Tesina de Máster

Desarrollo de una Herramienta para el Diseño de Reconfiguraciones Seguras en Sistemas de Computación Autónoma

Septiembre de 2010, Valencia

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

UNIVERSIDAD POLITECNICA DE VALENCIA

Autor
Salvador Ibiza Molines

Director
Vicente Pelechano Ferragud

AUTOR

Salvador Ibiza Molines
salibmo@fiv.upv.es

DIRECTOR

Vicente Pelechano Ferragud
pele@dsic.upv.es

TABLA DE CONTENIDO

LISTA DE FIGURAS

1. INTRODUCCIÓN

1.1 INTRODUCCIÓN

Desde hace unos pocos años hasta la actualidad el ser humano ha visto como la evolución tecnológica ha fomentado unos cambios que han afectado directamente a las actividades del día a día. Esta evolución intenta poner a disposición de todas las personas nuevos servicios, nuevos dispositivos… En definitiva cualquier elemento tecnológico que facilite o enriquezca las tareas en casi todos los ámbitos.

El software y todos los elementos hardware que lo acompañan están sujetos a los cambios que con el tiempo se van sucediendo. Dichos cambios pueden ser debidos a problemas en los sistemas, por la incorporación de nuevos elementos, o simplemente porque han cambiado las necesidades de los usuarios. Esto obliga a que los sistemas se deban actualizar o adaptar a las nuevas condiciones si quieren continuar ofreciendo sus servicios con garantías.

Hasta el momento estas adaptaciones se llevan a cabo de una forma artesanal, lo que obliga a detener el sistema. Pero la creciente e imparable variedad de servicios que emergen cada día hace que adaptar los sistemas de forma manual resulte cada vez más complejo.

Es por ello, que cada vez se tiene más presente el desarrollo de sistemas donde el propio sistema se adecue para hacer frente a las nuevas condiciones. Cada vez más el software necesita adaptar dinámicamente su comportamiento en tiempo de ejecución para dar respuesta a los condiciones que constantemente van cambiando. Para hacer frente a los cambios los sistemas deben tener en cuenta no solo las propias variaciones a nivel de la infraestructura software sino también en el entorno físico [3] y las necesidades de los usuarios.

Como consecuencia, la adaptabilidad se presenta como una capacidad subyacente necesaria, especialmente para sistemas altamente dinámicos como los sistemas sensibles al contexto [2, 3] o sistemas ubicuos [4, 5]. Estos sistemas han alcanzado un nivel de complejidad donde el esfuerzo humano necesario para su ejecución y mantenimiento está fuera del alcance de las personas. Además con el aumento y la irrefrenable gama de servicios que surgen cada día a nuestro alrededor la simplicidad es una propiedad valorada por los usuarios, como se indica en [6, 7]

Con la idea de hacer frente a esta complejidad nace la corriente de la computación autónoma [8]. La computación autónoma persigue alcanzar entornos computacionales que evolucionen sin requerir de

operadores externos. Es decir, un sistema con capacidades autónomas puede instalar [9], proteger [10] y mantener actualizados [11] sus componentes en tiempo de ejecución sin ser necesaria la intervención humana.

La computación autónoma por sí sola únicamente constituye una idea teórica de cómo gestionar la evolución del software. Por ello, se requiere de una metodología de desarrollo software que pueda servir de apoyo para llevar a cabo esta visión. Por sus cualidades la metodología utilizada es el desarrollo dirigido por modelos.

En este trabajo se aplica la visión de la computación autónoma haciendo uso de un método de desarrollo software dirigido por modelos [12]. Además se combina con los principios básicos de otras disciplinas, como por ejemplo, la producción de software basada en líneas de producto [13], los sistemas pervasivos [14, 15] y las técnicas de modelado.

La combinación de estas tecnologías pretende desarrollar sistemas capaces de adecuar su propio comportamiento de acuerdo a los cambios en el entorno de ejecución reduciendo al mínimo la intervención de los usuarios.

1.2 OBJETIVO DE LA TESINA

El trabajo desarrollado en esta tesina de máster combina las técnicas de líneas de producto software y el desarrollo dirigido por modelos para la construcción de software que siga las bases de la computación autónoma.

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 sin necesidad de intervención humana.

La construcción de software siguiendo las bases del desarrollo dirigido por modelos (los modelos como centro del desarrollo) y la metodología de las líneas de producto software (reutilización de componentes) permitirá dar respuesta a las necesidades que surjan en entornos que evolucionan. Para cumplir esas necesidades el sistema deberá modificar su comportamiento de acuerdo a las condiciones del entorno sin la intervención humana y de una forma totalmente dinámica.

Los continuos cambios de comportamiento pueden conducir, en ocasiones, la configuración del sistema a situaciones no válidas. Los estados no válidos se alcanzan cuando la configuración de los diferentes componentes que forman el sistema no cumplen las restricciones. Obviamente, dichos estados no son deseables en el sistema.

Este trabajo surge con el objetivo de:

1) Proporcionar algún mecanismo que gestione estas situaciones no deseadas. Deberá detectar y resolver las condiciones que dejan al sistema en dicho estado. En definitiva, impedir que el sistema alcance configuraciones inadecuadas pero sin perjudicar su funcionalidad.

2) Desarrollar una herramienta que de soporte a las técnicas presentadas. Esta herramienta facilitará la especificación inicial del sistema, el estudio de los cambios en la configuración, ayudará a definir restricciones acerca del comportamiento y validará las diferentes configuraciones que permiten al sistema evolucionar. Además, implementará los mecanismos que identifican y resuelven las situaciones anómalas y dejan el sistema libre de errores.

3) Demostrar mediante el planteamiento de casos de estudio aplicados a contextos reales las técnicas propuestas haciendo uso de la herramienta de soporte.

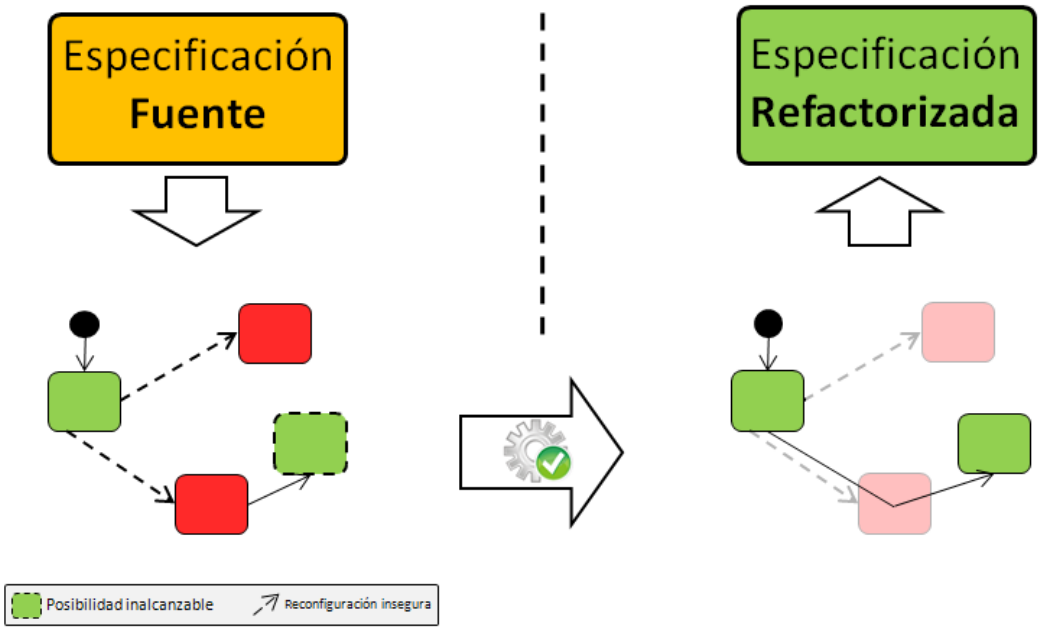


Figura 1 - Objetivo de la tesina

La figura 1 intenta ilustrar de forma simplificada el objetivo que se persigue con esta tesina. A partir de una especificación, Especificación Fuente, se genera un conjunto de posibilidades enlazadas a través de reconfiguraciones. Este conjunto de posibilidades, llamado espacio de posibilidades, contiene unos errores. El primer tipo de error son reconfiguraciones que conducen al sistema a posibilidades inválidas, coloreadas de rojo. Estas reconfiguraciones se denominan reconfiguraciones inseguras. El segundo tipo de error son aquellas posibilidades que son accedidas a través de posibilidades inválidas. Estas posibilidades se denominan posibilidades inalcanzables.

Con el propósito de gestionar los cambios en el entorno se utilizan los modelos de variabilidad [16]. Esta técnica de modelado permite definir las modificaciones que el sistema debe aplicar para ajustar su comportamiento en base a unas determinadas condiciones. Es decir, especifica las posibles configuraciones que puede tomar el sistema a lo largo de su ejecución.

Los principios básicos de la metodología de líneas de producto también proporcionan técnicas que contribuyen al desarrollo de software autónomo y reconfigurable. La característica principal de esta técnica es promocionar la reutilización de componentes. Se basa en gestionar por separado los elementos comunes que comparten todos los componentes de un sistema, que podrán ser reutilizados, y los elementos específicos que solo afectan a determinados componentes y que deberán diseñarse específicamente.

Aplicar la visión del paradigma de desarrollo dirigido por modelos pretende aprovechar las facilidades que este método de desarrollo software proporciona. El objetivo de esta metodología es automatizar la transformación de una especificación software abstracta en un producto software completamente funcional. De acuerdo a lo que esta metodología defiende, en este trabajo, los modelos serán la base de todo desarrollo.

En definitiva, siguiendo estas iniciativas se realizarán los cambios necesarios, refactorings, para conseguir un espacio de posibilidades libre de errores sin penalizar la funcionalidad del sistema. Las técnicas desarrolladas para conseguirlo se implementarán en una herramienta de soporte.

1.3 ESTRUCTURA DE LA TESINA

A continuación se comenta una breve descripción de los puntos más importantes a tratar y cómo quedan organizados en los siguientes apartados del documento.

- Capítulo 2: **Contexto**.

  En este capítulo se sitúa la propuesta de desarrollo dentro de un contexto. El objetivo es presentar la problemática que se pretende resolver y las características de la metodología usada para conseguirlo.

- Capítulo 3: **Panorámica de la propuesta**.

  Este capítulo provee los principales elementos y características de los enfoques relacionados con este trabajo, con el objetivo de proporcionar al lector los conceptos básicos para la comprensión del documento. Por una parte se definen todos los conceptos clave y por otra, los procesos básicos que se aplican sobre los elementos.

- Capítulo 4: **Refactoring de la reconfiguración**.

  Este capítulo constituye el objetivo del desarrollo de la propuesta. En él se analiza y describe con detalle cómo se realizan los procesos de refactoring y se acompaña con ejemplos paso a paso.

- Capítulo 5: **Herramienta de soporte a la propuesta**.

  Este capítulo presenta la aplicación de los procesos de refactoring descritos anteriormente haciendo uso de una herramienta que ha sido desarrollada en el contexto del proyecto Eclipse. Se describen una a una las diferentes perspectivas implementadas y la forma de utilizarlas.

- Capítulo 6: **Casos de estudio**.

  Este capítulo constituye un ejemplo completo de la aplicación de la propuesta desarrollada. Para ello, se toman un par de casos de estudio y se describe cómo evolucionan los modelos a lo largo del proceso. El primer caso trata un ejemplo simplificado y más intuitivo, mientras que el segundo caso pretende demostrar la escalabilidad de la propuesta.

- Capítulo 7: **Conclusiones**.

  Este capítulo presenta la contribución principal de este trabajo, los resultados obtenidos y las conclusiones alcanzadas.

- Capítulo 8: **Trabajos futuros**.

  En este capítulo se exponen otras áreas de investigación donde las técnicas, métodos y herramientas utilizadas en este trabajo pueden contribuir.

2. CONTEXTO

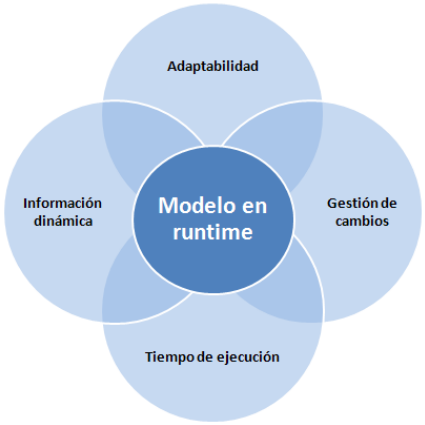


Figura 2 - Alcance del capítulo 2

En este capítulo se introduce el contexto en el que se sitúa el trabajo. Para ello, se expone inicialmente el entorno donde se ejecuta el tipo de sistemas software bajo estudio y los obstáculos que se pretenden superar. El entorno de ejecución está caracterizado por los continuos cambios y se espera que los sistemas software que lo forman se ajusten a ellos.

Posteriormente se propone una metodología que cumple los requisitos necesarios para trabajar eficazmente en estos entornos. Esta metodología son los modelos en tiempo de ejecución y se espera que sea capaz de afrontar los obstáculos que puedan surgir.

2.1 PLANTEAMIENTO DEL PROBLEMA

En la actualidad, las personas cada día dependen más de la funcionalidad proporcionada por el conjunto de sistemas software que nos envuelven. Estos sistemas deben ser robustos y ofrecer continuamente los servicios para los que han sido diseñados. Esta tarea puede parecer sencilla pero no lo es, ya que el entorno, las necesidades de los usuarios y sus objetivos cambian constantemente, por lo que se espera que el sistema software también lo haga.

