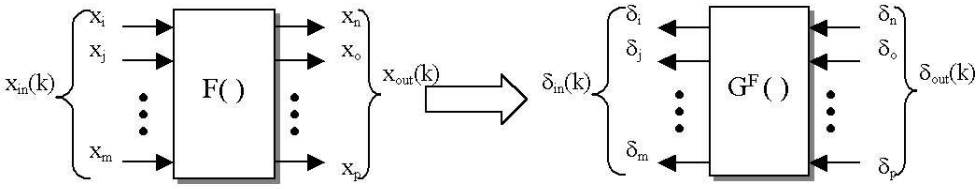

Figura I.13.: Conversión recíproca de función de varias variables

Un caso de interés es el las redes neuronales con estructuras en capas. Esta red puede ser interpretada como una función de varias variables. En este caso el producto $G^F(\boldsymbol{x_{in}}(k))\delta_{\boldsymbol{out}}(k)$ puede ser calculado directamente a través de una alimentación hacia atrás del término $\delta_{\boldsymbol{out}}(k)$ sin necesidad de calcular los términos de la matriz G^F de manera explícita.

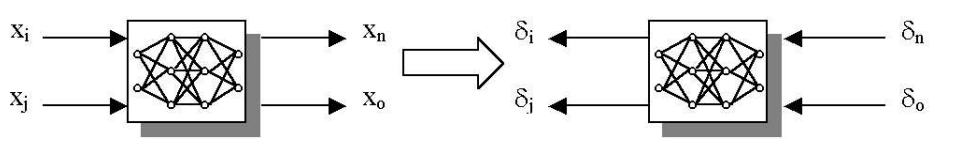

Figura I.14.: Conversión recíproca de redes neuronales con estructura en capas

5. Los operadores de retardo son reemplazados por operadores de avance

   Un operador q^{-1} introduce un retardo unidad a su entrada: $x_i(k) = q^{-1}x_j(k) = x_j(k-1)$. En la red recíproca, este operador es sustituido por un avance unidad: $\delta_j(k) = q^{+1}\delta_i(k) = \delta_i(k+1)$. Por tanto el sistema resultante es no causal. En general, cualquier subsistema lineal $H(q^{-1})$ en la red original es sustituido por $H(q^{+1})$ en la red recíproca.

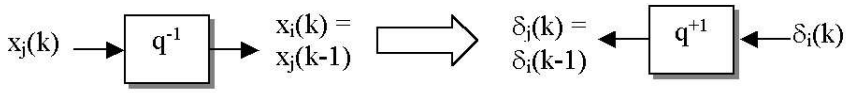

Figura I.15.: Conversión recíproca de operadores de avance

6. Las salidas se convierten en entradas

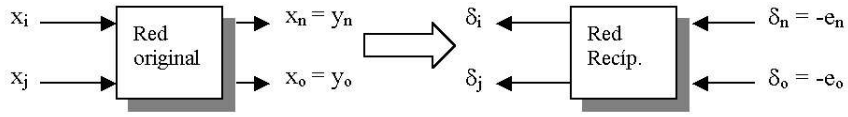

Figura I.16.: Conversión recíproca de salidas

Por inversión del flujo de señales, los nodos de salida de la red original $x_n(k) = y_n(k)$ se convierte en los nodos de entrada en la red recíproca. El término e_n se refiere a la componente del vector error (definido en el caso de la función de coste cuadrática como la diferencia entre el valor deseado de la salida y el valor actual de la misma) correspondiente a la salida y_n. En el caso de una función de coste J genérica, este término se sustituiría por $\partial J/\partial y_n$.

Estas seis reglas permiten la construcción directa de la red recíproca a partir de la red original, aunque este conjunto no es el mínimo posible. Por ejemplo, un sumador puede ser considerado como un caso especial de una función de varias variables. No obstante, el conjunto descrito permite una mayor claridad en la construcción. Por ello, ha sido la elegida para su empleo en la herramienta *Evenet2000*.

J. Librería y Entorno de Cálculo para el Entrenamiento de Redes Neuronales Evenet2000

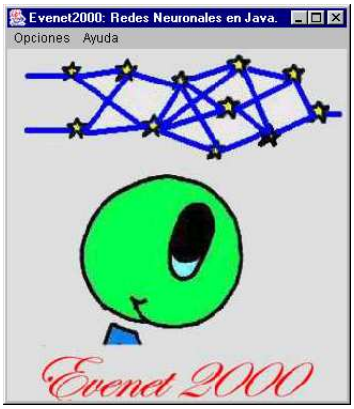

Figura J.1.: Logotipo de Evenet2000

En este apéndice se presenta en detalle la herramienta de cálculo implementada en Java Evenet2000[1], cuyas clases han sido empleadas en el sistema de agentes presentado en este trabajo. Esta aplicación, desarrollada en la Universidad de La Laguna [Gon00, GMH+00, GHM+01, GHSM01, GHM+03a, GHM+03b, GHMM03] se basa en la técnica de Grafo de Flujo de Señal presentada en la sección I, para el diseño y entrenamiento de redes neuronales de arquitectura arbitraria, aunque, tal como se verá más adelante puede ser empleada para problemas de optimización generales. De esta forma su usuario puede evitar el engorroso trabajo de deducir las fórmulas del gradiente cuando se diseña una nueva arquitectura de red neuronal. El concepto innovador de la herramienta es el tratamiento de esta técnica mediante la programación orientada a objetos (sección 1.1), puesto que la división de las redes en diferentes bloques así lo sugiere.

En la descripción de la herramienta se ha procurado seguir un orden cronológico: desde la planificación de la estructura hasta la implementación de un editor gráfico para el diseño de nuevas arquitecturas de redes. El primer subapartado versará sobre la división de las redes en elementos básicos (sumador, punto de bifurcación...). A continuación se tratará de cómo ensamblar estos elementos para el cálculo de la salida y del gradiente de la red. Finalmente se analizan los tres paquetes en los que se ha

[1] http://evenet.cyc.ull.es

estructurado la herramienta de cálculo. El primero, denominado `calculos`, contiene las clases de los diversos elementos de red, algoritmos, algoritmos de optimización del paso de entrenamiento,... El segundo, denominado `interfase` almacena las clases relativas a la implementación de una interfaz de usuario. El último de ellos, llamado `editorGrafico`, se refiere a las clases que permiten utilizar un editor gráfico para la construcción de redes de arquitectura arbitraria.

Finalmente aclararemos que el empleo de herramienta es extensible a los sistemas de optimización en general y cálculo de funciones, como se demuestra en la sección 5.2.

J.1. Elementos de Red

Basándose en las reglas de transformación contempladas en la sección I y en la estrategia de la Programación Orientada a Objetos explicada en la sección 1.1, el objetivo a alcanzar es implementar una herramienta de cálculo cuyas principales armas sean la generalidad (se puede tratar cualquier estructura arbitraria de redes neuronales) y las derivadas de la propia OOP (fácil reutilización y ampliación del código,...). El lenguaje de programación escogido para tal fin ha sido Java (apartado 1.1.9) por sus características de fuerte orientación a objetos y posibilidad de ejecución de manera independiente de la plataforma a través de una máquina virtual.

Se ha visto anteriormente que las reglas de transformación de una red arbitraria en una red recíproca se originan a partir de una división de esta red arbitraria en bloques. Cada uno de estos bloques presenta un comportamiento bien definido para el cálculo tanto del vector de salida de la red como de las componentes del vector gradiente. Por tanto parece apropiado tratar el entrenamiento de una red a través de la Programación Orientada a Objetos, implementando una clase para cada tipo de bloque.

El primer paso en el diseño de la herramienta consiste, pues, en decidir qué objetos básicos permiten construir una red neuronal de arquitectura arbitraria así como cuál es su comportamiento. Se ha optado por la implementación del conjunto de elementos mencionados con anterioridad más otros añadidos por conveniencia. El esquema de las acciones que realizan en la red original (cálculo de la salida) como en la recíproca (propagación de la derivada) se indican en la tabla J.1. En lo que sigue los términos *elemento antecesor* y *posterior* se refieren ambos a los elementos de la red original.

Aparte de estos elementos constituyentes de cualquier red neuronal arbitraria, y debido a su frecuente aparición en los diseños, se ha incluido el objeto `Neurona`, constituido por un sumador, un bloque de función de activación, un elemento unidad, un peso (estos dos elementos implementan el sesgo de la neurona) y un punto de bifurcación. Se muestra de forma esquemática en la Figura J.2.

Cuadro J.1.: Elementos de Red implementados en la herramienta Evenet2000

Elemento de Red	Red original	Red Recíproca
Neurona de Entrada	Entrada de la red.	Salida de la red.
Salida	Salida de la red.	Entrada de la red.
Sumador	Devuelve la suma de sus entradas.	Transmite a todos sus antecesores el término de error que le llega, a modo de punto de bifurcación.
Bloque Función de Activación	Devuelve a su salida el valor de la función de activación evaluada en su entrada.	Transmite a su antecesor el término de error multiplicando por la derivada de la función de activación evaluada en su entrada.
Punto de Bifurcación	Transmite el valor de su entrada a todos sus elementos sucesores.	Transmite a su antecesor en la red original la suma sus entradas.
Elemento Unidad	Elemento cuya salida es siempre la unidad. Su utilidad consiste en la implementación para el término del sesgo.	No propaga el error al no tener antecesores.
Peso	Devuelve a su salida el valor de la entrada multiplicado por el peso asignado.	Transmite a su antecesor el término de error multiplicado por el peso.
Retardo	Produce un retardo unidad.	Produce un avance unidad.

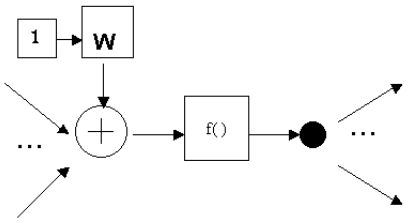

Figura J.2.: Representación en bloques de la clase *Neurona*.

Se debe hacer hincapié en que cualquier otra combinación de elementos es fácilmente implementable, sin más que crear los objetos correspondientes y conectarlos de manera adecuada.

J.2. Cálculo de la salida y propagación del error

Con lo señalado, se puede establecer una primera jerarquía de clases, derivando los tipos de objeto escogidos de una superclase llamada `ElementoRed`.

Tal como se ha recalcado con anterioridad, el objetivo último es conseguir una herramienta de cálculo lo suficientemente general y una librería potente de fácil uso. En aras de esa generalidad buscada, se define la clase `FuncionActivacion`. Con esta clase, el usuario no se verá restringido a utilizar las funciones proporcionadas por el programador, sino que podrá reutilizar el código creando nuevas funciones. Así, por ejemplo, no se tendrá que repetir el código de la clase `BloqueFuncionActivacion` cada vez que quiera cambiar la función de activación, simplemente crear un nuevo objeto `FuncionActivacion` y asignarlo al objeto `BloqueFuncionActivacion`.

Esta clase se declara abstracta, puesto que no se va a crear un objeto derivado directamente de esta clase. Aquí se definen los métodos comunes a todas las funciones de activación, que son:

- `dimeSalida`: proporciona el valor de salida de la función evaluada en el valor pasado como argumento.
- `dimeDerivada`: proporciona la derivada de la función de activación.
- `dimeDerivadaSalida`: proporciona la derivada de la función de activación aceptando como argumento la salida de la función.

A partir de esta clase heredan las clases `Sigmoide` y `TangenteHiperbolica`, implementadas por ser de las más utilizadas en las redes neuronales. Como se ha recalcado, la creación de nuevos objetos que hereden de la misma superclase es sencilla.

Volviendo a los elementos de red, las dos acciones básicas de los elementos deben ser el cálculo de la salida de la red y la derivada, con lo que el siguiente paso del diseño consiste en determinar el modo de relacionar estos objetos. Se ha optado por que cada objeto conozca cuáles son sus elementos antecesores. En el caso de

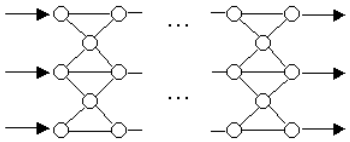

Figura J.3.: Esquema de Red.

los puntos de bifurcación además se debe conocer el número de elementos que le suceden en la red. En un primer intento cada uno de los bloques básicos de la red presenta dos atributos llamados `salida` y `derivada`, correspondientes al valor actual de la salida y el error propagado respectivamente. Del mismo modo se han incluido tres métodos: `dimeSalida`, `recibeDerivada` y `mandaDerivadaAnteriores`, que se analizan a continuación.

J.3. Cálculo de la salida de la red: el método `dimeSalida`.

