Capítulo 1

Antecedentes

Este capítulo trata de describir a grandes rasgos las técnicas, tecnologías y herramientas utilizadas para el desarrollo del presente Proyecto Fin de Carrera. En primer lugar, se introducirá el caso de estudio que se utilizará de forma recurrente a lo largo del proyecto, que es una línea de productos software para software de control para hogares inteligentes. Para ello se describen en primer lugar diversos conceptos relacionados el dominio del proyecto, como son las líneas de productos software y los árboles de características. A continuación, se describen dos principales herramientas de Ingeniería de Lenguajes Dirigida por Modelos utilizadas: *Ecore* y *EMFText*. Por último, se describe brevemente la arquitectura de plugins de Eclipse, dado que nuestro proyecto debía integrarse en dicho entorno.

Índice

1.1. Caso de Estudio: Software para Hogares Inteligentes

El objetivo último del presente proyecto es la construcción de una línea de productos software sobre la plataforma .NET para hogares automatizados y/o inteligentes.

El objetivo de estos hogares es aumentar la comodidad y seguridad de sus habitantes, así como hacer un uso más eficiente de la energía consumida. Los ejemplos más comunes de tareas automatizadas dentro de un hogar inteligente son el control de las luces, ventanas, puertas, persianas, aparatos

de frío/calor, así como otros dispositivos, que forman parte de un hogar. Un hogar inteligente también busca incrementar la seguridad de sus habitantes mediante sistemas automatizados de vigilancia y alerta de potenciales situaciones de riesgo. Por ejemplo, el sistema debería encargarse de detección de humos o de la existencia de ventanas abiertas cuando se abandona el hogar.

El funcionamiento de un hogar inteligente se basa en el siguiente esquema: (1) el sistema lee datos o recibe datos de una serie de sensores; (2) se procesan dichos datos; y (3) se activan los actuadores para realizar las acciones que correspondan en función de los datos recibidos de los sensores.

Todos los sensores y actuadores se comunican a través de un dispositivo especial denominado puerta de enlace (*Gateway*, en inglés). Dicho dispositivo se encarga de coordinar de forma adecuada los diferentes dispositivos existentes en el hogar, de acuerdo a los parámetros y preferencias especificados por los habitantes del mismo. Los habitantes del hogar se comunicarán con la puerta de enlace a través de una interfaz gráfica. Este proyecto tiene como objetivo el desarrollo de un hogar inteligente como una línea de productos software, con un número variable de plantas y habitaciones. El número de habitaciones por planta es también variable. La línea de productos deberá ofrecer varios servicios, que podrán ser opcionalmente incluidos en la instalación del software para un un hogar determinado. Dichos servicios se clasifican en funciones básicas y complejas, las cuales describimos a continuación.

Funciones básicas

1. *Control automático de luces:* Los habitantes del hogar deben ser capaces de encender, apagar y ajustar la intensidad de las diferentes luces de la casa. El número de luces por habitación es variable. El ajuste debe realizarse especificando un valor de intensidad.

2. *Control automático de ventanas:* Los residentes tienen que ser capaces de controlar las ventanas automáticamente. De tal modo que puedan indicar la apertura de una ventana desde las interfaces de usuario disponibles.

3. *Control automático de persianas:* Los habitantes podrán subir y bajar las persianas de las ventanas de manera automática.

4. *Control automático de temperatura:* El usuario será capaz de ajustar la temperatura de la casa. La temperatura se medirá siempre en grados celsius.

Funciones complejas

1. *Control inteligente de energía:* Esta funcionalidad trata de coordinar el uso de ventanas y aparatos de frío/calor para regular la temperatura interna de la casa de manera que se haga un uso más eficiente de la energía. Por ejemplo, si se recibe la orden de calentar la casa, a la vez que se activan los radiadores se cerrarán las ventanas para evitar las pérdidas de calor.

2. *Presencia simulada:* Para evitar posibles robos, cuando los habitantes abandonen la casa por un periodo largo de tiempo, se deberá poder simular la presencia de personas en las casas. Hay dos opciones de simulación (no exclusivas):

   a) *Simulación de las luces:* Las luces se deberán apagar y encender para simular la presencia de habitantes en la casa.

   b) *Simulación de persianas:* Las persianas se deberán subir y bajar automática para simular la presencia de individuos dentro de la casa.

