

**UNIVERSIDAD DE MÁLAGA**

**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**INGENIERÍA INFORMÁTICA**

**HERRAMIENTA PARA EL MODELADO Y CONFIGURACIÓN DE  
MODELOS DE CARACTERÍSTICAS**

**Realizado por:**

Jose Ramón Salazar Ramírez

**Dirigido por:**

Pablo Sánchez Barreiro

Lidia Fuentes Fernández

Departamento  
Lenguajes y Ciencias de la Computación

Málaga, Noviembre 2009



**UNIVERSIDAD DE MÁLAGA**

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente/a Dº/Dª. \_\_\_\_\_

Secretario/a Dº/Dª. \_\_\_\_\_

Vocal Dº/Dª. \_\_\_\_\_

para juzgar el proyecto Fin de Carrera titulado:

**Herramienta para el modelado y configuración de modelos de características**

realizado por Dº Jose Ramón Salazar Ramírez

tutorizado por Dº Pablo Sánchez Barreiro y Dª Lidia Fuentes Fernández

ACORDÓ POR \_\_\_\_\_ OTORGAR LA CALIFICACIÓN DE \_\_\_\_\_

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

El Presidente

El Vocal

El Secretario

Fdo:

Fdo:

Fdo:

Málaga, a            de            del 2009



# Índice general

<b>Lista de figuras</b>	<b>6</b>
<b>1. Introducción</b>	<b>7</b>
1.1. Introducción . . . . .	7
1.2. El proyecto AMPLE . . . . .	10
1.3. Objetivos y Metodología . . . . .	11
1.4. Hydra . . . . .	14
1.5. Estructura del documento . . . . .	15
<b>2. Antecedentes</b>	<b>17</b>
2.1. Líneas de Producto Software . . . . .	17
2.1.1. Modelos de características . . . . .	19
2.1.2. Desarrollo Software Dirigido por Modelos . . . . .	26
2.2. Ingeniería de Lenguajes de Modelado . . . . .	27
2.3. Desarrollo de metamodelos con Ecore . . . . .	28
2.4. Desarrollo de editores de modelado gráficos con GMF . . . . .	30
2.5. Desarrollo de editores de modelado textuales con TEF . . . . .	31
2.6. Resolutores de restricciones . . . . .	33
2.7. Sumario . . . . .	34
<b>3. Estado del arte</b>	<b>35</b>
3.1. Herramientas para el modelado de características . . . . .	35
3.1.1. FAMA . . . . .	35
3.1.2. FeatureIDE . . . . .	36
3.1.3. MFM . . . . .	36
3.1.4. S2T2 . . . . .	37
3.1.5. Requiline . . . . .	37
3.1.6. fmp . . . . .	37
3.1.7. SPLOT . . . . .	38

3.1.8.	xFeature . . . . .	38
3.1.9.	pure::Variants . . . . .	39
3.1.10.	CaptainFeature . . . . .	39
3.1.11.	Sumario herramientas de modelado . . . . .	39
3.2.	Análisis automatizado de los modelos de características . . . . .	40
3.2.1.	CSP . . . . .	40
3.2.2.	SAT . . . . .	41
3.2.3.	BDD . . . . .	41
3.2.4.	OCL . . . . .	42
3.2.5.	Sumario del Análisis Automatizado . . . . .	42
<b>4.</b>	<b>Desarrollo de un editor gráfico para modelos de características</b>	<b>45</b>
4.1.	Introducción . . . . .	45
4.2.	Metamodelo . . . . .	45
4.3.	Editor gráfico . . . . .	46
4.4.	Ejemplo práctico: Caso de uso de un SmartHome . . . . .	48
4.5.	Sumario . . . . .	49
<b>5.</b>	<b>Desarrollo de un editor textual para restricciones</b>	<b>51</b>
5.1.	Introducción . . . . .	51
5.2.	Metamodelo . . . . .	51
5.3.	Editor textual . . . . .	54
5.4.	Ejemplo práctico: Caso de uso de un SmartHome . . . . .	55
5.5.	Sumario . . . . .	55
<b>6.</b>	<b>Desarrollo de un editor gráfico para configuraciones</b>	<b>57</b>
6.1.	Introducción . . . . .	57
6.2.	Validación de restricciones con clones . . . . .	57
6.3.	Metamodelo . . . . .	59
6.4.	Editor gráfico . . . . .	60
6.5.	Ejemplo práctico: Caso de uso de un SmartHome . . . . .	63
6.6.	Sumario . . . . .	63
<b>7.</b>	<b>Validación de restricciones</b>	<b>65</b>
7.1.	Introducción . . . . .	65
7.2.	Validación de configuraciones con restricciones usando Choco . . . . .	66
7.3.	Ejemplo práctico: Caso de uso de un SmartHome . . . . .	66
7.4.	Sumario . . . . .	68

<b>8. Conclusiones y Trabajos Futuros</b>	<b>69</b>
8.1. Conclusiones . . . . .	69
8.1.1. Sumario . . . . .	69
8.1.2. Conclusiones . . . . .	70
8.1.3. Discusión . . . . .	71
8.1.4. Evaluación . . . . .	72
8.2. Trabajos Futuros . . . . .	74
<b>A. Manual de Usuario</b>	<b>83</b>
A.1. Instalación y desinstalación de Hydra . . . . .	83
A.1.1. Instalación . . . . .	84
A.1.2. Desinstalación . . . . .	85
A.2. Actualización . . . . .	87
A.3. Guía de uso . . . . .	87
A.3.1. Creación de proyecto Hydra . . . . .	87
A.3.2. Entorno . . . . .	88
<b>B. Acrónimos</b>	<b>97</b>





# Índice de figuras

1.1. Arquitectura de Hydra . . . . .	14
2.1. Proceso de desarrollo en líneas de productos software . . . . .	18
2.2. Modelo de características con cardinalidad . . . . .	20
2.3. Modelo de características sin cardinalidad . . . . .	20
2.4. Comparación de modelos con y sin referencias . . . . .	21
2.5. Modelo de características de un Smart Home. . . . .	22
2.6. Notaciones utilizadas en los diagramas de características. . . . .	23
2.7. Ejemplo explicativo de modelo de características. . . . .	24
2.8. Configuración de un Smart Home. . . . .	25
2.9. Descripción abstracta de Hydra . . . . .	27
2.10. Ejemplo de metamodelo básico en Ecore . . . . .	29
2.11. Subconjunto simplificado del metamodelo Ecore . . . . .	29
2.12. Flujo de trabajo en GMF . . . . .	31
2.13. Editor TEF de ejemplo . . . . .	32
3.1. Resumen herramientas analizadas. . . . .	39
4.1. Metamodelo del editor para modelos de características . . . . .	47
4.2. Correspondencia entre metaelementos Ecore y notaciones gráficas GMF. . . . .	47
4.3. Imagen del editor gráfico de modelos de características. . . . .	49
4.4. Modelo de características de un caso de uso de un SmartHome. . . . .	50
5.1. Metamodelo del editor de restricciones . . . . .	52
5.2. Editor de restricciones en acción. . . . .	55
5.3. Restricciones para nuestro caso de uso. . . . .	55
6.1. Distintos casos para las restricciones . . . . .	58
6.2. Ejemplo del quinto problema descrito . . . . .	59
6.3. Metamodelo del editor de configuraciones . . . . .	60
6.4. Correspondencia entre metaelementos Ecore y notaciones gráficas GMF. . . . .	61

6.5. Imagen del editor gráfico de configuraciones. . . . .	62
6.6. Imagen del editor gráfico de configuraciones. . . . .	64
7.1. Traducción del diagrama a restricción lógica . . . . .	67
8.1. Caso de uso de un escenario de ventas. . . . .	73
8.2. Caso de uso de un middleware. . . . .	74

# Capítulo 1

## Introducción

Esta memoria de Proyecto Fin de Carrera presenta *Hydra*, una herramienta para el modelado de características<sup>1</sup>, desarrollada dentro del contexto del proyecto AMPLE, para líneas de productos software<sup>2</sup>. Este capítulo introduce los términos Línea de Producto Software y Modelo de Características. A continuación situamos este trabajo dentro del contexto del proyecto AMPLE, y por último describimos la motivación y objetivos de este trabajo.

### 1.1. Introducción

El objetivo de una línea de productos software es crear la infraestructura adecuada para una rápida y fácil producción de sistemas software similares, destinados a un mismo segmento de mercado. Dado que estos productos son similares, comparten una serie de aspectos comunes pero también presentan ciertas variaciones entre ellos[42, 30, 36, 24]. Las líneas de productos software se pueden ver como análogas a las líneas de producción industriales, donde productos similares o idénticos se ensamblan y configuran a partir de piezas prefabricadas bien definidas, que son reutilizadas para la construcción de productos similares. Un ejemplo clásico es la fabricación de automóviles, donde se pueden crear decenas de variaciones de un único modelo de coche con un solo grupo de piezas cuidadosamente diseñadas y una fábrica específicamente concebida para configurar y ensamblar dichas piezas.

Una pieza clave en la creación y desarrollo de una línea de productos es el análisis y especificación de qué elementos son comunes y qué elementos son variables dentro del conjunto de productos similares producidos por la línea de productos software. Para realizar dicha tarea se suelen construir *modelos de características*[31, 9, 16]. Dichos modelos

---

<sup>1</sup>En inglés, *feature modelling*

<sup>2</sup>En inglés, *Software Product Line*

definen una descomposición jerárquica, en forma de árbol, de las características de una serie de productos similares. Un *característica* se define, de forma genérica, como un elemento visible del sistema, de interés para alguna persona que interactúa con el sistema, ya sea un usuario final o un desarrollador software[31]. Por ejemplo, una característica de un coche sería su color. Una característica puede ser descompuesta en varias subcaracterísticas que pueden ser obligatorias, opcionales o alternativas. Continuando con el mismo ejemplo, podemos entender que un utilitario convencional tiene obligatoriamente cuatro ruedas, puede opcionalmente tener GPS y se puede adquirir en una serie de colores alternativos. Para obtener un producto específico, el usuario debe crear una configuración de este modelo de características, es decir, una selección de características que quiere que estén presentes en su producto.

Una configuración o selección de características debe de obedecer las reglas del modelo de características. Por ejemplo, dentro de un conjunto de alternativas mutuamente excluyentes, como el color de un automóvil, sólo una alternativa de entre todas las posibles debe ser seleccionada. Existen una serie de restricciones que no se pueden modelar con la sintaxis básicas de los modelos de características. Un ejemplo de tales tipos de restricciones son las *restricciones de implicación*, que especifican que la selección de una característica obliga a la selección de otra característica diferente. Por ejemplo, la selección de GPS para un automóvil podría obligar a la inclusión en dicho automóvil de mandos integrados en el volante para el manejo tanto de la radio como del GPS. Estas restricciones se suelen modelar de forma externa al modelo de característica, usando algún tipo de formalismo adicional, como lógica proposicional. En nuestro ejemplo, la anterior restricción se modelaría como una simple implicación entre características. Surge por tanto un nuevo reto dentro del modelado de características, que es la validación de las configuraciones respecto a tales restricciones externas.

Por último, cabe destacar que en los últimos años se ha añadido a los modelos de características un simple pero importante concepto, como son las *características clonables*[16], que son características que pueden aparecer con un diferente número o cardinalidad dentro de un producto. Por ejemplo, el modelo de características de una casa puede tener como característica clonable *planta*, dado que una casa posee un número variable de plantas, es decir, usando nuestra terminología, diferentes clones de la característica *planta*. Esta ligera modificación hace que se pueda modelar variabilidad estructural, tal como que una casa tenga un número variable de plantas, en los modelos de características, acercando su potencia expresiva a la de los lenguajes de dominio específico para líneas de producto software[42].

En la actualidad existen diversas de herramientas, en su mayoría académicas, tales como RequiLine[49] o fmp[2], siendo ésta última posiblemente la más conocida. No obstante,

## 1.1. Introducción

---

no existe ninguna herramienta en este momento que posea las siguientes características de forma conjunta:

- Una interfaz gráfica, amigable y que, en la medida de lo posible, asista al usuario en la creación de configuraciones.
- Soporte el modelado y la configuración de características clonables.
- Permita la especificación de restricciones externas entre características, incluyendo características clonables.
- Sea capaz de determinar si una determinada configuración es válida, es decir, no sólo obedece las reglas sintácticas del modelo de características, sino que también se satisfacen las restricciones externas. Estas restricciones externas pueden estar definidas sobre características clonables.

Cabe destacar además que, hasta donde alcanza nuestro conocimiento, no existe herramientas de modelado de características alguna que permita definir y validar restricciones externas que involucren características clonables. Para resolver tales limitaciones de las herramientas actuales para el modelado de características, hemos creado *Hydra*. Hydra es una herramienta para el modelado de características que proporciona:

1. Un editor completamente gráfico y amigable al usuario para la construcción de modelos de características;
2. un editor textual y una sintaxis propia para la especificación de restricciones entre características;
3. Un editor gráfico, asistido y amigable al usuario para la creación de configuraciones de modelos de características;
4. Un validador que comprueba que las configuraciones creadas satisfacen las restricciones definidas para el modelo de características.

Además, Hydra soporta totalmente el modelado de *características clonables*, siendo posible tanto especificar características clonables a nivel de modelo como configurar dichas características clonables. Es también posible definir restricciones que involucren características clonables así como validar tales restricciones. Hydra está implementado como un plug-in para Eclipse y está basado en estándares de facto dentro de la comunidad de modelado, tales como Ecore[44] o GMF[21], lo que favorece su interoperabilidad con otras herramientas.

Como ha sido comentado anteriormente, Hydra se encuadra dentro un proyecto mayor, que es el proyecto AMPLE. La siguiente sección describe la motivación y objetivos de dicho proyecto.

### 1.2. El proyecto AMPLE

El proyecto AMPLE<sup>3</sup> es un proyecto de investigación financiado por la Comisión Europea dentro del 6º Programa Marco. Está compuesto por ocho socios, divididos en cinco universidades (University of Lancaster, Universidade Nova de Lisboa, Technische Universität Darmstad, École des Mines de Nantes, University of Twente y la Universidad de Málaga) más tres socios industriales (Siemens AG, SAP AG y Holos).

Actualmente las líneas de productos software implantadas en entornos industriales están basadas en procesos manuales, basados en herramientas clásicas para el manejo de la variabilidad, tal como compilación condicional y preprocesadores. Dichas técnicas resultan en muchas ocasiones insuficientes. La opción deseable sería disponer de lenguajes de programación y herramientas con soporte específico para la gestión de la variabilidad. Otra carencia común a estas líneas de productos software industriales es la ausencia de una gestión sistemática de la trazabilidad o relaciones existentes entre los artefactos software creados a través de las diferentes etapas del ciclo de vida software.

El objetivo del proyecto AMPLE es proporcionar una metodología de desarrollo para líneas de productos software, que abarque desde la fase de ingeniería de requisitos hasta su implementación, que mejore: (1) la modularización de los elementos variables de una línea de productos software; (2) la gestión de tales elementos variables y (3) la generación y mantenimiento de información sobre la trazabilidad de los diferentes artefactos de una línea de productos software. Para lograr este objetivo, el proyecto AMPLE intenta aplicar técnicas de orientación a aspectos<sup>4</sup> y dirigidas por modelos<sup>5</sup>, con objeto de solventar las carencias actuales de las líneas de productos software.

El desarrollo de software orientado a aspectos<sup>6</sup>[45, 33] pretende mejorar la modularización de los sistemas software, encapsulando en módulos especiales, llamados *aspectos*, propiedades que, usando técnicas convencionales, como orientación a objetos [43], no podrían ser adecuadamente encapsuladas en un único módulo, apareciendo dispersas y enmarañadas a través de los diferentes módulos que componen una aplicación. La aplicación de técnicas aspectuales a las líneas de productos software pretende mejorar la modularización de los elementos variables, mediante su encapsulación en aspectos.

---

<sup>3</sup><http://www.ample-project.net>

<sup>4</sup>En inglés, *aspect-oriented*

<sup>5</sup>En inglés, *model-driven*

<sup>6</sup>En inglés, *aspect-oriented software development* o AOSD

### 1.3. Objetivos y Metodología

---

El desarrollo de software dirigido por modelos [46] considera a los modelos software no como simples medios para documentación o comunicación, sino como artefactos claves para el desarrollo de un sistema software. Un sistema software se desarrolla mediante la definición de diferentes modelos con diferentes niveles de abstracción, más una serie de transformaciones automáticas definidas entre tales modelos. Uno de los objetivos del proyecto AMPLE es aplicar técnicas dirigidas por modelos a las líneas de productos software para automatizar la gestión de los elementos variables, así como la generación de información de trazabilidad entre artefactos definidos a diferentes niveles de abstracción.

La siguiente sección describe las motivaciones y la metodología usadas para crear *Hydra*, una herramienta para el modelado de características que se enmarca dentro del proyecto AMPLE.

### 1.3. Objetivos y Metodología

Como ya se ha comentado en la sección de introducción, existen diversas herramientas para el modelado de características, pero no existe actualmente ninguna herramienta que posea de forma conjunta una serie de elementos de interés para el proyecto AMPLE. Más concretamente, no existe ninguna herramienta que contemple el modelado, configuración y validación de *características clonables*. Estas características clonables son imprescindibles, como se explicará más detalladamente en la Sección 2.1.1, para el modelado de la variabilidad estructural. Por tanto, el objetivo de este proyecto es suplir, en la medida de lo posible, dicha carencia mediante el desarrollo de una herramienta para el modelado de características que soporte el modelado, la configuración y la validación de características clonables. Además, como objetivos secundarios, queremos que dicha herramienta: (1) sea gráfica<sup>7</sup>; (2) amigable al usuario; (3) asista al usuario en la creación de configuraciones en la medida de lo posible; y (4) sea fácilmente integrable con otras herramientas de modelado.

Resumiendo, los objetivos de este proyecto son los que se enumeran a continuación:

1. Desarrollar un editor completamente gráfico y amigable al usuario para la construcción de modelos de características, incluyendo soporte para el modelado de características clonables.
2. Desarrollar un editor textual y una sintaxis propia para la especificación de restricciones entre características, incluyendo restricciones que involucren características clonables.

---

<sup>7</sup>A excepción del editor de restricciones entre características, que deberá ser textual, por razones de usabilidad.

3. Desarrollar Un editor gráfico, asistido y amigable al usuario para la creación de configuraciones de modelos de características, incluyendo soporte para la configuración de características clonables.
4. Crear un validador que compruebe que las configuraciones creadas satisfacen las restricciones definidas para el modelo de características, incluso cuando estas restricciones contengan características clonables.

Para satisfacer dichos objetivos, se realizaron las tareas que se describen a continuación:

1. **Estudio del estado del arte** El objetivo de esta fase es adquirir los conceptos necesarios para la realización del proyecto, más concretamente, familiarizarse con las líneas de productos software[42], modelos de características, sin y con características clonables[34, 16, 3], y el desarrollo software dirigido por modelos[10]. Dentro de esta fase se incluye también un estudio de las herramientas de modelado existentes actualmente, como fmp [15], FAMA [47] o pure::variants [8], así como de las diferentes tecnologías, tales como OCL (Object Constraint Language) [41] o CSP (Constraint Solver Problem) [48], usadas por estas herramientas para la validación de configuraciones.
2. **Desarrollo de un editor gráfico de modelos de características** El primer paso para conseguir nuestros objetivos, fue crear un editor gráfico para la construcción de modelos de características, que incluyese soporte para el modelado de características clonables. De acuerdo con los principios de la ingeniería de lenguajes de modelado (ver Sección 2.2), esta fase implicaba el desarrollo de un *metamodelo* para los modelos de características, más una notación gráfica para dicho metamodelo. De acuerdo con los estándares *de facto* en la comunidad de modelado, el metamodelo se realizó en Ecore [12] y la notación gráfica en GMF (*Graphical Modelling Framework*) [21].
3. **Desarrollo de un editor de restricciones externas entre características** El objetivo de este editor es soportar la especificación de restricciones externas, definidas por el usuario, entre características. Como ya se ha comentado con anterioridad, tales restricciones son expresiones similares a fórmulas lógicas. Estas expresiones resultan más fáciles de construir usando una sintaxis textual en lugar de una sintaxis gráfica. Por tanto, se optó por la creación de un editor textual. Al igual que en en la etapa anterior, el primer paso fue la construcción de un metamodelo para estas restricciones externas, metamodelo que se creó en Ecore de acuerdo con las practicas



### 1.3. Objetivos y Metodología

---

convencionales dentro de la ingeniería de lenguajes de modelado. La notación textual para este metamodelo se desarrolló usando TEF (*Textual Editing Framework*)<sup>8</sup>. Se eligió TEF entre otras alternativas posibles, como xText<sup>9</sup> or TCS (Textual Concrete Syntax) [29], por la facilidad de uso que proporcionaba TEF.

4. **Desarrollo de un editor gráfico de configuraciones** El siguiente paso para satisfacer nuestros objetivos era desarrollar un editor gráfico para la creación de configuraciones de modelos de características, que soportase la configuración de características clonables. Al igual que en los casos anteriores, en primer lugar se creó un metamodelo para configuraciones de modelos de características y a continuación se creó una sintaxis gráfica para dicho metamodelo. Al igual que en los casos anteriores, el metamodelo se creó usando Ecore y la sintaxis gráfica GMF. A diferencia de los casos anteriores, la interfaz gráfica de la herramienta tuvo que ser largamente extendida con acciones y funcionalidades específicas que hiciesen la creación de configuraciones tan asistida como fuese posible.
5. **Desarrollo de un validador de configuraciones** El siguiente paso de acuerdo con nuestros objetivos fue la creación de un validador de configuraciones que comprobase que las configuraciones creadas satisficieran las restricciones externas especificadas por el usuario. Esta tarea implicaba la elección de la técnica de validación más adecuada, así como integrar dicha técnica con los editores creados en los puntos anteriores. La técnica elegida fue el uso de *resolutores de restricciones* [38], más concretamente de la herramienta *Choco* [20]. Los detalles y la justificación de esta elección se comentarán más adelante, más concretamente, en la sección 3.2. Por tanto, en este paso tuvimos que resolver el problema de como transformar la información contenida en el modelo de características, en el modelo de restricciones y el modelo de configuración en información procesable por Choco. La siguiente subtarea fue estudiar como procesar esta información de forma adecuada con Choco, y como interpretar la salida ofrecida por Choco de forma que se convirtiese en información útil para el usuario.
6. **Validación y Pruebas** Con objeto de evaluar, probar y verificar el correcto funcionamiento de nuestra herramienta, creamos modelos de características, restricciones y configuraciones de dos casos de estudio industriales, proporcionados por SAP AG y Siemens AG dentro del contexto del proyecto AMPLE. Las diferentes configuraciones también fueron adecuadamente validadas.

---

<sup>8</sup> <http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/index.html>

<sup>9</sup> <http://www.eclipse.org/Xtext/>

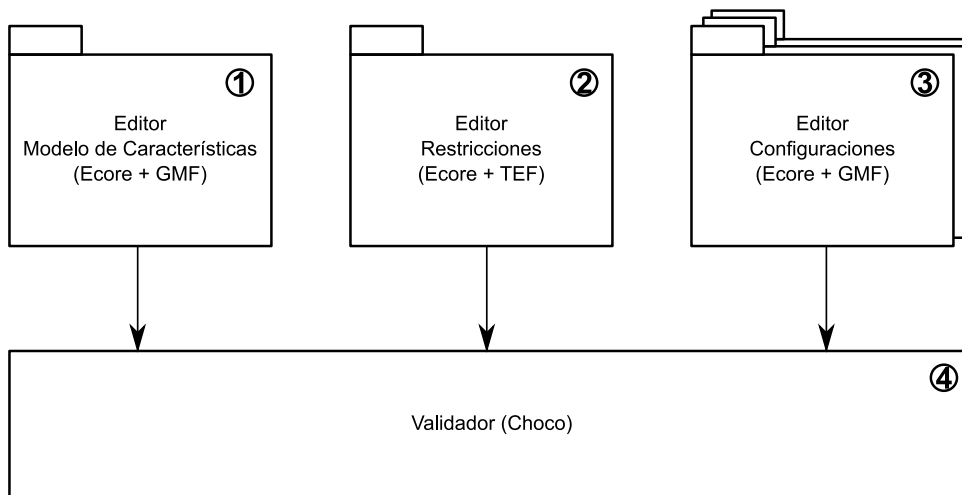


Figura 1.1: Arquitectura de Hydra

La siguiente sección ofrece una descripción general de los principales componentes de *Hydra*, la herramienta desarrollada para satisfacer los objetivos descritos en esta sección siguiendo la metodología arriba descrita.

## 1.4. Hydra: Una herramienta gráfica para la creación, configuración y validación de modelos de características con clones

La metodología descrita en la sección anterior da como resultado una herramienta gráfica para el modelado, configuración y validación de características, incluyendo características clonables, a la cual hemos llamado *Hydra*. Hydra es un monstruo mitológica cuya forma es similar a la de un dragón de múltiples cabezas. Los modelos de características, si se reflejan verticalmente, poseen una forma similar a la de un animal de múltiples cabezas. Por esta razón, decidimos llamar a nuestra herramienta *Hydra*.

En concordancia con la metodología de desarrollo, Hydra posee un arquitectura con cuatro componentes claramente diferenciados, tal como ilustra la Figura1.1. Cada uno de estos componentes se corresponde con cada uno de los puntos 2-5 de la metodología de desarrollo descrita en la sección anterior. Usando estos componentes, el proceso de modelado, configuración y validación de características sería tal como se describe a continuación. Tal proceso, usaremos como ejemplo una familia de productos software para el control automático de hogares inteligentes.

1. En primer lugar, el usuario crearía un modelos de características. Para ello usa el *editor de modelos de características* (Figura 1.1, etiqueta 1). En nuestro ejemplo,

## 1.5. Estructura del documento

---

el usuario crearía un modelo que especificase que una casa puede contener varias plantas, que cada planta puede tener varias habitaciones, y que cada habitación o planta puede tener diferentes facilidades, tales como el manejo automático de luces, ventanas o climatizadores.