El método `dimeSalida` proporciona el valor de la salida del elemento, invocando, en su caso, el método del mismo nombre de sus elementos antecesores en la red. Por ejemplo, en el caso del objeto `Neurona`, al invocar este método del punto de bifurcación, éste necesita el valor de la salida del bloque de función de activación, por lo que invocará el `dimeSalida` de éste. El bloque necesita a su vez la salida del sumador para el cálculo de su salida, por lo que la petición se propaga hasta llegar a elementos de la red sin antecesores (elementos unidad, neuronas de entrada). Una vez que se llega a este tipo de elementos, la salida es conocida (1 en el caso del elemento unidad y el valor correspondiente del par de entrenamiento para las neuronas de entrada), produciéndose el cálculo sucesivo de las salidas de los elementos de la red en un barrido hacia delante. Para visualizarlo de modo gráfico supóngase que se dispone de una red neuronal representada en la Figura J.3, y en la que cada elemento es representado por un círculo blanco.

Supóngase ahora que se desea determinar el vector de salida de la red. Para ello se invoca el método `dimeSalida` de los elementos `Salida` de la red. Al necesitar los valores de las salidas de sus elementos anteriores, el elemento se quedará en espera hasta que el método de los antecesores le devuelva el valor requerido. El proceso continúa hasta que se llega a elementos sin antecesores en la red. En la Figura J.4 se ilustra este proceso, representando con un círculo gris los elementos que se encuentran en espera y con una flecha las invocaciones al método `dimeSalida`.

Los elementos sin antecesores en la red tienen un valor definido de salida por lo que al ser invocado su `dimeSalida` devuelven directamente este valor a los elementos que se lo han requerido. De este modo, los sucesivos elementos de la red van realizando los cálculos necesarios para determinar sus respectivas salidas. Así se van terminando de ejecutarse los métodos. De modo gráfico se puede asemejar a un flujo hacia delante en las salidas. En la Figura J.5, se representa con un círculo negro aquellos elementos

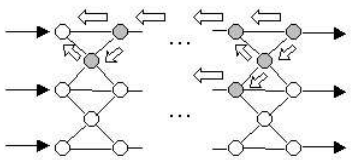

Figura J.4.: Cálculo de la salida. Flujo de peticiones.

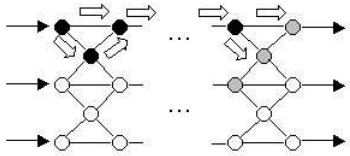

Figura J.5.: Envío de los valores de la salida.

cuya salida se encuentra actualizada (habiendo finalizado el método `dimeSalida`). En esta ocasión las flechas significan el envío de los valores de las salidas a los elementos posteriores a través del `return`.

Un elemento puede tener varios sucesores, por lo que para el mismo vector de entradas su método `dimeSalida` sería invocado varias veces, con lo que a su vez se invocarían nuevamente los métodos de sus elementos anteriores repitiéndose todo el proceso. De este modo se realizan varias veces los mismos cálculos lo cual, obviamente, es muy desfavorable desde el punto de vista computacional. Para solucionar esta circunstancia, se define una variable común a todos los elementos de red (declarada como `static` en la superclase `ElementoRed`) llamada `iteracionGlobal`. Esta variable indica el número de veces que se ha calculado el vector de salida de la red, mientras que cada elemento cuenta con una variable propia `iteracion`, que informa sobre las veces que ese elemento en cuestión ha invocado al `dimeSalida` de sus antecesores. Si la variable particular no coincide con la global (lo que significa que el elemento en cuestión no ha actualizado el valor de su salida) realiza las peticiones necesarias a sus antecesores, actualiza el valor de su salida y de su iteración particular y devuelve este valor a quien lo haya solicitado. En caso de que coincidan los dos valores, simplemente devuelve la salida sin realizar llamadas a método alguno. De este modo no se realizan cálculos innecesarios. Estas consideraciones se traducen en el siguiente esquema de método:

```
public double dimeSalida()
{if (iteracion != iteracionGlobal)
  {salida=...
   iteracion = iteracionGlobal;
  }
 return salida;
}
```

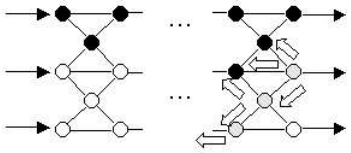

Figura J.6.: Cálculo de la salida. Nuevo flujo de petición.

Por tanto si ahora se desea calcular la salida de otro elemento `Salida`, el flujo de peticiones sería algo como lo representado en la Figura J.6.

Un caso particular lo constituye el elemento de retardo. Como se ha indicado, la función de este elemento es introducir un retardo unidad a su entrada. En primera instancia, en su implementación se añade a sus atributos un búffer donde se almacena el valor de la salida de su elemento antecesor en la red en el instante anterior y que es devuelto tras la invocación de su dimeSalida, sin que se realicen llamadas al método de su antecesor. El búffer se actualiza a través de la llamada al método `renovarElemento` de cada uno de los retardos al final de cada cálculo de la salida global, puesto que, de lo contrario, las salidas de sus antecesores podrían no estar actualizadas. Además, de esta manera, se evitan los bucles infinitos de llamada al `dimeSalida` en las realimentaciones [Gon00].

J.4. Propagación de los términos de la derivada: métodos `recibeDerivada` y `mandaDerivadaAnteriores`

Una vez calculadas todas las salidas, el siguiente paso es determinar los términos del gradiente de los pesos. Los encargados son los dos métodos `recibeDerivada` y `mandaDerivadaAnteriores`. Debido a que los operadores de retardo actúan como operadores de avance en la red recíproca, el sistema se vuelve no causal, por lo que los cálculos deben empezar por la última etapa (último par de entrenamiento), disminuyendo el índice de etapa. Así que antes de calcular el gradiente deben haber sido calculadas todas las salidas.

Para cada par de vectores de entrenamiento, se calcula el vector de error, definido a partir de la función de coste utilizada para el entrenamiento. Las componentes de este vector son pasadas a los elementos de salida correspondientes a través su método `recibeDerivada`.

Para los elementos con un único sucesor en la red, se actualiza el valor de su variable derivada (δ) y se invoca directamente a su método `mandaDerivadaAnteriores`. Este método llama a su vez a los métodos `recibeDerivada` de sus elementos antecesores en la red, suministrándoles el valor de δ. En aquellos elementos de red que no propaguen el término de la derivada al no tener antecesores, el método `recibeDerivada` se encuentra vacío. En los puntos de bifurcación, al tener varios sucesores, este método incluye un contador de las invocaciones, de tal modo que al recibir la derivada de alguno de sus sucesores, suma este valor a su variable `derivada`,

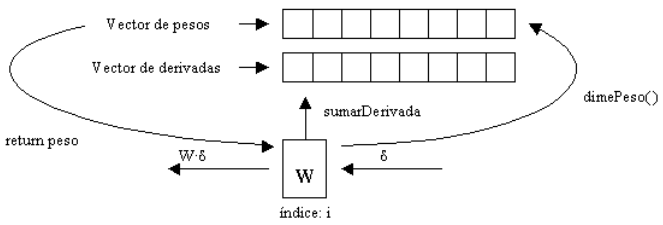

Figura J.7.: Vector de pesos y vector de derivadas de pesos.

actualiza el contador y comprueba si éste coincide con su número de sucesores. En caso afirmativo, pone el contador a 0 (preparándolo para próximas etapas) e invoca al método `mandaDerivadaAnteriores`.

A través de las invocaciones a estos métodos se consigue abarcar la red en un barrido hacia atrás, a semejanza del visto en el cálculo de la salida.

Hasta ahora, únicamente se ha visto cómo se transmite el valor de la derivada. Como es lógico, esta estructura debe calcular en algún momento las variaciones a aplicar sobre los pesos. Para ello se define en el objeto `Red` un vector global de pesos y un vector de derivadas de los pesos, de modo que los elementos `Peso` puedan acceder a él a través de un índice particular. Cada vez que se invoca el `recibeDerivada` de un `Peso`, se calcula la derivada que va a transmitir a sus elementos antecesores (multiplicando por el valor actual de la componente del vector de pesos correspondiente a su índice) y además la contribución a la derivada del coste respecto al peso en el par de entrenamiento actual, que se almacenará en la componente del vector de derivadas de los pesos correspondientes al índice del objeto `Peso` (Figura J.7).

La contribución a la variación del peso por cada par de entrenamiento es la derivada recibida por la salida del elemento anterior correspondiente a ese par. Para pedir este valor el objeto `Peso` invocará el método `dimeSalidaActual` del antecesor en la red. Cada vez que se realiza el barrido hacia atrás de cada par de entrenamiento, se suman las contribuciones al vector de derivadas. Es cuando se tienen calculadas las contribuciones de todos los pares cuando se realiza la actualización de los pesos, siguiendo el algoritmo de optimización escogido.

Como se ha mencionado anteriormente, y análogamente al caso descrito para el método `dimeSalida`, los elementos de retardo presentan algunas variaciones en el cálculo de su derivada para su propagación. Para un mayor detalle se puede consultar [Gon00].

Es importante reseñar que estos métodos se aplican en algoritmos que precisan del cálculo del gradiente. Esto no supone restricción alguna, puesto que si se utiliza un algoritmo que no lo necesite (por ejemplo Camino Aleatorio) los métodos para calcular la salida son igualmente válidos, mientras que los métodos de la derivada no serán invocados.

J.5. El objeto Red

Este objeto ha sido introducido en apartados anteriores. En esta sección será tratado en más profundidad. Es el encargado de realizar las operaciones relativas a la red: calcular la salida de un vector de entradas dado, calcular el gradiente de los pesos,... No obstante, buscando la mayor generalidad, y que la librería no sólo sirva para entrenamiento de redes neuronales, se hace heredar esta clase de una nueva superclase llamada `ObjetoEntrenable`. Un punto clave de este objeto es el vector de pesos. Como se ha indicado, a los objetos `Peso` se les debe haber asignado un índice correspondiente a la componente del vector de su peso. Con este índice puede acceder tanto a este vector de pesos como al vector de derivadas. Esto implica que los objetos `Peso` deben conocer no sólo su índice sino también el objeto `Red` al cual pertenecen para poder acceder a este vector.

El objeto `Red` implementado en la herramienta de cálculo ofrece al usuario diversos métodos relacionados con estos vectores, entre los que se encuentran:

- `fijarPesosIniciales`: fija como vector inicial de pesos para el entrenamiento (variable pesosIniciales) el pasado como argumento. El usuario puede acceder a este vector, por ejemplo, si quiere reproducir las condiciones iniciales para otro algoritmo de aprendizaje.
- `fijarPesos`: fija como vector actual de pesos del entrenamiento el vector pasado como argumento del método.
- `limpiarDerivadas`: pone a 0 las componentes del vector de derivadas para calcular la próxima actualización del vector de pesos.
- `dimePesos`: devuelve el vector de pesos actual del entrenamiento.
- `dimeDerivada`: devuelve el vector actual de derivadas de los pesos del entrenamiento.
- `inicializarPesos`: crea un vector de pesos aleatorio.

Como es lógico pensar, el objeto `Red` debe proporcionar unos métodos que calculen la salida a partir de un vector de entradas y que produzca la propagación del vector de error a través de los elementos de la red. Por este motivo, se debe analizar qué información precisa el objeto `Red` para realizar sus cometidos: debido a las conexiones escogidas (cada elemento de red conoce sus elementos anteriores), en principio bastaría con que el objeto `Red` conociese los elementos de salida. De este modo, cuando se pretenda calcular la salida de la red, el objeto `Red` invocaría el método `dimeSalida` de estos elementos. Sin embargo, queda el detalle de cómo imponer el vector de entradas a la red. Para resolver esto el objeto `Red` necesita asimismo conocer el conjunto de entradas a la red (las neuronas de entrada presentes en el sistema). Para fijar las entradas, se incluye un método `cambiaSalida` en los objetos `NeuronaEntrada` que fijan su valor de salida al valor pasado como argumento del método. Por lo tanto, el objeto `Red` conoce tanto las entradas como las salidas de la red. La forma escogida para ello es a partir de dos arrays: `vectorNeuronasEntrada` y `vectorSalidas`. Por otro lado, y recapitulando lo visto anteriormente sobre el

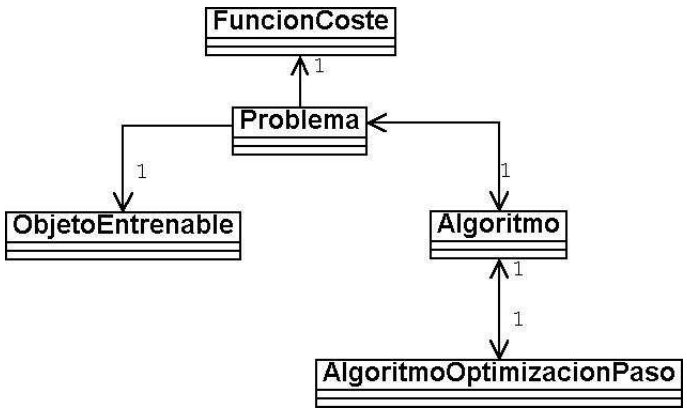

Figura J.8.: Estructura de un entrenamiento.

cálculo de la salida y la propagación del error a través de la red, un objeto `Red` debe conocer los retardos presentes en su estructura. Se ha definido de otro array con estos elementos (`listaDelays`). Los retardos deben ser actualizados al final del cálculo de la salida para cada par de entrenamiento (de esto se encarga un método denominado `renovarObjeto`) y sus derivadas actualizadas al principio de cada propagación del error (método `renovarDerivadasObjeto`).