Todos estos cambios en el comportamiento suponen un gran reto para los sistemas, ya que obligan a que el software disponga de algún mecanismo que le permita adaptarse a las nuevas necesidades para seguir cumpliendo su propósito. Tradicionalmente estos cambios obligan a detener la ejecución, realizar una serie de modificaciones en la configuración y volver a poner en marcha el sistema. Pero esto no siempre es posible. Para algunos sistemas críticos, su nivel de exigencia es tan elevado que detener la ejecución temporalmente constituye un grave problema, o bien, simplemente la alta frecuencia de los cambios no lo permite.

Es por ello, que cada vez más el software necesita adaptar dinámicamente su comportamiento en tiempo de ejecución para dar respuesta a las condiciones que constantemente van cambiando. Los sistemas deben tener en cuenta no solo los propios cambios a nivel de la infraestructura software sino también en el

entorno físico [1]. Una prometedora manera de gestionar la complejidad de estos cambios en entornos de ejecución es desarrollar mecanismos que permitan la adaptación del software. Estos mecanismos se caracterizarían por ser capaces de llevar a cabo los cambios contextuales sin detener la ejecución.

La dificultad de diseñar software capaz de cambiar el comportamiento de un sistema no es el único obstáculo a superar. Además se necesita que esas modificaciones de funcionamiento distraigan lo menor posible a los usuarios de esos servicios. La complejidad y el número de sistemas que nos rodean han alcanzado un nivel donde el esfuerzo humano necesario para su ejecución y mantenimiento esta fuera del alcance de las personas. Debido a ello, la responsabilidad de gestionar los cambios en el funcionamiento de los sistemas no puede recaer sobre el usuario, sino que debe ser el propio sistema el encargado de su auto-gestión.

En definitiva, no sólo nos enfrentamos al reto de diseñar software que adapte su comportamiento sino que además lo haga de una forma dinámica y autónoma.

2.2 SOLUCIÓN PROPUESTA

La metodología propuesta para dar respuesta a la problemática anterior son los modelos en runtime [17, 18]. Los modelos en runtime, también llamados modelos en tiempo de ejecución, cuentan con unas particulares asociadas que permiten solucionar los principales retos de este problema. Su principal característica es que son capaces de adaptar el sistema sin necesidad de detener su ejecución. Mediante estos modelos se pretende que los sistemas ajusten su comportamiento de forma dinámica y con la mínima intervención humana.

De forma genérica los modelos se utilizan para presentar una abstracción o visión reducida de un sistema que cumple un determinado propósito. Los modelos de desarrollo, producidos bajo la metodología de la Model-Driven Engineering (MDE), sirven habitualmente para apoyar el diseño de un determinado software. Están asociados con las entidades que intervienen durante el ciclo de desarrollo software. Por ejemplo, los modelos independientes de plataforma software se utilizan para describir sistemas pero dejando a un lado las características propias de las tecnologías afectadas.

Lo que se pretende es aprovechar las capacidades de los modelos MDE con el objetivo de dirigir el comportamiento autónomo del sistema en tiempo de ejecución. La idea de gestionar el comportamiento autónomo de los sistemas mediante los modelos basados en la aproximación MDE constituye la base de los modelos en tiempo de ejecución.

Formalmente un modelo en tiempo de ejecución es una representación causalmente conectada del sistema asociado que destaca la estructura, comportamiento, o metas del sistema desde una perspectiva de espacio del problema.

Los modelos en tiempo de ejecución, a diferencia de los tradicionales, no solo representan el sistema sino que además se utilizan para obtener información dinámica del estado y comportamiento del sistema durante su ejecución. Pueden verse como modelos de desarrollo que evolucionan dinámicamente y que realizan cambios sobre el diseño software.

La elección de los modelos en tiempo de ejecución para alcanzar el comportamiento autónomo de los sistemas software está fundamentada en dos razones:

- Si los modelos reflejan la arquitectura de un sistema y su contexto operacional, entonces los modelos pueden proporcionar información actualizada y exacta para dirigir las siguientes acciones.

- Si un modelo es un sistema conectado, entonces las adaptaciones pueden ser hechas a nivel de modelo en lugar de a nivel de sistema.

Cabe destacar que para que se cumplan las asunciones anteriores el modelo debe representar con exactitud el sistema gestionado. El modelo debe estar estrechamente conectado con el sistema, del tal modo que cualquier cambio producido en el sistema debe quedar también reflejado en el modelo, o viceversa.

Según lo anterior, el modelo puede ser usado para verificar que la integridad del sistema se mantiene cuando se aplica algún cambio. Por ejemplo, para garantizar que el sistema continuará operando adecuadamente antes de aplicar los cambios planificados. Esto es posible ya que los cambios se aplican en primer lugar sobre el modelo, el cual mostrará el estado del sistema tras la adaptación, incluyendo cualquier violación de las restricciones o de los requisitos. Si el nuevo estado del sistema es aceptable los cambios se llevarán a cabo sobre el propio sistema, consecuentemente el modelo y la implementación serán consistentes.

Como en el caso de los modelos de desarrollo tradicionales, un modelo en tiempo de ejecución soporta el razonamiento. Los usuarios pueden hacer uso de ellos para dar apoyo a la monitorización de estados dinámicos o para observar el comportamiento en tiempo de ejecución.

Los modelos en tiempo de ejecución también pueden tomar parte en la generación automática de artefactos de implementación que serán acoplados al sistema durante la ejecución e incluso por el propio sistema

El soporte que estos modelos dan a la evolución del diseño software puede empañar la línea entre modelos de desarrollo y modelos en tiempo de ejecución. Lo cual hace que puedan ser vistos como modelos de desarrollo vivos que permiten la evolución dinámica y la realización de diseños software.

Los modelos en tiempo de ejecución se caracterizan por:

- Se centran en el comportamiento.

  Mientras que los modelos tradicionales definen la estructura, estos modelos en tiempo de ejecución centran su atención en los eventos y actividades que tienen lugar en el sistema.

- Visión declarativa.

  Aunque los modelos son procedurales, están enfocados hacia los objetivos del sistema.

- Funcionales vs no-funcionales.

  En general los modelos tienden a centrarse en aspectos funcionales pero deben tener en cuenta que es igual de necesario capturar las características que representan aspectos no funcionales. Por ello, los modelos en runtime, no solo representan el sistema sino que además se utilizan para obtener información del estado actual y los cambios que pueden llevarse a cabo.

- Formales vs informales.

  Muchos de estos modelos están basados en representaciones formales. A pesar de ello, también se utilizan representaciones informales derivadas de modelos de programación o abstracciones del dominio. La ventaja de hacer uso de modelos formales es que dan soporte al razonamiento automático sobre el estado del sistema. En oposición puede que esta formalidad no sea adecuada para capturar o expresar de una forma tan elegante los conceptos del dominio.

Los modelos en tiempo de ejecución se utilizan en entornos que cambian continuamente y que de una forma autónoma y dinámica deben reconfigurarse. Especialmente donde se agregan y eliminan nuevos componentes con frecuencia. La tendencia de estos modelos es servir de apoyo para incrementar la automatización de la toma de decisiones con respecto a la adaptación del software.

Los modelos en tiempo de ejecución soportan el comportamiento autónomo de los sistemas cuando se lanzan producen cambios en el entorno. Teniendo en cuenta la naturaleza de estos modelos si se aplican en el dominio de este problema se pueden obtener resultados satisfactorios.

3. PANORÁMICA DE LA PROPUESTA

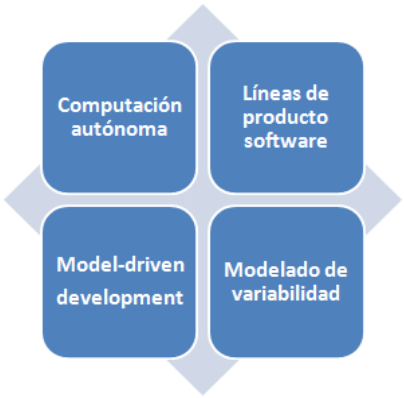


Figura 3 - Alcance del capítulo 3

En este capítulo se pretende ofrecer una visión panorámica de la propuesta presentada. Esta visión consiste en mostrar una definición de todos los elementos que la componen. Por lo tanto, este capítulo presenta los conceptos principales y las características con el objetivo de proporcionar una noción básica para entender todo el trabajo.

En primer lugar, se describen las técnicas y métodos que intervienen directa o indirectamente en la solución propuesta. El paradigma del desarrollo dirigido por modelos, los modelos de variabilidad, la computación autónoma o las líneas de producto son las piezas fundamentales que sustentan este trabajo.

En segundo lugar, se definen los conceptos principales con la terminología técnica que posteriormente se utilizará para referirse a cada uno de los componentes. Los conceptos de espacio de posibilidades, resolución o configuración son sólo una muestra.

En tercer lugar, se comenta el objetivo que se pretende alcanzar en este trabajo mediante el desarrollo de la propuesta. Conocer la metodología y los conceptos principales serán esenciales para entender cómo se alcanza el objetivo.

Finalmente, el resto del capítulo describe los procedimientos que internamente se aplican para la gestión y manipulación de los componentes.

3.1 ÁMBITO TECNOLÓGICO

A continuación, de forma resumida, se describen las disciplinas y técnicas más importantes que se ven involucradas en este trabajo. Los principios básicos que rigen estas metodologías constituyen los pilares sobre los que se sustenta la propuesta.

Computación Autónoma es una iniciativa lanzada por IBM en 2001. Su objetivo final es desarrollar sistemas informáticos capaces de auto-gestionarse para superar rápidamente la complejidad

creciente de la informática de gestión de sistemas, y para reducir la barrera que la complejidad supone para un mayor crecimiento.

Model Driven Development (MDD) es un paradigma para la captura de todos los aspectos importantes de un sistema software a través de modelos. Estos modelos no son sólo artefactos de documentación auxiliar, sino que son artefactos fuente y se puede utilizar para el análisis automatizado y/o la generación de código. El objetivo de este paradigma es traducir automáticamente una especificación abstracta del sistema en un producto software completamente funcional.

Líneas de producto software es una metodología de desarrollo software centrada en el desarrollo de componentes que puedan ser ampliamente reusables. Esta perspectiva cambia el enfoque tradicional del desarrollo de software para soluciones específicas al desarrollo de software que pueda ser adaptado y reutilizado para gran variedad de soluciones finales. La gestión de la variabilidad es el principio fundamental de las líneas de producto, las cuales implican la separación del producto en tres partes: componentes comunes, partes compartidas por algún pero no todos los productos y productos individuales con sus requisitos específicos.

Modelado de la variabilidad es una técnica de modelado cuyo objetivo es capturar los puntos de variación de un sistema que evoluciona. Mediante esta técnica se pretende que el sistema por sí mismo pueda adaptar su comportamiento para dar respuesta a los cambios que se vayan sucediendo. Estos modelos son ampliamente utilizados en el desarrollo software siguiendo la metodología de líneas de producto.

3.1.1 COMPUTACIÓN AUTÓNOMA

En octubre de 2001, IBM lanzó un manifiesto [8] que describe la visión de Computación Autónoma. El propósito es contrarrestar la complejidad de los sistemas de software haciendo que los sistemas se autogestionen. Pero paradójicamente esto hace que los sistemas sean aun más complejos. Esta complejidad, según se argumenta, se puede integrar en la infraestructura del sistema, que a su vez puede ser automatizada. La similitud del enfoque descrito con el sistema nervioso del cuerpo humano, que alivia el control básico de nuestra conciencia, dio a luz el término de Computación Autónoma [19, 20].

Inspirada por la biología, la computación autónoma ha evolucionado como una disciplina para la creación de sistemas software y aplicaciones que se autogestionan en un intento de superar las complejidades y la incapacidad de mantener los sistemas actuales y emergentes con eficacia.

En la propuesta de IBM surge el concepto de computación autónoma. En su manifiesto, los sistemas complejos computables eran comparados al cuerpo humano, el cual es un sistema complejo, pero cuenta con un sistema nervioso autónomo que cuida las funciones corporales sin necesidad de ser conscientes de todas las funciones. IBM sugirió que este sistema complejo computable debería también tener características autónomas, como por ejemplo ser capaz del mantenimiento y optimización de las tareas, reduciendo así la carga de trabajo de los administradores del sistema. IBM también concretó estas propiedades para la auto-gestión: auto-configuración, auto-optimización, auto-reparación y auto-protección.

Según Alan Ganek [21, 22] la computación autónoma:

> "Autonomic computing is the ability of systems to be more self-managing.
> The term autonomic comes from the autonomic nervous system, which controls many organs and muscles in the human body. Usually, we are unaware of its workings because it functions in an involuntary, reflexive manner –for example, we don't notice when our heart beats faster or our blood vessels change size in response to temperature, posture, food intake, stressful experiences and other changes to which we're exposed. And, by the way, our autonomic nervous system is always working"

A continuación se describen brevemente las principales propiedades de la computación autónoma según la visión de IBM. Para más información consultar [23, 24].

- Auto-configuración.

  Un sistema de computación autónoma se configura así mismo de acuerdo a los principales objetivos. Por ejemplo, indicando lo que se desea sin especificar necesariamente cómo hacerlo. 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 que pueden ser detectados son muy variados. Desde problemas en el hardware hasta problemas de tipo software [25].

- 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.

Se afirma que a medida que estos aspectos se convierten en propiedades de una arquitectura general se fusionarán en una sola cualidad de auto-mantenimiento. La arquitectura de un sistema de computación autonómica será una colección de componentes llamados elementos autónomos, los cuales encapsulan los elementos gestionados. Un elemento gestionado puede ser hardware, software o un sistema completo.

Un elemento autónomo es un agente, la gestión de su comportamiento interno y las relaciones con otros elementos se basa en unas políticas. Su funcionamiento está dirigido por unos objetivos, por otros elementos o por contratos establecidos mediante negociación con otros elementos. Un sistema de auto-gestión se deriva tanto de las interacciones entre los agentes como de los agentes internos de la autogestión.

