Garantizando evoluciones seguras en sistemas auto-adaptables

Trabajo Final de Máster

Máster en Ingeniería del Software, Métodos Formales y Sistemas de Información

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Autor:
María Ortega López

Directores:
María Gómez Lacruz
Joan Fons i Cors

Garantizando evoluciones seguras en sistemas auto-adaptables

Autor

María Ortega López

Directores

María Gómez Lacruz

Joan Fons i Cors

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1. INTRODUCCIÓN

En este capítulo vamos a introducir el trabajo realizado en este trabajo final de máster. Concretamente introduciremos el contexto de los sistemas auto-adaptables y sensibles al contexto, plantearemos el problema de la evolución y mantenimiento de éstos sistemas y la solución basada en desarrollo dirigido por modelos que se propone.

1.1 INTRODUCCIÓN

En los últimos años la evolución de la tecnología ha ido creciendo exponencialmente. Día a día las diferentes tecnologías están más presentes en nuestras vidas, incluso dependemos totalmente de ellas para nuestras tareas diarias. Los nuevos dispositivos aportan nuevas funcionalidades hasta el punto de interactuar o depender del entorno para su funcionamiento.

El software y los elementos hardware asociados están sujetos a cambios que van ocurriendo con el tiempo. Estos cambios pueden deberse a problemas en los sistemas, por la incorporación de nuevos componentes, o por variaciones de las necesidades de los usuarios. Esto requiere que los sistemas se actualicen y adapten a las nuevas condiciones para continuar ofreciendo su servicio con garantías.

Tradicionalmente, estas adaptaciones se han realizado de forma manual, lo que obligaba a detener el sistema para aplicar los cambios. Pero cada vez se tiene más presente el desarrollo de sistemas donde el propio sistema se adapta automáticamente en tiempo de ejecución a las nuevas condiciones. Esta adaptabilidad se considera una capacidad necesaria en sistemas altamente dinámicos como los sistemas *sensibles al contexto* [1] [2] o *sistemas ubicuos* [3].

La *auto-adaptación* proporciona a un sistema software la capacidad de adaptar su comportamiento y/o estructura en ejecución en respuesta a cambios en el entorno y en el propio sistema sin intervención humana [4]. Esto hace que los sistemas funcionen de una forma autónoma, adaptándose a la situación que les rodea automáticamente para hacernos la vida más fácil.

Para desarrollar sistemas auto-adaptables, los ingenieros identifican en tiempo de diseño los posibles cambios que pueden ocurrir y especifican cómo se deberá comportar el sistema ante éstos. Sin embargo, en entornos altamente dinámicos no se pueden prever todas las eventualidades a priori. Por ello, los sistemas auto-adaptables deben estar abiertos a evolución con el objetivo de manejar nuevos requisitos y no quedar obsoletos.

Actualmente las actividades de evolución y mantenimiento del software se llevan a cabo con el sistema parado [5]. Sin embargo los procesos de evolución y mantenimiento en los sistemas auto-adaptables deben llevarse a cabo en ejecución.

En éste ámbito, se requieren nuevas técnicas y herramientas para dar soporte a la evolución en tiempo de ejecución de sistemas auto-adaptables. Y concretamente que permitan

la corrección, mejora e incorporación de nuevas capacidades que no fueron previstas inicialmente en diseño.

1.2 PLANTEAMIENTO DEL PROBLEMA

Un sistema software con propiedades autónomas es capaz de instalar, configurar, adaptar y mantener sus componentes en tiempo de ejecución para seguir ofreciendo su funcionalidad. Este tipo de sistema trabaja en entornos cambiantes y su comportamiento debe evolucionar con el tiempo.

La aproximación tradicional para evolucionar un sistema software a una nueva versión consiste en parar el sistema, aplicar los cambios requeridos y reiniciar el sistema de nuevo. Esta aproximación, llamada *stop-and-restart*, interrumpe la ejecución del sistema, dejándolo no disponible por cierto tiempo (por ejemplo las actualizaciones de Windows). Para sistemas críticos, cuyo nivel de exigencia es tan elevado que detener la ejecución es un grave problema, o bien los cambios se dan muy frecuentemente, ésta aproximación es inaceptable. Como es el caso de los sistemas auto-adaptables.

En sistemas auto-adaptables, los procesos de evolución y mantenimiento tienen que ser llevados a cabo mientras el sistema está en ejecución. Los sistemas auto-adaptables suelen ser muy complejos, por ello se necesitan técnicas y herramientas para garantizar evoluciones seguras, es decir, que la nueva versión del sistema no introduce errores ni inconsistencias en el sistema en funcionamiento.

1.3 SOLUCIÓN PROPUESTA

La *Ingeniería Dirigida por Modelos (MDE)* se centra en el uso de modelos durante el proceso de desarrollo software. Pero los modelos también pueden jugar un papel importante en tiempo de ejecución. Esta iniciativa se conoce como *Modelos en Tiempo de Ejecución* [6]. Siguiendo esta aproximación, el conocimiento de los sistemas auto-adaptables se desarrolla mediante descripciones representadas como modelos. De esta manera los modelos no son sólo artefactos de primer orden en el desarrollo, sino que también permiten monitorizar, validar y adaptar el comportamiento del sistema en ejecución. Los modelos capturan en todo momento el conocimiento del sistema (información del sistema y el entorno), para entender su situación actual y dirigir el comportamiento autónomo.

Siguiendo una aproximación de Modelos en Tiempo de Ejecución, la evolución del sistema auto-adaptable puede realizarse a nivel de modelo, y no de implementación, es decir evolucionando los modelos que constituyen el conocimiento del sistema. Así el proceso de evolución de los sistemas auto-adaptables en ejecución estará compuesto por las siguientes fases: en primer lugar, se obtiene una copia de los modelos que describen el conocimiento del sistema, donde se incorporan los cambios dados por la evolución; en segundo lugar, con el

objetivo de no introducir errores ni inconsistencias en el sistema en ejecución, los cambios se analizan y se corrigen errores en caso de detectarse. Finalmente, si como resultado de la fase de análisis obtenemos que los nuevos modelos son válidos, éstos pueden aplicarse en el sistema en ejecución.

En sistemas de gran tamaño, el análisis de los modelos implica gran complejidad y no puede realizarse a simple vista por el ingeniero. Por ello, el objetivo de este trabajo final de máster es la implementación de una herramienta para dar soporte a la fase de análisis del proceso de evolución, de manera que permita al ingeniero del sistema analizar la nueva versión evolucionada de los modelos para detectar errores y ofrecer mecanismos para obtener automáticamente una versión libre de errores.

1.4 OBJETIVOS DEL TRABAJO FINAL DE MÁSTER

Para solucionar los problemas planteados, el objetivo principal de éste trabajo final de máster es la implementación de una herramienta para dar soporte a la fase de análisis del proceso de evolución de sistemas auto-adaptables. La herramienta permite garantizar que la nueva versión evolucionada de los modelos no contiene errores ni inconsistencias. Este objetivo se puede descomponer en los siguientes subobjetivos:

1. Proporcionar mecanismos para la ***detección*** y ***explicación de errores*** en los modelos evolucionados.
2. Proporcionar ***refinamientos*** para corregir automáticamente los errores detectados en los modelos evolucionados.

1.5 ESTRUCTURA DEL TRABAJO FINAL DE MÁSTER

En este apartado vamos a explicar resumidamente los diferentes capítulos de los que consta esta memoria:

- Capítulo 2, Contexto: en este apartado se va a presentar el contexto en el que se va a localizar el desarrollo del trabajo final de máster. Concretamente explicaremos ciertos conceptos referentes al ámbito tecnológico: Computación Autónoma, Ingeniería Dirigida por Modelos y Líneas de Producto Software. También se explica en qué consisten los distintos tipos de modelos usados en el modelado de los sistemas. Incluyendo qué modela cada uno, con qué notación y también las transformaciones de uno a otro.

- Capítulo 3, Desarrollo de la propuesta: aquí explicaremos el proceso de evolución de sistemas en tiempo de ejecución, y más en concreto la fase de análisis. Y finalmente explicaremos el proceso de validación y refinamiento de los distintos tipos de modelos.

- Capítulo 4, Caso de estudio: en este apartado se presenta un caso de estudio para ejemplificar el uso de la herramienta.

- **Capítulo 5, Herramientas de soporte:** aquí se explican tanto las herramientas existentes utilizadas, como las distintas tecnologías empleadas.

- **Capítulo 6, Conclusiones y trabajos futuros:** En este capítulo se presentan las contribuciones principales del desarrollo que se ha realizado, los resultados y conclusiones obtenidas. Y además se exponen las posibles futuras líneas de trabajo y los diferentes aspectos relacionados con el trabajo final de máster que no se han abordado y que se podrían llevar a cabo en desarrollos futuros.

2. CONTEXTO

En este capítulo vamos a explicar el contexto en el que se realiza este trabajo final de máster. En primer lugar se explican conceptos clave del ámbito tecnológico, como la Computación Autónoma, la Ingeniería Dirigida por Modelos y las Líneas de Producto Software. Y en segundo lugar, explicaremos qué son los Sistemas Auto-Adaptables y veremos los Modelos de Variabilidad que van a usarse en esta propuesta.

2.1 ÁMBITO TECNOLÓGICO

2.1.1 COMPUTACIÓN AUTÓNOMA

La complejidad de los sistemas software actuales ha obligado a la comunidad de ingeniería de software a investigar maneras innovadoras de desarrollar, desplegar, gestionar y evolucionar sistemas y servicios software. Además, ante el incremento de la complejidad, los sistemas software deben ser más versátiles, flexibles, resistentes, fiables, eficientes energéticamente, recuperables, personalizables, configurables y auto-optimizables mediante la adaptación a los cambios que puedan ocurrir en sus contextos operacionales, entornos y requisitos de sistema [7].

Según Alan Ganek [8] [9]la computación autónoma se define como:

"La computación autónoma es la habilidad de los sistemas de ser más auto-gestionables. El término autónomo proviene del sistema nervioso autónomo, que controla diversos órganos y músculos en el cuerpo humano. Generalmente, nosotros no somos conscientes de su funcionamiento, ya que funciona de manera involuntaria y reflexiva –por ejemplo, no nos damos cuenta cuando nuestro corazón late más deprisa o nuestros vasos sanguíneos cambian de tamaño en respuesta a la temperatura, la postura, el consumo de alimentos, las experiencias estresantes y otros cambios a los que estamos expuestos. Y nuestro sistema nervioso autónomo está siempre trabajando"

La computación autónoma contempla entornos de computación que evolucionan sin la necesidad de intervención humana [10]. Un sistema con capacidades autónomas instala, configura y mantiene sus propios componentes en tiempo de ejecución. La definición de abstracciones y modelos apropiados para entender, controlar y diseñar comportamiento autónomo es el objetivo principal de la computación autónoma.

A continuación describimos brevemente algunas de las principales propiedades de la computación autónoma según la visión de IBM [11] [12]:

- Auto-configuración: Un sistema de computación autónoma se configura así mismo de acuerdo a sus principales objetivos. Esto significa la capacidad de poder instalar nuevos componentes de acuerdo a las necesidades del usuario y de la plataforma.

- Auto-optimización: Un sistema de computación autónoma optimiza el uso de sus recursos. Éste puede decidir iniciar el cambio a un sistema proactivo con el objetivo de mejorar el rendimiento o la calidad del servicio.

- Auto-reparación: Un sistema de computación autónoma detecta y diagnostica los problemas. El tipo de problemas detectados pueden ser muy variados, desde problemas en el hardware hasta problemas de tipo software.

- Auto-protección: Un sistema de computación autónoma se protege a sí mismo de ataques maliciosos pero también de usuarios finales que accidentalmente toman acciones dañinas. Estos sistemas se ajustan para mantener la seguridad, privacidad y protección de los datos.

2.1.2 INGENIERÍA DIRIGIDA POR MODELOS (MDE)

La Ingeniería Dirigida por Modelos (MDE) es una metodología de desarrollo de software que se centra en crear y explotar modelos de dominio, esto es, representaciones abstractas del conocimiento y actividades que rigen un dominio de aplicación en particular. Una aproximación MDE pretende incrementar la productividad maximizando la compatibilidad entre sistemas (mediante el uso de modelos estandarizados), simplificando el proceso de diseño (mediante el uso de patrones de diseño recurrentes en el dominio de la aplicación) y promoviendo la comunicación entre los ingenieros y equipos trabajando en el sistema (mediante la estandarización de la terminología) [13].

Dentro de MDE podemos enmarcar:

- El Desarrollo Dirigido por Modelos (MDD), que es un paradigma de desarrollo donde los modelos son el centro de desarrollo de software.

- La Arquitectura Dirigida por Modelos (MDA), es una arquitectura, propuesta por el Object Management Group (OMG), que proporciona un conjunto de guías para estructurar especificaciones expresadas como modelos, siendo estos modelos los artefactos principales del desarrollo de software. El principal objetivo de MDA es separar el diseño de la arquitectura de las tecnologías de implementación.

La llegada de MDD y de MDA ha cambiado la forma de utilizar los modelos en el desarrollo de software. La noción de Ingeniería Dirigida por Modelos (MDE) surge como un paradigma generalizador de la aproximación MDA para el desarrollo software.

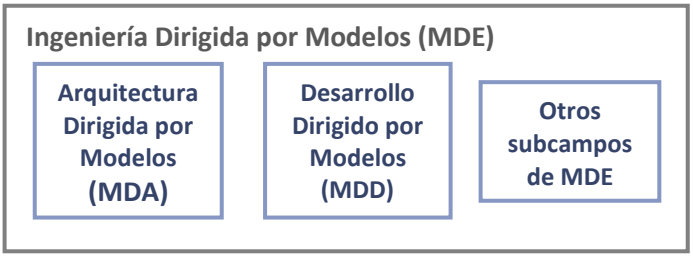


Figura 1: Visión del paradigma MDE

El tipo de proceso que se propone está basado en el hecho de que el desarrollo está dirigido por la especificación de los modelos y sus transformaciones. La capacidad para transformar diferentes representaciones de modelos es lo que caracteriza el Desarrollo Dirigido por Modelos del desarrollo tradicional, donde los modelos únicamente se utilizaban para esbozar un diseño.

Las tecnologías basadas en Ingeniería Dirigida por Modelos combinan lo siguiente [13]:

- *Lenguajes de modelado específicos de dominio* (DSMLs) que formalizan la estructura, comportamiento y requisitos de la aplicación dentro de un dominio particular. Los DSMLs se describen usando metamodelos, que definen la relación entre conceptos en un dominio y concretamente especifican la semántica clave y las limitaciones asociadas con esos conceptos del dominio.

- *Motores de transformación y generadores* que analizan ciertos aspectos de los modelos y sintetizan varios tipos de artefactos, como código fuente, entradas simuladas, o representaciones de modelos alternativas. La habilidad para sintetizar artefactos ayuda a asegurar la consistencia entre las implementaciones de aplicación y la información de análisis asociada a los requisitos funcionales y de calidad de servicio capturados por los modelos.

La Arquitectura Dirigida por Modelos, como ya se ha mencionado, es una especificación concreta del Desarrollo Dirigido por Modelos. MDA concibe la construcción de modelos de software a distintos niveles de abstracción: en primer lugar, un modelo independiente de la computación (Computational-Independent Model, CIM), que modela el contexto y requisitos de negocio; en segundo lugar, un modelo independiente de la plataforma (Platform-Independent Model o PIM), que es un modelo con un alto nivel de abstracción e independiente de cualquier tecnología de implementación; y por último, un PIM puede transformarse en uno o más modelos dependientes de plataforma (Platform Specific Model, PSM), que especifica el sistema en términos de la implementación en una determinada tecnología. El último paso a realizar sería la transformación de cada uno de los PSM a código.

Figura 2: Niveles de abstracción de modelos MDA

El Desarrollo Dirigido por Modelos permite capturar los aspectos importantes de un sistema software a través de los modelos. En comparación con la implementación de código fuente, los modelos pueden plasmar las necesidades y requisitos del sistema de una manera más directa, sin entrar en detalles técnicos. La principal característica de toda esta corriente es que los modelos no son artefactos auxiliares de documentación, sino que son artefactos fuente que serán transformados en el futuro código del programa.

2.1.3 LÍNEAS DE PRODUCTO SOFTWARE (SPL)

Las Líneas de Producto Software, también conocidas como SPL (*Software Product Line*), fueron definidas por Clements y Northrop [14] como:

"Un conjunto de sistemas software intensivos, que comparten un conjunto común y gestionado de características que satisfacen las necesidades específicas de un segmento o misión particular del mercado y que son desarrollados a partir de un conjunto común de activos principales de una manera prescrita"

La estrategia de Línea de Productos se ha utilizado comúnmente en la industria, pero ha sido recientemente cuando ha empezado su influencia en los procesos de desarrollo de software. Este enfoque tiene como objetivo conseguir mejoras en la productividad y el tiempo de comercialización mediante el diseño de un conjunto de productos con muchas partes en común. En cierta manera, supone un nuevo esquema de reutilización de software, que ha demostrado ser eficaz en la experiencia industrial real.

Las Líneas de Producto Software tratan de producir familias de sistemas parecidas en lugar de producir sistemas individuales. La ingeniería de productos consiste en tres procesos principales: ingeniería de dominio, ingeniería de aplicación y gestión. La Figura 3 ilustra un gráfico de los 3 procesos principales de la ingeniería de productos y sus relaciones.

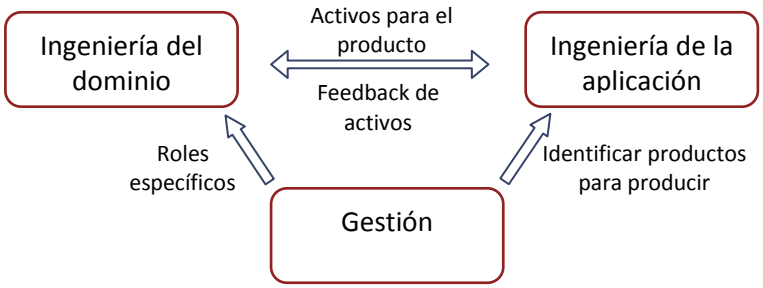


Figura 3: Elementos Línea de Productos Software

- Ingeniería de dominio:

La ingeniería de dominio se define como la actividad de recogida, organización y almacenamiento de experiencias pasadas en la construcción de sistemas completos o parciales en un dominio particular en forma de activos reutilizables, así como para proporcionar los mecanismos adecuados para su reutilización en futuros sistemas.

Podemos decir que la ingeniería de dominio se relaciona con la identificación de los elementos comunes, la variabilidad de los productos en la Línea de Productos y la implementación de los elementos compartidos. Así es posible explotar los elementos comunes y

enriquecer el producto con los elementos variables. En general, la ingeniería de dominio se divide en análisis del dominio, análisis del diseño e implementación del dominio.

- Ingeniería de la aplicación:

La ingeniería de la aplicación es el proceso de construcción de un sistema dentro de un dominio, concretamente, es responsable de derivar un producto concreto a partir de una SPL utilizando una aproximación "diseñar para reutilizar". Para conseguir esto, se usan los elementos reutilizables desarrollados previamente.

Durante la ingeniería de la aplicación, los productos individuales se desarrollan mediante la selección de artefactos de configuración compartidos y, si es necesario, añadiendo extensiones específicas del producto. Este proceso se divide en análisis, diseño e implementación de la aplicación.

- Gestión:

El proceso de gestión es un proceso separado, donde los temas organizacionales se manejan específicamente. Este proceso es el encargado de dar recursos, coordinar y supervisar el dominio y aplicaciones de las actividades de ingeniería.

Las líneas de producto son una exitosa aproximación para hacer posible la reutilización de componentes dentro de una organización. Una Línea de Producto Software estándar está formada por una arquitectura de línea de producto, un conjunto de componentes software y un conjunto de productos.

2.2 CONCEPTOS BÁSICOS

2.2.1 SISTEMAS AUTO-ADAPTABLES

Los sistemas auto-adaptables son capaces de razonar acerca de su estado interno y del entorno. Una de las técnicas utilizadas para la construcción de sistemas auto-adaptables es la introducción de un bucle de control, que se encarga de guiar el comportamiento autónomo [15]. De acuerdo a esta estrategia, el sistema monitoriza continuamente el entorno de ejecución, analiza la información, planifica cómo adaptar el sistema, y ejecuta un plan de adaptación basado en el conocimiento del sistema y el entorno.

Este conocimiento puede ser representado de distintas formas: descripciones textuales, metadatos en código, o modelos. En este trabajo final de máster se utilizan abstracciones basadas en modelos. El conocimiento contiene descripciones sobre el sistema, las políticas de adaptación y el entorno. A continuación se describen detalladamente cada una de las representaciones utilizadas para modelar el conocimiento del sistema.

Cuando el sistema auto-adaptable está en ejecución, el diseñador debe poder evolucionar el conocimiento del sistema para incorporar nuevas capacidades que permitan responder a requisitos no previstos en el diseño original. Los modelos evolucionados serán validados y procesados para obtener los cambios reales que deben aplicarse al sistema en tiempo de ejecución.

2.2.2 MODELADO DE VARIABILIDAD

La idea clave del modelado de variabilidad es la separación de, por una parte, los puntos comunes del sistema y, por otra parte, los puntos de variación. De este modo, se construyen los elementos comunes mientras que los elementos variables son expresados de una manera eficiente para que puedan gestionarse cuando corresponda.

Los puntos comunes son aquellas asunciones sobre componentes que son iguales en todos los sistemas. Por el contrario, son puntos de variación aquellas asunciones ciertas solamente en algunos sistemas, tal como un componente con diferente especificación para al menos dos sistemas.

Por tanto, los modelos de variabilidad nos permiten describir las distintas variantes de ejecución que pueden darse en el sistema, de manera que, el sistema sea capaz de responder a los cambios que se den en el contexto, consultar estos modelos de variabilidad y determinar los cambios necesarios en su arquitectura.

2.2.2.1 MODELADO DE CARACTERÍSTICAS

En este trabajo final de máster en concreto vamos a utilizar Modelos de Características para modelar la variabilidad. Un Modelo de Características representa la información de todos los posibles productos de una Línea de Productos Software, en términos de características y relaciones entre ellas. Los modelos de características son un tipo especial de modelo de variabilidad extensamente usado en la ingeniería de Líneas de Producto Software [16]. Un Modelo de Características se representa como un conjunto jerárquico de características compuesto por:

- Relaciones entre una característica padre y sus características hijas.

- Restricciones '*cross-tree*' que son típicamente la inclusión o exclusión entre características.

A continuación mostramos un ejemplo de Modelo de Características inspirado en la industria del teléfono móvil (ver Figura 4). El modelo ilustra cómo las características se usan para especificar y construir software para teléfonos móviles.