Al ser el perceptrón multicapa uno de los tipos de red más frecuente, se ha construido la clase `MLP`, descendiente de la clase `Red`, por lo que el usuario de la herramienta de cálculo puede construir una red de este tipo de una manera sencilla, sin más que indicar el número de capas y las neuronas por cada capa.

J.6. Construcción de una librería de cálculo

Hasta el momento se ha elaborado una serie de clases que permiten emular una red neuronal con sus métodos para calcular su salida y propagar su vector de error a través de sus elementos. Sin embargo, el objetivo principal no es quedarse en este punto, sino construir una potente herramienta de cálculo con una librería lo suficientemente amplia para adaptarse a las necesidades del investigador de redes neuronales. Para ello, se debe elaborar un plan de integración de las clases implementadas hasta ahora en un conjunto más amplio. Tras un detallado análisis sobre el entrenamiento de una red neuronal, se concluyó que éste se puede estructurar de modo básico en los 5 bloques descritos en la Figura J.8: `Problema`, `Red`, `Algoritmo`, `Algoritmo de Optimización` y `Función de Coste`.

Se propone la existencia de un objeto `Problema` que sirve de nexo entre los objetos `Algoritmo` (que coordina todo el entrenamiento), `FuncionCoste` y `Red` (como `ObjetoEntrenable`). De este modo, los algoritmos implementados no se ven restringidos a ser utilizados únicamente en un entrenamiento de redes neuronales, sino que pueden ser utilizados en otras aplicaciones de modo independiente al resto de clases. De ahí que no se hable en `Algoritmo` de vector de pesos sino de vector de diseño que se pretende optimizar.

El objeto `Problema` debe conocer el objeto `Red` que se pretende entrenar, la función de coste utilizada (objeto `FuncionCoste`), y los patrones de entrenamiento. Un entrenamiento típico (por ejemplo siguiendo un algoritmo de Gradiente Descendente) constaría de los pasos descritos a continuación[2].

1. Una fase de inicialización de variables. En esta fase se generan los valores iniciales de los pesos (que para el objeto `Algoritmo` no es más que un vector de diseño que modificándolo, trata de reducir el valor de una función), el vector de derivadas de los pesos en la red y otros procesos necesarios antes del entrenamiento en sí.

2. Se pide al objeto `Algoritmo` que empiece a optimizar el vector de diseño (vector de pesos) hasta que se cumpla las condiciones de parada. El `Algoritmo` genera un nuevo vector de diseño. Para ello puede haber requerido el vector gradiente al objeto `Problema` y/o un valor adecuado del paso de entrenamiento a un objeto `AlgoritmoOptimizacionPaso` (que no es más que la implementación de un algoritmo de optimización unidimensional).

3. Una vez `Algoritmo` genera este vector pide a `Problema` el coste del nuevo vector. `Problema` entonces calcula la salida (le pide el valor al objeto `Red`) y el coste (presenta al objeto `FuncionCoste` el conjunto de salidas calculadas y los vectores de salida deseados y este objeto le devuelve el valor del coste). El sistema se encuentra listo ya para la siguiente iteración.

Como se ha indicado, el proceso de cálculo de los nuevos vectores de diseño y de su coste se repite hasta las condiciones de parada. Como se comprueba cada objeto realiza su función sin conocer más de la información que necesita para realizarlo: por ejemplo la red no sabe que está siendo entrenada ni el algoritmo ve la red en ningún momento. Esto permite una gran ventaja: la fácil reutilización del código, tan valorada en la OOP.

J.6.1. La clase `FuncionCoste`

La clase `FuncionCoste` presenta analogía con la `FuncionActivacion`. Se declara abstracta, ya que no se crean directamente objetos de esta clase, sino de sus clases derivadas. En este caso los métodos implementados son: `calcularCoste` (a partir de los valores actuales de la salida y los valores deseados se calcula el coste según la función escogida) y `generarVectorError` (devuelve el vector de error dadas las salidas actuales y deseadas). De ésta heredan la clase `Cuadratica`, que reproduce la función de coste cuadrática, y la clase `CuadráticaPonderada`, que incluye una ponderación en los coeficientes del sumatorio.

J.6.2. La clase `Algoritmo`

La clase `Algoritmo` se declara abstracta para reunir las características comunes a todos los algoritmos. Esta clase presenta como variables importantes el array de diseño

[2]Nótese que no es una clasificación general: en algunos entrenamientos no son necesarias todas las fases.

actual y el coste (error) de este vector. También contiene el vector de diseño que hasta el momento ha producido el menor coste (`vectorDisMinimo`) y el valor de ese mínimo (`errorMinimo`). La razón de la existencia de estas dos últimas variables es que existen algoritmos (por ejemplo el Enfriamiento Progresivo) en los que de una iteración a la siguiente no es inhabitual que el coste aumente.

La función de este objeto es optimizar el vector de diseño en cuanto a la función de coste asignada al `Problema`. Por esto el objeto `Algoritmo` debe conocer la existencia del objeto `Problema`, por lo que se precisa de la variable problema para poder comunicarse con el objeto y pedirle el valor del coste del vector de diseño actual (en algunos casos la comunicación puede incluir otros aspectos como el cálculo del gradiente).

De esta clase derivan una serie de subclases encargadas de implementar algunos de los métodos de entrenamiento vistos en H.4: `GradienteDescendente`, `GradienteConjugado`, `CaminoAleatorio`, `EnfriamientoProgresivo` y `AlgoritmoGenetico`.

J.6.3. La clase `AlgoritmoOptimizacionPaso`

En algunos casos el proceso de optimización se puede mejorar mediante la optimización, a su vez, del paso de entrenamiento. Este es el cometido de la superclase `AlgoritmoOptimizacionPaso` y todas sus subclases. En cada iteración el objeto generado a partir de una subclase de `Algoritmo` (concretamente `GradienteDescendente` o `GradienteConjugado`, que son los algoritmos para los cuales la optimización de paso cobra sentido), pide un valor óptimo del paso de entrenamiento al objeto derivado de alguna subclase de `AlgoritmoOptimizacionPaso`, que realiza los cálculos pertinentes y devuelve el valor optimizado del paso de entrenamiento.

En general, el proceso de optimización va a estribar en la estimación de un mínimo de la función de coste respecto al valor del paso de entrenamiento en la dirección calculada por `Algoritmo`, por lo que el problema de optimización del paso se reduce a minimizar una función unidimensional.

En este proceso los objetos `Algoritmo` y `AlgoritmoOptimizacionPaso` precisan intercambiar mensajes (mediante la invocación de sus métodos). Así, en cada iteración del proceso de optimización el `Algoritmo` pide al objeto `AlgoritmoOptimizacionPaso` el valor del paso de entrenamiento optimizado. Esta petición se realiza a través del método `optimizaPaso`. Lo más frecuente es que para calcular el paso de entrenamiento, el objeto `AlgoritmoOptimizacionPaso` precise estimar el coste de una serie de vectores de diseño de prueba. Para esta estimación de coste invoca al método `dimeCoste` del objeto `Algoritmo`, que a su vez enviará su petición a `Problema`.

De lo anterior se deriva que ambos objetos se deben *ver* para poder invocar sus respectivos métodos. Esto implica que el objeto `Algoritmo` presente un atributo `algoritmoPaso` y a la vez que el objeto `AlgoritmoOptimizacionPaso` contenga un atributo `algoritmo`.

Las subclases de esta clase implementan algunos de los algoritmos descritos en el apartado H.5: `PasoFijo` (paso de optimización constante), `AjusteParabolico` y `SeccionOro`.

J.6.4. La clase `Problema`

Como se mencionó anteriormente, esta clase tiene como cometido el servir de comunicación entre los objetos `Algoritmo` y `Red` (en el caso general `ObjetoEntrenable`). Podría parecer en un principio que, de modo análogo al descrito en las superclases anteriores, al servir de nexo debiera conocer la existencia de ambos objetos presentando dos atributos: `objetoProblema` y `algoritmo`. No obstante, choca de modo conceptual con la filosofía OOP y en concreto con la ocultación. El objeto `Problema` no requiere del objeto `Algoritmo`, sino a la inversa. Por tanto, `Problema` no posee el atributo `Algoritmo`.

Los otros parámetros que debe conocer este objeto son los patrones de entrenamiento (`paresEntrenamiento`, `entradaEntrenamiento` y `salidaEntrenamiento`) y la función de coste `funcionCoste` que se desea utilizar. Estos son los parámetros que fijan las condiciones del problema de entrenamiento.

En cuanto a los métodos que aparecen en esta superclase, éstos pueden dividirse de manera clara en dos grupos: aquellos métodos de inicialización de los parámetros del problema (por ejemplo el vector de pesos) y de petición de datos al objeto `Problema`.

J.6.5. Ejemplo de entrenamiento

Con lo visto hasta ahora el lector puede hacerse una idea de cómo utilizar la librería de cálculo. Sin embargo, y a fines de aclarar más aún este apartado, se describe a continuación un ejemplo de entrenamiento de una red MLP (perceptrón multicapa) mediante un algoritmo de Gradiente Descendente, un algoritmo de optimización del paso de entrenamiento Sección de Oro, y una función de coste Cuadrática.

En primer lugar, se deben crear los 5 objetos que se van a relacionar para el entrenamiento: *ObjetoEntrenable*, *Algoritmo*, *AlgoritmoOptimizacionPaso*, *Problema* y *FuncionCoste*. Para ello se invoca al constructor de la clase correspondiente con los argumentos adecuados. En el caso que nos ocupa el objeto a entrenar es un MLP. Consultando la documentación de la herramienta, se comprueba que los parámetros de invocación son:

```
  public MLP (int numeroCapas, int[] neuronasPorCapa,
FuncionActivacion funcionActivacion) {
```

por lo tanto se necesita crear un vector indicando el número de neuronas por cada capa y el objeto `funcionActivacion`. Si por ejemplo, la función de activación escogida es la sigmoide, se debe crear previamente este objeto, y por lo tanto se debe investigar el constructor del objeto `Sigmoide`.

```
  public Sigmoide ()
```

Este constructor no admite argumentos, por lo que se puede crear directamente. En el caso que se investiga, se desea entrenar una red MLP con tres capas, con 3 neuronas de entrada, 4 en la capa intermedia y 3 en la salida. La creación del objeto correspondiente debe presentar un código similar a este:

```
/* inicialización del número de capas */
int numeroCapas = 3;
/* vector del número de neuronas por capa */
int[] neuronasPorCapa = new int[numeroCapas];
neuronasPorCapa={3,4,3};
/* función de activación */
Sigmoide funcion= new Sigmoide();
/* MLP */
MLP redEntrenable = new MLP (numeroCapas,neuronasPorCapa,funcion);
```

Con estas sencillas líneas se obtiene un MLP de las dimensiones señaladas con todos sus elementos (neuronas, pesos,...). En el caso de una red más general se deberían crear los elementos particulares, conectarlos y hacerlos depender del objeto `Red`. Basta con comunicar a este objeto las entradas, las salidas y los elementos de retardo de la red construida. Cada objeto `Peso` se construiría con el índice que le permitiese acceder al vector de pesos global. En nuestro caso, todas estas operaciones vienen incluidas en el constructor del MLP.

