Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Migración de aplicaciones monolíticas a basadas en microservicios

Migrating monolitics applications to microservices-based architectures

Pedro Antonio Lima Adrián

La Laguna, 4 de julio de 2019

D. **Vicente José Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor.

D. **Antonio Estévez García**, con N.I.F. 43.615.500-V responsable del área de I+D en Open Canarias, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Análisis de aplicaciones de arquitecturas monolíticas para su migración a arquitecturas basadas en microservicios*"

ha sido realizada bajo su dirección por D. **Pedro Antonio Lima Adrián**,con N.I.F.78.649.772-F.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de julio de 2019

Agradecimientos

Muchas gracias a todas aquellas personas que me han acompañado estos cuatro años en la Universidad, tanto compañeros como profesores y familia.

Asimismo, debo agradecer a todos aquellos que me ayudaron a lo largo de mis estudios en primaria, secundaria y bachillerato, permitiéndome entrar a la Universidad y sentar las bases del conocimiento que hoy en día poseo.

Sin todos ellos no habría llegado tan lejos como lo he hecho.

Licencia

* Si quiere permitir que se compartan las adaptaciones de tu obra y quieres permitir usos comerciales de tu obra (licencia de Cultura Libre) indica:

Resumen

El objetivo de este Trabajo de Fin de Grado será el recabar información sobre diferentes algoritmos, procedimientos y técnicas heurísticas que ayuden al análisis del programa para su migración de arquitectura.

Para ello se dará información de ciertos aspectos relevantes en dicho análisis como son las arquitecturas monolítica y basada en servicios o el uso de *KDM*, *Knowledge Discovery Metamodel*. Con esta información se podrá tener una visión más amplia del proceso y mejorar nuestra comprensión del mismo. Algo fundamental a la hora de casos prácticos donde, en general, serán aplicaciones complejas o un conjunto de ellas, incluso sistemas completos.

En cuanto a las técnicas y algoritmos se intentará que puedan usarse como ideas para mejoras de la solución actual propiciando diferentes enfoques en distintos campos relacionados con el análisis de programas o sistemas como son técnicas de minería de datos o de grafos junto a algoritmos similares al *Program Sicing* actualmente utilizado, siendo algunas de ellas teóricas mientras otras serán implementaciones.

Palabras clave: Arquitetura Monolítica, Arquitectura basada en Microservicios, Knowledge Discovery Metamodel, Program Slicing, Minería de procesos, Chopping, Roll Up, Program Dependency Graph, Program Graph Representation, Cobol.

Abstract

The main goal of the Final Degree Project is to get information about algorithms, procedures or techniques to analize applications before the migration to another architecture.

To achieve this objective, this document includes information about other aspects which plays an important role in the software analysis like the monolitics and microservices-based architectures or KDM, Knowledge Discovery Datamodel. The reader can get a general view and a better comprehension of the process. This is something fundamental in real projects which have complex application or a group of them, even complete systems too.

Concerning to the techniques and algorithms related with this preocesses, they are included in this document to be used like ideas to improve the actual solution. These techniques are based in differents areas related with Program's or systems' analysis. Some of them are techniques from Data Mining and Graphs while others are from algorithms similar to Program Slicing, which is the actual solution. The document includes both types of ideas, where some solutions has a theoritical approach and the others follows a practical one.

Keywords: *Monolitic Architecure, microservices-based architectures, Knowledge Discovery Metamodel, Program Slicing, Process Mining, Chopping, Roll Up, Program Dependency Graph, Program Graph Representation, Cobol.*

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1.1. Tema

El tema que se tratará en este Trabajo de Fin de Grado será “El análisis requerido para la migración de un aplicativo desde una arquitectura monolítica a una basada en microservicios”.

Por supuesto esta migración de sistemas necesita tener un plan para llevar a cabo este cambio, el cual se lleva a cabo mediante un análisis de los servicios y plataformas utilizadas. Este análisis requiere investigar sobre muchos temas distintos, en este caso, para este Trabajo de Fin de Grado se decidió enfocarse en las técnicas heurísticas o algoritmos que permitan reconocer los patrones tanto precursores, referentes a la arquitectura de origen, como patrones de solución, referentes a la arquitectura de destino, y así poder realizar de manera exitosa las transformaciones necesarias para la migración de arquitectura.

1.2. Situación

Hoy en día, el auge de los dispositivos inteligentes y la tendencia de cambio en periodos cortos en aplicativos, siendo en la mayoría de casos de varias versiones al año, hacen que un sistema monolítico pierda su validez.

Esto no quiere decir que sean malos, sino que para la demanda actual del mercado y los consumidores es preferible el uso de sistemas más flexibles, facilitando los cambios, y más ágiles, permitiendo que dichas actualizaciones del aplicativo sean lo más rápidas posibles.

Además este cambio de arquitectura monolítica a otras como la basada en microservicios no solo depende de los consumidores y el mercado, sino también de los equipos encargados de dar mantenimiento y seguir desarrollando el sistema.

Por ejemplo, puede darse el caso de tener que duplicar algún servicio, ya sea por medidas de seguridad o por dar una mejor respuesta intentando evitar que se sature el acceso a cierto servicio por la gran demanda que posee, y, en ambos casos, si la arquitectura es monolítica se deberá duplicar todo el sistema, en cambio, si fuese una arquitectura basada en microservicios bastaría con duplicar solo el servicio necesario. Esto solo ha sido un ejemplo de la diferencia en escalabilidad entre ambos sistemas, pero es suficiente para entender el ahorro de costes que supondría escalar con la arquitectura basada en microservicios y el gasto que representa duplicar todo el sistema en una arquitectura monolítica.

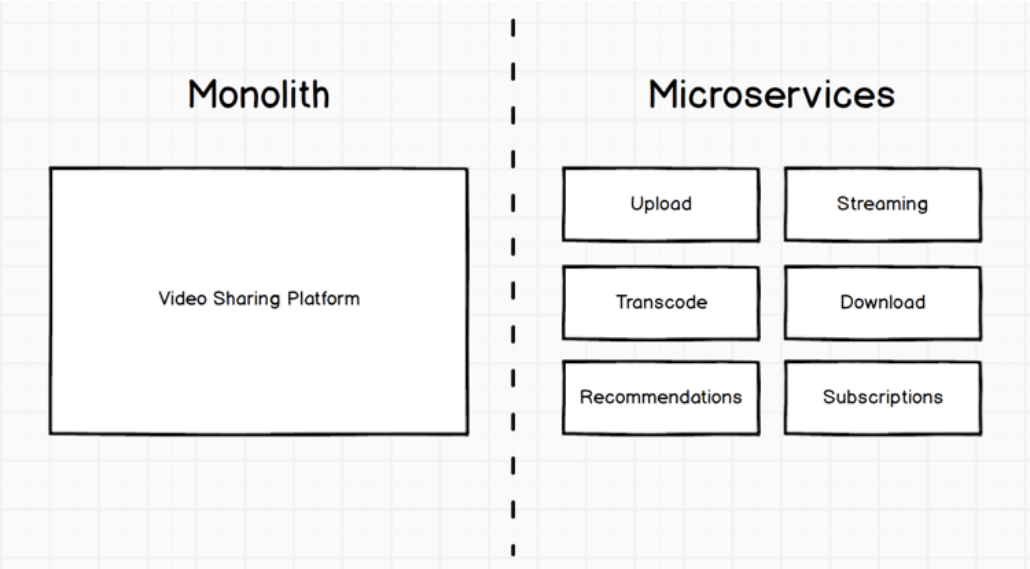

Figura 1.1: Ejemplo de componentes entre una aplicación monolítica y una basada en microservicios

1.3. Relevancia

Mientras un sistema monolítico se va volviendo cada vez más complejo, empiezan a aparecer problemas de desarrollo y mantenimiento del mismo. En la actualidad, la solución más utilizada para los problemas que surgen en los sistemas monolíticos con el paso del tiempo es el cambio a sistema basado en microservicios.

Esta solución la han efectuado y efectúan multitud de empresas en el mundo. No importa si es una empresa grande, mediana o pequeña sino dependerá de lo complejo que sea su sistema. Normalmente, esta modernización de sistema requiere una inversión que suele realizarse cuando los números avalan dicho cambio en los beneficios, pero si cada vez más empresas y organizaciones lo llevan a cabo significa que el resultado es satisfactorio.

Por supuesto, no todas las empresas necesitarán cambiar su arquitectura, pero con la extensión de internet, aplicaciones y servicios a nivel mundial y su acceso continuo por parte de un masivo número de usuarios, las arquitecturas que permitan adaptarse mejor a este nuevo entorno serán muy solicitadas. Un ejemplo claro, que ya hemos mencionado anteriormente, es la arquitectura basada en microservicios. En el capítulo 2 se hablará más a fondo sobre dicha arquitectura y los beneficios que conlleva su utilización frente a la monolítica.

Este Trabajo de Fin de Grado se realizó con la ayuda de la empresa “Open Canarias”. Dicha empresa es un claro ejemplo de cómo el análisis del sistema y servicios de una organización para una posible modernización está muy demandado. Algunos clientes que han contado con ella para este proceso han sido empresas del IBEX35 como “Santander” o “Iberdrola”. Organizaciones con gran capital que para su modernización requieren de una gran inversión económica y necesitan un análisis para aumentar el porcentaje de éxito del proceso, además del ahorro de recursos y personal que resulta a posteriori.

1.4. Enfoque

Este tema es muy amplio abarcando múltiples campos. La principal base del Trabajo de Fin de Grado será investigar sobre aquellas técnicas heurísticas o algoritmos capaces de ayudar a la hora de analizar los programas necesarios.

En la mayoría de los casos nos encontraremos con sistemas con decenas, cientos o miles de programas, cada uno con muchas líneas de código. Algo de esta magnitud resulta complicado de analizar de forma manual. Por esta razón se utilizan técnicas y algoritmos con el de fin de analizar de la mejor manera posible dicha información.

En general, se intentará construir grafos de dependencias entre programas para visualizar las relaciones entre ellos y vislumbrar las implicaciones para poder hacer una división efectiva de servicios. Sin embargo, incluso este resultado puede resultar ambiguo, debido a lo complicado que puede ser el grafo resultante. Así que también son interesantes las técnicas o algoritmos que puedan ayudar a simplificar dicho resultado y, por tanto, ayudar a la comprensión del sistema.

Actualmente, un algoritmo muy utilizado es el “Slicing” [15]. Este algoritmo se usa para ”trocear”la información que se quiere analizar, pudiendo separar “trozos” de ella y estudiar aquellos que se necesite. Más adelante se hablará más a fondo de esto junto a otros posibles algoritmos en el capítulo 3.

Capítulo 2

Arquitecturas relevantes

En este capítulo se hablará sobre la arquitectura monolítica y la arquitectura basada en microservicios [1]. Existen muchas más arquitecturas, pero estas son las utilizadas en el caso de ejemplo y por tanto serán las que se detallarán.

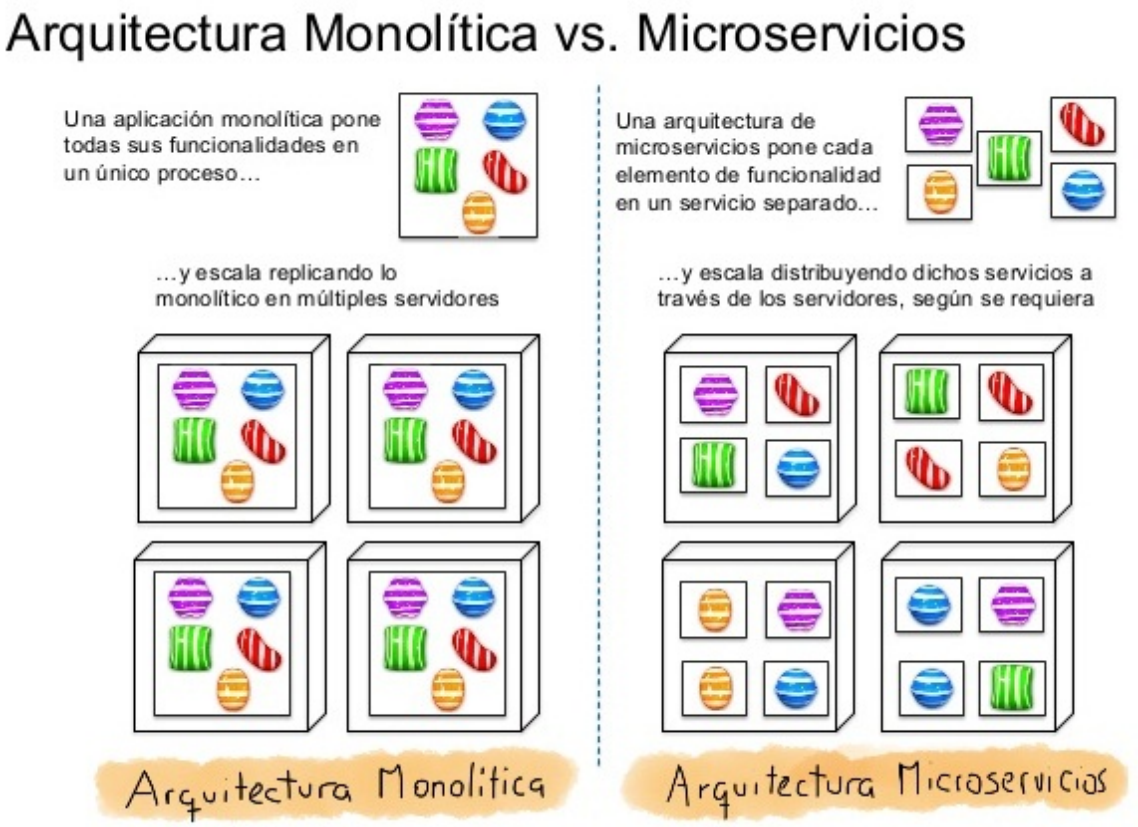

Figura 2.1: Diferencia entre arquitectura monolítica y basada en microservicios

2.1. Arquitectura monolítica

Las arquitecturas monolíticas[11] fueron muy utilizadas hace décadas dónde no era una época tan demandante de continuos cambios rápidos disponiendo de un sistema flexible y de escalabilidad alta. Ciertos conceptos que son fundamentales a día de hoy aún no se conocían o estaban en fase de aceptación, siendo ahora estos conceptos quienes marcan la arquitectura a utilizar en la mayoría de los casos.

La arquitectura monolítica corresponde con un sistema de tres capas. En primer lugar, tenemos una capa de presentación; seguida de una capa de aplicación; y, en último lugar, una capa de datos.

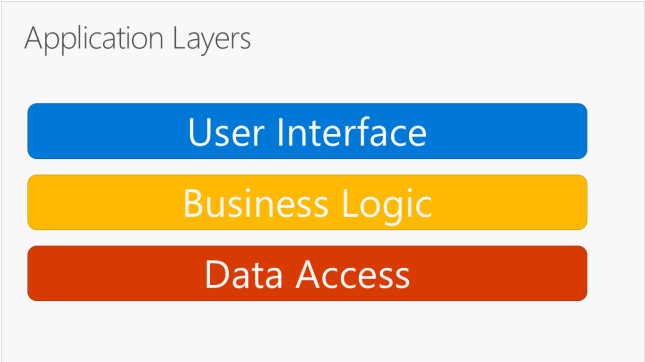

Figura 2.2: Esquema de arquitectura de tres capas

Básicamente, por un lado, la aplicación entrega datos al navegador web que opera en el nivel de presentación y proporciona los medios para que los usuarios soliciten información de la aplicación, la vean y, generalmente, la manipulen o cambien.

Por otra parte, la aplicación lee y escribe información desde y hacia el nivel de datos, donde una base de datos u otro dispositivo de almacenamiento o aplicación la organiza y la mantiene.

Por último, la aplicación consiste en la lógica para interactuar con los otros dos niveles y transformar los datos según lo solicite el usuario.