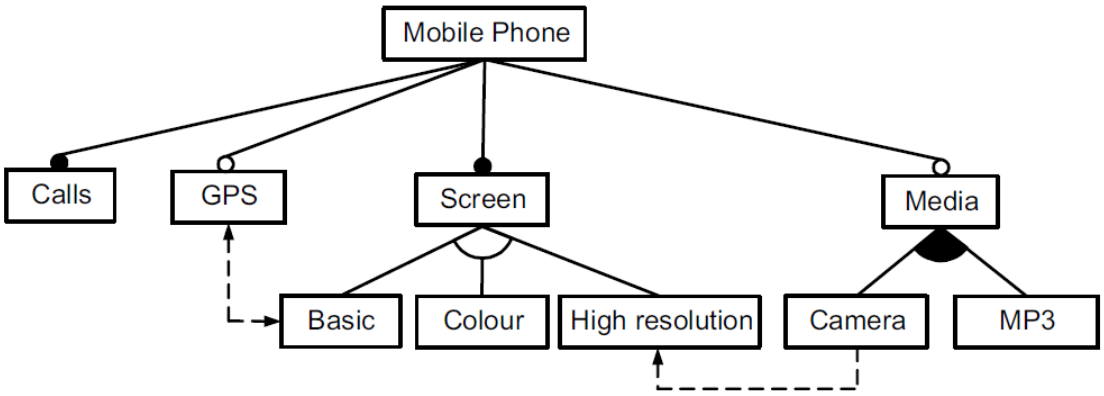


Figura 4: Modelo de Características de ejemplo (extraído de [16])

Las relaciones entre una característica padre y sus características hijas pueden ser las siguientes:

- ***Obligatoria:*** Una característica hija tiene una relación *Obligatoria* con su característica padre cuando el hijo se incluye en todos los productos donde el padre aparezca. En el ejemplo de la Figura 5, podemos ver que todos los teléfonos móviles soportan llamadas.

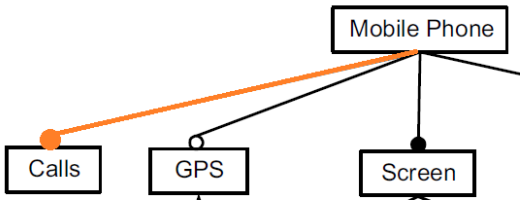


Figura 5: Ejemplo de relación Obligatoria

- ***Opcional:*** Una característica hija tiene una relación *Opcional* con su característica padre cuando el hijo puede estar incluido opcionalmente en todos los productos donde el padre aparezca. En el ejemplo (Figura 6), el software del teléfono puede opcionalmente incluir el soporte de GPS.

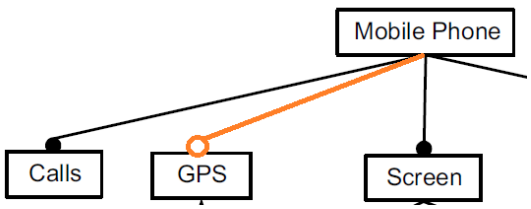


Figura 6: Ejemplo de relación Opcional

- ***Alternativa:*** un conjunto de características hijas tiene una relación *Alternativa* con su característica padre cuando sólo una característica hija puede seleccionarse cuando la característica padre sea parte del producto. En el ejemplo (Figura 7), los teléfonos pueden incluir soporte para pantalla *Basic*, *Colour* o *High Resolution*.

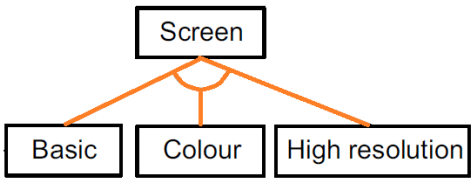


Figura 7: Ejemplo de relación Alternativa

- ***Or:*** un conjunto de características hijas tiene una relación *Or* con su característica padre cuando una o más de ellas puede ser incluida en los productos en los que su característica padre aparezca. En el ejemplo (Figura 8), si *Media* esta seleccionada, *Camera* o *Mp3*, o ambas, pueden ser seleccionadas.

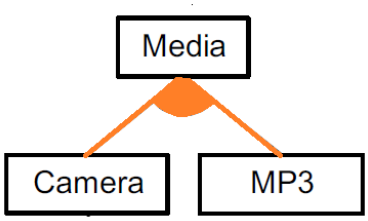


Figura 8: Ejemplo de relación Or

Además de las relaciones padre-hijo entre características, un Modelo de Características también puede contener restricciones *cross-tree* entre características.

- ***Requiere:*** Si una característica A requiere la característica B, la inclusión de A en un producto implica la inclusión de B. En el ejemplo (Figura 9), los móviles que incluyen cámara deben contener también soporte para pantalla *High Resolution.*

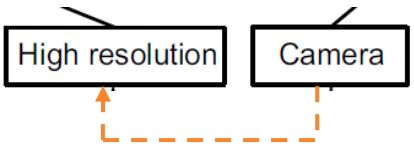


Figura 9: Ejemplo de relación Requiere

- ***Excluye:*** Si una característica A excluye a la característica B, ambas características no pueden ser parte del mismo producto. En el ejemplo (Figura 10), *GPS* y pantalla *Basic* son características incompatibles.

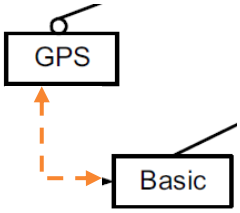


Figura 10: Ejemplo de relación Excluye

2.2.2.2 MODELADO DE CONFIGURACIONES

Una configuración representa el estado del sistema en un determinado instante de tiempo. Una configuración está formada por un conjunto de pares (Característica, Estado). El estado de una característica puede ser Activa o Inactiva. Las distintas combinaciones entre características activas e inactivas forman el conjunto de configuraciones que el sistema puede tomar.

El estado general del sistema o configuración queda reflejado a través de las características activas e inactivas del sistema. Una configuración contiene el estado de todas las características del sistema.

En la Tabla 1 se muestra una configuración de ejemplo basada en el Modelo de Características de la Figura 4:

$Configuración_{ejemplo}$ = {(Calls, activa), (GPS, inactiva), (Screen, activa), (Basic, inactiva), (Colour, activa), (High resolution, inactiva), (Media, activa), (Camera, activa), (MP3, activa)}

Tabla 1: Configuración de ejemplo

En la Figura 11 se muestra el Modelo de Configuración asociado a la $Configuracion_{ejemplo}$. Las características verdes representan características activas, mientras que las características rojas representan características inactivas.

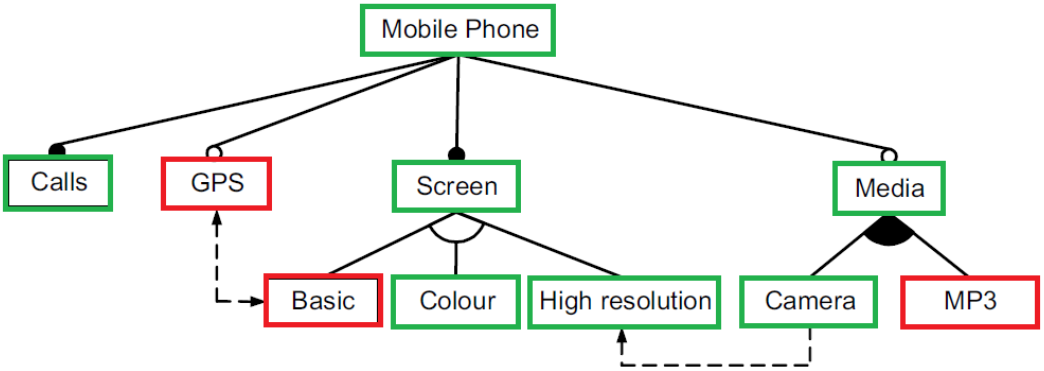


Figura 11: Ejemplo de Modelo de Configuración

2.2.2.3 MODELADO DE RESOLUCIONES

Una resolución es la representación de un conjunto de cambios sobre el estado de distintos elementos del sistema. Es formada por una lista de pares (Característica, Acción). Cada Resolución se asocia con una condición de contexto y representa un cambio en la configuración del sistema, mediante la activación/desactivación de características. La ejecución de una resolución viene determinada por el cumplimiento de la condición.

Siguiendo el Modelo de Características de la Figura 4, podemos encontrar resoluciones de la forma:

$R_{simplephone}$ = {(Calls, activar),(Screen, activar),(Basic, activar)}

$R_{newphone}$ = {(Calls, activar),(Screen, activar),(High resolution, activar)(GPS, desactivar),(Media, activar),(Camera, activar)}

Tabla 2: Resoluciones de Ejemplo

La resolución $R_{simplephone}$ indica que cuando se cumple la condición de contexto "simplephone" se deben activar las características "Calls", "Screen" y "Basic". Por otro lado, la resolución $R_{newphone}$ indica que cuando se cumple la condición de contexto "newphone" las características "Calls", "Screen", "High resolution", "GPS", "Media" y "Camera" deben activarse.

La representación de una resolución se realiza con el uso de configuraciones parciales, es decir, modelos de configuraciones donde se activan/desactivan sólo un subconjunto de las características. A continuación se presenta el ejemplo de resolución $R_{newphone}$(Figura 12):

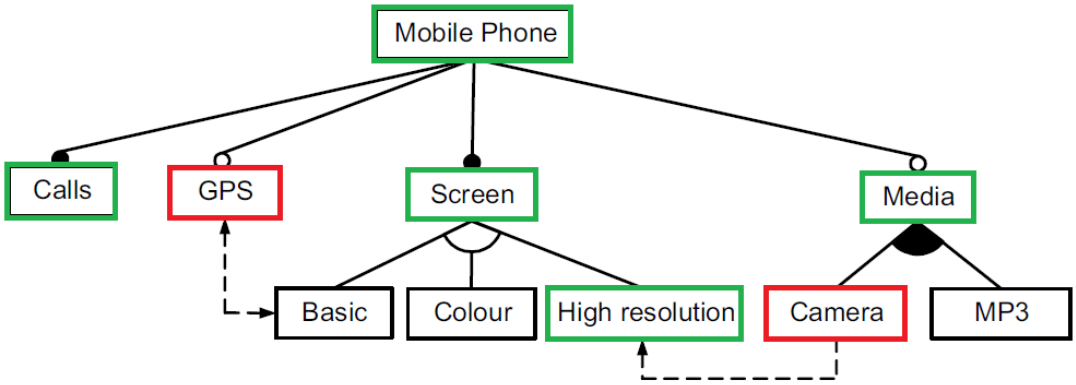


Figura 12: Ejemplo de Configuración Parcial (Resolución)

Se puede ver que la estructura de la configuración del sistema en un momento dado es similar a la estructura de las resoluciones. La diferencia entre ambos conceptos reside en que las configuraciones son representaciones completas del estado del sistema. Mientras que las resoluciones son como acciones que se aplican sobre las configuraciones, bajo unas condiciones, para generar una serie de cambios; es decir, que las resoluciones indican los cambios a realizar en el sistema para pasar de una configuración a otra. El proceso de aplicar resoluciones sobre una configuración es definido como reconfiguración.

2.2.3 MODELADO DEL ESPACIO DE ADAPTACIÓN

El Espacio de Adaptación representa todas las posibles configuraciones que pueden darse en el sistema a partir del modelado de la variabilidad del sistema. El Espacio de Adaptación puede representarse mediante una Máquina de Estados. El Espacio de Adaptación está formado por un conjunto de estados (configuraciones) y sus respectivas transiciones (reconfiguraciones) que permiten navegar entre los estados. A partir de un Modelo de Configuración y un Modelo de Resoluciones es posible obtener automáticamente el Espacio de Adaptación representado como una Máquina de Estados.

La Figura 13 nos da una visión conceptual del proceso de transformación. Podemos observar como a partir de un Modelo de Configuración que representa el estado inicial del sistema, y un Modelo de Resoluciones, es posible obtener mediante una transformación de modelos un Modelo de Máquina de Estados que representa el Espacio de Adaptación del sistema.

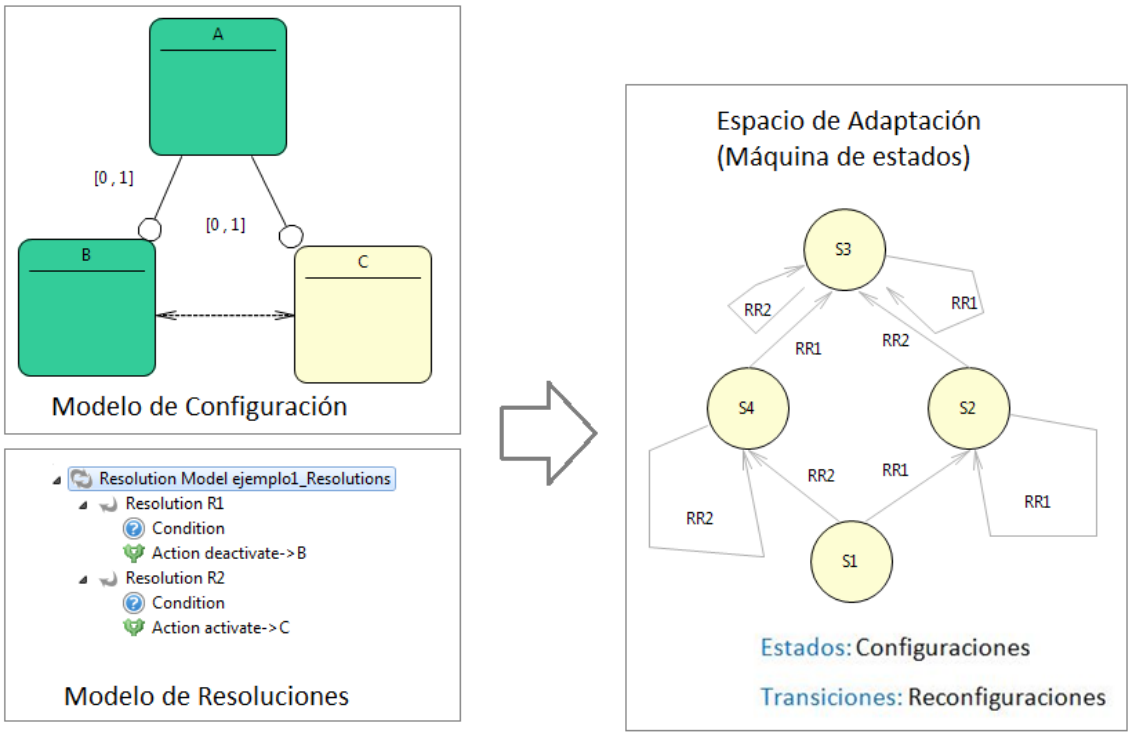


Figura 13: Transformación del modelo de características al espacio de posibilidades

El proceso de generación del Espacio de Adaptación se lleva a cabo mediante la aplicación sucesiva de las resoluciones contenidas en el Modelo de Resoluciones sobre la configuración inicial del sistema. La aplicación de cada una de las resoluciones genera una reconfiguración del sistema. Hay que añadir que es posible que no todas las resoluciones del Modelo de Resoluciones se disparen, ya que solo se disparan aquellas resoluciones cuya condición de contexto se cumpla. En otras palabras, su ejecución vendrá determinada por el cumplimiento de la condición.

Las nuevas Configuraciones (estados de la Máquina de Estados) quedan conectadas al estado inicial que, junto a una resolución, ha generado la nueva configuración.

Como vemos en la Figura 14, a partir del estado A, se han disparado las reconfiguraciones A1, A2, y A3, y se han generado los nuevos estados: B, C, D. Los estados creados a su vez pueden generar nuevos estados a partir de la ejecución de reconfiguraciones. Si se genera un estado asociado a un Modelo de Configuración ya existente no se crea un nuevo estado, sino que se conecta con el estado existente. El espacio de Adaptación crece hasta que para todos los estados se hayan aplicado todas las posibles reconfiguraciones, y no sea posible obtener nuevas configuraciones.

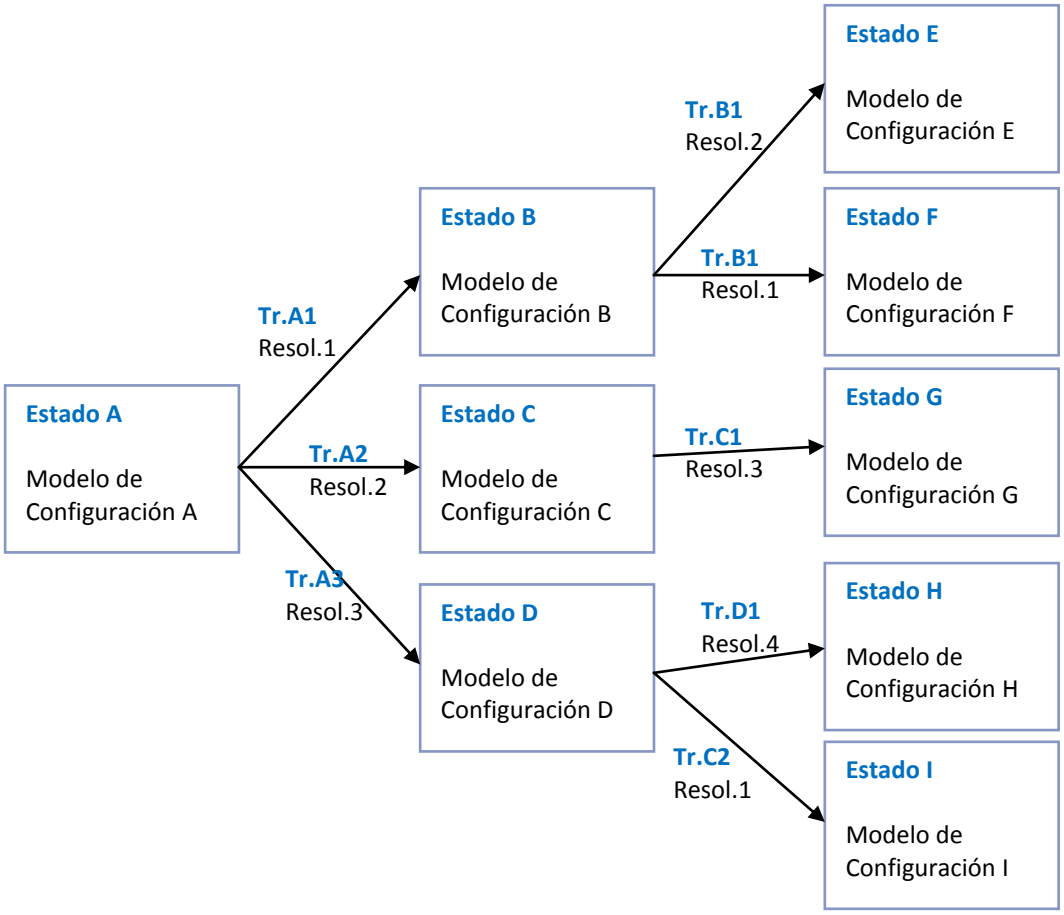


Figura 14: Ejemplo generación de Espacio de Adaptación

3. DESARROLLO DE LA PROPUESTA

Hasta ahora hemos visto como representar el conocimiento de un sistema auto-adaptable mediante el uso de modelos. Estos modelos son consultados y actualizados para adaptar el comportamiento del sistema en ejecución. Todo sistema software requiere evolución y mantenimiento, por ello, el diseñador del sistema debe poder evolucionar los modelos de conocimiento para añadir nuevas capacidades al sistema.

En este capítulo, en primer lugar, vamos a dar una visión general del proceso de evolución de los sistemas auto-adaptables. En segundo lugar, se explica en qué consiste y cómo se ha implementado la última fase de éste proceso, que incluye la validación y refinamiento de los nuevos modelos evolucionados, para asegurar que el sistema se mantiene en un estado consistente después de la evolución.

La implementación de la fase de validación y refinamiento se ha realizado sobre la plataforma Moskitt4SPL [17]. Moskitt4SPL es una herramienta libre y abierta que da soporte al modelado de Líneas de Producto Software (SPL). Esta herramienta está basada en MoSKitt y está construida sobre Eclipse haciendo uso de las tecnologías de las Eclipse Modelling Tools, como EMF, GMF o ATL. Ha sido desarrollada por el Centro de Investigación en Métodos de Producción de Software (ProS) de la Universitat Politécnica de València (UPV).

3.1 VISIÓN GENERAL DEL PROCESO DE EVOLUCIÓN

El objetivo del proceso de evolución es obtener una nueva versión de las descripciones (modelos) del sistema, donde se incorporen los cambios requeridos por la evolución. Es inaceptable evolucionar un sistema a una configuración inválida o inconsistente. Para que esto no ocurra, los cambios deben ser analizados antes de aplicarlos en el sistema en funcionamiento.

La estrategia *Replication with Validation* [18] propone la creación y mantenimiento de copias de los modelos con fines de validación. De manera que los cambios se aplican y analizan en éstas copias primero, y si no se detectan errores, la evolución puede ser aplicada con seguridad en el sistema en funcionamiento. La Figura 15 resume los pasos básicos llevados a cabo en el proceso de evolución. Específicamente, el proceso comprende tres pasos:

Paso 1: Extraer una copia del conocimiento.

Paso 2: Evolucionar el conocimiento a una nueva versión.

Paso 3: Validar la nueva versión evolucionada.

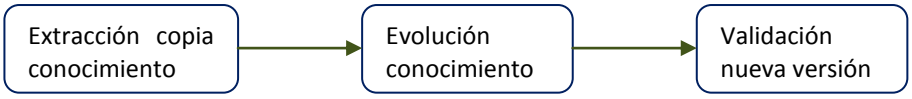


Figura 15: Pasos del Proceso de Evolución

El objetivo de este trabajo final de máster es dar soporte al paso 3 del proceso de evolución: validación de la nueva versión de los modelos evolucionados. Esta fase debe proporcionar mecanismos y herramientas para validar los efectos de la evolución.

En este trabajo final de máster se propone un proceso de validación y la herramienta de soporte para comprobar y garantizar que la evolución no produce errores ni deja al sistema en un estado inconsistente. El proceso de validación comprende dos pasos:

-1. Detección y explicación de errores.

-2. Refinamiento de los modelos para eliminar errores.

3.2 VALIDACIÓN Y REFINAMIENTO DE LOS MODELOS DEL SISTEMA

Los modelos que constituyen el conocimiento de un sistema en tiempo de ejecución son: un modelo de variabilidad (Modelo de Características, FM), un Modelo de Configuración (CM), un modelo de políticas de adaptación (Modelo de Resoluciones, RM) y un Modelo de Arquitectura. En los siguientes apartados describimos el procedimiento de validación de los distintos modelos que constituyen el conocimiento del sistema, exceptuando el Modelo de Arquitectura que queda fuera del alcance de este trabajo.

El primer paso es comprobar la validez del Modelo de Características. Ya que el Modelo de Resoluciones y Modelo de Configuración se definen en términos del Modelo de Características, tras la evolución del Modelo de Características hay que comprobar que las resoluciones y configuraciones siguen siendo válidas en la nueva versión. Además, aunque las resoluciones sean válidas, la aplicación de éstas sobre determinadas configuraciones puede generar configuraciones inválidas. Por ello es esencial garantizar que todas las posibles configuraciones que se generen a partir del Modelo de Resoluciones son válidas. Para ello, obtenemos el espacio de adaptación generado a partir del Modelo de Configuración y el Modelo de Resoluciones, y se validan cada una de las posibles configuraciones. La Figura 16 resume las fases del proceso de validación del Modelo de Características evolucionado.

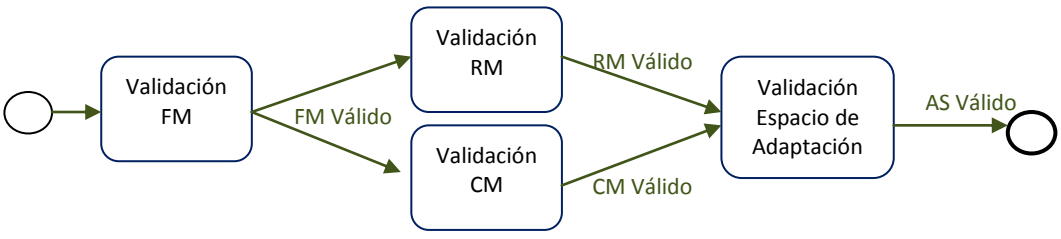


Figura 16: Pasos propuestos en el Proceso de Validación.