Una vez creada la red, se puede crear el objeto `Problema`. El constructor tiene la forma:

```
public Problema(ObjetoEntrenable objetoProblema, double[][]
paresEntrenamiento, FuncionCoste funcionCoste) {
```

Suponiendo un caso sencillo con dos pares de entrenamiento con la siguiente forma:

1,1,1—0,1,0
1,0,0—1,1,0

se ubican estos datos en un array bidimensional de 2x6 (2 pares de entrenamiento, 6 datos por par de entrenamiento) nombrándolo, por ejemplo, `vectorPatrones`. La función de coste escogida es la cuadrática (constructor sin argumentos, con lo que de paso se construye el objeto `FuncionCoste`). Un posible código es:

```
/* inicializar el vector de patrones */
double[] vectorPatrones = new double[2][6];
/* asignar las componentes del vector */
vectorpatrones={ {1,1,1,0,1,0} , {1,0,0,1,1,0} };
/* creación de la función de coste */
Cuadratica funcionCoste= new Cuadratica();
Problema problema= new Problema(redEntrenable,vectorPatrones,
funcionCoste);
```

El siguiente paso es construir los objetos `Algoritmo` y `AlgoritmoOptimizacionPaso`, que en este caso corresponden a un `GradienteDescendente` y a una optimización a través de `SeccionOro`. Como siempre el usuario debe tener presente los constructores de los objetos. Así para el

GradienteDescendente, se tiene:

```
  public GradienteDescendente (Problema problema, double[]
vectorDis )
```

Por lo tanto, deben haber sido creados el problema (lo cual ya está hecho) y un vector de diseño inicial. La mayoría de las ocasiones se utiliza un vector con valores aleatorios. Si es éste el caso, el usuario puede crearlo análogamente al vector de patrones, pero con una serie de inconvenientes: por ejemplo debería saber la dimensión del vector de pesos, lo que requiere una serie de cuidadosos cálculos. Además debe tener en cuenta que ese vector de diseño inicial del Algoritmo debe coincidir con el vector de pesos inicial de Red, con lo que debe pasarlo también a este objeto a través del método fijoVector de Problema. Sin embargo, existe otra estrategia más simple: inicializar primero el vector de pesos de Red y después pasarlo como argumento del constructor de Algoritmo. En el caso analizado se desea inicializar el vector con pesos aleatorios comprendidos entre -1 y 1, y con la dimensión adecuada. La herramienta de cálculo implementa diversos métodos para inicializar los pesos. Una posibilidad sería:

```
  /* inicialización del vector de pesos de la red */
  problema.inicializarPesos();
  /* construir el objeto Algoritmo */
  GradienteDescendente algoritmo = new GradienteDescendente(problema,
problema.dimeVectorPesos());
```

Además, al ser el Gradiente Descendente un algoritmo que necesita un vector de derivadas, es necesario inicializar este vector.

```
  /* inicialización del vector de derivadas de la red */
  red.iniciarDerivadas();
```

Una vez creado el Algoritmo, queda por construir el AlgoritmoOptimizacionPaso, y en concreto uno que implemente el algoritmo Sección de Oro. Una vez más se analiza el constructor correspondiente.

```
  public SeccionOro(Algoritmo algoritmo)
```

Por lo que la construcción es simple. Tal como se indicó el objeto Algoritmo debe conocer la existencia de este objeto para poder requerirle el valor del optimizado del paso de entrenamiento. Esto se consigue mediante la invocación del método fijarAlgoritmoPaso. En caso de no invocarlo, el algoritmo de optimización por defecto es PasoFijo (no se optimiza el paso de entrenamiento). El código resultante es:

```
  /* construcción del objeto SeccionOro */
  SeccionOro algoritmoPaso = new SeccionOro(algoritmo);
```

```
  /* indicar al algoritmo qué algoritmo de optimización de paso se
está utilizando */
  algoritmo.fijarAlgoritmoPaso(algoritmoPaso);
```

Con estas líneas se han construido los objetos que intervienen en el entrenamiento y se encuentran preparados para los cálculos. Recapitulando, los pasos seguidos hasta ahora han sido:

1. Crear el objeto `Red`. Para ello, en el caso más general se deben crear objetos previos (elementos de red, funciones de activación...) y relacionarlos con este objeto (por ejemplo a través del constructor).
2. Crear el objeto `FuncionCoste`.
3. Crear el objeto `Problema`. Para ello se necesita haber creado el objeto a entrenar (paso 1), la función de coste a usar (paso 2) y un array con los pares de entrenamiento de dimensiones [número de pares][número de entradas + salidas].
4. Crear el objeto `Algoritmo`. Su constructor necesita un vector de diseño que coincida con el vector inicial de pesos del objeto a entrenar. Este vector puede ser generado de múltiples formas. En el caso de un vector con pesos aleatorios se invoca el método `inicializarPesos` del objeto `Problema`. Asimismo requiere un objeto `Problema` (paso 3).
5. Crear el objeto `AlgoritmoOptimizacionPaso`. Su constructor acepta como argumento el objeto `Algoritmo` (paso 4).
6. Relacionar los objetos `Algoritmo` y `AlgoritmoOptimizacionPaso`. Para ello se invoca al método `fijarAlgoritmoPaso` de este último objeto.

Ya inicializados todos los objetos que forman parte del entrenamiento, éste puede comenzar. El usuario debe invocar al método `comienzaOptimiza` del objeto `Algoritmo`. En el caso analizado se trata de un `GradienteDescendente`, cuyo método es de la forma

```
  public void comienzaOptimiza (double errorDeseado, int
iteracionesMaximas, double learningRateInicial) {
```

Este método fija las condiciones del entrenamiento y del fin de su ejecución. El entrenamiento se detiene cuando se obtenga un coste menor que el deseado o se alcance el número máximo de iteraciones. Suponiendo un coste máximo deseado de 0.001, un número máximo de iteraciones de 1000, y un valor inicial del paso de entrenamiento de 0.25 (aunque este último será optimizado por el objeto `SeccionOro`), el usuario de la librería únicamente debería incluir la siguiente línea de código:

```
  /* comienza el entrenamiento */
  algoritmo.comienzaOptimiza(0.001,1000,0.25);
```

El usuario, gracias a la OOP, no conoce realmente cómo funciona este método; simplemente sabe que cuando termine podrá acceder a un vector que le indica la evolución del coste con el número de iteraciones y a un vector de diseño que cumple con las especificaciones requeridas.

Una vez finalizado el aprendizaje, el usuario estará interesado en saber los resultados del entrenamiento (vector de pesos que hace mínimo el coste, evolución de la función de coste...). La herramienta de cálculo permite acceder a estos aspectos mediante una serie de métodos ya referenciados. Por ejemplo para obtener el vector de pesos y la evolución del error (el `Algoritmo` almacena los valores del coste en un vector), el código debe ser:

```
  /* obtención del vector de pesos de mínimo coste*/
  double[] pesosEntrenamiento = algoritmo.dimeVectorDisMinimo();
  /* obtención del coste mínimo durante el entrenamiento */
  double errorMinimo = algoritmo.devuelveErrorMinimo();
  /* obtención de la evolución del error */
  double[] vectorError = algoritmo.evolucionError();
```

Con esto, el usuario habrá realizado el entrenamiento deseado. El código completo de este ejemplo es:

```
  /* inicialización del número de capas */
  int numeroCapas = 3;
  /* vector del número de neuronas por capa, de tres elementos */
  int[] neuronasPorCapa = new int[numeroCapas];
  neuronasPorCapa= {3,4,3};
  /* función de activación */
  Sigmoide funcion= new Sigmoide();
  /* MLP */
  MLP redEntrenable = new MLP (numeroCapas,neuronasPorCapa,funcion);
  /* inicializar el vector de patrones */
  double[] vectorPatrones = new double[2][6];
  /* asignar las componentes del vector */
  vectorpatrones={ {1,1,1,0,1,0} , {1,0,0,1,1,0} };
  /* creación de la función de coste */
  Cuadratica funcionCoste= new Cuadratica();
  Problema problema= new Problema(redEntrenable,vectorPatrones,
funcionCoste);
  /* inicialización del vector de pesos de la red */
  problema.inicializarPesos();
  /* construir el objeto Algoritmo */
  GradienteDescendente algoritmo = new GradienteDescendente(problema,
problema.dimeVectorPesos());
  /* inicialización del vector de derivadas de la red */
  redEntrenable.iniciarDerivadas();
  /* construcción del objeto SeccionOro */
  SeccionOro algoritmoPaso = new SeccionOro(algoritmo);
```

```
  /* indicar al algoritmo qué algoritmo de optimización de paso se
está utilizando */
  algoritmo.fijarAlgoritmoPaso(algoritmoPaso);
  /* comienza el entrenamiento */
  algoritmo.comienzaOptimiza(0.001,1000,0.25);
  /* obtención del vector de pesos de mínimo coste*/
  double[] pesosEntrenamiento = algoritmo.dimeVectorDisMinimo();
  /* obtención del coste mínimo durante el entrenamiento */
  double errorMinimo = algoritmo.devuelveErrorMinimo();
  /* obtención de la evolución del error */
  double[] vectorError = algoritmo.evolucionError();
```

J.6.6. Inclusión del *Real Time Recurrent Learning* (RTRL)

Con los elementos vistos hasta ahora se pueden realizar entrenamientos basados en la Backpropagation y Backpropagation Through Time. Sin embargo, el objetivo principal de esta librería es ser lo más general posible y ofrecer a los usuarios un amplio surtido de algoritmos. Esta es la justificación de la inclusión del RTRL en esta librería. Además, esto permite analizar si esta estructura de objetos es lo suficientemente versátil para admitir ampliaciones.

El primer paso consiste en comprobar si este esquema de problema choca o no contra la filosofía básica de estructuración implementada. La respuesta es no. El entrenamiento va a constar de los mismos objetos básicos.

Los problemas empiezan a surgir cuando se investiga si los métodos implementados para la Backpropagation y la Backpropagation Through Time son válidos para el RTRL. El método de actuar para propagar el error en estos algoritmos es calcular todas las salidas de la red para todas las etapas y después retroceder en las etapas para calcular la propagación del error. Recuérdese que el motivo para esto es que en la red recíproca los operadores de retardo son sustituidos por operadores de avance, por lo que el sistema se convierte en no causal. Para la propagación del error es necesario haber calculado el error correspondiente de las etapas posteriores. Esto no ocurre en el RTRL, sino todo lo contrario. Para calcular los términos p_{ji}^{h} de una determinada etapa se deben haber calculado los de la etapa anterior. Esto supone, entre otras repercusiones, que el método `calculaGradiente` del objeto `Problema` deba cambiar bastante. Ahora el proceso debe consistir en calcular la salida de una etapa y acto seguido calcular el gradiente, que en este caso se resume en determinar los coeficientes p_{ji}^{h}. Para resolver este aspecto y otros relacionados con la modificación de métodos de este objeto se harán uso de dos propiedades de la OOP, la herencia y el polimorfismo. Se crea un nuevo objeto que herede de `Problema` y que sobrescriba los métodos que es necesario modificar. Este objeto se denomina `ProblemaRTRN`. De nuevo, se remite al lector interesado en un mayor detalle a [Gon00].

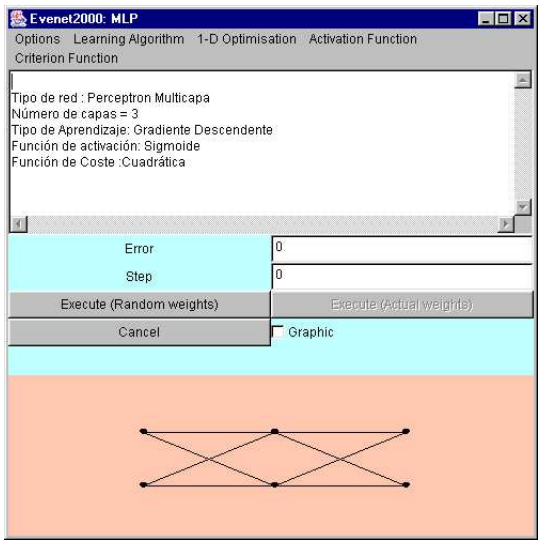

Figura J.9.: Marco para el entrenamiento de un perceptrón multicapa en la herramienta Evenet2000.

J.7. El paquete `interfase`

El trabajo realizado no se ha limitado a la construcción de la librería de cálculo sino que se ha dotado a la herramienta de un entorno gráfico que simplifique su utilización. De este modo se puede realizar el entrenamiento sin necesidad de programar código.