Este tipo de arquitectura simplemente está desactualizada. Esto se debe al cambio de prioridades en las demandas del mercado y aspectos técnicos. No depende del número de capas que posea, sino de su idea de aplicación única, que choca con la moda actual de cambios constantes y rápidos sobre un sistema flexible para cumplir con la demanda del mercado tanto a nivel de actualizaciones como de nuevas funcionalidades a distintos tipos de dispositivos y entornos.

El principal problema recae en que este tipo de servicios son un todo, el menor cambio que se deba realizar conlleva la reconstrucción y prueba de todo el servicio. Aún pese a modularse, seguirá siendo dependiente en gran medida con lo que el problema seguirá existiendo.

Y por supuesto no hay que olvidar el gran fallo de esta arquitectura recae en la escalabilidad. Un concepto que no estaba asentado como lo está actualmente y que es de suma importancia debido al importe económico.

La principal idea, hoy en día, es variar el servicio según la demanda, pero si se quisiera hacer eso con un sistema monolítico, que debe escalarse como un todo y no por componentes, representaría un costo elevado.

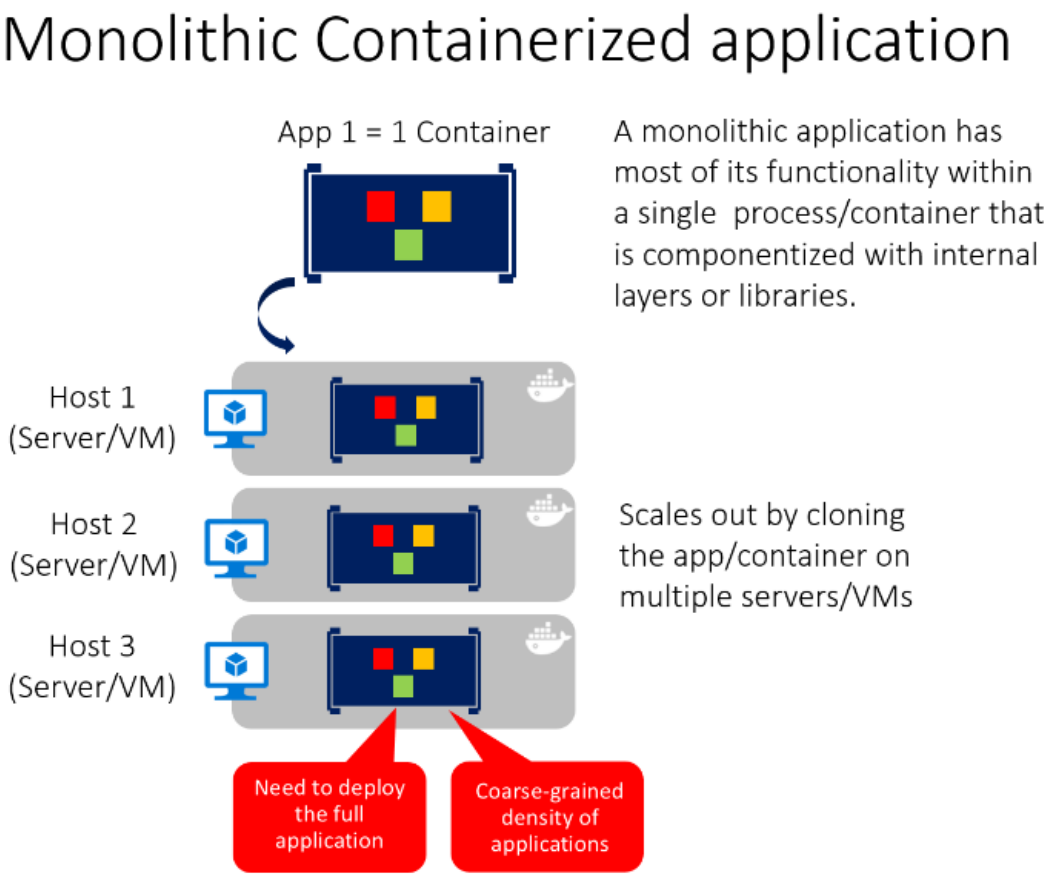

Figura 2.3: Esquema de escalabilidad en una arquitectura monolítica

2.2. Arquitectura basada en microservicios

Para adaptarse a las demandas actuales se modificó el esquema de arquitectura de tres capas a cuatro[12]. Esto permite dar al sistema agilidad, flexibilidad y escalabilidad, gracias a diseñarse pensando en la era de los móviles y dispositivos inteligentes.

La primera capa **Cliente** será la novedad, permitiendo adaptar la experiencia a las condiciones de dispositivos y funcionalidades del usuario aprovechando las otras tres capas de las que se apoya.

Además, tendremos en segundo lugar, la capa de **Entrega** que se encargará de entregar la optimización adecuada para la mejor experiencia del usuario usando los datos de la capa Cliente. En este nivel también se encontrarán el uso de caché con algoritmos de almacenamientos y las herramientas de monitorización y de resolución de problemas en tiempo real.

En tercer lugar, está la capa **Agregación o Incorporación** que, básicamente, es una capa API para integrar y permitir la comunicación entre los servicios internos y externos. También se encarga de compilar y componer el contenido que se entregará al cliente.

Por último, la capa **Servicios** proporciona los datos y funcionalidades que las otras capas necesitan. Enfocado en la idea de microservicios[14] es abierto y libre, haciendo hincapié en la integración y composición de servicios existentes de la empresa y bibliotecas de código abierto.

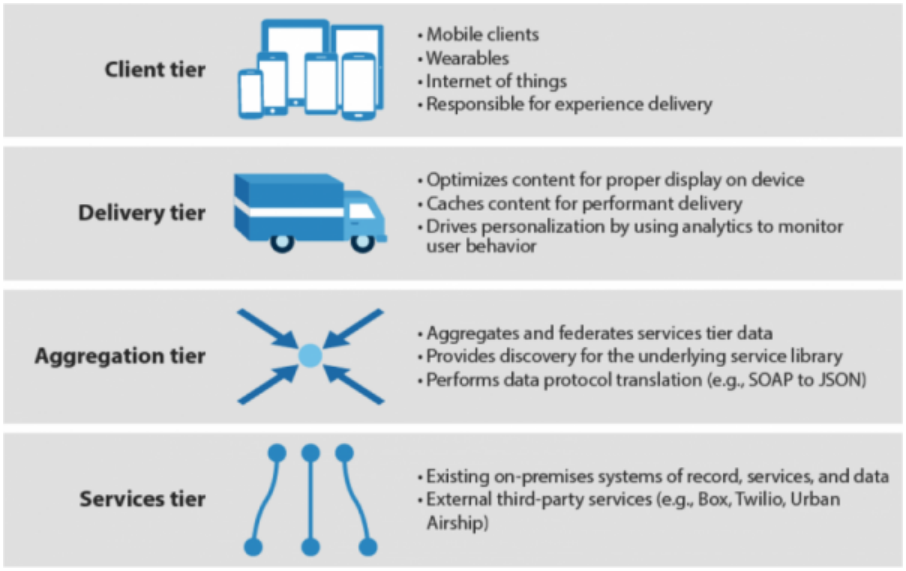

Figura 2.4: Esquema de arquitectura de cuatro capas

2.3. Ventajas de la arquitectura basada en microservicios

Enfocando el tema en las ventajas que supone disponer de una arquitectura basada en microservicios a una monolítica, debemos resaltar cuatro :

1. Pequeños componentes independientes :
    - Desarrollar cada componente sin interferir con otros.
    - Menor curva de aprendizaje del componente al ser de menor tamaño, por tanto, menos tiempo, esfuerzo y errores.
    - Componentes más simples, lo que lleva a poder desarrollar más funcionalidades del mismo.
2. Despliegue :
    - Despliegues más simples y pequeños que ayudan a la velocidad y reducción de errores.
    - Despliegue independiente, es decir, pudiendo desplegar la cantidad que se desee desde uno solo a todos a la vez.
3. Escalabilidad :
    - Es posible solo escalar el componente necesario, no todos, con lo que es flexible y se reducen costos.
4. Reutilización :
    - Componentes más simples y precisos pueden usarse en otros sistemas, servicios o productos con poco o nada de adaptación.

2.4. Aspectos a tener en cuenta en arquitecturas basadas en microservicios

Hasta hace poco tiempo era complicado llevar a cabo una sistema basado en microservicios, sin embargo, el auge en tecnologías de contenedores y de orquestación, con *Docker* y *Kubernetes* como "lanzas de ataque" respectivamente, ha facilitado el avance en la implementación técnica.

Por supuesto no siempre será necesario un sistema basado en microservicios y será preferible otro que se adapte mejor a la circunstancia, por contra, habrán casos donde un sistema basado en microservicios será recomendable :

- Una aplicación monolítica necesita ayuda para escalar y mejorar su rendimiento. Pueden darse dos casos :

    1. Extender componentes modulares desde la aplicación monolítica, siempre que estén bien diseñados y permitan un correcto acoplamiento con una aplicación basada en microservicios.
    2. Crear la aplicación desde cero usando la información de la existente.

- Se desea una aplicación basada en microservicios desde el inicio. En este caso, hay tres opciones :

    1. Tiene gran importancia la modularidad y descentralización.
    2. Se prevé un gran volumen de transacciones o de tráfico.
    3. Se dispone de equipos adecuados para el diseño, desarrollo e implementación de aplicaciones rápidamente, sobretodo al inicio.

A continuación se hablará de partes clave de un sistema basado en microservicios :

1. **Delimitar correctamente el alcance de la funcionalidad** : Ya sea particionando los servicios por funcionalidad lógica o definiéndolo según las líneas de código previsibles para un equipo de desarrollo en un tiempo corto de semanas.

2. **API** : Es necesaria para la comunicación entre los servicios, dónde suele utilizarse API de servicios web REST, aunque existen otros mecanismos.

3. **Tráfico** : Es necesario gestionar el tráfico del sistema, desde realizar seguimiento del estado y prepararse ante caídas o respuestas que tomen mucho tiempo hasta la respuesta ante sobrecarga de la API o la creación o finalización de trabajos por parte de los servicios para adaptarse a las variaciones del tráfico a tiempo real.

4. **Datos** : Habrá aplicaciones que necesiten disponer de servicio aún cuando ocurran fallos de conexión o de red que impidan la descarga de datos, donde este acceso a los datos podrá realizarse mediante un almacenamiento compartido redundante o mediante el uso de caché de almacenamiento compartido.

5. **Monitorización** : Se debe permitir el cambio continuo de recursos, junto a capturar datos de monitorización y mostrar la información cambiante relativa a las aplicaciones cada determinada frecuencia.

2.5. Caso práctico tomado como ejemplo

El caso más común a la hora de modernización del sistema es el siguiente. Uno conformado por un **Mainframe** con sistema operativo **ZOS**, una plataforma con **Cobol** como lenguaje de programación, **CICS** como gestor de transacciones, junto a **DB2** como base de datos; estas transacciones pueden ser interactivas o por lotes utilizando **DDL** y **JCL**, respectivamente.

El objetivo será pasar de dicho sistema a uno basado en microservicios que implique todo lo dicho en apartados anteriores.

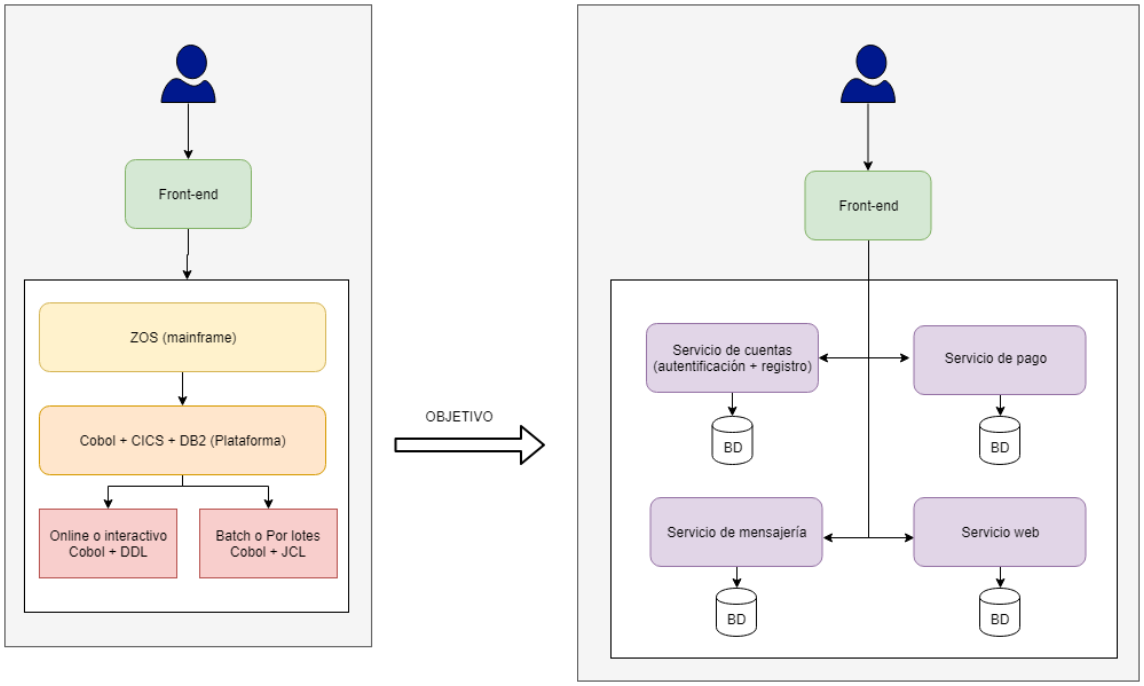

Figura 2.5: Esquema de caso práctico tomado como ejemplo

Normalmente, estos sistemas dispondrán de miles o decenas de miles de líneas de código a analizar. Para hacerlo se utilizan grafos y técnicas heurísticas o algoritmos. En nuestro caso, en **Open Canarias** se utiliza el **Slicing** como técnica para analizar los programas del tipo aplicaciones software en **OC Rosetta**[15] para proyectos **KDM** o **Knowledge Discovery Metamodel**[8].

Estos temas son los que veremos a continuación en los siguientes capítulos.

Capítulo 3

KDM, Knowledge Discovery Metamodel

3.1. Descripción técnica

Knowledge Discovery Metamodel o *KDM*[8] es un metamodelo **MOF** o **Meta-Object Facility** el cual se utiliza para proporcionar un sistema de tipos con el que mediante un conjunto de esquemas se pueden definir la estructura, significado y comportamiento de los objetos y un conjunto de interfaces que permiten la creación, almacenamiento y manipulación de dichos esquemas.

KDM define un formato de intercambio **XMI** o **XML Metadata Interchange** para compartir diagramas de modelado entre herramientas y define una *API* sobre la que se pueden construir nuevas herramientas de modernización.

Básicamente, *KDM* es un modelo **entidad-relación** permitiendo constituir relaciones entre los distintos sistemas que interactúan. Se define una ontología para describir dichos sistemas de software existentes que permitirá determinar las variables recogidas para algún conjunto y establecerá relaciones entre ellas.

Knowledge Discovery Metamodel define múltiples jerarquías de entidades vía contenedores o grupos y se pueden componer agrupando entidades en contenedores tipados, que representarán colecciones de dichas entidades.

El fin de utilizar *KDM* es del tipo analítico, no de ejecución. En él se estandarizan aproximaciones existentes de descubrimiento de conocimiento y está alineado con el estándar **ISO/IEC 11404 Language Independent Datatypes and SBVR**.

Una parte de *KDM* se denomina **Program Elements Layer** que representa información similar a los **AST** o **Abstact Syntax Tree** para un determinado lenguaje de programación.

Sin embargo, son un poco distintas entre sí, ya que la **Program Elements Layer** de *KDM* tiene un comportamiento independientemente del lenguaje de programación utilizado, ya que un procedimiento en C o en Fortran utilizarán el mismo metaelemento para su representación.

