Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Ingeniería Informática

Lenguaje de Dominio Específico para un sistema agrícola de producción vegetal

Domain-Specific Language for an agricultural plant production system

Carlos Díaz Calzadilla

La Laguna, 10 de junio de 2021

D. **Coromoto León Hernández**, con N.I.F. 78605216-W profesora Catedrático de Universidad del área de Lenguajes y Sistemas Informáticos, adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora

D. **Carlos Segura González**, con N.I.F. 78704244-S investigador tipo C del Centro de Investigación en Mátemáticas - CIMAT, Guanajuato, México, como co-tutor

I N F O R M A (N)

Que la presente memoria titulada:

"*Lenguaje de Dominio Específico para un sistema agrícola de producción vegetal*"

ha sido realizada bajo su dirección por D. **Carlos Díaz Calzadilla**, con N.I.F. 43380941-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 10 de junio de 2021

Agradecimientos

Agradezco a mi tutora del trabajo de fin de grado por haberme ayudado y guiado por el desarrollo de esta memoria y en el proceso de implementación y diseño del prototipo. También agradezco a mi familia por haberme apoyado hasta el final y hacer que esto fuera posible.

Licencia

Resumen

Este Trabajo de Fin de Grado muestra la implementación de un Lenguaje de Dominio Específico (Domain-Specific Language - DSL) para la planificación vegetal. El objetivo principal ha sido desarrollar un sistema de planificación vegetal, que se centraría en un correcto diseño e implementación de la estructura del lenguaje, siendo algo secundario otras funcionalidades como obtener una salida bien definida o realizar comprobaciones de datos. Antes que nada se comentará la metodología que se ha utilizado, desarrollo iterativo y creciente, para abarcar el desarrollo del sistema. De cara a la parte teórica se hará una introdución de los DSLs y luego se describirán los Árboles de Análisis Sintáctico. Además de esto se explicará la herramienta que se ha utilizado para llevar a cabo esta implementación que es Meta Programming System (MPS), continuando con una explicación del prototipo desarrollado, así como los pasos llevados a cabo para su implementación. Cuando ya todo esto esté explicado se comentará como se realizó la validación del sistema. Para que los resultados proporcionados por el sistema sean válidos debe completar una serie de pruebas realizadas.

Palabras clave: lenguaje dominio específico, cultivo, sistema, planificación.

Abstract

This Final Degree Project shows the implementation of a Domain-Specific Language (DSL) for crop planning. The main objective has been to develop a plant planning system, which would focus on a correct design and implementation of the language structure, with other functionalities being secondary, such as obtaining a well-defined output or performing data checks. First of all, the methodology that has been used, iterative and growing development, to cover the development of the system, will be discussed. Regarding the theoretical part, an introduction of the DSLs will be made and then the Syntactic Analysis Trees will be described. In addition to this, the tool that has been used to carry out this implementation, which is the Meta Programming System (MPS), will be explained, continuing with an explanation of the developed prototype, as well as the steps carried out for its implementation. When all this is explained, it will be discussed how the system validation was carried out. For the results provided by the system to be valid, you must complete a series of tests performed.

Keywords: domain-specific language, planification, crop, system,

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

La agricultura comenzó una vez que las personas empezaron a plantar hierbas por sus semillas en el Cercano Oriente. Con el paso de los años ha ido evolucionando tanto la estrategias de cultivo, las herramientas utilizadas, los tipos de cultivo, etc. Hoy en día con todas las tecnologías que se tienen se ha logrado la mecanización agraria (cosa que era impensable hace 200 años), que se considera uno de los 20 mayores logros de la ingeniería del siglo XX. No solo los sistemas agrícolas han mejorado, también la difusión de la radio y la televisión (medios de comunicación),así como de la informática, son de gran ayuda, al facilitar informes meteorológicos, estudios de mercado, etc.

Es cada vez más habitual ver como muchas profesiones se digitalizan. En el caso de la agricultura, pese a la antigüedad del sector, también se suma a este grupo, hasta el punto de tener hoy en día una inmensa cantidad de aplicaciones para poder mejorar el rendimiento del trabajo en el campo. De ahí surge el software de gestión agrícola que ayuda a resolver muchas tareas de diferente complejidad. Hay muchas ventajas de la introducción del software en la agricultura, pero entre ellas destacan:

- Mayor cantidad y calidad de información: Uno de los aspectos fundamentales en este sector es estar bien informados. Gracias a este tipo de aplicaciones se puede obtener información metereológica en el tiempo de cultivo, las novedades en las técnicas de cultivo utilizadas, etc.
- Coordinación entre trabajadores: Esta también puede ser mejorada gracias al software a partir de aplicaciones y herramientas que los permitan estar contectados, realizar un reparto más eficiente y una planificación de las tareas más cómoda y rápida.

- Organización en la gestión del trabajo: Esta es una gran comodidad ya que hoy en día con un solo click se tiene a completa disposición las tareas, facturas, horarios,... Además, todo esto supone un gran ahorro de tiemo frente a tener todos estos datos a papel.
- Comodidad y productividad: En este campo, con el software se permite realizar un mayor número de tareas en un tiempo más corto, por ende, se aumenta la productividad y el beneficio obtenido.

Además de las ventajas que puede tener un software agrícola, cabe mencionar que facilita el proceso de gestión de una explotación agrícola, abarcando actividades como el mantenimiento y registros, rotación de cultivos, control de plagas, previsión de rendimientos, gestión de riesgos, etc. Según su uso y el tipo de herramienta el software agrícola se puede clasificar en diferentes grupos:

- La gestión de explotación agrícola contiene la gestión y finanzas de la empresa, la gestión agrícola y el sistema de planificación de los recursos empresariales.
- La gestión de cultivos abarca tanto el seguimiento del crecimiento de las plantas como la predicción del rendimiento agrícola.
- La gestión y finanzas cubre cuestiones generales de la gestión de la explotación agrícola como el mantenimiento del inventario, control y orientación de los trabajadores,...
- La planificación de los recursos empresariales, permite obtener información sobre el estado de los cultivos, trabajo de los trabajadores y la actividad agrícola.
- La agricultura de precisión y el análisis predictivo aportan la optimización de los insumos y incrementan el rendimiento del campo.
- La monitorización y sensores de campo cubren una gran cantidad de características como control de plagas, gestión de malas hierbas, fertilización y el riego.
- Las plataformas de datos a gran escala son software que va a permitir acumular información que provenga de múltiples fuentes sin interpretarla.
- Las plataformas para el análisis de grandes datos.

- El software de exploración.

Los sistemas de información de gestión agrícola tienen alta importancia, sobre todo para apoyar el proceso de toma de decisiones en el negocio agrícola. Desarrollar un sistema de este tipo es complejo y requiere un diseño e implementación correcta de modelos para poder apoyar la comprensión, la mejora de la comunicación y el análisis de las decisiones de diseño.

Tradicionalmente, se solían usar los lenguajes de programación de propósito general(General Purpose Languages - GPL) para desarrollar estos programas para la agricultura. Sin embargo, el uso de los GPL, hace que a veces haya mayor complejidad y dificultades a la hora de expresar algunos conceptos dentro de la agricultura. Por ello, el objetivo de este trabajo es diseñar e implementar un prototipo de lenguaje para un dominio específico (Domain Specific Language - DSL), en concreto el de la gestión de la producción agrícola vegetal. Puesto que un DSL se basa en abstracciones que se encuentran directamente ligadas con el dominio para lo que está construido el lenguaje.

En este trabajo se va a perseguir una implementación de un DSL para un sistema de planificación vegetal centrada en una estructura bien definida, posicionando en un segundo plano la calidad de la salida o las comprobaciones de los valores introducidos. Con esto se persigue desarrollar un sistema que facilite la planificación de cultivos, que sea eficiente y que no sea necesario tener una alta formación en este tipo de tecnologías para el uso, es decir, que esté a un nivel alto de abstracción.

1.1. Motivación

La motivación principal de este Trabajo de Fin de Grado es poder implementar un DSL con unas características específicas que no se encuentran presentes en lenguajes de este tipo, como lo son tener una recogida de datos desde ficheros, implementar algunas operaciones básicas y tener un diseño e implementación bien definidos entre otros.

Asimismo, se persigue mejorar las herramientas que se utilizan a día de hoy en el sector de la agricultura. Teniendo un Lenguaje de Dominio Específico que nos permita definir de una manera sencilla y eficiente las planificaciones de producción vegetal se podrá reducir los esfuerzos y costes invertidos en el sector primario aumentando a la eficiencia de los mismos.

Adicionalmente, este trabajo nos permitirá comprender de una manera más clara como se desarrollan y como funcionan los Lenguajes de Dominio Específico y ver cuales son sus ventajas y desventajas frente a los lenguajes de propósito general.