3.2.1 VALIDACIÓN Y REFINAMIENTO DEL MODELO DE CARACTERÍSTICAS

Ante la aparición de nuevos requisitos es necesario evolucionar los Modelos de Características. Los cambios posibles son la adición o eliminación de características, de relaciones entre características (obligatoria, opcional, alternativa o exclusiva), y de restricciones entre características (requiere o excluye).

La herramienta Moskitt4SPL incorpora un editor gráfico (Figura 17) de Modelos de Características. En modelos grandes es prácticamente imposible detectar errores de forma visual, por tanto se requieren mecanismos para automatizar la detección y diagnóstico de errores en Modelos de Características. En este trabajo se ha extendido la herramienta Moskitt4SPL para incluir estos mecanismos de análisis y refinamiento de Modelos de Características. A continuación se explica con más detalle en qué consisten estos mecanismos y cómo se han implementado.

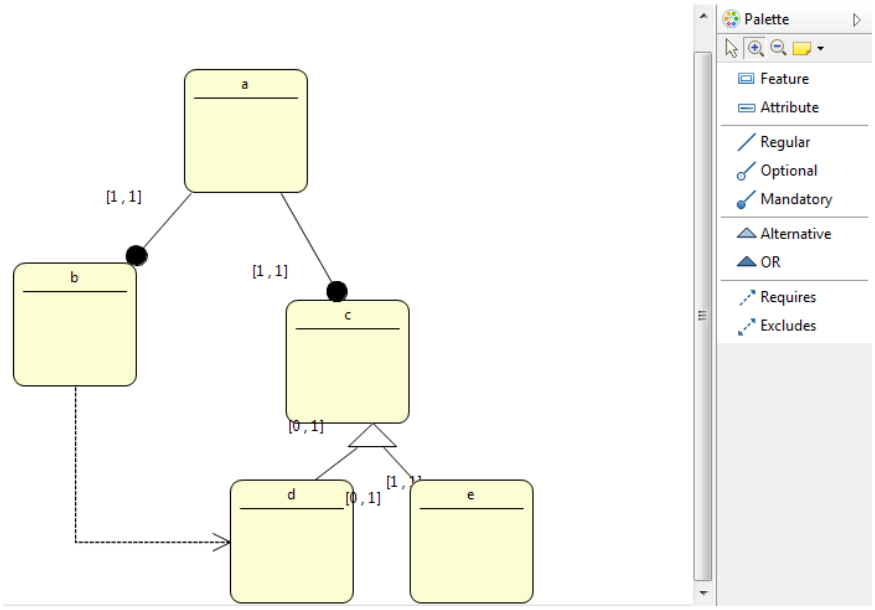


Figura 17: Editor gráfico de Modelos de Características en MOSKitt4SPL

El proceso de validación del Modelo de Características comprende 2 fases: la fase de análisis, en la cual se detectan los errores o anomalías y se realizan las explicaciones sobre éstos; y la fase de refinamiento, en la cual se aplican refinamientos al modelo para corregir las anomalías encontradas automáticamente.

ANÁLISIS DEL MODELO DE CARACTERÍSTICAS

Para dar soporte al proceso de análisis del Modelo de Características se utiliza el framework FAMA [19]. FAMA es un framework que permite analizar automáticamente Modelos de Características a través de diversas operaciones:

- *Validation:* dado un Modelo de Características, éste método devuelve si el modelo es válido o no. Concretamente comprueba si un modelo no es vacío, o en otras palabras, si puede generarse al menos un producto a partir de él.

- *Error detection*: esta operación comprueba si hay anomalías en el Modelo de Características, y si las hay, las devuelve como resultado.

- *Error explanations*: cuando un modelo tiene anomalías, esta operación busca explicaciones para esas anomalías.

Haciendo uso de estas operaciones, se ha implementado un análisis de las posibles anomalías que pueden darse en los Modelos de Características [16]:

- *El Modelo de Características es vacío:* un modelo es vacío si no es posible definir una configuración (producto) a partir de éste. Podemos detectar si un modelo es vacío mediante la operación *Validation* de FAMA.

- *Característica Falso-Obligatoria*: una característica es *falso-obligatoria* si se incluye en todos los productos aunque no se haya modelado como tal.

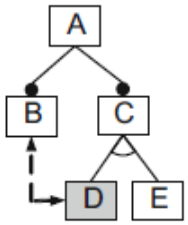
- *Característica muerta*: una característica está muerta si no puede aparecer en ninguno de los productos de la Línea de Producto Software. Las características muertas están causadas por un mal uso de restricciones *cross-tree*.

Las últimas dos anomalías se detectan con la operación *ErrorDetection* de FAMA, y se puede obtener la explicación de porqué el modelo es inválido con la operación *ModelExplanations* de FAMA.

REFINAMIENTO DEL MODELO DE CARACTERÍSTICAS:

Un refinamiento consiste en una serie de modificaciones aplicadas al Modelo de Características con el fin de eliminar las anomalías encontradas en la fase de análisis. Mediante el resultado de las operaciones de análisis de FAMA, en este trabajo final de máster se han implementado una serie de refinamientos:

- Eliminar una relación *cross-tree*: es posible que para evitar características muertas o relaciones *Falso-Obligatoria,* sea necesario eliminar una relación *excluye* o *requiere e*ntre dos características. En la Tabla 3 se muestran una serie de ejemplos:



Anomalías:
- False-mandatory Feature: E
- Dead Feature: D

Refinamientos posibles:
- Eliminar B–excluye–D

Dada la relación Excluye entre las características B y D, y la relación Obligatoria de A y B, que dada la activación de A, implica la activación de B; en el Alternativo de C con D y E, la característica D nunca se activará, siendo una característica muerta, y E siempre se activará, siendo una característica Falso-Obligatoria. Por ello si eliminamos la relación Excluye

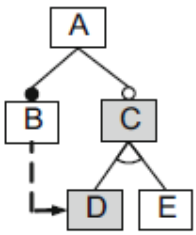
	evitamos la anomalía.
	**Anomalías:** *- False-mandatory Feature: D* *- Dead Feature: E* **Refinamientos posibles:** - Eliminar B–requiere–D Este caso es similar al anterior, pero en este caso, la relación Requiere entre B y D, implica que en el Alternativo de C con D y E, siempre se active D y nunca E.

Tabla 3: Ejemplos Refinamiento del Modelo de Características: eliminar una relación cross-tree

- <u>Cambiar una relación entre características padre-hijo:</u> también es posible cambiar una relación entre una característica padre y una característica hija. En estos casos se opta por hacer la relación menos estricta, cambiando relaciones del tipo *Obligatoria* (donde obligatoriamente, si se activa la característica padre, debe activarse la hija) a una relación *Opcional* (donde la activación de la característica hija es opcional). En la Tabla 4 se muestran una serie de ejemplos:

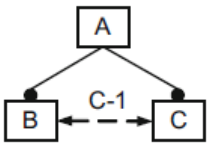
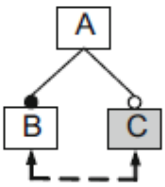
	**Anomalías:** *- Feature Model is void* **Refinamientos posibles:** - Cambiar A–Obligatorio–B por un Opcional. - Cambiar A–Obligatorio–C por un Opcional. En este ejemplo, dadas las relaciones Obligatorio de B y C con A, siempre deben estar activadas B y C cuando A lo esté. Pero dada la relación Excluye entre B y C esto no sería posible. Para evitar que el modelo sea vacío se puede cambiar alguna de las dos relaciones Obligatorio por un Opcional.
	**Anomalías:** *- DeadFeature: c* **Refinamientos posibles:** - Cambiar A– Obligatorio–B por un Opcional . En este ejemplo, como B siempre se activará con A, y tiene una relación Excluye con C, C siempre está inactiva, por ello es una característica muerta. Cambiando la relación Obligatorio de A y B por un Opcional, B no se activará siempre con A, y C podrá activarse.

Tabla 4: Ejemplos Refinamiento del Modelo de Características: cambiar una relación entre características padre-hijo

En la Tabla 5 presentamos los ejemplos anteriores con la anomalía encontrada y el conjunto de refinamientos posibles para corregir la anomalía:

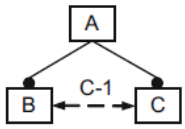
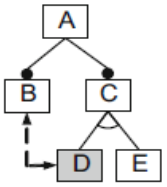
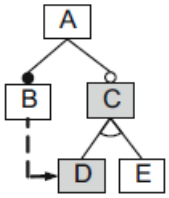
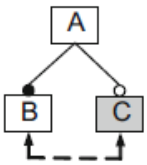
Ejemplo	Anomalías	Refinamientos Posibles
	*- Feature Model is void*	- Eliminar B– Excluye –C - Cambiar A– Obligatorio –B por un Opcional. - Cambiar A– Obligatorio –C por un Opcional.
	*- False-mandatory Feature: e* *- Dead Feature: d*	- Eliminar B– Excluye –D - Cambiar A– Obligatorio –B por un Opcional.
	*- False-mandatory Feature: d* *- Dead Feature: e*	- Eliminar B–Requiere–D - Cambiar A– Obligatorio –B por un Opcional.
	*- Dead Feature: c*	- Eliminar B–Excluye–C - Cambiar A– Obligatorio –B por un Opcional.

Tabla 5: Relación Anomalías - Refinamientos del Modelo de Características

La Figura 18 muestra un resumen de los pasos del proceso de validación del Modelo de Características. En primer lugar se comprueba si el Modelo de Características es vacío mediante la operación *Void FM* de FAMA. Si el modelo es válido (no es vacío), entonces se llama a la operación *AnomaliesDetection*, para comprobar si existen anomalías en el modelo. Si no existen anomalías el modelo se considera válido. Si el modelo es vacío o se han encontrado anomalías, podemos obtener explicaciones sobre estos errores con la operación *ModelExplanations*. A partir de estas explicaciones se ha realizado la implementación de los refinamientos explicados anteriormente (*RefactorModel*).

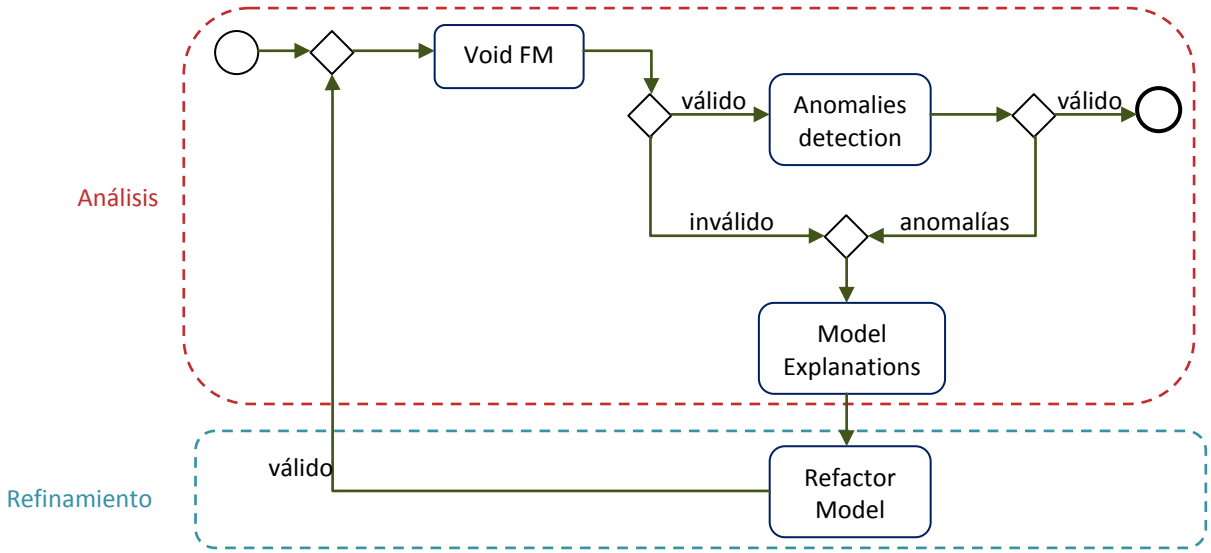


Figura 18: Validación del Modelo de Características

IMPLEMENTACIÓN EN LA HERRAMIENTA

En la herramienta Moskitt4SPL, existen opciones en el menú contextual de los Modelos de Características para la validación ("*ValidateModel*") y explicación de errores ("*Detect Error onModel*" y "*Explain Errors on Model*"), pero sólamente indican si el modelo es válido o no, y los errores encontrados por separado. En la implementación realizada para la validación y refinamiento de los Modelos de Características en este trabajo se ha añadido e implementado en su menú contextual las siguientes opciones:

- *Validate and Refine FM*: en la que se comprueba si el modelo es vacío o existen anomalías, y además de mostrar las explicaciones de estas anomalías, se ofrecen los refinamientos posibles para solucionar la anomalía encontrada. El refinamiento seleccionado se aplica sobre una copia del modelo. La Figura 19 muestra el menú contextual de un modelo de Características, donde se ilustra la opción *"Validate and Refine FM"* añadida, y la ventana donde se muestran las anomalías encontradas y los refinamientos implementados.

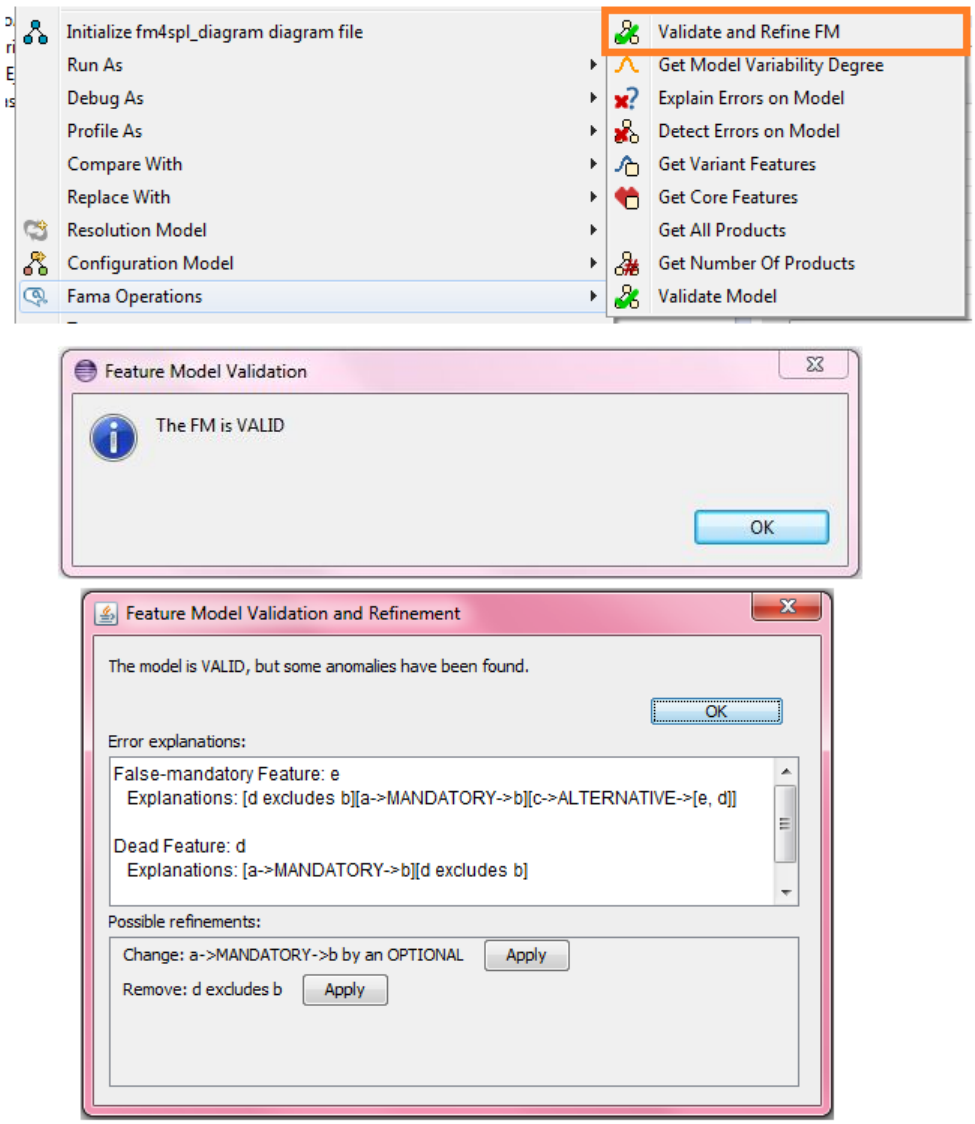


Figura 19: Ventana explicación anomalías y refinamientos de un Modelo de Características (Opción: Validate and Refine FM)

- *ConfigurationModel / Validate and Create new CM*: en la herramienta es posible crear un nuevo Modelo de Configuración a partir de un Modelo de Características, mediante la opción *Create New Configuración Model*. En este trabajo se ha implementado la opción*Validate and Create new CM* que permite realizar la validación del Modelo de Características antes de generar el Modelo de Configuración. De esta manera si el Modelo de Características es inválido puede aplicarse un refinamiento y si es válido se puede generar el Modelo de Configuración de manera segura. En la Figura 20, se muestra la opción en el menú contextual, y las 2 posibles ventanas resultantes según la validez del Modelo.

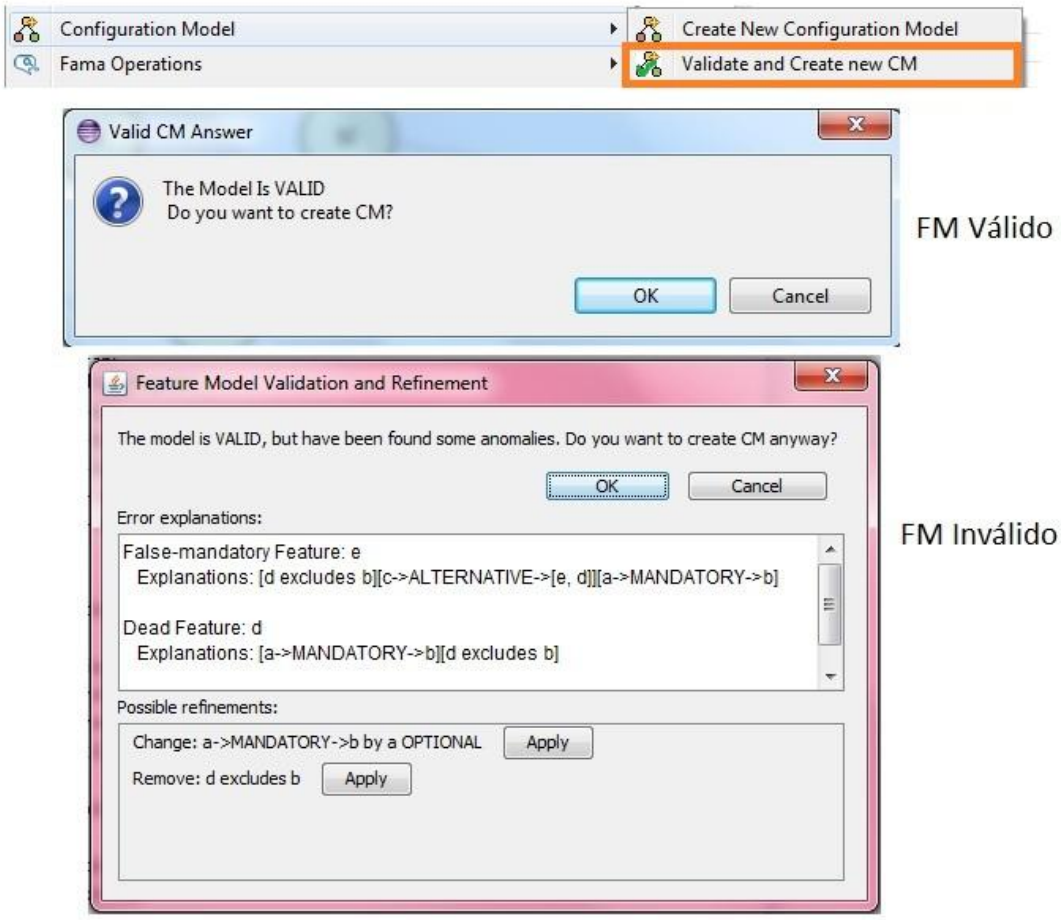


Figura 20: Ventana explicación anomalías y refinamientos de un Modelo de Características (Opción: Validate and Create new CM)

- *ResolutionModel / Validate and Create new RM:* del mismo modo que con el Modelo de Configuración, un Modelo de Resoluciones también puede generarse a partir de un Modelo de Características. La opción *Validate and Create new RM*, permite la validación y refinamiento del Modelo de Características antes de la generación de un modelo de Resoluciones. En la Figura 21 se muestra la opción en el menú contextual y las 2 posibles ventanas resultantes.

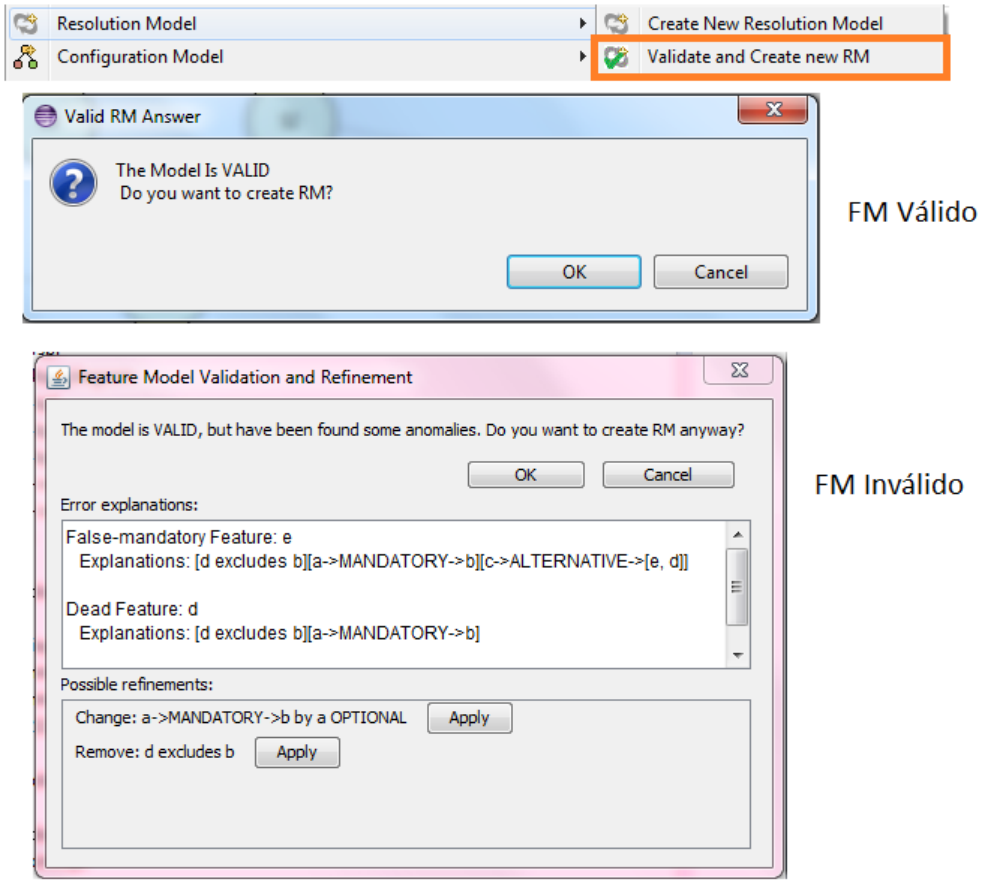


Figura 21: Ventana explicación anomalías y refinamientos de un Modelo de Características (Opción: Validate and Create new RM)

Dado un modelo inválido, si se selecciona un refinamiento, éste se aplica sobre una copia del modelo original. La herramienta pide al usuario un nombre para el nuevo modelo, como se muestra en la Figura 22:

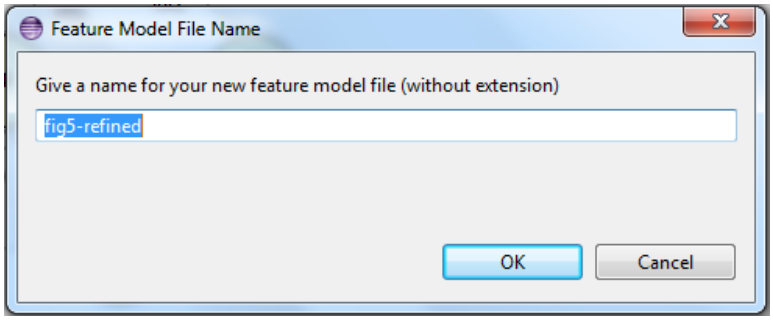


Figura 22: Ventana asignación nombre de nuevo Modelo de Características

3.2.2 VALIDACIÓN Y REFINAMIENTO DEL MODELO DE CONFIGURACIÓN

A partir de un Modelo de Características es posible definir diferentes Modelos de Configuración, que representan configuraciones sobre éste. Por tanto, para crear una configuración, es necesario haber creado previamente un Modelo de Características que sea válido.