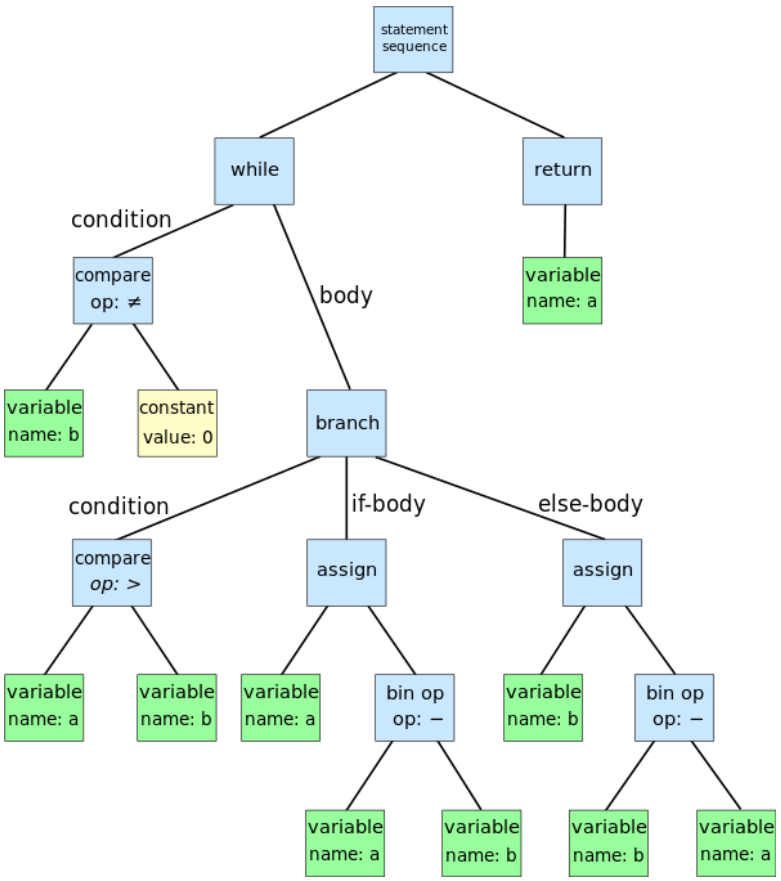

Figura 3.1: Ejemplo de *AST* (*Abstract Syntax Tree*)

3.2. Breve recorrido histórico

A continuación se dará una breve descripción de su historia cuyo inicio data de Junio de 2003 con la creación del grupo de trabajo para *KDM*.

En Noviembre de 2003, se presenta la solicitud de propuesta de *KDM*, llamada **RFP** o **Request For Proposal**, adoptada por la **OMG** o **Object Management Group** en Mayo de 2006.

El **OMG** es un consorcio internacional de estándares tecnológicos sin ánimo de lucro fundado en 1989 y cuyos estándares están dirigidos por organizaciones, usuarios, instituciones académicas y agencias gubernamentales.

La primera especificación recomendada **KDM 1.0** se dispuso en *OMG* a principios de 2007. Y, hoy en día, varias compañías dan soporte de *KDM* en sus productos :

- **KDM Analythics**
- **Benchmark Consulting**[2] : utilizando *KDM* para escenarios de modernización de Cobol. Esta empresa posee un producto llamado **IRIS** que permite modernizar sistemas *mainframe* de *CA Datacom* con *DB* a *DB2* y *Oracle*, teniendo como lenguaje de programación Cobol y generando de él las reglas de negocio que posea el sistema en formato *JRules*, basado en Java.
- **El Proyecto MoDisco Eclipse** : usando *KDM* para la ingeniería inversa basada en modelos.
- **Adaptive** : dando soporte *KDM* en la hoja de ruta,

3.3. Estructura de KDM

KDM posee una estructura dividida en cuatro capas con un total de doce paquetes. A continuación se explicará brevemente dicha estructura:

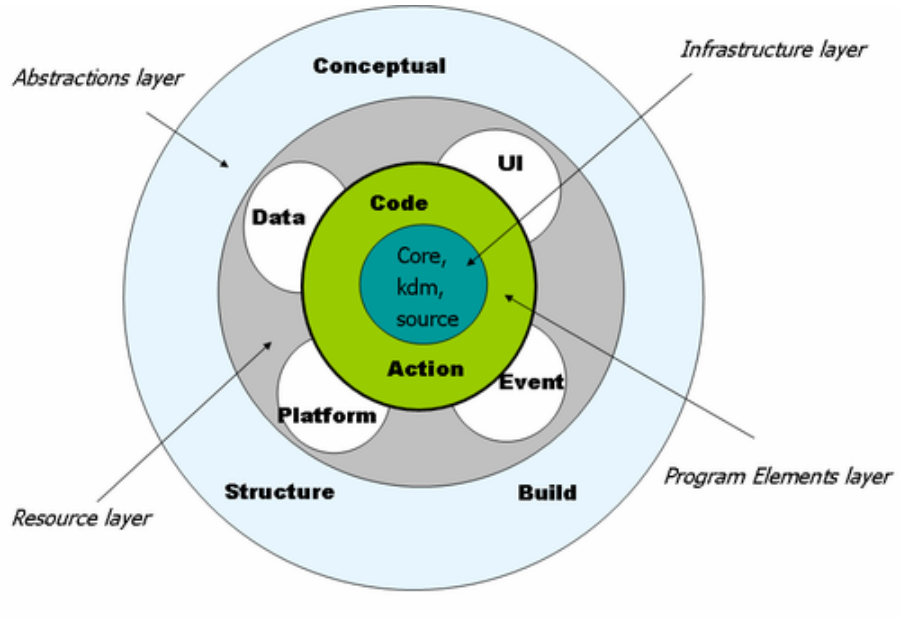

Figura 3.2: Esquema de estructura *KDM*

1. **Capa de Infraestructura o Infraestructure layer** : Es el núcleo común en el que se apoyan los demás paquetes y posee los paquetes **Core**, **Kdm** y **Source**. Además, cabe resaltar que existe una alineación entre el núcleo de *KDM* y el **Resource Description Framework** o **RDF**.

2. **Capa de Elementos del programa o Program Elements layer** : Consiste en dos paquetes **Code** y **Action**.

   El primero, **Code**, representa elementos utilizados en los lenguajes de programación, tales como clases, tipos de datos, variables, etc. Su propósito es similar al **Common Application Metamodel** o **CAM** de **Enterprise Application Integration** o **EAI** de *OMG*.

   El segundo paquete de esta capa, **Action**, captura los elementos de comportamiento de bajo nivel permitiendo obtener una representación intermedia de alta fidelidad de cada componente del sistema al combinarse con el paquete **Code**.

3. **Capa de Recursos o Resource layer** : Permite representar entidades y relaciones determinadas por la plataforma de tiempo de ejecución o *runtime platform* proporcionando elementos de modelado integrados en el núcleo de *KDM* y posee los siguientes cuatro paquetes **Platform**, **UI**, **Event** y **Data**.

   El paquete **Platform** representa el entorno operativo, relacionado con el sistema operativo, el *middleware*, etc.

   Por otro lado, el paquete **UI** representa conocimiento relacionado con las interfaces de usuario del sistema y el paquete **Event** representa el conocimiento relacionado con los eventos y las transiciones de estado del sistema.

Por último, el paquete **Data** alineado con otra especificación de *OMG*, **Common Warehouse Metamodel** o **CWM**, representa los elementos relacionados con los datos persistentes y otros tipos de almacenamiento de datos.

4. **Capa de Abstracción o Abstraction layer** : representa las abstracciones de dominio y aplicación, poseyendo tres paquetes **Conceptual**, **Structure** y **Build**.

   El paquete **Conceptual** está alineado con otra especificación de *OMG*, denominada **Semantics of Business Vocabulary and Rules** o **SBVR**, representando el conocimiento del dominio empresarial y las reglas comerciales.

   El segundo paquete es **Structure** que describe los elementos del metamodelo para representar la organización lógica del sistema en subsistemas, capas y componentes.

   Y, el último paquete de esta capa, es **Build** para representar el punto de vista de ingeniería respecto al sistema software.

3.4. Herramientas de análisis de software

Existen múltiples herramientas de análisis de software para el mantenimiento y la modernización de programas o sistemas. Estos pueden proporcionar un conocimiento sobre estos códigos, o incluso ofrecer transformaciones semiautomáticas, con el fin de comprender y evolucionar.

Las herramientas típicas de análisis de software incluyen uno o más analizadores para lenguajes de programación, una representación intermedia, una capa de integración, y luego, un componente de análisis.

Las fuentes a analizar por estas herramientas son, principalmente, solo el código fuente. Otras fuentes de conocimiento del ciclo de vida del desarrollo del software incluyen archivos de configuración de implementación, código de máquina, componentes de terceros, repositorios de administración de configuración de software de información, entre algunos otros.

El modelo de información interno es a menudo la columna vertebral de la herramienta analítica que separa la complejidad de analizar el código fuente de la complejidad de realizar la extracción de conocimiento.

Es frecuente que el modelo interno de una herramienta de análisis de software sea una forma de **Árbol Abstracto de Sintaxis** o **AST**. Los *AST* son esenciales para la construcción del compilador, sin embargo, crean una barrera entre las herramientas enfocadas en diferentes idiomas, ya que el componente de análisis está limitado a la ontología de un lenguaje de programación particular. Hay varias consecuencias negativas de eso como la poca interoperabilidad entre las herramientas de análisis o que cada proveedor de herramientas tiene que sobresalir en varias tecnologías diversas para poder obtener y comprender el conocimiento requerido en cada sistema. Como resultado, cada herramienta tiene sus propias fortalezas y debilidades.

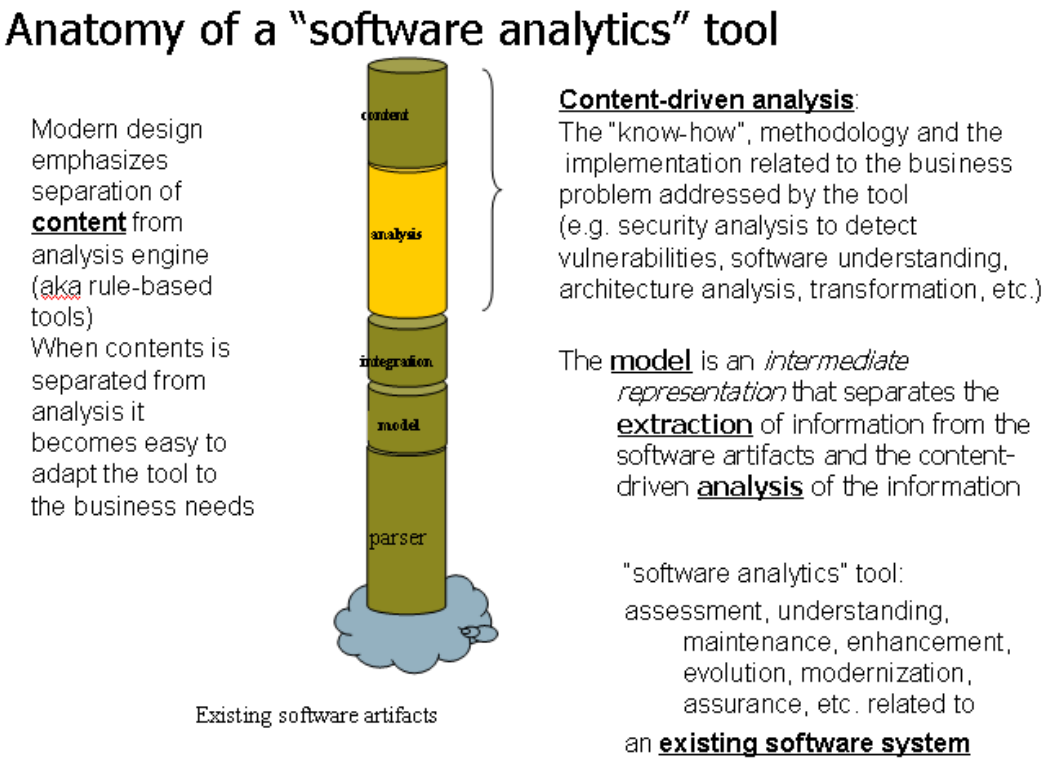

Figura 3.3: Esquema de una herramienta de análisis de software

3.5. Razones para usar KDM

Las herramientas de análisis de software se ven favorecidas al usar *KDM* para generar una profunda integración semántica entre ellas. Estas herramientas son de gran importancia porque tratan al software como datos, de los que se puede sacar conocimiento.

El campo principal de actuación es en la modernización de software siendo una industria afianzada que admite las fases de mantenimiento y evolución del ciclo de vida del desarrollo de software o más conocido por sus siglas en inglés **SDLC**.

Figura 3.4: *SDLC* o Ciclo de vida del desarrollo de software

Una de las mayores preocupaciones durante dichas fases de desarrollo de software

es la **integridad**, es decir, la cohesión de los activos de software con el fin de modificar de manera eficiente la base de código existente.

Sin embargo, las herramientas actuales de mantenimiento ofrecen poca interoperabilidad, por lo que para obtener una cohesión de la aplicación los clientes deben realizar integraciones de **punto a punto**[7] entre estas herramientas.

El **modelo punto a punto** significa que cada aplicación tuvo que ser personalizada para comunicarse con las otras aplicaciones y piezas de TI. Es un trabajo tedioso, propenso a fallos y difícil de mantener en el tiempo debido a actualizaciones en la infraestructura y aplicaciones.

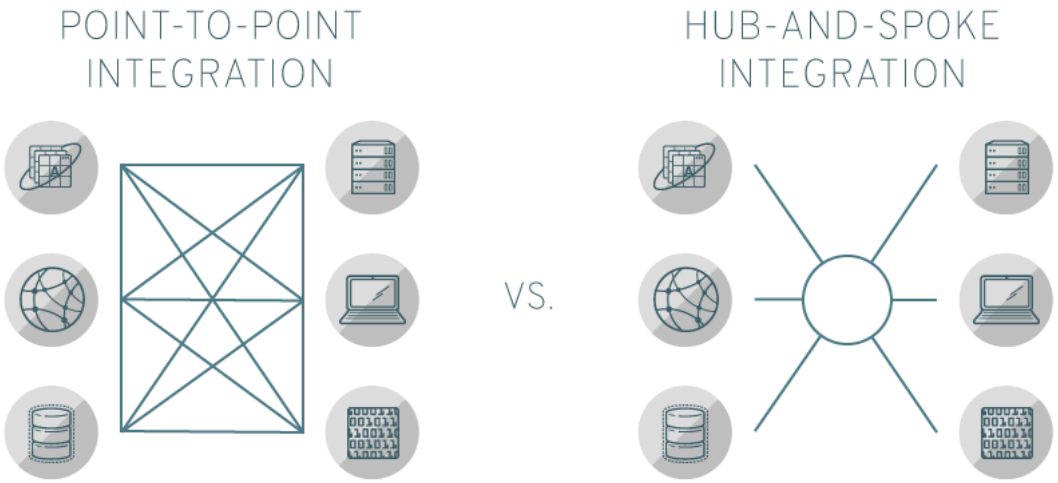

Figura 3.5: Esquema de modelo punto a punto

Todas estas herramientas de las que estamos hablando tienen fortalezas y debilidades, por esta razón a continuación se resaltarán causas que pueden llevar a necesitar utilizar *KDM* para generar una integración profunda entre las herramientas :

- **Diferentes lenguajes de implementación**.
- **Diferentes plataformas runtime**.
- **Análisis de aplicaciones** que utilizan marcos de aplicaciones, componentes *COTS*, es decir, componentes de terceros y uso de servicios.
- **Necesidad de comprender la arquitectura actual** de un sistema de software y administrar los cambios desde la perspectiva arquitectónica.
- **Necesidad de combinar análisis estático con análisis de arquitectura y métrica**.
- **Necesidad de integrar el análisis estático y dinámico de sistemas**.
- **Necesidad de correlacionar datos** de gestión de desarrollo de software de varias fuentes.

En cuanto a los datos, *KDM* define el formato de la base de datos de desarrollo de software que se puede utilizar para la administración y seguimiento de activos de software, dónde es relevante que una serie de tareas de mantenimiento posean una visión coherente de todos los datos del *SDLC*, ya que esta información puede ser analizada y ayudar a la gestión del desarrollo del software.