1.2. Objetivo

En consecuencia, el objetivo de este trabajo es diseñar e implementar el prototipo de un Lenguaje de Dominio Específico enfocado a realizar la planificación de un sistema de producción vegetal. Además hubo algunos objetivos secundarios como lo son el correcto diseño de la entrada, poder realizar una carga de datos a partir de fichero y poder implementar algunas operaciones básicas.

1.3. Metodología

Para abordar el desarrollo del sistema se ha aplicado el enfoque metodológico denominado desarrollo iterativo y creciente (o incremental). Este permite planificar el proyecto a realizar en distintos bloques que se denominan iteración, teniendo en cada una de estos bloques distintas tareas que se irán repitiendo hasta obtener el resultado final de la iteración. El proceso consiste de dos etapas fundamentales:

- Etapa de inicialización: En esta etapa se crea una nueva versión del sistema y tiene como objetivo crear un producto con el que se pueda interactuar.
- Etapa de iteración: Aquí se realiza un rediseño e implementación de una tarea determinada y un análisis de la versión más reciente del sistema.

Una de las ventajas que tiene el uso de esta metodología es que provee de soporte para determinar la efectividad de los procesos y la calidad del producto. Además, permite mejorar y ajustar el proceso a un ambiente en particular.

En los siguientes capítulos, se describe con detalles el trabajo que ha sido necesario para abordar cada una de las tareas. En el Capítulo 2, Antecedentes y estado actual del tema, se podrá observar los primeros pasos para abarcar el proyecto. En el Capítulo 3, Herramientas y Tecnologías, se especifican cada una de las herramientas, lenguajes y tecnologías empleadas durante el desarrollo. El Capítulo 4 está dedicado a mostrar los conceptos básicos de como se ha realizado e implementado el prototipo del sistema. En los capítulo 5 y 6 - Conclusiones y líneas futuras - y - *Summary and Conclusions*-, se encuentran las reflexiones obtenidas de haber realizado este proyecto así como las mejoras que podrían realizarse en un futuro. Por último, en el Capítulo 7, Presupuesto, se encuentra una estimación del coste que ha implicado la realización del prototipo de esta aplicación.

Capítulo 2

Antecedentes y estado actual del tema

Lo primero que se realizó fue una revisión bibliográfica sobre Lenguajes de Dominio Específico para un sistema agrícola de producción vegetal.

Para obtener buenos resultados, se realizó un conjunto de búsquedas en distintas bases de datos. Luego, se utilizó la herramienta proporcionada por la Universidad de la Laguna llamada puntoQ. Gracias a esto, se pudo buscar en diferentes bases de datos de datos cómo Scopus, Wos y Dialnet. Cabe destacar que se buscó en Google Schoolar para mejorar la diversidad y calidad de resultados.

2.1. Metodología

Como ya se había mencionado anteriormente se hizo un conjunto de búsquedas en distintas bases de datos para así poder ampliar la cantidad y calidad de los resultados. Por otro lado, mencionar que las búsquedas se realizaron, tanto en bases de datos inglesas como españolas, pero en el caso de las españolas no se encontró ninguna información sobre el tema. Sin embargo, en las bases de datos inglesas se consiguió encontrar un estudio similar y otros bastante parecidos.

2.2. Artículos seleccionados

Se han realizado búsquedas en español y en inglés. Las palabras clave que se han utilizado han sido:

- lenguaje dominio específico - domain-specific language
- granja - farm
- cultivo - crop

- sistema - system

Con ello, la mayor parte de las búsquedas dieron resultados trabajos de Lenguajes de Dominio Específico relacionados con los paisajes. Como se mencionó anteriormente, también se realizó una búsqueda en bases de datos españolas, para ver si existía algún proyecto relacionado, pero no se encontró ninguno que implementara un DSL para un sistema de producción vegetal. Por otro lado, en las bases de datos inglesas se pudieron obtener resultados, entre los que destacan los siguientes artículos:

- Para este primer resultado, se encontró un sistema que, a partir de un lenguaje de dominio específico facilitaba la gestión de una granja [7]. A continuación se expone un pequeño resumen sobre el artículo:

  La agricultura de precisión es un concepto de gestión agrícola que aplica tecnología de información avanzada y los principios correspondientes para aumentar la producción y los beneficios económicos, a menudo también con la intención de reducir el impacto en el medio ambiente.

  Dentro de esta, los sistemas de información de gestión agrícola tienen alta importancia, sobre todo para apoyar el proceso de toma de decisiones en el negocio agrícola. Desarrollar un sistema de este tipo es complejo y requiere un diseño e implementación correcta de modelos.

  Por ello, para poder llevar a cabo este sistema se va a utilizar un DSL ya que, está optimizado para una clase determinada de problemas en un dominio de aplicación particular. Generalmente los DSL son pequeños y no tan complejos como lo sería un lenguaje de programación, además que se han aplicado con éxito en diferentes dominios de aplicaciones.

- Como segundo resultado tenemos un sistema web de bajo costo para monitorizar y controlar un invernadero agrícola [8]:

  El sistema realiza una monitorización mediante la captura de variables climatológicas a partir de sensores y microcontroladores, que se encuentran instalados en el interior del invernadero. Esto surge debido a una necesidad en el rubro de la agricultura en una zona desértica como lo es la Región de Arica y Parinacota de Chile, donde las áreas de cultivo mediante invernaderos o mallas antivirus no son despreciables. Las tecnologías implementadas en el sistema no están muy extendidas pero, dado las características que poseen facilitan el funcionamiendo del sistema, como por ejemplo su conexión a Internet.

Además, el sistema fue sometido a diversas pruebas en un laboratorio de cultivo de tejidos vegetales, dando resultados favorables y verificando que el sistema está totalmente operativo, las cuales fueron supervisadas por un ingeniero agrícola.

- Para acabar, encontramos otro que habla de un software creado específicamente para el sector agropecuario [1]:

  El objetivo que persiguen es elaborar un diagnóstico de forma aproximada de la situación actual entre el sector informático y el sector agropecuario en Pampeana. Asimismo tiene como objetivo secundario analizar que grado de potencialidad y factibilidad poseía la conformación o consolidación de una oferta de software especializado. El último objetivo es realizar ciertas recomendaciones básicas a la hora de considerar una política de promoción de esta vinculación intersectorial.

A continuación, se muestra una tabla de comparación 2.1 entre las diferentes aplicaciones que han sido diseñadas en los artículos [7] y [8] , hablando del tipo de software, si la aplicación es comercial o no, el lenguaje de programación en el que se ha implementado...

Aplicación	Tipo de Software	Modelo de desarrollo de software	Aplicación comercial	Servicio online	Lenguaje de programación
Artículo 1 [7]	Software libre	No es código abierto	Si tiene (PC)	Si	Java
Artículo 2 [8]	Software libre	Es código abierto	Si	Si	Javascript

Tabla 2.1: Tabla de comparación de aplicaciones

Como se ha visto a lo largo de este capítulo, los Lenguajes de Dominio Específico no se encuentran tan demandado como parece. A pesar de existir ciertas implementaciones y ciertos intentos de llevar estas tecnologías al mundo agrícola, aún falta invertir muchos esfuerzos para que este tipo de aplicaciones se extiendan por este sector.

Por ello con este sistema se persigue aportar una nueva forma de almacenar los datos, una implementación sencilla para los usuarios, un ejemplo dirigido de un sistema a pequeña escala, el uso de nuevas herramientas para implementar DSL y así ver la potencia y eficacia de las mismas. Además, esto va a ayudar con el manejo de la herramienta para resolver problemas como la conexión con bases de datos externas, la comprensión de algunos plugins, y demás. No todo se centra en resolución de errores o problemas, también puede ayudar a crear una implementación usando este sistema como base.

Capítulo 3

Herramientas y tecnologías de desarrollo

Para este capítulo se va a poner especial atención en explicar qué se entiende por Lenguaje de Dominio Específico. Además se definirán qué son los Árboles de Análisis Sintácticos y, para acabar, en la sección 3.3 se va a introducir una herramienta de modelado que permite la generación automática de este tipo de lenguajes, así como hablar de sus características y de como funciona.

3.1. Lenguaje de Dominio Específico (*Domain Specific Language - DSL*)

Un Lenguaje de Dominio Específico es un lenguaje de programación con un nivel superior de abstracción optimizado y especializado en resolver una clase específica de problemas. Crear un DSL vale la pena cuando se quiere resolver un tipo de problema en particular o permite expresar soluciones de una forma más clara que con otros lenguajes más generales. En muchos casos, dada la abstracción que poseen, los DSL están destinados no para ser usados por expertos en software, sino por no programadores que son versados en el dominio de aplicación del DSL.

Usar DSL ofrece múltiples ventajas. El beneficio más evidente es que, una vez que se tiene un lenguaje y un motor de transformación, el trabajo en el aspecto del desarrollo de software (a partir del DSL) se vuelve mucho más eficiente, ya que no se tiene que realizar algunas tareas manualmente. Además, usar un DSL puede mejorar la calidad del producto creado, ya sea, teniendo menos errores, mejor cumplimiento de la arquitectura y un mantenimiento más sencillo.