Como ya se ha explicado en apartados anteriores, un Modelo de Características especifica toda la variabilidad presente en una Línea de Productos. Por ello, las distintas configuraciones sobre el Modelo de Características también se llaman productos generados, ya que cada una de las configuraciones representa un producto posible de la Línea de Productos.

La herramienta MOSKitt4SPL ofrece un editor gráfico para crear Modelos de Configuraciones (Figura 23)

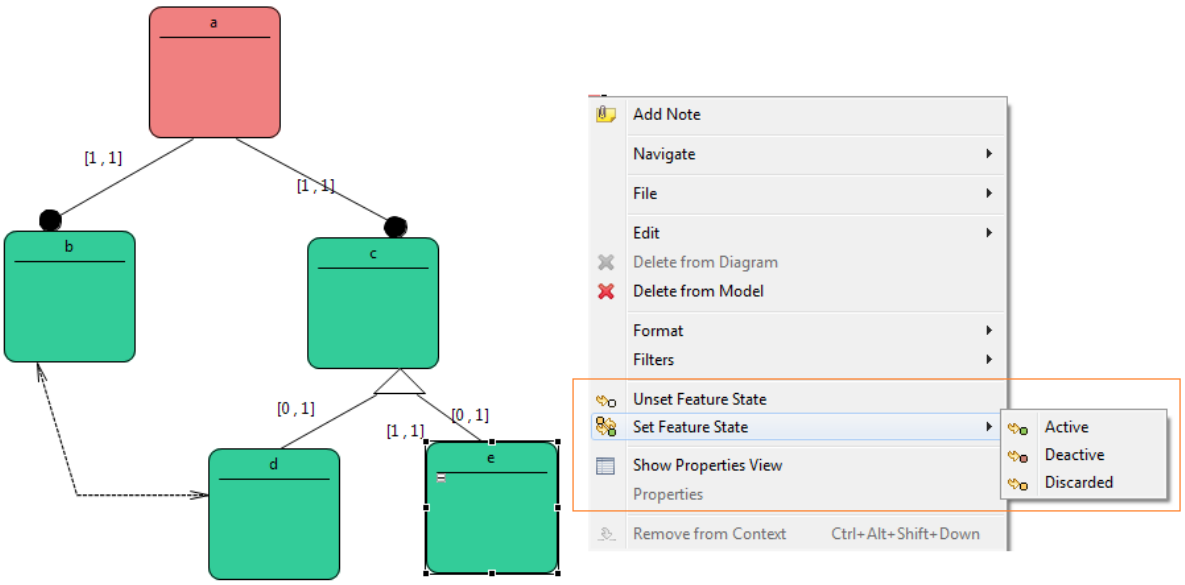


Figura 23: Editor Gráfico de Modelos de Configuración

Cada configuración se representa como un conjunto de estados de las características. Los posibles estados definidos en MOSKitt4SPL son *Activo*, *Inactivo* y *Descartado*. En la creación de una configuración se pueden activar/desactivar características y que el modelo de configuración resultante sea inválido. Para evitar esta situación, se ha realizado la implementación en Moskitt4SPL de una serie de refinamientos que se explican a continuación.

REFINAMIENTOS POSIBLES EN UN MODELO DE CONFIGURACIÓN:

De igual manera que con el modelo de Características, es posible realizar refinamientos sobre los Modelos de Configuración. Concretamente estos refinamientos se centran en modificar el estado de las características para que el modelo resultante sea válido. A continuación en la Tabla 6 se ilustran algunos ejemplos de Modelos de Configuración con anomalías y los posibles refinamientos:

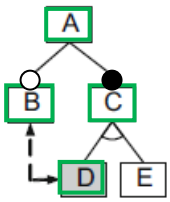
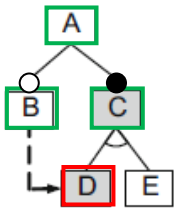
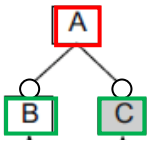
Ejemplo	Anomalías	Refinamientos Posibles
	B y D están Activados simultáneamente. Esta configuración es inválida por la relación de exclusión que existe entre B y D.	- Desactivar D - Desactivar B
	B está activado y D está desactivado simultáneamente. Por tanto la configuración es inválida porque la relación de inclusión que existe entre B y D obliga a seleccionar D siempre que B esté seleccionada.	- Activar D - Desactivar B
	Las características B y C están activadas. Mientras que A no lo está. Esta configuración es inválida, ya que las características hijas con una relación Obligatoria con su característica padre, sólo pueden estar activadas cuando su padre también lo éste.	- Activar A

Tabla 6: Relación Anomalías - Refinamientos del Modelo de Configuración

La herramienta Moskitt4SPL, que se extiende en éste trabajo, incluía la validación de Modelos de Configuración, mediante la opción del menú contexual "*ValidateProduct*", que únicamente indica si el modelo es válido o no, y la explicación de errores, con la opción "Explain Errors on Product". En este trabajo se ha añadido una nueva opción en el menú contextual del Modelo de Configuración. Esta nueva opción *"Validate and Refine Product*" nos da paso a una ventana donde se indica si el modelo es válido, y si no lo es, se listan las anomalías y errores detectados, y se sugieren refinamientos para corregirlos. En la Figura 24 se muestra las posibles ventanas resultantes de aplicar la acción *"Validate and Refine Product*".

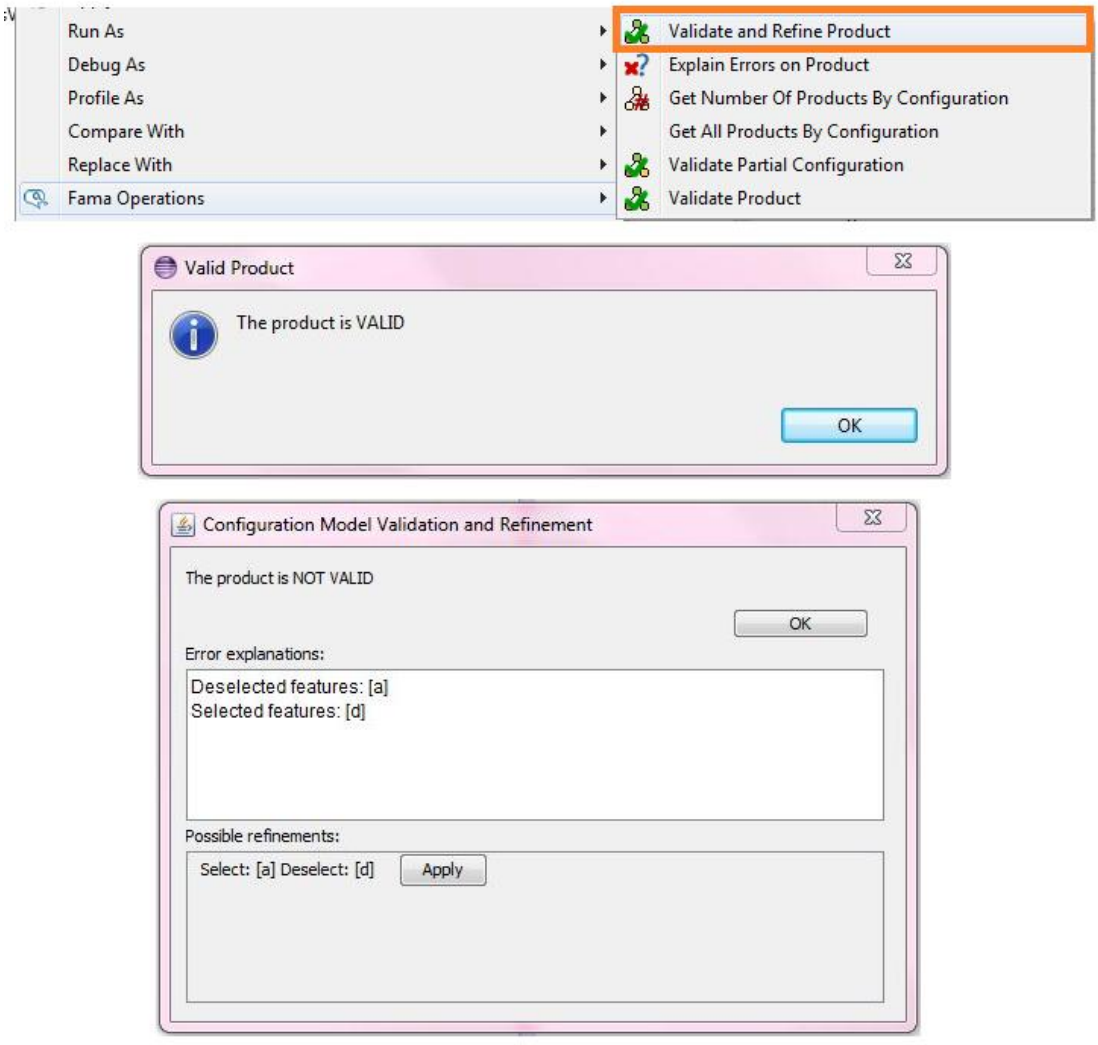


Figura 24: Ventana explicación anomalías y refinamientos de un Modelo de Configuración

En el caso de los Modelos de Configuración, la validez del modelo se comprueba con la operación de FAMA: *ValidProduct*; y las explicaciones de errores (*Error explanations* en la ventana) consisten en una lista de características seleccionadas y deseleccionadas. Estas listas se

han obtenido mediante la operación *ExplainInvalidProduct* de FAMA. Los refinamientos que se han implementado consisten en seleccionar o deseleccionar aquellas características que lo requieran, según el resultado obtenido en la operación. Por ejemplo, si obtenemos "*Deselectedfeatures: [a]*", esto indica que la característica *a* está deseleccionada, pero debería estar seleccionada. Por ello el refinamiento que se ofrece es "*Select: [a]".*

Si el Modelo de Configuración es inválido, el refinamiento seleccionado se aplica sobre una copia del modelo. En este caso, la herramienta pide al usuario un nuevo nombre para el nuevo modelo, mediante la ventana mostrada en la Figura 25.

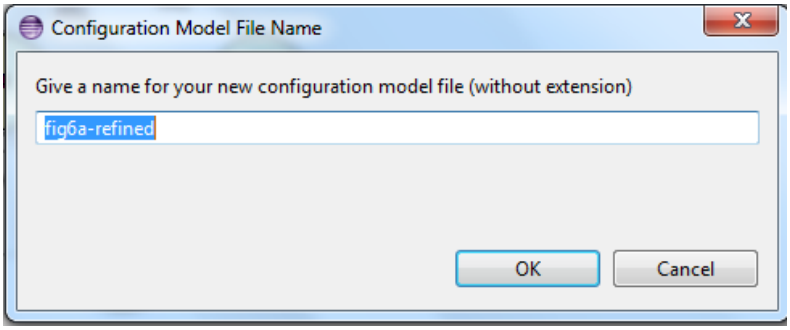


Figura 25: Ventana asignación nombre de nuevo Modelo de Configuración

3.2.3 VALIDACIÓN Y REFINAMIENTO DEL MODELO DE RESOLUCIONES

Un Modelo de Resolución está formado por un conjunto de Resoluciones, que expresan reconfiguraciones sobre las distintas configuraciones posibles del sistema. Las Resoluciones se definen en base a 3 elementos:

-Condición: condición de contexto que dispara la ejecución de la resolución.

- Acción: cambio de estado del Modelo de Características asociado. Puede ser Activar o Desactivar.

- Guarda: condicionamiento de ejecución de una resolución en un estado (de la Máquina de Estados).

Por tanto, el proceso de evolución de la nueva versión del Modelo de Resoluciones consiste en la adición o eliminación de resoluciones o acciones a las resoluciones, y modificación de condiciones o guardas. La herramienta Moskitt4SPL nos ofrece un editor textual del Modelo de Resoluciones (Figura 26):

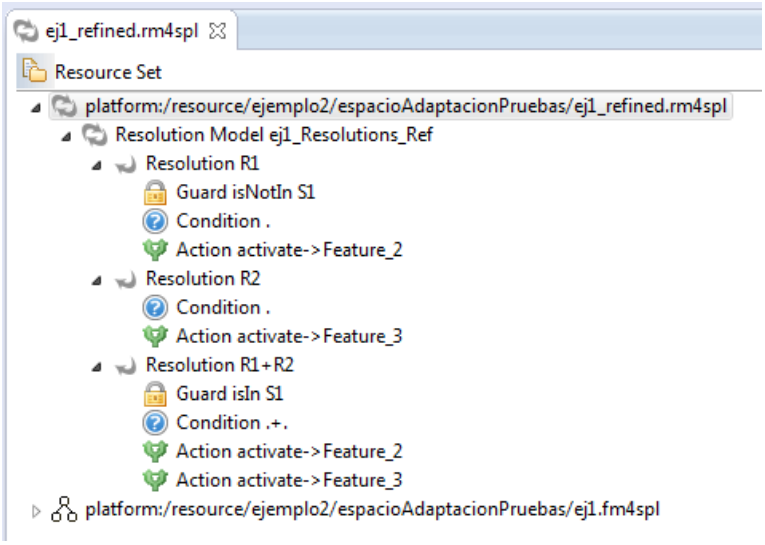


Figura 26: Editor Textual del Modelo de Resoluciones

Cuando se evoluciona el Modelo de Características, debe comprobarse que el modelo de resoluciones sigue siendo válido, ya que las resoluciones se definen en términos de características del Modelo de Características. Las resoluciones se presentan mediante configuraciones parciales. La Figura 27 explica el proceso de validación del Modelo de Resoluciones. En primer lugar se comprueba si el Modelo de Resoluciones es válido, con la operación *Valid Partial Configuration*. Si el modelo es inválido puede ser a causa del Modelo de Características o bien a causa de las Resoluciones del modelo. En este trabajo se ha implementado la segunda opción, de manera que se obtienen las explicaciones de las anomalías que hacen el modelo inválido, con la operación *Config. Explanations*, y se ofrecen una serie de refinamientos para las Resoluciones (*Refactor Resolutions*).

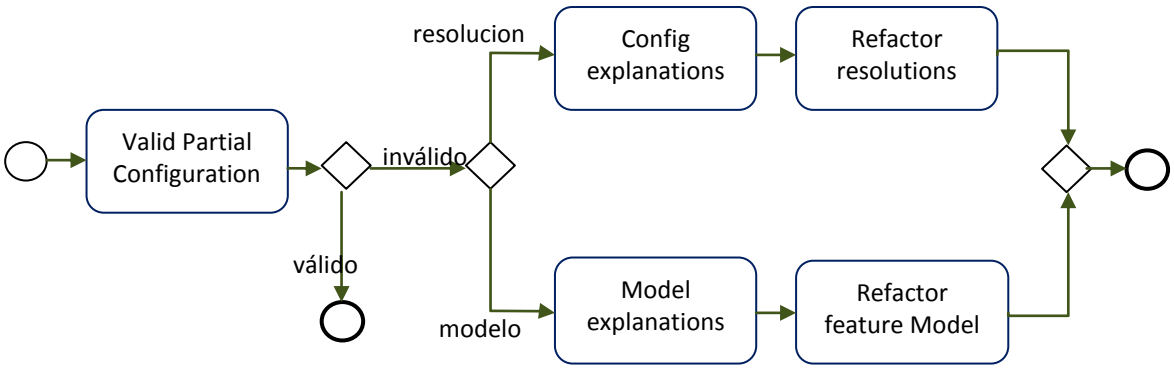


Figura 27: Validación del Modelo de Resoluciones

En la herramienta se ha implementado la validación y refinamiento de Modelos de Resoluciones. Concretamente se tienen en cuenta los siguientes errores que causan que un Modelo de Resoluciones sea inválido:

- Activación de dos características que mantienen una relación de *exclude*: si dos características tienen una relación de *exclude* entre ellas, entonces no es posible su activación simultánea. Puede estar activada una de las dos, o ninguna.

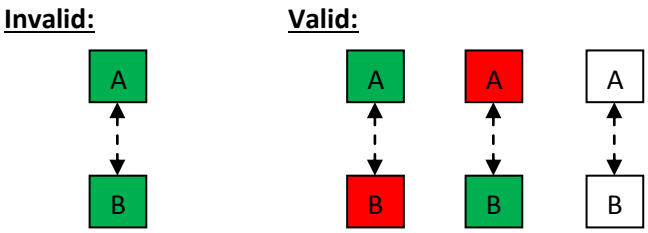


Figura 28: Ejemplos Relaciones Exclude (Validación Modelo Resoluciones)

- Activación de una característica con una relación *requires*, donde la segunda característica no está activada: si una característica tiene una relación *requires* con otra, entonces al activarse la primera necesita obligatoriamente que se active la segunda.

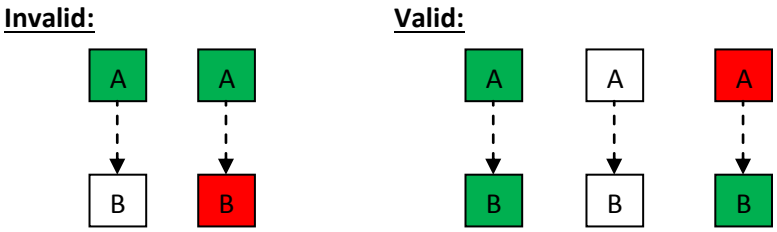
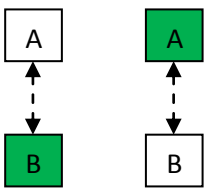


Figura 29: Ejemplos Relaciones Requires (Validación Modelo Resoluciones)

En la Tabla 7 se muestra un resumen de las anomalías que pueden encontrarse en las Resoluciones y los refinamientos que han sido implementados para solucionarlas.

Ejemplo	Anomalía	Refinamientos Posibles
	Activación de dos características que mantienen una relación de *exclude.*	-Eliminar acción: activar A -Eliminar acción: activar B 

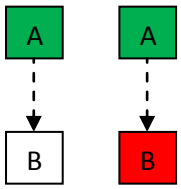
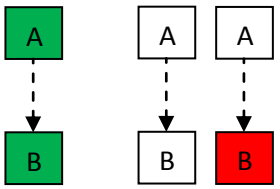
	Activación de una característica con una relación *requires*, donde la segunda característica no está activada. Pueden haber dos opciones: que no se haya definido un estado para la característica o que se haya marcado como desactivada.	-Añadir acción: activar B - Eliminar acción: activar A 

Tabla 7: Relación Anomalías - Refinamientos del Modelo de Resoluciones

En todos los casos presentados en la tabla anterior también se ha implementado como refinamiento la eliminación por completo de la resolución inválida.

IMPLEMENTACIÓN EN LA HERRAMIENTA

En la herramienta Moskitt4SPL se ha implementado íntegramente la validación y refinamiento de los Modelos de Resoluciones. Se ha añadido la opción "*Validate Resolutions"* en el menú contextual de los Modelos de Resoluciones.

Para cada Resolución se buscan las anomalías indicadas en la Tabla 7, y en caso de encontrar alguna, se indica la explicación y una lista desplegable de refinamientos posibles. En la Figura 30 se muestra la ventana resultante de la operación "*Validate Resolutions*".

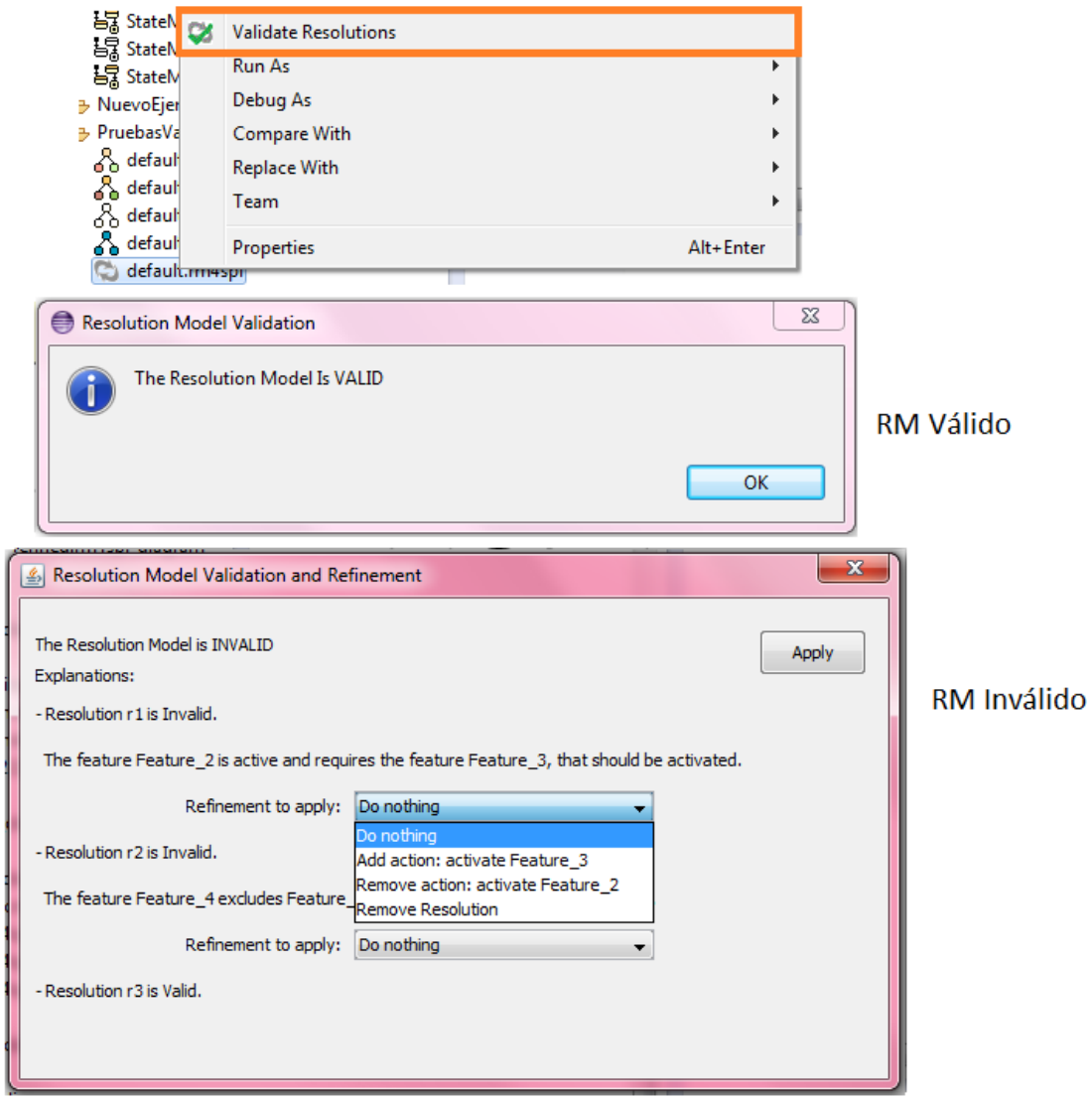


Figura 30: Ventana resultante de la validación del Modelo de Resoluciones

Si el usuario decide aplicar un refinamiento, éste se aplica sobre una copia del modelo. En la Figura 31 se muestra la ventana para dar nombre al nuevo Modelo de Resoluciones.