Por último, con respecto al tema de la integración entre herramientas, existen dos niveles de integración :

1. **De servicios** : las herramientas deberán cumplir con el mismo estándar de servicio e infraestructura.
2. **De Ontología** : las herramientas deberán compartir la misma ontología relacionada con la aplicación, generando el concepto de **archivo fuente**. Ante esto, *KDM* proporciona una extensa ontología relacionada con las aplicaciones de software existentes.

Para poder enlazar las herramientas a *KDM* se debe crear un puente o conexión a partir de los modelos internos de las herramientas y deberán cumplir ciertos criterios definidos en las especificaciones de *KDM* para llevar a cabo el enlace exitosamente, lo que podemos llamar **Ecosistema KDM**.

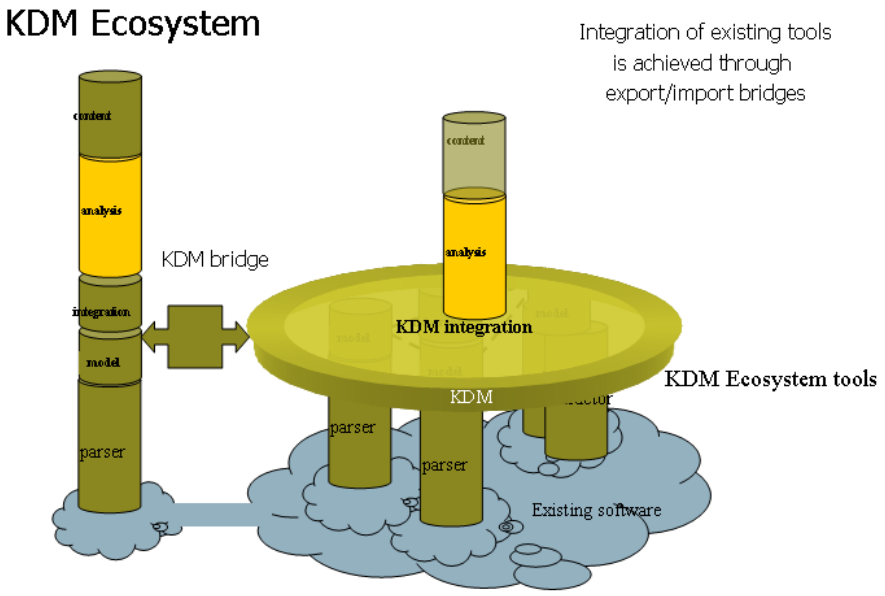

Figura 3.6: Ejemplo de ecosistema *KDM* con unión a herramienta

Capítulo 4

Técnicas y algoritmos

4.1. ¿Qué se busca?

Durante un proceso de modernización de un sistema se utilizan herramientas para analizar el software de dicho sistema o aplicativo, tal y como se nombró en el capítulo anterior3.

Estas herramientas tienen una finalidad que radica en ver el software como datos y, a través del análisis, poder extraer conocimiento de ellos.

Esta finalidad no es solo propia de estas herramientas utilizadas en la modernización de sistemas, sino que se encuentran también en industrias como el **Big Data** y el **Business Intelligence**[10]. Es más, se podría decir que dichas industrias se basan únicamente en esto, en a partir de una gran cantidad de datos poder definir patrones, conocimiento, relaciones y dependencias entre ellos.

Al llegar a este punto del Trabajo de Fin de Grado y empezar a buscar técnicas o algoritmos que nos permitan conseguir un análisis de las relaciones, dependencias y conocimiento del sistema dado, llegué a la conclusión de que si intentamos realizar algo que ya se realiza en otros campos de investigación como el **Big Data** y el **Business Intelligence**, simplemente, deberíamos aprender de ellos.

Y es que la modernización de sistemas tiene la facilidad para aprovecharse de todos los avances en estos campos y así mejorar los resultados del análisis, aumentar la capacidad de manejar las relaciones o dependencias resultantes de manera que cada vez sean más intuitivas y sencillas, entre otros beneficios.

Las herramientas de **Business Intelligence** pueden dividirse en tres grandes categorías :

1. **Gestión de datos** : Encargadas de la depuración y estandarización de datos de procedencia diversa hasta su extracción, transformación y traslado a un sistema determinado.

2. **Aplicación para descubrir nuevos datos** : Relacionado con el **Data Mining** o **Minería de datos**, permite recopilar y evaluar nueva información para aplicar sobre ella técnicas de análisis. Suelen usarse técnicas de análisis predictivo con el fin de realizar proyecciones de futuro.

3. **Reporting** : Encargadas de la visualización gráfica e intuitiva de la información recabada, o de sus integración en cuadros de mando para un seguimiento y evaluación.

Para nuestro tema de modernización resultan muy interesantes **las aplicaciones para descubrir nuevos datos**, sobretodo las técnicas analíticas relacionadas con el **Data Mining** utilizadas. Algunos ejemplos de estas técnicas son :

- **Algoritmos de descubrimiento** como alfa, alfa+, alfa++ o beta[4].
- **Minería heurística**[13]
- **Fuzzy Mining**[3]
- **Dicing o Slicing**[6]
- **Roll up**[5]

Lo curioso es que muchas de estas técnicas son a la vez técnicas para trabajar con grafos, esta es la principal razón de investigar esta vertiente para conseguir soluciones al tema del Trabajo de Fin de Grado.

Sin duda, dentro de este campo se podrán encontrar técnicas que desbanquen al algoritmo de **Slicing** del que se hablará a continuación o se empleen en conjunto con él, en cualquier caso, esta fue la rama de investigación seguida.

4.2. Solución actualmente utilizada (*Program Slicing*)

La solución actual que se posee se basa en el algoritmo de **Slicing**, más concretamente el **Program Slicing**[15], soportando el paradigma de programación estructurada aceptando variables globales en Cobol.

Program Slicing es una técnica o algoritmo que consiste en dividir un todo, en este caso, un programa en partes que se denominan **Slices** o, en castellano, **Lonchas**. Para realizar esta división, al Slicing se le proporciona un **criterio** basado en un **punto del programa** o <s >que corresponde con una instrucción o sentencia de código y, en una representación gráfica, a un vértice. El criterio se complementará con el conjunto de variables pertenecientes al *slice* o <V >, así el criterio se conforma con una forma o estructura **s, V**.

El programa utilizado será el código correspondiente al **CodeModel** del modelo *KDM* dado y el *slice* resultante representa un subconjunto del programa no necesariamente ejecutable.

Cabe resaltar que entre los puntos del programa y el criterio de partida se produce una relación de dependencia que varía según el tipo de *Slicing* utilizado.

En todo caso, se puede conocer si un algoritmo de *Slicing* es correcto o no y, para ello, se utilizará el siguiente esquema :

Sin embargo, aún teniendo un algoritmo de *Slicing* correcto, no hay uno que sea óptimo probado científicamente. Esto se debe a que computar *slices* de tamaño óptimo es indecible, por lo que habrán algoritmos que sean óptimos para ciertas circunstancias, pero en otras condiciones no lo sean.

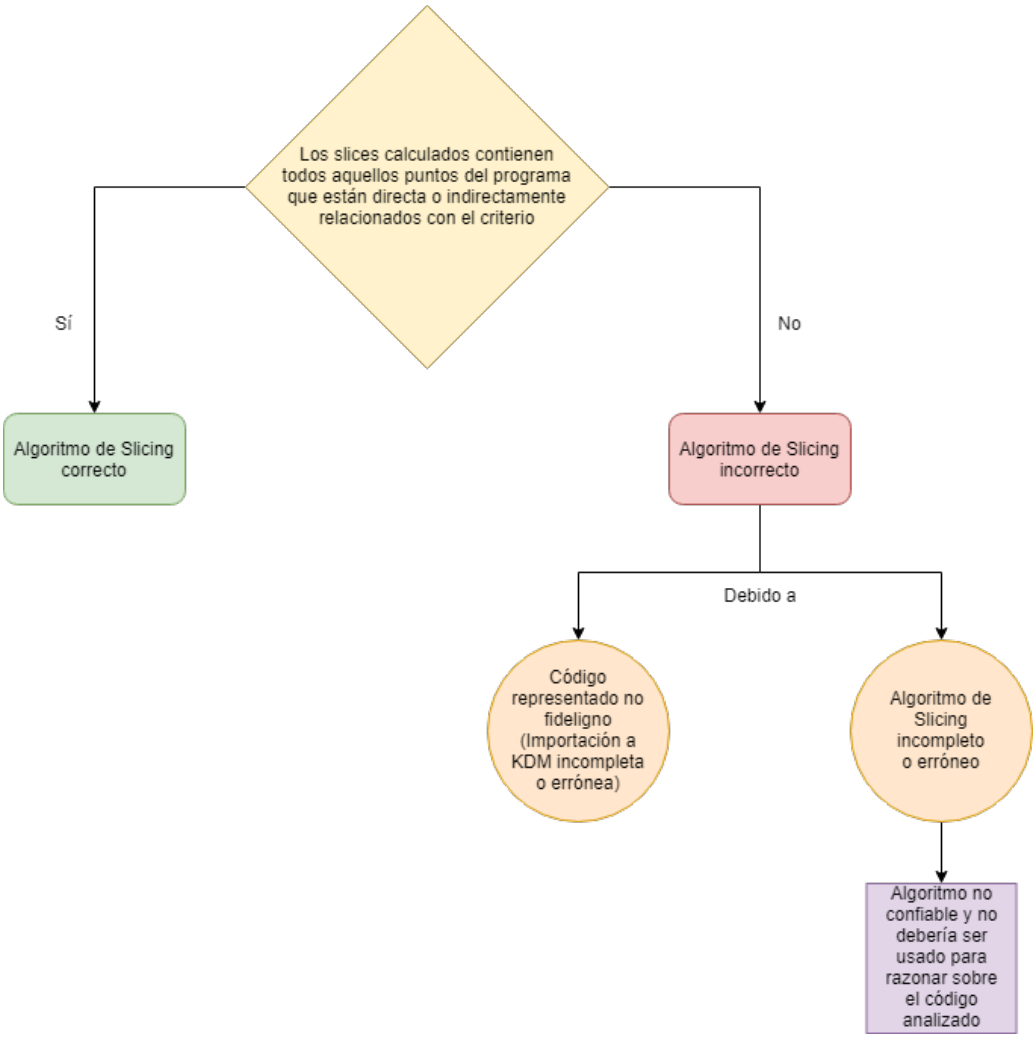

Figura 4.1: Esquema sobre verificación del algoritmo de *Slicing*

Hay diferentes tipos de *Slicing* e incluso en esta implementación conviven distintos de ellos. La primera diferenciación proviene a través del comportamiento de las variables.

Por un lado, disponemos del **Slicing Estático** en el que solo se manejará el código fuente para la obtención de *slices*.

Por otro lado, los casos en que ciertas variables toman valores concretos y de interés dentro de un rango específico son usadas por el **Slicing Dinámico**.

Las variables que se analizarán podrán ser de uno de los siguientes cuatro efectos :

1. **Sobre el entorno de ejecución** : System.out.println(x);
2. **Sobre otras variables** : a = x + 2;
3. **Sobre la misma variable** : x = x + b - 3;
4. **Sin efecto** : x + 2 ; *Estas podrían ser descartadas*.

Cabe destacar en el campo de variables que el algoritmo no soporta el fenómeno de **Aliasing** de zonas de memoria que pueden estar representadas o referenciadas por más de un nombre simbólico del programa, factor que suele darse en el uso de punteros, vectores o matrices.

Debido a poder obtener información adicional que restrinja el contexto de ejecución del programa, el **Slicing Dinámico** permite generar *slices* de menor tamaño y, por tanto, ser más óptimo. Sin embargo, esto solo será posible en ciertos casos estudiados, en el resto se deberá seguir usando el *Slicing* Estático con *slices* de mayores dimensiones.

Por otra parte, dependiendo del sentido de propagación de dependencias del criterio se puede diferenciar entre **Backward Slicing** y **Forward Slicing** :

- **Backward Slicing** : Sigue un sentido contrario a la ejecución del programa y halla aquellas partes del programa desde las cuáles se alcanza al criterio. También es posible pensarlo como **¿Quién me utiliza?** o **¿De dónde vengo?**.
- **Forward Slicing** : Sigue el sentido normal de la ejecución del programa y responde a si cambias **x** instrucción, línea de código o punto del programa que partes del código pueden ser afectadas. También responde a **¿Quién depende de mí?** o **¿A dónde voy?**.

Por último, también podrá diferenciarse el *Slicing* en :

- **Reaward** : Componente que propaga el cálculo de *Slicing* a los procedimientos que invocan las partes del código analizadas.
- **Onward** : Componente que propaga el cálculo de *Slicing* a los procedimientos llamados desde **call sites** que se han confirmado que pertenecen al *slice*.

Una vez hemos definidos los tipos de *Slicing* encontrados en la solución actual continuaremos con la visualización de soluciones. Esto se realizará mediante representaciones con grafos dirigidos y se utilizarán de tres clases :

1. **CFG** o **Control-Flow Graph** : Describe la totalidad de caminos posibles de código que pueden ser recorridos durante la ejecución.

2. **PDG** o **Program Dependency Graph** : Grafo conformado por dos subgrafos, siendo uno referente a datos **DDG** o **Data Dependency Graph** y otro referente a sentencias de control **CDG** o **Control Dependency Graph**.

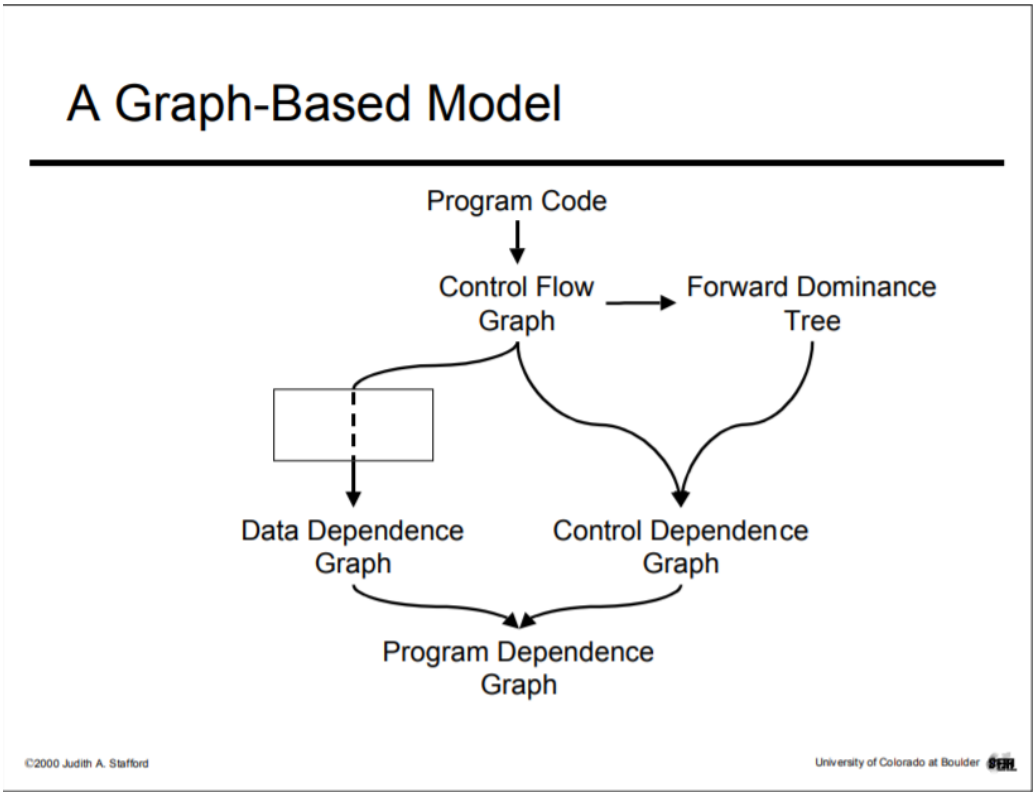

Figura 4.2: Esquema de relación entre el código del programa y el grafo *PDG*

3. **SDG** o **System Dependency Graph** : Grafo que relaciona todos los **PDGs** mediante nodos y aristas adicionales para facilitar el cálculo interprocedural.