Por otro lado se encuentran los Lenguajes de Propósito General (como Java, C o Ruby), que son lenguajes de programación cuyos procedimientos, instrucciones y estructuras de datos están diseñados para resolver todo tipo de problemas. Además de esto, los GPL pueden ser usados para varios propósitos como acceso a bases de datos, comunicación entre computadoras, comunicación entre dispositivos, captura de datos, cálculos matemáticos, diseño de imágenes o páginas. A diferencia de los DSL, estos lenguajes siempre son turing-completos, es decir, son capaces de expresar cualquier algoritmo, y pudiendo ser aplicados a cualquier dominio. Sin embargo, utilizar un GPL para atacar problemas muy específicos, puede significar un mayor esfuerzo del que uno desearía, dado que las abstracciones que estos dominios plantean no se encuentran soportadas (de forma nativa) por el lenguaje; no son primitivas

Tanto los GPL como los DSL, se encuentran orientados a resolver problemas, sin embargo, se diferencian en muchos aspectos, tal y como vemos en la Tabla 3.1.

	GPL	DSL
Turing-completo	Sí	No
Dominio	General	Específico
Conceptos	Generales: función (en funcional), clase, objeto, método (en OOP), variable, predicado (en lógico), etc.	Del (único) dominio

Tabla 3.1: Tabla de comparación entre GPL y DSL

3.2. Árbol de Análisis Sintáctico (*Abstract Syntax Tree - AST*)

Un Árbol de Análisis Sintáctico es una estructura de datos que se usa en los compiladores, debido a que permite representar la estructura del código de un programa. Un AST suele ser el resultado del analizador sintáctico en la fase de un compilador. El AST tiene diferentes propiedades importantes:

- En comparación al código fuente el AST no incluye algunos elementos (como elementos de puntuación, corchetes, puntos y coma,...).

- El AST puede ser modificado y mejorado con propiedades para cada elemento que contiene. En el caso del código fuente es imposible hacer este cambio, ya que esto implicaría modificarlo.
- Suele contener información adicional sobre el programa, debido a las consecutivas fases de análisis del compilador.

Los ATSs son importantes debido a que los lenguajes suelen ser ambiguos por naturaleza. Para poder evitar esta ambigüedad los lenguajes de programación se encuentran especificados con una gramática libre de contexto (*Context Free Grammar - CFG*). Sin embargo, hay algunos aspectos de los lenguajes de programación que no pueden ser expresados por una CFG.

Aunque hay otras estructuras que están relacionadas con el funcionamiento del compilador, el AST tiene una única función. Para poder comprender dicha función y como se obtiene se van a explicar las diferentes fases de análisis que tiene un compilador.

3.2.1. Análisis

La fase de análisis trata de la comprobación de la corrección del programa fuente, todo esto según la definición del lenguaje. Para poder enviar todas esta información, el trabajo se divide en diferentes tareas, que es lo que vamos a denominar como fases y son:

Análisis léxico

En esta primera fase, se hace una lectura del programa fuente de izquierda a derecha, realizando a su vez, una agrupación en componentes léxicos (también denominados como tokens), que son secuencias de caracteres con un significado. Asimismo, todos los espacios en blanco, comentarios y el resto de la información innecesaria se elimina del programa fuente. Entre las diversas clasificaciones de los componentes léxicos tenemos los identificadores, operadores, constantes numéricas, signos de puntuación,...

Como la tarea que se realiza es un caso de coincidencia de patrones especial, se necesita de métodos de especificación y reconocimiento de patrones. Por ello se usan como herramienta principal los autómatas finitos que acepten expresiones regulares.

Análisis sintáctico

Para esta fase, se determina si la secuencia de tokens sigue la sintaxis definida por el lenguaje y obtiene la estructura jerárquica en forma de árbol (AST), donde los nodos van a representar las construcciones de alto nivel del lenguaje. Además se van a determinar las relaciones que van a tener los componentes léxicos (es lo mismo a realizar un análisis gramatical sobre una frase en lenguaje natural). La estructura sintáctica va a estar definida a partir de gramáticas independientes del contexto.

Análisis semántico

Para acabar, en el análisis semántico se realizan las comprobaciones necesarias sobre el árbol sintáctico para determinar el correcto significado del programa. Entre las tareas principales tenemos:

- Verificación e inferencia de tipos en asignaciones y expresiones.
- Declaración del tipo de variables antes de su uso.
- Declaración de funciones antes de su uso.
- Correcto uso de operadores.
- El ámbito de las variables.
- Correcta llamada de funciones.

El análisis semántico suele agregar atributos como tipos de datos a la estructura del árbol sintáctico.

3.3. MPS (*Meta Programming System*)

Primero que todo mencionar que para crear este sistema se utilizó una herramienta llamada MPS (Meta Programming System) que, a diferencia de los lenguajes de programación tradicionales con sintaxis y semántica estrictas, MPS permite crear un lenguaje desde cero y que sus usuarios lo modifiquen o amplíen. Uno de los mayores problemas de ampliar lenguajes son los analizadores o parsers, que se encargan de analizar las cadenas para crear estructuras de datos. Los parsers restringen la representación persistente de código a una única anotación, lo que significa que no pueden combinarse fácilmente con otros parsers, y esto evita la modularización del lenguaje.

Una de las alternativas que se propone en MPS es: El código siempre se mantenga en lo que se denomina un árbol de sintaxis abstracta (AST), una estructura de datos que consta de nodos con un conjunto de atributos (propiedades, hijos y referencias) y describe completamente el código del programa. Entonces se puede llegar a la conclusión que MPS se encarga de visualizar el AST de una forma sencilla y proporcionar una herramienta para una edición efectiva.

Al crear un lenguaje en MPS, se definen las reglas de edición de código y especifica cómo se presentará el código al usuario. Además de esto, permite especificar el sistema de tipos del lenguaje y sus restricciones, cómo un conjunto de reglas aplicadas a su lenguaje. Todas estas características juntas permiten a MPS verificar el código del programa sobre la marcha, para hacer que programar con el nuevo lenguaje sea fácil y menos propenso a errores.

Por último, MPS usa un enfoque generativo. Esto significa que puede definir generadores para que su lenguaje transforme las entradas de los usuarios finales en un lenguaje más convencional, normalmente de uso general. Actualmente, MPS es especialmente bueno generando código Java, aunque no solo en esto. También puede generar C, C sharp, XML, FHTML, PDF, LaTeX, JavaScript y más.

3.3.1. DSL en MPS

Con MPS se pueden definir editores a medida para un nuevo lenguaje y hacer que el uso de estos DSL sea mucho más simple y eficaz.

A la hora de desarrollar un DSL es mejor que se establezcan las sentencias con poca libertad, es decir, supongamos que tenemos un DSL que se encarga de dibujar figuras sobre un canvas, cuando llegue una persona para intentar usar este DSL, no va a conocer los parámetros necesarios para escribir correctamente la sentencia. En cambio, si las sentencias que definimos las hacemos fijas (poco variables) entonces conseguiremos que sea mucho más fácil de utilizar.

Además, sabemos que los DSL no requieren de un analizador o parser, por lo que se pueden crear lenguajes sensibles al contexto. Esto quiere decir que, si por ejemplo se usa java cómo lenguaje para crear el DSL, entonces java puede tener definida la sentencia break, pero nosotros para nuestro lenguaje le podemos dar una definición diferente, evitando que haya problemas, ya que conoce directamente que está definido y accede directamente al AST dado que no tiene ningún parser.

Esto no es todo, MPS permite combinar varios lenguajes sin tener la preocupación de tener que combinar gramática. Esto sobre todo da una alta flexibilidad y da una mayor facilidad a la hora de programar.

Para poder definir un DSL, en MPS, primero es necesario definir un proyecto. Una vez que lo tengamos entonces ya se podrá crear el DSL. Con este una vez definido vamos a tener diversas posibilidades. A nuestro DSL se le van a poder añadir plugins, estos van a permitir que se puedan añadir herramientas externas que ni están implementadas directamente en MPS (como en el caso de querer usar tablas, funciones matemáticas,...). Además de los plugins, se pueden añadir dependencias a otros lenguajes ya creados, para así usar su sintaxis. Por otro lado, dentro de un lenguaje tenemos varios elementos importantes (que se explicarán más adelante en el apartado de características), entre los que podemos destacar dos:

- Concepts: Son los elementos que van a representar los nodos de nuestro lenguaje. Si se tuvieran que definir como en un lenguaje de programación se asemejarían a las clases. En estos elementos es donde vamos a definir las dependencias y las características de los nodos.
- Editor: Va a permitir dar forma visual a los conceptos que se hayan creado.