Un sistema autónomo multi-agente se ejecutará en las arquitecturas orientadas a agentes de proporciones todavía desconocidas. Estas arquitecturas presentan numerosos retos de ingeniería que hay que resolver, con la participación de los ciclos de vida de los agentes y de la gestión de relaciones tales como la

negociación y la confianza, sólo por mencionar algunas. También hay retos científicos, como la inducción del comportamiento global, las teorías de control, el aprendizaje de las máquina, etc.

En IBM, la iniciativa de la computación autonómica se expande a través de todos los niveles de la administración de equipos, desde los niveles más bajos de hardware hasta los niveles más altos de los sistemas software. El IBM System Journal ha compilado una lista de este trabajo. A nivel de hardware, los sistemas se actualizan de forma dinámica. A nivel de software, el código del sistema operativo se sustituye de forma también dinámica. A nivel de aplicación, las bases de datos auto-validan los modelos de optimización y los servidores web se reconfiguran de forma dinámica gracias a los agentes para adaptar su funcionamiento al servicio actual.

Estos son solo algunos ejemplos para probar que, aunque algunas de las características propias de la computación autonómica parecen estar muy lejos, algunas de las ideas de esta nueva iniciativa ya se han puesto en práctica.

3.1.2 DESARROLLO DIRIGIDO POR MODELOS

La llegada de iniciativas como el desarrollo dirigido por modelos, más conocida como Model-Driven Development (MDD), pero también de la Model-Driven Architecture (MDA) está cambiando la forma de utilizar los modelos en el desarrollo de software. El MDD es un paradigma donde los modelos son el centro del desarrollo. La MDA es un framework para el desarrollo software propuesto por la Object Management Group (OMG) en 2001 [26] (MDA es una forma concreta de MDD. La noción de Model-Driven Engineering (MDE) surge posteriormente como un paradigma generalizador de la aproximación MDA para el desarrollo software [12].

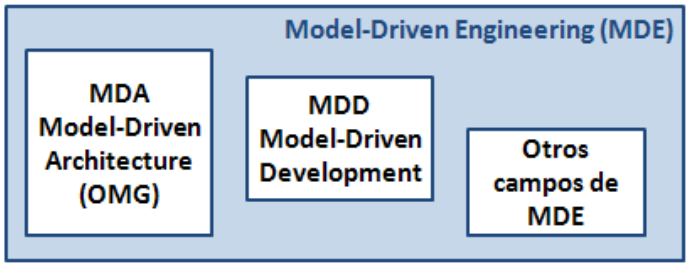


Figura 4 - Visión del paradigma MDE

La figura 4 muestra de forma simplificada como se relacionan los campos del paradigma MDE.

El tipo de proceso que promocionan estas corrientes es 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.

Según las palabras de Agrawal [27]:

> "The models are not merely artifacts of documentation, but living documents that are transformed into implementations. This view radically extends the current prevailing practice of using UML: UML is used for capturing some of the relevant aspects of the software, and some of the code (or its skeleton) is automatically generated, but the main bulk of the implementation is developed by

> hand. MDA, on the other hand, advocates the full application of models, in the entire life-cycle of the software product."

El objetivo de estos métodos es traducir automáticamente una especificación abstracta del sistema en un producto software completamente funcional.

El Model-Driven Software Development (MDSD) surge con la idea de poder construir un modelo de un sistema software que luego sea posible transformar en una cosa real [28]. Se han utilizado modelos durante mucho tiempo en el campo de desarrollo de software. Desde lenguajes de especificación formal y ejecutable (como OBLOG [29], TROLL [30] o OASIS [31]), que no han sido ampliamente aceptados por la industria, hasta las notaciones más aceptadas (como UML [32]) y procesos (como RUP [33]), modelos que están presentes en el área de desarrollo software.

Stuart Kent [12] define la ingeniería del desarrollo dirigido por modelos como una extensión de la arquitectura que apoya también visión pero con la noción de proceso de desarrollo de software (es decir, MDE emergió posteriormente como una generalización de la MDA para el desarrollo de software). MDE se refiere a la aplicación sistemática de modelos como los artefactos de ingeniería a través de todo el ciclo de vida de desarrollo. Kurtev cuestiona los actuales procesos de la ingeniería del desarrollo dirigido por modelos [34] ([35], [36] se refieren a un enfoque específico). En general, estos enfoques introducen conceptos, métodos y herramientas [37]. Todos ellos se basan en el concepto de modelo, meta-modelo, y la transformación de modelos.

La Model-Driven Architecture, como se mencionó anteriormente, es una especificación concreta del Model-Driven Development. La MDA clasifica los modelos en 2 clases: los modelos independientes de plataforma (PIMs) y los modelos específicos de plataforma (PSMs) [38]. Un PIM es "una visión de un sistema desde un punto de vista independiente de la plataforma". Del mismo modo, un PSM es "una visión de un sistema desde un punto de vista dependiente de la plataforma". En este caso, la definición de la plataforma se convierte en fundamental.

El desarrollo dirigido por modelos permite capturar todos los aspectos importantes de un sistema software a través de los modelos adecuados. Comparado con la implementación de código fuente los modelos son capaces de plasmar las necesidades del sistema de una forma más directa, sin entrar en detalles de implementación. La característica que destaca toda esta corriente es que los modelos no son artefactos auxiliares de documentación, sino que constituyen los artefactos fuente que serán transformados en el futuro código del programa.

3.1.3 LÍNEAS DE PRODUCTO SOFTWARE

La producción en masa fue popularizada por Henry Ford a principios del siglo XX. McIlroy acuño este término dentro del desarrollo software en masa en 1968 [39], este fue el principio de las líneas de producto software. En 1976, Parnas introdujo el concepto de familia de producto software como resultado de la producción de software en masa [40]. El uso de características para dirigir el desarrollo fue propuesto por Kang en la década de los 90 [41]. Poco después, aparecieron las primeras conferencias sobre esta novedosa aproximación [42, 43].

Las líneas de producto software, también conocidas como SPL (*Software Producto Line*), fueron definidas por Clements y Northfop como:

> "a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [13]

Esta definición puede ser descompuesta en sus cinco principales puntos:

1. **Productos**. "A set of software-intensive systems..."

   Las líneas de producto software cambian el enfoque del desarrollo centrado en un único sistema software por el desarrollo de líneas de producto. Los procesos de desarrollo no tienen la intención de construir una única aplicación, sino varias (por ejemplo, 10, 100, 10.000 o más). Esto obliga a un cambio en la ingeniería de procesos en la que se introduce una distinción entre la ingeniería del dominio y la ingeniería de aplicaciones. Esto supone, la construcción por un lado de los elementos reutilizables o comunes (plataforma) y por otro lado se introduce la variabilidad de la línea de productos.

2. **Características**. "...sharing a common, managed set of features…"

   Las características son las unidades (es decir, incrementos de funcionalidad de la aplicación) mediante las cuales se pueden construir productos distintos y definidos en una SPL [44].

3. **Dominio**. "...that satisfy the specific needs of a particular market segment or mission..."

   Una línea de producto software se crea dentro del ámbito de aplicación de un dominio. Un dominio es un ámbito especializado de conocimiento, una área de especialización, o una colección cuya funcionalidad está relacionada [45].

4. **Elemento básico**. "...are developed from a common set of core assets..."

   Son desarrollados a partir de un conjunto común de los elementos básicos. Un elemento básico es un artefacto o recurso que se utiliza en la producción de más de un producto en una línea de productos software [13].

5. **Plan de Producción.** "...in a prescribed way".

   Indica cómo ha sido producido cada producto. El plan de producción constituye una descripción de cómo los elementos básicos se van a utilizar para desarrollar un producto en una línea de productos y especifica cómo utilizar el plan de producción para construir el producto final [46]. El plan de producción conecta todos los elementos reutilizables para construir los productos finales. En síntesis es una parte del plan de producción.

Entonces, la metodología de las líneas de producto software se basan en producir familias de sistemas software similares en oposición al desarrollo individual.

La ingeniería de líneas de producto software se compone de tres actividades principales: ingeniería de dominio (también llamada desarrollo de activos básicos), ingeniería de aplicaciones (también llamada desarrollo de productos) y la gestión. Estas tres actividades son complementarias y proporcionan información entre ellas.

- **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 (por ejemplo, arquitectura,

modelos, código, etc.), así como para proporcionar los mecanismos adecuados para su reutilización en futuros sistemas [47].

Es decir, la ingeniería de dominio está relacionada con la identificación de los elementos comunes, la variabilidad de los productos en la línea de productos y la implementación de los elementos compartidos. De este modo es posible explotar los elementos comunes y enriquecer posteriormente el producto con los elementos variables.

Siguiendo la aproximación "diseñar para reutilizar", la ingeniería del dominio es la encargada de determinar los puntos comunes y variables dentro de la familia de productos. En general, la ingeniería de dominio se divide en análisis del dominio, análisis del diseño e implementación del dominio.

- **Ingeniería de la aplicación.**

  Es el proceso de construcción de un sistema dentro de un dominio. La ingeniería de la aplicación es responsable de derivar un producto concreto a partir de una línea de producto utilizando una aproximación diseñar para reutilizar. Para conseguirlo, se utilizan los elementos reutilizables desarrollados previamente.

  Durante la ingeniería de la aplicación, los productos individuales son desarrollados mediante la selección de artefactos de configuración compartidos y, donde sea necesario, añadiendo extensiones especificas del producto. Este proceso está dividido en análisis, diseño e implementación de la aplicación.

- **Gestión.**

  Es un proceso separado donde los temas organizacionales se manejan de forma específica. Este proceso es el responsable de dar recursos, coordinar y supervisar el dominio y aplicaciones de las actividades de ingeniería.

Puede consultarse [13, 48] para obtener más información acerca de estos procesos.

Las líneas de producto constituyen 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.

3.1.4 MODELADO DE VARIABILIDAD

El modelado de la variabilidad se considera como la tecnología instrumental para el desarrollo de una amplia variedad de sistemas software de una manera consistente e integral [16]. La idea clave consiste en separar por una parte los puntos comunes del sistema y por otra los puntos de variación. De este modo, se construyen los elementos comunes mientras que los elementos variables se expresan de una forma eficiente para que puedan ser gestionados cuando corresponda.

Se consideran puntos comunes aquellas asunciones sobre componentes con la misma especificación en todos los sistemas. En oposición, se consideran puntos de variación aquellas asunciones verdaderas en solo algunos sistemas, tal como un componente con diferente especificación para al menos dos sistemas.

Los modelos de variabilidad permiten describir las diferentes variantes de ejecución que un sistema puede tomar. La idea es que el sistema sea capaz de responder a los cambios que se den en el contexto. Es decir, que el sistema por sí mismo pueda consultar estos modelos de variabilidad con el objetivo de determinar los cambios necesarios en su arquitectura.

El modelado de la variabilidad permite definir una serie de propiedades sobre el sistema a representar. En estos modelos los elementos están organizados de forma jerárquica, generalmente en una estructura tipo árbol. Su notación hace posible expresar restricciones acerca de la relación entre diferentes elementos enlazados, como por ejemplo obligatoriedad u opcionalidad. Pero también, sobre elementos que no están conectados directamente haciendo uso de mecanismos de dependencia, como por ejemplo requiere o excluye.

De las diferentes técnicas que están disponibles para el análisis de variabilidad, se ha elegido un lenguaje de modelado basado en características [49]. El modelado de características se utiliza ampliamente para la especificación de la funcionalidad de un sistema en un nivel de abstracción elevando.

3.1.4.1 MODELADO DE LA VARIABILIDAD Y LÍNEAS DE PRODUCTO SOFTWARE

El modelado de la variabilidad está estrechamente relacionado con las líneas de producto. Las líneas de producto software ofrecen los métodos, herramientas y técnicas para la creación y mantenimiento de una colección de sistemas software similares a partir de un conjunto de activos de software compartidos.

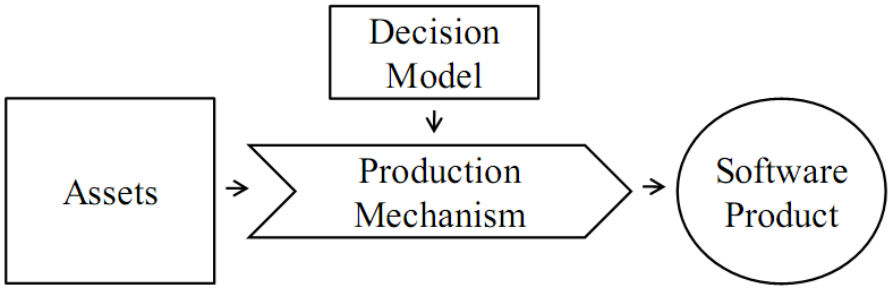


Figura 5 - Conceptos principales de las líneas de producto software

Las líneas de producto software pueden ser descritas en base a cuatro conceptos, como ilustra la figura 5.

- **Activos software de entrada** (Assets).

  Una colección de activos software (como requisitos, componentes de código fuente, casos de uso, arquitectura y documentación) que pueden ser configurados y compuestos de diferentes formas para crear todo tipo de productos en una línea de producción. Cada uno de estos activos tiene un rol bien definido dentro de la arquitectura de la línea. Para dar cabida a la variación de los productos, algunos de estos activos pueden ser opcionales y otros pueden ofrecer puntos de variación. Esto permite obtener variantes para proporcionar diferentes comportamientos.

- **Modelo de decisión** (Decision Model).

  Las decisiones describen características opcionales o variables para los productos de la línea de producción. Cada producto en cada línea de producción es de forma única por sus decisiones de producción (las opciones para cada característica opcional y variable en el modelo de decisión).

- **Mecanismo de producción y proceso** (Production Mechanism).

  Los mecanismos para componer y configurar productos de los activos software de entrada. Las decisiones de producción se utilizan durante el proceso de producción para determinar qué activo software de entrada se utiliza y cómo, para configurar los puntos de variación dentro de dichos activos.