Existe una clase, llamada `Evenet2000`, cuya misión es mostrar una ventana principal con el logotipo de la herramienta, inicializando todo el entorno gráfico del programa. Todas las ventanas de este entorno derivan jerárquicamente del objeto `java.awt.Frame`. Estas clases, por su analogía, parece adecuado que deriven de una superclase común, denominada `MarcoSecundario`. Las clases que heredan de esta superclase son:

- `AbrirMLP`: abre la ventana correspondiente a la construcción y entrenamiento de un perceptrón multicapa, y representa el perceptrón que se va a entrenar (Figura J.9). Su barra de menús permite seleccionar los diversos parámetros del entrenamiento como por ejemplo los criterios de parada, la función de activación de las neuronas o la función de coste a minimizar.

- `AbrirRecurrente`: abre la ventana análoga para una red que se pretende entrenar mediante RTRL.

Aparte de estas clases principales, existen dos familias de clases de ventanas relativas al cambio de parámetros de entrenamiento y mostrar los resultados del entrenamiento, como por ejemplo el conjunto de pesos y la gráfica de evolución del error (Figura J.10).

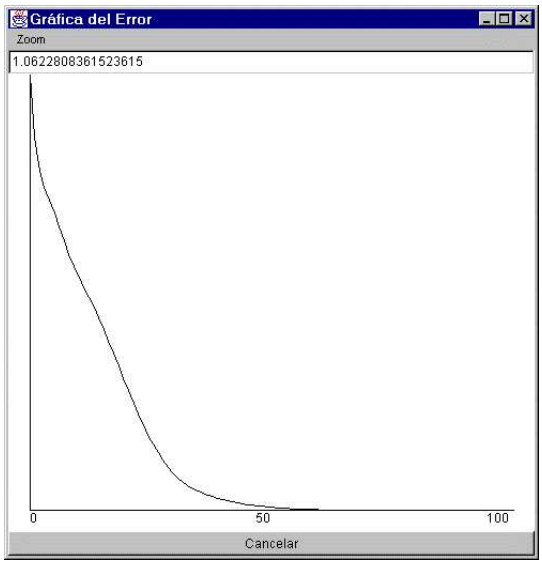

Figura J.10.: Gráfica de evolución del error de entrenamiento de una red neuronal en la herramienta Evenet2000.

J.8. El paquete `editorGrafico`

Con las clases descritas en el paquete `interfase`, el usuario puede realizar de manera cómoda el entrenamiento de una perceptrón multicapa o de una red recurrente. Sin embargo, queda oculta su gran ventaja potencial: la posibilidad de entrenar una red neuronal de arquitectura arbitraria. Por tanto, se hace patente la necesidad de dotar a la herramienta de un módulo en el que el usuario pueda diseñar fácilmente su propia red y entrenarla con la misma comodidad que un perceptrón multicapa o una red recurrente.

De entre las opciones posibles se estimó que lo más adecuado era la elaboración de un editor gráfico (ver Figura J.11) donde se representasen mediante iconos los elementos básicos de una red (aquellos derivados de la superclase `ElementoRed` en el paquete de cálculos) y que se pudiesen interconectar, formando así una red que posteriormente debería poder ser entrenada.

La clase principal de este paquete es `EditorGrafico`, y que muestra una ventana donde el usuario puede situar los iconos de cada elemento, moverlos, conectarlos, salvar el diseño en un fichero de texto,...

En este diseño no se abandonaron los principios de la OOP. Así, a cada uno de los posibles componentes de una red se le asignó una clase que representaba su icono en el editor gráfico. De este modo que se definieron las clases `IconoNeurona`, `IconoSumador`, `IconoSalida`... El usuario puede colocar este elemento en la posición deseada, mediante el arrastre de ratón. Al mover este icono, las variables del objeto referidas a sus coordenadas en la ventana cambian de modo automático (aún cuando el usuario no haya terminado de arrastrarlo por la pantalla). En caso de que el usuario quiera activar otro icono (por ejemplo para moverlo) basta con pulsar el botón del ratón sobre ese icono. Si el usuario desease eliminar el icono activado lo podría hacer a través del botón de *pop-up* del ratón.

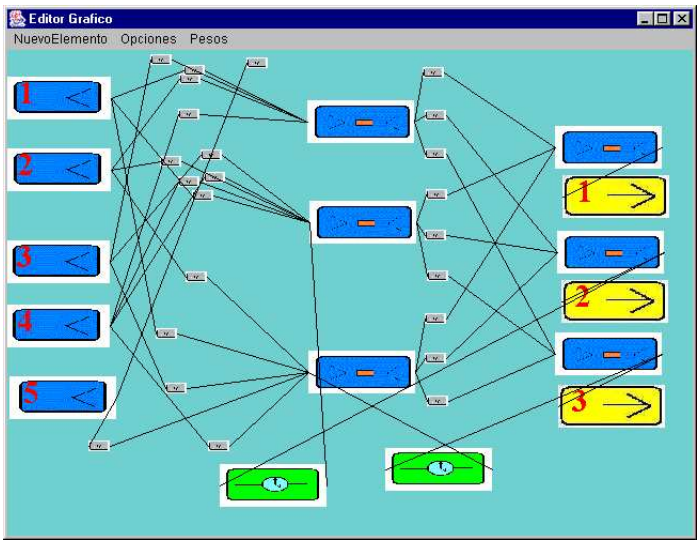

Figura J.11.: Editor Gráfico de la herramienta Evenet2000.

Existen elementos de red que presentan características que el usuario debería ser capaz de fijar. Por ejemplo un bloque de función de activación necesita conocer su función. Esto se traduce en que a partir de los iconos `IconoNeurona` e `IconoBloque`, el usuario puede seleccionar su función de activación a través de una lista desplegable.

Una vez diseñada le red en la pantalla se puede grabar en un fichero. Se ha optado por un fichero de texto para facilitar su modificación y depuración. Una de las mejoras posibles consistiría en el almacenamiento de la información en formato XML[3]. Se ofrece al usuario la posibilidad de utilizar una red grabada como un único bloque. De esta forma se pueden definir nuevos elementos de red a partir de los existentes y utilizarlos en nuevos diseños. A modo de ejemplo se muestra en la Figura J.12, una red de cinco entradas y tres salidas, cuya estructura se encuentra grabada en un fichero de nombre *Prueba13.red*.

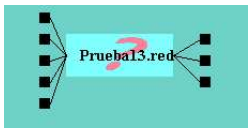

Figura J.12.: Ejemplo de utilización de una red grabada como bloque en el editor gráfico de la herramienta Evenet2000.

Tras el diseño de la red, se ha dotado a la herramienta de una nueva ventana llamada `AbrirRedFichero`, derivada de la clase `MarcoSecundario` e incluida en el paquete `interfase`, y que permite traducir la red grabada a un objeto entrenable. De

[3]De hecho, a fecha de escritura de este trabajo existe un proyecto ofertado para los estudiantes de la Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna

este modo, el usuario puede diseñar una red neuronal con arquitectura arbitraria y entrenarla, sin necesidad de programar ninguna clase de código.

Bibliografía

[ABH+02] Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. DAML-S: Web Service Description for the Semantic Web. In *The First International Semantic Web Conference (ISWC)*, Sardinia (Italia), Junio 2002.

[AG97] K. Arnold and J. Gosling. *El Lenguaje de Programación Java*. Ed. Addison-Wesley, 1997.

[ALE] ALECOP. *Control de Procesos Industriales CPI-100*. ALECOP.

[Alo02] E. Alonso. AI and agents: State of the art. *Artificial Intelligence Magazine*, pages 25–29, 2002.

[AM00] D. García Alonso and J. Pavón Mestras. Introducción al estándar FIPA. Technical Report UCM-DSIP 98-00, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Febrero 2000.

[BBD+00] Sean Bechhofer, Jeen Broekstra, Stefan Decker, Michael Erdmann, Dieter Fensel, Carole Goble, Frank van Harmelen, Ian Horrocks, Michel Klein, Deborah McGuinness, Enrico Motta, Peter Patel-Schneider, Steffen Staab, and Rudi Studer. An informal description of Standard OIL and Instance OIL. Technical report, Noviembre 2000.

[BCTR03] Fabio Bellifemine, Giovanni Caire, Tiziana Trucco, and Giovanni Rimassa. JADE Programmer's Guide. Technical report, CSELT, Febrero 2003.

[BDBW97] Jeffrey M. Bradshaw, Stewart Dutfield, Pete Benoit, and John D. Woolley. KAoS: Toward an industrial-strength generic agent architecture. In J. M. Bradshaw, editor, *Software Agents*. AAAI Press/The MIT Press, Cambridge, MA, 1997.

[BdlCT97] M.V. Belmonte, J.L. Pérez de la Cruz, and F. Triguero. Interaccion entre agentes: Utilidad, Coaliciones y Negociacion. *Revista Iberoamericana de Inteligencia Artificial.*, (6):57–67, 1997.

[Bec] V.M. Becerra. Advanced System Identification (4/CY/O8). http://www.personal.rdg.ac.uk/ shs99vmb/notes/asi/Lecture5.pdf.

[Bec02] Dave Beckett. Expressing Simple Dublin Core in RDF/XML, Julio 2002.

[BHGS01] Sean Bechhofer, Ian Horrocks, Carole Goble, and Robert Stevens. OilEd: A Reason-able Ontology Editor for the Semantic Web. *Lecture Notes in Computer Science*, 2174:376–385, 2001.

[BLS03] A. Bostan, Gregoire Lecerf, and Eric Schost. Tellegen's Principle into Practice. In *ISSAC 2003 (ACM)*, 2003.

[BM99] C. Baumer and Thomas Magedanz. Grasshopper - A Mobile Agent Platform for Active Telecommunication. In Sahin Albayrak, editor, *Intelligent Agents for Telecommunication Applications, Third International Workshop, IATA '99*, pages 19–32, Estocolmo, Suecia, Agosto 1999. Springer.

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE – A FIPA-compliant agent framework. In *Proceedings of PAAM'99*, pages 97–108, Abril 1999.

[BRK94] N. Bidstrup, H. Rasmussen, and T. Knudsen. Self-Tuning Speed Regulator for CVC Induction Motor Drive. In *3rd IEEE Conf. on Control Applications*, Glasgow, Agosto 1994.

[Bru91] J. C. Brustolini. Autonomous agents: characterization and requirements. Technical Report CMU-CS-91-204, Carnegie-Mellon University, 1991.

[Bry92] Martin Bryan. An Introduction to the Standard Generalized Markup Language (SGML), 1992.

[CB98] Deepika Chauhan and Albert D. Baker. JAFMAS: A Multiagent Application Development System. In Katia P. Sycara and Michael Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 100–107, St. Paul, Minnepolis, USA, Mayo 1998. ACM Press, New York.

[CFF+92] H. Chalupsky, T. Finin, R. Fritzson, D. McKay, S. Shapiro, and G. Weiderhold. An overview of KQML: A knowledge query and manipulation language. Technical report, KQML Advisory Group, Abril 1992.

[CFL+98] R. S. Cost, T. Finin, Y. Labrou, X. Luan, Y. Peng, I. Soboroff, J. Mayfield, and A. Boughanam. Jackal: A Java-based Tool for Agent Development. In *AAAI-98, Workshop on Tools for Agent Development*, Madison, WI, 1998.

[CG89] N. Carriero and D. Gelertner. Linda in context. *CACM*, 32(4):444–458, Abril 1989.

[CGK+01] H. Chalupsky, Y. Gil, C.A. Knoblock, K. Lerman, J. Oh, D.V. Pynadath, T.A. Russ, and M. Tambe. Electric Elves: Applying agent technology

to support human organizations. In *Proceedings of the Thirteenth Innovative Applications of Artificial Intelligence Conference (IAAI-01)*, pages 51–58, Menlo Park, 2001. AAAI Press.

[Cha00] Pierre-Antoine Champin. RDF tutorial. Technical report, http://www710.univ-lyon1.fr/ champin/rdf-tutorial/, Junio 2000.

[CJ96] D. Cockburn and N. R. Jennings. ARCHON: A Distributed Artificial Intelligence System for Industrial Applications. In G. M. P. O'Hare and N. R. Jennings, editors, *Foundations of Distributed Artificial Intelligence*. John Wiley & Sons, 1996.

[CJ03a] Roger D. Costello and David B. Jacobs. XML Design (A Gentle Transition from XML to RDF), 2003.

[CJ03b] Roger L. Costello and David B. Jacobs. OWL Web Ontology Language, 2003.

[CL95] Philip R. Cohen and Hector J. Levesque. Communicative Actions for Artificial Agents. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, pages 65–72, San Francisco, CA, USA, 1995. The MIT Press: Cambridge, MA, USA.

[CMUP97] P. Campolucci, A. Marchegiani, A. Uncini, and F. Piazza. Signal-Flow-Graph Derivation of on-line gradient learning algorithms. In *Proc. ICNN-97*, pages 1884–1889, Houston, June 1997.