3.3.2. Características

Editor de proyecciones

Un editor de proyecciones permite al usuario editar la representación de código de Árbol de sintaxis abstracto (Abstract syntax tree - AST) de una forma eficiente. Puede replicar el comportamiento de un editor textual para notaciones textuales, un editor de diagramas para lenguajes gráficos, un editor tabular para editar tablas, etc.

Cómo ya se había dicho, a la hora de crear un lenguaje, también se crea o se establece la forma de editar dicho lenguaje, pudiendo ser de múltiples formas no solo una.

Compatibilidad de IDE

MPS es un potente IDE que facilita y permite utilizar el máximo potencial de los DSL. Algunas de las funcionalidades que incluye son finalización de código, navegación, refactorización, detección de errores, arreglos rápidos,

depuración de DSL, versionado de lenguajes con migraciones automáticas e integración con sistemas de control de versiones.

El IDE de MPS permite una mayor facilidad para tomar contacto con los DSL. Primero una vez se crea el proyecto, se nos crea una interfaz parecida a la siguiente:

Una vez que se ha creado un lenguaje aparecen varios elementos claramente diferenciados, cada uno con su parte de funcionamiento y son:

- **structure**: MPS le otorga todos los medios para definir la estructura del lenguaje. Aquellos elementos que se necesitan para definir nuestro lenguaje son los Conceptos y la estructura del lenguaje que expone conceptos e interfaces de conceptos, así como sus miembros: propiedades, referencias e hijos.

  Ahora vamos a explicar un poco qué son los Conceptos. Un Concepto define la estructura de una instancia de concepto, un nodo del futuro AST que representa el código escrito en su lenguaje. El Concepto dice qué propiedades pueden contener los nodos, a qué nodos se puede hacer referencia y qué nodos secundarios están permitidos.

  Además de los Conceptos, tenemos las Interfaces de concepto. Las Interfaces de conceptos representan rasgos independientes, que pueden heredarse e implementarse mediante muchos conceptos diferentes. Para poder entenderlo mejor, vamos a crear un ejemplo. Si tenemos una instancia de Concepto con un nombre para identificarlo, puede implementar la interfaz INombreConcepto en su Concepto y obtendrá la propiedad del nombre más el comportamiento asociado y las restricciones agregadas a su Concepto.

- **editor**: Una vez que ya tengamos definida la estructura de nuestro lenguaje, el siguiente paso será permitir que los desarrolladores puedan desarrollar ASTs fácilmente con dicho lenguaje. La descripción de un editor consta de descripciones de las celdas que contiene. Llamamos a estas descripciones "modelos celulares". Por ejemplo, si desea que su editor consista en una celda única con texto no modificable, cree en la descripción de su editor un modelo de celda constante y especifique ese texto. Si desea que su editor conste de varias celdas, cree un modelo de celda de colección y luego, dentro de él, especifique modelos de celda para sus elementos. Etcétera.

- **constraints**: El lenguaje de estructura a veces puede ser insuficiente para expresar restricciones avanzadas sobre la estructura del len-

guaje. El aspecto constraints le brinda una forma de definir tales restricciones adicionales. Las constraints se pueden usar para indicar qué concepto se debe usar en su lugar cuando se necesita crear un nodo del concepto abstracto. A la hora de definir las constraints tenemos que tener en cuenta que se pueden agrupar en children, parents, ancestors o también pueden ser nodos raíz del modelo.

- **behaviour**: Durante la manipulación del árbol de sintaxis, las operaciones comunes a menudo se extraen a métodos de utilidad para simplificar la tarea y reutilizar la funcionalidad. Es posible extraer dichas utilidades en métodos estáticos o crear envoltorios de nodos que contengan el código de la utilidad en métodos virtuales. Sin embargo, en MPS está disponible una solución mejor: el aspecto del lenguaje behaviour. Permite crear métodos de instancia virtuales y no virtuales, métodos estáticos y constructores de instancias de conceptos en los nodos.

  Los métodos en el aspecto de comportamiento se implementan usando BaseLanguage, generalmente mejorado con extensiones:

  - El lenguaje de colecciones para una API de colección intuitiva.
  - El lenguaje de cierres para un uso fácil de las funciones literales.
  - El lenguaje smodel para manipular el código (estructura).

- **typesystem**: Un sistema de tipos es parte de una definición de lenguaje que asigna tipos a los nodos en los modelos escritos usando el lenguaje. El lenguaje del sistema de tipos también se usa para verificar ciertas restricciones en los nodos y sus tipos. La información sobre los tipos de nodos es útil para:

  - Encontrar errores de tipo.
  - Comprobar las condiciones de los tipos de nodos durante la generación para aplicar solo las reglas de generador adecuadas.
  - Proporcionar la información necesaria para determinadas refactorizaciones (por ejemplo, para la refactorización de la "variable de extracción").

- **generator**: Este elemento especifica la traducción de construcciones codificadas a partir del idioma de entrada a otras construcciones cofidicadas en el idioma de salida. En general el idioma de salida suele

ser baseLanguage (que es el que se encuentra integrado en MPS), que comparte con java casi el mismo conjunto de construcciones. A diferencia de cualquier otro aspecto del lenguaje, el aspecto generador no es un modelo único. La especificación del generador puede comprender muchos modelos de generador , así como modelos de utilidad. Como propiedades fundamentales del generador tenemos:

- Depende de generadores : Esto permite que el generador dependiente haga referencias a plantillas definidas en otro generador
- Restricciones de mapeo: Se pueden especificar relaciones de prioridad entre las reglas de mapeo. Si tal relación involucra otras reglas de generador, entonces también se requiere declarar una dependencia en ese generador.

3.4. Diseño de prototipo

Para realizar el diseño del prototipo se utilizó una hoja de cálculo disponible en el área de agricultura del Cabildo de Tenerife [3]. Esta contiene datos sobre diferentes cultivos, ya sea el tiempo de espera hasta la recogida, la densidad, la producción, los días hasta su recolección, etc. Toda esta información ha sido de mucha ayuda para definir de forma más clara las características que tiene un producto. Una vez que todo esto estaba claro, se continuó con los esbozos del diseño del prototipo. Partiendo que ya se tenía el producto definido, se crearon conjuntos para poder agruparlos. Así se conseguía que se pudieran crear diferentes grupos. Ya con esto, se pasó a la parte de planificación, que se formaba de un grupo de conjuntos, de tal forma que se puedan escoger que conjuntos se quiere planificar, además de la superficie que se quiera ocupar. Por último, tenemos el elemento fundamental que es el productor que es capaz de crear las planificaciones. Todos estos diseños fueron sufriendo cambio hasta llegar al que se tiene en la Figura 3.1