- **Producto software de salida** (Software product).

  La colección de todos los productos que pueden obtenerse de una línea de producto. El objetivo de las líneas de producto es determinar el conjunto de productos software de salida que se pueden producir a partir de unos activos software y su modelo de decisión.

Algunas de estas ideas proceden de aproximaciones como Program Factoring [50] o Domain Engineering [51]. La característica diferenciadora que distingue las líneas de producción de otros esfuerzos previos es su postura predictiva frente a la oportunista de reutilización software. Hasta el momento estas aproximaciones se limitaban a poner componentes software generales en una librería esperando que pudieran ser reutilizados. En cambio, en las líneas de producción software solo crean artefactos software cuando está previsto utilizarlos en uno o más productos en una línea bien definida.

Con este propósito, los principales objetivos de las líneas de producto software son:

- **Capitalizar en común** mediante la consolidación y compartición dentro de los activos de un sistema, evitando así la duplicación y divergencia.

- **Gestionar la variación** para reducir el tiempo, esfuerzo, coste y complejidad de crear y mantener una línea de producción de sistemas software similares. Esto se logra definiendo claramente los puntos de variación y el modelo de decisión, con la ubicación y dependencias para una variación determinada.

En definitiva, la ingeniería de la línea de producción software (SPLE) optimiza el desarrollo de sistemas individuales dentro de una dominio de aplicación mediante la promoción de características comunes y la gestión de las diferencias de una forma sistemática. En la SPLE, sistemas individuales pueden construirse rápidamente a partir de activos reusables, como un conjunto de componentes y/o una plataforma común.

3.1.4.2 MODEL DRIVEN Y LÍNEAS DE PRODUCTO SOFTWARE

El desarrollo de software generativo [52] y otras aproximaciones relacionadas, como Software Factories [53], han ido propagando la integración de las líneas de producto software y el desarrollo dirigido por modelos. Para obtener más información acerca de este tema se puede consultar [54, 55].

La ingeniería de SPL y el MDD, ambas técnicas explicadas con detalle en apartados previos, no son solo complementarias, sino que además su integración supone un gran potencial para alcanzar grandes metas. Mientras el MDD facilita la representación de los diferentes aspectos de una línea de producto de una forma abstracta, la SPL proporciona una definición concreta del alcance de la aplicación. Esto supone una base sólida para la elaboración y selección de lenguajes de modelado [56].

Las ventajas clave de utilizar MDD en sintonía con la variabilidad de SPS son, por una parte, capturar de forma rigurosa los elementos comunes y variantes de una familia de sistemas; por otra parte, ayudar a automatizar las tareas repetitivas que deben cumplirse para cada instancia de un determinado productos.

3.2 CONCEPTOS BÁSICOS

En este segundo apartado se presenta una descripción de todos los conceptos y terminología utilizada para exponer la propuesta.

3.2.1 MODELADO DE CARACTERÍSTICAS

Los modelos de características forman parte de una técnica de modelado de la variabilidad cuyo objetivo central es capturar los puntos de variación de un sistema. Fueron introducidos por primera vez en Feature-Oriented Domain Analysis method (FODA) por Kang en 1990 [41].

Un modelo de características es una representación compacta de todos los elementos del sistema en términos de características. Los modelos se representan gráficamente por medio de diagramas de características.

Los modelos de características son ampliamente utilizados durante toda la línea de proceso de desarrollo de productos donde habitualmente se utilizan como entrada para producir otros bienes, como los documentos, la definición de la arquitectura, o piezas de código.

Los diagramas vienen definidos en base a:

- **Concepto.**

  Un concepto es el conocimiento sobre las propiedades de un objeto y que permite categorizarlo.

- **Característica.**

  Una característica es una propiedad importante de un concepto. Un modelo de características representa características comunes y variables de un concepto, así como las dependencias entre las características variables.

- **Diagrama de características**.

  Un diagrama de características es una representación visual de estos conceptos, características y relaciones. Captura los puntos de variación de un sistema. El diagrama puede tener tantos niveles como sea necesario para describir la variación del dominio que se está modelando.

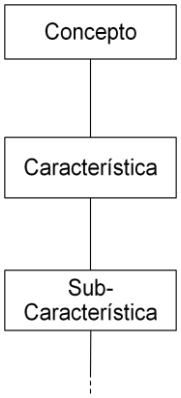


Figura 6 - Estructura modelo de características

La figura 6 representa la estructura que sigue un modelo de características.

El modelado de características [57] es ampliamente usado para describir el conjunto de productos en una línea de producto software en términos de características. En estos modelos, las características están conectadas jerárquicamente como en una estructura de árbol, a través de relaciones de variabilidad, como opcional, obligatoria, única o múltiple opción. Además de poder incluir opcionalmente restricciones excluyentes o requeridas.

La notación del modelo permite definir una serie de propiedades y restricciones también de forma gráfica:

- **Restricción opcional.**

  La restricción opcional, figura 7, indica que la activación de la característica hija es opcional, no necesaria, a la activación del padre.

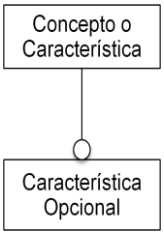


Figura 7 - Restricción opcional

- **Restricción obligatoria.**

  La restricción obligatoria, figura 8, indica que la activación de la característica padre obliga a la activación de la característica hija.

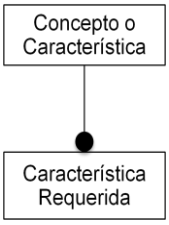


Figura 8 - Restricción obligatoria

- **Restricción or.**

  La restricción or, figura 9, permite indicar que la activación de la característica padre supone la activación de una o más características hijas.

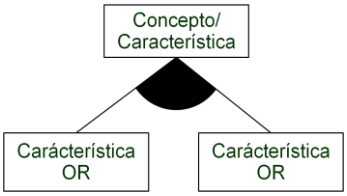


Figura 9 - Restricción OR

- **Restricción alternativa.**

  La restricción alternativa (xor), figura 10, indica que la activación de la característica padre implica la activación de una y sólo una de las alternativas hijas.

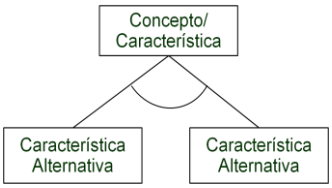


Figura 10 - Restricción alternativa

Las restricciones vistas hasta ahora tienen lugar en relaciones de padre-hijo entre características. Las dos siguientes pueden afectar a características en cualquier nivel de la jerarquía.

- **Características excluyentes**

  Esta restricción, figura 11, se utiliza para indicar que la activación de una característica implica que no se pueden activar el resto de características con las que se excluye.

Figura 11 - Restricción excluye

- **Características requeridas**

  Esta restricción, figura 12, se utiliza para indicar que una determinada característica necesita de la activación de otra/s para su activación.

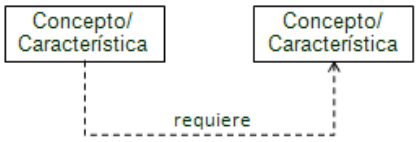


Figura 12 - Restricción requiere

En la figura 13 se presenta un modelo de características muy simple que engloba todos los conceptos que se han comentado hasta el momento.

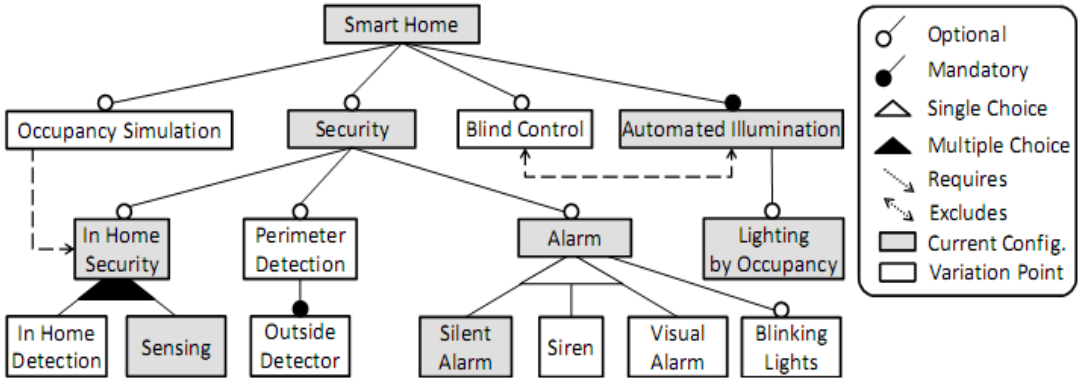


Figura 13 - Ejemplo modelo de características

Mediante [[FM]] se denota todas las características (activas o inactivas) en un modelo de características. Se define la configuración actual, CC (Current Configuration), de un sistema como el conjunto de características (F) que están activas en el modelo.

$$CC = \{F\} \mid F \epsilon F \in [[FM]] \wedge F.estado = Activo \wedge CC \subseteq FM\}$$

Por ejemplo, la CC del modelo de características de la figura 13 se expresaría:

$$CC_{fig13} = \{SmartHome, Security, InHomeSecurity, Sensing, Alarm, AutomatedIllumination, SilentAlarm, LightingByOccupancy\}$$

3.2.2 RESOLUCIÓN

Se entiende por resolución (R) la representación de un conjunto de cambios sobre el estado de diferentes elementos del sistema. Está formada por una lista de pares (F, S), donde F indica una característica (feature) y S el estado (state) de la característica.

Cada resolución está asociada a una condición de contexto y representa un cambio en la configuración del sistema, en términos de activación/desactivación de características. La ejecución de una resolución viene determinada por el cumplimiento de una determinada condición.

La representación formal de una resolución es la siguiente:

$$R_x = \{(F, S)\} \epsilon F \in [FM] \wedge S \in \{Activo, Inactivo\}$$

La resolución R_x está compuesta por un conjunto de pares (F, S) donde F es una de las características del modelo FM y S indica el estado, activo o inactivo, de dicha característica. La definición de una resolución puede afectar parcial o completamente las características del sistema.

A continuación se presenta la estructura de una resolución:

$R_{condición}$ = {(Característica_1, Estado_1), (Característica_2, Estado_2), (Característica_3, Estado_3), (Característica_n, Estado_n)}

Sobre un caso práctico por ejemplo, una SmartHome, se podrían encontrar resoluciones de la forma:

$R_{EmptyHome}$ = {(PresenceSimulation, Active), (InHomeDetection, Active), (LightingByPresence, Inactive)}

$R_{Comfort}$ = {(PipedMusic, Active), (AutomatedIllumination, Active)}

La resolución $R_{EmptyHome}$, indica que, cuando la Smart Home está vacía (condición), debe modificar su configuración para desactivar la iluminación por presencia y activar los servicios de simulación de presencia y detección de intrusos en el hogar.

La resolución $R_{Comfort}$, indica que, cuando la Smart Home está en modo confort (condición) se deben activar los servicios de iluminación automática y el hilo musical.

Las dos resoluciones anteriores son ejemplos del tipo de cambios, reconfiguración, que se pueden llevar a cabo mediante resoluciones.

3.2.3 CONFIGURACIÓN

Una configuración representa el estado del sistema en un determinado instante de tiempo. Está formada por un conjunto de pares (F, S), donde de nuevo F indica la característica y S indica el estado de esa característica. Las diferentes combinaciones entre características activas e inactivas constituyen el conjunto de configuraciones que puede tomar el sistema. Cada una de dichas configuraciones diferentes forma, lo que más tarde se denominará como posibilidades.

El estado general del sistema o configuración queda reflejado a través de las características activas e inactivas del sistema. A diferencia de una resolución, una configuración contiene el estado de todas las características del sistema.

A continuación se presenta una configuración de ejemplo:

$Configuración_{ejemplo}$ = {(Característica_1, Estado_1),
(Característica_2, Estado_2),
…
(Característica_n, Estado_n)}

La configuración del sistema puede cubrir tres objetivos, de ahí que se definan tres tipos de configuraciones.

- **Self-configuring**

  Este tipo de configuración tiene como objetivo que los nuevos tipos de dispositivos puedan ser incorporados al sistema. Esta incorporación debe ser lo más transparente al usuario posible.

Por ejemplo, cuando un nuevo sensor de presencia se agrega en un lugar de la casa, los diferentes servicios para el hogar inteligente, como la seguridad o el control de iluminación, automáticamente pueden hacer uso de él sin necesidad que el usuario lo tenga que configurar.

- **Self-healing.**

  Este tipo de configuración pretende que cuando un dispositivo se retira o falla, el sistema sea capaz de adaptarse para ofrecer sus servicios de forma alternativa, y así reducir el impacto de la pérdida del dispositivo.

  Por ejemplo, si una alarma falla, la Smart Home pueda encender y apagar las luces como una alternativa a la alarma fallida.

- **Self-adapting.**

  Este tipo de configuración se basa en que las necesidades del usuario son diferentes dependiendo del usuario y del momento dado. El sistema debe ser capaz de ajustar sus servicios para cumplir con las preferencias del usuario.

  Por ejemplo, cuando un usuario sale de casa, los servicios del hogar deben ser reorganizados para dar prioridad a los sistemas de seguridad.

Se puede ver fácilmente que la estructura que representa la configuración del sistema en un momento dado es similar a la estructura de las resoluciones. La diferencia entre ambos conceptos estriba en que las configuraciones son representaciones completas del estado del sistema. Mientras que las resoluciones se pueden ver como acciones que se aplican sobre las configuraciones, bajo unas condiciones, para generar una serie de cambios. El proceso de aplicar una o más resoluciones sobre una configuración se define como reconfiguración.