2. A continuación, el usuario especificaría explícitamente aquellas restricciones que no se puedan expresar automáticamente usando la sintaxis propia de los modelos de características. Para ello usa el *editor de restricciones* (Figura 1.1, etiqueta 2). Por ejemplo, el usuario crearía una fórmula lógica que especificase que la selección de manejo automático de climatizadores en una planta obliga a la selección de al menos un climatizador en cada habitación existente en dicha planta.
3. Con objeto de crear software para un hogar específico, el usuario crea una configuración del modelo de características creado anteriormente. Para ello usa el *editor de configuraciones* (Figura 1.1, etiqueta 3). De un mismo modelo de características, se puede crear un número variable, incluso ilimitado de configuraciones. En nuestro ejemplo, el usuario especificaría que una casa concreta posee dos plantas, con dos habitaciones en la planta inferior y tres en la superior, y que desea incluir gestión automática de luces y climatizadores para el hogar completo. Esta configuración se realiza de manera tan asistida como sea posible.
4. Para comprobar que esta configuración satisface las restricciones anteriormente definidas, el usuario ejecutaría el *validador* (Figura 1.1, etiqueta 4). Dicho validador comprueba automáticamente si la configuración dada satisface las restricciones definidas por el usuario para el modelo de características. En caso de que no se satisfagan dichas restricciones, el validador informa de manera adecuada acerca de por qué no se satisfacen dichas restricciones, es decir, del origen de los errores en la configuración.

## 1.5. Estructura del documento

Tras este capítulo de introducción, esta memoria se estructura tal como se describe: El Capítulo 2 introduce brevemente los conceptos y herramientas que han sido utilizados para el desarrollo de este proyecto. El Capítulo 3 describe el estado del arte actual del modelado de características y analiza diversas herramientas en este momento existentes para el modelado de características. Dicho capítulo también contiene un análisis de las técnicas de validación utilizadas por las herramientas de modelado de características actualmente existentes. Los Capítulos 4, 5 y 6 describen el desarrollo del editor gráfico para

la creación de modelos de características, del editor textual para la especificación de restricciones externas y del editor gráfico de configuraciones, respectivamente. El Capítulo 7 describe el desarrollo de una de las partes más importantes de Hydra, el validador de configuraciones. El Capítulo 8 contiene conclusiones y trabajos futuros. Por último, se proporciona un manual de usuario como apéndice.

# Capítulo 2

## Antecedentes

Este capítulo provee a grandes rasgos de las técnicas, tecnologías y herramientas utilizadas para la creación de Hydra. El capítulo comienza introduciendo las líneas de productos software donde explica qué es un modelo de características y el desarrollo de software dirigido por modelos. Luego explica los procesos de desarrollo de diferentes tecnologías que se utilizarán para crear Hydra, entre ellos EMF, GMF, TEF y Choco.

### 2.1. Líneas de Producto Software

El objetivo de las líneas de productos software es crear la infraestructura para la rápida producción de sistemas software para un segmento de mercado específico, donde estos sistemas software son similares, y aunque comparten un subconjunto de características comunes, también presentan variaciones entre ellos [42, 30, 36, 24].

Según Clements y Northrop [14], una línea de productos software consiste en: “*Un conjunto de sistemas de software que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento particular de mercado y que se desarrollan a partir de un conjunto común de activos de una forma preestablecida*”.

El principal logro en las líneas de productos software es, construir productos específicos lo más automáticamente posible a partir de un conjunto de elecciones y decisiones adoptadas sobre un modelo común, conocido como modelo de referencia, que representa la familia completa de productos que la línea de productos software cubre.

El desarrollo de líneas de productos software se compone de dos procesos de desarrollo software diferentes pero relacionados (ver figura 2.1), conocidos como *ingeniería del dominio* e *ingeniería de la aplicación*.

En el nivel de *ingeniería del dominio*, empezamos por los documentos de requisitos que describen una familia de productos similares para un segmento de mercado específico. Entonces, diseñamos una arquitectura e implementación de referencia para esta familia de

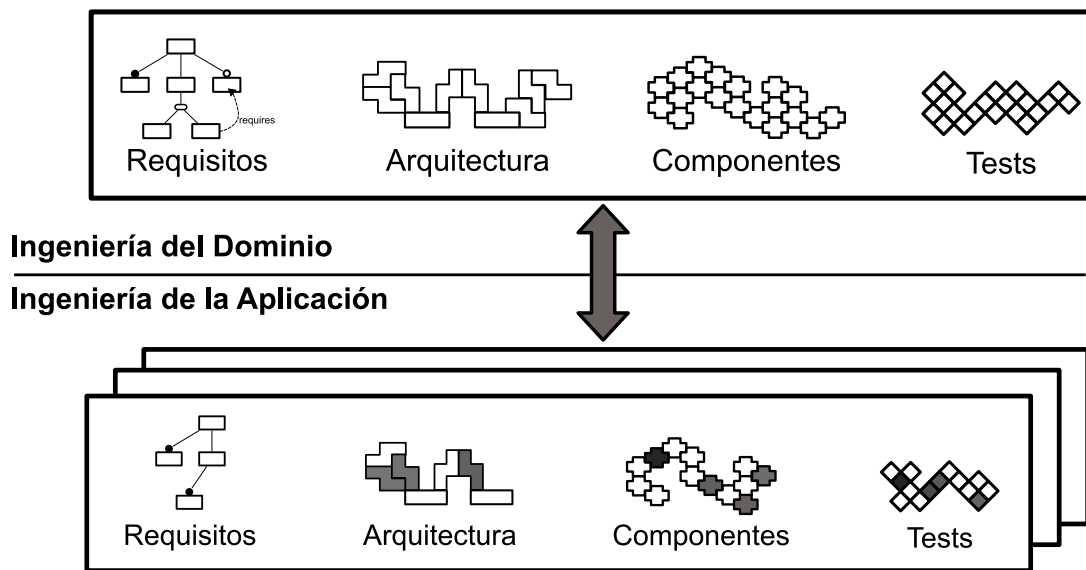


Figura 2.1: Proceso de desarrollo en líneas de productos software

productos. Esta arquitectura de referencia contiene los elementos que son comunes para todos los productos en la familia, pero también debe contener mecanismos para permitir las diferentes variaciones de los diferentes productos pertenecientes a la misma familia.

En el nivel de *ingeniería de la aplicación*, comenzamos con un documento de requisitos de un producto específico. Este documento de requisitos establece las variaciones específicas que deben de ser incluidas en este producto específico. Con esta información, introducimos estas variaciones en la arquitectura de referencia y en la implementación, obteniendo como resultado un producto software único.

El principal beneficio de adoptar la metodología de líneas de productos software es la reducción en tiempo y esfuerzo en desarrollo para desarrollar productos específicos pertenecientes a la misma familia.

La Ingeniería de la Líneas de Productos Software introducen nuevas características comparado con los sistemas de desarrollo de software únicos: *diseño de la variabilidad y gestión y derivación de productos*.

El diseño de variabilidad se refiere a la incorporación de mecanismos de variación (por ejemplo, componentes plugin, mejoras orientadas a aspectos) a los conjuntos centrales que permiten la construcción de un conjunto reutilizable de software que representan el rango completo o la familia de productos, incluyendo ambos sus partes comunes y sus variaciones [5].

La derivación del producto es el proceso de construcción de productos software específicos, dada una configuración específica, por ejemplo, un conjunto válido de variantes, han sido seleccionados, siguiendo las directivas para la composición común y los conjuntos software variables [18].

## 2.1. Líneas de Producto Software

---

### 2.1.1. Modelos de características

Para ser capaces de responder a las necesidades particulares de las demandas y producir líneas de productos software de forma sistemática, una de las cuestiones clave es establecer una forma de especificar los productos que una línea de productos software es capaz de producir. La solución natural para el caso de la línea de productos software parecía simple: los productos de una línea de productos software se diferencian por sus características, siendo una característica 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 es usada para capturar propiedades comunes o diferenciar entre sistemas de una misma familia” [19]. De esta forma, un producto se determina por las características que posee. Para describir todos los productos que una línea de productos software es capaz de producir, es necesario disponer de un modelo que permita describir todas las posibles combinaciones de características de estos productos. Por esta razón, se proponen los modelos de características como forma de describir una línea de productos software. Los modelos de características están reconocidos como una de las contribuciones más importantes en la ingeniería de línea de productos software [19]. Uno de sus objetivos principales es capturar las divergencias (*variabilities*) y aspectos comunes (*commonalities*) entre los distintos productos.

Para ello los modelos de características organizan el conjunto de características jerárquicamente mediante las siguientes relaciones entre ellos:

1. Relaciones entre una característica padre o compuesta y un conjunto de características hijas o subcaracterísticas.
2. Relaciones no jerárquicas que suelen del tipo: si la característica A aparece, entonces la característica B se debe incluir (o excluir).

A las primeras las llamaremos *restricciones estructurales* y a las segundas *restricciones de usuario*.

Desde su aparición en 1990 [31], han habido muchas propuestas que han tratado de extender, modificar o mejorar el modelo de características original. A pesar de ello, hoy en día aún no hay un consenso sobre la notación a utilizar. Básicamente hay dos conjuntos de propuestas: aquellas que proponen modelos de características con cardinalidades (o características clonables) [16] y aquellas que no hacen uso de cardinalidades [3]. Mediante cardinalidades podemos representar algunas relaciones de forma más compacta que sin ellas, siendo en la expresividad de ambas propuestas equivalente en la mayoría de los casos, exceptuando cuando la cardinalidad superior es infinita, es en este caso donde la segunda propuesta no puede dar una solución. Otro motivo por el cual la primera

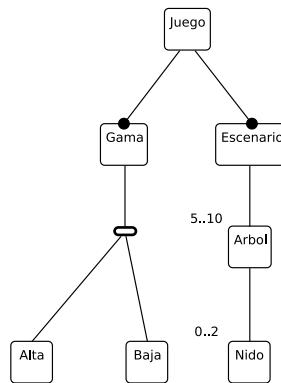


Figura 2.2: Modelo de características con cardinalidad

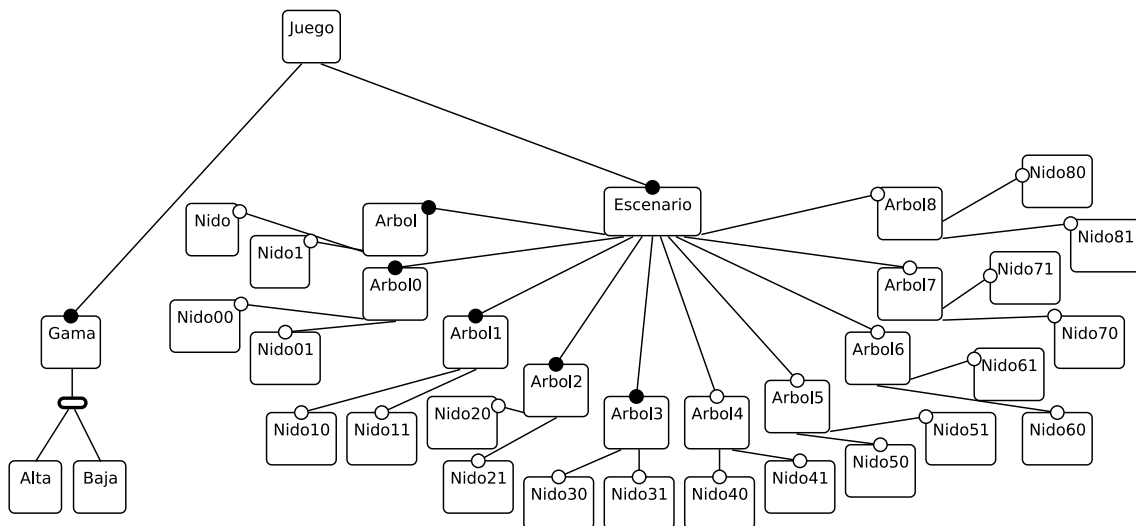


Figura 2.3: Modelo de características sin cardinalidad

propuesta es mejor es por el aumento de legibilidad en los diagramas. Podemos comprobar esta afirmación observando las figuras 2.2 y 2.3, las dos son equivalentes, pero se observa que la primera es mucho más compacta que la segunda ( 7 nodos de características frente a 35).

Por otra parte también existe otra propuesta con menor difusión pero que aporta simpleza y expresividad al diagrama 2.4: se trata de las referencias [16]. Cuando una característica referencia a otra significa que tiene los mismos descendientes que el nodo referenciado. Un nodo solo puede referenciar a otro, en cambio un nodo puede ser referenciado por cualquier número de características (siempre y cuando las referencias no provoquen un ciclo infinito).

Independientemente de la notación utilizada en los modelos de características, existe



## 2.1. Líneas de Producto Software

---

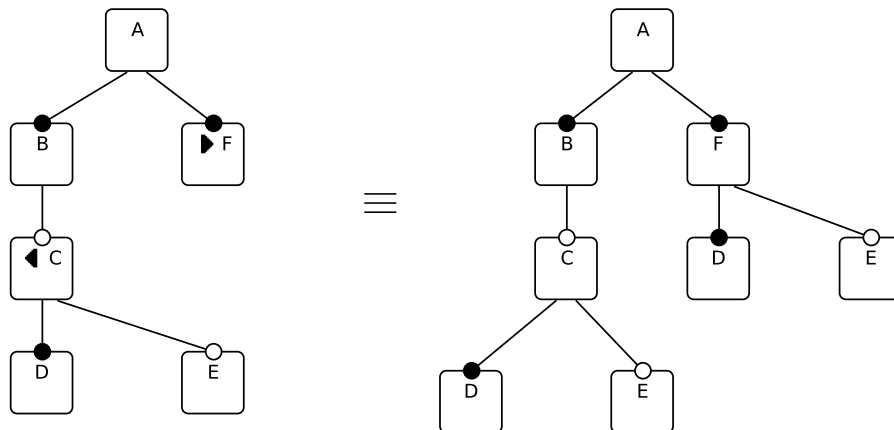


Figura 2.4: Comparación de modelos con y sin referencias

un problema con ellos que aún no está suficientemente resuelto: el análisis automático de los modelos de características [4]. Este análisis consiste en la observación de propiedades como el número de productos que describe (cardinalidad), qué productos describe con unas determinadas características (filtros) y si contiene algún tipo de error. Los modelos de características se utilizan como entradas para otros procesos del desarrollo de línea de productos software como en la ingeniería de requisitos o programación orientada a características. Por tanto el análisis automático de modelos de características puede ser de ayuda para aquellos procesos que se basan en ellos.

La edición de modelos de características es una actividad propensa a errores [31]. La elevada complejidad que pueden alcanzar los modelos de características conteniendo cientos de características y relaciones, hacen de la gestión manual inviable en estos casos. Pueden aparecer errores como características muertas (dead feature), falsas características opcionales (false-optional features) y modelos de características que no contienen productos o vacíos (void feature models). Es necesario tratar estos errores para no afectar al desarrollo de su línea de productos software. En el tratamiento se engloban tres actividades: la detección de errores, explicación de los errores detectados y resolución de los mismos. Puesto que la gestión manual es inviable en la mayoría de los casos, es necesario un soporte automatizado para realizar estas operaciones.

### Diagrama de características

Un diagrama de características es la representación gráfica de un modelo de características. Nosotros vamos a utilizar la notación expuesta por Czarnecki en [16] que incluye la descripción de las características clonables (con cardinalidad superior mayor a uno)

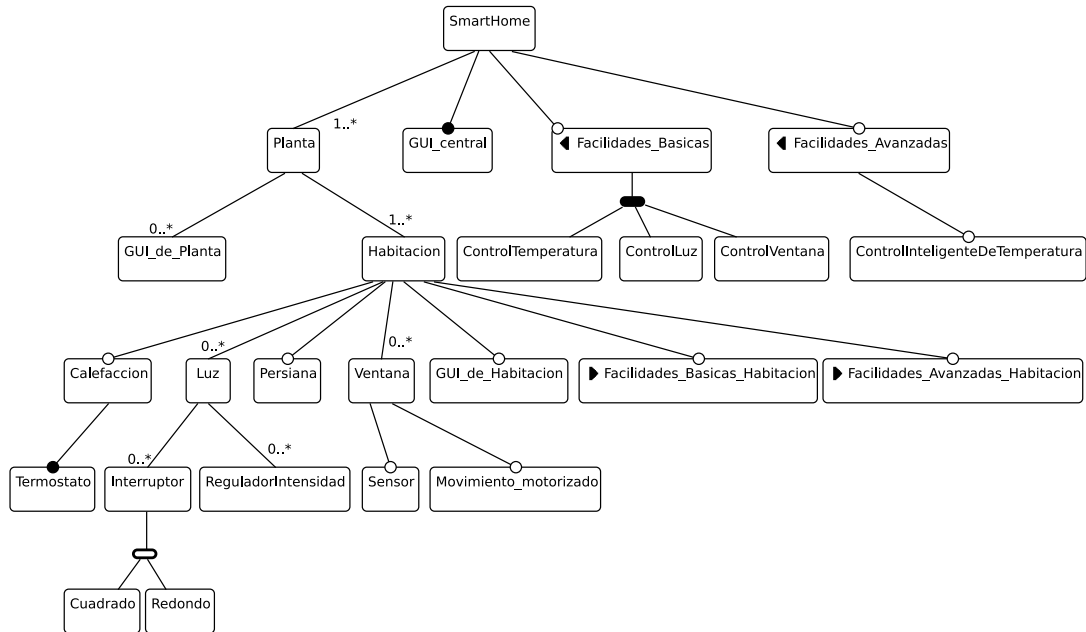


Figura 2.5: Modelo de características de un Smart Home.

y las referencias, ya que consideramos que de esta forma se aporta mayor expresividad, por tanto el usuario no está sujeto a las restricciones de implementación, además de que los diagramas pueden ser mucho más compactos.

Pero, ¿qué es una característica?. Podemos encontrar una definición del mismo en [31], “una característica es una característica visible por el usuario final de un sistema”.

En la figura 2.5 observamos un ejemplo de diagrama de características con la mayoría de las funcionalidades explicadas. Este ejemplo corresponde al caso de estudio de un Smart Home que tiene un número variable de plantas y habitaciones que ofrecen los siguientes servicios categorizados en básicos y avanzados.

En él podemos observar que una casa inteligente tiene al menos una planta, cada planta puede tener un número indeterminado de interfaces de usuario, así mismo, cada planta contiene al menos una habitación, donde podemos controlar o no la calefacción, persiana, un número variable de luces y ventanas...

Las relaciones entre características son las siguientes:

1. **Opcional:** La característica hija puede estar o no seleccionada.
2. **Obligatoria:** La característica es requerida.
3. **Simple:** La característica tendrá cardinalidad  $\langle m..n \rangle$ , siendo  $0 \leq m \leq n$ ,  $m \in \mathbb{N}$ ,  $n \in \mathbb{N} \cup \infty$  y  $m$  el mínimo número de características que podemos seleccionar y  $n$  el máximo.

## 2.1. Líneas de Producto Software

---

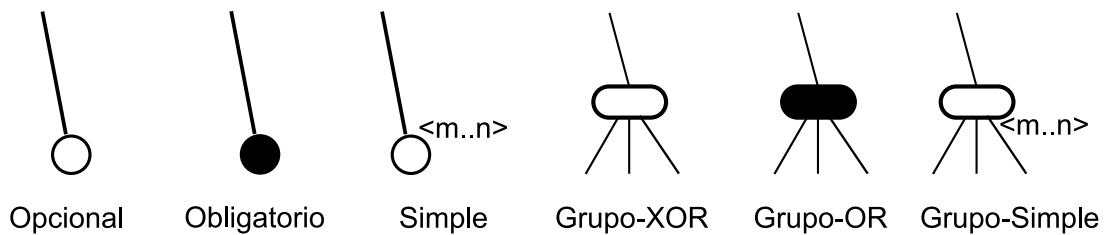


Figura 2.6: Notaciones utilizadas en los diagramas de características.

4. **Grupo-xor:** Sólo una características de las pertenecientes al grupo será seleccionada.
5. **Grupo-or:** Podremos seleccionar mínimo una característica de las pertenecientes al grupo, como máximo todas.
6. **Grupo-Simple:** El número de características seleccionadas del grupo vendrá dado por su cardinalidad  $\langle m..n \rangle$ , siendo  $0 \leq m \leq n \leq k$  ( $k$  es el número de hijos del grupo de características) y el número de características seleccionadas perteneciente a ese intervalo.

Además dispondremos de restricciones de usuario como la implicación entre características y otras que hablaremos profundamente más adelante.

### Operaciones en el análisis automatizado de los modelos de características

El análisis automatizado de los modelos de características es una tarea reconocida como crítica desde sus inicios[31]. No existe un consenso de qué operaciones deben de estar incluidas en esta tarea. Sin embargo vamos a analizar algunas de las más interesantes extraídas de [7].

- **Satisfacibilidad del modelo de característica.** La satisfacibilidad de un modelo de características es una operación básica. Un modelo de características es satisfacible (o válido) si al menos podemos derivar un producto a partir de él. Un modelo de características básico sin restricciones externas y bien construido nunca será insatisfacible. Por tanto la satisfacibilidad vendrá dada por las restricciones de usuario.
- **Cálculo del número de productos.** El cálculo del número de productos de un modelo de características revela información acerca de la flexibilidad y complejidad de la línea de productos software. Mientras mayor sea el número de productos, mayor será la flexibilidad y más complejo será nuestra línea de productos. Con

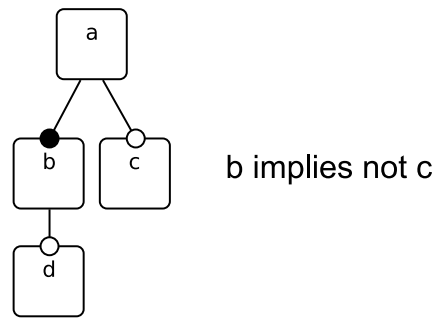


Figura 2.7: Ejemplo explicativo de modelo de características.

esta operación podemos determinar la satisfacibilidad del modelo, si el número de productos es mayor de cero, entonces nuestro modelo es satisfacible.

- **Obtención de configuración mínima** Es posible que queramos una configuración con el menor número de características que sea válido y se derive de un modelo de características.
- **Especializar un modelo** Esta operación reduce el número de productos de un modelo de características mediante la selección (o desección) de características. Las operaciones anteriores están basadas en modelos estáticos, es decir, que no cambian en el tiempo, sin embargo los modelos de características evolucionan en el tiempo y esto cambia el número de productos en cada paso. Como último paso obtendremos un modelo de características que represente únicamente un producto
- **Propagación** Esta operación devuelve un modelo de características donde algunas características son automáticamente seleccionadas (o deseleccionadas). Esta operación tiene sentido durante un proceso de configuración por etapas donde el usuario puede elegir (o eliminar) características y la plataforma automáticamente propaga su decisión por el modelo.
- **Detección de características muertas** Un modelo de características puede tener inconsistencias internas que conducen a la aparición de características muertas. Una característica muerta es una característica que nunca aparece en ninguna configuración válida de un modelo de características.
- **Motivos de inconsistencia** Esta operación devuelve los motivos por el cual un modelo de características dado es insatisfacible.

Por ejemplo, analicemos el modelo de características de la figura 2.7. Este modelo deriva dos productos, por lo que el modelo es satisfacible. Su configuración mínima

## 2.1. Líneas de Producto Software

---

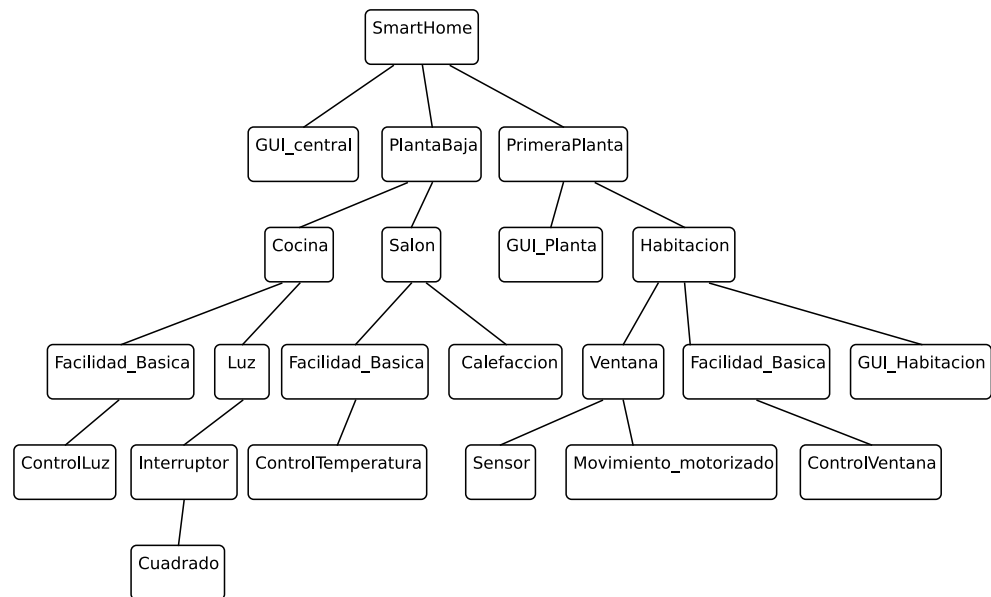


Figura 2.8: Configuración de un Smart Home.

está compuesta por las características a y b. La característica c es una característica muerta, puesto que su selección violaría la restricción de usuario  $b \text{ implies not } c$ , de este modo sería interesante que mediante propagación se eliminase este nodo, puesto que lo único que hace es inducir a error. Por otra parte, en caso de que fuese seleccionada, se debería de explicar que se está violando la restricción.

### Configuración de un modelo de características

Un modelo de características[17] describe el espacio de configuraciones de una familia de sistemas. El ingeniero podrá especificar un miembro de una familia de sistemas mediante la selección de las características deseadas del modelo de características dentro de las restricciones de variabilidad definidas por el modelo (por ejemplo, la elección de un elemento de un conjunto de características alternativas).

El proceso de derivar una configuración de un diagrama de características también se conoce como *proceso de configuración*. El *proceso de especialización* es un proceso de transformación que a partir de un diagrama de características, se obtiene otro diagrama de características, de manera que el conjunto de las configuraciones denotada por el diagrama de este último es un subconjunto de las configuraciones denotada por el primer diagrama. Por tanto podemos decir que el diagrama de este último es una especialización del anterior. Un diagrama de características totalmente especializado denota una sola configuración (ver Figura 2.8).

La relación entre un diagrama de características y una configuración es comparable a la existente entre una clase y su instancia en programación orientada a objetos.

El proceso de la especificación de un miembro de la familia también se puede realizar en etapas, donde en cada fase se eliminan algunas configuraciones. Nos referiremos a este proceso como *configuración por etapas*.