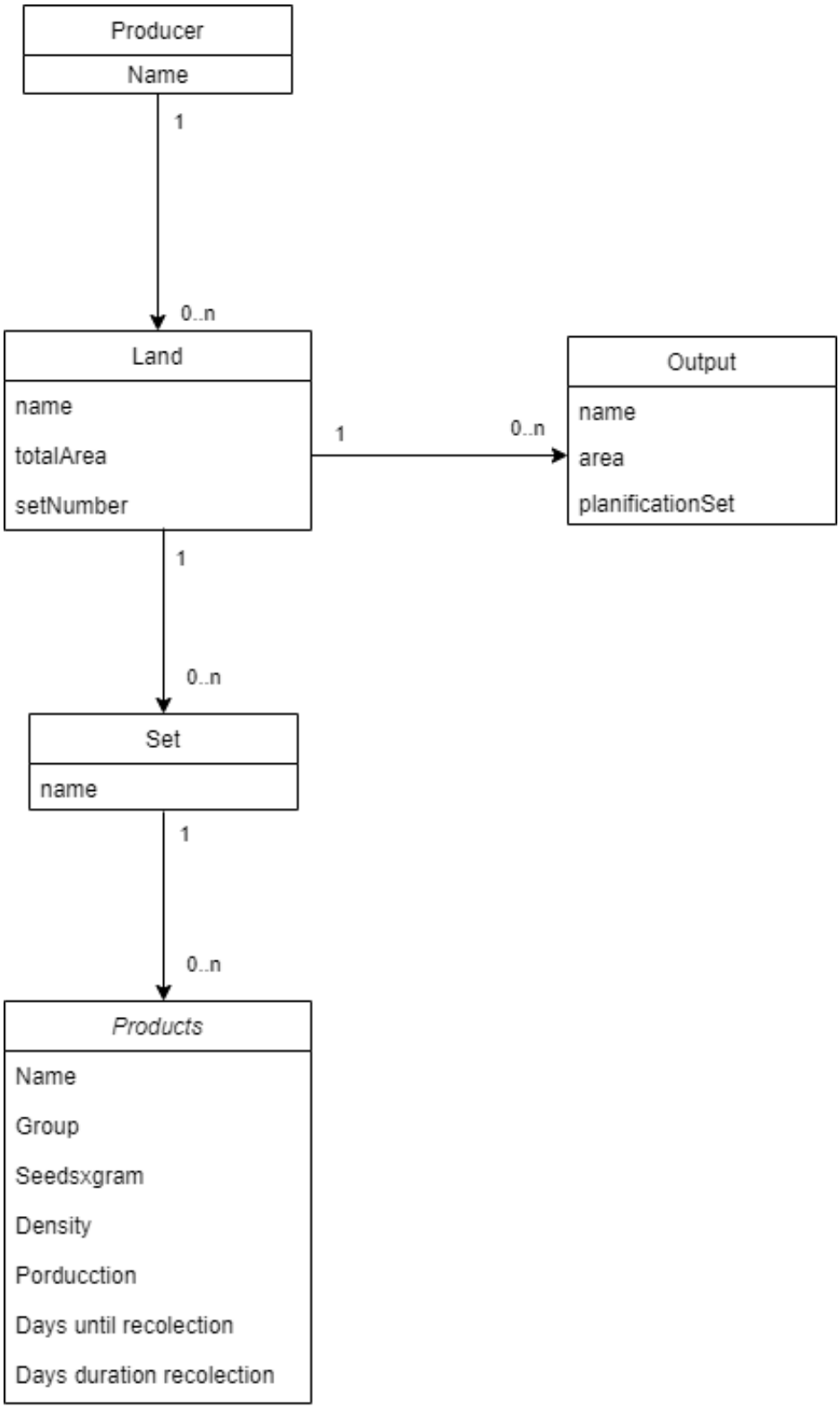

Figura 3.1: Diagrama UML

Capítulo 4

Prototipo

En este capítulo se presenta como se ha desarrollado e implementado este proyecto.

4.1. Presentación

Para comenzar con el desarrollo del sistema, fue necesario partir de una idea o un diseño del mismo. Mencionar que conforme se iba desarrollando se establecieron diferentes propuestas de diseño hasta llegar a la que se ha implementado al final. Todo lo que se ha realizado en este proyecto se encuentra en el siguiente proyecto de Github: `https://github.com/ULL-ESIT-GRADOII-TFG/tfg-carlos-diaz-calzadilla-software`. En la Figura 4.1 se puede ver el repositorio de Github.

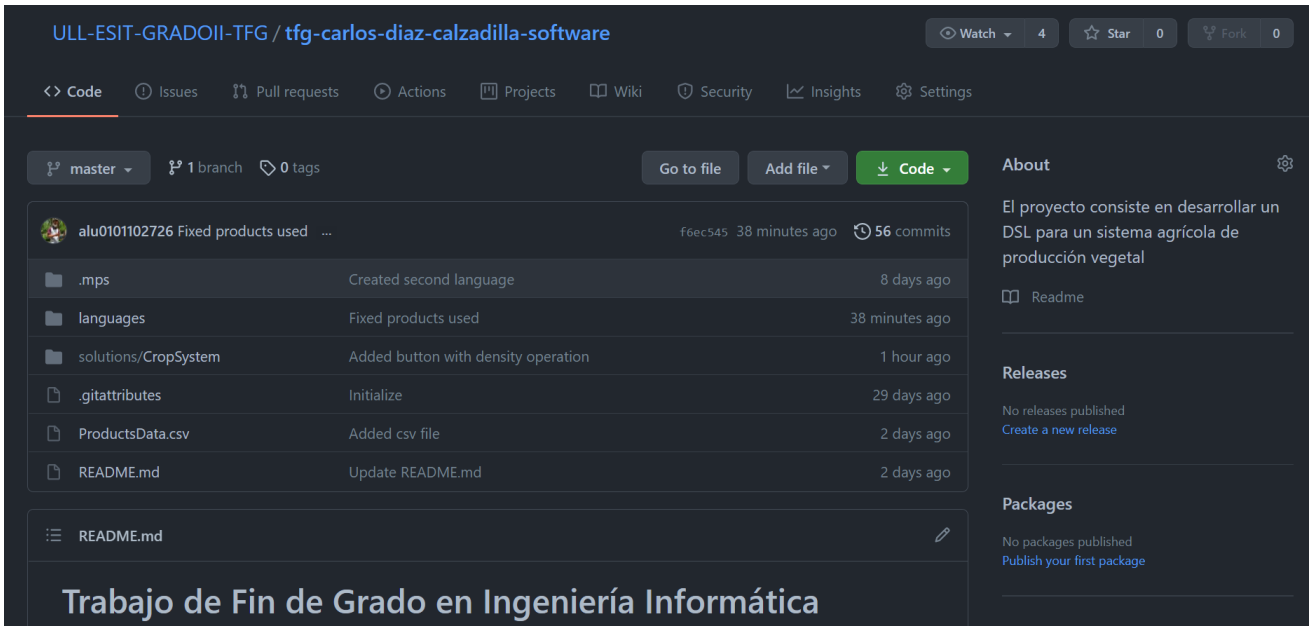

Figura 4.1: Captura del repositorio en Github

4.2. Arquitectura del Software

Una vez que se ha presentado el sistema, ahora se va a comentar como se ha desarrollado el mismo y que es lo que se ha hecho. Para llevar a cabo la implementación esto se realizó en 4 etapas diferentes:

Para comenzar con el desarrollo del sistema, fue necesario partir de una idea o un diseño del mismo. Mencionar que conforme se iba desarrollando se establecieron diferentes propuestas de diseño hasta llegar a la que se ha implementado al final. Importante destacar que se creó un repositorio en Github al inicio del proyecto, para así tenerlo documentado conforme se iba desarrollando. A continuación se explicará de cómo se implementó el sistema y cuales son las principales funcionalidades que dispone el mismo.

Para llevar a cabo la tarea del desarrollo lo primero que se hizo fue crear un nuevo proyecto en MPS y darle nombre al nuevo lenguaje que se iba a implementar. Además de ello se creó una Solución, que es un módulo que va a tener la instancia de nuestro nodo raíz para ir probando todos los cambios que se fueran realizando. Una vez que todo esto ya estaba creado se pasó a desarrollar la estructura.

Como ya se había mencionado antes, uno de los elementos fundamentales dentro de MPS son los conceptos. Estos nos van a definir las características de cada elemento, así como las dependencias que hay entre todos ellos. Por eso mismo, esta fase era de vital importancia, ya que con los elementos bien definidos y sus dependencias ya la base de la estructura estaría desarrollada. Con ello se empezaron a crear los primeros conceptos, que fueron los productos y los terrenos. Para los conceptos se recogieron algunas de las características del Excel del agrocabildo de Tenerife como son: nombre, semillas por gramo, densidad, producción, días hasta la recolección y días durante la recolección. Debido a que la mayor parte de los valores se encontraban como flotantes y MPS, no dispone de este tipo de dato, se sugirió tratar como una cadena de texto y ya luego, cuando se fueran a hacer cálculos pues hacer su pertinente transformación. Por otro lado tenemos el terreno, que se formaba de un conjunto de productos (desde 0 hasta n), además de sus características como el nombre, área total (para la planificación) y el número de cultivos. Mencionar que algunas propiedades de las tierras se tuvieron que crear en un nodo externo para que, a la hora de realizar la generación de código, fuera de una forma más clara y sencilla.

Ya con esto desarrollado, se vio la necesidad de crear otro concepto. Debido a que si se iba a hacer una planificación de cultivos, era necesario que estos se pudieran distinguir entre ellos, más que nada para poder especificar qué productos se querían plantar en una zona y cuáles en otra. Por ello surge el concepto de conjunto, que va a tener un nombre y un conjunto de productos. Con esto, ahora era más fácil agrupar los productos para su planificación. Por último, faltaba un elemento general que pudiera tener las tierras. Aquí aparece el productor, el cual podía tener de 0 a n tierras, teniendo cada una un conjunto de productos.

Con la estructura bien definida el siguiente paso fue crear el editor, es decir, crear la forma en la que se querían ver dichos conceptos para recoger la entrada de los datos. Para algunos conceptos, conforme estos se iban definiendo, también se iban creando los pertinentes editores.

Los editores en MPS aportan una abismal diferencia para visualizar los conceptos. Esto se debe a que gracias a la cantidad de plugins que hay se pueden utilizar muchas formas distintas desde funciones, diagramas de flujo hasta tablas. Hablemos del caso más importante. En este tenemos a los productos. Se buscaba que cuando el usuario fuera a introducir los datos de entrada, tuviera una forma que no fuera muy compleja debido a su sintaxis y que, además fuera sencilla de utilizar. Por ello se empleó el uso de tablas. Este tipo de dato es muy común, sobre todo en programas como excel y además es bastante intuitivo, por lo que ayuda a que el usuario no tenga que dedicar parte de su tiempo a entender la sintaxis y se pueda centrar en rellenar la tabla. Gracias a uno de los plugins de MPS [9] se pudo abarcar esta idea.