La figura siguiente presenta un ejemplo para dejar claros ambos conceptos:

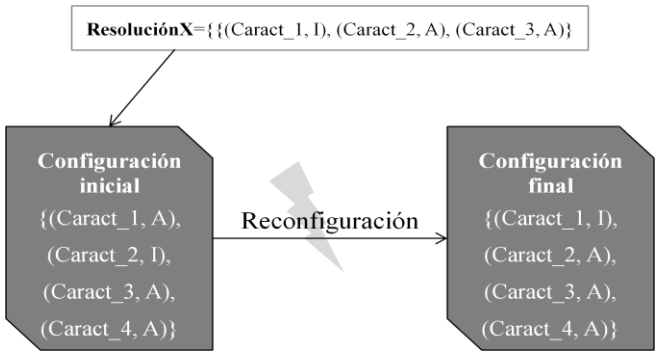


Figura 14 - Ejemplo de reconfiguración

La figura 14 ejemplifica el proceso de reconfiguración. A partir de una configuración *Configuración inicial* se aplica una resolución *ResoluciónX* que dispara un conjunto de cambios. Es decir, supone una reconfiguración del sistema, generando una nueva configuración *Configuración final*.

La configuración de partida cuenta con cuatro características: Caract_1, Caract_2, Caract_3 y Caract_4, todas activas excepto la Caract_2 que permanece inactiva.

La resolución, que se ejecutará cuando se cumpla su condición asociada, dispara la activación de la Caract_2 y Caract_3 y la desactivación de la Caract_1. Esta resolución aplicada sobre la *Configuración inicial* genera una reconfiguración llevando el sistema hasta una nueva configuración, *Configuración final*.

La Configuración resultante refleja el cambio de estado en las características. La característica Caract_2, hasta ahora inactiva, ha pasado a estar activada. La característica Caract_1, hasta ahora también activa, ha pasado a inactiva. La característica Caract_3 mantiene su estado ya que su estado inicial coincide con el estado indicado por la reconfiguración, por lo tanto, continúa activada.

Aquellas características que no se han visto afectadas por la resolución mantienen su estado previo a la reconfiguración, esto ocurre para Caract_4.

Como se puede apreciar, ambas configuraciones contienen todas las características definidas en el sistema, mientras que la resolución afecta solo a algunas de ellas. Aunque también es posible definir resoluciones que realicen cambios sobre todas.

3.2.4 POSIBILIDAD

Una posibilidad es la representación de los estados por los que puede ir pasando el sistema con el tiempo. Está especificada mediante una configuración y una lista de resoluciones.

Una posibilidad está compuesta por una configuración que, tal como se ha explicado en el punto anterior, mantiene el estado de las características del sistema. Además también cuenta con una lista de resoluciones que podrán ser aplicables sobre dicha configuración.

La aplicación de cada una de las resoluciones de la lista sobre su configuración generará nuevas configuraciones que quedarán representadas a través de posibilidades. Es decir, se obtendrán nuevas posibilidades cuya configuración vendrá dada por la resolución que la ha generado.

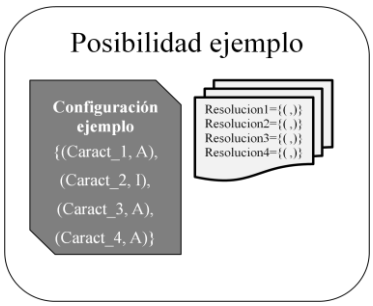


Figura 15 - Ejemplo de posibilidad

La figura 15 representa de forma visual los elementos que forman una posibilidad. En esta posibilidad de ejemplo, por una parte se puede ver el conjunto de pares (Característica, Estado) que conforman su configuración y por otra el listado de resoluciones.

3.2.5 ESPACIO DE POSIBILIDADES

Un espacio de posibilidades es la representación del modelo de características en términos de posibilidades y reconfiguraciones.

Está formado por un conjunto de estados (posibilidades) y sus respectivas transiciones (reconfiguraciones) que permiten la navegación entre los diferentes estados.

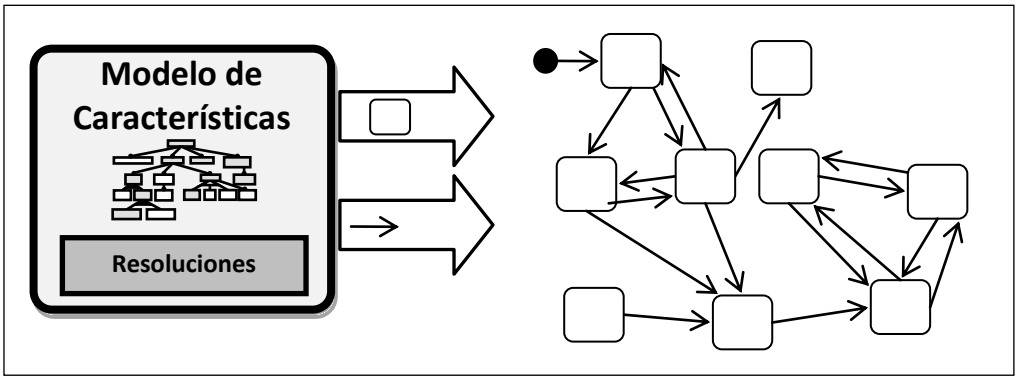


Figura 16 - Transformación del modelo de características al espacio de posibilidades

La figura 16 ofrece una visión conceptual del proceso de transformación. A partir del modelo de características, donde quedan representados en términos de características todos los servicios y elementos del sistema se aplican una serie de transformaciones. Estas transformaciones llevan al modelo a una nueva representación basada en un espacio de posibilidades.

Un espacio de posibilidades, a rasgos generales, no es más que la representación de todas las posibles combinaciones que pueden darse en el sistema.

El espacio de posibilidades surge a partir de la aplicación sucesiva de resoluciones sobre las posibilidades del sistema. Inicialmente se dispone de una única posibilidad que contiene una determinada configuración, habitualmente la configuración inicial o actual. Como se ha comentado en el apartado 3.2.4 cada posibilidad cuenta con una lista de resoluciones. La aplicación de cada una de estas resoluciones genera nuevas posibilidades.

Es importante comentar que para cada posibilidad solo se disparan las resoluciones asociadas cuya condición de contexto se cumpla. Es decir, su ejecución vendrá determinada por el cumplimiento de la condición. Esto supone que en algunas posibilidades puede que no se disparen todas las resoluciones de su lista.

Las nuevas posibilidades quedan conectadas a la posibilidad que, junto a una resolución, ha generado la nueva configuración. Cada posibilidad tendrá una configuración única e irrepetible dentro del modelo de características.

Como ejemplifica la figura 17 a partir de la posibilidad inicial, posibilidad_0, se han disparado las resoluciones: resolución_1, resolución_2, resolución_3… resolución_N. Estas resoluciones han generado nuevas configuraciones y quedan representadas en las posibilidades: pos_1, pos_2, pos_3…

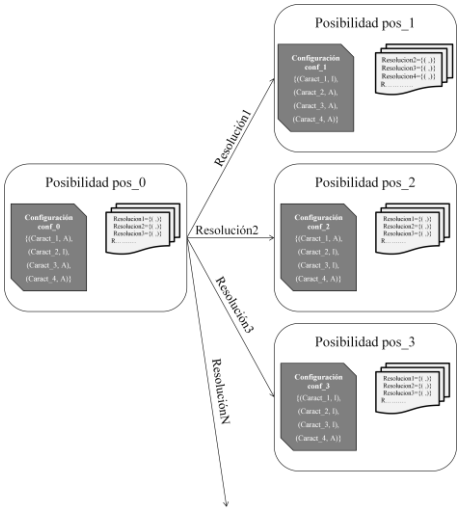


Figura 17 - Ejemplo espacio de posibilidades

Las posibilidades creadas a su vez podrán generar nuevas posibilidades a partir de la ejecución de su lista de resoluciones sobre su configuración. Si la aplicación de alguna resolución genera una configuración ya existente no se crea una nueva posibilidad, sino que se conecta con la posibilidad existente.

Como se aprecia en la figura 18 a partir de cada posibilidad se disparan unas resoluciones que generan nuevas configuraciones del sistema representadas en forma de posibilidades. Además aquellas resoluciones que generan configuraciones ya creadas a partir de otras posibilidades se conectan directamente a ellas sin replicar la posibilidad. Nótese que cada posibilidad debe ser única, ya que, representa una configuración posible dentro del sistema.

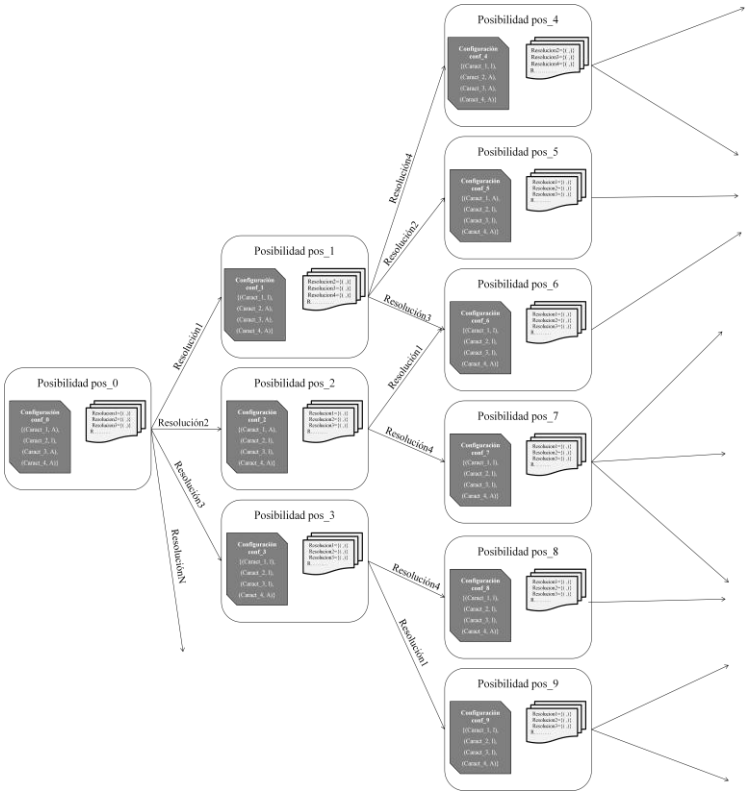


Figura 18 - Ejemplo espacio de posibilidades

El espacio de posibilidades continuará creciendo hasta que para todas las posibilidades se hayan aplicado la lista de resoluciones que disponen y sea posible obtener nuevas configuraciones. Llegado a este punto se completa el proceso de obtención del espacio de posibilidades.

Finalmente el espacio de posibilidades representará todas las posibles configuraciones que pueden darse en el sistema. El tamaño del espacio varía en función del número de resoluciones que se definan.

3.2.6 ESPACIO DE POSIBILIDADES ABSTRACTO

Un espacio de posibilidades abstracto consiste en la simplificación del espacio de posibilidades original. Esta simplificación se hace en base a unas determinadas propiedades que coinciden de unas posibilidades a otras. Su objetivo es aportar una vista más simplificada, y por lo tanto, más comprensible.

El tamaño de un espacio de posibilidades puede resultar excesivamente grande provocado por el número de resoluciones definidas. Las resoluciones incrementan la variedad de configuraciones y con ellas el número de posibilidades. Esto puede derivar en numerosas posibilidades que conforman un espacio difícilmente manejable.

Mediante la transformación de un espacio de posibilidades a un espacio de posibilidades abstracto se pretende aportar una versión reducida y que sea más fácil de tratar. Esta simplificación se caracteriza por que la nueva representación mantiene la información acerca de las propiedades más importantes. En próximos apartados se detallará el proceso de transformación para alcanzar la representación abstracta.

En definitiva un espacio de posibilidades abstracto es una visión simplificada de un espacio de posibilidades.

3.2.7 REFACTORING

En el ámbito de la ingeniería del software se denomina refactoring al proceso de reestructurar o modificar un código fuente con el objetivo de mejorar su consistencia interna y claridad. Pretende mejorar la comprensión del código o cambiar su estructura y diseño para facilitar el mantenimiento en el futuro. No añade nuevas funcionalidades. Este proceso altera su estructura interna sin cambiar su comportamiento externo, es decir, los cambios realizados no deben afectar su funcionalidad.

En lo que a este trabajo respecta se utiliza el término refactoring para referirse a los procesos que permiten redefinir el comportamiento de determinados componentes. Los refactorings aquí explicados se llevan a cabo para evitar o resolver situaciones no deseadas en el sistema.

3.3 DESARROLLO DE LA PROPUESTA

Desde la especificación del modelo de características hasta su representación en el espacio de posibilidades se llevan a cabo unos procesos intermedios que inevitablemente introducen errores.

El espacio de posibilidades resultante puede contener errores procedentes de la fase de modelado o del proceso de transformación. Estos errores provocan que algunas posibilidades sean etiquetadas como inválidas y que algunas resoluciones se marquen como inseguras.

Se considera que una resolución es insegura cuando incumple las restricciones impuestas y consecuentemente, conduce al sistema a estados no deseados. La configuración de dichos estados, representados como posibilidades, se marca como inválida.

El objetivo de esta propuesta es, por una parte detectar y corregir aquellas posibilidades que llevan al sistema a configuraciones no deseadas. Por otra parte, que los mecanismos aplicados para resolver dichas configuraciones no perjudiquen la funcionalidad del conjunto del sistema. Además para favorecer la aplicación de los mecanismos desarrollados se implementará una herramienta de soporte.

Para alcanzar este objetivo se aplican los principios básicos de los paradigmas presentados anteriormente.

El éxito de los mecanismos implementados vendrá determinado por los resultados derivados de su aplicación en casos de estudio que reflejen un sistema real.

3.4 VALIDACIÓN DE LAS CONFIGURACIONES

Como se ha comentado anteriormente, las posibilidades están formadas, entre otros elementos, por configuraciones. A su vez estas configuraciones están formadas por pares de características-estado. En conjunto el estado de las características asociadas a una configuración deben ser coherentes con el modelo de características, de tal modo que se ajuste a las restricciones que este impone.