### 2.1.2. Desarrollo Software Dirigido por Modelos

El desarrollo software dirigido por modelos<sup>1</sup>[10], es una nueva tecnología para desarrollo software donde los modelos son protagonistas en el proceso de desarrollo software, en vez de simples medios para el propósito de documentación. Utilizando un enfoque dirigido por modelos, el producto software se obtiene mediante sucesivos refinamientos de modelos definidos a diferentes niveles de abstracción. Estos modelos son automáticamente procesados por herramientas, permitiendo que partes de un modelo a un nivel de abstracción específico a sean generadas automáticamente desde los modelos definidos a más alto nivel de abstracción. Así, cada propiedad de un sistema software (por ejemplo, un sistema distribuido de comunicación) puede ser especificado por medio de un nivel de abstracción más adecuado para esa propiedad y refinada sucesivamente por medio de una transformación de modelo automático hasta que el código implementado es obtenido [10]. El principal beneficio de las técnicas de MDD es la automatización de tareas repetitivas, quienes lideran la reducción en el esfuerzo necesario en el desarrollo y ayuda a incrementar la calidad.

En la línea de productos software, la composición de un producto a partir de un conjunto de recursos software reutilizables es un consumidor de tiempo y un proceso pesado. Por ejemplo, a nivel de código, la instanciación de un producto específico dentro de un línea de productos software a menudo implica escribir grandes archivos de configuración y scripts de compilación, con dependencias complicadas entre ellos. A fin de superar este inconveniente, los diferentes intentos de aplicar técnicas dirigidas por modelos a Ingeniería de Productos Software han emergido durante los últimos años [25]. Ellos quieren automatizar las tareas repetitivas, propensa a errores y consumidoras de tiempo de los procesos de desarrollo del línea de productos software, como la composición de recursos reutilizables software.

Una vez definidos los conceptos en los que se engloba el proyecto, continuaremos describiendo las tecnologías a utilizar.

---

<sup>1</sup>En inglés, *Model-Driven Development* o MDD

## 2.2. Ingeniería de Lenguajes de Modelado

---

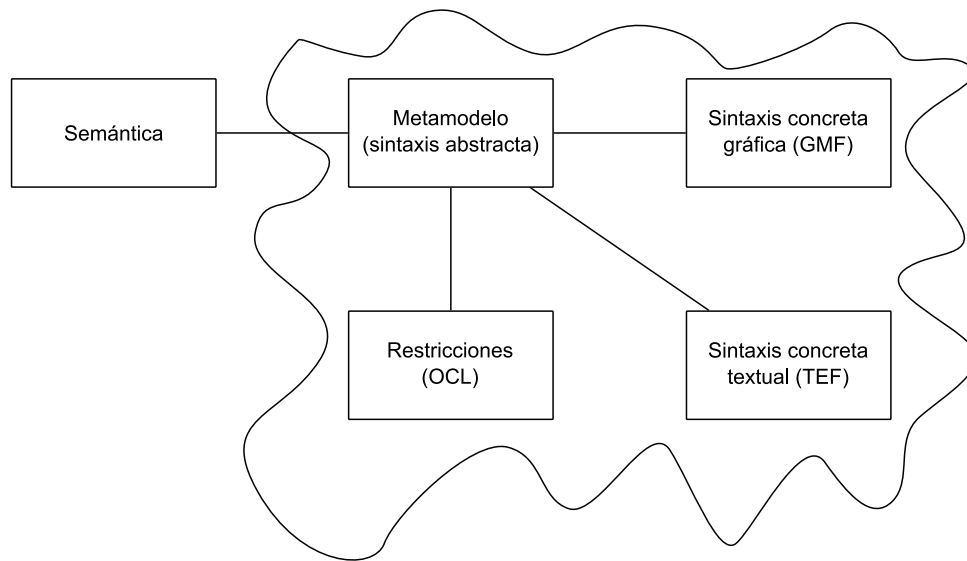


Figura 2.9: Descripción abstracta de Hydra

## 2.2. Ingeniería de Lenguajes de Modelado

Con el auge del desarrollo software dirigido por modelos aumenta la necesidad de crear nuevos lenguajes de modelado. Ello lleva a la aparición de una nueva disciplina, que es la *ingeniería de lenguajes de modelado* [35]. El proceso de ingeniería de un nuevo lenguaje de modelado está compuesto de diversas fases, las cuales se explican con ayuda de la Figura 2.9:

1. El primer paso es crear las reglas de construcción o sintaxis de nuestro lenguaje de modelado. Esto se realiza mediante la creación de un metamodelo, o modelo de nuestro lenguaje que describe usando conceptos parecidos a los de los diagramas de clases, la *sintaxis abstracta* o reglas para la construcción de modelos de nuestro lenguaje. Dicho metamodelo se construye usando un lenguaje de metamodelado. Existen diversos lenguajes de metamodelado, tales como KM3 [28] o MOF [22], aunque Ecore [12] está considerado como el estándar *de facto* dentro de la comunidad de modelado. Normalmente no es posible especificar cualquier tipo de restricción entre los elementos de un lenguaje de modelado usando exclusivamente los elementos del lenguaje de metamodelado. Por ello, tales lenguajes de metamodelado se acompañan con un lenguaje de especificación de restricciones. OCL [41] es el lenguaje de restricciones más usado para desarrollar esta tarea.
2. Como se ha comentado en el punto anterior, un metamodelo establece la sintaxis abstracta de nuestro lenguaje de modelado, pero no especifica la *sintaxis concreta* o notación de nuestro lenguaje de modelado. Por tanto el siguiente paso en la inge-

nería de un nuevo lenguaje de modelado, es la definición de una notación o sintaxis concreta para nuestro lenguaje. Esta notación puede ser tanto textual como gráfica. Para la definición de notaciones gráficas, GMF es la herramienta más popular cuando se trabaja con Ecore. Para notaciones textuales, TCS [29], xText <sup>2</sup> y TEF <sup>3</sup> pueden ser usadas, no existiendo aún un estándar *de facto* para la definición de notaciones textuales.

3. Por último, nos quedaría definir la semántica del lenguaje de modelado. Existen un amplio rango de técnicas para desarrollar esta tarea, tales como: (1) definir la semántica de cada elemento de manera informal; (2) especificar la semántica de cada elemento usando un lenguaje formal, tales como máquinas de estado abstractas; o (3) crear generadores que transformen los elementos de modelado en elementos de otro lenguaje con un significado bien definido.

Las siguientes subsecciones describen brevemente las herramientas usadas para abordar cada uno de estos puntos en este proyecto, a excepción del último punto que está fuera nuestro ámbito.

### 2.3. Desarrollo de metamodelos con Ecore

EMF<sup>4</sup>[12] es un marco de trabajo de Eclipse que unifica Java, XML y UML, permitiendo a los desarrolladores construir rápidamente aplicaciones robustas basadas en modelos simples.

El metamodelo usado para representar modelos en EMF se llama Ecore. Ecore es en sí mismo, un metamodelo EMF, y así es su propio metamodelo, es decir, un metametamodelo.

Podemos observar en la figura 2.10 un ejemplo básico de Ecore con el que vamos a ver sus partes más importantes. El metamodelo define grafos, éstos están compuestos de un número indeterminado de nodos y de relaciones. Las relaciones tienen un origen y un destino, ambos referencian a un nodo, el cual tiene como atributos “nombre” (de tipo EString, una subclase de String), y “tipo” (el cual tomará como valor uno de los literales que están definidos en el tipo especial enumerado definido por el usuario TipoNodo).

En la figura 2.11 podemos apreciar el núcleo del metamodelo Ecore con los elementos principales para definir un metamodelo. Esencialmente este modelo define cuatro tipos de objetos.

---

<sup>2</sup><http://www.eclipse.org/Xtext/>

<sup>3</sup><http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/index.html>

<sup>4</sup>Siglas del inglés *Eclipse Modelling Framework*



### 2.3. Desarrollo de metamodelos con Ecore

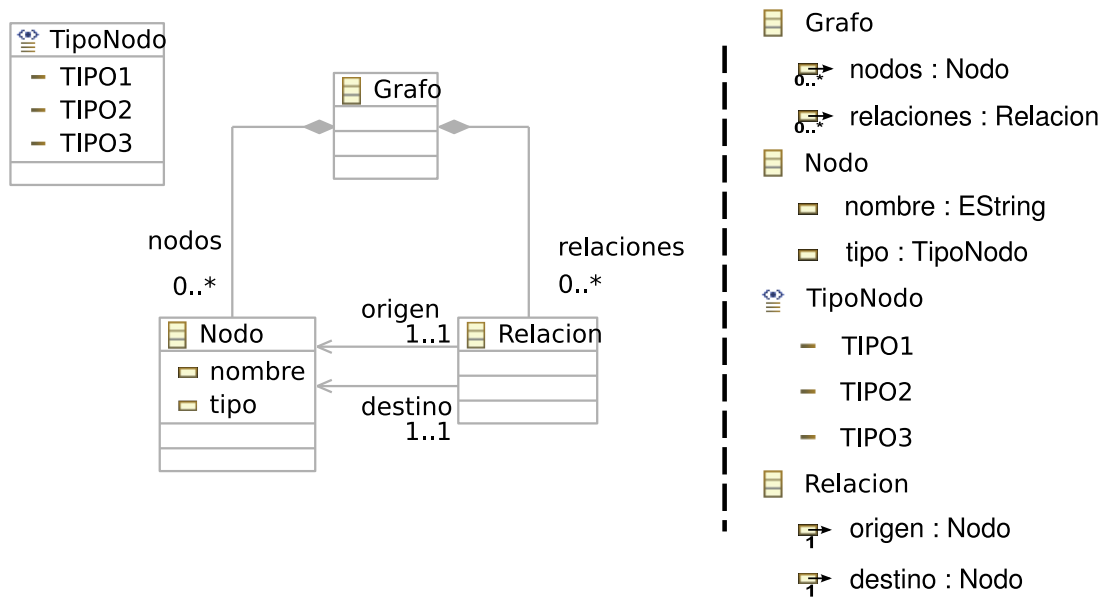


Figura 2.10: Ejemplo de metamodelo básico en Ecore

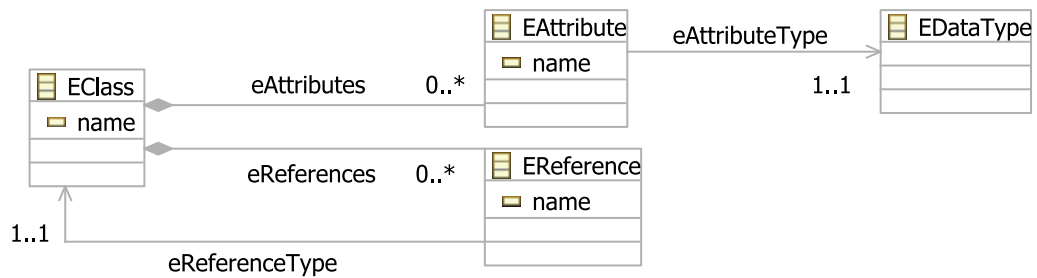


Figura 2.11: Subconjunto simplificado del metamodelo Ecore

1. **EClass** define clases realmente. Las clases están definidas por el nombre y pueden contener un número de atributos y referencias. Para dar soporte a la herencia, una clase puede referir a otras clases como supertipos.
2. **EAttribute** modela atributos, el componente de los datos de un objeto. Está identificado por un nombre, y tiene un tipo.
3. **EDatatype** modela los tipos de los atributos, representando tipos de datos primitivos y objetos definidos en Java, pero no en EMF. También están identificados por un nombre.
4. **EReference** es utilizado como asociación entre clases; modela el final de una asociación. Al igual que con los atributos, las referencias están identificadas por un nombre y tienen un tipo. En cualquier caso, este tipo debe de ser EClass. Si la asociación

es navegable en la otra dirección, debe de existir una referencia correspondiente. En una referencia se especifican los límites superiores e inferiores de su multiplicidad. Finalmente, una referencia puede ser usada para representar una asociación fuerte, llamada “contención”, la referencia especifica si hay que realizar la semántica de contención.

Podemos advertir que los nombres de las clases corresponden cercanamente a los términos UML. Esto es porque Ecore es un subconjunto pequeño y simplificado de UML (se utiliza un subconjunto de UML en vez del mismo porque éste es demasiado ambicioso modelando, más de lo que el núcleo de EMF puede soportar).

### 2.4. Desarrollo de editores de modelado gráficos con GMF

Desde que nació la idea de GMF<sup>5</sup>, se decidió que sería dirigido por modelos lo máximo posible. De esta forma tenemos que un diagrama se define mediante una colección de modelos que conducen a los generadores de código.

La figura 2.12 ilustra los componentes principales y modelos utilizados dentro de un proyecto GMF[21]. Para empezar, un proyecto GMF se crea y hace referencia a un modelo del dominio. Un modelo de definición gráfica diseña las figuras (nodos, enlaces, compartimientos...) que se utilizan para representar los elementos del modelo de dominio en el diagrama. Existe un modelo de definición de herramientas que define la paleta de herramientas y otros elementos para el uso en la creación de diagramas. El modelo de asignación enlaza los elementos de las definiciones gráficas y de herramientas al modelo del dominio. Seguidamente se realiza una transformación desde el modelo de asignación a un modelo generador, donde podremos modificar algunos detalles. Finalmente se realiza una transformación a código, generando un plug-in para crear nuestro diagrama.

**Modelo de definición gráfica** El modelo de definición gráfica define los elementos gráficos presentes en la superficie del diagrama. Se utilizará una galería de figuras (formas, etiquetas, líneas, etc) para representar los nodos, conexiones, compartimientos, y las etiquetas de diagrama. Las figuras podrán ser reutilizadas, es decir, dos elementos diferentes pueden ser representados por la misma figura.

**Modelo de definición de herramientas** Los diagramas suelen incluir una paleta de herramientas para crear y trabajar con contenido gráfico. El propósito del modelo

---

<sup>5</sup>Siglas del inglés *Graphical Modelling Framework*

## 2.5. Desarrollo de editores de modelado textuales con TEF

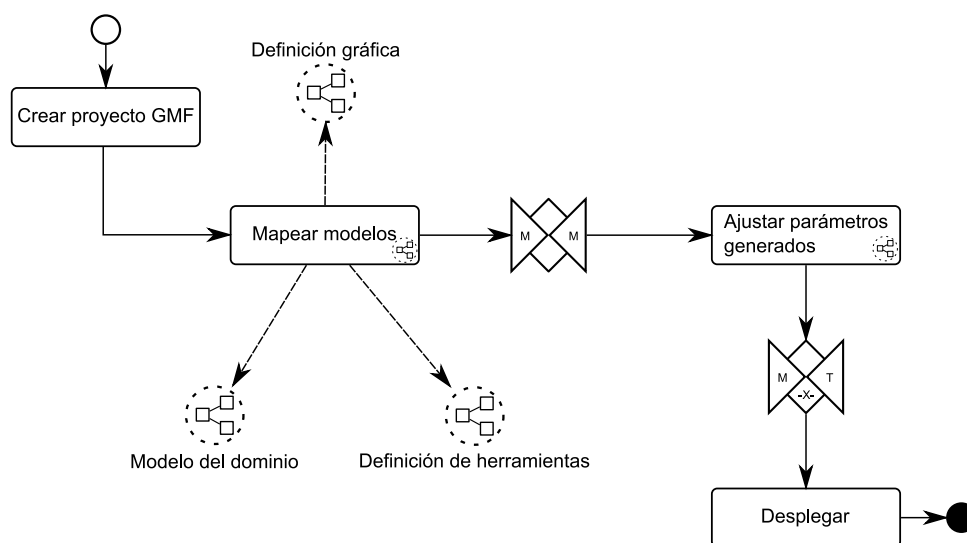


Figura 2.12: Flujo de trabajo en GMF

es la definición de herramientas para especificar estos elementos. El modelo de herramientas incluye actualmente a los elementos de la paleta, la barra de herramientas y otros menús, aunque lamentablemente en la versión actual de GMF, el generador utiliza solo el elemento de la paleta.

**Modelo de asignación** Tal vez el más importante de todos los modelos de GMF es el modelo de asignación. Aquí, los elementos de la definición del diagrama (nodos y enlaces) se asignan al modelo de dominio y a los elementos de las herramientas. El modelo de asignación utiliza Object Constraint Language[41] (OCL) de muchas maneras, incluyendo la inicialización de las características de los elementos creados, la definición de enlaces, las limitaciones de nodos, y la definición de modelo de auditorías y métricas. Las auditorías identifican los problemas en la estructura o el estilo de un diagrama y su subyacente modelo de dominio.

**Modelo generador** Como se menciona en el resumen, el modelo generador añade información al modelo de asignación para generar código.

## 2.5. Desarrollo de editores de modelado textuales con TEF

El marco de edición textual TEF permite crear editores de texto para lenguajes basados en EMF. TEF provee un lenguaje de definición de sintaxis llamado TSL (basado en una gramática (E)BNF[26]). En este lenguaje, el ingeniero puede describir una notación textual dado un metamodelo Ecore. Para esta descripción TSL, el marco puede

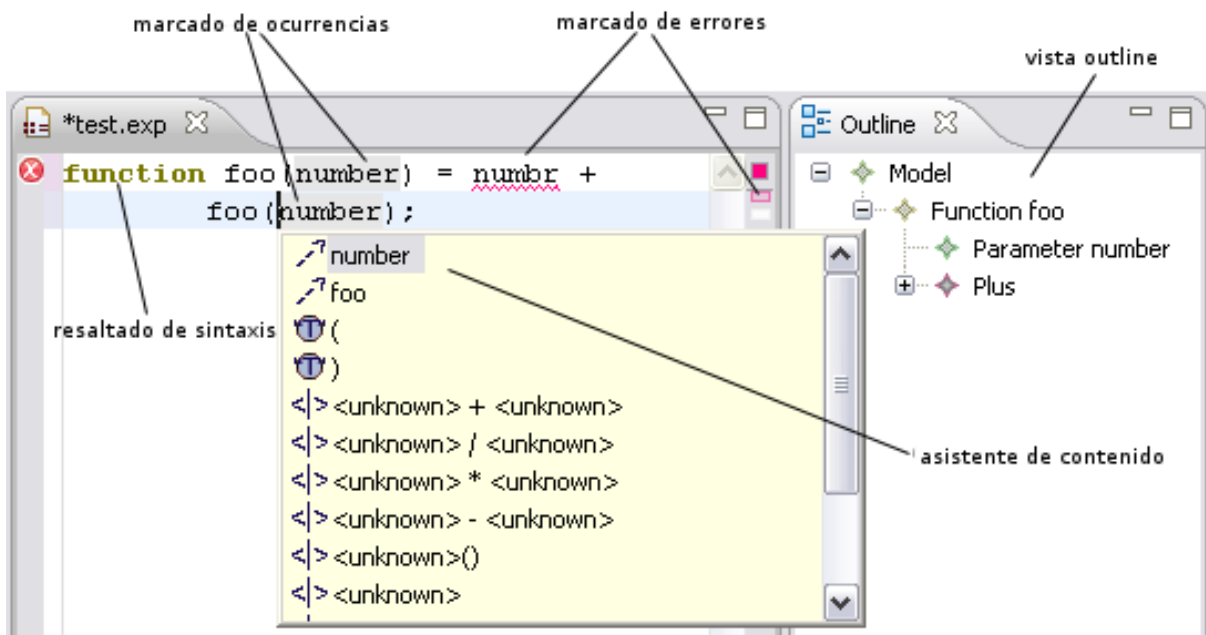


Figura 2.13: Editor TEF de ejemplo

generar automáticamente un editor TEF. Los editores TEF son editores de texto decorados mediante Eclipse. Esto significa que tienen las mismas facilidades que por ejemplo, un editor Java. Algunas de sus características son: resaltado de sintaxis, asistencia a la auto-conclusión de código, navegación inteligente, información contextual o visionado de errores.

TEF es en sí mismo un plugin eclipse, y cada editor TEF creado será otro plugin eclipse.

Para crear un editor TEF es necesario crear primeramente un metamodelo que se adapte a las necesidades de nuestro problema. Luego se define la gramática TSL, donde además asignamos los elementos de la gramática con los elementos del metamodelo. El resto del trabajo es automático y se generará el código necesario para crear un editor textual para nuestro problema. Este código puede ser modificado para adaptar ciertos comportamientos que no se pueden definir en la gramática.

En la figura 2.13 se observa un editor textual de ejemplo TEF donde se pueden notar la mayoría de sus características. Nótese que el texto introducido es una instancia del modelo Ecore definido, como se puede observar en la vista outline. Por tanto, fácilmente podremos exportar nuestro fichero de texto a XML o semejantes.

## 2.6. Resolutores de restricciones

Un modelo de características con restricciones adicionales puede ser traducido a un conjunto de variables y un conjunto de restricciones sobre esas variables. Una asignación de valor que satisface las restricciones corresponde a una configuración correcta. Si asignamos valores a todas las variables y se satisfacen las restricciones tenemos una configuración completa. Si la asignación de valores es parcial (es decir, no a todas las variables) tenemos una especialización parcial[17], que a veces es también citada como configuración parcial.

La satisfacción de restricciones ofrece numerosos algoritmos que son de interés para el modelado de características y las herramientas de configuración basadas en características. Los principales algoritmos de interés incluyen:

- Chequeo de restricciones: Comprueba si una asignación de variables (completa o parcial) satisface las restricciones.
- Propagación de restricciones: Infere los valores de las variables sin valores desde los valores de las variables con valor.
- Satisfacibilidad de restricciones: Comprueba si un conjunto de restricciones tiene al menos una solución.
- Soluciones con restricciones: Calcula las soluciones para un conjunto de restricciones.
- Cálculo del número de soluciones: Calcula el número de soluciones completas para un conjunto de restricciones.

Una herramienta de modelado de características puede ofrecer facilidades basadas en algoritmos de satisfacción de restricciones para ayudar al desarrollador crear modelos de características correctos. Ejemplos de estas facilidades son:

- Chequeo de consistencia del modelo: La satisfacibilidad de restricciones permite determinar si un modelo de características tiene al menos una configuración correcta.
- Detección de anomalías: Anomalías como “características muertas”, por ejemplo, características que no son parte de ninguna configuración correcta y así nunca serán seleccionadas[50].
- Métricas: El número de configuraciones correctas es una métrica útil indicando el grado de variabilidad de un modelo de características [51]. Puede ser computado calculando el número de soluciones existentes que satisfacen las restricciones.

Los algoritmos de satisfacción de restricciones son particularmente interesantes en las herramientas de configuraciones basadas en características. Ejemplos de estas facilidades basadas en la satisfacción de restricciones en configuraciones basadas en características son:

- **Chequeo de configuración:** Comprobar que una configuración completa satisface un conjunto de restricciones es relativamente simple.
- **Mostrar el número de configuraciones restantes:** Calcular el número de configuraciones después de cada paso en la configuración proporciona al usuario una mejor idea de como marcha el proceso de configuración. Si el número es uno, el usuario ha encontrado una configuración completa. Si el número es cero, el usuario ha creado una configuración incorrecta y necesitará revisar los pasos que ha seguido para crear la configuración.
- **Propagación de elecciones en la configuración:** Seleccionar una característica puede requerir que otras características estén seleccionadas en cuyo caso pueden ser inferidos automáticamente y seleccionados en pasos posteriores utilizando la propagación de restricciones. En general, el valor de una característica puede restringir el dominio de los valores disponibles de otras variables.
- **Autocompletación de configuraciones:** Resolver las restricciones puede ser usado para la computación de las soluciones (incluyendo soluciones parciales) que pueden ser presentados al usuario.
- **Depuración de configuraciones incorrectas:** En el caso de que se dé una configuración incorrecta, se debe de mostrar el origen de la misma para que el usuario pueda tomar las medidas correctas.

### 2.7. Sumario

Durante el capítulo de antecedentes hemos relatado los conceptos necesarios para entender el ámbito del problema, así como los procesos de desarrollo genéricos de las tecnologías que utilizaremos para desarrollar Hydra.

En el siguiente capítulo profundizaremos en el estado actual de las herramientas de modelado de características así como de las alternativas existentes para la resolución de restricciones, con el que concluiremos

# Capítulo 3

## Estado del arte

*Para entender el presente y comprender el futuro, antes hay que conocer el pasado.*

En este capítulo estudiaremos el estado del arte actual, con el que analizaremos las distintas herramientas de modelado de características y de análisis de los modelos para seleccionar las mejores cualidades de ellos para el primer caso y escoger el mejor analizador en el segundo caso.

### 3.1. Herramientas para el modelado de características

Hasta la fecha, existen multitud de herramientas para el modelado e características, la mayoría fueron realizadas por distintas universidades como proyectos de investigación y publicadas relativamente hace poco tiempo. La mayoría coincide en permitir crear modelos de características, otros permiten hacer solo configuraciones gráficamente, algunos solo validan los modelos, en cualquier caso todas las herramientas tienen alguna carencia. Interesa especialmente saber cómo funcionan realmente estos, qué analizadores utilizan y qué funcionalidades han añadido y cuales no.

En las siguientes subsecciones iremos dando un repaso a casi la totalidad de herramientas publicadas y creadas para este desempeño.

#### 3.1.1. FAMA

FAMA[47] está desarrollado por un equipo de la Universidad de Sevilla, liderado por David Benavides. Permite el análisis automatizado de los modelos de características integrando algunos de los resolutores más comúnmente propuestos (BDD, SAT y CSP), siendo esta una característica poco común. Dispone de un plugin de Eclipse gráfico desarrollado

en EMF bajo licencia Open-Source. Se puede obtener el motor de análisis por separado, de hecho FAMA está trabajando conjuntamente con MOSKITT o pure::Variants para integrar sus herramientas. El proyecto está en funcionamiento actualmente y lanzaron su última versión en Agosto del 2009.

Entre sus bondades señalables encontramos que dispone de atributos en las características, aunque todavía no es totalmente funcional este rasgo. Por contra no dispone de características clonables, y las restricciones que permite son muy básicas (únicamente permite implicación y exclusión). Tampoco permite las referencias entre características ni realizar configuraciones.

#### 3.1.2. FeatureIDE

FeatureIDE[32] está desarrollado por la University of Magdeburg de Alemania. Es un IDE basado en Eclipse que integra AHEAD, FeatureC++ y herramientas de FeatureHouse como herramientas de composición y compiladores entre otros. Es tanto un editor gráfico como textual, puesto que la construcción de los modelos de características están basados en la gramática GUIDSL [3]. Permite configuraciones y la creación de restricciones avanzadas. Utiliza como resolutor *SAT*. Como defectos podemos citar que no dispone de características clonables, ni la posibilidad de referencias entre características y que no dispone de atributos.

#### 3.1.3. MFM

El Centro de Investigación en Métodos de Desarrollo de Software (ProS) ha desarrollado MOSKitt Feature Modeler (MFM<sup>1</sup>), una herramienta Open-Source basada en Eclipse y desarrollada utilizando EMF, GMF y ATL. Actualmente el proyecto está en funcionamiento.