La tabla productos era muy sencilla, constaba de unos campos de título para explicar al usuario que debía añadir en cada celda y luego, las celdas para introducir dichos valores. Con esto, ahora cada vez que se fuera a introducir un producto, se obtendría una tabla. En la Figura 4.2 se puede observar un ejemplo de esta visualización.

Además de los productos se tuvo que crear un editor para los demás elementos. Mencionar que si no se desarrolla un editor para un concepto, entonces MPS pone algunos valores por defecto. En los otros casos se hizo una visualización más sencilla que constaba de mencionar algunas de las propiedades, para que el usuario añadiera los valores y luego añadir los hijos. Con esto último me refiero a que si en la dependencia tenemos definido que un set se forma de un conjunto de productos, entonces en el

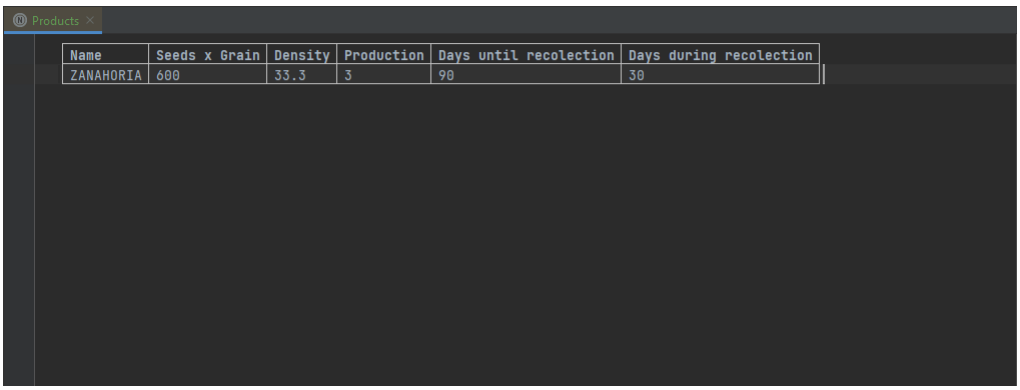

Figura 4.2: Editor con formato tabla para productos

editor, hay que decirle que añada ese conjunto, para que así dependiendo de la cantidad definida dentro del conjunto pues se vayan añadiendo donde se quiera. Antes de acabar de comentar la edicióna, en la Figura 4.3 se puede apreciar un ejemplo de visualización de los conceptos definidos.

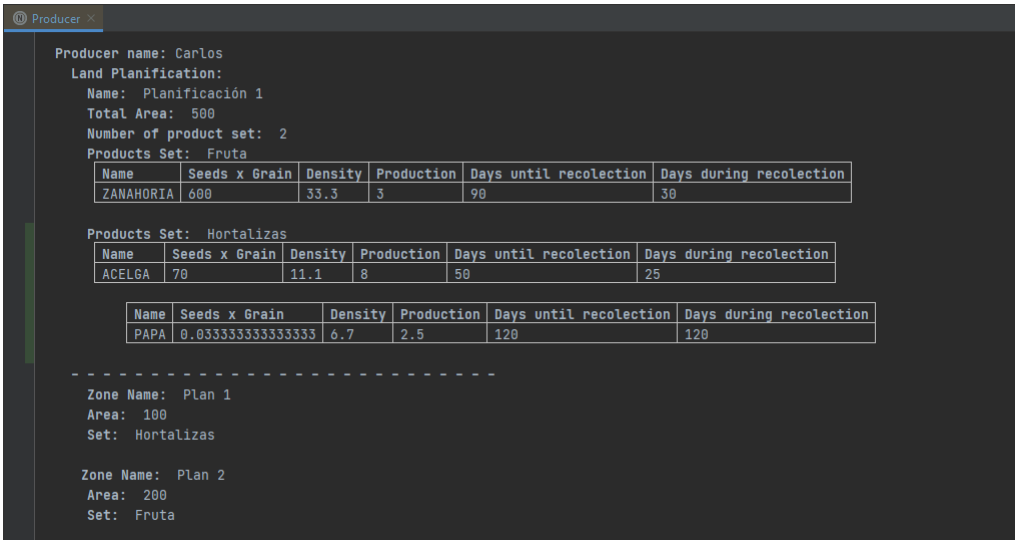

Figura 4.3: Ejemplo Productor (MPS)

Una vez que ya tenemos los aspectos básicos del sistema, que nos van a definir tanto la estructura de los datos como la visualización de los mismos, se pasó a crear la salida. Con esto me refiero a crear el generador, es decir, el módulo que va a permitir pasar de nuestro lenguaje al de Java (en nuestro caso). Primero de todo se tuvo que definir las rutas de mapeo. Esto es importante debido hay que decirle al programa que cuando se cree, en la solución, un módulo de productor, quiero que me genere el código en Java mediante la clase que se va a implementar. Ya con esta ruta de mapeo

definida lo que falta es crear ese elemento para realizar la conversión a partir de una clase en Java.

Para este paso se tuvo que crear una clase que se extendiera de JFrame, dado que esta era la librería de UI que se iba a utilizar. Para poder comprender el desarrollo del código comentar que en MPS las clases creadas en Java utilizan un sistema de macros. Con este consiguen que se puedan crear bucles dependiendo del número de elementos de un conjunto, que se creen variables con nombres o valores de alguno de los conceptos introducidos,...

Ya con esto comentado vamos a dar paso a explicar las partes más importantes. Para esta aplicación se utilizó un FlowLayout para poder ordenar los componentes en un flujo direccional. Una vez definido el tipo de layout a utilizar se pasó a la parte más importante de este conversor. Esta consistía en convertir las tablas que se habían añadido en la entrada, incluyendo los valores introducidos a una en JFRAME. Antes que nada, mencionar que todo el código que se va a explicar se va a iterar para cada conjunto del set. Por eso mismo fue necesario añadir una macro $LOOP$ para iterar en ese conjunto.Se empezó creando un nuevo panel, que iba a representar cada conjunto de productos. Con esto ya creado se pasó a definir la tabla, esto se hizo mediante una matriz de objetos, la cual contenía un array de objetos que representaban los valores introducidos en la tabla y luego un array de string que representa la cabecera de la tabla. Pero claro, ahora iba a ser necesario que todo esto se repitiera para cada entrada dentro de los productos. Para poder solventar esto, se creó una macro $LOOP$ que itere sobre los objetos del producto, creando una variable de array de objetos por cada producto y consiguiendo así que se crearan tantos valores en la tabla como valores tuviera la entrada del programa. Esto podría tener un fallo y es que si se tuvieran muchos productos distintos, entonces la variable se estaría sobrescribiendo muchas veces, Para poder resolver este problema, solo fue necesario añadir una Macro de propiedad para que cambiara el nombre de la variable al nombre que se le había dado al set. Por último mencionar que se creó un borde para el panel para hacerlo más bonito. A continuación se muestra el código respectivo:

```
$LOOP${
  JPanel panel = new JPanel();
  String[] columnNames = {"name", "seeds_x_gram", "density",
  "production", "daysUntilRecolection", "daysDuringRecolection"};
  $LOOP$Object[] $data = {"$Name", "$seeds_x_gram", "$density",
  "$production", "$daysUntilRecolection", "$daysDuringRecolection"};
```

```
  Object[][] table = {$LOOP$data};
  panel.setBorder(BorderFactory.createTitledBorder(BorderFactory.
    createEtchedBorder(), "$Tittle", TitledBorder.CENTER,
    TitledBorder.TOP));
  panel.add(new JScrollPane(new JTable(table, columnNames)));
  add(panel);
}
```

Además de esta parte, se tuvo que generar los demás elementos como el nombre de la zona, el área y los cultivos que se querían planificar. Para estos se creó un nuevo panel, donde para cada planificación de tierras, es decir, crear un macro $LOOP$ para iterar sobre este elemento. Esto se hizo igual para el área y los cultivos, añadiendo las respectivas JLabels, lo único es que, para escoger los cultivos se realizaron JComboBox para así tener un menú desplegable donde las opciones son el nombre de los sets. En la Figura 4.4 tenemos un ejemplo de salida con la entrada que se definió antes en la Figura 4.3.