[Coe95] M. Coen. SodaBot: A Software Agent Construction System, 1995.

[Com02] DAML Joint Committee. DAML Query Language (DQL) , Agosto 2002.

[Cov98] Stefan Covaci. The First Reference Implementation of the OMG MASIF Mobile Agent System Interoperability Facility, 1998.

[CvHH+01] Dan Connolly, Frank van Harmelen, Ian Horrocks, Deborah McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Annotated DAML+OIL Ontology Markup. Technical report, W3C, Diciembre 2001.

[CyC97] Cyc Ontology Guide. Technical report, Cycorp, Agosto 1997.

[DCM03] DCMI. An Overview of the Dublin Core Metadata Initiative, 2003.

[dKLW98] M. d'Inverno, D. Kinny, M. Luck, and M. Wooldridge. A formal specification of dMARS. In M. Singh, A. Rao, and M. Wooldridge, editors, *Intelligent Agents IV: Proceedings of the Fourth International Workshop on Agent Theories, Architectures, and Languages*, pages 155–176. Springer-Verlag, 1998.

[DLL3] Darryl N. Davis, Yuan Luo, and Kecheng Liu. Combining KADS with Zeus to Develop a Multi-Agent E-Commerce Application. In 3-4, editor, *International Journal of Electronic Commerce Research*. 315-335, 2003 3.

[DMM00] Stefan Decker, Prasenjit Mitra, and Sergey Melnik. Framework for the Semantic Web: An RDF Tutorial. *IEEE Internet Computing*, 4(6):68–73, 2000.

[DPD95] M. Dobrijevic, J.P. Parisot, and I. Dutour. A study of chemical systems using Signal Flow Graph theory: Application to Neptune. *Advances in Space Research*, 16(2):105, 1995.

[Eck98] Bruce Eckel. *Thinking in Java*. Ed. Prentice Hall, 1998.

[Edw97] P. Edwards. Intelligent Agents = Learning Agents. In J. L. Nealon and N. S. Taylor, editors, *Proceedings of the UK Intelligent Agents Workshop*, pages 157–161, Oxford, 1997. SGES Publications.

[Ekl01] Niklas Eklund. An Information Retrieving Framework for Multi-Agent Systems. Master's thesis, Department of Microelectronics and Information Technology. Royal Institute of Technology., Stockholm, Sweden, Junio 2001.

[Fee00] Feedback. PROCON Process Control Training Systems. http://www.fbk.com/full/products/pdfs/process/proconcts.pdf, 2000.

[FFMM93] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML – A Language and Protocol for Knowledge and Information Exchange. In *International Conference on Building and Sharing of Very Large-Scale Knowledge Bases*, Tokyo, Diciembre 1993.

[FFMM94] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.

[FG96] S. Franklin and A. Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. In *Intelligent Agents III. Agent Theories, Architectures and Languages (ATAL'96)*, volume 1193, Berlin, Germany, 1996. Springer-Verlag.

[FIP96] FIPA Rationale. Technical report, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, 1996.

[FIP01a] FIPA ACL Message Structure Specification. Technical Report XC00061E, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01b] FIPA Agent Management Specification. Technical Report XC00023H, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Octubre 2001.

[FIP01c] FIPA Brokering Interaction Protocol Specification. Technical Report XC00033F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01d] FIPA Contract Net Interaction Protocol Specification. Technical Report XC00029F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01e] FIPA Dutch Auction Interaction Protocol Specification. Technical Report XC00032F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01f] FIPA English Auction Interaction Protocol Specification. Technical Report XC00031F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01g] FIPA Iterated Contract Net Interaction Protocol Specification. Technical Report XC00030F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01h] FIPA Ontology Service Specification. Technical Report XC00086D, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01i] *FIPA-OS Developers Guide*. http://fipa-os.sourceforge.net/tutorials.htm, 2001.

[FIP01j] FIPA Personal Assistant Specification. Technical Report XC00083B, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01k] FIPA Personal Travel Assistance Specification. Technical Report XC00080B, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01l] FIPA Query Interaction Protocol Specification. Technical Report XC00027F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Agosto 2001.

[FIP01m] FIPA Recruiting Interaction Protocol. Technical Report -XC00034F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Enero 2001.

[FIP02a] FIPA Abstract Architecture Specification. Technical Report SC00001L, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FIP02b] FIPA Agent Message Transport Service Specification. Technical Report SC00067F, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FIP02c] FIPA Communicative Act Library Specification. Technical report, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, SC00037J, Diciembre 2002.

[FIP02d] FIPA Content Language Specification. Technical Report SC00008I, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FIP02e] FIPA Proposal for Borda Count Interaction Protocol. Technical Report f-in-00092, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FIP02f] FIPA Propose Interaction Protocol Specification. Technical Report XC00036G, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Noviembre 2002.

[FIP02g] FIPA Request Interaction Protocol Specification. Technical Report SC00026H, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FIP02h] FIPA Subscribe Interaction Protocol Specification. Technical Report SC00035H, FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS, Diciembre 2002.

[FL99] Tim Finin and Yannis Labrou. Tutorial on Agent Communication Languages. First International Symposium on Agent Systems and Applications and the Third International Symposium on Mobile Agents, ASA/MA'99, Octubre 1999.

[Flo03] Adina Magda Florea. Negotiation techniques. Technical report, University Politehnica of Bucarest, 2003.

[FM99] Roberto A. Flores-Mendez. Towards a Standardization of Multi-Agent System Frameworks, 1999.

[FMT97] Tim Finin, James Mayfield, and Chelliah Thirunavukkarasu. A Security Architecture for Agent Communication Language, 1997.

[FNJS00] P. Faratin, P. Buckle N. Jennings, and C. Sierra. Automated Negotiation for Provisioning Virtual Private Networks using FIPA-Compliant Agents. In *The Fifth International Conference and Exhibition on the Practical Application Of Intelligent Agents And Multi-Agent Technology (PAAM-2000),*, pages 185–202, Manchester, UK, 2000.

[Fon01] C. Mosquera Fontán. Análisis y estudio experimental de herramientas basadas en agentes. Master's thesis, Departamento de Tecnología de la Información y las Comunicaciones, Facultad de Informática, Universidade Da Coruña, Julio 2001.

[FOSG97] J. M Fonseca, E. de Oliveira, and A Steiger-Garção. Multi-Agent Negotiation algorithms for resources cost estimation: A case study. In *Proceedings do Encontro Português de Inteligência Artificial EPIA'97*, Octubre 1997.

[FSJ98] P. Faratin, C. Sierra, and N. R. Jennings. Negotiation Decision Functions for Autonomous Agents. *Int. Journal of Robotics and Autonomous Systems*, 24(3-4):159–182, 1998.

[FWW+93] Tim Finin, Jay Weber, Gio Wiederhold, Mike Genesereth, Don McKay, Rich Fritzson, Stu Shapiro, Richard Pelavin, and Jim McGuire. Specification of the KQML Agent-Communication Language, 1993.

[GAH+03] V. Gyurjyan, D. Abbott, G. Heyes, E. Jastrzembski, C. Timmer, and E. Wolin. FIPA agent based network distributed control system. In *2003 Computing in High Energy and Nuclear Physics (CHEP03) La Jolla, Ca, USA*, Marzo 2003.

[GB99] Z. Guessoum and J.P. Briot. From active objects to autonomous agents. *IEEE Concurrency*, 7(3):68–76, 1999.

[GF92] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Stanford, CA, USA, 1992.

[GHM+01] E.J. Gonzalez, A. Hamilton, L. Moreno, J. Sigut, and R. Marichal. Evenet-2000: Designing and Training Arbitrary Neural Networks in Java. In Springer-Verlag, editor, *Bio-Inspired Applications of Conectionism*, pages 104–110, 2001.

[GHM+03a] E. J. Gonzalez, A. Hamilton, L. Moreno, R. M. Aguilar, and R. L. Marichal. Neural Networks Teaching using Evenet-2000. *Computer Applications in Engineering Education*, 11-1:1–5, Mayo 2003.

[GHM+03b] E.J. Gonzalez, A. Hamilton, L. Moreno, R.M. Aguilar, and R.L. Marichal. A Planning for Neural Networks Teaching in Control Using a Java-Based Toolkit. In *Proceedings of the 11th Mediterranean Conference on Control and Automation MED03*, Rodas, Grecia, 2003.

[GHM+03c] E.J. González, A.F. Hamilton, L. Moreno, R.L. Marichal, and S. Torres. MASPLAN: A Multi-Agent System for Automated Planning and Scheduling in a Universitary Research Group Scenario. In *Proceedings of The 1st Multidisciplinary International Conference on Scheduling : Theory and Applications (MISTA 2003)*, pages 42–55, Agosto 2003.

[GHMM03] E.J. Gonzalez, A. Hamilton, L. Moreno, and R.L. Marichal. Evenet2000 as Software Application for System Identification and Control. In *Proceedings of the 11th Mediterranean Conference on Control and Automation MED03*, Rodas, Grecia, 2003.

[GHSM01] Evelio J. González, Alberto F. Hamilton, Marta Sigut, and G.N. Marichal. Evenet 2000: Una Herramienta para el Diseño y Entrenamiento de Redes Neuronales de Arquitectura Arbitraria . In *Actas de las XXII Jornadas de Automática*, Barcelona, España, 2001.

[Gil97] Don Gilbert. Intelligent Agents: The Right Information at the Right Time, 1997.

[GK97] Michael R. Genesereth and Steven P. Ketchpel. Software Agents. *Communications of the ACM*, 37(7), 1997.

[GMF+03] J. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubezy, H. Eriksson, N. F. Noy, and S. W. Tu. The Evolution of Protege: An Environment for Knowledge-Based Systems Development. *International Journal of Human-Computer Interaction*, 58(1):89–123, 2003.

[GMH+00] E.J. González, L. Moreno, A. Hamilton, J.D. Piñeiro, R.L. Marichal, and G.N Marichal. Evenet2000: A New Java-Based Neural Network Toolkit. In *Proceedings of the Second ICSC Symposium on Engineering of Intelligent Systems (EIS 2000)*, Paisley, Escocia, Junio 2000.

[Gon00] E.J. Gonzalez. Diseño e Implementación en Java de una Librería y un Entorno de Cálculo para Entrenamiento de Redes Neuronales Basados en la Técnica de Grafo de Flujo de Señal. Memoria de Licenciatura en la Universidad de La Laguna, 2000.

[Gru93] T. R. Gruber. A translation approach to portable ontologies. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Ham95] Alberto Hamilton. *Estrategias de Control Optimo Basadas en Programación Dinámica y Redes Neuronales para sSistemas MIMO Continuos No Lineales*. PhD thesis, Universidad de La Laguna, 1995.

[Hew77] C. Hewitt. Viewing Control structures as Patterns of Passing Messages. *Artificial Intelligence*, 8(3):323–364, 1977.

[HGB] Heather Holmback, Mark Greaves, and Jeffrey M. Bradshaw. Agent A, Can You Pass the Salt? The Role of Pragmatics in Agent Communication .

[HH93] Don R. Hush and Bill Home. Progress in Supervised Neural Networks. *IEEE Signal Processing Magazine*, pages 8–39, Enero 1993.

[Hon99] V. Honavar. Intelligent Agents and Multi-Agent Systems. IEEE Conference on Evolutionary Computation (CEC), 1999.

[HP 03] HP Labs Semantic Web Research. *Jena 2 Manual*, Agosto 2003.

[HR95] B Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72(12):329–365, 1995.

[HS99] Michael N. Huhns and Larry M. Stephens. Multiagent Systems and Societies of Agents. In Gerhard Weiss, editor, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, pages 79–120. The MIT Press, Cambridge, MA, USA, 1999.

[HSMP96] M. Harteneck, R. Stewart, I. McWhirter, and I. Proudler. Algorithmic Engineering Applied to the QR-RLS Adaptive Algorithm, 1996.

[Igl97] C. Iglesias. Fundamentos de los agentes inteligentes. Technical Report UPM/DIT/GSI 16/97, Departamento de Ingeniería de Sistemas Telemáticos. Universidad Politécnica de Madrid., 1997.

[IGV96] Carlos A. Iglesias, José C. González, and Juan R. Velasco. *Intelligent Agents II*, chapter MIX: A General Purpose Multiagent Architecture. M. Wooldridge and J. P. Mueller and M. Tambe, 1996.

[Ins99] Mageland Institute. Introduction to CORBA. http://developer.java.sun.com/developer/onlineTraining/corba/, Diciembre 1999.