Modeling Software KIT (MOSKitt) es una herramienta CASE libre, basada en Eclipse que está siendo desarrollada por la Conselleria de Infraestructuras y Transporte (CIT) de Valencia para dar soporte a la metodología gvMétrica (una adaptación de Métrica III a sus propias necesidades). gvMétrica utiliza técnicas basadas en el lenguaje de modelado UML.

Su arquitectura de plugins la convierte no sólo en una Herramienta CASE sino en toda una Plataforma de Modelado en Software Libre para la construcción de este tipo de herramientas.

MOSKitt se desarrolla en el marco del proyecto gvCASE, uno de los proyectos integrados en gvPontis, el proyecto global de la CIT para la migración de todo su entorno

---

<sup>1</sup><http://oomethod.dsic.upv.es/labs/>



### 3.1. Herramientas para el modelado de características

---

tecnológico a Software Libre. De momento MFM es el único proyecto que utiliza MOSKitt.

Entre sus bondades nos encontramos con que dispone de una interfaz muy fácil de utilizar con diferentes notaciones gráficas disponibles, atributos en las características e interoperabilidad con el plugin FMP de Czarnecki. Entre sus defectos podemos citar la ausencia de características clonables, la imposibilidad de crear configuraciones, inexistencia de referencias entre atributos y restricciones básicas

#### 3.1.4. S2T2

S2T2[11] es el resultado de la investigación de las líneas de productos software por Lero (el centro de investigación de Ingeniería del Software de Irlanda).

Es una aplicación de Java Open-Source independiente que permite configuraciones y comprobación de restricciones básicas. El modelo está basado en una gramática y utiliza como resolutor *SAT*. Sorprendentemente no dispone de un editor gráfico para modelar, únicamente permite realizar configuraciones para un modelo textual previamente dado. No dispone de características clonables, referencias, atributos o restricciones avanzadas.

#### 3.1.5. Requiline

RequiLine[49] está desarrollada por el grupo de investigación “Software Construction I3” de la Universidad RWTH Aachen de Alemania. El proyecto está actualmente abandonado desde el 2005.

Requiline es una aplicación para Windows que requiere de la conexión con una base de datos. Es gratis de momento y de baja facilidad de uso al menos en sus inicios. El editor gráfico está basado en la notación FORM. Dispone de un validador de restricciones consistente (aunque no permite restricciones avanzadas), gestor de usuarios con diferentes vistas y un interfaz XML. Las características disponen de atributos y se pueden realizar configuraciones entre otras propiedades.

En resumen, una herramienta muy completa en algunos aspectos que ha dejado de lado algunas características realmente importantes, como la usabilidad o algunos aspectos relacionados con el modelado de características.

#### 3.1.6. fmp

FMP[2] (Feature Modeling Plug-in) está desarrollado por la Universidad de Waterloo, Canadá, concretamente por K. Czarnecki, uno de los autores que más artículos ha escrito acerca del modelado de características, así como un referente para la mayoría de las herramientas citadas. FMP que es Open-Source permite ser utilizado en Eclipse como

plug-in realizado en EMF o en el Rational Software Modeler (RSM) o Rational Software Architect (RSA) mediante el plug-in fmp2rsm.

El proyecto está oficialmente finalizado en el 2006, aunque hace poco se lanzó una nueva versión que mejoraba el editor de restricciones. Utiliza como resolutor BDD. Posee muchas características que funcionan como la creación de configuraciones, atributos, restricciones avanzadas... Pero también posee otras que no funcionan correctamente como es el caso de las características clonables o las referencias entre características.

La herramienta peca de ser demasiado pretenciosa, puesto que se han implementado a medias muchas funciones, a pesar de eso es una de las herramientas más comúnmente utilizadas para el modelado de características.

#### 3.1.7. SPLOT

El objetivo principal de SPLOT<sup>2</sup> es poner la investigación de las líneas de productos software en práctica a través una herramienta online enfocada a usos académicos y a maestros del área. Está desarrollada por la Universidad de Waterloo de Canadá.

SPLOT es una herramienta web, dispone de una base de datos con gran cantidad de modelos de características base. El usuario puede subir su propio modelo de características que debe de ser escrito en XSMML. Además el usuario puede crear una configuración a partir de un modelo que esté en esa base de datos y validarla.

Es una herramienta muy simple y minimalista, por lo que no disponemos de características clonables, referencias, atributos, restricciones avanzadas o interfaz gráfica. Utiliza SAT como resolutor.

#### 3.1.8. xFeature

La herramienta fue diseñada por Ondrej Rohlik y Alessandro Pasetti. El desarrollo inicial fue hecho bajo un contrato de la ESA con P&P Software GmbH y el Automatic Control Laboratory del ETH-Zürich. xFeature<sup>3</sup> sigue extendiéndose (aunque la última versión de la herramienta data del 2005) y siendo utilizada por el ETH-Zürich en contexto del proyecto ASSERT.

Es una herramienta Open-Source muy completa, presentada como plug-in de Eclipse, que presenta características clonables, referencias, atributos y una versión no completa de restricciones avanzadas. Como puntos innovadores tiene el uso de tecnología estándar (XML y Eclipse) [13].

---

<sup>2</sup><http://www.splot-research.org/>

<sup>3</sup><http://www.pnp-software.com/XFeature/Home.html>

### 3.1. Herramientas para el modelado de características

Nombre	Facilidad de uso	Plataforma	GUI	Características clonables	Configuraciones	Referencias	Restricciones avanzadas	Licencia
FaMa	Media	Librería en Java	Si	-	-	-	-	Open-Source
FeatureIDE	Media	Plugin Eclipse	Si	-	Si	-	*Si	Open-Source
MFM	Alta	Plugin Eclipse	Si	-	-	-	-	Open-Source
S2T2	Media	Java Application	-	-	Si	-	-	Open-Source
Requiline	Baja	Win Application	Si	-	Si	-	-	Free at the moment
fmp	Media	Plugin Eclipse	Si	Si	Si	-	Si, Or->Or	Open-Source
SPLIT	Alta	Web	-	-	Si	-	-	-
xFeature	Baja	Plugin Eclipse	Si	Si	Si	Si	Si, Or->Or	Open-Source
Pure::Variants	Media	Plugin Eclipse	Si	-	Si	-	Si	Copyright
Hydra	Máxima	Plugin Eclipse	Si	Si	Si	Si	Si	Open-Source
Captain Feature	Media	Java Application	Si	Si	Si	Si	Si	GNU

Figura 3.1: Resumen herramientas analizadas.

#### 3.1.9. pure::Variants

pure::Variants[8] es una herramienta de pago creada por pure:systems desarrollada como plug-in de Eclipse. Permite perfilar y gestionar eficientemente todas las partes de los productos software con sus componentes, restricciones y términos de uso.

En su versión de prueba hemos podido comprobar sus características realmente, y a pesar de ser de pago, no es una herramienta totalmente expresiva en lo que a modelos de características se refiere, puesto que carece de características clonables, referencias o atributos. En cambio tiene otras propiedades como la generación de código y la interoperabilidad con SAP o MatLab.

#### 3.1.10. CaptainFeature

Herramienta bajo licencia GNU que se presenta como una aplicación Java muy completa. Dispone de características clonables, referencias, atributos, restricciones avanzadas... Pero ha tenido muy poca promoción y pocos saben de su existencia a pesar de ser tan expresivo.

El proyecto está acogido por Sourceforge<sup>4</sup> y está oficialmente finalizado desde el 2005.

Uno de sus puntos débiles es la interfaz gráfica y su usabilidad, además de algunos problemas derivados, como que el proyecto únicamente compila con la versión 1.4 del JDK de Java.

#### 3.1.11. Sumario herramientas de modelado

Anteriormente se han analizado las herramientas existentes relacionadas con los modelos de características. Hemos realizado una tabla comparativa (ver Figura 3.1) con algunos de los aspectos más importantes analizados.

<sup>4</sup><https://sourceforge.net/projects/captainfeature/>

De todas las herramientas analizadas, sólo tres implementan características clonables, una de ellas (fmp) tiene un comportamiento imprevisible reconocido públicamente por su creador. De esas tres, dos de ellas implementan restricciones avanzadas pero con un comportamiento que no siempre es el que queremos (para mayor profundidad, dirigirse al capítulo 6.2). Y todas ellas tienen una usabilidad en mayor o menor medida baja.

En la tabla comparativa podemos ver otros datos, todos previamente descritos individualmente para cada herramienta.

Pasemos ahora a estudiar tres de las diferentes propuestas existentes para el análisis automatizado de los modelos de características con el fin de analizar ciertos cometidos expuestos previamente en el capítulo 2.6.

## 3.2. Análisis automatizado de los modelos de características

Existe una gran variedad de técnicas y herramientas que pueden ser utilizadas para el análisis automatizado de los modelos de características. En las siguientes subsecciones iremos analizando algunas de las más utilizadas, analizando su forma de resolver problemas, el nivel de adaptación a nuestro problema y una comparación entre todas.

### 3.2.1. CSP

El problema de la satisfacción de restricciones o CSP[48] (Constraint Satisfaction Problem) es un problema matemático definido como un conjunto de objetos que deben de satisfacer un número de restricciones o limitaciones. CSP representa a las entidades de un problema como una colección homogénea de restricciones finitas sobre variables, las cuales son resueltas por métodos de satisfacción de restricciones. CSP se utiliza intensamente en investigación tanto en la inteligencia artificial, investigación operativa, bases de datos o sistemas de recuperación.

Formalmente, el problema de satisfacción de restricciones se define como la terna  $\langle X, D, C \rangle$ , donde:

- $X$  es un conjunto de  $n$  variables  $x_1, \dots, x_n$ .
- $D = \langle D_1, \dots, D_n \rangle$  es un tupla de dominios finitos donde se interpretan las variables  $X$ , tal que la  $i$ -ésima componente  $D_i$  es el dominio que contiene los posibles valores que pueden asignarse a la variable  $x_i$ . La cardinalidad de cada dominio es  $d_i = |D_i|$ .
- $C = \{c_1, c_2, \dots, c_p\}$  es un conjunto finito de restricciones. Cada restricción  $k$ -aria  $c_i$  está definida sobre un conjunto de  $k$  variables  $var(c_i) \subseteq X$ , denominado su ámbito,

### 3.2. Análisis automatizado de los modelos de características

---

y restringe los valores que dichas variables pueden simultáneamente tomar. Todas las restricciones definidas en un CSP son conjuntivas, de manera que una solución debe de satisfacer a todas ellas.

- Una solución a un CSP es una asignación  $(a_1, a_2, \dots, a_n)$  de valores a todas sus variables, de tal manera que se satisfagan todas las restricciones del CSP. Es decir, una solución es una tupla consistente que contiene todas las variables del problema.

La programación de restricciones es la propuesta más flexible. Puede ser usada para desempeñar la mayoría de las operaciones actualmente identificadas en los modelos de características. Sin embargo, la programación de restricciones revela cierta debilidad cuando calculan ciertas operaciones en modelos de características de medio y gran tamaño como por ejemplo al calcular el número de posibles combinaciones de características.

#### Choco

Choco es una librería Java para el problema de satisfacción de restricciones (CSP) y programación de restricciones[27] (CP) desarrollada por el École des Mines de Nantes, Francia. Está construido con mecanismos de propagación basados en eventos con estructuras de backtracking.

Seleccionamos este resolutor porque parece ser uno de los más populares dentro de la comunidad investigadora y porque es el que mayor flexibilidad dispone a la hora de expresar restricciones.

#### 3.2.2. SAT

Una fórmula proposicional es una expresión consistente en un conjunto de variables booleanas (literales) conectados por operadores lógicos ( $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ ). El problema de satisfacibilidad proposicional o SAT[52] consiste en decidir cuando una fórmula proposicional dada es satisfacible, por ejemplo, unos valores lógicos pueden ser asignados a sus variables de una forma que haga la fórmula verdadera.

Los resultados de rendimiento para los resolutores SAT son ligeramente mejores en resultados que CSP, la expresividad que obtenemos con CSP es mucho mayor, por lo que podemos definir las restricciones de manera muy simple.

#### 3.2.3. BDD

Un diagrama de decisión binario o BDD[1] (Binary Decision Diagram) es una estructura de datos usada para representar una función booleana. Un BDD es un grafo acíclico con

una raíz, dirigido y compuesto por un grupo de nodos de decisión y dos nodos terminales llamados terminal-0 y terminal-1. Cada nodo en el grafo representa una variable en una función booleana, y tiene dos nodos hijos que representan la asignación de la variable a 0 ó 1. Todos los caminos desde la raíz al terminal-1 representan las asignaciones de variables para los que la función booleana es verdad en cambio todos los caminos al terminal-0 representan las asignaciones de variables para los que la función booleana devuelve falso.

Aunque el tamaño de los BDD pueden ser reducidos de acuerdo a algunas reglas establecidas, la debilidad de este tipo de representación es el tamaño de la estructura de datos la cual puede variar entre rango lineal o exponencial dependiendo del orden de la variable. Calcular el mejor ordenamiento de variable es un problema NP-duro. Sin embargo el problema de memoria es compensado por los tiempos resultantes. Mientras que CSP y SAT son incapaces de encontrar el número total de soluciones para modelos de características de tamaño medio o grande en un tiempo razonable, BDD puede encontrarlo en un tiempo insignificante.

#### 3.2.4. OCL

OCL2.0[41] (Object Constraint Language 2.0) fue adoptado en octubre de 2003 por el grupo OMG como parte de UML 2.0. OCL es un lenguaje para la descripción formal de expresiones en los modelos UML. Sus expresiones pueden representar invariantes, precondiciones, postcondiciones, inicializaciones, guardias, reglas de derivación, así como consultas a objetos para determinar sus condiciones de estado. Se trata de un lenguaje sin efectos de borde, de manera que la verificación de una condición, que se presupone una operación instantánea, nunca altera los objetos del modelo. Su papel principal es el de completar los diferentes artefactos de la notación UML con requerimientos formalmente expresados.

OCL está perfectamente integrado con EMF, por lo que lo hemos utilizado para realizar comprobaciones básicas sobre el diagrama, como por ejemplo: restricciones para la correcta creación estructural del diagrama, comprobación de identificadores únicos (los nombres de las características), correcta escritura de las cardinalidades de las características si procede...

#### 3.2.5. Sumario del Análisis Automatizado

En [6] se realizan algunas pruebas para realizar distintas pruebas para medir el rendimiento, la memoria ocupada... y comparando tres tipos de resolutores, BDD, CSP y SAT. En ellos se puede observar que BDD puede llegar a ocupar hasta 10 veces más de memoria o que CSP es quien más tiempo tarda. No obstante todos tienen sus pros y contras, BDD

### 3.2. Análisis automatizado de los modelos de características

---

consume muy poco tiempo y CSP es muy flexible.

Otros autores como Nakajima[40] utilizan como motor de resolución automático Alloy. Mannion [39] muestra como utilizar Z/EVES y Alloy para el razonamiento. D. Batory [3] utiliza LTMS (Logic-Truth Maintenance Systems) para la búsqueda de configuraciones válidas.





# Capítulo 4

## Desarrollo de un editor gráfico para modelos de características

Habiendo estudiado los antecedentes y el estado del arte vamos a empezar a explicar el desarrollo de Hydra y más concretamente de sus componentes, explicando inicialmente el desarrollo del editor gráfico de modelos de características.

### 4.1. Introducción

Lo más esencial en una herramienta relacionada con los modelos de características es una funcionalidad que permita el modelado. Existen muchas formas de abordar el problema, se podría haber hecho un editor textual para definir modelos de características que tengan estructura de gramática o un editor semigráfico estilo XML (como los editores EMF). Pero nosotros hemos optado por crear una interfaz lo más cómoda posible para el usuario, por ello hemos elegido GMF.

El editor gráfico deberá permitir crear diagramas intuitivamente, permitir la rápida edición y creación de elementos y crear modelos lo más compacto posible. Se podrán editar los atributos y el modelo seguirá siendo consistente

### 4.2. Metamodelo

Inicialmente partimos del metamodelo del plugin fmp de Czarnecki, pero debido a restricciones de GMF y conceptos del metamodelo con los que no estábamos de acuerdo (en éste, las características poseen cardinalidad, cuando la cardinalidad la debe de poseer la relación que va a dicha característica, pero este no contempla las relaciones entre características), se tuvo que modificar el metamodelo

El metamodelo del editor de modelos de características está representado en la figura 4.1, en él podemos observar que un proyecto se compone básicamente de nodos y de relaciones. Un nodo puede ser un *Feature* (correspondiente a el concepto de característica) o un *FeatureGroup* (nodo que agrupa features y que deben de respetar cierta condición). Las relaciones pueden ser de dos tipos también, *RelationFeature* y *RelationFG*, el primero relaciona features y el segundo relaciona features con *FeaturesGroup*. La principal diferencia entre ambos es que el primero posee cardinalidad mientras que el segundo no. Existe en el metamodelo tipos de usuario enumerados, como *FeatureGroupType*, *FeatureType* o *ValueType*, los dos primeros son derivados de la cardinalidad del *FeatureGroup* y del *RelationFeature* respectivamente, el tercero está puesto con vistas al futuro desarrollo de atributos en las características. Relacionado con los atributos podemos ver que un feature está compuesto de cero o un *TypedValue*, la idea es que el usuario introduzca un valor y defina su tipo de entre los existentes en los definidos por *ValueType*. Los atributos que posee *Project* son en su mayoría internos para el correcto funcionamiento de la herramienta, excepto *numberOfProducts* que permite ver en las propiedades del proyecto el número de productos que genera el modelo.

### 4.3. Editor gráfico

Ya hemos definido la parte abstracta de nuestro editor gráfico con el metamodelo, ahora describiremos la parte gráfica, para ello existen en GMF el modelo de definición gráfica y el modelo de definición de herramientas como se explicó anteriormente en los antecedentes.

El modelo de definición gráfico es simple, para ello se han definido cuatro figuras (ver Figura 4.2), una figura rectangular con una etiqueta de texto que representará visualmente a la característica, una figura que está compuesta por una imagen SVG con una etiqueta que representará al nodo de grupo características, una línea con una imagen en el extremo destino con una etiqueta que simboliza los enlaces entre características y un enlace solitario que se referirá a los enlaces que vayan hacia o desde un nodo grupo de características.

Por la parte del modelo de definición de herramientas definimos acciones para la creación de estos elementos (características, nodos de grupo de características, enlaces entre características, y enlaces hacia o desde nodos de grupo).

Una vez creados el modelo del dominio, el modelo de definición gráfica y el modelo de definición de herramientas, los enlazamos todos en el modelo de asignación (por ejemplo, a la metaclassa *Feature* le asignamos la figura rectangular con etiqueta de texto y además le asignamos la herramienta de creación de características).

Tras la transformación modelo a modelo, ajustar algunos parámetros específicos y

### 4.3. Editor gráfico

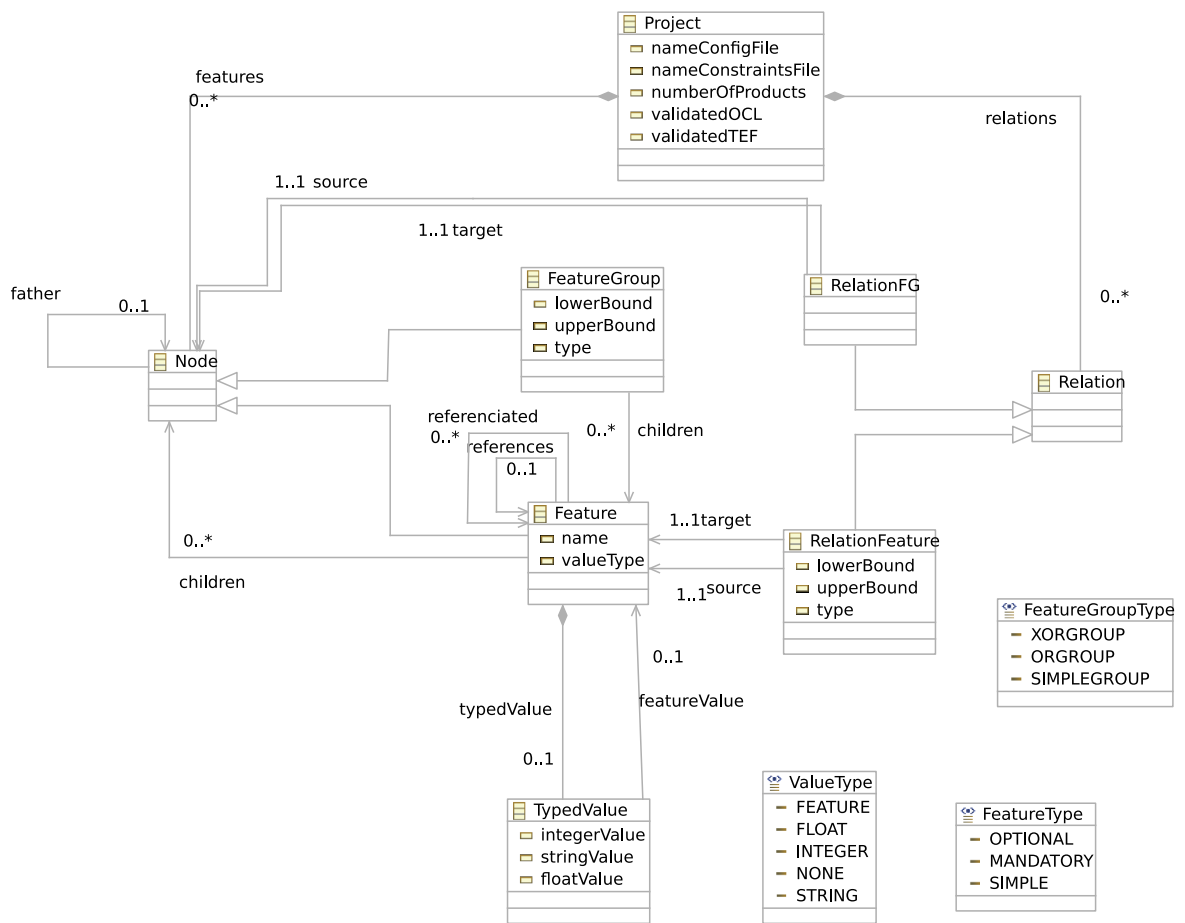


Figura 4.1: Metamodelo del editor para modelos de características

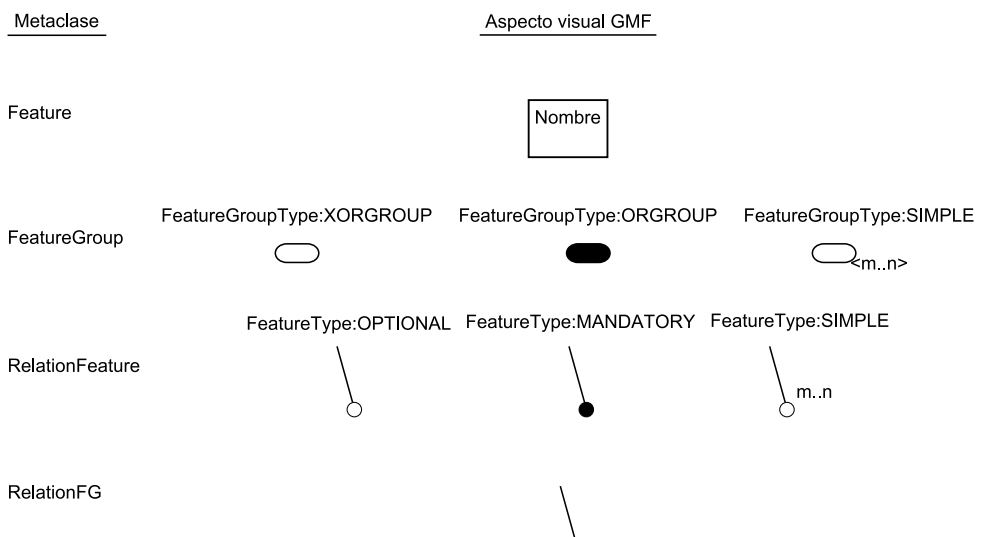


Figura 4.2: Correspondencia entre metaelementos Ecore y notaciones gráficas GMF.

transformar este último a código, obtenemos la primera versión de nuestro editor gráfico. Con éste, podemos crear modelos de características. Pero el editor aún es demasiado

flexible, permite al usuario a crear diagramas sin sentido, grafos, cardinalidades y nombres incorrectos, no existe consistencia entre los datos (si el límite inferior de una cardinalidad es cero y el límite superior de la misma es uno, el tipo del enlace que deriva de estos atributos, debe de ser opcional) ni entre los datos y su representación gráfica (si el enlace es opcional, el decorador del extremo final debe de ser un círculo vacío y no se debe mostrar la etiqueta de las cardinalidades)...

La mayoría de estos requisitos que son imprescindibles para que el editor cree modelos de características correctos no se pueden definir de momento por los modelos disponibles, por lo que tendremos que solucionar cada uno de los problemas a bajo nivel.

Existen diferentes tipos de problemas que abordaremos de forma diferente:

**Problemas básicos** OCL está perfectamente integrado con GMF, pudiendo definir restricciones OCL en el modelo de asignación. Con OCL podemos dar valores por defecto en la creación de los elementos y validar pequeños detalles de nuestro modelo como restricciones respecto al nombre de las características (no puede haber nombres repetidos, no puede haber características sin nombre...).

**Consistencia en atributos** Cuando cambiamos valores de algunos atributos, se debe de mantener cierta consistencia respecto a otros atributos y al aspecto gráfico. Para el primero utilizaremos la API de EMF. Para el segundo usaremos la de GMF.

**Permutación práctica de tipos en un elemento** Existen dos objetos que tienen diferentes representaciones gráficas según su tipo. Éstos son las relaciones entre características y los nodos grupos de características. Para ello se han instalado listeners en estos objetos que esperan al evento de la doble pulsación de ratón. En ese instante entrará en acción un comando de GMF que permitirá el cambio de decorador en el caso de la relación, y el cambio de la imagen en el nodo.

En la figura 4.3 podemos ver un modelo de características construido por nuestro editor gráfico.

### 4.4. Ejemplo práctico: Caso de uso de un SmartHome

Durante la descripción del desarrollo de los diferentes componentes de Hydra, vamos a desarrollar un modelo de características de un caso de uso de un SmartHome proporcionado por Siemens AG.

En la figura 4.4 podemos ver una imagen del diagrama realizada y exportada con Hydra. Es el mismo descrito en el capítulo 2.1.1.