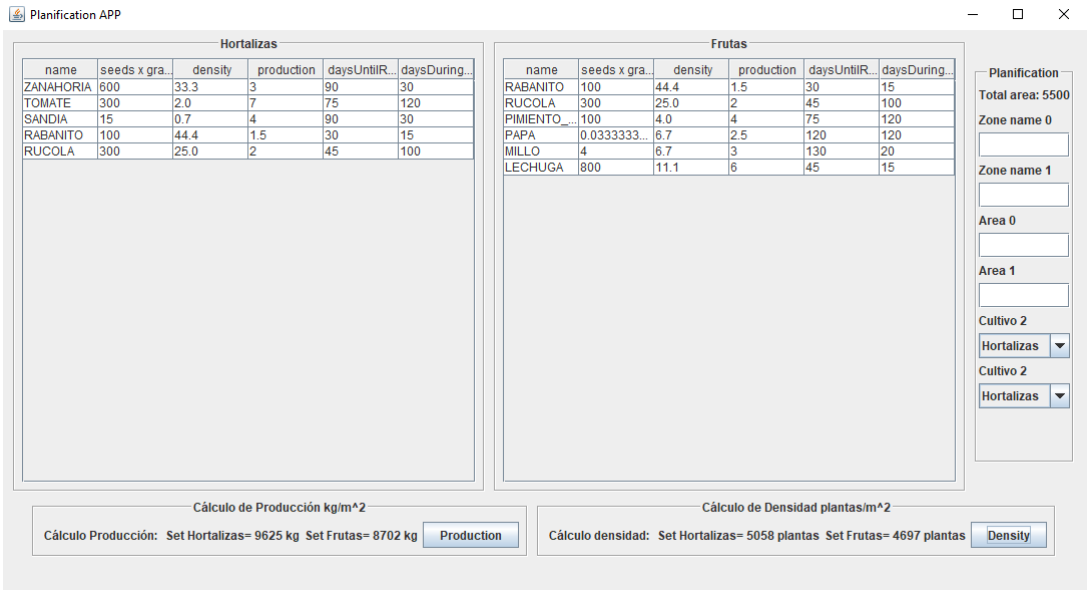

Figura 4.4: Ejemplo Productor (APP generada en Java)

Además de todo lo ya mencionado, como se puede ver en la Figura 4.4 se han añadido dos operaciones básicas. Primero tenemos un cálculo de la producción en kilogramos por metro cuadrado, en la que se hace una distribución uniforme de la cantidad de superficie total. Para este ejemplo la superficie total ha sido de 5500 metros cuadrados, entonces este valor se divide dependiendo del número de conjuntos que tengamos, que como en este caso son dos pues a cada conjunto le pertenecen 2750 metros cuadrados. Ya con estos valores, para cada producto se le asigna una cantidad equitativa, en este caso tenemos en el conjunto de hortalizas 5

productos diferentes, por lo que este valor (2750) lo dividimos entre 5 productos, e igual para los demás conjuntos dependiendo de los productos que contengan. Para acabar se multiplica cada cantidad de metros cuadrados asignada a los productos por el valor de la tabla de producción que, una vez todos sumados, se conseguirá la producción en kilogramos de todo el conjunto. A su vez se realizó una operación similar para la densidad, obteniendo el número de plantas total para cada conjunto.

Ya con todo definido, el siguiente paso era implementar las funcionalidades avanzadas. Dado que la introducción de los datos de entrada no era muy cómoda, debido a que el usuario necesitaba añadir cada producto además de sus correspondientes valores para las características, entonces se sugirió implementar un sistema de autocompletado que, una vez que se añadiera el nombre del producto, mediante la combinación ctrl + espacio poder autocompletar el nombre según los disponibles en la base de datos y añadir sus valores automáticamente. Para realizar esta tarea, se escogió a partir del excel del agrocabildo y se exportaron los datos en un formato csv. Para que estos datos sean correctamente leídos por el sistema, los nombres de los productos deben no tener espacios, se pueden sustituir por barra baja y los elementos de separación son puntos y coma dado que poner una como como separación daba problemas ya que había números en flotante que usaban ya la coma. Una vez que ya se tenían estos datos en un csv, se importaron al proyecto. Con esto, una vez que se implementara el usuario podría disponer de un sistema de autocompletado para los elementos que se encuentren en dicho csv.

Para llevar a cabo dicha implementación, se creó un modelo de transformación, que va a permitir, que los valores de algunos elementos, como por ejemplo las propiedades de los productos se modifiquen respecto a dicho modelo. En este lo que se logró fue que el sistema hiciera una búsqueda a tiempo real de lo que el usuario estaba escribiendo y ver si encontraba algún nombre de algún producto que contenga esas letras. En el caso de la implementación realizada, se autocompleta con la palabra que primero encuentre (están ordenados de forma alfabética), es decir,si ponemos una A, tendremos la opción de calabaza, zanahoria, acelga,... entonces, en este caso nos mostrará acelga dado que es el primer elemento de la tabla. Una vez que se haya encontrado un nombre que cace con lo introducido se va a ejecutar un código. En nuestro caso, dado que queremos que el usuario pueda autocompletar todas las características con solo añadir el nombre entonces se va a comprobar si el nombre del producto coincide con el

nuestro, en caso de que no, se pasa al siguiente. Pero si coincide, entonces vamos a asignar a las diferentes propiedades del nodo sus características recogidas en el csv (descartando aquellas que no nos interesan). Para poder entender mejor esto se muestra en la Figura 4.5 como funciona el autocompletado.

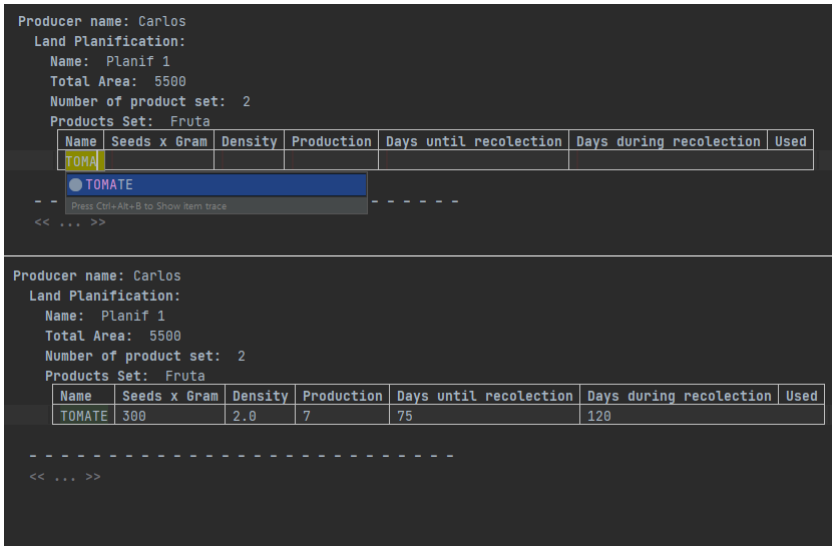

Figura 4.5: Funcionalidad avanzada: Autocompletado

Con esta funcionalidad se conseguía que el usuario no necesitara añadir todos los productos a mano y además, facilitaba que no tuviera que recordar las características de los mismos, dando mayor versatilidad y eficiencia para introducir la entrada. Sin embargo, esto sería una tarea costosa en el caso de que se quisiera añadir muchos productos, ya que sería necesario introducirlos uno a uno. Para poder resolver este problema se creó un segundo DSL. Para empezar se crearon los conceptos y editores de forma similar al caso anterior. Hubo una variante y esta fue en el editor del producto. La idea inicial era que este segundo DSL permitiera que al crear un conjunto de productos, se añadieran todos los productos que se encontraban en la base de datos. Con esto el usuario solo necesitaría marcar aquellos que quisiera. Para poder detectar esta marca, en el editor del producto se añadió un checkbox.

Ya con la base definida, ahora iba a ser necesario que se hiciera un correcto autocompletado. A partir de un menú de transformación se logró realizar esto. El autocompletado consistía en añadir la frase .Add all", con el uso de ctrl + espacio y cuando se confirmara, añadiera todos los valores de las tablas. Esta fue una tarea un poco más compleja que la anterior pero lo que se hizo fue recorrer cada una de las filas hasta el final, y por cada

fila que se encontraba crear una nueva instancia de producto, rellenar este nodo con los pertinentes valores y luego añadirla después de la actual. Ya con esto se obtiene lo que se puede apreciar en la Figura 4.6.