Todas estas funciones son opcionales. Las personas interesadas en adquirir el sistema podrán incluir en una instalación concreta de este software el número de funciones que ellos deseen. La siguiente sección describe la planificación general realizada para desarrollar este caso de estudio.

1.2. Líneas de producto software

El objetivo de una *línea de productos software* [**?**, **?**] es crear una infraestructura adecuada a partir de la cual se puedan derivar, tan automáticamente como sea posible, productos concretos pertenecientes a una familia de productos software. Una familia de productos software es un conjunto de aplicaciones software similares, que por tanto comparten una serie de características comunes, pero que también presentan variaciones entre ellos.

Un ejemplo clásico de familia de productos software es el software que se encuentra instalado por defecto en un teléfono móvil. Dicho software contiene una serie de facilidades comunes, tales como agenda, recepción de llamadas, envío de mensajes de texto, etc. No obstante, dependiendo de las capacidades y la gama del producto, éste puede presentar diversas funcionalidades opcionales, tales como envío de correos electrónicos, posibilidad de conectarse a Internet mediante red inalámbrica, radio, etc.

La idea de una línea de productos software es proporcionar una forma automatizada y sistemática de construir productos concretos dentro de una familia de productos software mediante la simple especificación de qué características deseamos incluir dentro de dicho producto. Esto representa una alternativa al enfoque tradicional de desarrollo software, el cual se basaba

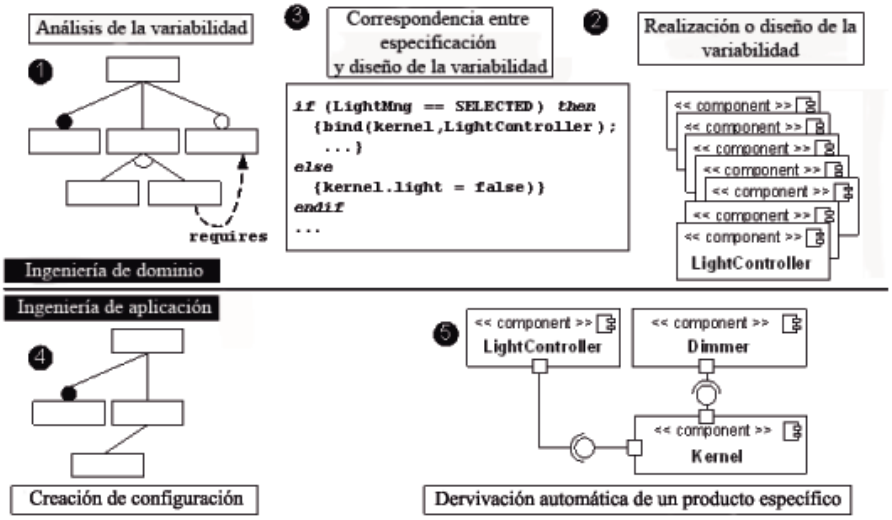

Figura 1.1: Proceso de desarrollo de una línea de productos software

simplemente en seleccionar el producto más parecido dentro de la familia al que queremos construir y adaptarlo manualmente.

El proceso de creación de líneas de producto software conlleva dos fases: *ingeniería del dominio* (en inglés, *Domain Engineering*) e *ingeniería de aplicación* (en inglés, *Application Engineering*) (laFfigura 1.1 ilustra el proceso para ambas fases). La *ingeniería del dominio* tiene como objetivo la creación de la infraestructura o arquitectura de la línea de productos, la cual permitirá la rápida, o incluso automática, construcción de sistemas software específicos pertenecientes a la familia de productos. La *ingeniería de aplicación* utiliza la infraestructura creada anteriormente para crear aplicaciones específicas adaptadas a las necesidades de cada usuario en concreto.