## 4.5. Sumario

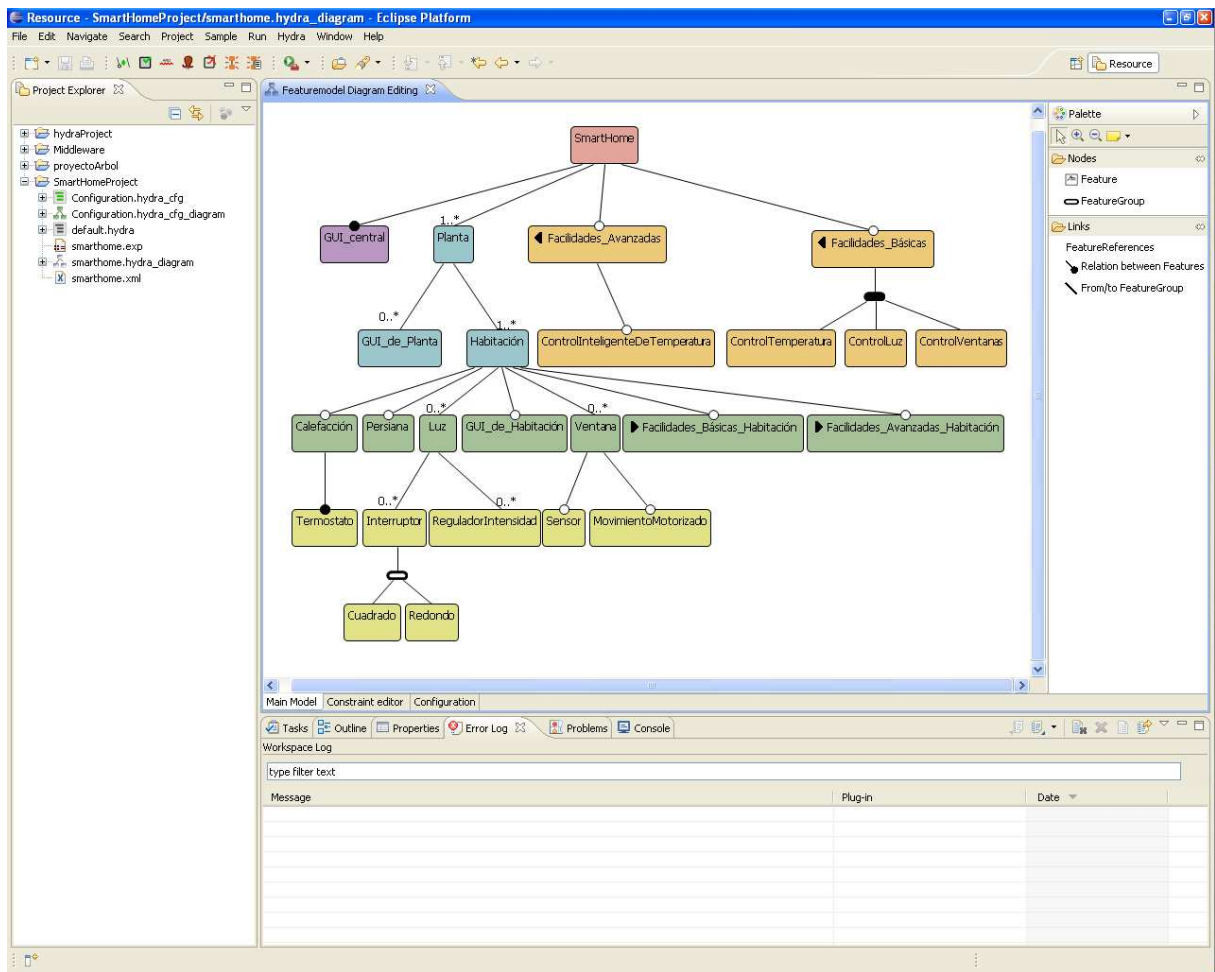


Figura 4.3: Imagen del editor gráfico de modelos de características.

## 4.5. Sumario

En este capítulo hemos descrito cómo se ha desarrollado el editor gráfico de características con GMF, así como los detalles más relevantes referentes a su diseño. El editor gráfico final posee muchos pequeños detalles implementados a bajo nivel para permitir crear únicamente modelos válidos y crearlos cómodamente que no se pueden definir en los modelos de GMF. Aún así, GMF es un buen punto de partida para la creación de editores gráficos estilo UML.

#### 4. Desarrollo de un editor gráfico para modelos de características

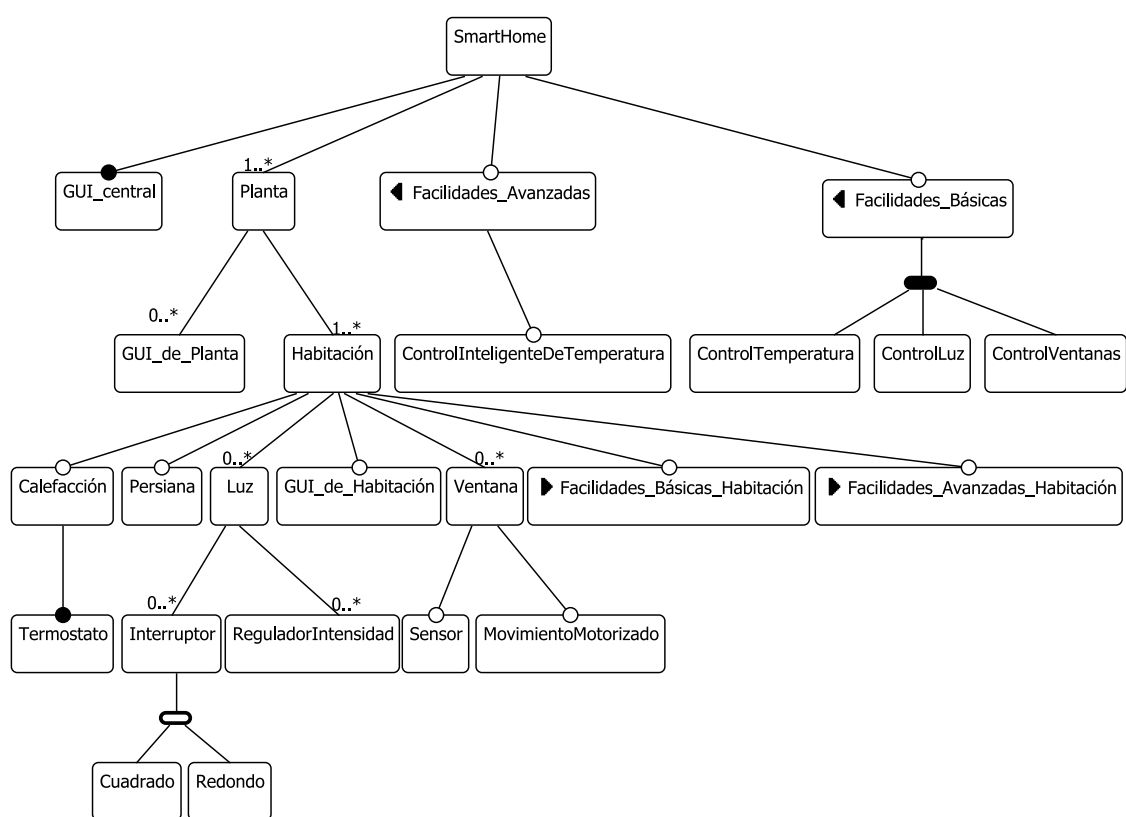


Figura 4.4: Modelo de características de un caso de uso de un SmartHome.

# Capítulo 5

## Desarrollo de un editor textual para restricciones

Este capítulo describe las características principales del editor textual para la edición de restricciones de usuario realizadas como componente de Hydra.

### 5.1. Introducción

Mediante las restricciones implícitas del diagrama no se puede llegar a expresar los modelos que el usuario puede desear, necesitamos de unas restricciones adicionales flexibles y lo suficientemente expresivas para que el usuario pueda crear configuraciones que tengan sentido para él.

En la mayoría de las herramientas analizadas en el estado del arte pudimos observar que únicamente poseen restricciones básicas del tipo implicación y exclusión entre características. Nosotros debido al problema semántico de estas dos restricciones sobre características clonables explicadas a fondo en el capítulo 6 y queriendo añadir expresividad, hemos añadido más operadores con el que se podrán crear todo tipo de restricciones de la forma que sea más fácil para el usuario.

### 5.2. Metamodelo

El elemento más importante del metamodelo es Expression, que representa el concepto de restricción de usuario, por tanto el modelo estará compuesto de cero o más restricciones. Una expresión puede ser:

- **Feature** Una característica. Si escribimos una característica, eso quiere decir que ésta debe de ser seleccionada en la especialización.

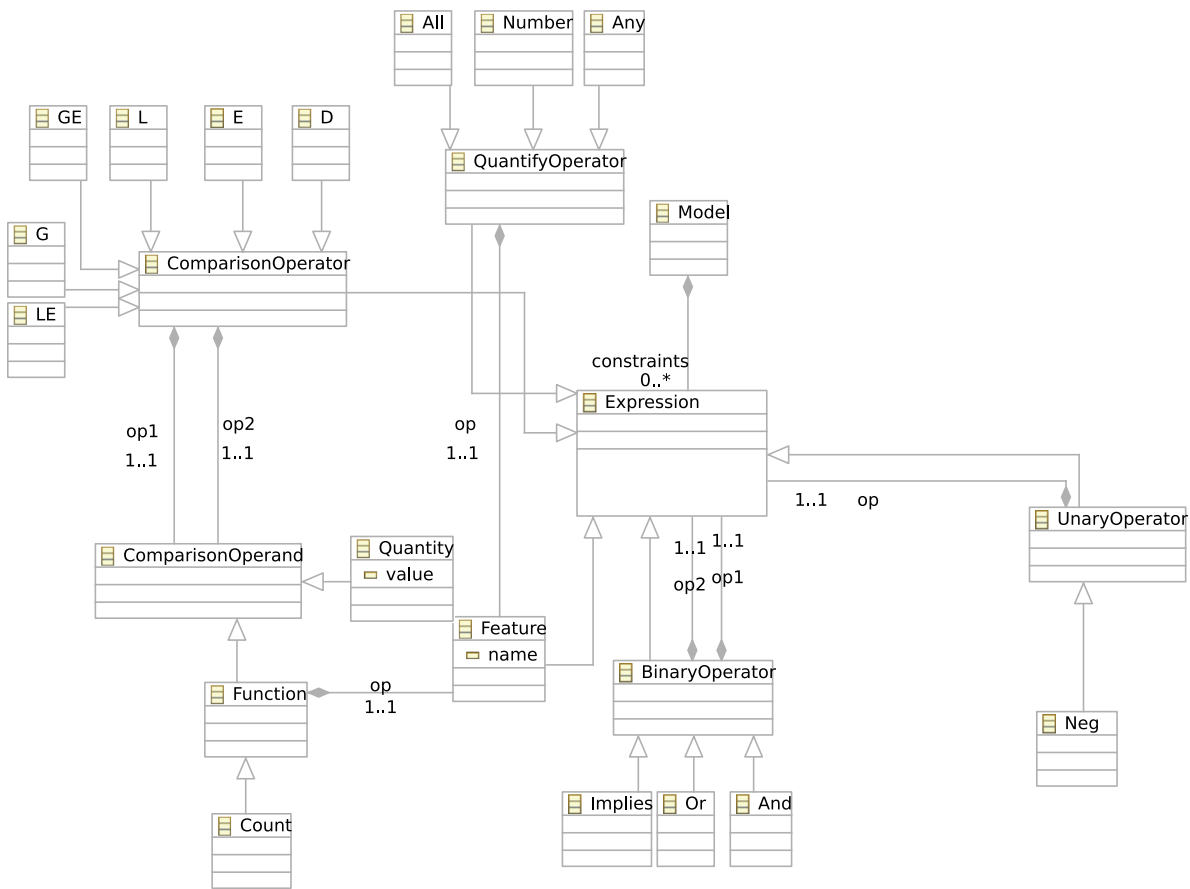


Figura 5.1: Metamodelo del editor de restricciones

- **UnaryOperator** Un operador unario hacia una expresión. Actualmente sólo se puede realizar el operador unario de la negación (*Neg*).
- **BinaryOperator** Operador binario de dos características. Las operaciones binarias existentes son la implicación (*Implies*) y and y or lógicos (*And*, *Or*).
- **QuantifyOperator** Con este tipo de operadores que se aplican a los *Features* podemos filtrar el número de características que serán seleccionadas. Si se aplican a una característica no clonable, estos operadores se comportan de similar forma y carecen de sentido, en cambio para características que aparezcan más de una vez en el diagrama (ya sea por ser una característica con cardinalidad, o ser descendiente de una característica de este tipo) el operador adquiere su sentido y expresividad. Para ello es necesario obtener todas las características del tipo deseado. Por ejemplo: un operador de este tipo afectando a la característica *Arbol* obliga a todas las características *Arbol* a ser seleccionadas, incluso a las características *Arbol* renombradas por el usuario en el editor de configuraciones. Existen tres operadores de este tipo:



## 5.2. Metamodelo

- **All** Este operador obliga a todas las características afectadas ser seleccionadas.
  - **Any** Con este operador obligamos a que exista al menos una característica de las características afectadas.
  - **Number** *Number* en realidad es un número entero, con él podemos obligar a que exclusivamente haya en nuestra configuración el número dado de características a las que se hace referencia en esta restricción.
- **ComparisonOperator** Con este tipo de operador, podemos realizar las comparaciones típicas sobre operandos especiales (*ComparisonOperand*). Entre ellas encontramos las operaciones “menor que” (*L*), “mayor que” (*G*), “menor igual que” (*LE*), “mayor igual que” (*GE*), “igual a” (*E*) y “distinto a” (*D*). Los operandos a usar pueden ser o cuantificadores (*Quantify*) que realmente son números enteros, o una función que devuelva un número entero. De momento sólo se ha definido la función que pasando por parámetro un *Feature*, devuelve el número de características que existe en el modelo (*Count*).

Una vez definido el metamodelo tenemos que definir la gramática ETSL. Ésta se define utilizando los elementos del metamodelo.

```

syntax(Model) "resources/expressions.ecore" {
  Model:element(Model) \rightarrow ConstraintList;
  ConstraintList \rightarrow (Constraint)*;

  Constraint \rightarrow Expression:composite(constraints) ws(empty) ";"
ws(statement);

  Expression \rightarrow Implies;
  Expression \rightarrow Term;

  Term \rightarrow And;
  Term \rightarrow Or;
  Term \rightarrow Factor;

  Factor \rightarrow AbstractFeature;
  Factor \rightarrow ComparisonOperation;
  Factor \rightarrow "(" ws(empty) Expression ws(empty) ";";
  Factor \rightarrow Neg;

  Implies:element(Implies) \rightarrow Expression:composite(op1) ws(space)
"implies"
ws(space) Term:composite(op2);
  And:element(And) \rightarrow Term:composite(op1) ws(space) "and"
ws(space) Factor:composite(op2);
  Or:element(Or) \rightarrow Term:composite(op1) ws(space) "or"
ws(space) Factor:composite(op2);
  Neg:element(Neg) \rightarrow "not" ws(space) Expression:composite(op);

  AbstractFeature \rightarrow Feature;
  AbstractFeature \rightarrow AllFeature;
  AbstractFeature \rightarrow AnyFeature;
  AbstractFeature \rightarrow NumberFeature;

  AllFeature:element(All) \rightarrow "all" ws(space)
Feature:composite(op);
  AnyFeature:element(Any) \rightarrow "any" ws(space)
Feature:composite(op);
  NumberFeature:element(Number) \rightarrow INTEGER:composite(value)
ws(space) Feature:composite(op);

```

```

ComparisonOperation \rightarrow LEOp;
ComparisonOperation \rightarrow LOp;
ComparisonOperation \rightarrow GOP;
ComparisonOperation \rightarrow GEOp;
ComparisonOperation \rightarrow EOp;
ComparisonOperation \rightarrow DOp;

LEOp: element (LE) \rightarrow ComparisonOperand: composite (op1) ws (space)
"<="
      ws (space) ComparisonOperand: composite (op2);
LOp: element (L) \rightarrow ComparisonOperand: composite (op1) ws (space)
"<"
      ws (space) ComparisonOperand: composite (op2);
GOP: element (G) \rightarrow ComparisonOperand: composite (op1) ws (space)
">"
      ws (space) ComparisonOperand: composite (op2);
GEOp: element (GE) \rightarrow ComparisonOperand: composite (op1) ws (space)
">="
      ws (space) ComparisonOperand: composite (op2);
EOp: element (E) \rightarrow ComparisonOperand: composite (op1) ws (space)
"=="
      ws (space) ComparisonOperand: composite (op2);
DOp: element (D) \rightarrow ComparisonOperand: composite (op1) ws (space)
"! ="
      ws (space) ComparisonOperand: composite (op2);

ComparisonOperand: element (Quantity) \rightarrow
INTEGER: composite (value);
ComparisonOperand: element (Count) \rightarrow "count" ws (space) "("
ws (space)
      Feature: composite (op) ws (space) ")" ";
Feature: element (Feature) \rightarrow IDENTIFIER: composite (name);
}

```

En este archivo definimos la gramática que tendrá nuestro editor. Para asignar a los elementos del metamodelo se utiliza *:element(Elemento)*, para asignar a las relaciones de composición del metamodelo se utiliza *:composite(Elemento)*. *ws(space)* corresponde a un número indeterminado de espacios en blanco. Las palabras entre comillas son las palabras reservadas de nuestro lenguaje. Por tanto las palabras *implies*, *not*, *all*, *any*, “<=”, “>=”, “<”, “>”, “=”, “! =”, *and*, *or*, “(”, “)”, “;” y *count*.

### 5.3. Editor textual

En la segunda pestaña de nuestro editor encontramos el editor de restricciones, en él el usuario definirá las restricciones necesarias para que su modelo de características genere configuraciones deseadas y no anómalas. Para ello existen, como hemos visto anteriormente, gran cantidad de operadores que podemos combinar para tener mayor expresividad. El editor textual ayuda en la medida de lo posible al usuario a escribir sus restricciones.

Al estar definido el lenguaje como una gramática, el editor va analizando lo que el usuario escribe, y cuando puede, muestra un menú contextual con los posibles elementos válidos que se pueden introducir, como podemos observar en la figura 5.2. El editor también subraya en rojo los elementos incorrectos

## 5.4. Ejemplo práctico: Caso de uso de un SmartHome

---

```
MoteLike implies TinyOS;
TinyOS implies nesC;
nesC implies Kernel_nesC;
nesC implies Comm_nesC;
(nesC and Location) implies Loc_nesC;
(FlatTopology or OneHopProtocol) implies not (SinkNode and ClusterHead);
HierarchicalTopology implies Role;
(HighCapacityDevices and Security) implies not SKC;
(Sensors and Security) implies SKC;
```

Static

- implies
- and
- or
- ;

Figura 5.2: Editor de restricciones en acción.

```
ControlTemperatura implies Calefacción;
ControlLuz implies Luz;
ControlVentanas implies Ventana ;
```

Figura 5.3: Restricciones para nuestro caso de uso.

## 5.4. Ejemplo práctico: Caso de uso de un SmartHome

Siguiendo con nuestro ejemplo, definiremos las restricciones pertinentes en nuestro editor de restricciones (Figura 5.3).

Estas restricciones son:

- ControlTemperatura implies Calefacción;
- ControlLuz implies Luz;
- ControlVentanas implies Ventana;

Con el asistente de contenido la escritura de restricciones es muy sencilla y no genera ningún problema.

## 5.5. Sumario

En este capítulo hemos visto cómo se ha definido el editor textual de restricciones de usuario de Hydra. Después de la introducción nos hemos dispuesto a crear el editor, definiendo su metamodelo y una gramática válida para nuestro propósito. Por último hemos añadido unas restricciones a nuestro caso de uso.



# Capítulo 6

## Desarrollo de un editor gráfico para configuraciones

En este capítulo hablaremos del desarrollo del editor gráfico con GMF, concretamente el editor gráfico de configuraciones, con el que el usuario podrá a partir de un modelo inicial (el modelo de características principal u otra configuración/especialización), crear una especialización de acuerdo a las restricciones estructurales implícitas y a las restricciones de usuario explícitas.

### 6.1. Introducción

Hemos hablado en capítulos anteriores del desarrollo de un editor gráfico para crear modelos de características y de un editor textual de restricciones. Por ahora el usuario puede crear un modelo de características y definir las restricciones externas, ahora hace falta de que a partir del modelo de características y las restricciones externas definidas, se puedan crear especializaciones y configuraciones válidas. Para ello se tendrá que crear un editor especial que permita visualmente al usuario seleccionar o eliminar características de su modelo final.

### 6.2. Validación de restricciones con clones

Uno de los problemas que nos encontramos a la hora de validar las restricciones es la problemática de la existencia de clones (características con cardinalidad mayor de uno).

Imaginemos que tenemos una restricción del tipo  $B \text{ implies } C$ , pasemos a analizar todas las posibilidades que nos encontramos, viendo sus representaciones gráficas en 6.1:

1. **B y C únicos.** Es el caso más simple, es evidente su funcionamiento.

2. **B único, C clonable.** ¿A qué se refiere esta implicación? A cuantas Cs implicamos, ¿a todas? ¿al menos una?.
3. **B clonable, C único.** En este caso podríamos decir prácticamente lo mismo, ¿con un único B ya se da la implicación, o necesitamos de más?
4. **B y C clonables.** Con este caso ya rizamos el rizo.
5. **A, B y C clonables.** En los casos anteriores hemos supuesto que A es único, pero, ¿y si fuese clonable? Imaginemos que existen dos A's, por tanto cada A tiene un subárbol idéntico, ¿en la implicación deberían de analizarse conjuntamente o separadamente las características implicadas?, es decir, ¿tienen la misma importancia las B's del primer A que del segundo?

Analizando los cuatro primeros problemas uno puede pensar que hacen falta operadores más específicos. Los operadores *implies* y *excludes* utilizados en las herramientas analizadas en el estado del arte solo sirven para el primer caso en el que las características son únicas.

Lógicamente podemos pensar en los siguientes operadores de implicación válidos únicamente para el caso 2:

- **B impliesOr C.**  $B \rightarrow \bigvee_{i=m}^n C_i$
- **B impliesAnd C.**  $B \rightarrow \bigwedge_{i=m}^n C_i$
- **B implies1 C.**  $B \rightarrow \bigoplus_{i=m}^n C_i$

Si extendiésemos a los demás casos en total tendríamos 32 operadores sólo para la operación de implicación y exclusión. Algo que complica demasiado la tarea de escribir restricciones para el usuario.

En vez de esto, pensamos en una solución más adecuada, potente y flexible: añadir operadores de cuantificación para las características (algo difícil de expresar con la lógica proposicional). Estos operadores se utilizan sobre las características y son:

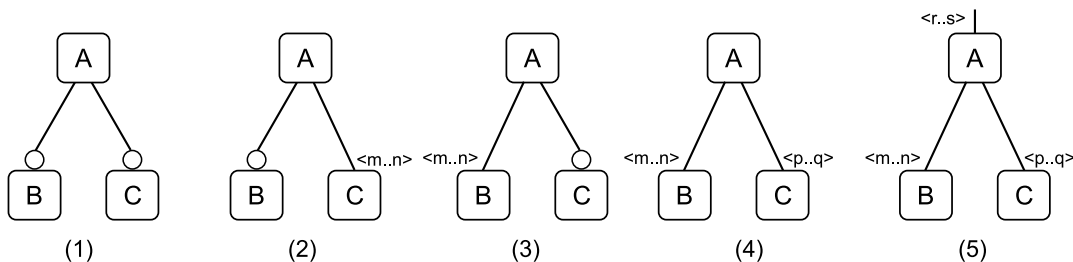


Figura 6.1: Distintos casos para las restricciones

### 6.3. Metamodelo

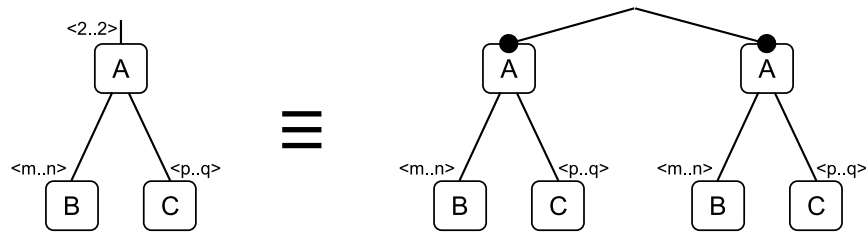


Figura 6.2: Ejemplo del quinto problema descrito

- **all:** Que obliga a que todas las características estén seleccionadas.
- **any:** Que obliga que al menos una característica esté seleccionada.
- **Número entero:** Que obliga a que haya en el modelo la cantidad escrita de características especificadas.

Añadiendo estos 3 operadores tenemos la semántica anterior y más. Por ejemplo,  $B \text{ implies } \bigvee_{i=m}^n C_i$  se traduciría a  $B \text{ implies } (\text{any } C)$  ( $B \text{ implies } (C > 0)$ ). Si no se pone ningún modificador, por defecto entra en acción el modificador *any*.

Pero, ¿qué pasa con el quinto caso? 6.2. Para ello tenemos que definir los ámbitos de actuación de la restricción. En este caso según habíamos dicho anteriormente, tendríamos implícitamente al escribir  $B \text{ implies } C$ ,  $\text{any } B \text{ implies any } C$ . Algo incorrecto, ya que existan características B debajo del primer A no debería de implicar la existencia de características C debajo del segundo A. Por tanto en realidad lo que debería de existir en este caso son dos reglas de implicación, una por cada A. Más genéricamente, al analizar una restricción, tenemos que hayar el subárbol para la que la restricción va a tener efecto, para ello calculamos el padre común mínimo de todas las variables que entran en juego en una restricción. Cada característica que sea padre común mínimo será raíz del subárbol al que aplicar la restricción.

Las herramientas descritas en el estado del arte, al no disponer la mayoría de características clonables no tienen este problema y sólo ofrecen dos operadores, la exclusión y la implicación, y las herramientas que disponen de características clonables sólo tienen nuestro comportamiento por defecto  $\text{any } X \text{ implies any } Y$ .

### 6.3. Metamodelo

Al ser una configuración un modelo de características especializado, su representación tiene que ser la misma que la de un modelo de características común.