[Jac81] R.G. Jacquot. *Modern digital control systems*. Marcel Dekker, 1981.

[JPC00] Heecheol Jeon, Charles J. Petrie, and Mark R. Cutkosky. JATLite: A Java Agent Infrastructure with Message Routing. *IEEE Internet Computing*, 4(2):87–96, 2000.

[JSW98] N.R. Jennings, K. Sycara, and M. Wooldridge. A Roadmap of Agent Research and Development. *Autonomous Agents and Multi-Agent Systems Journal*, 1:7–38, 1 1998.

[KCT99] P. D. Karp, V. K. Chaudhri, and J. Thomere. XOL: An XML-Based Ontology Exchange Language. Technical report, Julio 1999.

[KH01] P. Kogut and W. Holmes. AeroDAML: Applying Information Extraction to Generate DAML Annotations from Web Pages. In *First International Conference on Knowledge Capture (K-CAP 2001).*, 2001.

[KKSP00] Elizabeth A. Kendall, P. V. Murali Krishna, C. B. Suresh, and Chira G. V. Pathak:. An application framework for intelligent and mobile agents. *ACM Computing Surveys*, 32, 2000.

[Knu03] Holger Knublauch. An AI Tool for the Real World: Knowledge Modeling with Protégé. *JavaWorld*, Junio 2003.

[KSCK94] H. A. Kautz, B. Selman, M. Coen, and S. Ketchpel. An Experiment in the Design of Software Agents. In O. Etzioni, editor, *Software Agents — Papers from the 1994 Spring Symposium (Technical Report SS-94-03)*, pages 43–48. AAAI Press, 1994.

[Kuo82] B.C. Kuo. *Automatic Control Systems*. Prentice-Hall, 1982.

[Lab99] Ramón Mª Gómez Labrador. Agentes móviles y CORBA, 1999.

[Lau00a] M. Laukkanen. Evaluation of FIPA-OS 1.03. Technical report, Cellular System Development, Sonera Mobile Operator, Sonera Ltd., Helsinki, Finlandia, Febrero 2000.

[Lau00b] M. Laukkanen. Evaluation of JADE 1.2. Technical report, Cellular System Development, Sonera Mobile Operator, Sonera Ltd., Helsinki. Finlandia, Febrero 2000.

[Lee99] Lyndon C. Lee. Designing Progressive MultiAgent Negotiations. In *International Conference on Conceptual Modeling / the Entity Relationship Approach*, 1999.

[LF94] Yannis Labrou and Timothy W. Finin. A Semantics Approach for KQML - A General Purpose Communication Language for Software Agents. In *CIKM*, pages 447–455, 1994.

[LF97] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, Baltimore, MD 21250, Febrero 1997.

[LFP99] Yannis Labrou, Tim Finin, and Yun Peng. The current landscape of Agent Communication Languages . *Intelligent Systems*, 14(2), Marzo/Abril 1999.

[LH00] Sean Luke and Jeff Heflin. SHOE 1.01 Proposed Specification. Technical report, SHOE Project, Abril 2000.

[Mae94] Pattie Maes. Modeling adaptive autonomous agents. *Artificial Life, I*, 1,2, 1994.

[Mae95] Pattie Maes. Artificial Life Meets Entertainment: Lifelike Autonomous Agents. In *Communications of the ACM 38(11)*, pages 108–114, 1995.

[Mae97] P. Maes. Agents that Reduce Work and Information Overload. In Jeffrey M. Bradshaw, editor, *Software Agents*. AAAI Press/MIT Press, 1997.

[Mai03] P. Maisano. JACK Intelligent AgentsTM JACK Manual. Technical report, Agent Oriented Software Pty. Ltd., Abril 2003.

[Mar98] R.L. Marichal. Adquisición y Análisis Mediante Redes Neuronales de Potenciales Evocados, 1998.

[Mar99] G.N. Marichal. *Diseño de Políticas de Identificación y Control de Robots Basadas en Redes Neuronales y Sistemas Neuro-Fuzzy*. PhD thesis, Universidad de La Laguna, 1999.

[Mar03] R.L. Marichal. *Estudio de la Dinámica de una Red Neuronal Recurrente Discrtea y su Aplicación a la Detección de Patrones en Señales Biomédicas e Identificación de Sistemas*. PhD thesis, Universidad de La Laguna, Julio 2003.

[MBB+98] D. Milojicic, M. Breugst, I. Busse, J. Campbell, S. Covaci, B. Friedman, K. Kosaka, D. Lange, K. Ono, M. Oshima, C. Tham, S. Virdhagriswaran, and J. White. MASIF: The OMG mobile agent system interoperability facility. In *Proceedings of Mobile Agents*, pages 50–67, Stuttgart, Germany, Septiembre 1998.

[MHVB00] D. Montana, J. Herrero, G. Vidaver, and G. Bidwell. A Multiagent Society for Military Transportation Scheduling. *Journal of Scheduling*, 3(4), 2000.

[Mil98] E. Miller. An introduction to the resource description framework. *D-Lib Magazine*, Mayo 1998.

[MM98] Robin McEntire and Donald McKay. KQML Lite languages and services. In *AAAI-98 Workshop on Software Tools for Developing Agents*, Julio 1998.

[MSJ98] N. Matos, C. Sierra, and N. R. Jennings. Determining successful negotiation strategies: an evolutionary approach. In Y. Demazeau, editor, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS-98)*, pages 182–189, Paris, France, 1998. IEEE Press.

[Nau96] Patrick Naughton. *Manual de Java*. 1996.

[ND02] T.G. Nguyen and T.T. Dang. Agent Platform Evaluation and comparison. Technical report, Institute of Informatics. Slovak Academy of Sciences, Junio 2002.

[NFM00] N. F. Noy, R. W. Fergerson, and M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. In *Proceedings of the 12th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000)*, Juan-les-Pins, France, 2000.

[NNL99] H. Nwana, D. Ndumu, and L. Lee. Zeus: A collaborative agents toolkit. In *Proceedings of the Third International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology*, pages 377–392, 1999.

[NNLC] Hyacinth S Nwana, Divine T. Ndumu, Lyndon C. Lee, and Jaron C. Collis. ZEUS: a toolkit and approach for building distributed multi-agent systems. In Oren Etzioni, J. P. Müller, and Jeffrey M. Bradshaw, editors, *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, Seattle, WA, USA. ACM Press.

[Nwa95] H. S. Nwana. Software Agents: An Overview. *Knowledge Engineering Review*, 11-2:205–244, 1995.

[NWC00] W. Nejdl, M. Wolpers, and C. Capella. *Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik, Modellierung*, chapter The RDF Schema Revisited. Foelbach Verlag, Koblenz., 2000.

[OC99] S. Osowski and A. Cichocki. Learning in dynamic neural networks using signal flow graphs. *Int. Journal of Circuit Theory and Applications*, 27:209–228, 1999.

[Oga96] K. Ogata. *Sistemas de Control en Tiempo Discreto*. Prentice Hall, 1996.

[Oga98] K. Ogata. *Ingeniería de Control Moderna*. Prentice Hall, 1998.

[OH95] S. Osowski and J. Herault. Signal flow graphs as an efficient tool for gradient and hessian determination. *Complex Systems*, pages 29 – 45, 1995.

[OIL] Description of OIL. Technical report.

[OMG98] MASIF-RTF Results. Technical report, Object Management Group, 1998.

[OMG00a] Agent Technology. Green paper. Technical Report agent/00-09-01, Object Management Group, Septiembre 2000.

[OMG00b] Property Service v1.0. Technical report, Object Management Group, Abril 2000.

[OMG00c] Query Service v1.0. Technical report, Object Management Group, Abril 2000.

[OMG01] Event Service v1.1. Technical report, Object Management Group, Marzo 2001.

[OMG02a] Life Cycle Service v1.2. Technical report, Object Management Group, Septiembre 2002.

[OMG02b] Naming Service v1.0. Technical report, Object Management Group, Septiembre 2002.

[OMG02c] Transaction Service v1.3. Technical report, Object Management Group, Septiembre 2002.

[Ont03] OntoEdit Tutorial. Technical report, Ontoprise, Abril 2003.

[OPB01] James Odell, H. Van Dyke Parunak, and Bernhard Bauer. Representing Agent Interaction Protocols in UML. In Paolo Ciancarini and Michael Wooldridge, editors, *Agent-Oriented Software Engineering.22nd International Conference on Software Engineering (ISCE)*, pages 121–140. Springer-Verlag, Berlin, 2001.

[Oso94] S. Osowski. Signal flow graphs and neural networks. *Biological Cybernetics*, 70:387 – 395, 1994.

[PBH00] S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS Agent Platform:Open Source for Open Standards, Abril 2000.

[Pet00] Charles J. Petrie. Agent-Based Software Engineering . In Jeffrey Bradshaw and Geoff Arnold, editors, *Proceedings of the 5th International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 2000)*, Manchester, UK, 2000. The Practical Application Company Ltd.

[Pic94] S.W. Piché. Steepest descent algorithms for Neural Network Controllers and Filters. *IEEE Trans. on Neural Networks*, 5(2), 1994.

[Pir02] Teppo Pirtitioja. Agent-Augmented Process Aumotation System. Master's thesis, Helsinki University of Technology, Helsinki, Finlandia, Noviembre 2002.

[Pis98] Alan Piszcz. A Brief Overview of Software Agent Technology. Technical report, 1998.

[PRS+03] Juan Albino Méndez Pérez, César Lorenzo Rodríguez, Leopoldo Acosta Sánchez, Santiago Torres Álvarez, Alberto Hamilton Castro, and Hector Reboso Morales. Controlweb: Una herramienta para el análisis y simulación de sistemas de control en Internet. In *Actas de las XXIV Jornadas de Automática*, León, Septiembre 2003.

[PTAC00] D. Pynadath, M. Tambe, Y. Arens, and H. Chalupsky. Electric Elves: Immersing an agent organization in a human organization. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop*, pages 150–154, 2000.

[PTCC99] David V. Pynadath, Milind Tambe, Nicolas Chauvat, and Lawrence Cavedon. Toward Team-Oriented Programming. In *Agent Theories, Architectures, and Languages*, pages 233–247, 1999.

[Rai82] H. Raiffa. *The art and science of negotiation*. Cambridge, Mass.: Belknap Press of Harvard University Press., 1982.

[RG95] A. S. Rao and M. P. Georgeff. BDI-agents: from theory to practice. In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.

[RL94] Fulbright R.D. and Stephens L.M. Classification of multiagent systems. Technical report, University of South Carolina, Columbia, SC 29208,, Junio 1994.

[RN95] S. Russel and P. Norvig. Artificial Intelligence, 1995.

[RS91] C.R. Reeves and N.C. Steele. Neural networks and genetic algorithms. In *Proc. 8th Int. Conf. on Systems Engineering*, pages 166–173, 1991.

[RS02] P.T. Roksvaag and A. Skjelvan. System Identifier. A Software Tool. Master's thesis, Department of Engineering Cybernetics. Norwegian University of Science and Technology, Agosto 2002.

[SAHK02] I. Seilonen, P. Appelquist, A. Halme, and K. Koskinen. Agent based Approach to Fault Tolerance in Process Automation Systems. Technical report, University of Helsinki, 2002.

[SAV+02] Ilkka Seilonen, Pekka Appelqvist, Mika Vainio, Aarme Halme, and Kari Koskinen. A Concept of an Agent-Augmented Process Automation System. In *15th IFAC World Congress on Automatic Control*, Barcelona, Spain, Julio 2002.

[SCS94] D.C. Smith, A. Cypher, and J. Spohrer. KidSim: programming agents without a programming language. In *Communications of the ACM*, volume 37, pages 55–67, 1994.

[SD01] Michael Sintek and Stefan Decker. TRIPLE—An RDF Query, Inference, and Transformation Language. In *DDLP'2001*, Tokyo, Japón, Octubre 2001.

[SD02] Michael Sintek and Stefan Decker. TRIPLE—A Query, Inference, and Transformation Language for the Semantic Web. In *International Semantic Web Conference (ISWC)*, Sardinia, Italia, 2002.

[Sen97a] Sandip Sen. Developing an automated distributed meeting scheduler. *IEEE Expert*, 12(4):41–45, Julio-Agosto 1997.

[Sen97b] Sandip Sen. Multiagent Systems: Milestones and New Horizons. *Trends in Cognitive Science*, 1(9):334–339, 1997.

[SFJ97] C. Sierra, P. Faratin, and N. Jennings. A Service-Oriented Negotiation Model between Autonomous Agents. In *Proceedings of the 8th European Workshop on Modeling Autonomous Agents in a Multi-Agent World (MAAMAW-97)*, pages 17–35, Ronneby, Sweden, 1997.