En la fase de ingeniería del dominio, el primer paso a realizar sería un análisis de qué características de la familia de productos son variables y por qué y cómo son variables. Esta parte es la que se conoce como *análisis o especificación de la variabilidad* (Figura 1.1, punto 1). A continuación, se ha de diseñar una arquitectura o marco de trabajo para la familia de productos software que permita soportar dichas variaciones. Esta actividad se conoce como *realización o diseño de la variabilidad* (Figura 1.1, punto 2). El siguiente paso es establecer una serie de reglas que especifiquen como hay que instanciar la arquitectura previamente creada de acuerdo con las características seleccionadas por cada cliente. Esta fase es la que se conoce como *correspondencia entre especificación y diseño de la variabilidad* (Figura 1.1, punto 3).

En la fase de ingeniería de aplicación, se crearía una *configuración*, que no es más que una selección de características que un usuario desea incluir en su producto (Figura 1.1, punto 4). En el caso ideal, usando esta con-

Figura 1.2: Árbol de características simple para nuestro caso de estudio

figuración, deberíamos poder ejecutar las reglas de correspondencia entre especificación y diseño de la variabilidad para que la arquitectura creada en la fase de ingeniería del dominio se adaptase automáticamente generando un producto concreto específico acorde a las necesidades concretas del usuario (Figura 1.1, punto 5). En caso no ideal, dichas reglas de correspondencia deberán ejecutarse a mano, lo cual suele ser un proceso tedioso, largo, repetitivo y propenso a errores.

Este proyecto se centra en la primera etapa de este proceso de desarrollo, es decir en el análisis de la variabilidad de una familia de productos software mediante árboles de características. La siguiente sección proporciona una breve pero completa descripción acerca del funcionamiento de los árboles de características.

1.3. Árboles de características

Como se ha comentado en la sección anterior, una de las tarea claves para el éxito de una línea de productos software consiste en analizar la variabilidad existente en la familia de productos software que dicha línea de productos software pretende cubrir. Aquí es donde entran en juego los *Árboles de características* []. Una *característica* se define como "*un incremento en la funcionalidad del producto*", o más formalmente, "*una característica es una propiedad de un sistema que es relevante a algunos* stakeholders *y que es utilizada para capturar propiedades comunes o diferenciar entre sistemas de una misma familia*-[]. De este modo un producto queda representado por las características que posee.

Para poder capturar las divergencias y aspectos comunes entre los distintos productos de una misma familia, los árboles de características organizan de forma jerárquica el conjunto de características que posee una familia de productos. Cada característica se representa como un nodo ene la árbol de características. La raíz de dicho árbol es siempre el sistema o producto software cuya variabilidad estamos analizando. Cada característica se puede descomponer en varias subcaracterísticas, siendo estás últimas nodos *hijos* de la primera características, que actuaría como *padre*. Dependiendo de sis dichas subcaracterísticas son obligatorias, alternativas u opcionales, existen diversos tipos de relaciones padre-hijo.

La Figura 1.2 muestra com ejemplo un árbol de características que especifica la variabilidad inherente a nuestro caso de estudio, sin considerar que las plantas y habitaciones puedan configurarse de manera individual.

Una vez creado un árbol de características para una línea de productos software, podemos indicar las características que podemos incluir en un