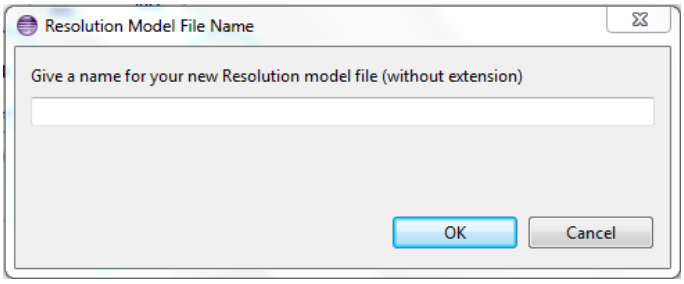


Figura 31: Ventana asignación nombre de nuevo Modelo de Resoluciones

3.2.4 VALIDACIÓN Y REFINAMIENTO DEL ESPACIO DE ADAPTACIÓN

Por último, una vez obtenidos un Modelo de Características, un Modelo de Configuración y un Modelo de Resoluciones válidos, debemos comprobar que todas las posibles configuraciones que se pueden generar a partir de esa especificación son válidas. Es decir, que el Espacio de Adaptación generado a partir de esos modelos es válido. Esto se debe a que una resolución puede ser válida de acuerdo a la definición de un Modelo de Características, pero sin embargo al aplicarla sobre una determinada configuración, puede generar una configuración inválida. En la Figura 32 se muestra un ejemplo de esta situación. Dado un Modelo de Características válido, se ha generado un Modelo de Configuración donde las características A y B están activadas, y un Modelo de Resoluciones con una Resolución R1 que activa C si se cumple su condición. Ambos modelos son válidos, pero cuando se genera el Espacio de Adaptación, el estado resultante de aplicar la Resolución R1 al Modelo de Configuración es inválido, ya que se genera un nuevo estado S2 donde las características B y C tienen una relación de exclusión y están activadas simultáneamente. Y por tanto el espacio de adaptación también es inválido.

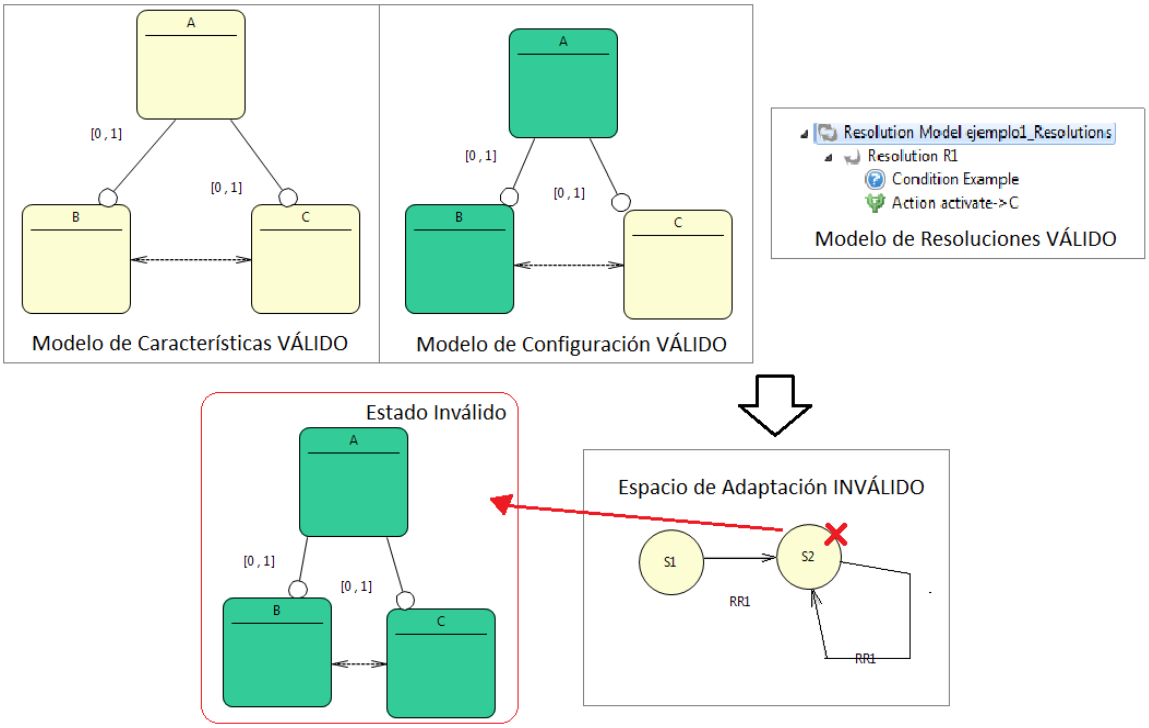


Figura 32: Ejemplo Generación de un Espacio de Adaptación no válido

El primer paso de la validación y refinamiento del Espacio de Adaptación es obtener la validez de cada una de las configuraciones. Si se encuentra una configuración inválida entonces existen dos posibilidades: (1) modificar el Modelo de Características, o (2) modificar el Modelo de Resoluciones. En este caso se ha optado por modificar el Modelo de Resoluciones. En la Figura 33 se muestra este proceso.

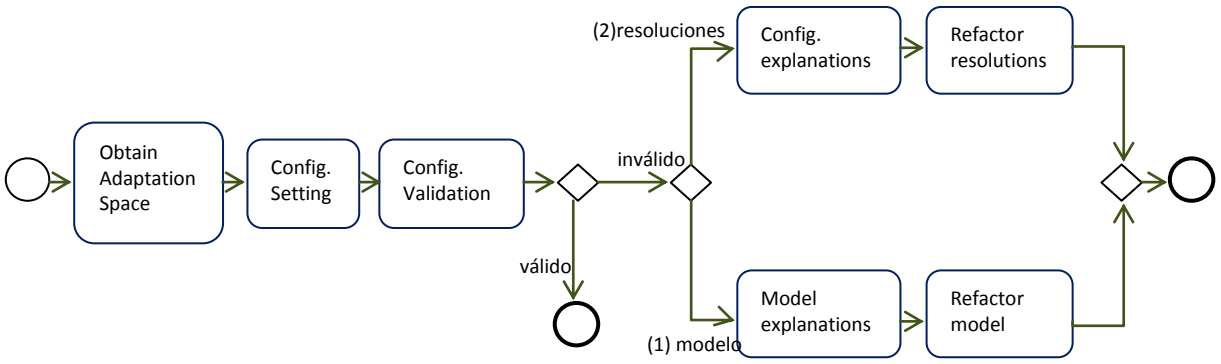


Figura 33: Validación del Espacio de Adaptación

IMPLEMENTACIÓN EN LA HERRAMIENTA

Para la implementación de la validación del Espacio de Adaptación en Moskitt4SPL, se ha generado para cada estado de la Máquina de Estados, la configuración correspondiente y se ha comprobado su validez, mediante la operación *ValidProduct* de FAMA. Si la Máquina de Estados tiene 1 o más estados inválidos, entonces el Espacio de Adaptación es inválido.

En MOSKitt4SPL se ha incorporado el mecanismo de validación y refinamiento del Espacio de Adaptación. Los objetivos del refinamiento son:

- *Evitar reconfiguraciones no seguras*: Se consideran configuraciones no seguras aquellas que llevan al sistema de una configuración válida a una no válida. Estas configuraciones son indeseables y deben ser eliminadas de la especificación.
- *Evitar configuraciones inalcanzables*: Cuando algunas reconfiguraciones son eliminadas, otras configuraciones pueden ser inalcanzables. Para no perder capacidad de reconfiguración, el refinamiento garantiza que todas las configuraciones son alcanzables con seguridad.

Seguidamente se explica en detalle el refinamiento implementado para el Espacio de Adaptación.

ALGORITMO DE REFINAMIENTO DEL ESPACIO DE ADAPTACIÓN

Para la implementación del refinamiento del Espacio de Adaptación vamos a basarnos en el algoritmo presentado en [20]. La Figura 34 muestra éste algoritmo:

Algorithm 1 Safe Reconfigurations and Reachability Refinement

Require: a set S = {$S1$, ... , Sn} ∈ PS and a set R= {$R1$, ... , Rn} of resolutions

Ensure: S without invalid configurations and unsafely unreachable configurations; R without unsafe reconfigurations.

```
1: for all s ∈ S do
2:    R_out ⇐ reconfigurations triggered from s
3:    for all Ri ∈ R_out do
4:       sr ⇐ configuration reachable by Ri
5:       if sr is invalid then
6:          Remove Ri by modifying the guard: [!s] Ri
7:          R_out−sr ⇐ reconfigurations triggered from sr
8:          for all Rj ∈ R_out−sr do
9:             Create a new resolution: [sr] Ri + Rj
10:         end for
11:      end if
12:   end for
13: end for
```

Figura 34: Algoritmo de refinamiento del Espacio de Adaptación

Empezando por la configuración inicial, el refinamiento comprueba iterativamente las reconfiguraciones alcanzables en cada configuración. Si alguna reconfiguración alcanza una configuración inválida, ésta reconfiguración se elimina. El refinamiento introduce guardas para aquellas resoluciones que lanzan reconfiguraciones no seguras, con el objetivo de evitar éstas de la especificación. Cuando alguna reconfiguración es eliminada puede ocurrir que otra configuración válida sea inalcanzable. Por ello el refinamiento calcula una nueva reconfiguración segura que directamente apunte a la configuración inalcanzable desde el origen valido de la antigua reconfiguración no segura.

Estas nuevas resoluciones son calculadas como la composición secuencial de acciones de la antigua resolución y cada una de las resoluciones lanzadas desde la configuración inválida. La condición de la nueva resolución se compone como la conjunción de condiciones de las resoluciones implicadas en la composición. Esto además introduce guardas a la nueva resolución, con el objetivo de evitar la generación de nuevas configuraciones a partir de la aplicación de ésta resolución a otras configuraciones.

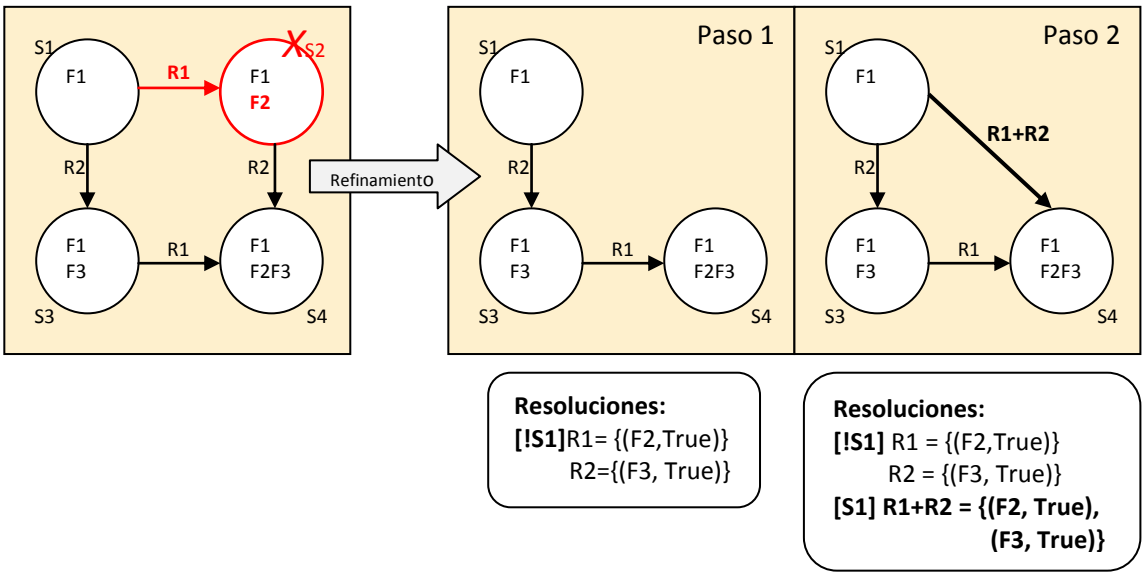


Figura 35: Ejemplo gráfico del refinamiento del Espacio de Adaptación

La Figura 35 muestra la aplicación del refinamiento en un ejemplo concreto. En la configuración S1 la reconfiguración lanzada por R1 es no segura porque lleva al sistema a una configuración inválida (S2). Por ello, el refinamiento evita la ejecución de la resolución R1 en la configuración S1 modificando su guarda. A continuación, para no perder capacidades de reconfiguración, el refinamiento genera una nueva resolución desde S1 a S4, como la composición de las resoluciones anteriores R1 y R2. La guarda de ésta nueva resolución asegura su ejecución sólo en la configuración S1.

En Moskitt4SPL se ha implementado este refinamiento. Concretamente se ha añadido al menú contextual de las Máquinas de Estados la opción "*ValidateAdaptationSpace*", de manera que en la ventana resultante se muestran, por una parte, los estados no válidos, y para cada uno, las explicaciones de por qué la configuración asociada no es válida; y por otra parte, si el usuario lo desea, se puede aplicar el refinamiento para eliminar estos estados inválidos. La Figura 36 muestra el menú contextual de un Espacio de Adaptación y la ventana resultante de aplicar la opción "*ValidateAdaptationSpace*".

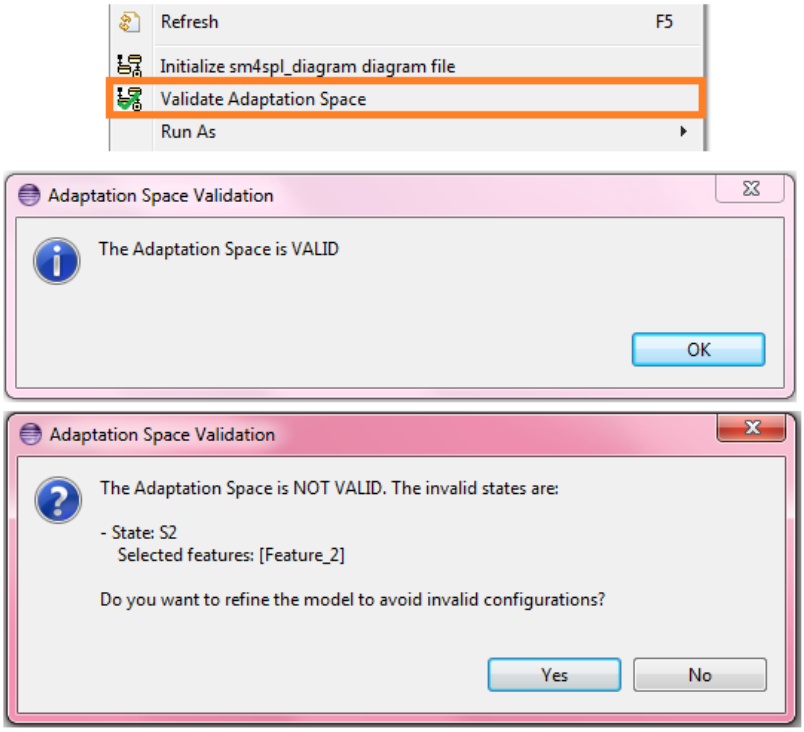


Figura 36: Ventana resultante de la Validación del Espacio de Adaptación

Si el usuario decide aplicar el refinamiento, éste se aplica sobre copias de los Modelos. Para ello la herramienta pide al usuario un nuevo nombre para la copia del Espacio de Adaptación (Figura 37).

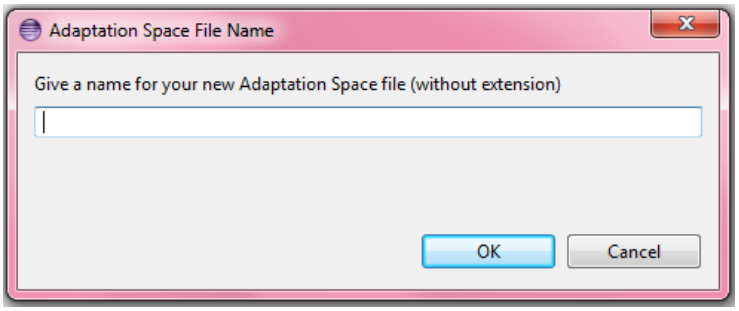


Figura 37: Ventana asignación nombre de nuevo Modelo de Resoluciones y Espacio de Adaptación

4. CASO DE ESTUDIO

En este capítulo se presenta un caso de estudio para ilustrar la aplicación de los conceptos y técnicas propuestos. El objetivo es comprobar el correcto funcionamiento de la herramienta implementada realizando un proceso completo y demostrar su aplicación sobre un problema real.

4.1 DESCRIPCION DEL SISTEMA INICIAL

A continuación vamos a presentar el caso de estudio con el que mostramos la implementación realizada en este trabajo.

El caso de estudio consiste en una aplicación móvil para asistir a los visitantes a la universidad. El sistema ofrece al usuario una serie de servicios: búsqueda de edificios o lugares y obtención de itinerarios para llegar a ellos según la localización del usuario, acceso a las reservas deportivas y a la biblioteca digital.

El servicio de localización del usuario y de obtención de itinerarios siempre está presente en el sistema. La localización puede realizarse usando el GPS del dispositivo o el Wifi de la universidad. Los itinerarios podrán obtenerse en forma textual o mediante un mapa de la universidad. Los servicios de reservas deportivas y de biblioteca estarán disponibles si el usuario lo requiere en la aplicación.

Seguidamente se muestra el Modelo de Características del sistema:

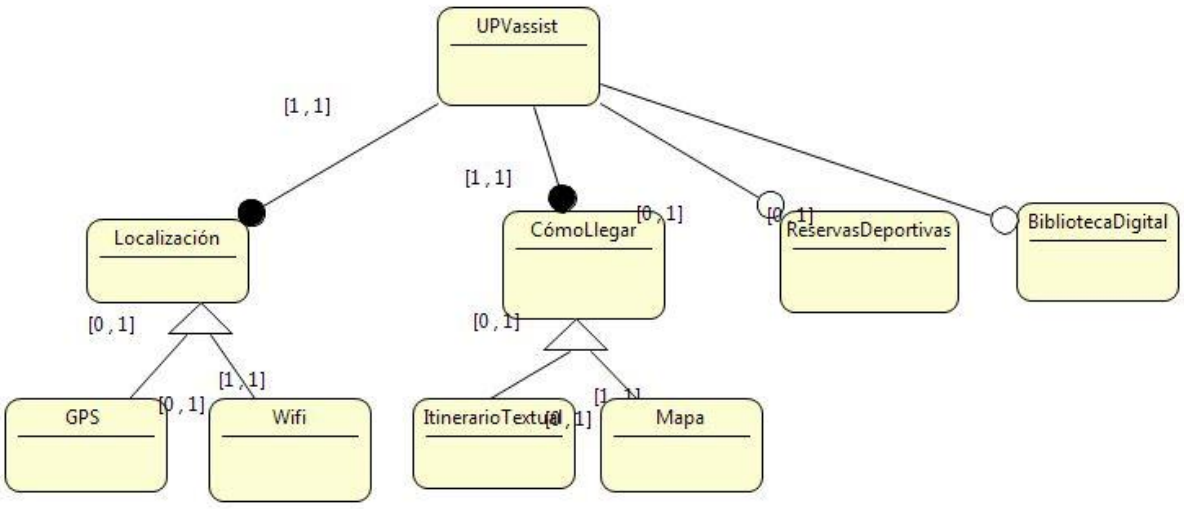


Figura 38: Caso de Estudio: Modelo de Características inicial

Como vemos en la Figura 38 las características *Localización* y *Cómo Llegar* estarán siempre activadas cuando la aplicación *UPVassist* esté en uso, ya que tienen una relación Obligatoria con la característica *UPVassist*. Las Características *Reservas Deportivas* y *Biblioteca Digital* son opcionales. Las características *GPS* y *Wifi* tienen una relación Alternativa con *Localización*, así que se activarán de manera alternativa. Y de igual manera ocurre con las características *ItinerarioTextual* y *Mapa*.

El modelo de Configuración del sistema se obtiene a partir del Modelo de Características, y nos muestra una configuración concreta del sistema. Este es un ejemplo posible:

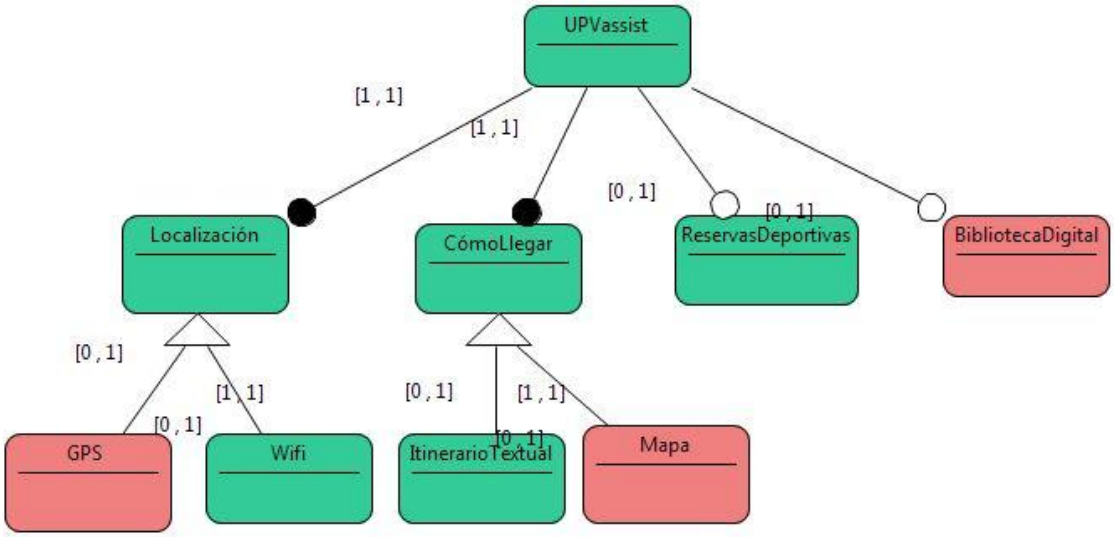


Figura 39: Caso de Estudio: Modelo de Configuración inicial

En la Figura 39, vemos que se ha optado por usar Wifi para la localización, ya que las características *Localización* y *Wifi* están activadas, y los itinerarios en forma textual, ya que *CómoLlegar* y *ItinerarioTextual* están activadas. También está activa la característica *ReservasDeportivas*, lo que indica que en esta configuración el servicio está disponible.