El metamodelo del editor de configuraciones es ligeramente diferente al metamodelo del editor de modelos de características (figura 6.3). Difiere principalmente en que

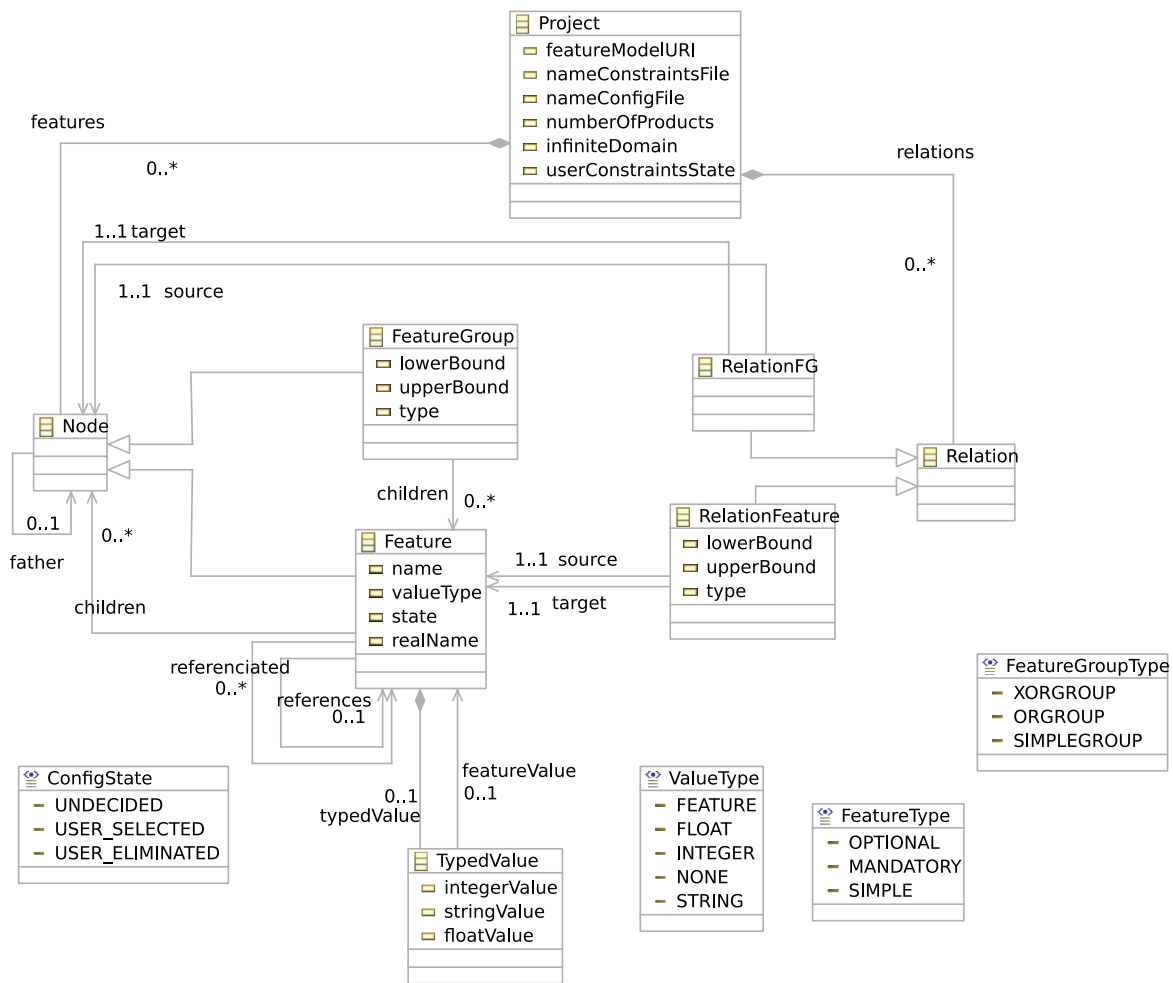


Figura 6.3: Metamodelo del editor de configuraciones

el metamodelo del editor de configuraciones tiene que incorporar el concepto de estado en la característica, para ello se ha creado un atributo en las características de tipo enumerado llamado *ConfigState* que contiene los tres posibles valores: *UNDECIDED*, *USER\_SELECTED* y *USER\_ELIMINATED* para características que aún no tienen asignadas un valor, características seleccionadas, y características eliminadas del modelo respectivamente.

## 6.4. Editor gráfico

Después de haber descrito el metamodelo del editor gráfico de configuraciones, continuaremos con la definición del mismo, para ello definiremos como hicimos en el editor de características los restantes modelos GMF, entre ellos el modelo de definición gráfica y el modelo de definición de herramientas.



## 6.4. Editor gráfico

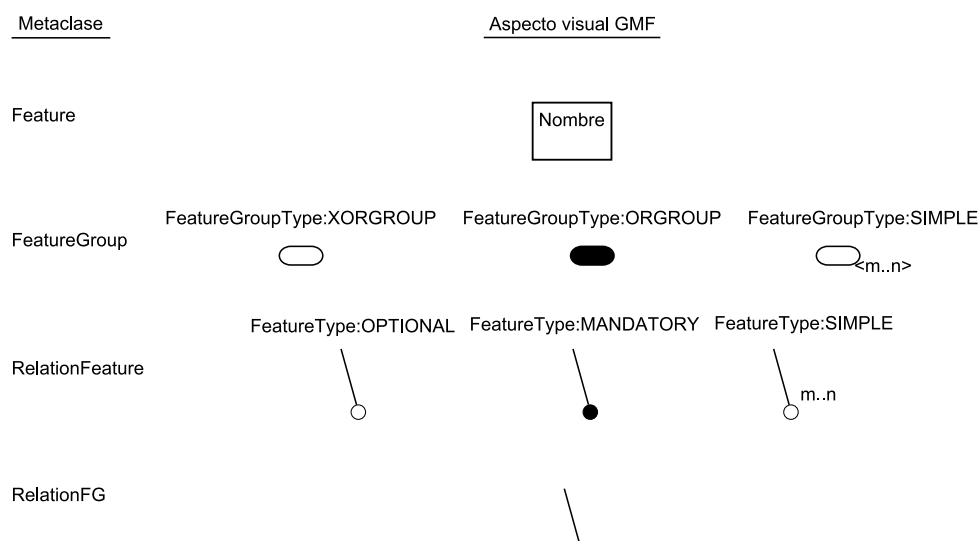


Figura 6.4: Correspondencia entre metaelementos Ecore y notaciones gráficas GMF.

El modelo de definición gráfico es prácticamente igual al definido para la parte de modelado de características, para ello se han definido cuatro figuras 6.4, una figura rectangular con una etiqueta de texto que representará visualmente a la característica, una figura que está compuesta por una imagen SVG con una etiqueta que representará al nodo de grupo características, una línea con una imagen en el extremo destino con una etiqueta que simboliza los enlaces entre características y un enlace solitario que se referirá a los enlaces que vayan a hacia o desde un nodo agrupador de características.

En el modelo de definición de herramientas no definimos ninguna herramienta para la paleta, puesto que en el editor de configuraciones no vamos a permitir al usuario crear ningún elemento, únicamente realizar acciones sobre un diagrama base dado.

En la Figura 6.5 podemos ver el editor gráfico de configuraciones Hydra. Nótese de la ausencia de la paleta de herramientas.

Nos quedaría por realizar los últimos pasos del proceso de desarrollo inicial de GMF. Enlazar todos los modelos anteriores en el modelo de asignación, y tras la transformación modelo a modelo, ajustar algunos parámetros específicos para transformar este último a código y obtener una primera versión de nuestro editor. Pero este editor está aún lejos de comportarse como queremos.

La mejor forma de comprobar que una configuración pertenece a un modelo de características es saber que la configuración se ha creado a partir de éste de forma correcta. Para crear especializaciones en Hydra se parte del modelo de características principal o de otra especialización, de esta forma (y mediante acciones estrictas) las especializaciones creadas derivan del modelo de características inicial, por lo que no habría que comprobar si uno deriva del otro.

## 6. Desarrollo de un editor gráfico para configuraciones

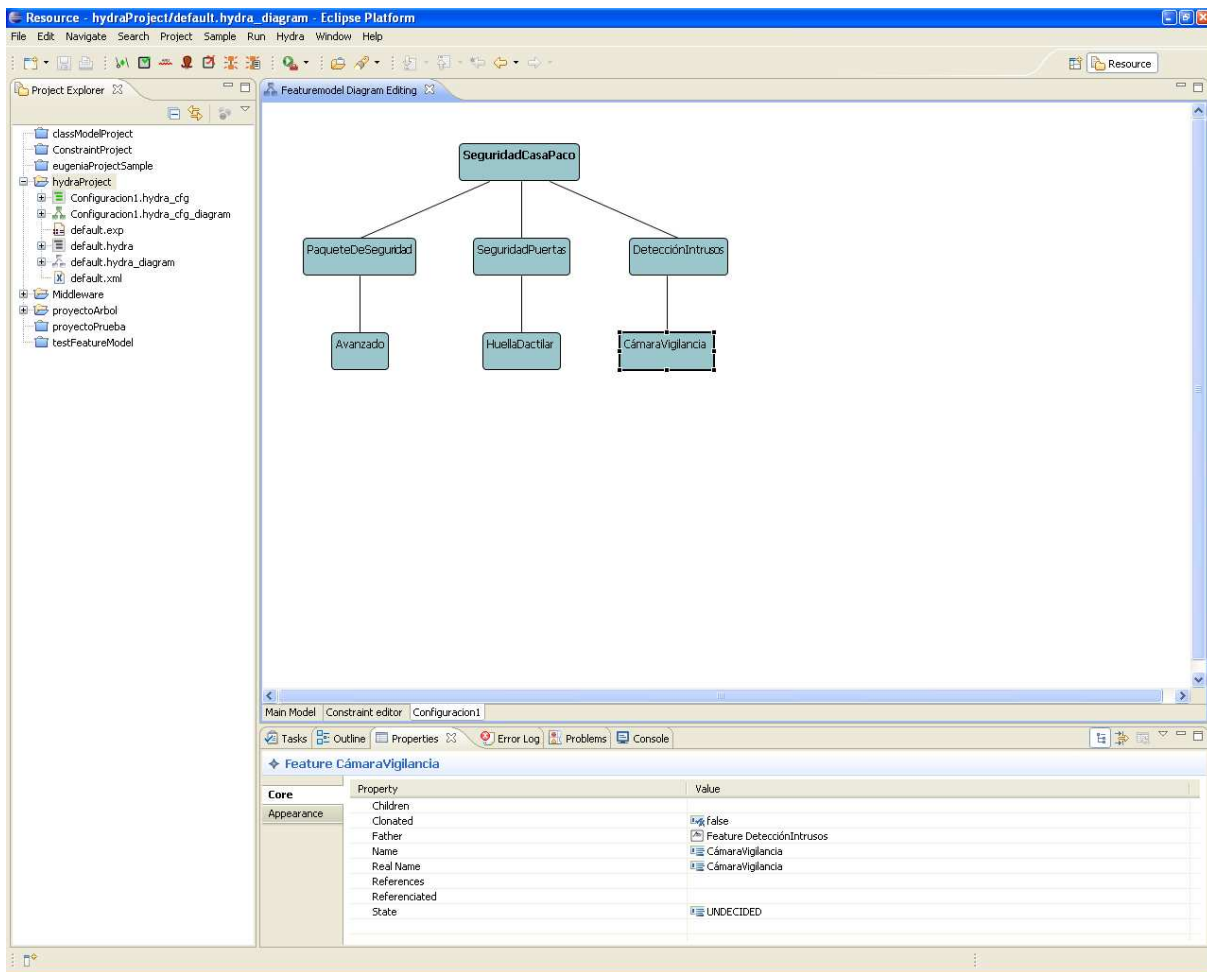


Figura 6.5: Imagen del editor gráfico de configuraciones.

Al igual que el editor de modelos de características, nuestro editor de configuraciones tiene que perfilarse para que no permita modelos erróneos. Aquí no podemos utilizar las mismas acciones que el editor gráfico de modelo de características porque esas acciones estaban más orientadas a la edición. En cambio, en este editor básicamente solo se pueden realizar dos acciones, seleccionar o eliminar una característica para que esté o no en nuestra configuración final

Dependiendo de si nuestra configuración contiene o no alguna característica con cardinalidad superior infinita, nuestro editor tendrá más o menos funcionalidades:

- **Configuraciones sin cardinalidad infinita** Tienen la mayor funcionalidad, podemos obtener del validador toda la información necesaria para que la propagación sea completa así como otros datos, como el número de productos. Especializando un modelo en este modo, nunca llegaremos a un estado incorrecto (a no ser que desde el inicio las restricciones de usuario provoquen este estado), puesto que la propagación impedirá al usuario seleccionar características muertas o eliminar características

## 6.5. Ejemplo práctico: Caso de uso de un SmartHome

---

comunes.

- **Configuraciones con cardinalidad infinita** Disponen de una propagación no completa, es decir, solo propaga el nuevo estado de una característica hacia sus hijos y hacia sus ancestros. La detección de características muertas y características comunes no puede ser realizada por las características de nuestro resolutor de restricciones.

Nuestro resolutor de restricciones es CSP, el cual define un número finito de variables con dominio finito, por lo tanto no podemos introducir infinitas características. En cualquier caso, analicemos una restricción de usuario con una característica de cardinalidad infinita, por ejemplo *a implies all b* siendo *b* con cardinalidad superior infinita. Esta restricción es imposible realizarla en el mundo real, por tanto nos encontramos ante un problema de comportamiento, no podemos modificar la gramática de las restricciones porque si no perderíamos la expresividad de las características clonables y por tanto las características con cardinalidad superior infinita.

Por tanto existen estos dos modos de funcionamiento en nuestro editor de configuraciones. El cambio de modo se realiza automáticamente en cuanto el usuario elimine las características de cardinalidad infinita.

## 6.5. Ejemplo práctico: Caso de uso de un SmartHome

A partir del modelo de características y las restricciones creadas anteriormente, partimos de una especialización en el que hemos seleccionado algunas características y eliminado otras. El resultado se puede ver en la figura 6.6.

Podemos observar que nuestra casa inteligente va una planta baja y una primera planta. En nuestra planta baja tendremos la cocina y el salón mientras que en la primera planta tendremos una habitación. Nótese que cocina y salón son en realidad la misma característica (“Habitación”) pero el editor de configuraciones nos permite ponerles un “apellido”.

## 6.6. Sumario

En este último capítulo hemos explicado el desarrollo del editor gráfico de configuraciones, que es similar al editor gráfico de modelos de características, aunque difiere en muchos detalles importantes. Mientras el primero está más orientado a la edición de modelos de características, el último está orientado a la configuración de los mismos, por ello sólo está permitido la selección y eliminación de características en el último.

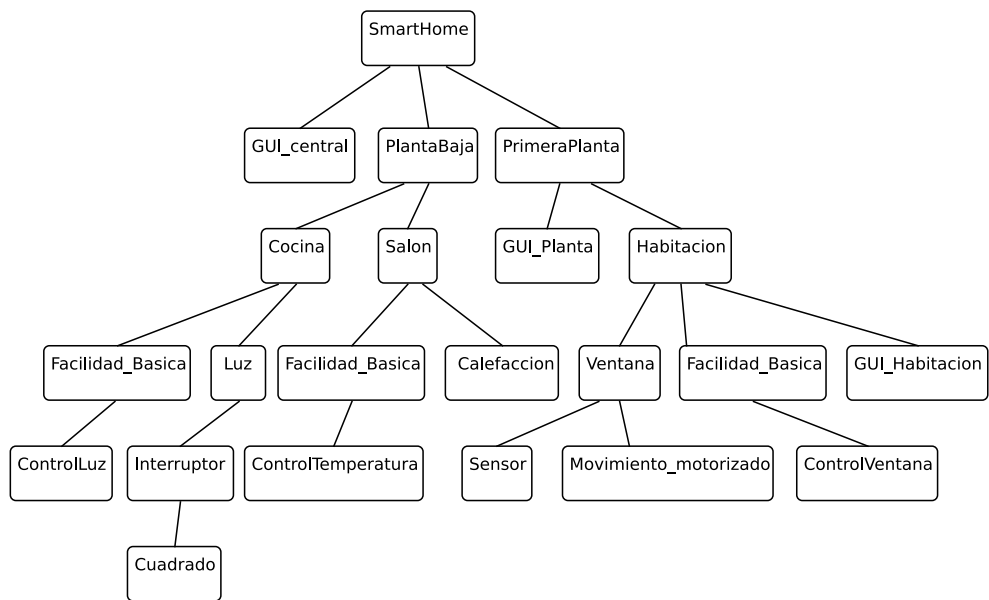


Figura 6.6: Imagen del editor gráfico de configuraciones.

# Capítulo 7

## Validación de restricciones

En este capítulo veremos cómo se ha abordado el problema de la validación de modelos de características y más concretamente de las configuraciones.

### 7.1. Introducción

La última pieza de nuestro puzzle es el validador de restricciones. Primeramente hemos creado un editor gráfico de modelos de características con el que hemos creado nuestro modelo principal. Seguidamente hemos creado un editor textual de restricciones, donde el usuario puede escribir restricciones adicionales permitiendo añadir expresar restricciones que no serían capaces de ser modeladas de manera implícita por los modelos de características. Acto seguido, ha sido creado otro editor gráfico, éste para crear especializaciones a partir del modelo principal.

Pero ahora queda lo más importante, que es saber si una configuración pertenece al modelo principal (esto no es necesario en Hydra, ya que las configuraciones son derivadas inicialmente del modelo principal, y las estrictas acciones mantienen el sistema correcto, por lo que no es posible crear una configuración a partir de un modelo y modificarla de tal manera que no se pueda derivar de su modelo principal) y si es correcto, es decir, si cumple las restricciones implícitas estructurales del diagrama y las restricciones explícitas del usuario. Para ello tendremos que crear un validador que de manera lógica resuelva el problema y si no es posible resolverlo, dar información acerca de los motivos por los que no es posible.

## 7.2. Validación de configuraciones con restricciones usando Choco

Previamente se estudió en los antecedentes cuales eran los sistemas utilizados para esta tarea. Nosotros entre gran cantidad de posibilidades elegimos Choco por su gran adaptación al problema entre otros motivos.

Choco es CSP, por lo que lo primero que tenemos que hacer es adaptar nuestro problema para que pueda ser resuelto mediante esta técnica.

Recordemos la definición de CSP:

CSP es una tupla de tres elementos de la forma  $(V, D, C)$  donde  $V \neq \emptyset$  es un conjunto finito de variables,  $D \neq \emptyset$  es un conjunto finito de dominios (uno para cada variable) y  $C$  son restricciones definidas en  $V$ .

Teniendo en cuenta esto, en nuestro problema el conjunto de variables  $V$  estará compuesto por todas las características (incluso las que están implícitas detrás de las cardinalidades). Cada variable tiene dominio  $[0, 1]$  que corresponde con la semántica de característica no seleccionada y seleccionada respectivamente. Por último tenemos que nuestro conjunto  $C$  estará compuesto por dos subconjuntos de restricciones como se ha hablado durante todo el documento.

Por una parte tenemos las restricciones implícitas que vienen dadas por la construcción del diagrama (figura 7.1). Por otro lado tendríamos las restricciones explícitas dadas por el editor de restricciones y escritas por el usuario.

Por lo tanto, el proceso de validación comienza analizando el diagrama a validar y creando recursivamente las variables y las restricciones implícitas. Seguidamente se analizan las restricciones explícitas (si existen) y se traducen para finalmente ejecutar el problema y obtener los resultados.

El validador creado realiza todas las acciones descritas en el capítulo 2.1.1. Podemos citar la satisfacibilidad, configuración mínima, número de productos, características comunes y características muertas (que se pasan a un gestor de resultados que propaga los resultados obtenidos y ejecutando las acciones pertinentes a cada característica)...

En caso de que el modelo no sea válido se muestra una ventana informativa con las restricciones causantes de dicha insatisfacibilidad.

## 7.3. Ejemplo práctico: Caso de uso de un SmartHome

Para validar nuestro ejemplo práctico, como se ha dicho anteriormente, analizaremos el diagrama para crear las variables y las restricciones implícitas, la traducción a nuestro lenguaje lógico de este tipo de restricciones lo podemos observar en la figura 7.1. Las res-

### 7.3. Ejemplo práctico: Caso de uso de un SmartHome

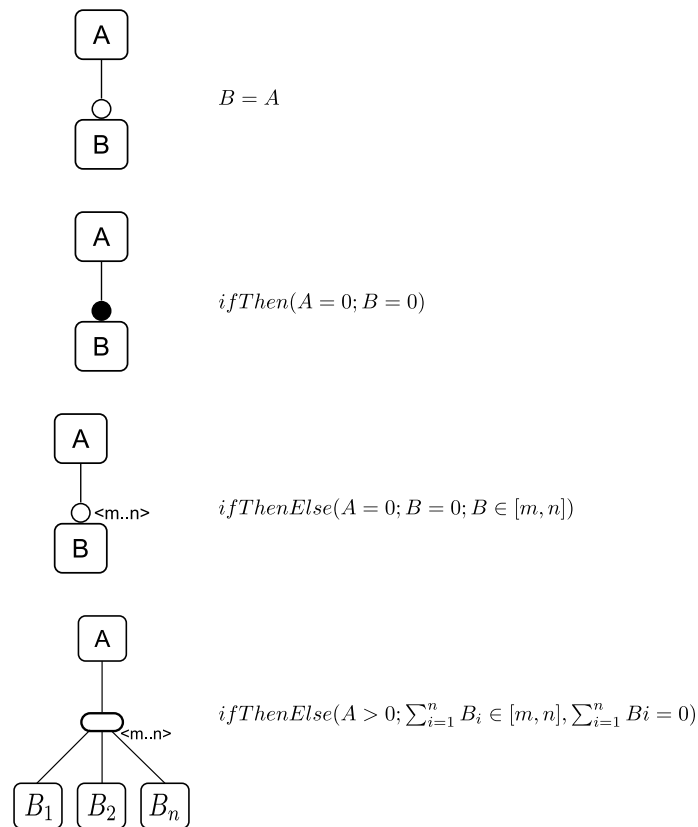


Figura 7.1: Traducción del diagrama a restricción lógica

tricciones explícitas de usuario, como se comentó anteriormente, se analizan por separado. Aquí vamos a analizar la traducción que se realizaría para las restricciones de nuestro caso de uso del SmartHome, recordemos cuáles eran estas restricciones.

- *ControlTemperatura implies Calefacción;*
- *ControlLuz implies Luz;*
- *ControlVentanas implies Ventana;*

Analicemos la primera restricción. Como ya se comentó en la sección “Validación de restricciones con clones” del capítulo 6, necesitamos definir un ámbito para las restricciones con características que a causa directamente o indirectamente, son clonables. Nuestra raíz de los subárboles para las restricciones coincide en este caso con la raíz del diagrama (*SmartHome*). A partir de aquí clasificaremos las características conforme a las raíces de los subárboles (como sólo existe una raíz, sólo obtenemos una restricción).

Todas las características deben de llevar un cuantificador, si no se especifica, por defecto se toma el cuantificador *any*. Por tanto la primera restricción es realmente *any ControlTemperatura implies any Calefacción* que se traduciría en Choco como: *implies(*

*gt( sum( getAllFeaturesNamed( ControlTemperatura)), 0), gt( sum( getAllFeaturesNamed( Calefacción)), 0)).*

Las demás restricciones se traducirían de forma similar, quedando resolver el problema.

### 7.4. Sumario

En este capítulo hemos explicado cómo hemos validado las restricciones en el modelo características usando CSP y concretamente la herramienta Choco.



# Capítulo 8

## Conclusiones y Trabajos Futuros

Este capítulo contiene un resumen del proyecto, un análisis crítico del trabajo realizado, donde se discuten sus bondades y limitaciones, así como una lista de potenciales trabajos futuros.

### 8.1. Conclusiones

#### 8.1.1. Sumario

Esta memoria de Proyecto Fin de Carrera ha presentado a *Hydra*, una herramienta para el modelado, configuración y validación de modelos de características, con soporte completo para el el modelado, configuración y validación de características clonables. Más concretamente, Hydra proporciona:

1. Un editor completamente gráfico y amigable al usuario para la construcción de modelos de características, con soporte para el modelado de características clonables.
2. Un editor textual y una sintaxis propia para la especificación de restricciones entre características, donde estas restricciones pueden incluir características clonables.
3. Un editor gráfico, asistido y amigable al usuario para la creación de configuraciones de modelos de características, con soporte para la configuración de características clonables.
4. Un validador que comprueba que las configuraciones creadas satisfacen las restricciones definidas para el modelo de características, incluso cuando dichas restricciones contienen características clonables.

El Capítulo 1 de este documento presentó brevemente el problema que queremos resolver, el marco general en el cual se encuadra el proyecto y los objetivos a seguir. El Capítulo

2 introdujo brevemente los antecedentes de este proyecto, explicando conceptos básicos necesarios para entender este documento, como el concepto de línea de producto software o modelo de características, así como las diferentes herramientas y tecnologías, tales como Ecore o GMF, usadas para el desarrollo de Hydra. El capítulo 3 estudió el estado del arte actual, analizando diversas herramientas para el modelado de características actualmente existentes, destacando carencias y bondades de cada una de ellas. También se realizó un análisis de las técnicas de validación usadas por dichas herramientas. Los capítulos 4, 5 y 6 describieron el desarrollo del editor gráfico para modelos de características, del editor textual de restricciones externas y el editor gráfico y asistido de configuraciones, respectivamente. Por último, el capítulo 7 explicó cómo se ha implementado el proceso de validación de las configuraciones con respecto a las restricciones definidas por el usuario.

Hydra, a diferencia de las herramientas de modelado actuales, ofrece soporte completo para el modelado, configuración y validación de modelos de características clonables. Más concretamente, es posible no sólo modelar características clonables, algo que hacen herramientas como, por ejemplo, FMP. Hydra además soporta la configuración de características clonables, algo que no ofrecen las herramientas como FMP. De hecho, hasta donde alcanza nuestro conocimiento, Hydra es la única herramienta que soporta de forma fiable y robusta la configuración de características clonables. En Hydra, esta configuración es además gráfica y asistida, proporcionando funcionalidades extras como la autocompletación automática, generación de diagramas en muy poco tiempo, referencias a otras características, actualización automática de cardinalidades tras una clonación, configuración asistida, cálculo automático de la disposición gráfica del subárbol clonado, etc. y todo ello con tecnologías estándares como EMF y GMF.

Hydra es Open Source y se distribuye como plugin de Eclipse, siendo totalmente funcional para todas las plataformas para las que se pueda usar Eclipse. Hydra es además muy intuitivo y simple, además de liviano en la validación.

Habiendo resumido a groso modo el proyecto, vamos a pasar a describir las conclusiones sacadas en este trabajo.

### 8.1.2. Conclusiones

Para el desarrollo de este proyecto se han utilizado herramientas que son estándares *de facto* dentro de la comunidad de modelado, tales como Ecore y GMF. Estas herramientas ofrecen funcionalidades bastante potentes, pero dada su reciente aparición, también presentan ciertas carencias que han de ser aún resueltas.