Figura 1.3: Árbol de características completo para nuestro caso de estudio

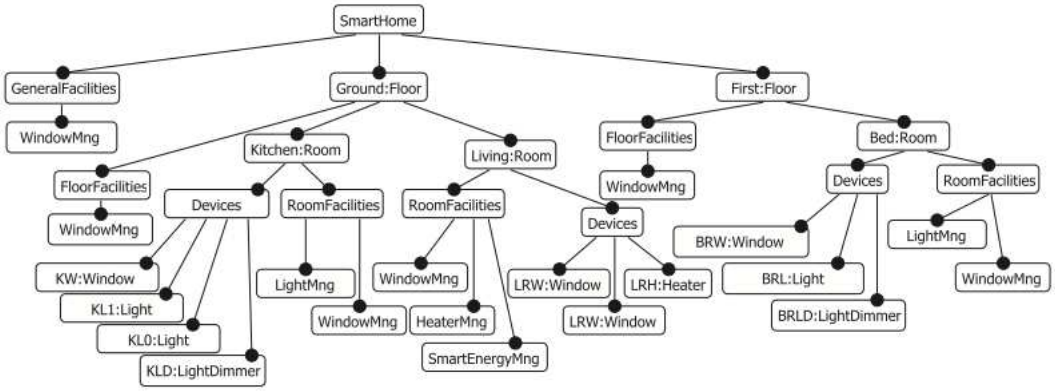

Figura 1.4: Posible modelo de configuración de una casa inteligente concreta

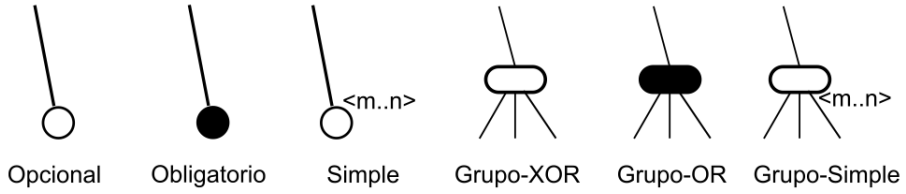

Figura 1.5: Tipos de relaciones entre características

producto software concreto mediante la creación de configuraciones. Una *configuración* no es más que una selección válida de características. La Figura **??** muestra en un ejemplo de configuración para el modelo de la Figura 1.2 donde se indica que el producto que deseamos construir debe incluir ≪TODO: COMPLETAR≫. Obviamente, dicho modelo debe satisfacer las restricciones externas declaradas.

Los árboles de características como los anteriormente expuestos no permiten modelar que pueda existir un número variable de ciertas características, como, en nuestro caso, de plantas y habitaciones, y que, además, cada instancia particular de una características pueda configurarse de forma distinta. Por ejemplo, podríamos decidir que el salón de la casa tenga control inteligente de temperatura, mientras que la cocina, que está sometida a mayores variaciones de temperatura, no contenga dicha características. Para solventar esta carencia, se introdujeron en los árboles de características el concepto de característica clonable.

La Figura 1.3 muestra el árbol de características para nuestro caso de estudio incluyendo *características clonables*.

A modo de resumen, la Figura 1.5 muestra las posibles relaciones que

pueden entre características, así como su representación gráfica. Dichas relaciones se describen a continuación:

Opcional La característica hija puede estar o no estar seleccionada

Obligatoria La característica siempre debe estar seleccionada.

Clonable La característica tendrá una cardinalidad $< m, n >$, siendo m y n números enteros que denotan el mínimo y el máximo respectivamente de características que podemos seleccionar.

Grupo-xor Sólo una de las características pertenecientes al grupo puede ser seleccionada.

Grupo-or Debemos seleccionar como mínimo una de las subcaracterísticas, pudiendo seleccionar más si lo deseamos.

Grupo con cardinalidad El número mínimo y máximo de características a seleccionar dentro del grupo vendrá determinado por su cardinalidad $< m, n >$.

1.4. *Eclipse Modelling Framework*

EMF o Eclipse Modeling Framework es un plug-in para eclipse que permite trabajar con la metodolía de Ingeniería Dirigida por Modelos. Tiene incorporado varios generadores de código que permiten, entre otra multitud de funciones, crear automáticamente la implementación de los modelos que creemos, así como su integración inmediata como plug-in con el entorno eclipse. Además, es capaz de integrarse con multitud de herramientas orientadas a tareas mucho más específicas dentro de la Ingeniería Dirigida por Modelos, de las cuales cabe destacar Ecore.

Ecore es la herramienta que permite la creación y edición de metamodelos, gracias a un entorno visual bastante atractivo que facilita enormemente el proceso. El fichero resultante de nuestra creación presentará automáticamente un formato estándar XMI, que es el utilizado para todo tipo de modelos y sus instancias. Ecore posee otro tipo de funcionalidades menos utilizadas que se engloban dentro del paquete Ecore Tools, enfocadas todas ellas al uso de los metamodelos y su validación.