Para modelar el conjunto de cambios que pueden darse en el sistema UPVassist, se ha modelado el siguiente Modelo de Resoluciones de ejemplo:

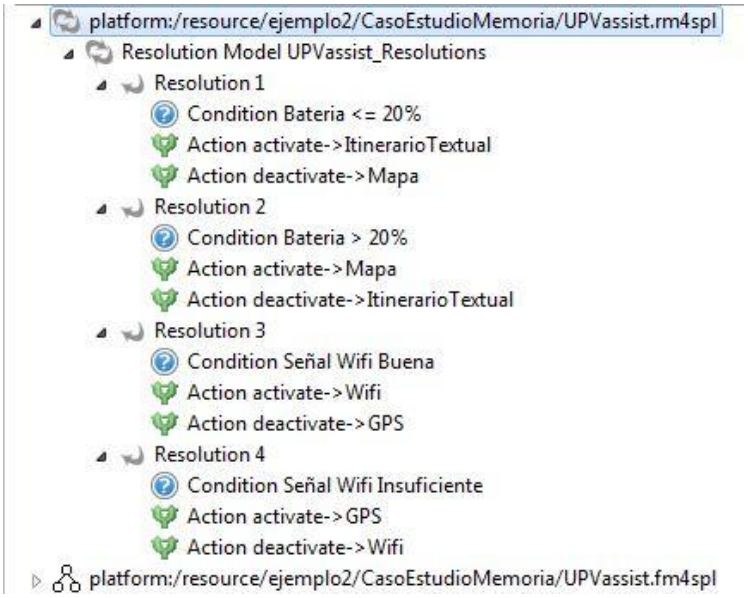


Figura 40: Caso de estudio: Modelo de Resoluciones inicial

En el Modelo de Resoluciones de la Figura 40, podemos observar 4 resoluciones distintas, que modelan 4 posibles cambios en el sistema según ciertas condiciones de contexto:

- Si la batería del dispositivo es menor al 20% se opta por usar los itinerarios en forma textual. En la *Resolución 1* distinguimos la acción de activar *ItinerarioTextual* y la acción de desactivar *Mapa*.

- Si la batería del dispositivo es mayor al 20% se opta por usar los itinerarios en forma de Mapa. En la *Resolución 2* distinguimos la acción de activar *Mapa* y la acción de desactivar *ItinerarioTextual*.

- Si existe una señal Wifi suficiente se activa el Wifi para la localización. En la Resolución 3 se activa *Wifi* y desactiva *GPS*.

- Si no existe una señal Wifi suficiente se activa la localización por GPS. En la Resolución 4 se activa *GPS* y desactiva *Wifi*.

Dados el Modelo de Características, el Modelo de Configuración y el Modelo de Resoluciones, se puede obtener el Espacio de Adaptación del sistema (Figura 41):

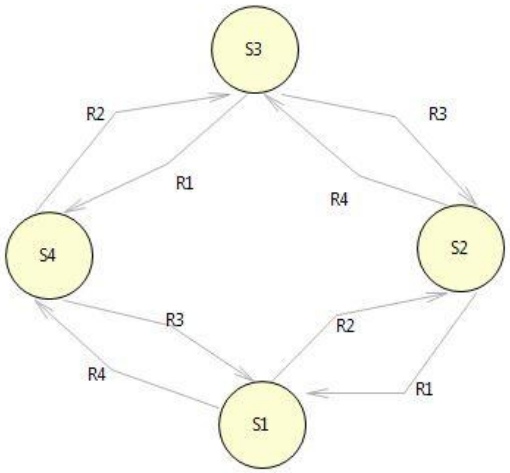


Figura 41: Caso de estudio: Espacio de Adaptación inicial

4.2 EVOLUCIÓN DEL SISTEMA

En este apartado vamos a realizar un ejemplo de proceso de evolución del sistema del caso de estudio, haciendo hincapié en la fase de validación y análisis que es la que se ha implementado en este trabajo.

A continuación vamos a explicar los cambios que se quieren aplicar en el sistema. En primer lugar, queremos añadir un nuevo servicio que muestre aquellos lugares de interés que se encuentran cerca del usuario. Este servicio requiere un Mapa para mostrar los lugares. Además del nuevo servicio, también queremos añadir una nueva restricción: dado que el uso de Mapas produce una gran transferencia de datos, queremos que sólo sea posible usarlos cuando el sistema esté usando Wifi para la localización del usuario.

En segundo lugar, se quiere cambiar el comportamiento del sistema cuando la batería del dispositivo es baja. En un principio se había modelado que si la batería es menor al 20% se cambia el uso de mapas al uso de itinerarios textuales; ahora se quiere añadir también el cambio de GPS a Wifi cuando se dé esta condición de contexto.

En los siguientes apartados se muestra con detalle el proceso de validación y refinamiento de los modelos evolucionados para el caso de estudio.

4.2.1 EVOLUCIÓN, VALIDACIÓN Y REFINAMIENTO DEL MODELO DE CARACTERÍSTICAS

El Modelo de Características evolucionado es el siguiente:

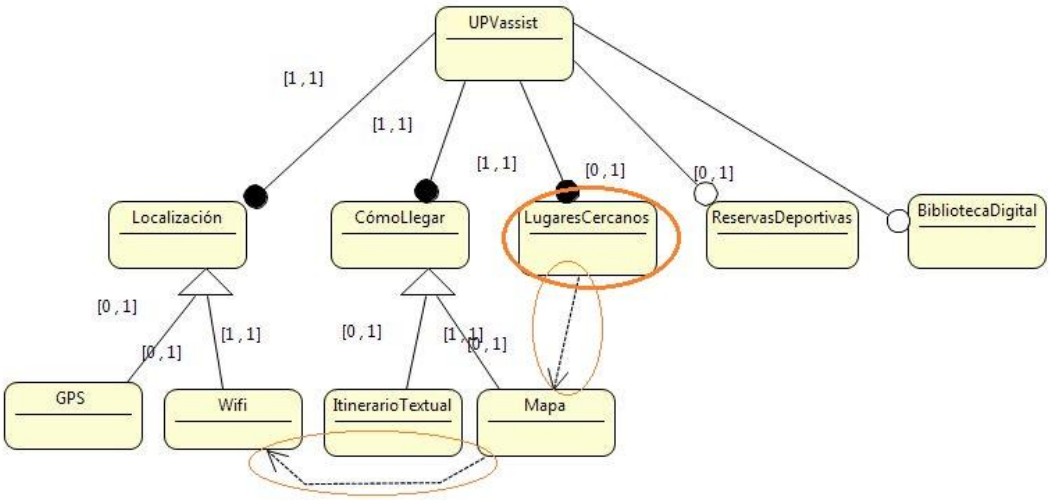


Figura 42: Caso de estudio: Modelo de Características evolucionado

En la Figura 42 vemos el Modelo de Características, en el cual se ha añadido la característica *LugaresCercanos*. También se ha añadido una relación Requiere entre *LugaresCercanos* y *Mapa*, y entre *Mapa* y *Wifi*.

A continuación queremos generar un Modelo de Configuración del sistema, pero ya que éste se define en términos de un Modelo de Características, debemos validarlo primero. Para ello podemos seleccionar la opción del menú contextual del Modelo de Características: *Configuration Model / Validate and Create new CM*, y obtenemos la siguiente ventana (Figura 43):

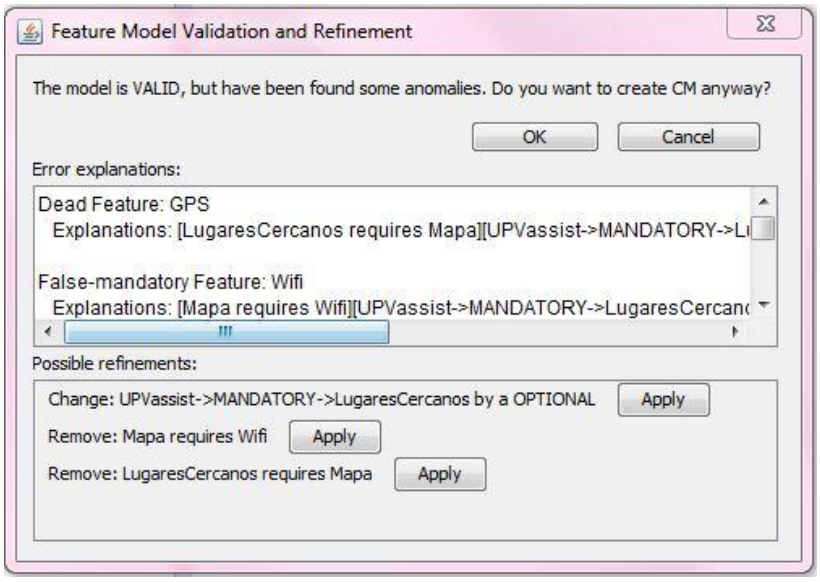


Figura 43: Caso de estudio: Ventana Validación FM Inválido

Las explicaciones de error obtenidas son las siguientes:

DeadFeature: GPS

Explanations: [Mapa requiresWifi][Localización->ALTERNATIVE->[GPS , Wifi]][LugaresCercanosrequires Mapa][UPVassist->MANDATORY->LugaresCercanos]

False-mandatoryFeature: Wifi

Explanations: [Mapa requiresWifi][LugaresCercanosrequires Mapa][UPVassist->MANDATORY->LugaresCercanos]

False-mandatoryFeature: Mapa

Explanations: [LugaresCercanosrequires Mapa][UPVassist->MANDATORY->LugaresCercanos]

DeadFeature: ItinerarioTextual

Explanations: [LugaresCercanosrequires Mapa][UPVassist->MANDATORY->LugaresCercanos][CómoLlegar->ALTERNATIVE->[Mapa, ItinerarioTextual]]

Si analizamos las explicaciones de errores vemos que el Modelo de Características tiene 2 errores concretos:

- La característica *LugaresCercanos* es Obligatoria y requiere *Mapa*, y por tanto *Mapa* es una Característica Falso-Obligatoria e *ItinerarioTextual* es una característica muerta.

- Y teniendo en cuenta lo anterior y que la característica *Mapa* requiere *Wifi*, entonces *Wifi* es una Característica Falso-Obligatoria y *GPS* es una característica muerta.

Para solucionar estas anomalías y que el modelo sea válido, se ofrecen los siguientes refinamientos:

- Change: UPVassist->MANDATORY->LugaresCercanos by an OPTIONAL.

- Remove: Mapa requires Wifi.

- Remove: Lugares Cercanos requires Mapa.

Estos refinamientos consisten básicamente en eliminar las restricciones Requiere o cambiar la relación Obligatoria de *LugaresCercanos* por una relación Opcional. Si aplicamos un refinamiento, por ejemplo el cambio de relación Obligatoria a Opcional, obtendremos el Modelo de Características refinado de la Figura 44, en el cual, la Característica *LugaresCercanos* ya no es Obligatoria sino Opcional:

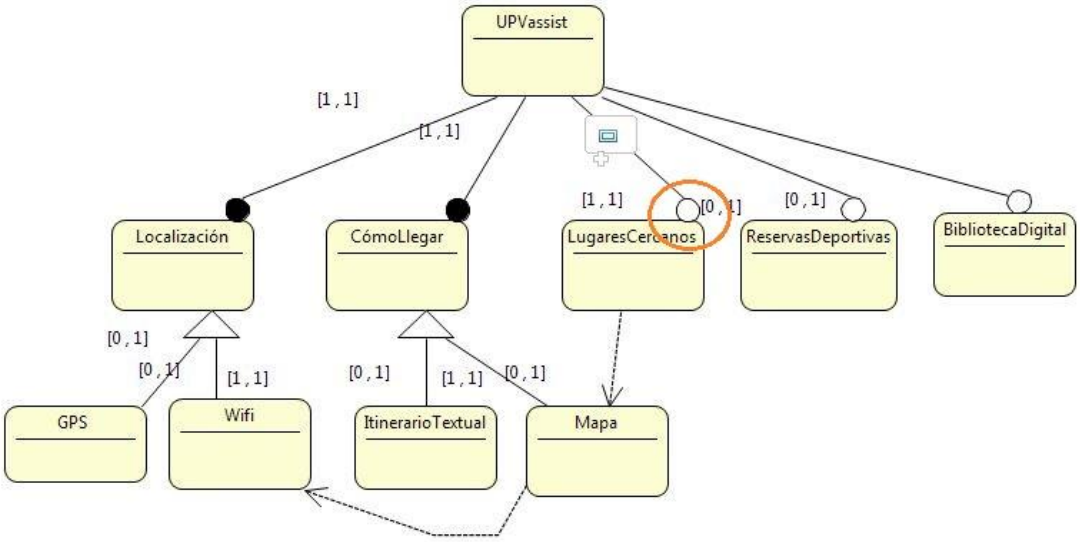


Figura 44: Caso de estudio: Modelo de Características refinado

Si validamos el Modelo de Caracteristicas refinado con la opción *Configuration Model / Validate and Create new CM*, comprobamos que es válido y la herramienta nos permite crear un Modelo de Configuración a partir de él.

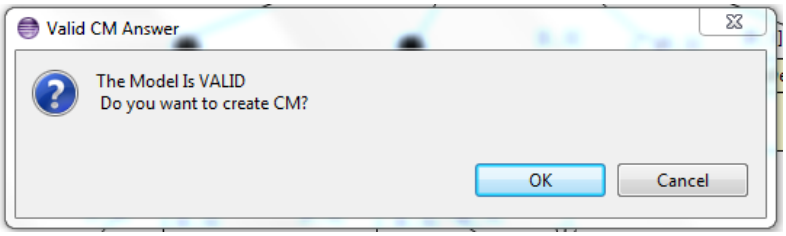


Figura 45: Caso de estudio: Ventana Validación FM Válido

4.2.2 EVOLUCIÓN, VALIDACIÓN Y REFINAMIENTO DEL MODELO DE CONFIGURACIÓN

Se ha definido el siguiente Modelo de Configuración (Figura 46) para representar la configuración inicial del sistema:

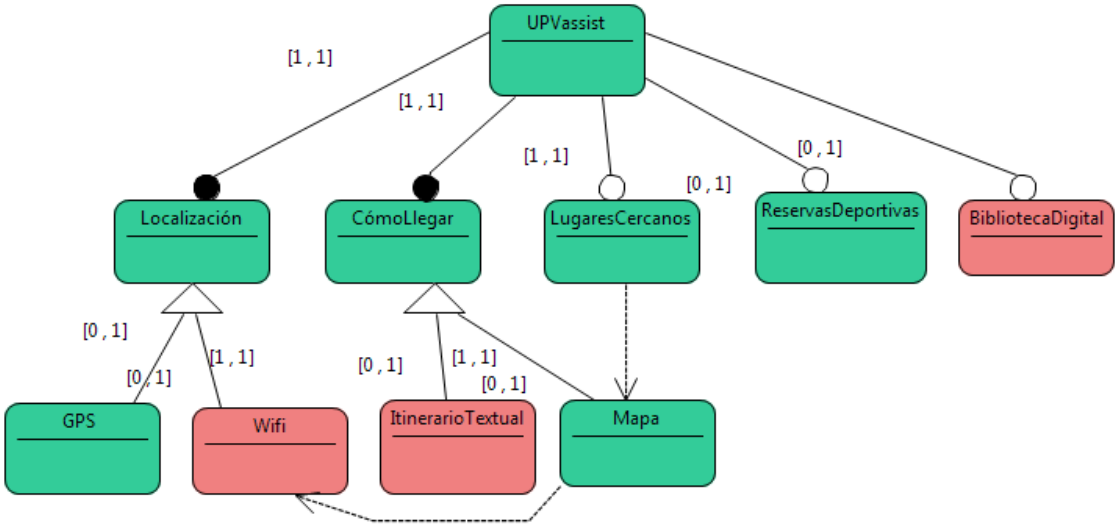


Figura 46: Caso de estudio: Modelo de Configuración evolucionado

En el Modelo de Configuración de la Figura 46, vemos que se opta por usar GPS para la localización, mapas para mostrar los itinerarios, y están activos los servicios *LugaresCercanos* y *ReservasDeportivas*. Si validamos el modelo con la opción del menú contextual *Validate and Refine Product* obtenemos la siguiente ventana:

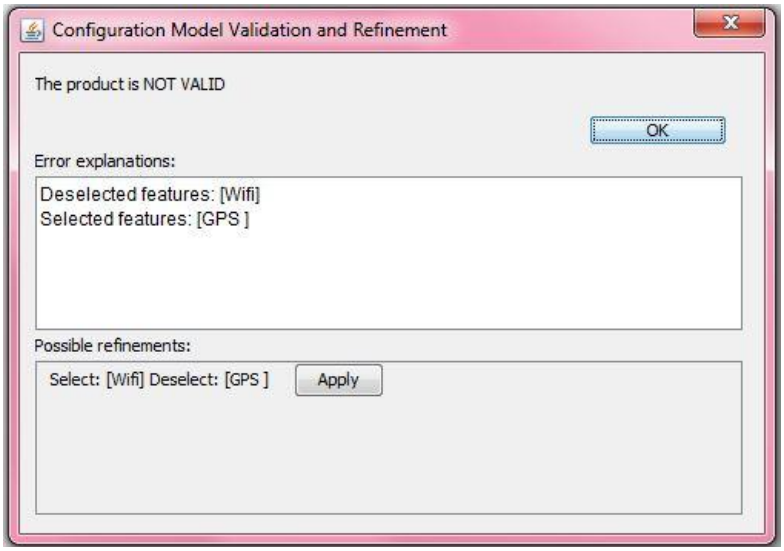


Figura 47: Caso de estudio: Ventana Validación CM Inválido

La ventana de la Figura 47, nos indica que la Caracteristica *Wifi* debería estar activada y *GPS* desactivada, ya que la Característica *Mapa* tiene una relación requiere con *Wifi*, y *Mapa*

está activada en esta configuración. Para solucionar este error y que el Modelo sea válido se ofrece un refinamiento. El resultado del refinamiento es el siguiente Modelo de Configuración refinado:

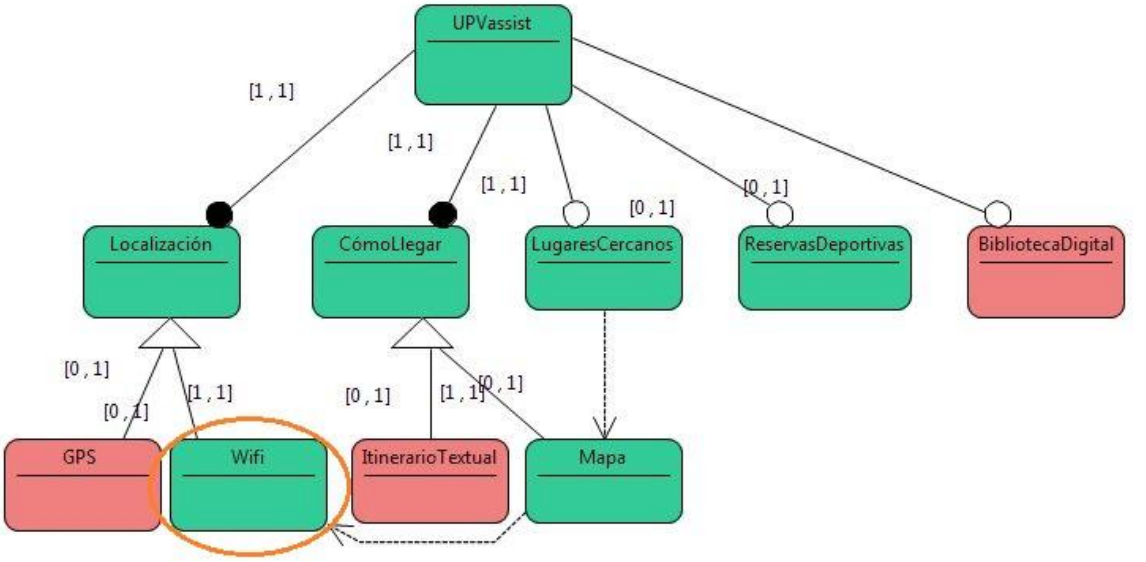


Figura 48: Caso de estudio: Modelo de Configuración refinado

En el Modelo de Configuración de la Figura 48 vemos que el refinamiento ha activado *Wifi* y ha desactivado *GPS*. Si validamos el Modelo refinado comprobamos que es válido:

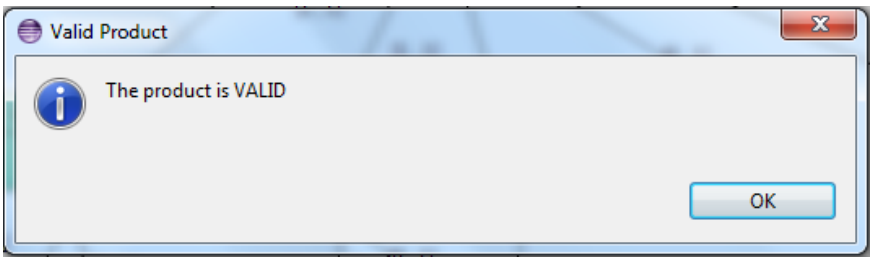


Figura 49: Caso de estudio: Ventana Validación CM Válido

4.2.3 EVOLUCIÓN, VALIDACIÓN Y REFINAMIENTO DEL MODELO DE RESOLUCIONES

En la Figura 50 se muestra el Modelo de Resoluciones evolucionado. Se ha decidido combinar las 4 resoluciones del Modelo anterior en 2 resoluciones, y se han añadido las resoluciones correspondientes a la activación/desactivación de la nueva Característica *LugaresCercanos*.