Por ejemplo, GMF facilita en gran medida el desarrollo de notaciones y editores gráficos para metamodelos definidos en Ecore, mediante la especificación de un empareja-

## 8.1. Conclusiones

---

miento ente elementos del metamodelo y su correspondiente símbolo gráfico. Mediante técnicas generativas, es bastante cómodo y rápido obtener editores simples que ofrezcan características interesantes como zoom, exportación de los modelos visuales a formatos de imágenes tales como JPEG o SVG, disposición automática de los elementos de modelado, coloreado de elementos del diagrama de forma interactiva con el usuario, etc. El desarrollo de diagramas básicos y visualmente atractivos es muy sencillo, y una vez adquirida la necesaria destreza, es posible crear desde cero un editor gráfico con muy poco esfuerzo.

No obstante, la incorporación manual de funcionalidades a editores automáticamente generados con GMF no es en absoluto una tarea trivial. Por ejemplo, la creación o eliminación dinámica de nodos ha sido uno de los mayores problemas encontrados al utilizar GMF. GMF dispone de diferentes estrictos entornos de lectura y escritura, haciendo que un mismo código que realiza una función en un lugar no sea válido en otro. El reto ha sido encontrar “el algoritmo definitivo” que en cualquier parte se comporte de la misma forma. Finalmente se consiguió esta meta a pesar de la multitud de formas encontradas no válidas tanto en la escasa documentación y en su foro de desarrollo.

Tareas como la descrita anteriormente requieren un profundo conocimiento de GMF a bajo nivel, conocimiento que es difícil de adquirir al no existir una documentación estable de GMF, entre otros motivos, porque tampoco parece existir una versión estable de GMF. Por tanto, este conocimiento se ha de adquirir mediante un procedimiento de prueba y error, procedimiento que puede ser laborioso y hasta frustrante en ciertas ocasiones.

Respecto al editor de restricciones con TEF podemos decir que su desarrollo ha sido bastante fácil, a pesar de que no existe ningún tipo de documentación, por lo que se podría haber mejorado (y se puede mejorar) el editor de restricciones.

Choco posee una página web con excelente documentación, a pesar de estar en fase beta. Existió algún problema al crear el validador, causado por un bug ya notificado a los desarrolladores.

### 8.1.3. Discusión

En esta subsección discutiremos acerca de la utilidad de Hydra, si es realmente necesario y otros temas de discusión que nos permitirán conocer de mejor manera a Hydra.

Existen muchas herramientas para el modelado de características cada una de ellas con sus bondades y defectos. La mayoría, tiene la capacidad de crear modelos de características simples o sin cardinalidad.

La cuestión es si es realmente necesario la existencia de características clonables. Realmente para la mayoría de los casos existe un modelo simple equivalente, aunque esto puede producir una explosión del número de características, además de la tarea tediosa de te-

ner que escribir los mismos subárboles un número de veces mayor que uno. Por tanto, es necesario tanto para la legibilidad del diagrama como para la comodidad al dibujarlo. De todas formas, existe un caso donde no existe equivalente directo y es al utilizar cardinalidad superior infinita, es obvio que no podemos dibujar infinitos subárboles, una posibilidad es definir un máximo, pero esto es una limitación molesta. Por ejemplo, en nuestro caso de uso de un SmartHome tenemos que existen de cero a infinitas plantas en nuestra casa. Es evidente que no vamos a tener infinitas plantas, pero estimar un máximo sería limitar nuestro modelo, el cual podríamos necesitar en un futuro para modelar un rascacielos.

Otra característica que posee Hydra es la gran potencia expresiva en las restricciones que posee. La mayoría de las herramientas que disponen de restricciones, éstas son simples, únicamente dejan definir restricciones del tipo implicación y exclusión entre características. Esto es aparentemente suficiente en modelos simples, pero una vez incluidas las características clonables nos encontramos con el problema descrito en el capítulo 6.2 respecto a la semántica de las restricciones. La definición de nuevos operadores y de los modificadores creados es totalmente necesario, aportando como efecto lateral, una semántica más compleja para el desarrollo de nuestros modelos de características.

La configuración asistida es realmente útil, en vez de ir especializando nuestro modelo de características ciegamente y validarlo al final del proceso con el que podemos obtener un modelo incorrecto, al utilizar la configuración asistida o propagación de nuestras acciones por todo nuestro diagrama, eliminamos la posibilidad de seleccionar características muertas o eliminar características comunes, manteniendo así nuestro diagrama correcto en todos los pasos y eliminando de esta forma, la posibilidad de crear modelos incorrectos.

Resumiendo, Hydra es una herramienta que es necesaria en el mundo de las líneas de productos software, la incorporación de características clonables y las funcionalidades añadidas como efectos laterales además de otras características suyas comentadas como su facilidad de uso, diseño gráfico. . . hacen de Hydra una herramienta única por el momento.

### 8.1.4. Evaluación

Hydra ha sido evaluado mediante casos reales de modelos de características provistos la mayoría por miembros del proyecto AMPLE. Entre ellos podemos destacar el modelo de características perteneciente a la línea de productos software de un SmartHome (el cual ha sido definido paso a paso al final de los capítulos 4, 5, 6 y 7), facilitado por Siemens AG. Un caso de uso fue proporcionado por SAP AG, el cual define un escenario de ventas[23]. En ambos Hydra realiza su propósito de forma completamente satisfactoria.

El caso de uso del SmartHome, es quizás el caso más completo dado, ya que tiene

## 8.1. Conclusiones

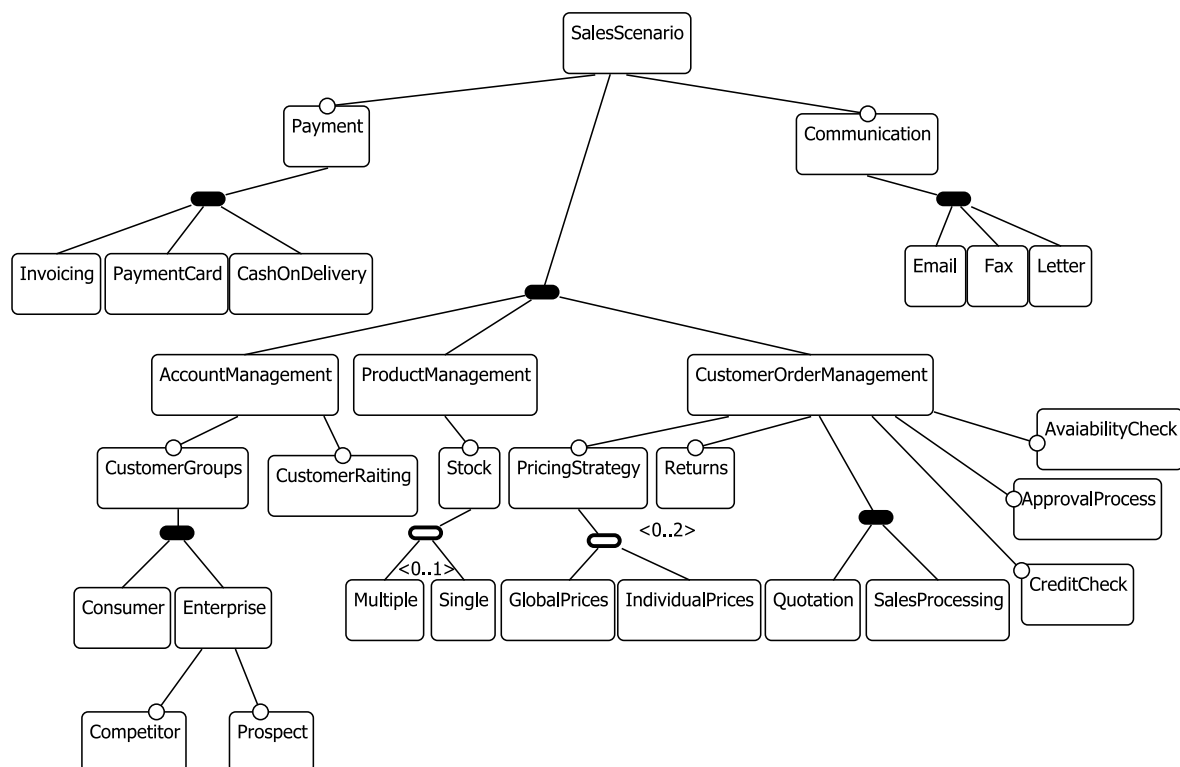


Figura 8.1: Caso de uso de un escenario de ventas.

muchos tipos de objetos como características clonables o cardinalidad infinita y además, en él podemos hacer uso de referencias. Es un modelo de característica no muy grande en el lienzo (25 nodos) siendo conceptualmente infinito por lo que sólo con Hydra podemos modelarlo y configurarlo. No posee muchas restricciones de usuario, tan sólo tres, pero eso es normal para un modelo tan pequeño.

Por otra parte tenemos el caso de uso proporcionado por SAP AG (ver Figura 8.1) que es bastante simple. No dispone de características clonables y, al igual que el modelo anterior, es de tamaño pequeño (30 características). En cambio el tipo de los grupos de características es *SIMPLE* o definido por el usuario, y concretamente uno de ellos no puede ser definido por herramientas que no dispongan de la posibilidad de crear grupos de características definidas por usuario y es el grupo de características con cardinalidad  $\langle 0 \dots 1 \rangle$  que posee dos descendientes. Al no tener ninguna restricción externa, este modelo produce 390.656 soluciones.

Finalmente una compañera del grupo realizó un modelo de características propio sobre un middleware (Figura 8.2). Éste es de tamaño mucho mayor que los anteriores, concretamente tiene 142 características, pero a pesar de eso el modelo produce únicamente 120 configuraciones debido a que posee muchas restricciones de usuario (concretamente 82).

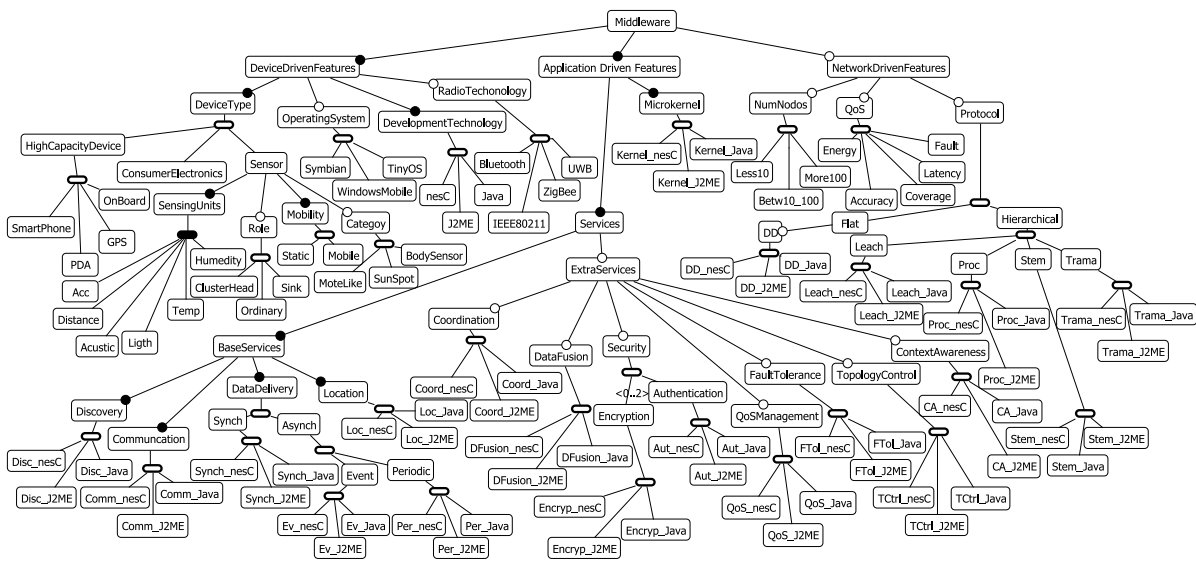


Figura 8.2: Caso de uso de un middleware.

## 8.2. Trabajos Futuros

Todo proyecto es siempre mejorable, existiendo una serie de características, que bien por falta de tiempo, por su excesiva complejidad o por ser de un interés secundario, nunca fueron incorporadas al proyecto o quedaron por resolver. Hydra no es una excepción. A continuación ofrecemos una lista de elementos que se podrían incorporar a Hydra como trabajos futuros:

- Hydra no soporta actualmente características con atributos. Inicialmente se pensó en soportarlas, razón por lo que están incorporados los atributos a nivel de metamodelo. Al no ser esta una funcionalidad de interés para el proyecto AMPLE, fue olvidada durante el desarrollo de los diferentes editores, por lo que dichos atributos no son modelables ni editables gráficamente, ni se pueden definir restricciones en función de dichos atributos. Por tanto, sería de interés soportar atributos en un futuro cercano, y que se pudiesen definir restricciones en función de dichos atributos, tales como *Ingeniero implies Sueldo.cantidad > 1800* o un ejemplo para nuestro caso de uso de la casa inteligente podría ser *Termostato.temperatura < 40*.
- Hydra no soporta actualmente características que referencien a otros modelos de características. Esta funcionalidad es de gran utilidad para descomponer un modelo de características de gran escala en modelos de tamaño más manejable. Dicha funcionalidad es también de interés cuando un cierto subárbol se repite a lo largo de un modelo de características. En tal caso, usando referencias, podemos extraer el subárbol redundante a un modelo de características externo, y sustituir cada ocurrencia por

## 8.2. Trabajos Futuros

---

una referencia. Resultaría por tanto de interés incorporar esta característica a Hydra de cara a trabajar con modelos de gran escala y evitar redundancias.

- Dentro del proyecto AMPLE se ha desarrollado un innovador lenguaje, llamado VML (Variability Modelling Language) [37] para la automatización del proceso de derivación de productos específicos dentro de la línea de productos software. VML es un lenguaje que permite especificar, de forma cómoda y cercana al usuario, qué acciones hay que realizar sobre un modelo que representa la familia completa de productos como consecuencia de la selección o deselección de una determinada características. VML funciona en la actualidad con FMP y no soporta características clonables. Sería por tanto una interesante línea de trabajo futuro el estudiar como integrar Hydra con el editor de VML, de forma que la edición de un modelo de características y su correspondiente especificación VML resulta lo más cómoda posible al usuario. Sería también interesante extender VML para dar soporte a las características clonables.





# Bibliografía

- [1] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, June 1978.
- [2] Michal Antkiewicz and Krzysztof Czarnecki. Featureplugin: feature modeling plug-in for eclipse. In Michael G. Burke, editor, *ETX*, pages 67–72. ACM, 2004.
- [3] Don Batory. Feature models, grammars, and propositional formulas. Technical Report CS-TR-05-14, The University of Texas at Austin, Department of Computer Sciences, April 11 2005. Fri, 28 Sep 107 13:03:38 GMT.
- [4] Don Batory, David Benavides, and Antonio Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47, December 2006.
- [5] Joachim Bayer, Sebastien Gerard, Øystein Haugen, Jason Xabier Mansell, Birger Møller-Pedersen, Jon Oldevik, Patrick Tessier, Jean-Philippe Thibault, and Tanya Widen. Consolidated product line variability modeling. In Timo Käkölä and Juan C. Dueñas, editors, *Software Product Lines - Research Issues in Engineering and Management*, pages 195–241. Springer, 2006.
- [6] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A first step towards a framework for the automated analysis of feature models. In *Managing Variability for Software Product Lines: Working With Variability Mechanisms*, 2006.
- [7] David Benavides, Antonio Ruiz Cortés, Pablo Trinidad, and Sergio Segura. A survey on the automated analyses of feature models. In José Cristóbal Riquelme Santos and Pere Botella, editors, *JISBD*, pages 367–376, 2006.
- [8] Danilo Beuche. Modeling and building software product lines with pure:: Variants. In *SPLC*, page 358. IEEE Computer Society, 2008.

- 
- [9] Danilo Beuche, Holger Papajewski, and Wolfgang Schroeder-Preikschat. Variability management with feature models. In *Software Variability Management Workshop*, pages 72–83, February 2003.
- [10] Sami; B. Beydeda, editor. *Model-Driven Software Development*. Springer, 2005.
- [11] Goetz Botterweck, Mikolás Janota, and Denny Schneeweiss. A design of a configurable feature model configurator. In David Benavides, Andreas Metzger, and Ulrich W. Eisenecker, editors, *Third International Workshop on Variability Modelling of Software-Intensive Systems, Seville, Spain, January 28-30, 2009. Proceedings*, volume 29 of *ICB Research Report*, pages 165–168. Universität Duisburg-Essen, 2009.
- [12] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy Grose. *Eclipse Modeling Framework*. Addison Wesley Professional, 2003.
- [13] Vaclav Cechticky, Alessandro Pasetti, O. Rohlik, and Walter Schaufelberger. XML-based feature modelling. In *ICSR*, volume 3107 of *Lecture Notes in Computer Science*, pages 101–114. Springer, 2004.
- [14] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley, Boston, MA, 2002.
- [15] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau, and Krzysztof Pietroszek. fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates. In Ralph E. Johnson and Richard P. Gabriel, editors, *OOPSLA Companion*, pages 200–201. ACM, 2005.
- [16] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.
- [17] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.
- [18] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *The Journal of Systems and Software*, 74(2):173–194, January 2005.
- [19] Ulrich W. Eisenecker and Krzysztof Czarnecki. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

- [20] Avenue Eugène Freyssinet. CHOCO: implementing a CP kernel français laburhe and the OCRE project team, April 03 2008.
- [21] Richard C. Gronback. *Eclipse modeling project : a domain-specific language toolkit*. Addison-Wesley, 1 edition, April 2009.
- [22] Object Management Group. Meta object facility (MOF) 2.0 core final adopted specification. Technical report, Object Management Group, 2004.
- [23] C. Gomes J. P. Pimentao R. Ribeiro B. Grammel C. Pohl A. Rummler C. Schwaninger L. Fiege M. Jaeger H. Morganho. Requirement specifications for industrial case studies. Technical report, AMPLE Project Deliverable D5.2, March 2008.
- [24] Svein O. Hallsteinsen, Gerard Schouten, Gert Boot, and Tor Erlend Fægri. Dealing with architectural variation in product populations. In Timo Käkölä and Juan C. Dueñas, editors, *Software Product Lines - Research Issues in Engineering and Management*, pages 245–273. Springer, 2006.
- [25] Øystein Haugen, Birger Møller-Pedersen, and Jon Oldevik. Comparison of system family modeling approaches. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*, pages 102–112. Springer, 2005.
- [26] ISO. *ISO/IEC 14977:1996: Information technology — Syntactic metalanguage — Extended BNF*. International Organization for Standardization, pub-ISO:adr, 1996.
- [27] J. Jaffar and M. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19-20:503–581, 1994.
- [28] Frédéric Jouault and Jean Bézivin. KM3: a DSL for metamodel specification. In *IFIP Int. Conf. on Formal Methods for Open Object-Based Distributed Systems, LNCS 4037*, pages 171–185. Springer, 2006.
- [29] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In Stan Jarzabek, Douglas C. Schmidt, and Todd L. Veldhuizen, editors, *Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, October 22-26, 2006, Proceedings*, pages 249–254. ACM, 2006.
- [30] Timo Käkölä and Juan C. Dueñas, editors. *Software Product Lines - Research Issues in Engineering and Management*. Springer, 2006.

- 
- [31] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
- [32] Christian Kästner, Thomas Thüm, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. FeatureIDE: A tool framework for feature-oriented software development. In *ICSE*, pages 611–614. IEEE, 2009.
- [33] Gregor Kiczales. Aspect-oriented programming. *ACM Computing Surveys*, 28(4es):154, December 1996.
- [34] Chang Hwan Peter Kim and Krzysztof Czarnecki. Synchronizing cardinality-based feature models and their specializations. In Alan Hartman and David Kreische, editors, *Model Driven Architecture - Foundations and Applications, First European Conference, ECMDA-FA 2005, Nuremberg, Germany, November 7-10, 2005, Proceedings*, volume 3748 of *Lecture Notes in Computer Science*, pages 331–348. Springer, 2005.
- [35] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley Professional, 2008.
- [36] Miguel A. Laguna, Bruno González-Baixauli, and José M. Marqués. Seamless development of software product lines. In Charles Consel and Julia L. Lawall, editors, *Generative Programming and Component Engineering, 6th International Conference, GPCE 2007, Salzburg, Austria, October 1-3, 2007, Proceedings*, pages 85–94. ACM, 2007.
- [37] Neil Loughran, Pablo Sánchez, Alessandro Garcia, and Lidia Fuentes. Language support for managing variability in architectural models. In Cesare Pautasso and Éric Tanter, editors, *Software Composition*, volume 4954 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2008.
- [38] Maher. A synthesis of constraint satisfaction and constraint solving. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.
- [39] Mike Mannion. Using first-order logic for product line model validation. In Garry Chastek, editor, *Software Product Lines: Proceedings of the Second Software Product Line Conference (SPLC2)*, LNCS 2379, pages 176–187, San Diego, CA, August 2002. Springer.
- [40] Shin Nakajima. Constructing FODA feature diagrams with a GUI-based tool. In *SEKE*, pages 20–25, 2009.

- [41] Object Modeling Group. *Object Constraint Language Specification, version 2.0*, June 2005.
- [42] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 1 edition, 9 2005.
- [43] Stephen R. Schach. *Object-Oriented and Classical Software Engineering*. McGraw-Hill, New York, NY, 2005.
- [44] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2 edition, 12 2008.
- [45] Peri L. Tarr, Harold Ossher, William H. Harrison, and Stanley M. Sutton Jr. *N degrees of separation: Multi-dimensional separation of concerns*. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 107–119, New York, NY, 1999. ACM.
- [46] Juha-Pekka Tolvanen. Domain-specific modeling and code generation for product lines. In *SPLC*, page 229. IEEE Computer Society, 2006.
- [47] Pablo Trinidad, David Benavides, Antonio Ruiz Cortés, Sergio Segura, and Alberto Jimenez. FAMA framework. In *SPLC*, page 359. IEEE Computer Society, 2008.
- [48] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, London, 1993.
- [49] Thomas von der Maßen and Horst Lichter. Requiline: A requirements engineering tool for software product lines. In Frank van der Linden, editor, *PFE*, volume 3014 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2003.
- [50] Thomas von der Maßen and Horst Lichter. Deficiencies in feature models. In *Workshop on Software Variability Management for Product Derivation*, Boston, MA, August 2004.
- [51] Thomas von der Maßen and Horst Lichter. Determining the variation degree of feature models. In J. Henk Obbink and Klaus Pohl, editors, *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*, pages 82–88. Springer, 2005.
- [52] Herbert S. Wilf. *Algorithms and Complexity*. Prentice-Hall, 1986.



# Apéndice A

## Manual de Usuario

El primer apéndice corresponde al manual de usuario de la herramienta desarrollada *Hydra* en el que se detallarán los conceptos básicos para desarrollar un modelo de características.

Primero relataremos como instalar y desinstalar Hydra dentro de Eclipse 3.5 para luego explicar su uso así como atajos de teclado y pequeñas acciones para una rápida y cómoda creación de nuestro modelo de características.

### A.1. Instalación y desinstalación de Hydra

Hydra está embebido en Eclipse, por lo tanto, el primer paso es obtener este IDE de su página oficial<sup>1</sup>. Después de descargarlo tenemos que descomprimirlo dentro de una carpeta elegida.

Hydra depende de TEF, para ello tendremos que instalarlo previamente desde su UpdateSite<sup>2</sup>.