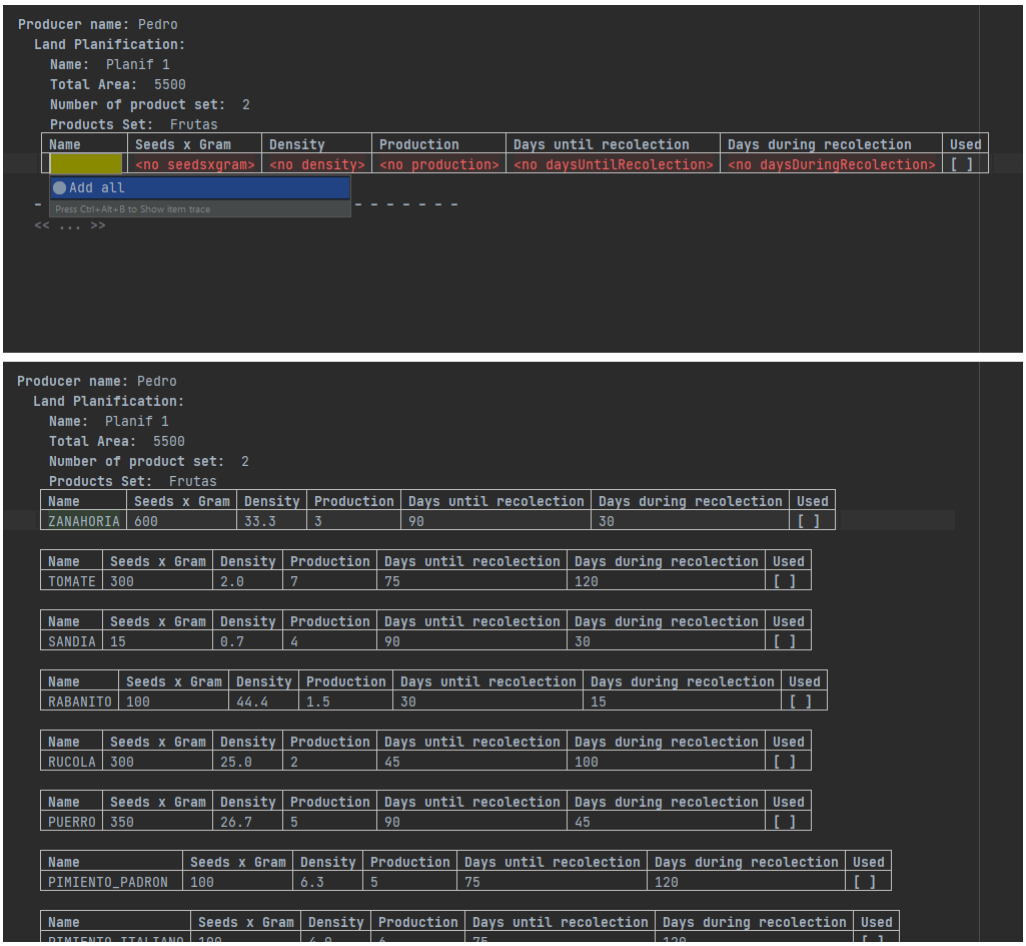

Figura 4.6: Funcionalidad avanzada: Autocompletado de todos los productos

Por último, fue necesario desarrollar otro generador. Como este es muy similar al anterior que habíamos creado se va a comentar los nuevos elementos que añade. A diferencia del anterior, para que un producto se ecuentre en la salida debe tener marcado la casilla de usado. Como este es un valor booleano, pues solo habría que comprobar si ese valor está a true, entonces se añade a la salida. Para hacer esto, primero se intentó utilizar un condicional if, pero esto daba problemas dado que el array de objetos (la fila del producto) se encontraba declarado dentro y se usaba fuera. Para poder solventar este error solo fue necesario utilizar una macro IF, que permitía que solo se ejecutara el interior si el valor de used estaba a true. Además de estas macros también se utilizó la de LOOP para poder iterar en los conjuntos disponibles y en los productos. Todo esto se consiguió a partir del siguiente código:

```
$LOOP${
  JPanel panel = new JPanel();
  Boolean val;
  String[] columnNames = {"name", "seeds_x_gram", "density",
    "production", "daysUntilRecolection", "daysDuringRecolection"};
  $LOOP$$IF$Object[] $data;

  $LOOP$$IF$data = new Object[]{"$Name", "$seeds_x_gram", "$density",
    "$production", "$daysUntilRecolection", "$daysDuringRecolection"};

  Object[][] table = {$LOOP$$IF$data};
  panel.setBorder(BorderFactory.createTitledBorder(BorderFactory.
    createEtchedBorder(), "$Tittle", TitledBorder.CENTER,
    TitledBorder.TOP));
  panel.add(new JScrollPane(new JTable(table, columnNames)));
  add(panel);

}
```

Una vez que ya se comprende toda la implementación se va a pasar a comentar la fase de validación y pruebas. Para empezar no se realizaron pruebas, es decir, no se implementaron tests, ya que los que ofrece MPS se encuentran más centrados en realizar comprobaciones de tipos de datos, valores, resultados de operaciones,.. pero, en este caso solo se realizó el diseño e implementación de la estructura. Sin embargo, conforme se iba desarrollando el código y las diferentes funcionalidades, se creaba un productor y se comprobaba que todas estás funcionaban correctamente antes de subirlas al repositorio. Por otro lado, para la validación se utilizaron las operaciones antes mencionadas. Con estas dos que fueron implementadas se pudo obtener un resultado que permitía conocer si el acceso a los datos era el correcto, si los datos se encontraban bien almacenados e incluso si el valor que salía por pantalla era el correcto. Gracias a esto se pudo comprobar que todos estos aspectos estaban correctamente implementados.

Capítulo 5

Conclusiones y líneas futuras

Durante este capítulo, se exponen las reflexiones que han aportado la elaboración del proyecto y las posibles características o servicios que podrían añadirse en un futuro.

5.1. Conclusiones

La elaboración de este proyecto me ha dado a conocer que los DSL son muy potentes, Además de ser sencillos de implementar (dependiendo un poco del lenguaje que se quiera conseguir). Por otro lado, la herramienta que se ha utilizado para crear los DSL, MPS, ha dado mucha variabilidad y facilidad para poder añadir una mejor interfaz para el usuario o incluso, poder generar luego código en Java y así crear una salida más atractiva. Personalmente he aprendido mucho en este TFG en todos los aspectos, tanto la forma de trabajar como los conocimientos que he adquirido para poder implementar el sistema.

5.2. Líneas futuras

En un futuro, dado que nos hemos centrado en la estructura de almacenamiento, se podría:

- Mejorar la salida de tal forma que resuelva el problema de planificación
- Hacer que vuelque los datos en algunas gráficas o mostrarlos directamente en la propia interfaz.
- Hacer mejoras internas para añadir otro tipo de planificaciones como la planificación de ventas y luego esto mostrarlo en la salida del programa.

- Mejorar la gestión de los datos desde el fichero en formato CSV (*Comma-Separated Values*).
- Mejorar la calidad de la salida.

Capítulo 6

Summary and Conclusions

6.1. Conclusions

The development of this project has made me aware that DSLs are very powerful, as well as being simple to implement (depending on the language you want to achieve). On the other hand, the tool that has been used to create the DSL, MPS, has given a lot of variability and ease to be able to add a better interface for the user, or even to later generate code in Java and thus create a more attractive output. Personally, I have learned a lot in this TFG in all aspects, both the way of working and the knowledge that I have acquired to be able to implement this system.

Capítulo 7

Presupuesto

En este capítulo, se encuentra un posible presupuesto para la realización de este proyecto.

Tarea	Horas	Coste
Revisión bibliográfica	30h	10€/h
Diseño de la arquitectura software	14h	8€/h
Diseño de prototipo y selección de las herramientas a utilizar	8h	8€/h
Implementación inicial	50h	15€/h
Implementación avanzada	130h	16€/h
Despliegue	5h	10€/h
Difusión de la memoria	30h	10€/h
TOTAL	267h	4296€

Tabla 7.1: Presupuesto estimado del proyecto

Finalmente, en la Tabla 7.1, se realiza una estimación del presupuesto que tendría la realización de este proyecto. Para ello, ha sido necesario calcular el número de horas empleadas en cada tarea y el coste por hora.

Bibliografía

[1] Ignacio Albornoz. Software para el sector agropecuario. *Littec, Buenos Aires*, 2006.

[2] Carlos Díaz Calzadilla. https://github.com/ULL-ESIT-GRADOII-TFG/tfg-carlos-diaz-calzadilla-software.

[3] Cabildo de Tenerife. http://www.agrocabildo.org/publicaciones_detalle.asp?id=662. Accedido en abril de 2021.

[4] Pascal Degenne, D Lo Seen, Didier Parigot, Rémi Forax, Annelise Tran, A Ait Lahcen, Olivier Curé, and Robert Jeansoulin. Design of a domain specific language for modelling processes in landscapes. *Ecological Modelling*, 220(24):3527–3535, 2009.

[5] Uqbar Foundation. https://sites.google.com/site/teoriadelenguajesformaless/1-7-fases-de-un-compilador. Accedido en junio de 2020.

[6] GitHub. https://github.com/. Accedido en julio de 2020.

[7] Desirée Groeneveld, Bedir Tekinerdogan, Vahid Garousi, and Cagatay Catal. A domain-specific language framework for farm management information systems in precision agriculture. *Precision Agriculture*, pages 1–40, 2020.

[8] Marylin Mamani, Marco Villalobos, and Raúl Herrera. Sistema web de bajo costo para monitorear y controlar un invernadero agrícola. *Ingeniare. Revista chilena de ingeniería*, 25(4):599–618, 2017.

[9] Slisson. https://github.com/slisson/mps-tables. Accedido en abril de 2021.

[10] Wikipedia. https://es.wikipedia.org/wiki/Compilador#Etapas_del_proceso. Accedido en junio de 2020.