En el algoritmo *Slicing* utilizado se han implementado tres variantes del recorrido por el grafo *PDG* :

1. Un recorrido estándar que es intraprocedural y no admite sensibilidad paramétrica, pero que si soporta la sensibilidad contextual.

2. Un recorrido interprocedural soportando tanto la sensibilidad paramétrica como la sensibilidad contextual, pero sin cálculo de *summaries*.

3. Un recorrido interprocedural con *summaries* soportando ambas sensibilidades.

Los modelos [2] y [3] referentes a recorridos interprocedurales pueden tener que lidiar con dificultades debido a dependencias cíclicas entre llamadas.

Cabe resaltar la importancia de los **Summaries**, ya que incluyen información de porciones de *slices* o información contextual que facilita el cálculo de *Slicing* para partes

del programa que son susceptibles de repeticiones de cálculo de *slices*, lo que ayuda en el tiempo de cómputo.

Sobre las representaciones en forma de grafos hay que resaltar el uso del lenguaje **PGR** o **Program Graph Representation** que formaliza la representación de grafos y el **SLC** que modela los *slices* resultantes.

4.3. Algoritmos de *Data Mining*

En este apartado abarcaremos el área de **Data Mining** o **Minería de datos** y nos adentraremos en el **Process Mining** o la **Minería de Procesos**[3]. Esta es una técnica de análisis de datos que integra la minería de datos de eventos con las técnicas de modelado de procesos. Existen tres tipos de minería de componentes :

- **Descubrimiento** : Se basa en el análisis de un **registro de eventos** y la producción de un modelo tomando como base las muestras de ejecución de los registros de eventos, sin utilizar información previa con el fin de descubrir los procesos reales.
- **Conformidad** : Se compone de técnicas para la comparación de eventos del registro de eventos con actividades del modelo de proceso con el fin de comprobar que el modelo es equivalente a la información almacenada en el registro de eventos para así detectar desviaciones, cuellos de botella e incongruencias.
- **Mejora** : Tiene la finalidad de mejorar el proceso existente utilizando la información del proceso real almacenada en el registro de eventos.

Figura 4.3: Esquema del modelado de procesos

Existen distintos algoritmos para el descubrimiento de modelos que podrían ser interesantes, algunos de ellos son :

- **Alpha Mining** : Se basa en descubrir una **red Petri** de flujo de trabajo a partir de registros aunque tiene limitaciones si encuentra bucles cortos de tamaño uno o dos y también tiene problemas al relacionar dependencias no locales, al igual que otros algoritmos de minería de procesos.
- **Alpha+ Mining** : Mejora del algoritmo *Alpha Mining* en la resolución de bucles cortos con éxito y mejora en el estudio de relaciones.
- **Inductive Miner** : Basado en la técnica **Divide y vencerás**, este algoritmo hará divisiones recursivas creando particiones con los eventos hasta hallar el modelo.
- **Heuristic Miner** : Mejora del *Alpha Miner* detectando los bucles cortos junto a saltos de actividades en los procesos y, además, tiene en cuenta las frecuencias en el análisis.
- **Genetic Mining** : Algoritmo de búsqueda heurística basado en encontrar soluciones óptimas mediante la teoría de la selección natural y la evolución biológica.
- **Fuzzy Mining** : Se intenta abstraer detalles de los modelos y permite simplificar la visualización mediante una segmentación o **clúster** que agrupa eventos con alta correlación y baja significancia, la cual determina la importancia relativa de eventos y relaciones del proceso.

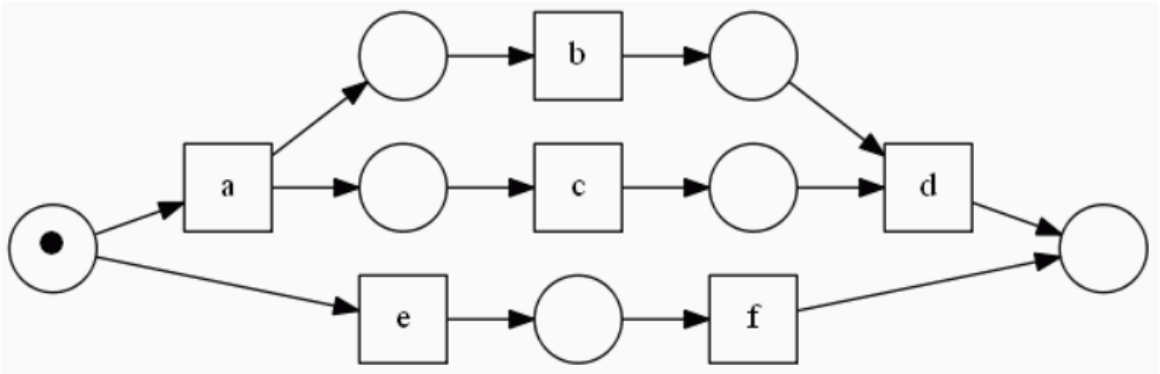

Figura 4.4: Ejemplo de red *Petri* del algoritmo *Alpha Miner*

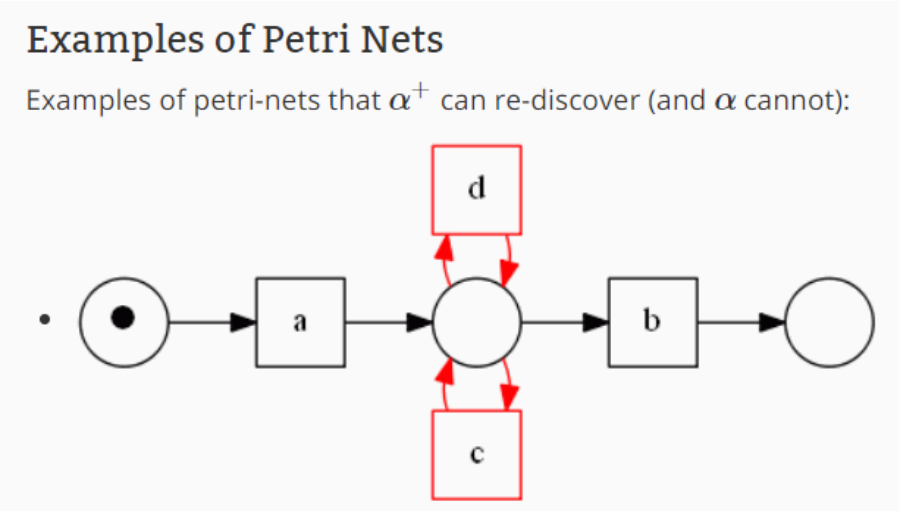

Figura 4.5: Ejemplo de red *Petri* del algoritmo *Alpha+ Miner*

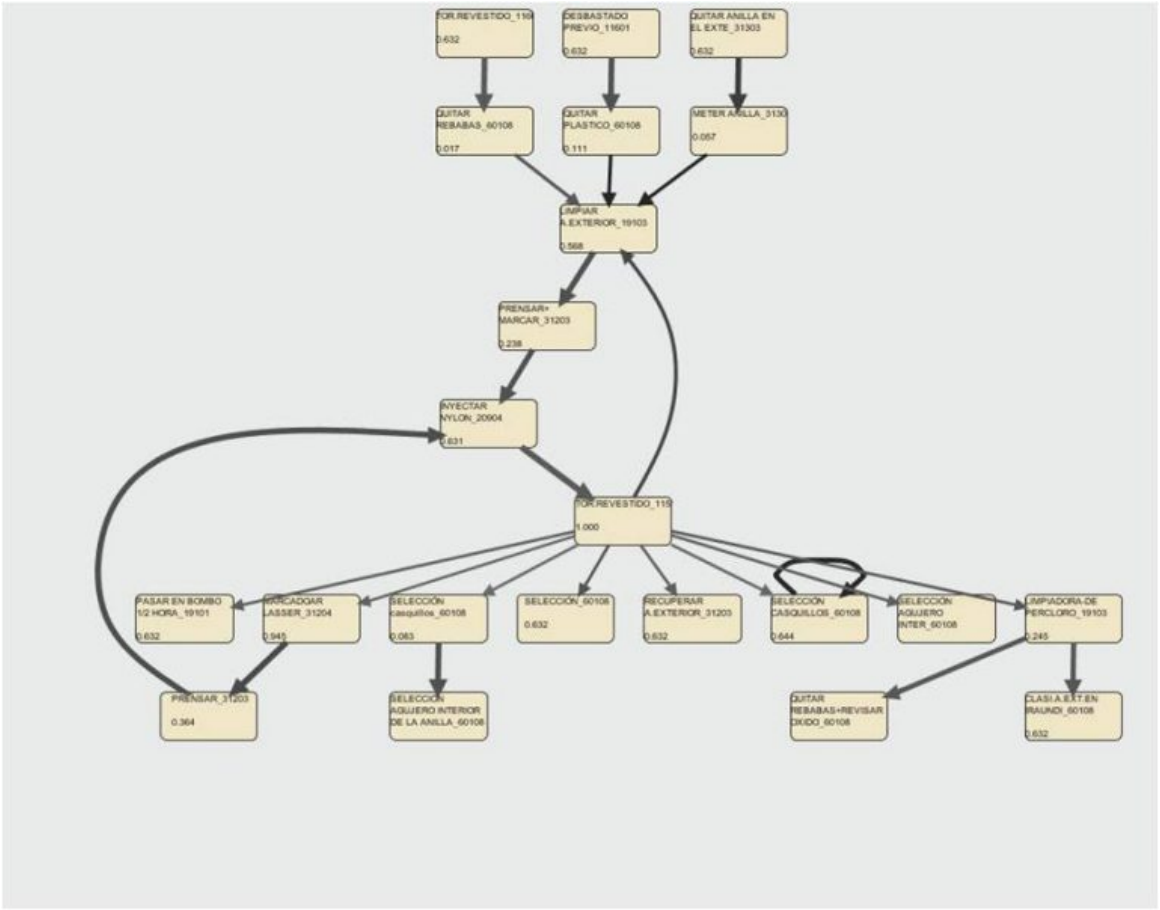

Figura 4.6: Ejemplo del algoritmo *Fuzzy Miner*

4.4. Técnicas de grafos

Me ha parecido recomendable investigar sobre técnicas de trabajo con grafos, ya que las soluciones del *Slicing* se representan en ellos y los grafos con los que se trabajan pueden ser sumamente complejos. Por ello, no esta de más señalar técnica que se puedan utilizar para sus simplificación si fuese necesario. A continuación, se señalan varias de ellas :

- **Roll Up** : Técnica utilizada para reducir las dimensiones o para subir de jerarquía en un modelo tipo árbol para reducir la información con la que trabajar.

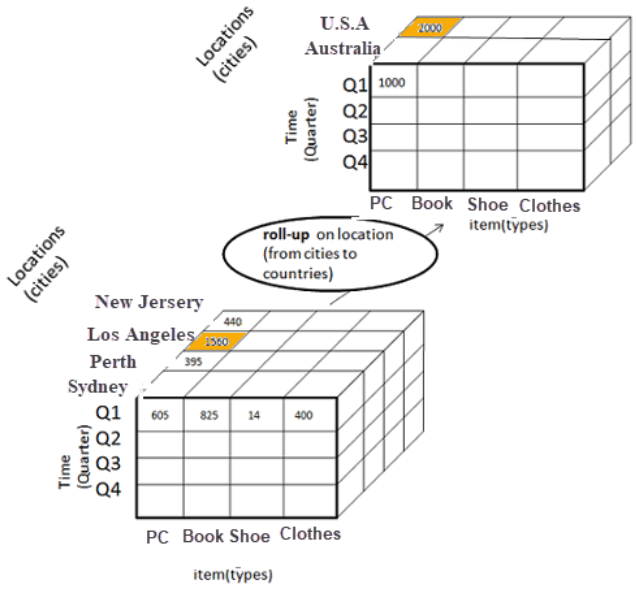

Figura 4.7: Ejemplo de *Rolling Up*

- **Drill Down** : Contrario a *Rolling Up*, se basa en aumentar las dimensiones o en bajar en el árbol de jerarquía incrementando la información utilizada.

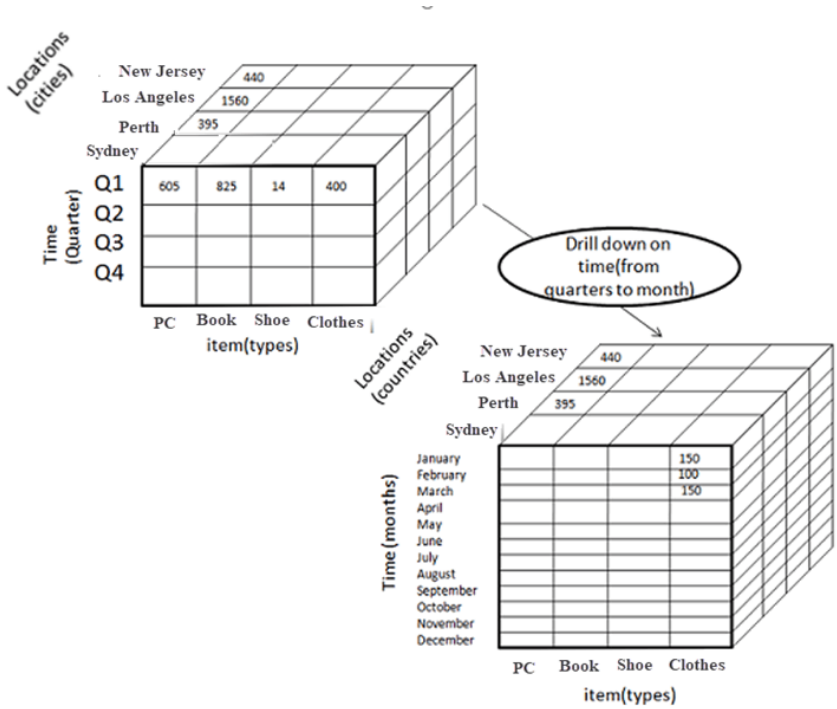

Figura 4.8: Ejemplo de *Drilling Down*

- **Slice** : Esta técnica sirve para crear un sub-elemento de una parte del total. Se suele referir a este sub-elemento como *slice* o *loncha*.

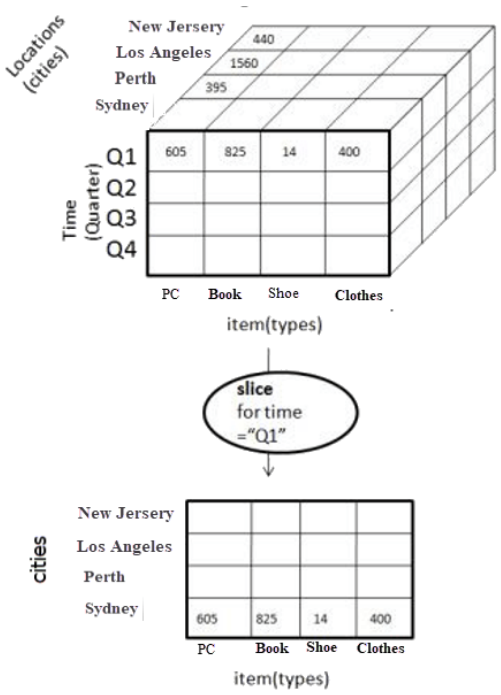

Figura 4.9: Ejemplo de *Slicing*

- **Dice** : Similar al *Slice*, pero en este caso se utilizan más dimensiones de información y generas sub-elementos mucho más complejos.

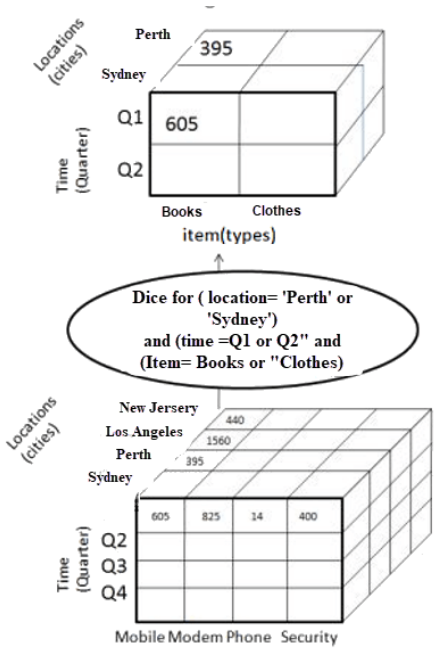

Figura 4.10: Ejemplo de *Dicing*

Actualmente ya se está utilizando una versión del *Slice* como es el **Program Slicing**, aunque la técnica **Roll Up** me parece relevante para el tema y se desarrollará un poco más a fondo.