Con el objetivo de cumplir estas restricciones las configuraciones deben ser validadas antes de incorporarse al modelo. Esta validación se lleva a cabo contra el modelo de características que se ha definido.

Los modelos de características cuentan con unos mecanismos que permiten definir propiedades y restricciones en el propio modelo. Estas propiedades y restricciones deben ser cumplidas por todas las configuraciones que puedan darse a lo largo de la ejecución del modelo. De este modo se controla que todos los posibles estados queden dentro de lo previsto.

A continuación se van a explicar las restricciones más habituales en los modelos de características. Para facilitar su comprensión se acompañan fragmentos extraídos del modelo de características del ejemplo presentado en la figura 13.

- **Restricción de opción**

  Esta restricción, figura 19, viene representada por una bola blanca. Se utiliza entre características que cuya relación es de padre-hijo. Indica que la característica hija es opcional a la activación de la característica padre. De tal modo, que podrá activarse la característica padre independientemente de la característica hija.

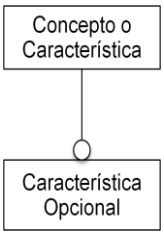


Figura 19 - Restricción de opción

La figura 20 es un fragmento extraído del modelo de características tomado como ejemplo. El significado de esta restricción aplicado sobre el fragmento supone que la activación de la característica *Perimeter Detection* es opcional para la ejecución de su característica padre *Security*.

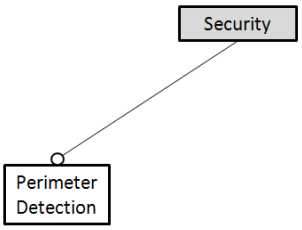


Figura 20 - Fragmento del ejemplo: opcional

- **Restricción de obligación.**

  Esta restricción, figura 21, se representa mediante una bola negra. Al igual que la anterior se utiliza entre características enlazadas por una relación padre-hijo. Indica que la característica hija actuará de forma equivalente a como lo hace su características padre. Es decir que se activará o desactivará siguiendo el comportamiento de su característica padre.

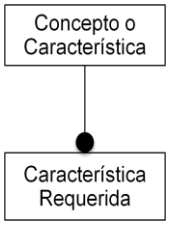


Figura 21 - Restricción de obligación

Esta restricción aplicada sobre el ejemplo, figura 22, significa que la característica *Outside Detector* es obligatoria para su característica padre *Perimeter Detection*. Esta restricción fuerza la ejecución de *Outside Detector* cuando se active *Perimeter Detection.*

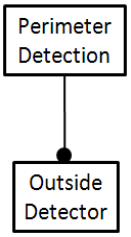


Figura 22- Fragmento del ejemplo: obligación

- **Restricción de características alternativas.**

  Esta restricción, figura 23, queda representada mediante un arco blanco entre las posibles características. Se utiliza en relaciones padre-hijo, donde habitualmente intervienen varias características hijas. Indica que la activación de padre supone la activación de una y sólo una de las hijas.

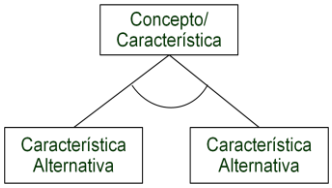


Figura 23 - Restricción alternativa

Esta misma restricción aplicada sobre el ejemplo, figura 24, significa que para la característica *Alarm* es necesario que una y sólo una de las características hija *Silent Alarm, Siren* o *Visual Alarm* esté activa cuando el padre lo esté.

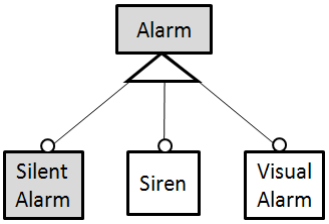


Figura 24 - Fragmento del ejemplo: alternativa

- **Restricción OR.**

  Esta restricción, figura 25, queda representada mediante un arco negro entre las características. Se aplica en relaciones padre-hijo, donde al igual que en la anterior suelen intervenir varias características hijas. Se utiliza para indicar que la activación de la característica padre supone la activación de al menos una de las hijas.

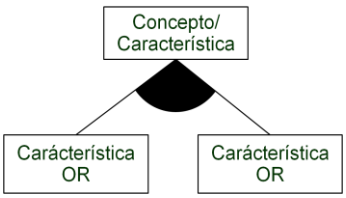


Figura 25 - Restricción OR

Esta restricción aplicada sobre el ejemplo, figura 26, indica que cuando se active la característica padre *In Home Security* pueden activarse *In Home Detection* y/o *Sensing* pero necesariamente una de las dos debe activarse.

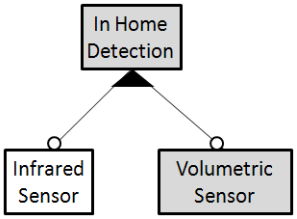


Figura 26 - Fragmento del ejemplo: OR

- **Restricción requiere.**

  Esta restricción, figura 27, se representa mediante una línea discontinua entre las características afectadas. Se aplica sobre características que pueden o no estar en diferentes niveles de la jerarquía pero no existe una relación parental entre ellas. Se utiliza para indicar que una característica requiere de otra para estar activa. La dirección de la fecha señala que característica es la requerida.

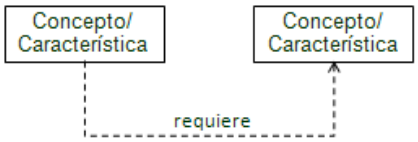


Figura 27 - Restricción requiere

Un ejemplo de este tipo de restricción lo podemos ver en el fragmento de la figura 28. La ejecución de *Occupancy Similation* requiere de la activación *In Home Security*.

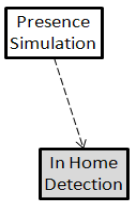


Figura 28 -Fragmento del ejemplo: requiere

- **Restricción excluye.**

  Esta restricción, figura 29, se representa mediante una línea discontinua bidireccional entre las características afectadas. Del mismo modo que la anterior se aplica sobre características que pueden o no estar en diferentes niveles de la jerarquía pero no existe una relación parental entre ellas. Se utiliza para indicar que la activación de una característica excluye la activación de otra.

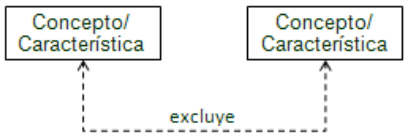


Figura 29 - Restricción excluye

Un ejemplo de esta restricción la encontramos en la figura 30 entre las características *Blind Control* y *Automated Illumination*. La activación de una de las características enlazas por la restricción excluye impide que la otra característica enlazada se active simultáneamente. De este modo, si se activa *Blind Control* no puede activarse *Automated Illumination* y de forma equivalente si la primera en activarse fuese *Automated Illumination*.

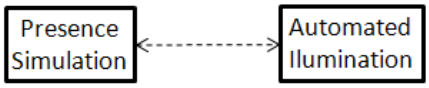


Figura 30 - Fragmento ejemplo: excluye

3.5 CÁLCULO DEL ESPACIO DE POSIBILIDADES

Para la obtención del espacio de posibilidades se requiere, por una parte del cálculo de las posibilidades y por otra del cálculo de las resoluciones. Es importante destacar que ambos realizados de forma simultánea permiten un cálculo más ágil.

3.5.1.1 CÁLCULO DE LAS POSIBILIDADES

Para realizar el cálculo del espacio de posibilidades es necesaria la configuración actual (CA) del modelo de características y la lista de las resoluciones ($[R_x]$).

A partir de estos dos componentes del sistema, se puede comenzar con el proceso que nos llevará a la obtención del espacio de posibilidades. A continuación se detallan los pasos.

En primer lugar, determinar una posibilidad origen o posibilidad 1. Ésta representa la configuración inicial del sistema.

El segundo paso consiste en partiendo de la posibilidad inicial ir aplicando de una en una las resoluciones de la lista proporcionada anteriormente. Cada resolución nueva que se aplique es una transición a una nueva posibilidad y un nuevo camino en el diagrama de posibilidades.

El tercer paso es comprobar si las nuevas posibilidades obtenidas son realmente nuevas o ya se han generado en algún otro camino del diagrama. Para comprobar si una posibilidad es nueva o ya existe se comprobarán todos los pares (Característica, Estado) resultantes de la unión CA y las diferentes resoluciones. Si todos los pares coinciden con los de alguna otra posibilidad se descarta. Pero solo con que la posibilidad generada difiera en uno de los parámetros de alguno de los pares, se considera una nueva posibilidad.

Al detectar una nueva posibilidad, se almacena y se lanza como posibilidad inicial para que busque nuevos caminos aplicando sus resoluciones.

El cálculo finaliza cuando por ninguno de los caminos abiertos se encuentre posibilidades nuevas. En ese momento se dispone de todas las posibilidades que representan los posibles estados, que se pueden generar partiendo de la CA y aplicándole las resoluciones obtenidas.

3.5.1.2 CALCULO DE LA MATRIZ DE NAVEGACIÓN

El cálculo de la matriz de navegación se tiene que realizar en paralelo con el cálculo de las posibilidades anteriormente explicado. De no hacerlo así se tendría un conjunto de posibilidades, pero sin las relaciones entre ellas. Es oportuno recordar que lo que se pretende obtener es un diagrama donde se representen unas posibilidades y las relaciones entre ellas.

La matriz de navegación es una matriz que recoge las transiciones entre las posibilidades como representa la figura 31.

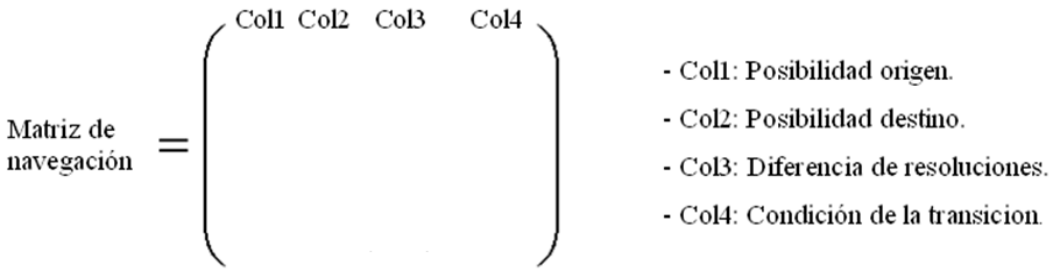


Figura 31 - Matriz de navegación incompleta

Cada transición está compuesta por:

> Transición = {Posibilidad origen, posibilidad destino,
> resolución de transición, condición }

- **Posibilidad origen**

  La posibilidad origen es la posibilidad utilizada para iniciar el proceso.

- **Posibilidad destino**

  La posibilidad destino es la nueva posibilidad, es decir, la posibilidad resultante de aplicar las resoluciones sobre la posibilidad origen.

- **Resolución de transición**.

  La resolución de transición son aquellos cambios aplicadas a la posibilidad origen para crear la nueva posibilidad destino.

- **Condición de transición**

  La condición de transición es el evento que tiene que cumplirse para que pueda ejecutarse la transición.

En la primera columna de la matriz se inserta la posibilidad de origen. En la segunda columna la posibilidad destino alcanzada al aplicar una determinada resolución, que quedará representada en la tercera columna. Por último, la cuarta columna define la condición asociada a la transición.

El número de filas de la matriz viene dado por el número de transiciones que se llevan a cabo.

Este proceso se debe realizar a la vez que el cálculo de posibilidades, porque en ese momento es muy sencillo almacenar las relaciones entre las posibilidades. Así que cada vez que se obtiene una nueva posibilidad después de comprobar que no existe se procede a crear una nueva transición.

3.5.1.3 EJEMPLO DE ESPACIO DE POSIBILIDADES:

Este último subapartado presenta un ejemplo para poner en práctica el proceso explicado en los dos apartados anteriores. Para ello se parte del diagrama de la figura siguiente.

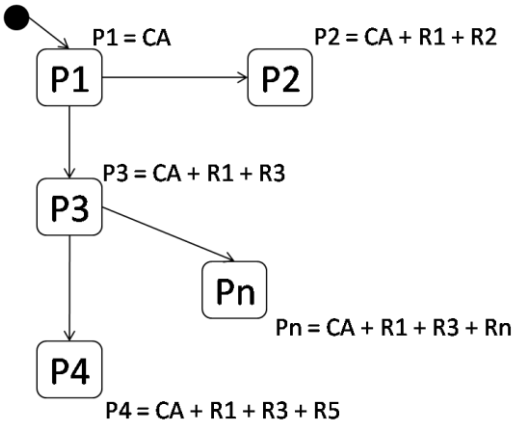


Figura 32 - Ejemplo espacio de posibilidades

Como se puede observar en la figura 32 el diagrama tiene cinco posibilidades (P1, P2, .., Pn) y cuatro transiciones entre ellas (P1-P2, P1-P3, P3-P4 y P3-Pn).

El círculo negro indica la posibilidad de partida. Esta primera posibilidad tiene una configuración inicial asignada que queda referenciada como configuración actual (CA).

La segunda posibilidad, P2, tiene como origen la posibilidad P1 y se le han aplicado las resoluciones R1 y R2. Siguiendo el cálculo por ese camino no se han encontrado nuevas posibilidades.

La tercera posibilidad, P3, surge como un nuevo camino que también tiene como origen P1. Aplicando las resoluciones R1 y R3 generan la nueva configuración.

Siguiendo este camino, con la posibilidad P3 como origen y aplicando la resolución R5 se genera la posibilidad P4. A partir de ella no se obtiene ninguna nueva posibilidad.

Se vuelve a la posibilidad P3 y se aplica la resolución Rn para conseguir la nueva posibilidad Pn.