Por último, EMF también incorpora una herramienta integrada con Ecore para la validación de las múltiples sintaxis concretas que podamos construir. EMF Validation Framework sirve, en otras palabras, como soporte en caso de que nuestro lenguaje pueda contener reglas adicionales que no puedan ser satisfechas únicamente con la descripción del metamodelo y la gramática. En el caso particular de nuestro editor para escificación y validación de restricciones hemos utilizado EMF Validation Framework para

comprobar si las características escritas por el usuario existen en el modelo importado, y también para determinar si tienen cardinalidad o no.

1.5. EMFText

EMFText es una herramienta específicamente diseñada para diseñar las gramáticas de los lenguajes que hayan sido diseñados previamente con un metamodelo de Ecore. Está especializado para la creación de Lenguajes Específicos de Dominio, aunque también se pueden crear lenguajes de propósito general.

Pero, como en casi todos los casos de este tipo de herramientas, su mayor virtud es la enorme cantidad de código autogenerado que produce, y que elimina al programador de tareas tediosas que además en muchos casos podrían resultar complicadas. Todo el código generado es completamente independiente de EMFText, es decir, podrá ser ejecutado en plataformas que no tengan la herramienta instalada.

Todo el código generado por EMFText está diseñado de tal modo que sea fácil de modificar en caso de que queramos poner en práctica algunas funcionalidades poco habituales. Se facilita mucho la labor a la hora de modificar estructuras como el postprocesador de nuestra gramática. Todas las gramáticas construidas tendrán que ser LL por defecto, a no ser que queramos modificar los parsers generados, posibilidad también disponible.

Otro tipo de funcionalidades implementadas, quizás no tan importantes pero también de gran utilidad, son el coloreado de código (por defecto o personalizable), función de completar código, generación del árbol parseado en el la vista de eclipse Outline o generación de código para crear un depurador para nuestro lenguaje.

EMFText permite la definición de gramáticas utilizando un lenguaje estándar para la definición de expresiones regulares, además de incorporar algunas particularidades propias que facilitan ciertas tareas. En la figura 1.6 se muestra una pequeña captura que contiene una porción de la gramática construida para nuestro editor de especificación y validación de restricciones.

1.6. Arquitectura de plugins de Eclipse

Un plug-in en Eclipse es un componente que provee un cierto tipo de servicio dentro del contexto del espacio de trabajo de eclipse, es decir, una herramienta que se puede integrar en el entorno Eclipse junto con sus otras funcionalidades. Dado que la herramienta Hydra fue diseñada como un plug-in para Eclipse, y nuestro editor es una parte de la misma, ha sido necesario aprender el manejo de algunas de las funcionalidades de la arquitectura de plug-ins de Eclipse.

```
START Model

OPTIONS {
    usePredefinedTokens="true";
}

TOKENS {
    DEFINE DIGIT $('0'..'9')+$;
    DEFINE DIRECCION $('A'..'Z'|'a'..'z'|'0'..'9'|'_'|'-'|'/'|'.')+$;
}

RULES {

Model::= "import" featureList[DIRECCION] ";" (constraints";")*;
Constraint::= operators | "(" operators ")";
BoolPriorityOperand1::= "(" boolPriorityOp1 ")" | boolPriorityOp1:BoolOperandChoices;
BoolPriorityOperand2::= "(" boolPriorityOp2 ")" | boolPriorityOp2;
NumPriorityOperand1::= numPriorityOp1:NumOperandChoices | "(" numPriorityOp1 ")";
NumPriorityOperand2::= numPriorityOp2 | "(" numPriorityOp2 ")";

    // Operaciones logicas
    And ::= binaryOp1 "and" binaryOp2;
    Or ::= binaryOp1 "or" binaryOp2;
    Xor ::= binaryOp1 "xor" binaryOp2;
    Implies ::= binaryOp1 "implies" binaryOp2;
    Neg ::= "!" unaryOp;
```

Figura 1.6: Trozo de la gramática de nuestro editor de especificación y validación de restricciones

En particular, se han utilizado mucho los puntos de extensión. Un punto de extensión en un plug-in indica la posibilidad de que ese plug-in sea a su vez parte de otro, o que haya otros plug-ins que sean parte de él. Esta particularidad permite no sólo la integración de nuestro editor con Hydra, sino también la personalización de menús y botones para él gracias a la creación de puntos de extensión con plug-ins de creación de menús y barras de herramientas.