Básicamente la idea del uso de *Roll Up* en grafos es combinar nodos para contraer el grafo buscando una reducción sin perder información.

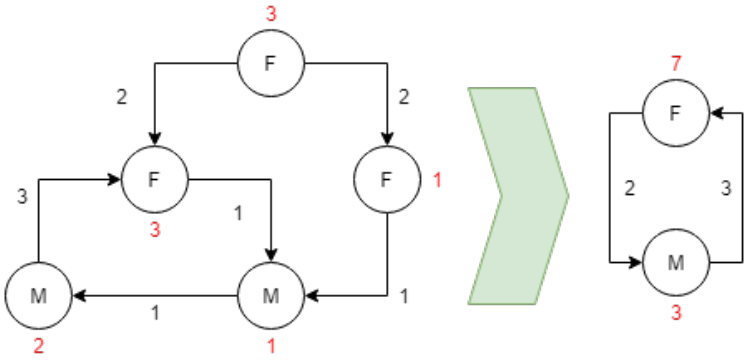

Figura 4.11: Ejemplo de *Roll Up* en grafos

La reducción del grafo vendrá determinada por el número de nodos, en nuestro caso, programas distintos que existan. El proceso será una contracción manteniendo el valor relativo de cada nodo según su importancia, por ejemplo el número de programas que dependen.

Sin embargo, puede perderse conectividad o puede conectarse zonas que no lo estaban en un principio. Además su resolución puede ser lenta si es elevada la cantidad de información y no siempre produce resultados óptimos.

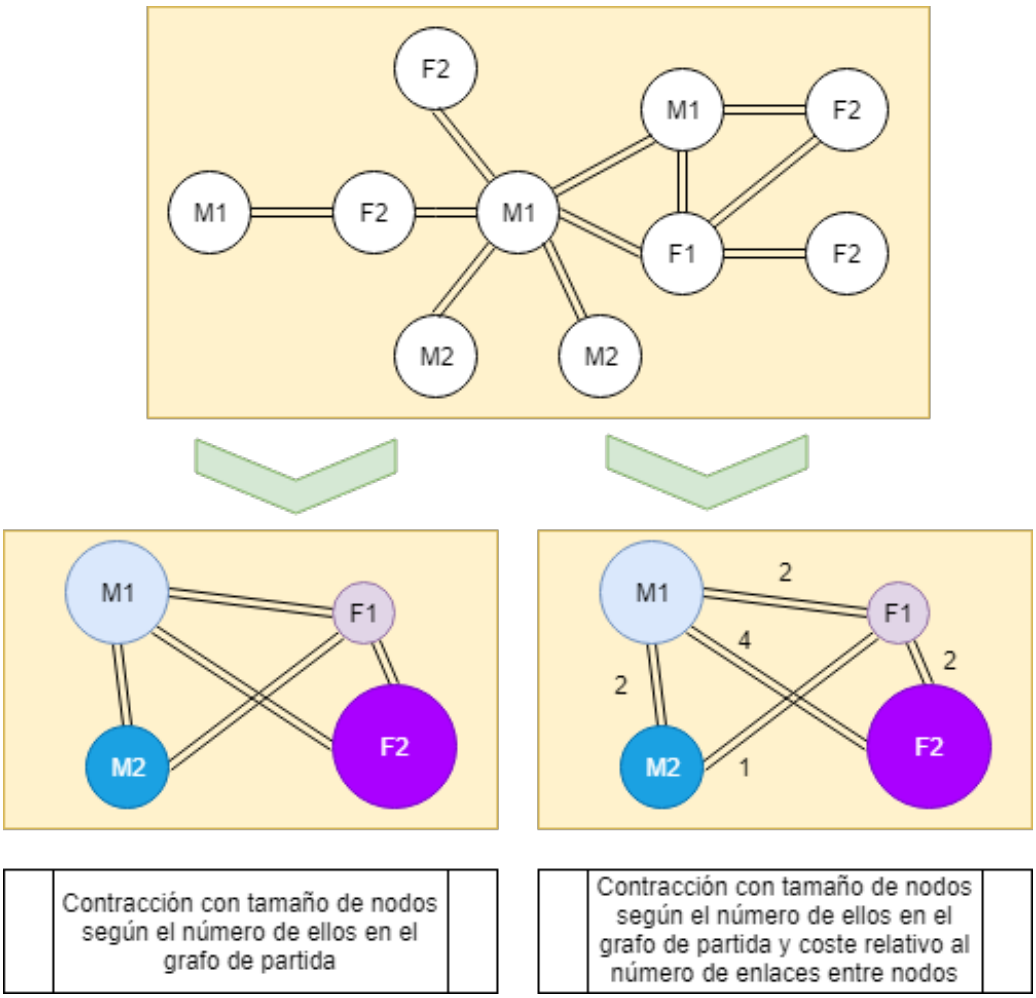

Figura 4.12: Ejemplo de *Roll Up* en grafos

4.5. Variante de *Program Slicing*

A continuación se desarrollará un enfoque sobre una variante del *Program Slicing* escrita por *Jens Krinke* del documento **Slicing, Chopping and Path Conditions with Barriers**[9].

El algoritmo de *Slicing* puede resultar complicado de entender porque te muestra el resultado sin explicar que ha ocurrido. Para poder explicarlo, el algoritmo debería de filtrar, aproximación que se recaba a continuación mediante el uso de **barriers**, las cuales no permiten el paso a ciertos nodos o por ciertas aristas durante la ejecución.

Otro enfoque es el **Chopping** que es un tipo de **Slicing con filtro**, el cual en un inicio poseía bastantes limitaciones, incluso teniendo que el elemento de origen y destino debían pertenecer al mismo procedimiento, pero existen versiones que ya no poseen dichas limitaciones.

```
1  #define TRUE 1
2  #define CTRL2 0
3  #define PB 0
4  #define PA 1
5
6  void main()
7  {
8    int p_ab[2] = {0, 1};
9    int p_cd[1] = {0};
10   char e_puf[8];
11   int u;
12   int idx;
13   float u_kg;
14   float cal_kg = 1.0;
15
16   while(TRUE) {
17     if ((p_ab[CTRL2] & 0x10)==0) {
18       u = ((p_ab[PB] & 0x0f) << 8)
             + (unsigned int)p_ab[PA];
19       u_kg = (float) u * cal_kg;
20     }
21     if ((p_cd[CTRL2] & 0x01) != 0) {
22       for (idx=0;idx<7;idx++) {
23         e_puf[idx] = (char)p_cd[PA];
24         if ((p_cd[CTRL2] & 0x10) != 0) {
25           if (e_puf[idx] == '+')
26             cal_kg *= 1.01;
27           else if (e_puf[idx] == '-')
28             cal_kg *= 0.99;
29         }
30       }
31       e_puf[idx] = '\0';
32     }
33     printf("Article: %7.7s\n   %6.2f kg    ",
             e_puf,u_kg);
34   }
35 }
```

Figura 4.13: *Chop* sobre código de ejemplo

```
1  #define TRUE 1
2  #define CTRL2 0
3  #define PB 0
4  #define PA 1
5
6  void main()
7  {
8    int p_ab[2] = {0, 1};
9    int p_cd[1] = {0};
10   char e_puf[8];
11   int u;
12   int idx;
13   float u_kg;
14   float cal_kg = 1.0;
15
16   while(TRUE) {
17     if ((p_ab[CTRL2] & 0x10)==0) {
18       u = ((p_ab[PB] & 0x0f) << 8)
             + (unsigned int)p_ab[PA];
19       u_kg = (float) u * cal_kg;
20     }
21     if ((p_cd[CTRL2] & 0x01) != 0) {
22       for (idx=0;idx<7;idx++) {
23         e_puf[idx] = (char)p_cd[PA];
24         if ((p_cd[CTRL2] & 0x10) != 0) {
25           if (e_puf[idx] == '+')
26             cal_kg *= 1.01;
27           else if (e_puf[idx] == '-')
28             cal_kg *= 0.99;
29         }
30       }
31       e_puf[idx] = '\0';
32     }
33     printf("Article: %7.7s\n   %6.2f kg    ",
             e_puf,u_kg);
34   }
35 }
```

Figura 4.14: Otro *Chop* sobre código de ejemplo

El **Barrier Slicing** o **Barrier Chopping** es un enfoque que integra la mejora de *permitir eliminar dependencias del grafo* del algoritmo **Steindl's Slicer** y la mejora de *eliminar partes del Slicing* del algoritmo **Dicing**. En el *Barrier Slicing* los nodos o aristas del grafo de dependencias pueden declararse como barreras o límites e impedir el paso por ellos. El problema del uso de barreras es que bloquean el uso de **summaries** lo que empeora el algoritmo y se debe rectificar buscando caminos de avance secundarios cada vez que una barrera impide el camino, lo que puede llevar a no siempre poderlos utilizar.

Cuando las barreras existen los nodos de origen y destino del criterio se denomina **Core Chop**. A continuación se muestra una tabla de ejemplo sobre una comparativa entre el *Core Chop* y el *Standar Chop* para evaluar el uso de las barreras.

```
1  #define TRUE 1
2  #define CTRL2 0
3  #define PB 0
4  #define PA 1
5
6  void main()
7  {
8    int p_ab[2] = {0, 1};
9    int p_cd[1] = {0};
10   char e_puf[8];
11   int u;
12   int idx;
13   float u_kg;
14   float cal_kg = 1.0;
15
16   while(TRUE) {
17     if ((p_ab[CTRL2] & 0x10)==0) {
18       u = ((p_ab[PB] & 0x0f) << 8)
             + (unsigned int)p_ab[PA];
19       u_kg = (float) u * cal_kg;
20     }
21     if ((p_cd[CTRL2] & 0x01) != 0) {
22       for (idx=0;idx<7;idx++) {
23         e_puf[idx] = (char)p_cd[PA];
24         if ((p_cd[CTRL2] & 0x10) != 0) {
25           if (e_puf[idx] == '+')
26             cal_kg *= 1.01;
27           else if (e_puf[idx] == '-')
28             cal_kg *= 0.99;
29         }
30       }
31       e_puf[idx] = '\0';
32     }
33     printf("Article: %7.7s\n   %6.2f kg    ",
             e_puf,u_kg);
34   }
35 }
```

Figura 4.15: *Core Chop* sobre código de ejemplo

Program	LOC	nodes	chops	normal chops			core chops		
				time	nodes	%	time	nodes	%
agrep	3968	22823	8100	2680	4609	20	4346	4035	17
ansitape	1744	8509	5776	651	1022	12	1580	901	10
cdecl	3879	13339	2809	241	690	5	655	501	3
ctags	2933	12961	10201	2162	1753	13	5396	1567	12
football	2261	18879	5329	1318	2285	12	4420	2060	10
gnugo	3305	7787	1444	139	2645	33	171	2232	28
simulator	4476	14705	15376	6767	4753	32	7460	4503	30

Figura 4.16: Tabla de comparación para evaluar uso de barreras

Con las barrearas se puede llegar a otra variante del *Slicing* o *Chopping* que disponga de todos o la gran mayoría de sus nodos con barreras. Con el suficiente análisis previo se podría llevar a un éxito, pero esta variante requiere un coste tremendo al tener que reconstruir el *IPDG* cada vez que la barrera cambie, siendo por esta razón preferible las variantes anteriores.

Ahora bien, *Slicing* es un algoritmo que puede responder a **¿Qué sentencias influyen sobre X?** y *Chopping* es capaz de responder a **¿Cómo una sentencia Y influye sobre X?**, pero ninguna de las dos responde al **¿Por qué una sentencia Y influye sobre X?**. Para responder esto se pueden utilizar los **Path Conditions** que dan un comportamiento similar a usar el **Conditioned Slicing**. Estos *Path Conditions* pueden crearse mediante el uso de *execution conditions*, pero el principal problema de utilizarlos recae en la escalabilidad y la eficiencia.

Capítulo 5

Conclusiones y líneas futuras

A lo largo de los capítulos anteriores se ha dado una idea general de la situación y ciertos aspectos a tener en cuenta para el análisis y modernización de aplicaciones.

Si bien está claro que el objetivo principal eran las técnicas y algoritmos encontradas, considero que las condiciones que las rodean deben ser resaltadas y dar una visión, aunque sea de manera básica, sobre ellas.

Cabe destacar, que a lo largo del tiempo dedicado a llevar a cabo esta memoria del Trabajo de Fin de Grado me he percatado de lo desigual del desarrollo de la informática. Con esto me refiero a que la informática posee multitud de áreas y, hoy en día, no hay ninguna que sea un punto muerto, todas se utilizan y dan resultados que son necesarios o, si no lo son en este momento, mejoran aspectos que en un futuro lo serán, y pese a tener esta situación tan idílica, dónde existe un gran potencial de mejora en cualquier área y que parece que siempre seguirá así, por lo menos en un futuro próximo, hay algunas de esas áreas que parecen olvidadas.

Si para este este Trabajo de Fin de grado hubiera elegido otra área de investigación como puede ser desarrollo web o desarrollo de aplicaciones móviles estoy convencido al 100 % que encontraría no solo mucha más documentación e información, sino que la mayoría sería muy reciente con antigüedades de simplemente unos meses o años. Además, encontraría muchos más videotutoriales, páginas especializadas y no tan especializadas, entre otros medios de búsqueda de información.

El punto al que quiero llegar es la falta de, por un lado, motivación, que puede ser perfectamente entendible porque siempre habrán áreas que reciban más atención que otras sea la disciplina que sea y, por otro lado, falta personal.

En general, no se si está bien o mal, pero siempre me gusta comparar la informática con la medicina. Ambas son disciplinas muy extensas, diría que las que más, y se dividen en un sin fin de áreas o campos como, por el lado de la medicina, cardiología, traumatología, medicina general, anestesiología, cirugía, ... Y así podría seguir más y más, pero lo importante y la diferencia que más me llama la atención es que ellos no son todos médicos, es decir, son cirujanos, son oftalmólogos, son médicos forenses, etc; y, en el caso de la informática, todos somos informáticos. Si, es cierto que a veces hacemos distinciones, pero en general nos denominados igual sea cual sea nuestra área y así nos conoce el público y nos conocemos entre la mayoría de nosotros.

La medicina tiene una historia muchísimo más larga que hace que quede un poco sin sentido esta comparación. Sin embargo, considero que la importancia de ambas son

parecidas y, si bien una salva vidas, la otra permite que esas vidas sean de calidad. Sin duda espero y deseo que el trabajo que se ha hecho las últimas décadas de fomentar y defender la importancia de la informática rinda frutos y, con ello, poco a poco se desarrolle más, se inculque desde antes y tengamos más compañeros en un futuro que puedan equilibrar la división de personal en las distintas áreas y fomenten esos campos que tienen un avance más lento o quizás a veces no sea lento sino continuado como es el caso de algunos apartados de la modernización de aplicativos o sistemas, pero seguro pasará en otras áreas.

Muchas de las ideas recogidas en este documento se han intentado que sean lo más reciente posible, pero en casos de algoritmos y técnicas, sobretodo en cuanto a implementaciones, no ha podido ser, en general, debido a que la mayoría de algoritmos utilizados dotan de principios del milenio. Pese a todo, ha sido satisfactorio encontrar opiniones, revisiones e investigaciones del tema recientes, de hace unos años, que vislumbra la importancia de esta área y su desarrollo creciente.

Para finalizar, me gustaría concluir diciendo que si bien no son soluciones reales que se puedan implementar inmediatamente como alternativa al algoritmo de *Program Slicing*, actualmente, utilizado por muchas organizaciones, si considero que algunas ideas aquí plasmadas pueden dar rienda a mejoras que ayuden a los fallos o no posibles implementaciones de ciertas funcionalidades del algoritmo.