A continuación y siguiendo con el ejemplo anterior se presenta la matriz de navegación asociada a este espacio de posibilidades.

$$\text{Matriz de navegación} = \begin{pmatrix} \text{Col1} & \text{Col2} & \text{Col3} & \text{Col4} \\ \text{P1} & \text{P2} & (+\text{R1}, \text{R2}) & \text{C1} \\ \text{P1} & \text{P3} & (+\text{R1}, \text{R3}) & \text{C2} \\ \text{P3} & \text{P4} & (+\text{R5}) & \text{C3} \\ \text{P3} & \text{Pn} & (+\text{Rn}) & \text{C4} \end{pmatrix}$$

- Col1: Posibilidad origen.
- Col2: Posibilidad destino.
- Col3: Diferencia de resoluciones.
- Col4: Condición de la transicion.

Figura 33 - Matriz de navegación completada

Como se observaba en la figura 33, el espacio de posibilidades son cuatro transiciones entre posibilidades. Esas serán el número de entradas que tendrá la matriz de navegación.

En la primera línea de la matriz de navegación se representa la transición entre la posibilidad P1 y la posibilidad P2. Donde las resoluciones de la transición son R1 y R2, y la condición de transición es la condición C1.

La siguiente transición es la que va de la posibilidad P1 a la P3, y las resoluciones añadidas son la R1 y la R3. Su condición de transición es la C2.

La transición que va de P3 a P4 viene dada por la resolución R5 y por condición la C3.

La última transición es la que enlaza la posibilidad P3 con la Pn a la que se ha añadido la resolución Rn y su condición de transición es la C4.

3.6 VALIDACIÓN DE LAS POSIBILIDADES

Para conseguir sistemas realmente fiables es indispensable realizar algún tipo de validación antes de aplicar las configuraciones. Es por ello, que a partir de la representación en términos de espacio de posibilidades se realiza un análisis que establece la validez o no de cada una de las configuraciones asociadas a las posibilidades.

Para conseguirlo es necesario validar cada posibilidad individualmente. Esto se consigue comprobando la configuración asociada a cada posibilidad en términos de las restricciones del modelo de características.

Esta comprobación se puede llevar a cabo utilizando un validador de modelos que realiza un análisis de variabilidad. En particular, se utiliza el framework FAMA [58]. FAMA es un framework para el análisis automático de modelos de características que integra algunas de las representaciones lógicas más utilizadas. Permite determinar si una determinada configuración del sistema es válida de acuerdo a unas restricciones y también puede proporcionar información si la configuración no lo es.

En la figura 34 se puede ver el espacio de posibilidades de la figura 32 pero con las posibilidades coloreadas en función de su validez. Las posibilidades cuya configuración ha resultado válida están pintadas de color verde mientras que las inválidas están de color rojo. Una posibilidad se marca como inválida cuando incumple las propiedades que se han definido en su modelo de características.

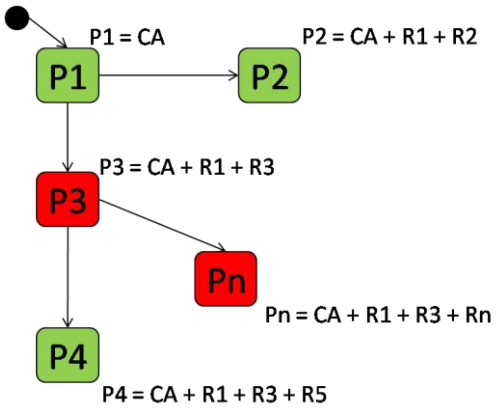


Figura 34 - Ejemplo espacio de posibilidades validado

Este sistema de colores es muy útil ya que a simple vista se puede observar dentro del espacio de posibilidades el volumen de estados validos e inválidos.

3.7 DEL EP AL EP ABSTRACTO

Este apartado pretende explicar en qué consiste el proceso de abstracción que realizará la transformación del espacio de posibilidades al espacio de posibilidades abstracto y cuáles son las ventajas de esta representación. Como se ha comentado anteriormente, el objetivo del espacio de posibilidades abstracto es ofrecer una versión lo más reducida posible pero sin perder información acerca de las propiedades más importantes.

La representación del modelo en un espacio de posibilidades puede generar un enorme número de posibilidades que resultarán difícilmente legibles y entendibles para una persona. Este proceso de abstracción pretende agrupar las posibilidades que comparten determinadas condiciones o propiedades. De este modo, la posterior representación generará un espacio de posibilidades más reducido y que sea fácilmente manejable.

La clave para realizar esta transformación es agrupar todas las posibilidades que tengan el mismo estado y que estén conectadas mediante transiciones. A continuación se detalla cómo se realiza dicha tarea.

Empezando por la posibilidad inicial, se recorren todas las posibilidades accesibles mediante transiciones hasta encontrar una posibilidad con estado diferente a la inicial.

Todas las posibilidades recorridas hasta el momento se convertirían en la nueva posibilidad inicial la cual tendrá el mismo estado que todas las posibilidades que lo integran. Esta nueva posibilidad se etiqueta con un valor que indica el número de posibilidades que ha sido posible agrupar.

A continuación, se sigue por la transición que nos lleva a la posibilidad que tenía un estado diferente y se recorren todas las posibilidades del mismo estado, para así poder agruparlos como se hizo con la anterior. Y así sucesivamente hasta completar todas las posibilidades.

La figura 35 presenta un ejemplo:

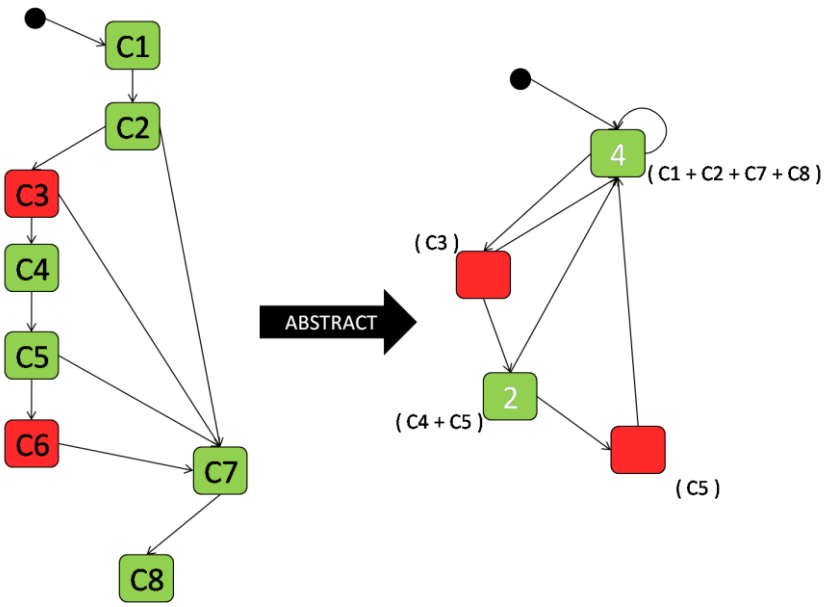


Figura 35 - Ejemplo resumido proceso abstracción

En la figura 35 el diagrama de la izquierda se corresponde con el espacio de posibilidades generado originalmente. Este espacio de posibilidades cuenta con 8 posibilidades (C1, C2, …, C8). Las posibilidades C3 y C6 han resultado inválidas tras el análisis mientras que el resto, coloreadas de verde, cumplen las restricciones.

Tras el proceso de abstracción el espacio resultante es el diagrama de la derecha. Como queda reflejado en la figura 35, las posibilidades C1, C2, C7 y C8 han quedado agrupadas en una única posibilidad. De forma similar las posibilidades C4 y C5 también se han compactado. El resto de posibilidades no cumple los requisitos necesarios para su abstracción. Como se ha comentado, las posibilidades abstractas quedan etiquetadas con el número de posibilidades que se han agrupado.

Para comprender como se ha llevado a cabo esta transformación se acompaña el paso a paso sobre el espacio de posibilidades original de la figura 35.

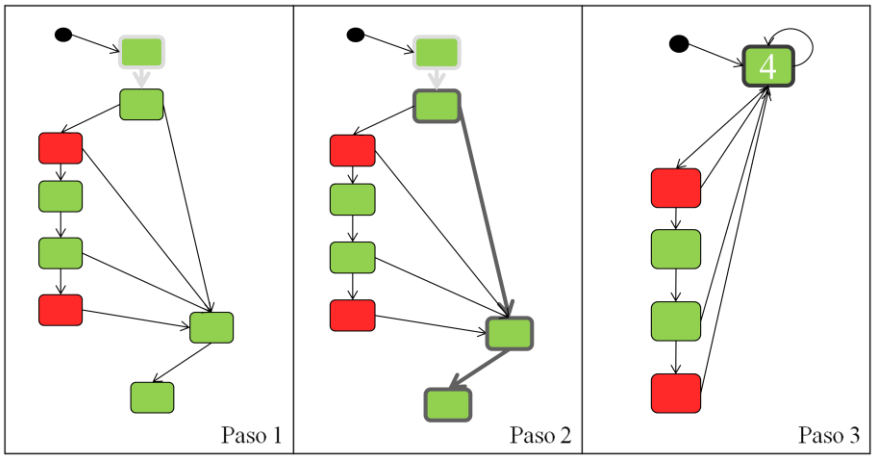


Figura 36 - Proceso de abstracción: paso 1-3

La figura 36 presenta los tres primeros pasos del proceso. El primer paso consiste en partiendo de la posibilidad inicial analizar una a una sus posibilidades vecinas. El objetivo es consultar su estado con el objetivo de agrupar el máximo número de posibilidades.

Para cada una de estas posibilidades se analiza a su vez a sus propias vecinas. De tal modo que, como se aprecia en el paso 2, se toma un camino aleatorio de los posibles y a partir de éste se recorren a su vez los posibles caminos que ofrece cada posibilidad. Cuando por una posibilidad ya se han alcanzado todos sus caminos se vuelve a la posibilidad anterior y se recorren los de ésta. Este proceso se repite mientras el estado de todas las posibilidades sea el mismo y no hayan sido previamente comprobadas.

En este caso, dado que la posibilidad de partida es válida se siguen recorriendo mientras todas las encontradas sean válidas. En este punto todavía no existen posibilidades que ya hayan sido visitadas. El proceso se detiene cuando se vuelve a la posibilidad inicial y el siguiente camino conduce a una posibilidad inválida.

Llegado a este punto se agrupan, como se puede ver en el paso tres, en una única posibilidad etiquetada con el número de posibilidades que engloba. En este caso, se han alcanzado cuatro posibilidades con el mismo estado.

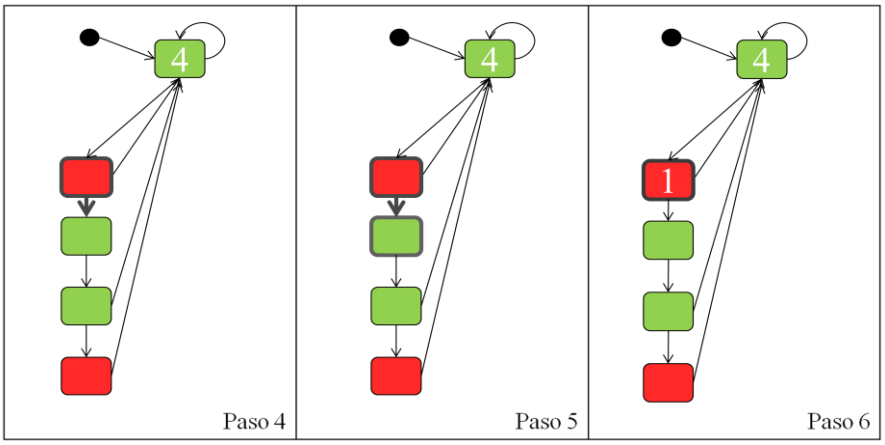


Figura 37 - Proceso de abstracción: paso 4-6

Como indica la figura 37, a partir de la posibilidad que se ha detectado con un estado diferente se recorren sus caminos. Como se aprecia en el paso 4, en el espacio existen dos transiciones posibles. De las dos transiciones posibles solo una de ellas conduce a una posibilidad no visitada.

En el paso 5 se toma esta transición. En este caso, dado que la posibilidad de partida era inválida y la alcanzada es válida no se sigue recorriendo el espacio por esta rama. Como no quedan más caminos posibles la posibilidad de partida se convierte en abstracta y queda marcada con un uno ya que no se ha podido agrupar con ninguna otra.

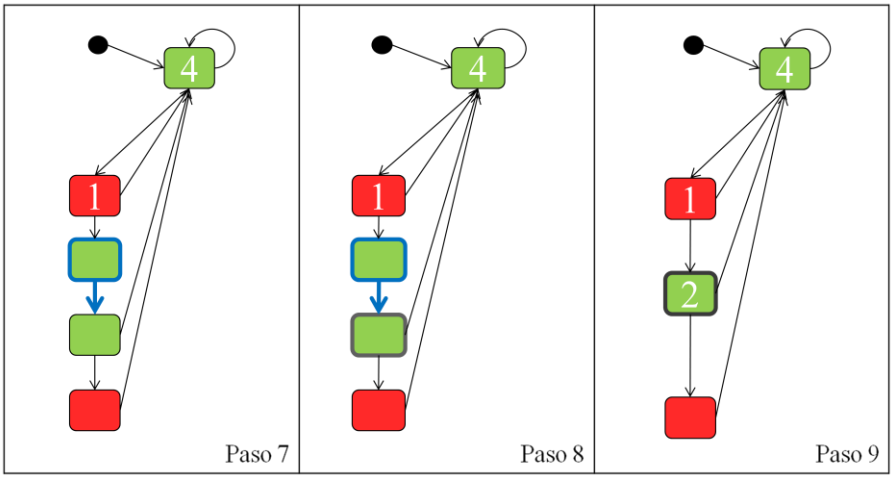


Figura 38 - Proceso de abstracción: paso 7-9

El proceso continúa, figura 38, a partir de la posibilidad detectada con estado diferente. Esta posibilidad únicamente dispone de un camino, como se puede ver en el paso 7. A su vez se toman los caminos de la posibilidad alcanzada. Aunque ahora existen dos caminos, obligatoriamente se consulta el que conduce a una posibilidad inválida, ya que el otro conduce a una posibilidad que ya ha sido previamente consultada. Esta posibilidad es la inicial.