[SHA97] Sandip Sen, Thomas Haynes, and Neeraj Arora. Satisfying User Preferences While Negotiating Meetings. *International Journal of Human-Computer Studies*, 47:407–427, 1997.

[Sho93] Y. Shoham. Agent Oriented Programming. *Artificial Intelligence*, 60(1):51–93, Marzo 1993.

[SL97] Tuomas Sandholm and Victor R. Lesser. Coalitions Among Computationally Bounded Agents. *Artificial Intelligence*, 94(1-2):99–137, 1997.

[SMB99] C.M. Sperberg-McQueen and Lou Burnard. *Guidelines for Electronic Text Encoding and Interchange (TEI Guidelines)*. TEI P3 Text Encoding Initiative, revised reprint edition, 1999.

[SPKR96] W. Swartout, R. Patil, Knight K., and T. Russ. Towards Distributed Use of Large-Scale Ontologies. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[SPVG01] Katia Sycara, Massimo Paolucci, Martin Van Velsen, and Joseph Andrew Giampapa. The RETSINA MAS Infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Marzo 2001.

[SS89] T. Söderström and P. Stoica. *System Identification*. Prentice-Hall, London, 1989.

[STLP00] P. Scerri, M. Tambe, H. Lee, and D. Pynadath. Dont cancel my Barcelona trip: Adjusting the autonomy of agent proxies in human organizations. In *Proceedings of the AAAI Fall Symposium on Socially Intelligent Agents — the human in the loop*, Barcelona, 2000.

[Str80] P.D. Straffin. *Topics in the Theory of Voting*. Birkhauser, 1980.

[Stu00] J. Sturm. *Desarrollo de Soluciones XML*. McGrawHill Iberoamericana, 2000.

[Tar00] M.A. Martín Tardío. *Manual Imprescindible de Java 2, Edición 2000*. Ed. Anaya, 2000.

[Tho93] S.R. Thomas. *PLACA: An agent-oriented programming language*. PhD thesis, Computer Science Department, Stanford University, 1993.

[UG96] Mike Uschold and Michael Grüninger. Ontologies: Principles, Methods and Applications. *Knowledge Engineering Review*, 11(2):93–155, 1996.

[Vas98] Venu Vasudevan. Comparing Agent Communication Languages. Technical report, Object Services and Consulting, Inc, Julio 1998.

[VGMI46] J. Velasco, J.C. González, L. Magdalena, and C. Iglesias. Multiagent-based control systems: a hybrid approach to distributed process. *Control Engineering Practice*, 4:1996, Junio 839-846.

[vHPSH01] Frank van Harmelen, Peter Patel-Schneider, and Ian Horrocks. A Model Theoretic Semantics for DAML+OIL. Technical report, Marzo 2001.

[vP97] D. H. van Parunak. Go to the ant: Engineering principles from natural agent systems. *Annals of Operations Research*, 75:69–101, 1997.

[W3C99] Namespaces in XML. Technical Report REC-xml-names-19990114, W3C, Enero 1999.

[W3C03a] OWL Web Ontology Language Overview. Technical report, World Wide Web Consortium, Marzo 2003.

[W3C03b] OWL Web Ontology Language Reference. Technical report, World Wide Web Consortium, Marzo 2003.

[W3C03c] RDF Vocabulary Description Language 1.0: RDF Schema. Technical report, World Wide Web Consortium, Enero 2003.

[W3C03d] Resource Description Framework (RDF): Concepts and Abstract Syntax. Technical report, World Wide Web Consortium, Enero 2003.

[WB94] E. Wan and F. Beaufays. Relating Real-Time-Backpropagation and Backpropagation-Through-Time: An Application of Flow Graph Interreciprocity. *Neural Computation*, 6, 2 1994.

[WB97] E. Wan and F. Beaufays. Diagrammatic Methods for Deriving and Relating Temporal Neural Network Algorithms. In *E.R. CAIANIELLO SUMMER SCHOOL on Adaptive Processing of Sequences*, Vietri sul Mare, Salerno, Septiembre 1997.

[Wer90] P.J. Werbos. Backpropagation through time: What it is and how to do it. In *Proceedings of the IEEE*, pages 1550–1560, Octubre 1990.

[Win94] P.H. Winston. *Inteligencia Artificial*. Addison-Wesley, Wilmington, tercera edición edition, 1994.

[WJ94] Michael Wooldridge and Nicholas R. Jennings. Intelligent Agents: Theory and Practice. HTTP://www.doc.mmu.ac.uk/STAFF/mike/ker95/ker95-html.h (Hypertext version of Knowledge Engineering Review paper), 1994.

[WW] Ann Wollrath and Jim Waldo. An Overview of RMI Applications. http://java.sun.com/docs/books/tutorial/rmi/overview.html.

[ZFP+01] Youyong Zou, Tim Finin, Yun Peng, Anupam Joshi, and Scott Cost. Agent Communication In DAML World. In *First GSFC/JPL Workshop on Radical Agent Cocepts*, 2001.

Índice alfabético

árboles de identificación, 250, 251, 289

AeroDAML, 108, 119
agente, 98, 125–127, 130, 132, 140, 145, 149, 150, 153–158, 165, 166, 173, 177, 190, 210, 214, 219, 242, 305, 355, 356
API, 47, 109, 114
Applet, 117
ARP, 110
arquetipo, 69
ASCII, 50, 51, 64
atributo, 39, 41, 60, 65, 66, 69, 71, 85, 95
atributos, 83
axioma, 40, 53, 91, 93, 95, 112

base de conocimiento, 39, 40, 47, 49, 50, 99, 109, 113
bottom, 51

C, 47
cardinalidad, 90, 97, 113
Castor, 108, 109
clase, 47, 48, 74, 82, 88, 90, 91, 93–95, 112, 114
cliente, 98, 113
comentario SGML, 57
Common Lisp, 47, 50, 113
concepto, 39, 41
conceptualización, 39
container, 85
control, 26, 168, 266, 267, 280, 284, 289, 296, 305, 308, 362
CORBA, 29, 32–34, 113, 155, 236, 355
Core Oil, 89
cosificación, 86, 89, 119
CSS, 67
CyC, 189, 222–224, 226, 228, 229, 243, 252, 256, 259
Cyc, 44, 45, 47, 108, 109
CycL, 44
Cycorp, 44, 109

DAML API, 111
DAML Validator, 109, 121
DAML Viewer, 117
DAML+OIL, 37, 55, 73, 94–96, 98, 99, 108, 111, 113, 117, 119, 121, 123
DAML+OIL Ontology Checker, 108, 121
DAML-ONT, 94
DAML-S, 99
daml:cardinality, 97
daml:collection, 98
daml:complementOf, 97
daml:hasClassQ, 97
daml:imports, 95
daml:intersectionOf, 98
daml:inverseOf, 97
daml:maxCardinality, 97
daml:minCardinality, 97
daml:ObjectProperty, 96
daml:oneOf, 98
daml:Restriction, 96
daml:samePropertyAs, 97
daml:TransitiveProperty, 97
daml:UnambiguousProperty, 97
daml:UniqueProperty, 97
daml:versionInfo, 95
DAMLViewer, 108

DARPA, 47, 55, 94
disjunción, 95, 98
dominio, 41, 113
DQL, 98
DTD, 54, 56, 59, 61, 62, 64, 65, 69, 87
Dublin Core, 42, 44, 86, 121

elemento padre, 57
elementos hijo, 57
entidad, 61, 120
espacio de nombres, 44, 66, 69, 70, 94, 123
etiqueta, 56, 62, 63, 65, 66, 72–74, 83, 96, 97
Excel, 117

faceta, 47–49
FaCT, 113, 221
factor de olvido, 278, 302
filler, 91
función de activación, 361, 396, 413
función de transferencia, 268, 269

gradiente, 280, 314, 365, 383, 388, 394, 400
GUI, 114

Heavy OIL, 89
herencia, 96, 112
hoja de estilo, 68
HTML, 37, 54, 62–64, 67, 72, 73, 113

IBM, 34
identificación, 250, 301, 304–306, 314, 317
individuo, 47, 48, 93, 112, 113
inferencia, 74, 88, 109, 113, 116, 117, 121
Instance OIL, 89, 93
instancia, 40, 48, 75, 90, 93, 113
Inteligencia Artificial, 38
interlingua, 48

JADE, 173, 175
Java, 47, 109, 110, 114–117
Jena, 108–110, 121, 122

KIF, 49–52
KIF estructurado, 50
KIF lineal, 50
KRS, 47, 48

lógica descriptiva, 88, 94, 113
lazo cerrado, 283
lenguajes de marcas, 42, 53–55, 63, 94, 99
licencia GPL, 112
Linux, 112
Lisp, 50

marca, 53–55, 67
marcas descriptivas, 55
marcas procedurales, 55
marco, 48, 112
marshalling, 109
MASCONTROL, 295, 297–301, 304, 305
MASIF, 292
metaclase, 48
metaconocimiento, 52
metadato, 42, 79
Microsoft Visio, 119

NLP, 119

OIL, 55, 88–90, 93–95, 112, 113
OilEd, 108, 111–113, 228
OKBC, 41, 47, 48, 54, 115
OntoEdit, 108, 110, 112, 116
OntoEdit Free, 116, 117
OntoEdit Professional, 116, 117
ontología, 37, 38, 40, 41, 44, 73, 74, 88–91, 95, 108, 109, 112–114, 116, 119
ontología de alto nivel, 40
ontología de aplicación, 40
ontología de dominio, 40
OOP, 23, 388, 403, 412
OpenCyc, 108, 109
OWL, 37, 55, 73, 109, 113, 123
OWL Converter, 123
OWLConverter, 109
OXML, 117

parser, 56, 61, 64, 72, 85, 110
perdedor de Condorcet, 192, 242
plug-in, 114, 116, 117

predicado, 46, 80, 84
Principio de distinción ontológica, 41
Protégé, 108, 110, 112, 114
Protégé-2000, 114, 115

rango, 113
RDF, 37, 38, 43, 55, 79–83, 87, 88, 94, 98, 108–111, 114, 115, 121
RDF API, 110, 111, 122
rdf:about, 83
rdf:aboutEach, 85
rdf:aboutEachPrefix, 85
rdf:Alt, 85
rdf:Bag, 85
rdf:Description, 84, 86
rdf:ID, 84, 86
rdf:li, 85
rdf:object, 86
rdf:predicate, 86
rdf:Seq, 85
rdf:subject, 86
rdf:type, 86
RDFS, 87–89, 94, 95, 113, 121
rdfs:comment, 87
rdfs:ConstrainProperty, 87
rdfs:ConstrainResource, 87
rdfs:domain, 87, 96
rdfs:isDefinedBy, 87
rdfs:label, 87
rdfs:range, 87, 96
rdfs:seeAlso, 87
rdfs:subPropertyOf, 87, 96
RDQL, 111
recuento de Borda, 194, 241
recurso, 79, 81, 86, 94, 119
redes neuronales, 283, 383, 387, 396, 402
relación, 39, 41, 46, 74
restricción, 71, 96
RMI, 30

Sección de Oro, 407
seriación, 113
servidor, 98, 113
SGML, 54–57, 61–64, 69, 72
SHF, 113
SHIQ, 113
SHOE, 54, 72, 73, 119
slot, 47–49, 90–94, 112
SQL, 111
SquishQL, 111
Standard OIL, 89–93, 113
statement, 86
subclase, 49, 90, 95
superclase, 25, 49, 93, 112

TRIPLE, 108, 121
triple, 80, 81, 83, 87, 110, 111

UML, 110, 114
Unicode, 64, 66
unmarshalling, 109
Upper Cyc Ontology, 44
URI, 38, 66, 79, 80, 83–86, 98, 110, 117, 119, 120, 122, 123
URI cualificado, 80
URL, 74

validación, 122
variables de estado, 268, 271, 272, 317–319, 321–324
VisioDAML, 108, 119

W3C, 37, 38, 54, 55, 63, 79, 94
Web Semántica, 37, 38, 42, 55, 81, 108, 114, 119, 121
Windows, 112
World Wide Web, 37, 38, 42, 54

XML, 37, 38, 54, 55, 63–74, 76, 79, 81–88, 94, 98–100, 109, 110, 114, 159, 166, 169, 176, 219, 234, 245, 301, 357, 413
XML bien formado, 64, 68
XML Infoset, 110
XML Schema, 54, 69–71, 87, 88, 98
XML validado, 69, 71
XOL, 55
XSL, 67, 68
XSLT, 67, 69