Capítulo 6

Conclusions and Future Work

Reading this document you will have a complete view of the aspects and situations related to migrate monolitics applications to microservices-based architectures.

The main goal in this file is write about the techniques and the algorithms found. Also, it is provided some information about other aspects related to them.

Computer Science hasn't got a balanced development. Some areas has a really good development while others are like forgotten. Despite the fact that every area of Computer Science has got hte potential to be useful to society at the moment or in a few time.

If I had choosen investigate about web or mobile's application development, I am sure I would have found more data and very fresh one. Of course, I would have found more videotutorials and web pages too.

My point is there are two problems. On the one hand, there are areas which are more attractive. This is something normal in a lot of things around the world and maybe we should not change.

On the other hand, some have a lack of workers and that's a really tough problem.

I like to compare Computer Science with Medicine because both of them have a lot of areas and they are very vast. Nevertheless, there is a big difference between Computer Science and Medicine. If you are a developper people refers to you like Computer Science's worker. However, if you are a surgeon in Medicine or a pediatrician, people address you like that and not like medic or doctor.

Medicine has a very long history while Computer Science not so it's impossible to ask the same knowledge and recognition. Although, from my point of view both of them are significant. Medicine helps to keep us alive and it's really impresive, but Computer Science has the job to make our lives and world easier, so it's important too.

Bit by bit everyone starts to see this. I hope things keep improving so much and we can have a lot of more co-Computer Science's workers in the near future. Maybe this is the solution to balance each area's influence and perhaps investigate in a faster way the areas which seems to have been forgotten.

I try to get the most recent information, but there are cases that this is not true. For example, in the case of the algorithms, they were created around 90s or 2000. However I could find some articles and web pages with only few months old. This is a very good

sign and I hpe things keep growing more and more becuase it's a topic very useful today and it will be in the future.

The ideas I wrote in this document aren't practical solutions to implement right away, but maybe these ideas can bring up new sight to the topic helping to impove actual solutions.

Capítulo 7

Presupuesto

7.1. Sección Uno

Tipo de trabajo	N° de Horas	Coste x Horas	Coste de trabajo	Sumatorio de Coste
Tiempo de reuniones	10	10€/h	100€	100€
Búsqueda de documentación	50	15€/h	500€	600€
Recolección de información	70	15€/h	1050€	1650€
Realización de memoria	45	10€/h	450€	2100€

Cuadro 7.1: Desglose de presupuesto

** No se ha utilizado ningún software específico ni licencia, por lo que no habrán costes derivados de estos.*

Apéndice A

Ejemplos de código KDM

A.1. Ejemplo en lenguaje C

```c
int main(int argc, char* argv[]) {
    printf("Hello, World\n");
}
```

Listing A.1: Hola mundo en C

```
<?xml version="1.0" encoding="UTF-8"?>
<kdm:Segment xmi:version="2.1" xmlns:xmi="http://www.omg.org/XMI" xmlns:action="http://kdm.omg.org/action"
      xmlns:code="http://kdm.omg.org/code" xmlns:kdm="http://kdm.omg.org/kdm" xmlns:source="http://kdm.omg
     .org/source" name="HelloWorld Example">
  <model xmi:id="id.0" xmi:type="code:CodeModel" name="HelloWorld">
    <codeElement xmi:id="id.1" xmi:type="code:CompilationUnit" name="hello.c">
      <codeElement xmi:id="id.2" xmi:type="code:CallableUnit" name="main" type="id.5" kind="regular">
        <source xmi:id="id.3" language="C" snippet="int main(int argc, char* argv[]) {}"/>
        <entryFlow xmi:id="id.4" to="id.12" from="id.2"/>
        <codeElement xmi:id="id.5" xmi:type="code:Signature" name="main">
          <source xmi:id="id.6" snippet="int main(int argc, char * argv[]);"/>
          <parameterUnit xmi:id="id.7" name="argc" type="id.25" pos="1"/>
          <parameterUnit xmi:id="id.8" name="argv" type="id.9" pos="1">
            <codeElement xmi:id="id.9" xmi:type="code:ArrayType">
              <itemUnit xmi:id="id.10" type="id.19"/>
            </codeElement>
          </parameterUnit>
          <parameterUnit xmi:id="id.11" type="id.25" kind="return"/>
        </codeElement>
        <codeElement xmi:id="id.12" xmi:type="action:ActionElement" name="a1" kind="Call">
          <source xmi:id="id.13" language="C" snippet="printf(&amp;quot;Hello, World!\n&amp;quot;);"/>
          <codeElement xmi:id="id.14" xmi:type="code:Value" name="&amp;quot;Hello, World!\n&amp;quot;" type="id.19
     "/>
          <actionRelation xmi:id="id.15" xmi:type="action:Reads" to="id.14" from="id.12"/>
          <actionRelation xmi:id="id.16" xmi:type="action:Calls" to="id.20" from="id.12"/>
          <actionRelation xmi:id="id.17" xmi:type="action:CompliesTo" to="id.20" from="id.12"/>
        </codeElement>
      </codeElement>
    </codeElement>
    <codeElement xmi:id="id.18" xmi:type="code:LanguageUnit">
      <codeElement xmi:id="id.19" xmi:type="code:StringType" name="char *"/>
      <codeElement xmi:id="id.20" xmi:type="code:CallableUnit" name="printf" type="id.21">
        <codeElement xmi:id="id.21" xmi:type="code:Signature" name="printf">
          <parameterUnit xmi:id="id.22" name="" type="id.25" kind="return" pos="0"/>
          <parameterUnit xmi:id="id.23" name="format" type="id.19" pos="1"/>
          <parameterUnit xmi:id="id.24" name="arguments" kind="variadic" pos="2"/>
        </codeElement>
      </codeElement>
      <codeElement xmi:id="id.25" xmi:type="code:IntegerType" name="int"/>
    </codeElement>
  </model>
  <model xmi:id="id.26" xmi:type="source:InventoryModel" name="HelloWorld">
    <inventoryElement xmi:id="id.27" xmi:type="source:SourceFile" name="hello.c" language="C"/>
  </model>
</kdm:Segment>
```

Listing A.2: *KDM XMI* de Hola mundo en C

A.2. Ejemplo en lenguaje Cobol

```
01 StudentDetails.
02 StudentId PIC 9(7).
02 StudentName.
03 FirstName PIC X(10).
03 MiddleInitial PIC X.
03 Surname PIC X(15).
02 DateOfBirth.
03 DayOfBirth PIC 99.
03 MonthOfBirth PIC 99.
03 YearOfBirth PIC 9(4).
02 CourseCode PIC X(4).

MOVE "Doyle" To Surname
```

Listing A.3: Record en Cobol

```
<?xml version="1.0" encoding="UTF-8"?>
<kdm:Segment xmi:version="2.1" xmlns:xmi="http://www.omg.org/XMI" xmlns:action="http://kdm.omg.org/action"
      xmlns:code="http://kdm.omg.org/code" xmlns:kdm="http://kdm.omg.org/kdm" name="Record Example">
  <model xmi:id="id.0" xmi:type="code:CodeModel">
    <codeElement xmi:id="id.1" xmi:type="code:CompilationUnit">
      <codeElement xmi:id="id.2" xmi:type="code:StorableUnit" name="StudentDetails" type="id.3">
        <codeElement xmi:id="id.3" xmi:type="code:RecordType" name="StudentDetails">
          <itemUnit xmi:id="id.4" name="StudentID" type="id.23" ext="PIC 9(7)"/>
          <itemUnit xmi:id="id.5" name="StudentName" type="id.6">
            <codeElement xmi:id="id.6" xmi:type="code:RecordType" name="StudentName">
              <itemUnit xmi:id="id.7" name="FirstName" type="id.24" ext="PIC X(10)" size="10"/>
              <itemUnit xmi:id="id.8" name="MiddleName" type="id.24" ext="PIC X" size="1"/>
              <itemUnit xmi:id="id.9" name="Surname" type="id.24" ext="PIC X(15)" size="15"/>
            </codeElement>
          </itemUnit>
          <itemUnit xmi:id="id.10" name="DateOfBirth">
            <codeElement xmi:id="id.11" xmi:type="code:RecordType" name="DateOfBirth">
              <itemUnit xmi:id="id.12" name="DayOfBirth" type="id.23" ext="PIC 99" size="2"/>
              <itemUnit xmi:id="id.13" name="MonthOfBirth" type="id.23" ext="PIC 99" size="2"/>
              <itemUnit xmi:id="id.14" name="YearOfBirth" type="id.23" ext="PIC 9(4)" size="4"/>
            </codeElement>
          </itemUnit>
          <itemUnit xmi:id="id.15" name="CourseCode" type="id.24" ext="PIC X(4)" size="4"/>
        </codeElement>
      </codeElement>
      <codeElement xmi:id="id.16" xmi:type="action:BlockUnit">
        <codeElement xmi:id="id.17" xmi:type="action:ActionElement">
          <codeElement xmi:id="id.18" xmi:type="code:Value" name="&amp;quot;Doyle&amp;quot;" type="id.24"/>
          <actionRelation xmi:id="id.19" xmi:type="action:Addresses" to="id.2" from="id.17"/>
          <actionRelation xmi:id="id.20" xmi:type="action:Reads" to="id.18" from="id.17"/>
          <actionRelation xmi:id="id.21" xmi:type="action:Writes" to="id.9" from="id.17"/>
        </codeElement>
      </codeElement>
    </codeElement>
    <codeElement xmi:id="id.22" xmi:type="code:LanguageUnit" name="Cobol common definitions">
      <codeElement xmi:id="id.23" xmi:type="code:DecimalType"/>
      <codeElement xmi:id="id.24" xmi:type="code:StringType"/>
    </codeElement>
  </model>
</kdm:Segment>
```

Listing A.4: *KDM XMI* de *Record* en Cobol

A.3. Ejemplo en Java

```
class foo {
    static <T> void fromArrayToCollection(T[] a, Collection<T> c) {
        for (T o : a) {
            c.add(o);
        }
    }

    void demo() {
        String[] sa = new String[100];
        Collection<String> cs = new ArrayList<String>();
        fromArrayToCollection(sa, cs);// T inferred to be String
    }
}
```

Listing A.5: *Template* en Java

```
<?xml version="1.0" encoding="UTF-8"?>
<kdm:Segment xmi:version="2.1" xmlns:xmi="http://www.omg.org/XMI" xmlns:action="http://kdm.omg.org/action"
      xmlns:code="http://kdm.omg.org/code" xmlns:kdm="http://kdm.omg.org/kdm" name="Template Example">
  <model xmi:id="id.0" xmi:type="code:CodeModel">
    <codeElement xmi:id="id.1" xmi:type="code:ClassUnit" name="foo">
      <codeElement xmi:id="id.2" xmi:type="code:TemplateUnit" name="fromArrayToCollection&lt;T>">
        <codeElement xmi:id="id.3" xmi:type="code:TemplateParameter" name="T"/>
        <codeElement xmi:id="id.4" xmi:type="code:MethodUnit" name="fromArrayToCollection" type="id.6">
          <entryFlow xmi:id="id.5" to="id.14" from="id.4"/>
          <codeElement xmi:id="id.6" xmi:type="code:Signature">
            <parameterUnit xmi:id="id.7" name="a">
              <codeElement xmi:id="id.8" xmi:type="code:ArrayType">
                <itemUnit xmi:id="id.9" type="id.3"/>
              </codeElement>
            </parameterUnit>
            <parameterUnit xmi:id="id.10" name="c" type="id.11">
              <codeElement xmi:id="id.11" xmi:type="code:TemplateType" name="Collection&lt;T1>">
                <codeRelation xmi:id="id.12" xmi:type="code:ParameterTo" to="id.3" from="id.11"/>
                <codeRelation xmi:id="id.13" xmi:type="code:InstanceOf" to="id.75" from="id.11"/>
              </codeElement>
            </parameterUnit>
          </codeElement>
          <codeElement xmi:id="id.14" xmi:type="action:ActionElement" name="a1" kind="Compound">
            <codeElement xmi:id="id.15" xmi:type="action:ActionElement" name="a1.1" kind="Call">
              <actionRelation xmi:id="id.16" xmi:type="action:Addresses" to="id.7" from="id.15"/>
              <actionRelation xmi:id="id.17" xmi:type="action:Calls" to="id.81" from="id.15"/>
              <actionRelation xmi:id="id.18" xmi:type="action:Flow" to="id.19" from="id.15"/>
            </codeElement>
            <codeElement xmi:id="id.19" xmi:type="action:ActionElement" name="a1.2" kind="Call">
              <codeElement xmi:id="id.20" xmi:type="code:StorableUnit" name="t1" type="id.88" kind="
    register"/>
              <actionRelation xmi:id="id.21" xmi:type="action:Addresses" to="id.40" from="id.19"/>
              <actionRelation xmi:id="id.22" xmi:type="action:Calls" to="id.83" from="id.19"/>
              <actionRelation xmi:id="id.23" xmi:type="action:Writes" to="id.20" from="id.29"/>
              <actionRelation xmi:id="id.24" xmi:type="action:Flow" to="id.25" from="id.19"/>
            </codeElement>
            <codeElement xmi:id="id.25" xmi:type="action:ActionElement" name="1.3" kind="Condition">
              <actionRelation xmi:id="id.26" xmi:type="action:Reads" to="id.20" from="id.25"/>
              <actionRelation xmi:id="id.27" xmi:type="action:TrueFlow" to="id.29" from="id.25"/>
              <actionRelation xmi:id="id.28" xmi:type="action:FalseFlow" to="id.39" from="id.25"/>
            </codeElement>
            <codeElement xmi:id="id.29" xmi:type="action:ActionElement" name="a1.4" kind="Call">
              <actionRelation xmi:id="id.30" xmi:type="action:Addresses" to="id.40" from="id.29"/>
              <actionRelation xmi:id="id.31" xmi:type="action:Calls" to="id.82" from="id.29"/>
              <actionRelation xmi:id="id.32" xmi:type="action:Writes" to="id.44" from="id.29"/>
              <actionRelation xmi:id="id.33" xmi:type="action:Flow" to="id.34" from="id.29"/>
            </codeElement>
            <codeElement xmi:id="id.34" xmi:type="action:ActionElement" name="a1.5" kind="Call">
              <actionRelation xmi:id="id.35" xmi:type="action:Addresses" to="id.10" from="id.34"/>
              <actionRelation xmi:id="id.36" xmi:type="action:Reads" to="id.44" from="id.34"/>
              <actionRelation xmi:id="id.37" xmi:type="action:Calls" to="id.84" from="id.34"/>
              <actionRelation xmi:id="id.38" xmi:type="action:Flow" to="id.19" from="id.34"/>
            </codeElement>
            <codeElement xmi:id="id.39" xmi:type="action:ActionElement" name="1.6" kind="Nop"/>
            <codeElement xmi:id="id.40" xmi:type="code:StorableUnit" name="iter" type="id.41" kind="
    register">
              <codeElement xmi:id="id.41" xmi:type="code:TemplateType" name="Iterator&lt;T1>">
                <codeRelation xmi:id="id.42" xmi:type="code:InstanceOf" to="id.78" from="id.41"/>
```

```
                <codeRelation xmi:id="id.43" xmi:type="code:ParameterTo" to="id.3" from="id.41"/>
              </codeElement>
            </codeElement>
            <codeElement xmi:id="id.44" xmi:type="code:StorableUnit" name="o" type="id.3" kind="local"/>
            <actionRelation xmi:id="id.45" xmi:type="action:Flow" to="id.15" from="id.14"/>
          </codeElement>
        </codeElement>
      </codeElement>
      <codeElement xmi:id="id.46" xmi:type="code:MethodUnit" name="demo" type="id.47">
        <codeElement xmi:id="id.47" xmi:type="code:Signature"/>
        <codeElement xmi:id="id.48" xmi:type="code:StorableUnit" name="sa" type="id.49" kind="local">
          <codeElement xmi:id="id.49" xmi:type="code:ArrayType" name="ar2">
            <itemUnit xmi:id="id.50" type="id.89"/>
          </codeElement>
        </codeElement>
        <codeElement xmi:id="id.51" xmi:type="action:ActionElement" name="demo.1" kind="New">
          <codeElement xmi:id="id.52" xmi:type="code:Value" name="100" type="id.90"/>
          <actionRelation xmi:id="id.53" xmi:type="action:Reads" to="id.52" from="id.51"/>
          <actionRelation xmi:id="id.54" xmi:type="action:Creates" to="id.49" from="id.51"/>
          <actionRelation xmi:id="id.55" xmi:type="action:Writes" to="id.48" from="id.51"/>
          <actionRelation xmi:id="id.56" xmi:type="action:Flow"/>
        </codeElement>
        <codeElement xmi:id="id.57" xmi:type="code:StorableUnit" name="cs" type="id.58" kind="local">
          <codeElement xmi:id="id.58" xmi:type="code:TemplateType" name="Collection&amp;lt;String&gt;">
            <codeRelation xmi:id="id.59" xmi:type="code:ParameterTo" to="id.89" from="id.58"/>
            <codeRelation xmi:id="id.60" xmi:type="code:InstanceOf" to="id.75" from="id.58"/>
          </codeElement>
        </codeElement>
        <codeElement xmi:id="id.61" xmi:type="action:ActionElement" name="demo.2" kind="New">
          <codeElement xmi:id="id.62" xmi:type="code:TemplateType" name="ArrayList&amp;lt;String&gt;">
            <codeRelation xmi:id="id.63" xmi:type="code:ParameterTo" to="id.89" from="id.62"/>
            <codeRelation xmi:id="id.64" xmi:type="code:InstanceOf" to="id.85" from="id.62"/>
          </codeElement>
          <actionRelation xmi:id="id.65" xmi:type="action:Creates" to="id.62" from="id.51"/>
          <actionRelation xmi:id="id.66" xmi:type="action:Writes" to="id.57" from="id.61"/>
          <actionRelation xmi:id="id.67" xmi:type="action:Flow"/>
        </codeElement>
        <codeElement xmi:id="id.68" xmi:type="action:ActionElement" name="demo.3" kind="Call">
          <codeRelation xmi:id="id.69" xmi:type="code:InstanceOf" to="id.2" from="id.68"/>
          <codeRelation xmi:id="id.70" xmi:type="code:ParameterTo" to="id.89" from="id.68"/>
          <actionRelation xmi:id="id.71" xmi:type="action:Reads" to="id.48" from="id.68"/>
          <actionRelation xmi:id="id.72" xmi:type="action:Reads" to="id.57" from="id.68"/>
          <actionRelation xmi:id="id.73" xmi:type="action:Calls" to="id.4" from="id.68"/>
        </codeElement>
      </codeElement>
    </codeElement>
    <codeElement xmi:id="id.74" xmi:type="code:LanguageUnit" name="Common Java datatypes">
      <codeElement xmi:id="id.75" xmi:type="code:TemplateUnit" name="Collection&amp;lt;T&gt;">
        <codeElement xmi:id="id.76" xmi:type="code:TemplateParameter" name="T"/>
        <codeElement xmi:id="id.77" xmi:type="code:ClassUnit" name="Collection"/>
      </codeElement>
      <codeElement xmi:id="id.78" xmi:type="code:TemplateUnit" name="Iterator&amp;lt;T&gt;">
        <codeElement xmi:id="id.79" xmi:type="code:TemplateParameter" name="T"/>
        <codeElement xmi:id="id.80" xmi:type="code:ClassUnit" name="Iterator">
          <codeElement xmi:id="id.81" xmi:type="code:MethodUnit" name="iterator" kind="constructor"/>
          <codeElement xmi:id="id.82" xmi:type="code:MethodUnit" name="next"/>
          <codeElement xmi:id="id.83" xmi:type="code:MethodUnit" name="hasNext"/>
          <codeElement xmi:id="id.84" xmi:type="code:MethodUnit" name="add"/>
        </codeElement>
      </codeElement>
      <codeElement xmi:id="id.85" xmi:type="code:TemplateUnit" name="ArrayList&amp;lt;T&gt;">
        <codeElement xmi:id="id.86" xmi:type="code:TemplateParameter" name="T"/>
        <codeElement xmi:id="id.87" xmi:type="code:ClassUnit" name="ArrayList"/>
      </codeElement>
      <codeElement xmi:id="id.88" xmi:type="code:BooleanType" name="Boolean"/>
      <codeElement xmi:id="id.89" xmi:type="code:StringType" name="String"/>
      <codeElement xmi:id="id.90" xmi:type="code:IntegerType" name="Integer"/>
    </codeElement>
  </model>
</kdm:Segment>
```