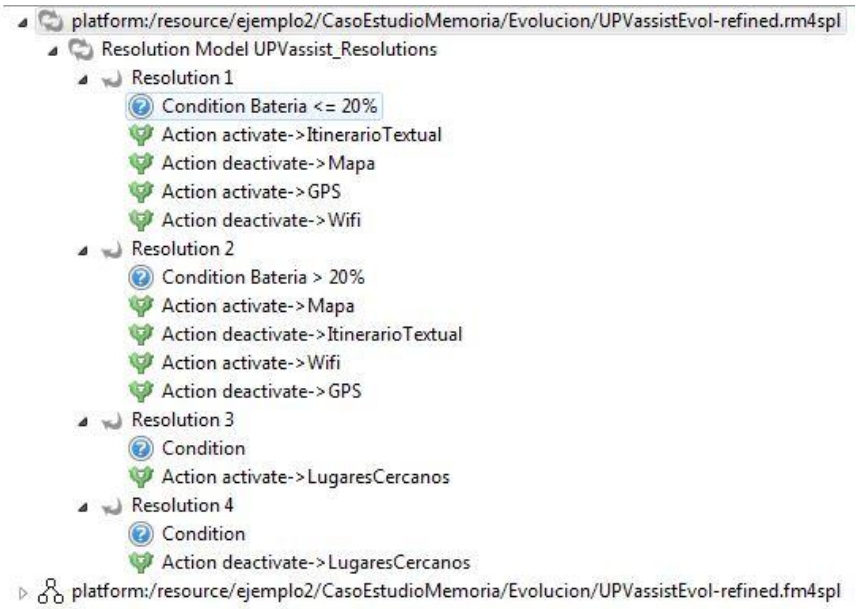


Figura 50: Caso de estudio: Modelo de Resoluciones evolucionado

Si validamos el nuevo Modelo de Resoluciones con la opción *Validate Resolutions* de su menú contextual obtenemos la siguiente ventana:

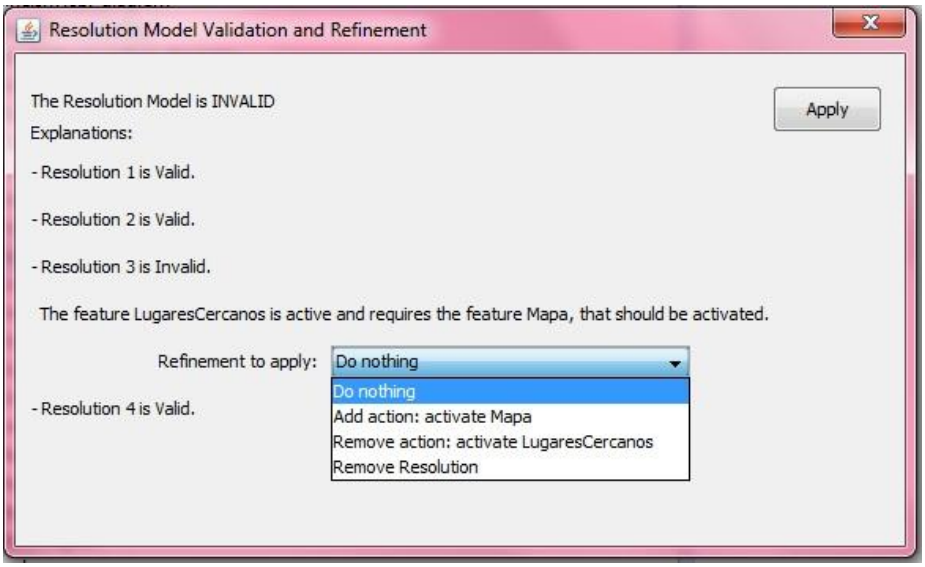


Figura 51: Caso de estudio: Ventana Validación RM Inválido (I)

En la ventana de la Figura 51 se muestra para cada Resolución su validez, y en caso de no ser válida, el motivo y una serie de refinamientos. La Resolución 3 es inválida, ya que, la

característica *LugaresCercanos*, que se activa en una acción de la Resolución 3, requiere la característica *Mapa*, por tanto, debería existir en esta misma resolución una acción que la activara. Así pues, los refinamientos que se ofrecen para solucionar este error son: añadir una acción que sea "activar *Mapa",* eliminar la acción "activar LugaresCercanos" o eliminar la resolución por completo. Si optamos por seleccionar el refinamiento de añadir una acción que sea "activar *Mapa"*, obtenemos el siguiente Modelo de Resoluciones refinado.

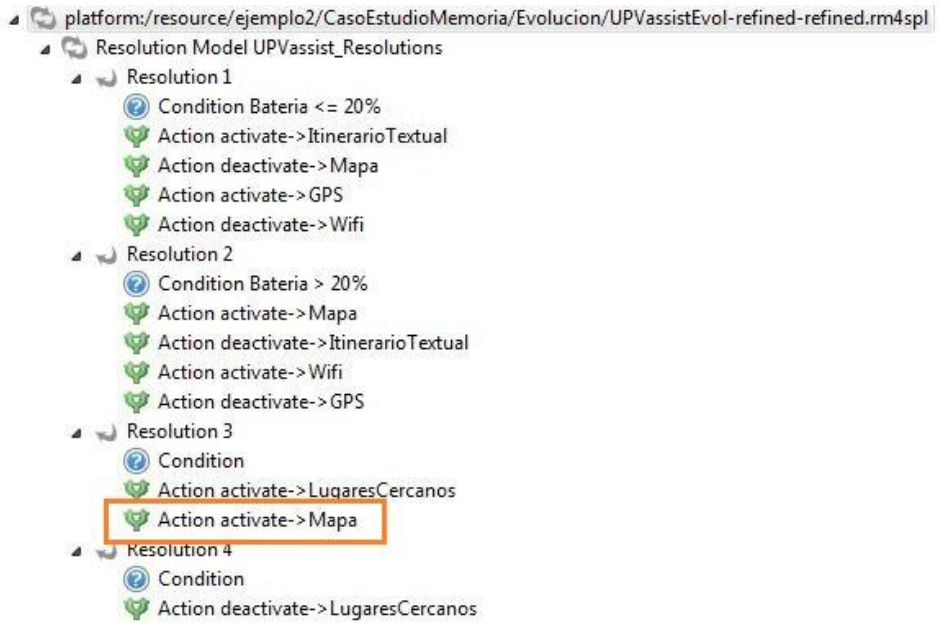


Figura 52: Caso de estudio: Modelo de Resoluciones refinado

En el Modelo de Resoluciones de la Figura 52 vemos que el refinamiento ha añadido a la Resolución 3 la acción de activar *Mapa*. Si comprobamos la validez de este nuevo Modelo de Resoluciones veremos que sigue siendo inválido. Esto es debido a que hemos añadido la activación de *Mapa*, y ésta característica requiere la activación de la característica *Wifi.*

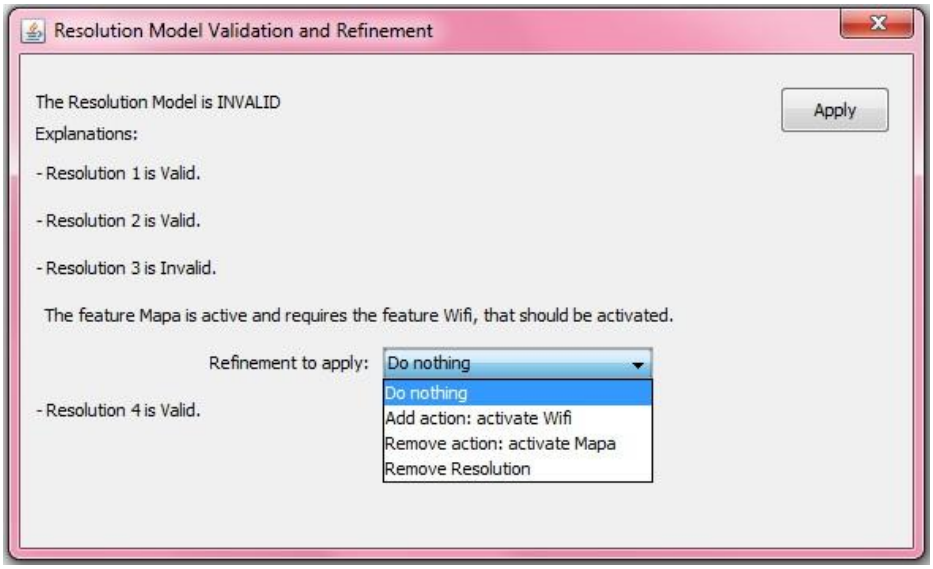


Figura 53: Caso de estudio: Ventana Validación RM Inválido (II)

Para hacer válido el modelo optamos por el refinamiento añadir acción que sea "*Activar Wifi*", de manera que la Resolución 3 queda como se muestra en la Figura 54:

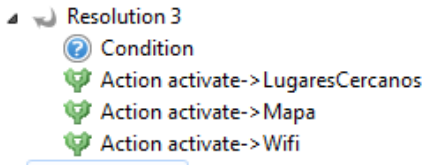


Figura 54: Caso de estudio: Resolución 3 después del refinamiento.

Finalmente, si volvemos a comprobar la validez del Modelo veremos que es válido (Figura 55):

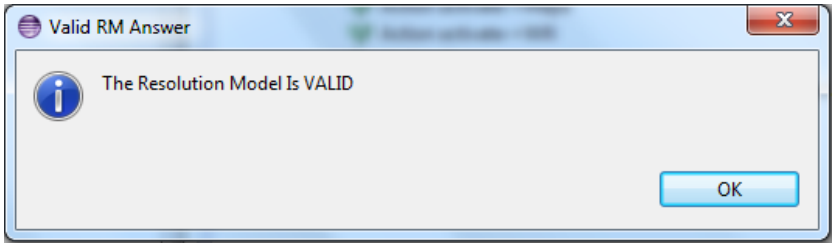


Figura 55: Caso de estudio: Ventana Validación RM Válido

4.2.4 EVOLUCIÓN, VALIDACIÓN Y REFINAMIENTO DEL ESPACIO DE ADAPTACIÓN

En los pasos anteriores hemos obtenido un Modelo de Características, un Modelo de Configuración y un Modelo de Resoluciones evolucionados, hemos podido validarlos y realizar refinamientos para obtener versiones válidas de éstos. A partir del Modelo de Configuración y el Modelo de Resoluciones podemos generar el Espacio de Adaptación que se muestra en la Figura 56:

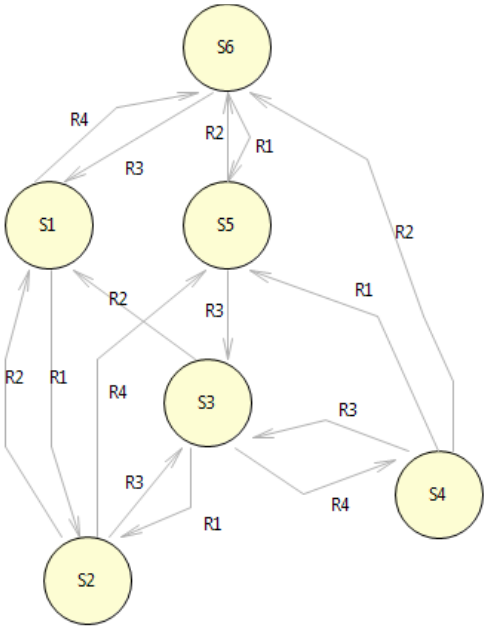


Figura 56: Caso de estudio: Espacio de Adaptación evolucionado

Para validar el Espacio de Adaptación usamos la opción *Validate Adaptation Space* de su menú contextual, y obtenemos la ventana de la Figura 57:

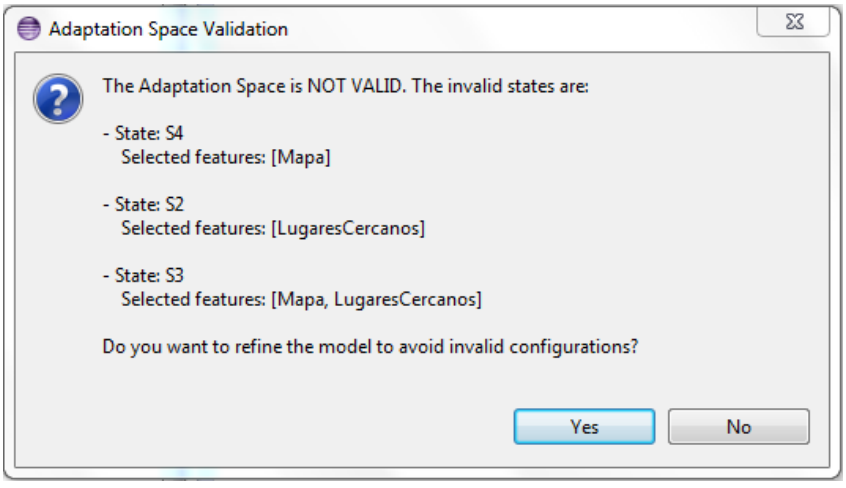


Figura 57: Caso de estudio: Ventana Validación SM Inválido

La validación nos indica que los estados S4, S2 y S3 son inválidos y nos da las explicaciones de error. Estas explicaciones de error nos dicen que características están Activadas

o Desactivadas y son causantes del error. Para analizar por qué estos estados resultan ser inválidos vamos a ver sus Modelos de Configuración asociados.

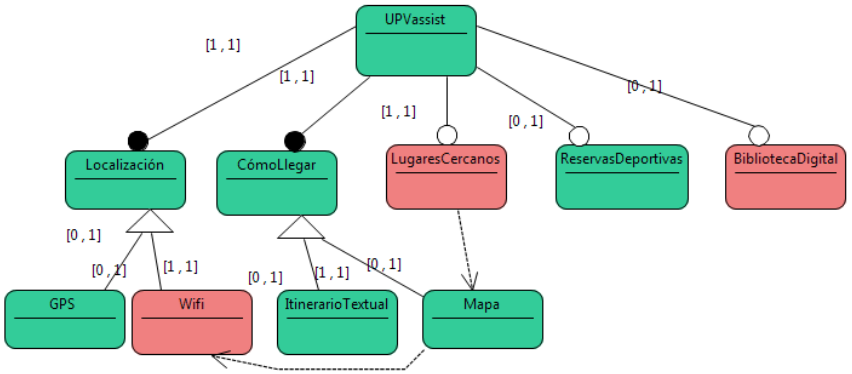
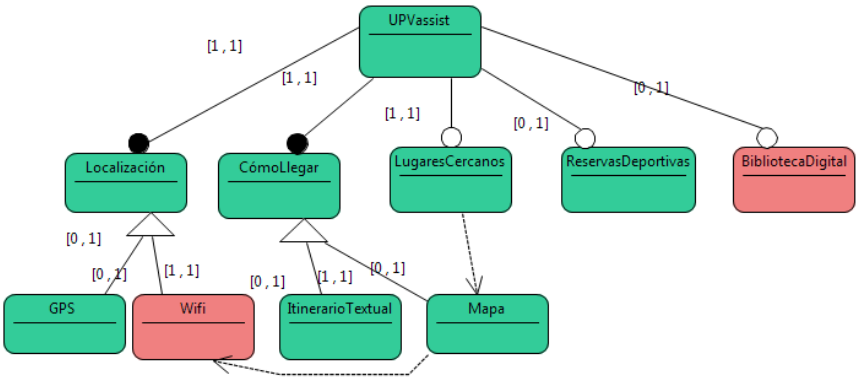
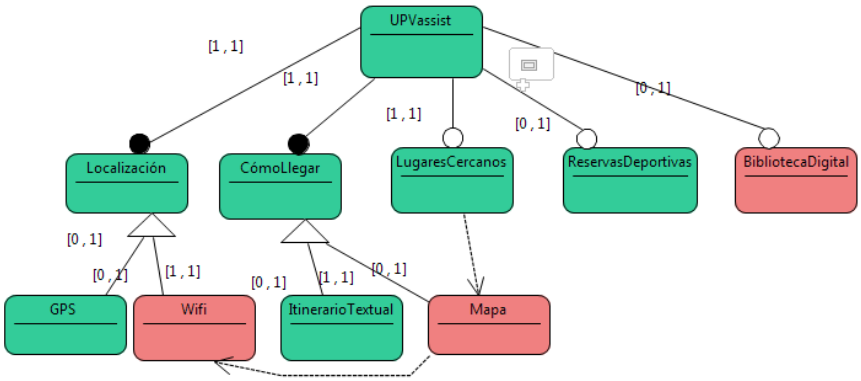
Estado S4	 *Mapa* está activado simultáneamente con *ItinerarioTextual*, y además requiere *Wifi*, y *Wifi*esta desactivado.
Estado S3	 Este caso es similar al anterior, pero si *Mapa* no debería estar activado, *LugaresCercanos* tampoco podría estarlo, debido a su relación *Requiere* con *Mapa*.
Estado S2	 *LugaresCercanos*esta activado, y requiere *Mapa*, que no está activado.

Tabla 8: Caso de estudio: Estados inválidos y CM asociados

Podemos deducir de los Modelos de Configuraciones de los estados inválidos que las configuraciones válidas, teniendo en cuenta las 3 características implicadas en las relaciones

Requiere del modelo, *Wifi, Mapa* y *LugaresCercanos*; son aquellas Configuraciones donde esté activada sólo *Wifi, Wifi*y *Mapa*, o las 3 características a la vez. Cualquier otra Configuración deja al sistema en un estado inconsistente, por no cumplirse las relaciones Requiere.

A continuación se muestra el resultado de aplicar la operación de refinamiento del Espacio de Adaptación:

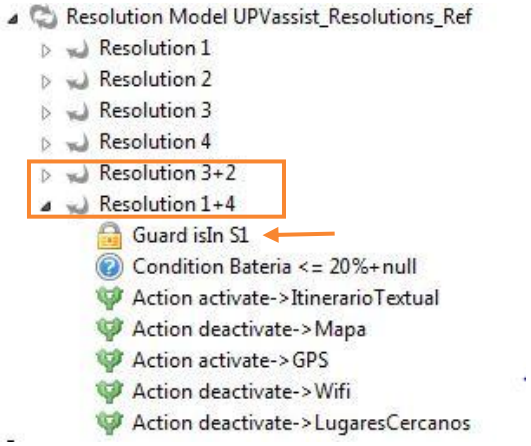


Figura 58: Modelo de Resoluciones resultado del Refinamiento del Espacio de Adaptación

En la Figura 58, vemos el Modelo de Resoluciones resultante, en el que podemos observar que se han creado nuevas resoluciones y se han añadido guardas, según el algoritmo de refinamiento. Si generamos el espacio de adaptación a partir del Modelo de Características y éste nuevo Modelo de Resoluciones, obtenemos la Máquina de Estados refinada.

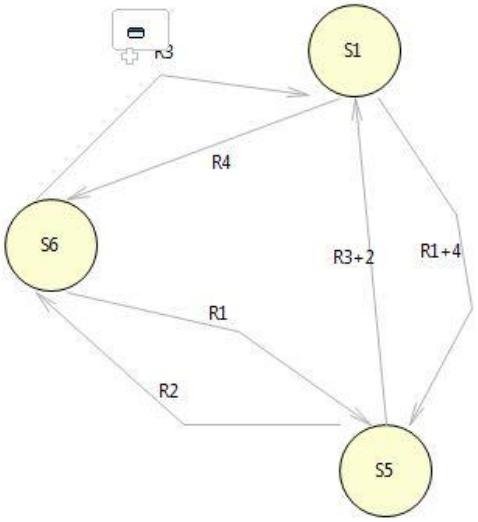


Figura 59: Máquina de Estados resultado del Refinamiento del Espacio de Adaptación

En la Máquina de Estados de la Figura 59 vemos que se han suprimido los estados inválidos, y se han generado nuevas transiciones como combinación de otras.

5. HERRAMIENTAS DE SOPORTE

En este capítulo vamos a explicar las principales herramientas existentes y tecnologías utilizadas en el desarrollo de éste trabajo fin de máster.

5.1 HERRAMIENTAS EXISTENTES

5.1.1 FAMA REASONER

FaMa [19] es una herramienta para analizar de forma automática Modelos de Características. FaMa puede usarse de 4 maneras: por una parte, se proporciona una interfaz para usuarios finales; por otra parte, se puede integrar una aplicación con FaMa, mediante: una versión java independiente, servicios web SOAP/WSDL, o una serie de bundles OSGi. En la herramienta Moskitt4SPL se utiliza FaMa mediante la versión Java independiente.

FaMa es capaz de analizar Modelos de Variabilidad para extraer información relevante para dar soporte a la toma de decisiones en el desarrollo de Líneas de Producto Software. Algunos ejemplos de operaciones de análisis son: extraer qué productos pueden encajar con la necesidad del cliente, cuál es el producto más económico y su coste, o cuáles son los errores de un Modelo de Características.

Las funcionalidades que FaMa ofrece han sido aplicadas en diferentes contextos:

- Herramientas de modelado de variabilidad: en editores visuales y otras herramientas CASE que utilizan Modelos de Variabilidad se han incorporado capacidades de análisis. Por ejemplo, en el editor visual de Modelos de Características de Moskitt4SPL [17].

- Configuradores de productos: FaMa proporciona capacidades para validar y sugerir correcciones sobre selecciones de características, o en otras palabras, configuraciones de productos.

- Reconfiguración de sistemas Dinámicos: FaMa también ha puede usarse en la recuperación o reconfiguración de sistema dinámicos, como casas inteligentes, cuando ocurren errores o las preferencias del sistema cambian.

A continuación se presenta una tabla con las distintas operaciones de análisis de Modelos de Características disponibles en FaMa, y una breve descripción:

Validation	Valida un modelo. Comprueba si un modelo no es vacío, en otras palabras, si tiene al menos un producto.
Products	Calcula todos los posibles productos válidos de un Modelo de Características.
Number of Products	Calcula el número de productos.
Commonality	Calcula apariciones de una característica dada en la lista de

	productos.
Variability	Calcula el grado de variabilidad de un Modelo de Características.
Valid product	Determina si un producto es válido para un modelo dado.
Valid configuration	Analiza si una Configuración es válida. Una Configuración es un producto no terminado si necesita más características para ser un producto válido.
Error detection	Busca errores en un Modelo de Características.
Error explanations	Cuando un modelo tiene errores, esta operación busca las explicaciones.
Invalid product explanation	Proporciona opciones para reparar un producto inválido para un modelo dado.
Core features	Calcula las características presentes en todos los productos.
Variant features	Calcula las características que no están presentes en todos los productos

Tabla 9: Operaciones FaMa

Las operaciones definidas en FaMa están implementadas por 4 razonadores distintos. Cada razonador implementa un subconjunto de operaciones usando solucionadores o algoritmos. Los 4 razonadores son: choco, sat4j, jacop y javabdd.

En la implementación realizada en esta tesis de máster se ha usado el framework FaMa para realizar la validación y refinamiento de alguno de los modelos de variabilidad. En concreto se ha usado para validar los Modelos de Características y Modelos de Configuración, y encontrar las explicaciones de errores, que han guiado la implementación de los refinamientos realizados en este trabajo. También se ha usado en la validación del Espacio de Adaptación, mediante la validación de cada una de las configuraciones parciales que constituyen los diferentes estados de la Máquina de Estados.

5.1.2 MOSKITT4SPL

Moskitt4SPL [17] es una herramienta de código abierto que da soporte al modelado de Líneas de Producto Software. Esta herramienta está basada en Moskitt [21] y está construida sobre Eclipse [22].

Para el modelado de Líneas de Producto Software, MOSKitt4SPL ofrece las siguientes características:

- Editor de Modelos de Características: permite crear Modelos de Características de manera gráfica (Figura 60).

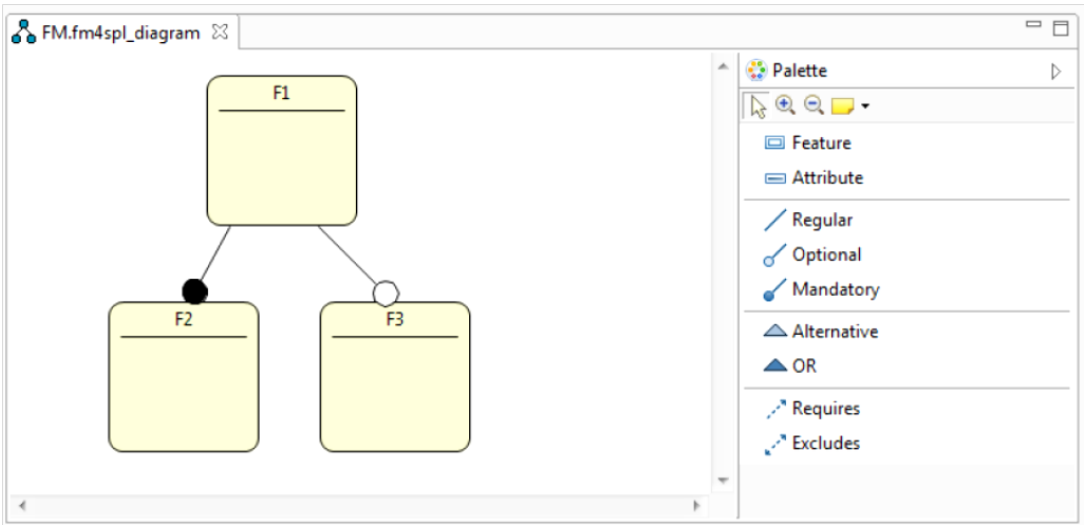


Figura 60: Editor de Modelos de Características en Moskitt4SPL

- Configurador de Modelos de Características. (Editor de Modelos de Configuración): a partir de un Modelo de Características, Moskitt4SPL permite crear y editar (Figura 61) Modelos de Configuración a partir de éste.