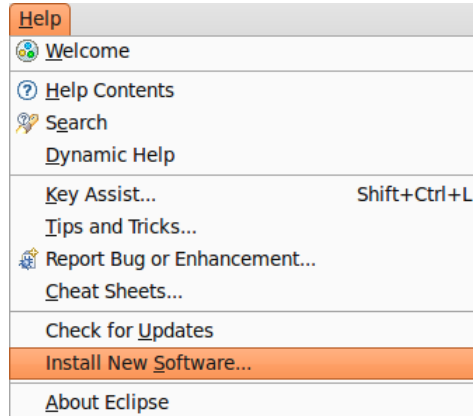
---

<sup>1</sup><http://www.eclipse.org/downloads>

<sup>2</sup><http://tef.berlios.de/updatesite>

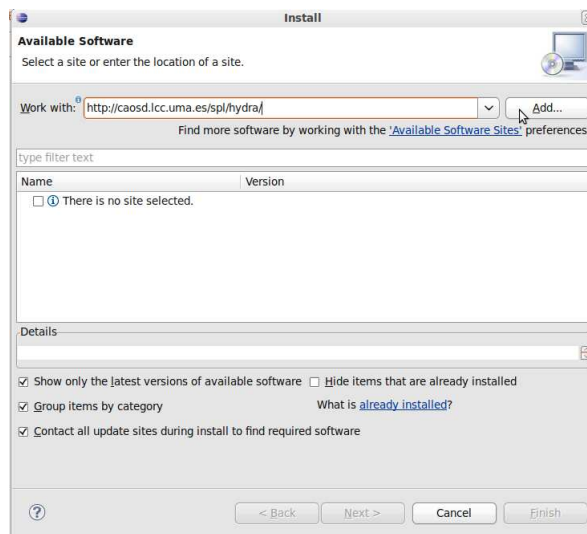
### A.1.1. Instalación

1. Una vez instalado Eclipse, vamos a *Help*→ *Install New Software...*



2. Escribimos en “Work with” la dirección donde está nuestro plugin:

- “http://caosd.lcc.uma.es/spl/hydra/”



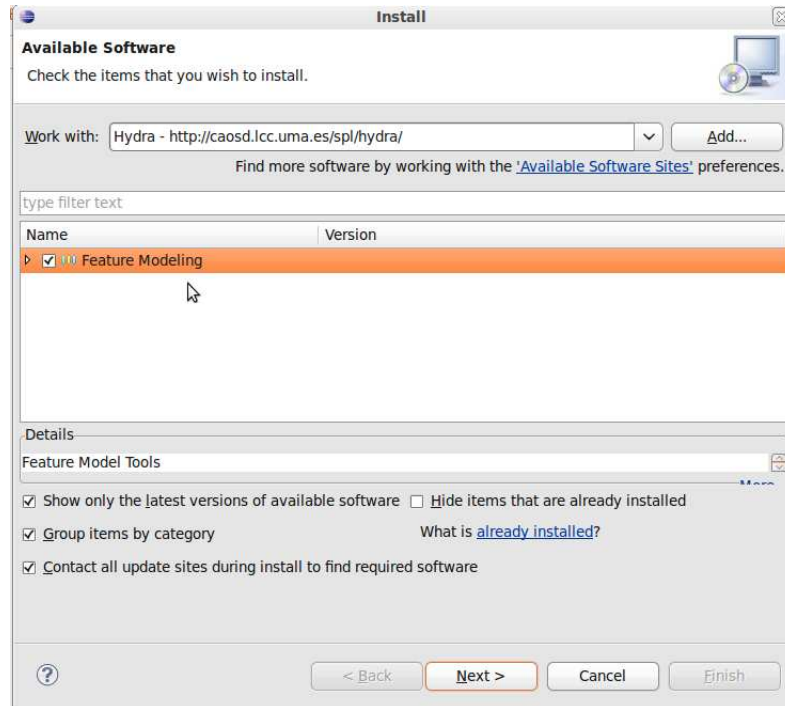
3. Una vez instalado Eclipse, vamos a *Help*→ *Install New Software...*





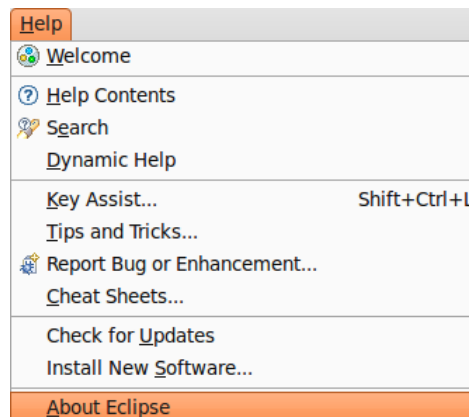
## A.1. Instalación y desinstalación de Hydra

4. Seleccionamos *Feature Modeling* y pasamos a la siguientes secciones hasta instalar Hydra.

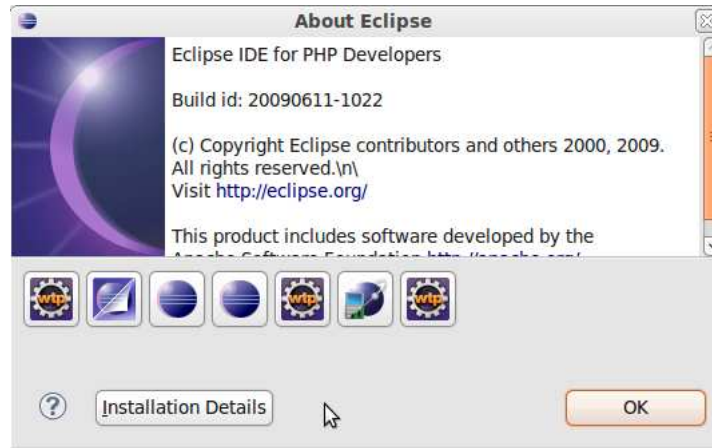


### A.1.2. Desinstalación

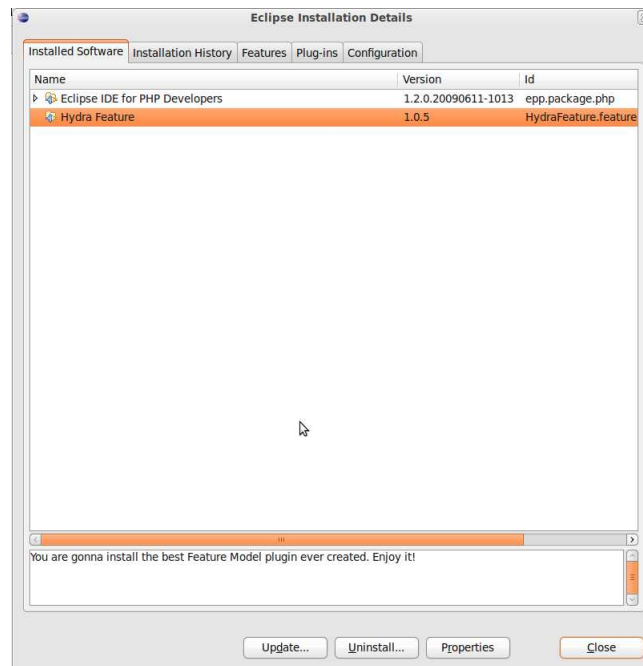
1. Abre Eclipse, y ve a *Help* → *About Eclipse*.



2. Click en *Installation Details*.

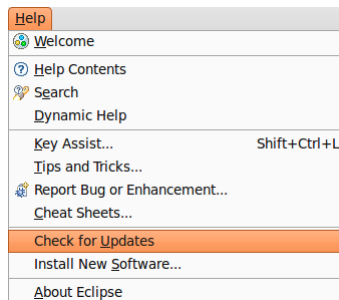


3. Seleccionamos *Hydra Feature* y le damos a desinstalar.



## A.2. Actualización

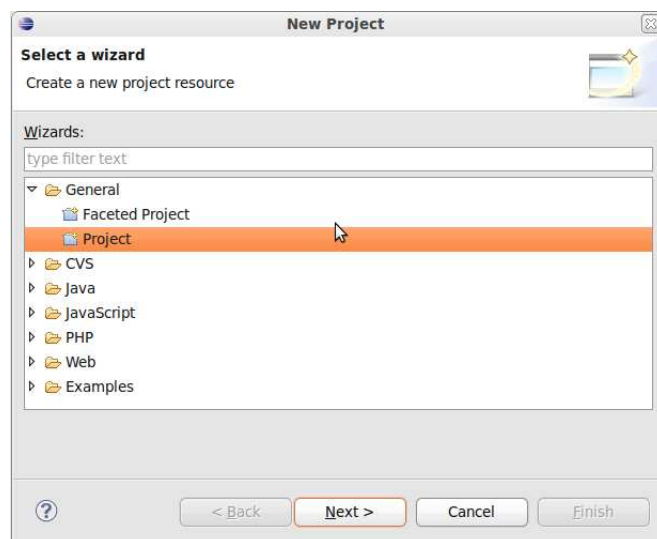
1. Abre Eclipse, y ve a *Help*→ *Check for updates*. Si existe una nueva versión de Hydra, aparecerá.



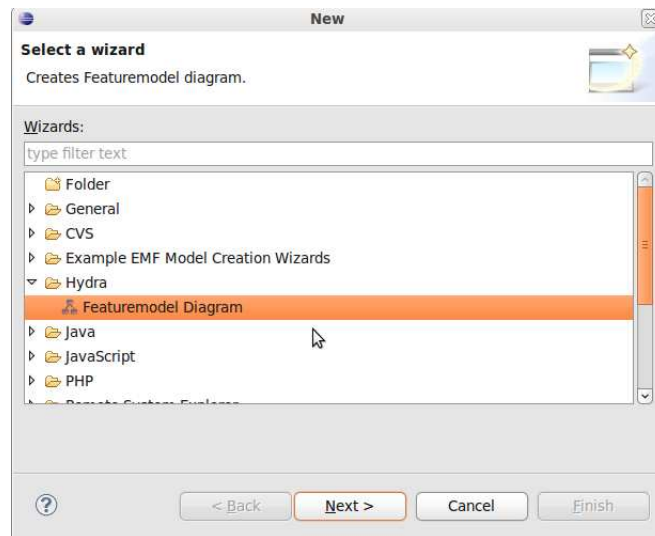
## A.3. Guía de uso

### A.3.1. Creación de proyecto Hydra

1. Para empezar a usar Hydra creamos un proyecto común. Para ello usamos *File*→ *New*→ *Project...* Le damos cualquier nombre y finalizamos.



- Una vez creado nuestro proyecto, crearemos un modelo de características con Hydra, para ello vamos a *File* → *New* → *Other...* y seleccionamos *Featuremodel Diagram* dentro de la carpeta *Hydra*. Le podemos dar cualquier nombre, pero es importante que no borremos la extensión del archivo al escribirlo (*hydra\_diagram*).



### A.3.2. Entorno

Hydra es un plugin multipágina de Eclipse, de esta forma pretendemos que cada proyecto sea lo más cómodo de editar posible. La primera pestaña corresponderá a nuestro modelo de características principal, en él crearemos las características, añadiremos las relaciones entre ellas y editaremos sus atributos. La segunda pestaña contiene el editor de restricciones en el que escribiremos las restricciones que implícitamente no se puedan modelar mediante el diagrama de características previamente modelado. Conforme hagamos especializaciones (desde nuestro modelo de características principal o desde otra especialización), aparecerán nuevas pestañas que corresponderán a los modelos de características de dichas especializaciones. En estos editores únicamente podremos seleccionar o eliminar características (especializar nuestro modelo) con el fin de ir reduciendo el número de productos posibles.

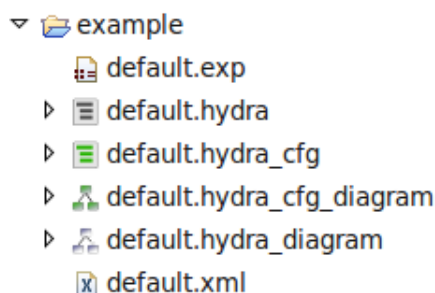
### Ficheros creados

Analizando el explorador de proyectos podemos ver que se han creado unos nuevos ficheros. El fichero con la extensión “hydra\_diagram” será nuestro principal fichero, en él está almacenado el aspecto gráfico de nuestro modelo principal. El fichero con extensión “hydra” será donde esté almacenado el modelo como en formato XMI. El fichero con extensión “exp” será nuestro editor de restricciones. Los ficheros “hydra\_cfg\_diagram” y

### A.3. Guía de uso

---

“hydra\_cfg” corresponden al aspecto gráfico y al modelo Ecore de nuestras configuraciones. Por último, el fichero “xml” almacena algunos datos de configuración de Hydra.

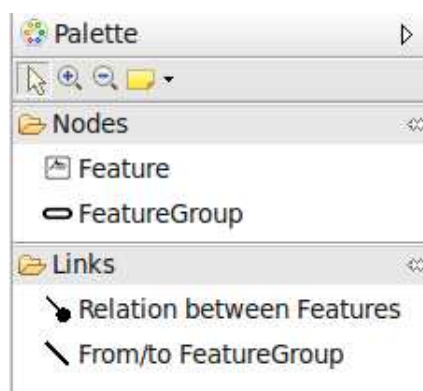


#### Creación de elementos

Si abrimos nuestro fichero principal nos encontraremos con el entorno gráfico de modelado de características de Hydra. La ventana principal será el lienzo, que es donde dibujaremos las características.

A la derecha podemos ver la paleta de herramientas en el podemos ver que existen 4 acciones de creación divididas en nodos y enlaces. En nodos podemos crear características y grupos de características. En los enlaces podemos crear enlaces entre características y enlaces hacia o desde grupos de características.

Para utilizar estas herramientas seleccionaremos alguna de ellas y seguidamente pincharemos en el lienzo si queremos crear nodos o en los nodos que queremos enlazar si queremos crear enlaces.



En la paleta existen además algunos botones que nos permiten crear notas y botones para alejar y acercar nuestro lienzo como si fuese un zoom.

Si tuviésemos que crear un modelo de características de esta forma tardaríamos mucho tiempo, para ello se han creado atajos.

Si seleccionamos una característica y hacemos click derecho en él, aparece un menú contextual. Dentro de *Feature Actions* podemos ver tres posibles acciones.

1. *Create Feature*. Esta acción nos crea una característica hija y la pone en modo edición.
2. *Create FeatureGroup and Feature*. Esta acción crea un grupo de características hijo y una característica dentro de ella en modo edición.
3. *Create sibling*. Crea una característica hermana independientemente de si su padre es una característica o un grupo de características.

Estas tres acciones también están asignadas a atajos de teclado, concretamente “Ctrl + 7”, “Ctrl + 8” y “Ctrl + 9” respectivamente.

## Modificar atributos

Existen dos objetos que tienen tres estados, estos estados dependen de los valores que posean, concretamente de los valores de la cardinalidad que tengan. Según sean estos valores, su tipo (que es derivado de estos dos valores) será diferente, y también así su aspecto visual. Estos objetos son los grupos de características y las relaciones entre características.

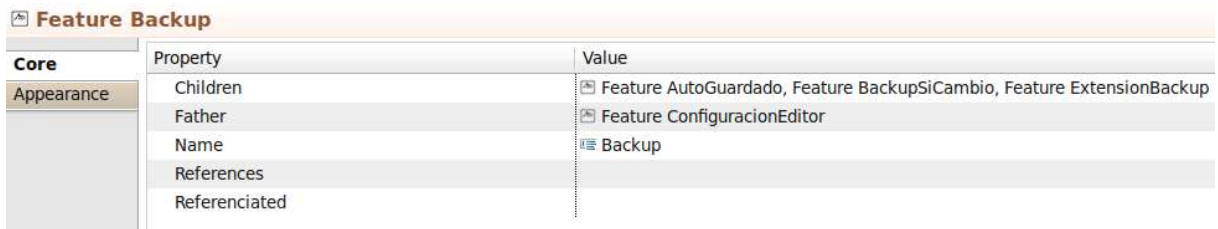
- Para las relaciones entre características tenemos los tipos **OPTIONAL**, **MANDATORY** y **SIMPLE** que corresponden a cardinalidades  $\langle 0..1 \rangle$ ,  $\langle 1..1 \rangle$  y  $\langle m..n \rangle$  (siendo  $m \leq n$ ,  $m \in \mathbb{N}$ ,  $n \in \mathbb{N} \cup \infty$ ) respectivamente. Para escribir el límite superior como infinito introduciremos cualquier número negativo, se representará como \* en el diagrama y conceptualmente corresponderá al infinito.
- Para los grupos de características tenemos los tipos **XORGROUP**, **ORGROUP** y **SIMPLE** que corresponden a cardinalidades  $\langle 1..1 \rangle$ ,  $\langle 1..\#hijos \rangle$  y  $\langle m..n \rangle$  (siendo  $m \leq n \leq \#hijos$ ,  $m \in \mathbb{N}$ ,  $n \in \mathbb{N}$ ) respectivamente.

Para cambiar entre estos tipos podemos modificar los atributos directamente en las propiedades del objeto (podemos modificar tanto las cardinalidades inferiores y superiores como el tipo del objeto) o bien haciendo doble click en su figura correspondiente con el que cambiaremos rápidamente de tipo.

En la parte inferior podemos ver las propiedades de nuestros elementos (si no se ven, hacemos click derecho en cualquier parte del lienzo y le damos a *Show properties view*) que serán diferentes para cada uno de los tipos de elementos. Estos son el proyecto los dos nodos y los dos enlaces.

### A.3. Guía de uso

---



Feature Backup		
Core	Property	Value
Appearance	Children	Feature AutoGuardado, Feature BackupSiCambio, Feature ExtensionBackup
	Father	Feature ConfiguracionEditor
	Name	Backup
	References	
	Referenciaded	

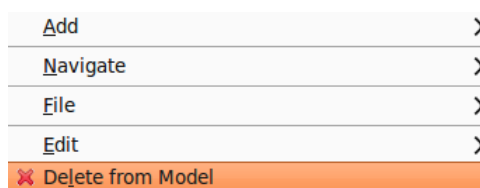
En ellos podemos editar algunos parámetros y ver los valores de otros.

Si nos fijamos, a la izquierda de los atributos aparecen dos pestañas, “Core” y “Appearance” donde podemos modificar aspectos gráficos de nuestro modelo, como colores de las características, tipo de letra, tamaño...

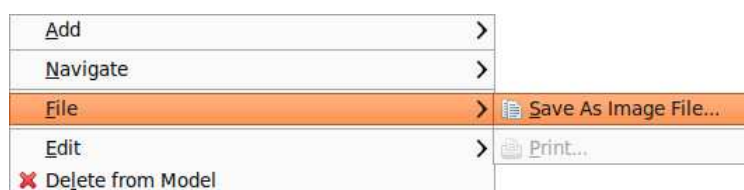


### Acciones

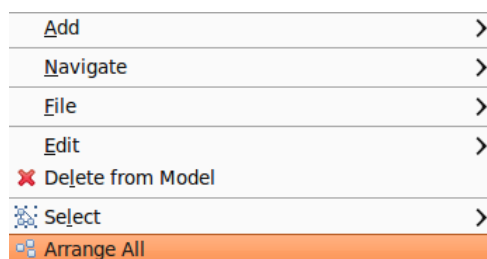
**Eliminar elementos** Si queremos eliminar nodos o enlaces creados (únicamente en el modelo de características principal), los seleccionaremos y haremos click derecho en ellos, escogiendo *Delete from Model*.



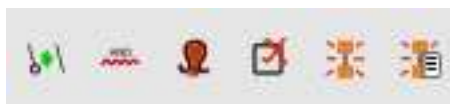
**Exportar diagrama** En el caso de que queramos exportar nuestro diagrama como una imagen, podremos hacer esto haciendo click derecho en el lienzo, *File* → *Save as image File*.



**Ordenar diagrama** Si queremos ordenar el diagrama, hacemos click derecho en el lienzo y seleccionamos *Arrange all*.



**Acciones principales** Existen un grupo de acciones globales cuyo acceso se realiza mediante botones de acciones o dentro del menú contextual Hydra.



Estas acciones son de izquierda a derecha:

- **Especializar un modelo.** Este botón permite crear una configuración a partir del modelo que esté en ese momento mostrándose. Podemos especializar tanto nuestro modelo de características principal, como una especialización de él para crear una más específica. De esta forma podemos decir que Hydra tiene la capacidad de crear configuraciones por etapas.
- **Validar restricciones de usuario.** Comprueba que las restricciones de usuario estén correctamente creadas. Esta acción se realiza automáticamente al guardar el diagrama.
- **Validar el diagrama.** Vuelve a validar nuestro modelo de características. Esta acción se realiza automáticamente cada vez que cambiamos de modelo y sea pertinente ejecutarse.
- **Eliminar configuración.** Elimina la configuración que se muestra del editor multipágina, así como los ficheros que lo representan.
- **Crear configuración mínima.** Calcula la solución mínima de nuestro modelo de característica y crea una configuración con ella.
- **Crear configuración mínima con fichero de restricciones externo.** Esta acción crea una configuración teniendo en cuenta además de los requisitos mínimos (restricciones estructurales del modelo inicial y restricciones de usuario) que las características escritas en un fichero dado deben de estar presentes



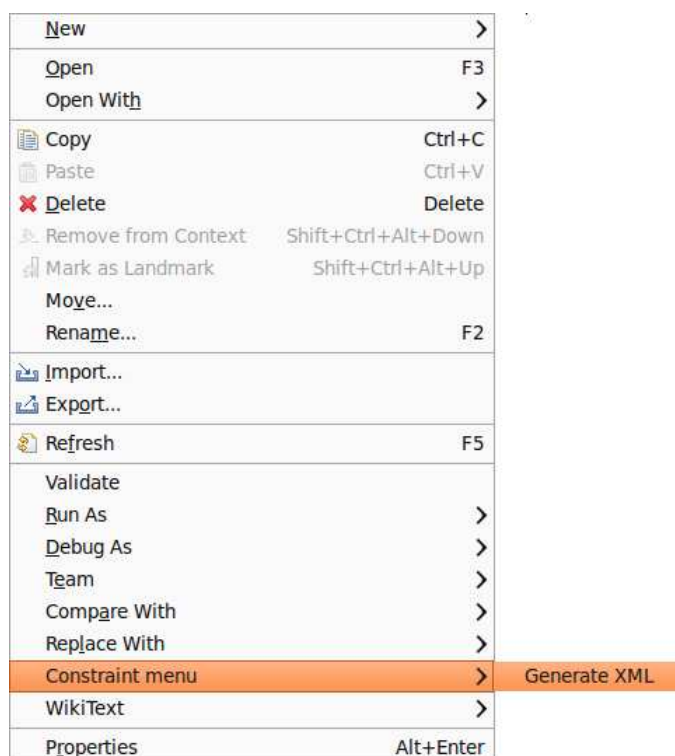
### A.3. Guía de uso

---

en dicha configuración mínima. Esta acción está diseñada específicamente para satisfacer las necesidades de una compañera del grupo GISUM<sup>3</sup>.

Las acciones que creen una nueva configuración no realizarán su cometido a no ser que el modelo de características principal esté correctamente construido y las restricciones de usuario estén bien formuladas.

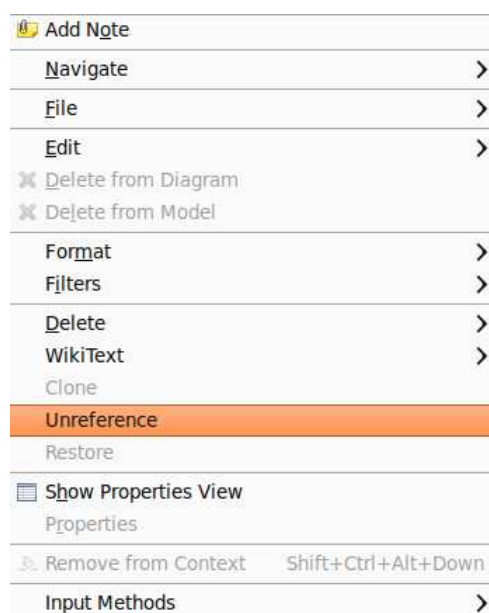
**Exportar restricciones** Si queremos exportar nuestro fichero de restricciones a un formato Ecore podemos hacerlo haciendo click derecho en el fichero de restricciones (extensión “ext”), y seleccionamos *Constraint menu* → *Generate XML*.



**Desreferenciar característica** En caso de que queramos que una característica que referencia a otra deje de referenciarla y tomar sus propios hijos tendremos que seleccionar dicha característica, hacer click derecho en ella y seleccionar “Unreference”.

---

<sup>3</sup>Grupo de Ingeniería del Software de la Universidad de Málaga



## Selección y eliminación de características en configuraciones

Como se comentó anteriormente, no existe la posibilidad de crear nuevos elementos en las configuraciones. Esto es así para asegurar la pertenencia al modelo de características original. Por tanto no existe paleta de creación de elementos ni tampoco está permitida la modificación de los atributos de los elementos (exceptuando el nombre de las características).

En el editor gráfico de configuraciones básicamente sólo se pueden hacer dos acciones, seleccionar una característica o eliminarla.

Para seleccionar una característica, tendremos que hacer doble click en la relación que va hacia dicha característica. Automáticamente se propagará el efecto de dicha selección por el diagrama para mantener el diagrama consistente.

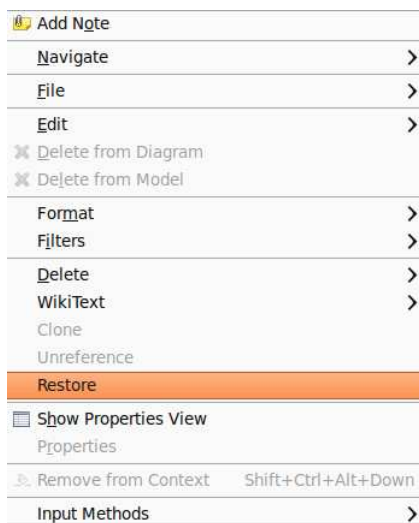
Si lo que queremos es eliminar una característica de nuestro modelo, sencillamente tendremos que hacer doble click en la característica elegida, provocando la eliminación del subárbol que tiene como raíz a la característica eliminada, además de propagarse por el resto del diagrama.

## Recuperación de características eliminadas

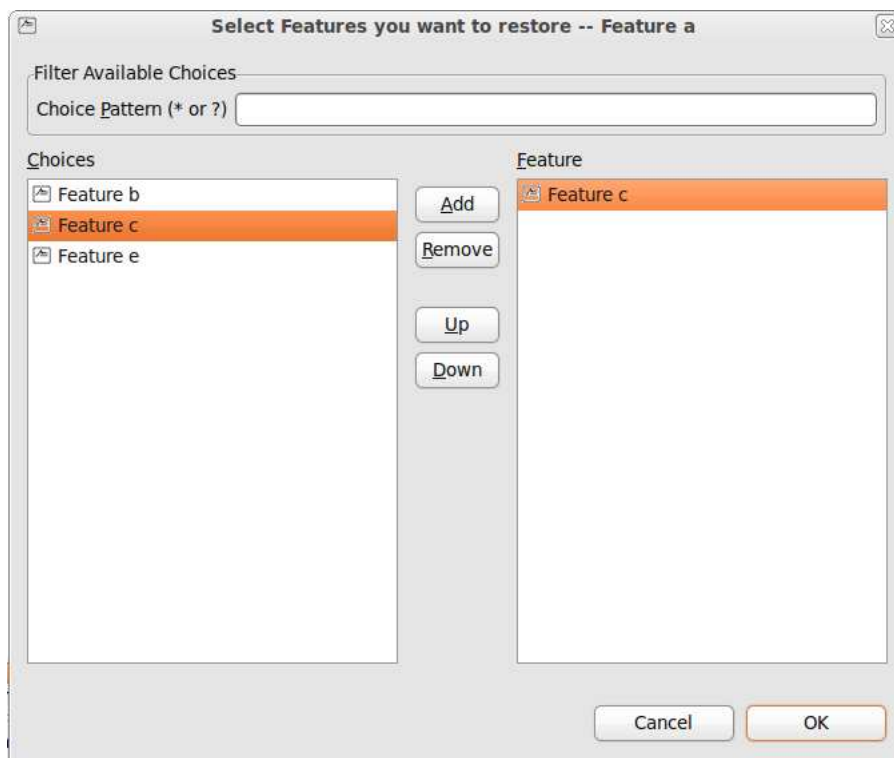
En el caso de que hayamos eliminado una característica y queramos recuperarla (en el caso de que sea posible recuperarla, si la característica que queremos recuperar es una característica muerta, el validador no nos dejará recuperarla) hacemos click derecho en la característica padre y seleccionamos del menú contextual la opción *Restore*.

### A.3. Guía de uso

---



Aparecerá una ventana con dos listas, inicialmente en la lista de la izquierda aparecerán las características que han sido eliminadas y podemos recuperar mientras que en la lista de la derecha se colocarán las características que queremos recuperar. Para ello seleccionamos cada característica y le damos al botón *Add* o hacemos doble click sobre ella.



Una vez terminada la selección, pulsamos *Ok* y se iniciará el proceso de recuperación de los subárboles completos que tienen como raíz a las características seleccionadas.

Recordar que esta acción sólo se puede hacer en el editor de configuraciones y que aunque se vaya a recuperar el subárbol, el validador comprobará si existe alguna carac-

terística de las que se van a recuperar que viole el estado de satisfacibilidad del diagrama, eliminándola de la recuperación.

# Apéndice B

## Acrónimos

AMPLE	Aspect-Oriented Model-Driven Product Line Engineering
AOSD	Aspect-Oriented Software Development
CSP	Constraint Satisfaction Problem
EMF	Eclipse Modelling Framework
FM	Feature Model
FMP	Feature Modelling Plug-in
GMF	Graphical Modelling Framework
GUI	Graphical User Interface
MDD	Model Driven Development
OO	Object-Oriented
OOP	Object-Oriented Programming
SPL	Software Product Line
TEF	Textual Editing Framework
UML	Unified Modelling Language
VML	Variability Modelling Language
XMI	XML Metadata Interchange
XML	eXtensible Markup Language