Listing A.6: *KDM XMI* de *Template* en Java

Apéndice B

Información extra de ayuda sobre Program Slicing y KDM

B.1. Program Slicing : Puntos del programa

Los puntos del programa en el algoritmo de *Program Slicing* se refieren a las instrucciones, sentencias o líneas de código que se ejecutan.

En este caso, se debe intentar que estos sean lo más simples posibles y si no fuera el caso deberían refactorizarse para que se cumpliera. Por ejemplo, como se muestra a continuación, una sentencia puede transformarse de la siguiente forma para simplificarla :

```
Código inicial      ->  i : if (a = b != c) {}
                        ----------------------
Código simplificado ->  i1 : a = b
                        i2 : if (a != c) {}
```

Existen tres tipos de puntos de programa :

- **Simple** : [a := b + c]
- **De control de flujo** : [if (condición)]
- **De llamada a proceso** o **Call site** : [a := MyFunction (b, c, d)]

Todos los puntos de programa involucrados en el cálculo de *slices* corresponden con **ActionElement** de *KDM* y la implementación de cualquier conjunto involucrado en el *Slicing* toma forma de **contenedor de Java**.

B.2. Program Slicing : Tipos, con ejemplo en código

Existen distintos algoritmos a implementar con *Program Slicing* según el nivel de los *slices* puedes utilizar **Intraprocedural** o **Interprocedural** o también se puede clasificar por los tipos de *slices* a realizar los cuales son:

- **Backward Slice**

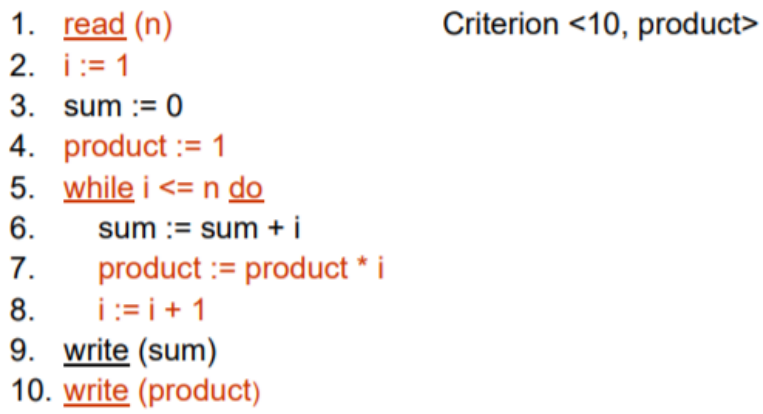

Figura B.1: Ejemplo en código de *Backward Slice*

- **Excecutable Slice**

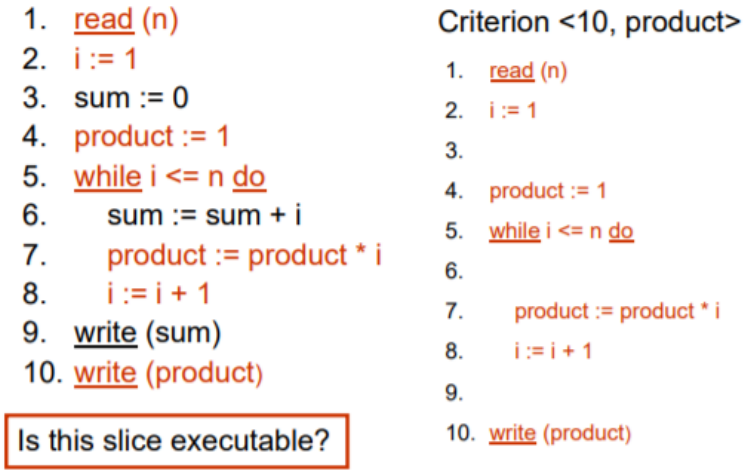

Figura B.2: Ejemplo en código de *Excecutable Slice*

- **Forward Static Slice**

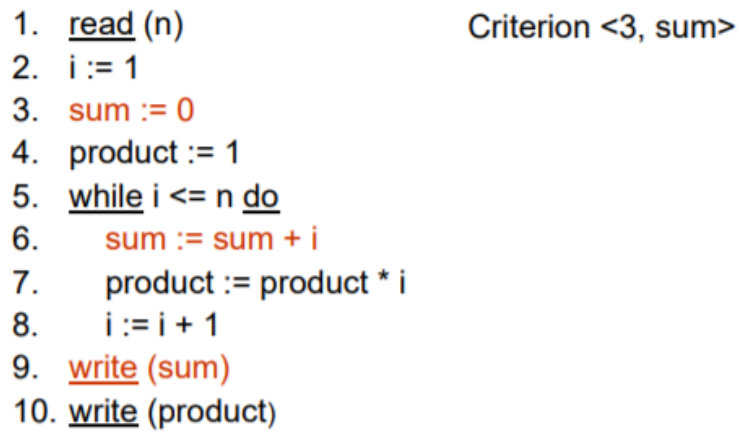

Figura B.3: Ejemplo en código de *Forward Static Slice*

- **Dynamic Slice**

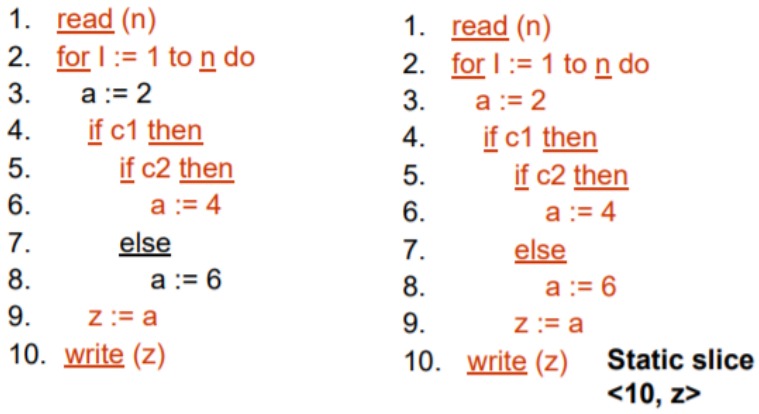

Figura B.4: Ejemplo en código de *Dynamic Slice*

- **Execution Slice**

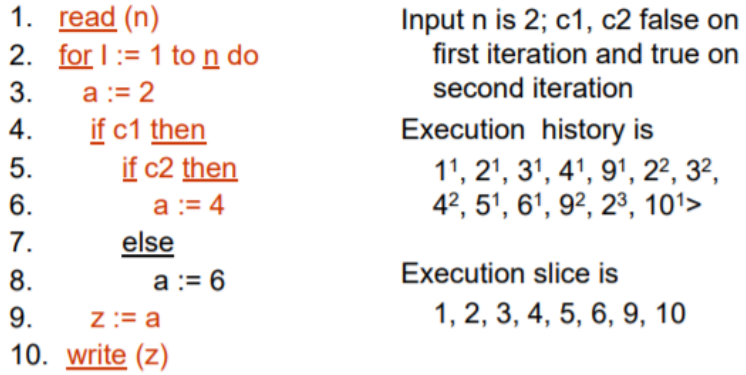

Figura B.5: Ejemplo en código de *Execution Slice*

B.3. Program Slicing : Usos

El *Program Slicing* es un algoritmo utilizado en muchas aplicaciones como por ejemplo :

- **CodeSurfer** para lenguajes C / C++ .
- **Unravel** y **Frama-C** para lenguaje C .
- **Indus** para lenguaje Java .

Además, son utilizados para solucionar diferentes tipos de problemas como :

- **Debugging**
- **Reverse Engineering**
- **Program Testing**

- **Program Comprehension**
- **Measuring Program**
- **Refactoring**

B.4. Relación de conceptos entre KDM y Código fuente

Un programa se modela mediante **CodeModel** de *KDM*. Este elemento posee **CompilationUnits** y **SharedUnits**.

- **CompilationUnits** : es una unidad de compilación o fichero de código fuente que en Cobol recibe el nombre de *program*.
- **SharedUnits** : representa un fichero importable o *copybook* en Cobol que puede tener código importado desde varios puntos del código fuente.

Por otra parte, un punto del programa es equivalente a un elemento del tipo **ActionElement** en *KDM*.

También se existe el *PDG* o *Program Dependency Graph* el cual representa el código de un procedimiento, función o método, definido como **ControlElement**, los cuales se modelan como hijos de *CompilationUnits* o *SharedUnits* y dependiendo de si corresponden a programación estructurada u orientada a objetos, se representarán mediante **CallableUnit** y **MethodUnit**, respectivamente.

En cuanto a las variables, estas corresponderán con los **DataElements** y una constante o literal con el tipo **ValueElement**.

Para concluir, las estructuras de datos se definirán como **StorableUnit** y cada uno de sus componentes como **ItemUnits**. Para las clases se utilizará la **ClassUnit** y los miembros de la clase se representarán mediante **MemberUnits**.

Bibliografía

[1] CampusMVP. Información de microservicios. https://www.campusmvp.es/recursos/post/la-muerte-de-la-locura-de-los-microservicios-en-2018.aspx. Accessed: 2019-05-28.

[2] Benchmark Consulting. Organización de modernización de aplicaciones con cobol. http://www.benchmarkconsulting.com/solutions.htm. Accessed: 2019-06-08.

[3] Sandra Seijo Fernández. Información sobre algoritmos de process mining. http://rtdibermatica.com/?p=2732. Accessed: 2019-07-01.

[4] Alexey Grigorev. Información sobre los algoritmos alpha y alpha+ mining. http://mlwiki.org/index.php/Alpha_Algorithm. Accessed: 2019-07-01.

[5] Guru99. Información sobre técnicas de grafos. https://www.guru99.com/online-analytical-processing.html. Accessed: 2019-06-22.

[6] Mary Jean Harrold. Tipos de slicing con ejemplo en código. https://www.cc.gatech.edu/~harrold/6340/cs6340_fall2009/Slides/BasicAnalysis6.pdf. Accessed: 2019-06-30.

[7] Red Hat. Modelo - punto a punto. https://www.redhat.com/es/topics/integration/what-is-integration. Accessed: 2019-05-28.

[8] KDMAnalytics. Información sobre kdm. https://kdmanalytics.com/resources/standards/kdm/. Accessed: 2019-06-02.

[9] Jens Krinke. *Slicing, Chopping, and Path Conditions with Barriers*. Kluwer Academic Publishers, 2004.

[10] Media. Información sobre business intelligence. https://blog.signaturit.com/es/que-es-business-intelligence-bi-y-que-herramientas-existen. Accessed: 2019-06-05.

[11] Microsoft. Arquitecturas de aplicaciones web. https://docs.microsoft.com/es-es/dotnet/standard/modern-web-apps-azure-architecture/common-web-application-architectures. Accessed: 2019-05-12.

[12] Patrick Nommensen. Cambio a arquitecturas de cuatro capas. https://www.nginx.com/blog/time-to-move-to-a-four-tier-application-architecture/. Accessed: 2019-05-20.

[13] Eindhoven University of Technology. Información sobre el algoritmo heuristic miner. https://www.futurelearn.com/courses/process-mining/0/steps/15639. Accessed: 2019-07-01.

[14] PowerData. Información de arquitecturas basadas en microservicios. https://blog.powerdata.es/el-valor-de-la-gestion-de-datos/cuando-y-como-deberias-utilizar-una-arquitectura-de-microservicios. Accessed: 2019-05-12.

[15] Victor Sánchez Rebull. *Algoritmo de Slicing: Especificación y estado de desarrollo*. Open Canarias, 2013.