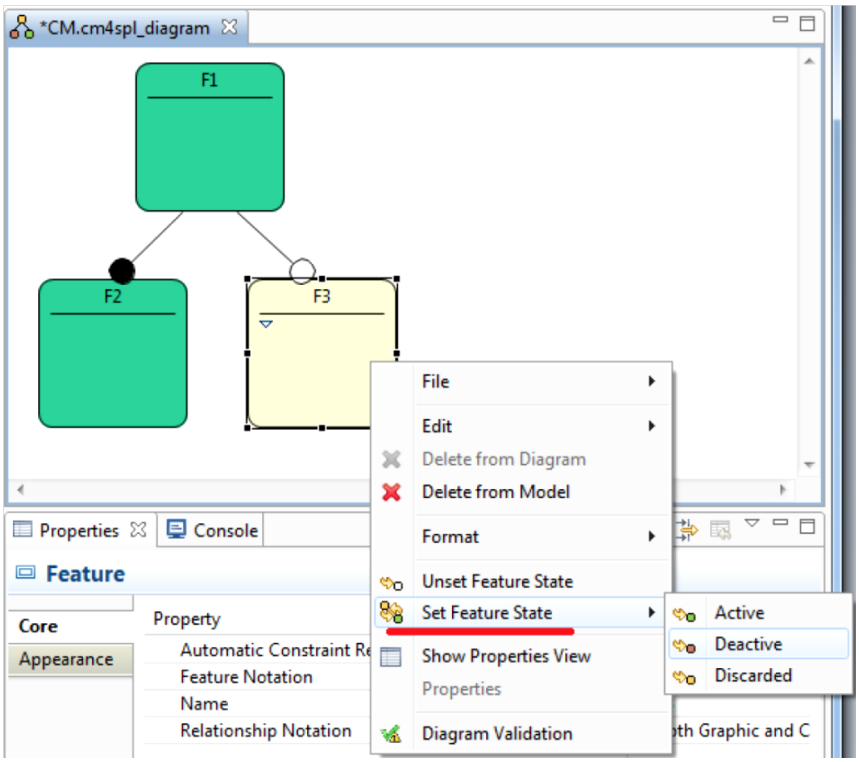


Figura 61: Editor de Modelos de Configuración en Moskitt4SPL

Las Líneas de Producto Software Dinámicas permiten posponer la configuración de productos software hasta el tiempo de ejecución, de manera que los sistemas son capaces de adaptarse a los cambios en el entorno en tiempo de ejecución.

Las características que MOSKitt4SPL ofrece para modelar Líneas de Productos Software Dinámicas son:

- Editor de Modelos de Resolución: Moskitt4SPL ofrece el editor textual para Modelos de Resoluciones mostrado en la Figura 62:

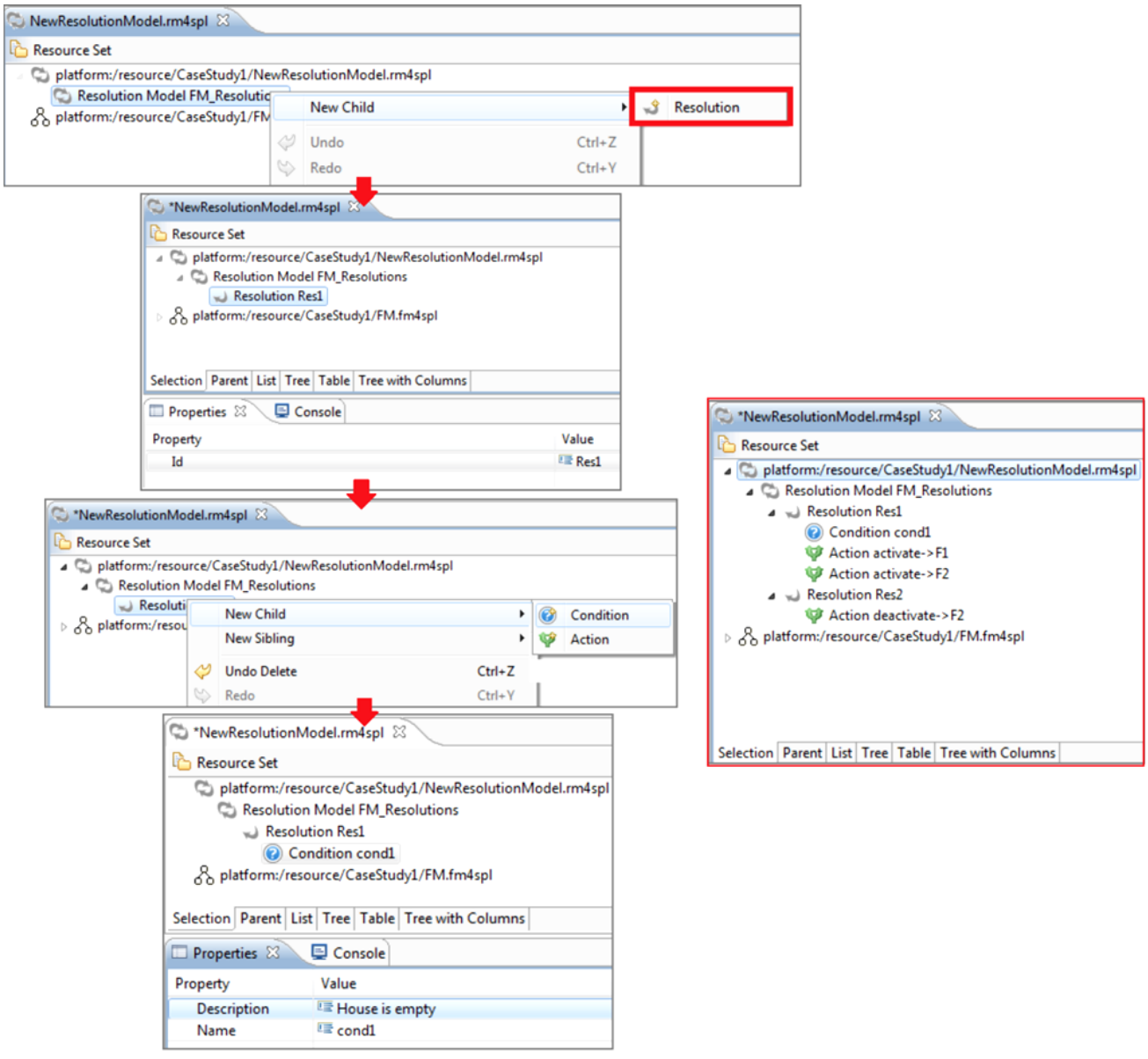


Figura 62: Editor de Modelos de Resoluciones en Moskitt4SPL

- Generación del Espacio de Adaptación: a partir de un Modelo de Configuración y un Modelo de Resoluciones, Moskitt4SPL permite generar el Espacio de Adaptación mediante los pasos mostrados en la Figura 63:

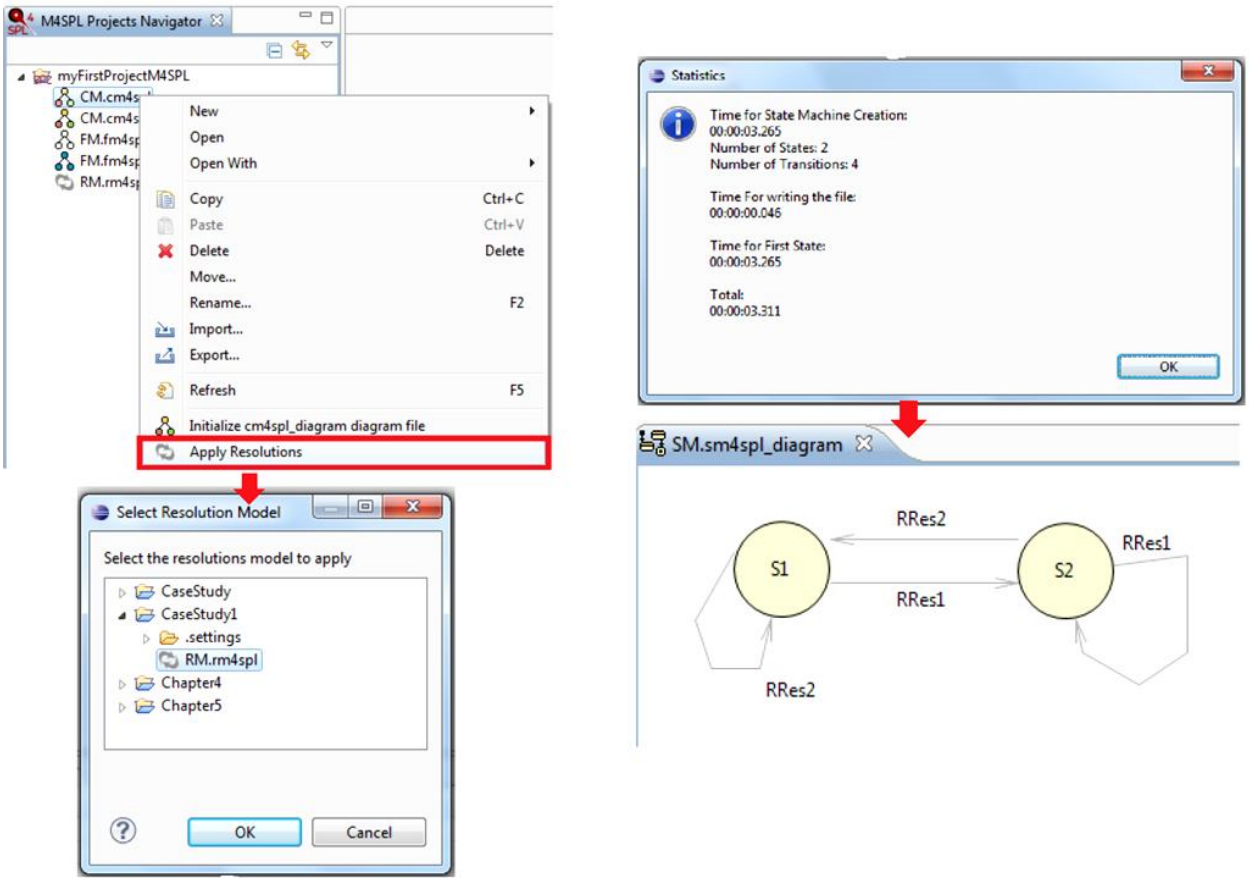


Figura 63: Generación del Espacio de Adaptación en Moskitt4SPL

En la herramienta Moskitt4SPL están integradas algunas de las operaciones del frameworkFaMa vistas en el apartado 5.1.2. En la Figura 64 se muestra la parte de las Operaciones FaMa en el menú contextual de un Modelo de Características.

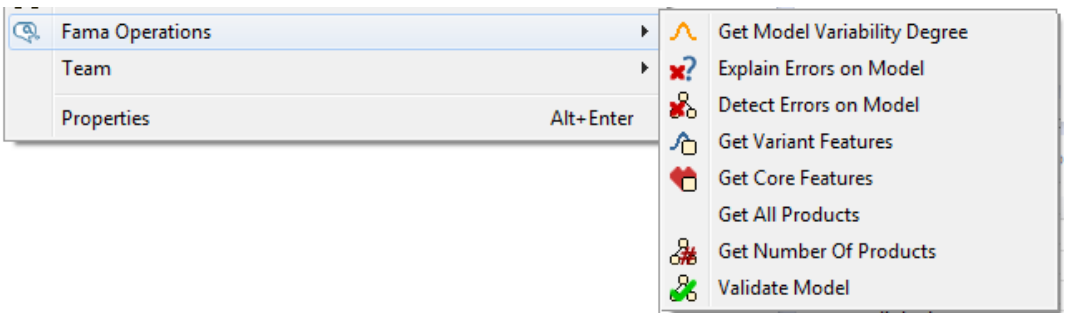


Figura 64: Operaciones FaMa para un Modelo de Características

En la herramienta Moskitt4SPL ya se incluía la validación de los Modelos de Características y de los Modelos de Configuración. En éste trabajo se ha extendido ésta funcionalidad, añadiendo una Validación de los modelos, donde no solamente se dice si son válidos o inválidos, sino que se muestran una las explicaciones de los errores y se proporcionan una serie de refinamientos para solucionarlos.

Respecto al Modelo de Resoluciones y el Espacio de Adaptación, la implementación de la validación se ha realizado íntegramente en este trabajo, ya que en Moskitt4SPL no estaba implementada esta funcionalidad. Y además, al igual que para el Modelo de Características y Modelo de Configuración, se han añadido refinamientos para solucionar los problemas encontrados.

5.2 TECNOLOGÍAS UTILIZADAS

5.2.1 JAVA

Java [23] es un lenguaje de programación desarrollado originalmente por Sun Microsystems, y fue publicado en 1995. Las aplicaciones de Java suelen compilarse a bytecode que puede ser ejecutado en cualquier máquina virtual java (JVM). De esta manera, Java es independiente de la arquitectura del sistema, es decir, fue diseñado para tener las mínimas dependencias de implementación como fuera posible, permitiendo así, que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo ("write once, run anywhere"). Es un lenguaje de programación de propósito general, que permite concurrencia, basado en clases y orientado a objetos.

Java tiene las siguientes características principales:

- Usa el paradigma orientado a objetos: la idea principal del paradigma orientado a objetos es que el software sea diseñado de manera que los tipos de datos que sean usados estén unidos a sus operaciones. De esta manera se combinan los datos ("estado") y las funciones ("comportamiento") en entidades llamadas objetos.

- Permite la ejecución de un programa en varios sistemas operativos distintos: la independencia de plataforma se refiere a que los programas escritos en Java pueden ser ejecutados de igual manera en cualquier tipo de hardware. Como ya se ha mencionado anteriormente, se debe poder escribir un programa una vez y ejecutarlo en cualquier dispositivo.

- Incluye soporte para trabajo en red y ejecutar código en sistemas remotos de manera segura: estas dos características se consiguen recurriendo a extensiones de Java como CORBA, Internet Communications Engine o OSGi.

- Contiene un recolector de basura: el problema de las fugas de memoria se soluciona mediante la recolección de basura (*automatic garbage collector*). En Java el programador determina cuándo se crean los objetos, pero es la ejecución de Java el que gestiona el ciclo de vida de los objetos.

La sintaxis de Java deriva principalmente de C++, pero Java combina la sintaxis para programación genérica, estructurada y orientada a objetos. Java es completamente orientado a objetos, todo es un objeto, y todo en Java está en alguna clase.

La herramienta Moskitt4SPL, que se ha extendido en éste trabajo, está implementada en Java.

5.2.2 ECLIPSE

Eclipse [22] es un entorno de desarrollo integrado (IDE, Integrated Development Environment) de código abierto y multiplataforma, para el desarrollo de software. El proyecto Eclipse fue originalmente creado por IBM en Noviembre del 2001. La Fundación Eclipse se creó en Enero de 2004 como una corporación sin ánimo de lucro.

Eclipse se diseñó con el objetivo de facilitar la edición, compilación y ejecución de programas durante la fase de desarrollo. Es una aplicación gratuita y de código abierto disponible en la red.

5.2.3 ECLIPSE MODELING TOOLS (EMF, GMF, ...)

Las Eclipse Modeling Tools (EMT) [24] están dentro del Eclipse Modeling Project, que se centra en la evolución y promoción de las tecnologías de desarrollo basadas en modelos. De manera que se proporcionan un conjunto unificado de frameworks de modelado, herramientas y estándares.

A continuación se presentan algunas partes de EMT:

- Eclipse Modeling Framework (EMF): es un framework de modelado y generación de código para construir herramientas y otras aplicaciones basadas en modelos.

- Graphical Modeling Framework (GMF): proporciona componentes e infraestructuras en tiempo de ejecución para desarrollar editores gráficos basados en EMF.

- Model to Model Transformation (M2M): se ha desarrollado un framework extensible para los lenguajes de transformación Modelo a Modelo.

- Atlas Transformation Language (ATL): es un lenguaje de transformación de Modelos.

- Model to Text Transformation (M2T): se centra en tecnologías para transformar modelos en texto (normalmente código fuente).

Moskitt4SPL está construida sobre Eclipse, y hace uso de EMF y GMF para la construcción de modelos mediante editores textuales y gráficos. Y hace uso de ATL para las transformaciones entre modelos de variabilidad.

6. CONCLUSIONES

En este capítulo se presentan las conclusiones obtenidas de la realización de éste trabajo final de máster y se proponen una serie de trabajos futuros relacionados.

6.1 CONCLUSIONES

Los sistemas hardware y software actuales dependen cada vez más de las condiciones de contexto en las que se encuentran. Los sistemas auto-adaptables son capaces de realizar cambios en su comportamiento según estas condiciones.

Para plasmar los cambios que se requieren en el sistema, es necesario tener un proceso de evolución que permita obtener nuevas versiones de las descripciones del conocimiento del sistema. La evolución mediante la aproximación basada en Modelos en Tiempo de Ejecución permite realizar la evolución de los sistemas en tiempo de ejecución, sin necesidad de detener el sistema. Esto es muy beneficioso en sistemas críticos o altamente dinámicos, como los sistemas sensibles al contexto o ubicuos.

En el proceso de Evolución de los sistemas auto-adaptables es esencial asegurar la corrección del sistema antes y después de la evolución. Por este motivo, la tarea de analizar y refinar los modelos que representan el comportamiento del sistema, antes de aplicarlos al sistema en ejecución, es muy importante. La implementación realizada en éste trabajo final de máster tiene como objetivo facilitar y formalizar ésta fase del proceso de evolución. Concretamente nos centramos en el análisis y refinamiento de los Modelos de Características, Modelos de Configuración, Modelos de Resoluciones y Espacios de Adaptación.

La herramienta Moskitt4SPL ya disponía de editores textuales y visuales de todos los Modelos de Variabilidad, incluyendo la generación del Espacio de Adaptación; y herramientas de validación de los Modelos de Características y Configuración. Pero era necesario completar la fase de análisis añadiendo la validación de los Modelos de Resolución y del Espacio de Adaptación, y refinamientos para todos los Modelos de Variabilidad.

Añadiendo esta funcionalidad conseguimos que la fase de análisis esté completa, y pueda verse como un proceso, en el que, en cada paso, un modelo se modifica para añadir los cambios requeridos por la evolución, se analiza en busca de errores o anomalías, y en caso de ser inválido, se refina para que sea válido. De esta manera aseguramos que el modelo generado en cada paso es válido, y a partir de éste se genera el siguiente. En el caso de la herramienta Moskitt4SPL, después de la implementación realizada en este trabajo, podemos generar de manera segura, a partir de un Modelo de Características, varios Modelos de Configuración o de Resoluciones; y a su vez, a partir de éstos, un Espacio de Adaptación.

6.2 TRABAJOS FUTUROS

La propuesta ha conseguido solucionar todos los problemas planteados y cumplir con los objetivos formulados, sin embargo aún queda mucho trabajo por hacer que podría abordarse en el futuro. Como trabajos futuros se proponen los siguientes:

- **Mejorar la visualización del Espacio de Adaptación.** La generación del Espacio de Adaptación puede resultar en una Máquina de Estados con un elevado número de estados que resultará difícilmente legible y entendible por una persona. Se propondrán técnicas para mejorar la visualización de los Espacios de Adaptación. Para ello se definirán mecanismos que permitan agrupar estados que compartan determinadas condiciones o propiedades, resultando en Espacios de Adaptación más reducidos que faciliten la visualización y entendimiento de los mismos.

- **Validación y corrección de modelos 'on-the-fly'.** Como trabajo futuro se incorporarán nuevas operaciones de FaMa para permitir informar al ingeniero 'on-the-fly' sobre errores que aparecen mientras diseña o evoluciona un modelo. Además se ofrecerán mecanismos para auto-completar de forma automática lo que falte en el modelo para ser válido.

- **Abordar otros dominios de aplicación.** En este trabajo la metodología y las técnicas propuestas se han aplicado en el dominio de los sistemas auto-adaptables, pero esto no significa que sus capacidades queden limitadas a este contexto. Es posible aprovechar el potencial de esta propuesta para aplicarlo a otros dominios como por ejemplo el *Cloud Computing*.

7. BIBLIOGRAFÍA

[1] "J. Hong, E. Suh, and S. Kim. Context-aware systems: A literature review and classification. Expert Syst. Appl., 36(4):8509–8522, 2009.".

[2] "B. Schilit, N. Adams, and R. Want. Context-aware computing applications. Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on, pages85–90, 8-9 Dec 1994.".

[3] "G. Banavar and A. Bernstein. Software infrastructure and design challenges for ubiquitous computing applications. Commun. ACM, 45(12):92–96, 2002.".

[4] "María Gómez, Joan Fons, and Vicente Pelechano. Evolución de Sistemas Auto-Adaptables mediante Modelos en Tiempo de Ejecución. (2012).".

[5] T Mens and S Demeyer, "Introduction and roadmap: History and challenges of software," *Chapter 1 of book "Software Evolution"*, Jan 1, 2008.

[6] "Blair, G., Bencomo, N., France, R.B.: Models@run.time. Comp. 42, 22–27 (2009)".

[7] R de Lemos, H Giese, H Müller, and M Shaw, "Software Engineering for Self-Adaptive Systems: A Research Roadmap.," *In Software Engineering for Self-Adaptive Systems, Betty H. Cheng, Rogério Lemos, Holger Giese, Paola Inverardi, and Jeff Magee (Eds.). Lecture Notes In Computer Science, Vol. 5525.*, 2009.

[8] "A.Ganek and R. J. Friedrich. The road ahead achieving wide-scale deployment of autonomic technologies. En Chairingthe Town hall meeting at the 3rd IEEE International Conference on Autonomic Computing, 2006.".

[9] "A. G. Ganek and T. A. Corbi. The dawning of the autonomic computing era. IBM Syst. J., 42(1):5–18, 2003.".

[10] C Cetina, P Giner, J Fons, and V Pelechano, "Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes," *Journal Computer*, vol. Volume 42 Issue 10 , pp. Pages 37-43 , October 2009.

[11] "J. O. Kephart and D. M. Chess.The vision of autonomic computing. Computer, 36(1):41–50,

2003.".

[12] "D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea, and M. Vanover. Autonomic personal computing. IBM Syst. J., 42(1):165–176, 2003.".

[13] "Schmidt, D.C. (February 2006). "Model-Driven Engineering". IEEE Computer 39 (2).".

[14] "P. C. Clements and L. Northrop. Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, August2001.".

[15] Y Brun et al., "Software engineering for self-adaptive systems. chap. Engineering Self-Adaptive Systems through Feedback Loops, pp. 48–70.," (2009).

[16] "David Benavides, Sergio Segura, Antonio Ruiz Cortés: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35(6): 615-636 (2010)".

[17] MosKitt4SPL. [Online]. www.pros.upv.es/m4spl

[18] "King, T., Ramirez, A., Cruz, R., Clarke, P.: An integrated self-testing framework for autonomic".

[19] Framework FAMA. [Online]. www.isa.us.es/fama

[20] María Gómez Lacruz, TFM: Designing Self-Adaptive Systems through Models at Run-time, Julio 2011.

[21] Moskitt. [Online]. http://www.moskitt.org/

[22] Eclipse. [Online]. http://www.eclipse.org/

[23] Java. [Online]. http://www.oracle.com/technetwork/es/java/index.html

[24] Eclipse Modeling Tools. [Online]. http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/junosr1

[25] Moskitt Feature Modeler. [Online].

http://www.moskitt.org/eng/proyectomfmmoskittfeaturemod/

[26] OSGi. [Online]. http://www.osgi.org/Technology/HomePage