Esta transición al alcanzar una posibilidad con un estado diferente hace que se detenga el proceso. En este punto se agrupan las posibilidades convirtiéndose en una única posibilidad abstracta como se indica en el paso 9.

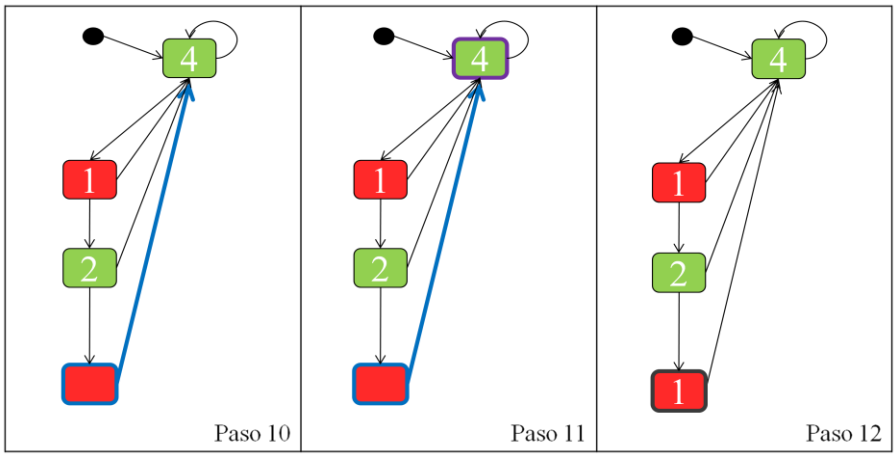


Figura 39 - Proceso de abstracción: paso 10-12

De nuevo el proceso, figura 39, continúa a partir de la posibilidad con estado diferente, paso 10. Al intentar consultar las posibilidades alcanzables a través de sus transiciones se sigue por la única transición de esta posibilidad.

La única posibilidad alcanzable ya ha sido previamente visitada y además su estado es diferente al actual. Esto supone que la posibilidad actual se convierta en una abstracta etiquetada con un uno ya que no ha podido agrupar a ninguna otra.

El paso 12 es la representación del espacio de posibilidades resultante después del proceso de abstracción, donde ya han sido recorridas todas las ramas. De nuevo en la figura 40 se puede ver como este proceso ha dejado el espacio más simplificado mediante la eliminación de información que en este punto se consideraba irrelevante.

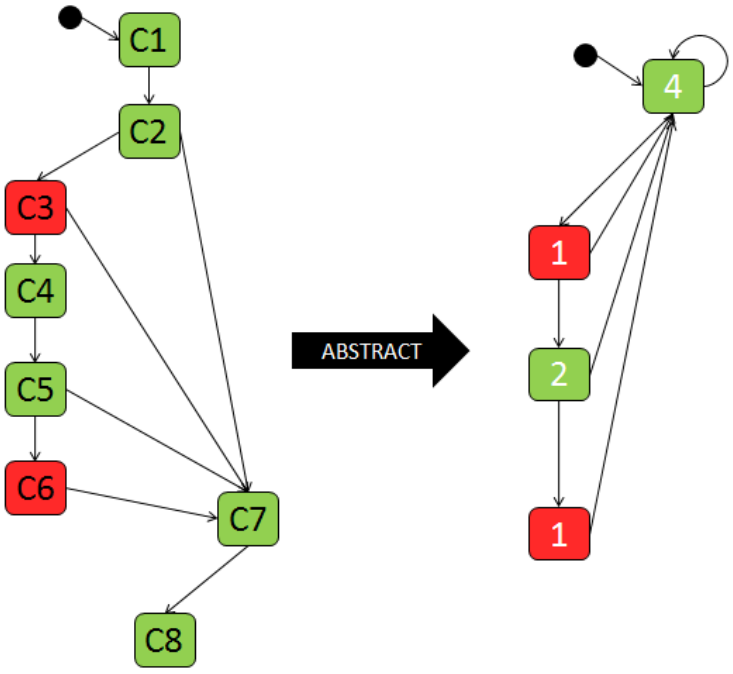


Figura 40 - Comparativa EP vs EP abstracto

Aunque en el ejemplo estudiado el número de posibilidades es pequeño y no genera problemas de interpretación su simplificación siempre es útil. El proceso de abstracción tiene mucho más sentido cuando tratamos con espacios de posibilidades grandes.

Por ejemplo, como refleja la figura 41, para un modelo de características muy simple de 8 características con dos posibles estados, el número máximo de posibilidades es enorme.

Model Name:	\src\SmartHome.MFI	Number of possibilities:	8
Numbers of features:	18	Number of valid:	6
Max nº of possibilitys:	4.166090266796	Number of invalid:	2

Figura 41 - Número de posibilidades

A pesar de que luego no se generan todos, aun estamos hablando de un orden de magnitud alto. Esto supone un tamaño que no se puede manejar a la hora de trabajar sobre el espacio de posibilidades.

El espacio de posibilidades abstracto solo mantiene la información que resulta relevante. Esto facilita en gran medida el trabajo de cualquier persona que deba tratar con este tipo de representaciones. Permitiendo que el trabajo sea mucho más sencillo y rápido que si se trabajase sobre los espacios de posibilidades originales.

4. REFACTORING DE LA RECONFIGURACIÓN

En este capítulo se presenta el principal objetivo del desarrollo de esta propuesta. Se explica el proceso de refactoring aplicado sobre un modelo de características. Su finalidad es proporcionar un modelo de características libre de errores.

Este apartado queda organizado como sigue. En primer lugar, se introducen las particulares generales del proceso de refactoring y todo lo necesario para entender la notación que se utilizará en los puntos siguientes.

En segundo lugar, se explica a nivel conceptual y de forma simplificada el primer refactoring para comprender en qué consiste. Este primer refactoring se denomina Remove Unsafe Reconfigurations y persigue eliminar las reconfiguraciones inseguras.

En tercer lugar, siguiendo la metodología del punto anterior, se detalla el segundo refactoring. Este refactoring se denomina Unsafely Reachable Possibilities y pretende alcanzar las posibilidades válidas aisladas tras el primer refactoring.

Finalmente, se presenta un ejemplo completo donde se aplican consecutivamente los dos procesos de refactoring. Sobre este ejemplo se detallan paso a paso las operaciones ejecutadas.

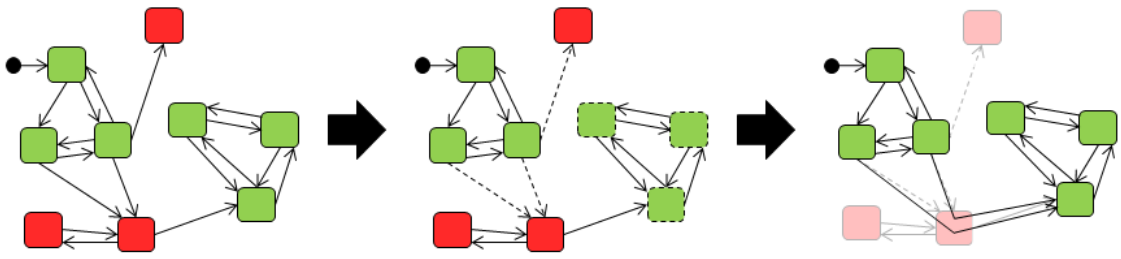


Figura 42 - Objetivo del capítulo 4

La figura 42 refleja el objetivo que persigue este capítulo. Desde el espacio de posibilidades obtenido tras el análisis, la posterior identificación de los errores y finalmente el espacio de posibilidades libre de errores.

4.1 CONCEPTOS BÁSICOS

Como se ha comentado en capítulos anteriores, un refactoring es un proceso que permite redefinir el comportamiento de determinados componentes sin influir en la funcionalidad. Los refactorings aquí explicados se llevan a cabo para evitar o resolver situaciones anómalas.

En este caso, el propósito del proceso de refactoring no es otro que obtener un modelo de características final libre de errores. Para ello, es necesario estudiar los errores que se pueden dar e implementar los mecanismos necesarios para impedir que se produzcan.

A partir del proceso de validación de las resoluciones se obtiene un modelo de características donde se marcan aquellas posibilidades que no cumplen las restricciones. Este modelo será la base para identificar el tipo de errores producido y en función de ello desarrollar una metodología que elimine cada uno de estos tipos de errores.

A continuación se muestra un modelo de características tras el proceso de validación. Como se aprecia, las posibilidades han sido coloreadas en base al resultado del análisis.

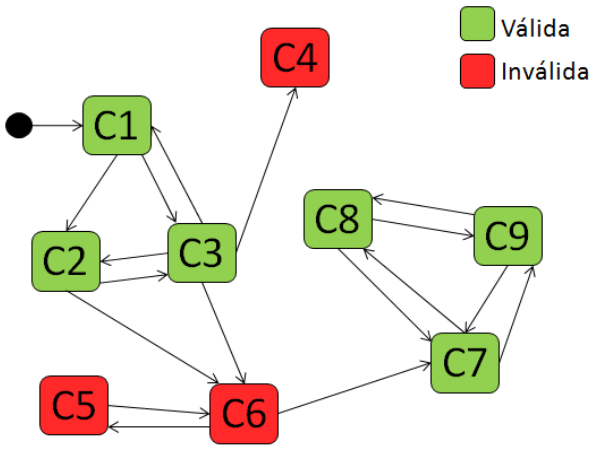


Figura 43 - Ejemplo espacio de posibilidades

El espacio de posibilidades de la figura 43 está compuesto por 9 posibilidades (C1, C2,…C9) con sus correspondientes configuraciones. Las posibilidades coloreadas de rojo (C4, C5 y C6) representan posibilidades inválidas. Este tipo de posibilidades no cumplen las restricciones del sistema por lo tanto deberían quedar fuera del modelo.

A partir de estas posibilidades inválidas se detectan dos situaciones no deseadas:

En primer lugar, las posibilidades inválidas no deben ser alcanzadas por ninguna reconfiguración. Las reconfiguraciones que disparan transiciones a alguna posibilidad inválida se denominan: reconfiguraciones inseguras y constituyen el primer refactoring. Se representan con una línea discontinua.

– – – –➔ Reconfiguración insegura

Las posibilidades C4, C5 y C6 coloreadas de rojo indican que constituyen configuraciones del sistema que no son válidas. Por tanto, las transiciones que disparan resoluciones hacia éstas deben ser marcadas como inseguras. Este es el caso de las transiciones entre C2-C6, C3-C6 y C3-C4.

Aplicado sobre el espacio de posibilidades tomado como ejemplo:

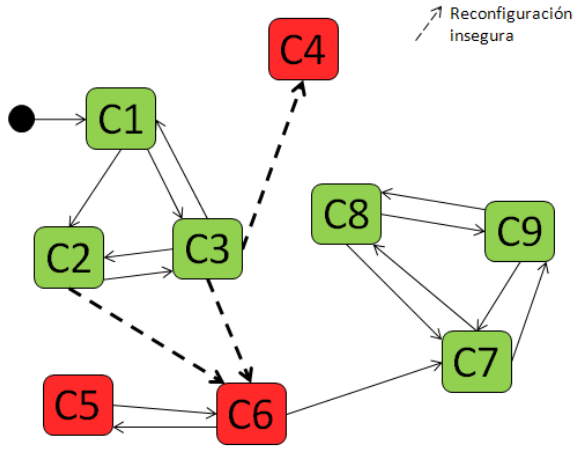


Figura 44 - Ejemplo espacio de posibilidades tras identificar el primer tipo de error

En segundo lugar, las posibilidades inválidas no deben lanzar transiciones a otras posibilidades del modelo. Las posibilidades alcanzadas a través de posibilidades inválidas se denominan: posibilidades inseguras alcanzables. Estas posibilidades, que no dejan de ser válidas, vienen representadas también por una línea discontinua.


Posibilidad válida alcanzable (de forma insegura)

Este tipo de posibilidades pueden verse como posibilidades alcanzables a través de reconfiguraciones inseguras o como posibilidades válidas inalcanzables después de eliminar las reconfiguraciones problemáticas. Así que no es de extrañar que en este apartado se refiera a ellas de forma distinta.

La única posibilidad inválida que dispara transiciones hacia posibilidades válidas del sistema es C6. Desde ella se alcanzan las configuraciones asociadas a las posibilidades C7, C8 y C9, esto supone que deben marcarse como posibilidades inseguras.

En el espacio de posibilidades tomado como ejemplo:

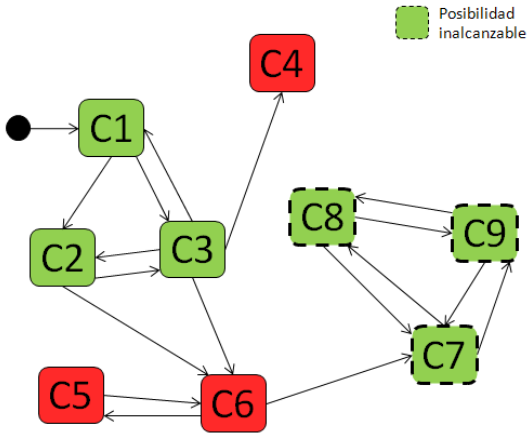


Figura 45 - Ejemplo espacio de posibilidades tras identifica el segundo tipo de error

Los dos procesos de refactoring deben aplicarse de forma sucesiva. El primer refactoring debe marcar las resoluciones inseguras y el segundo las posibilidades alcanzadas a través de posibilidades inválidas.

Aplicando los dos refactorings sobre el ejemplo el espacio de posibilidades queda representado como se indica en la figura 46:

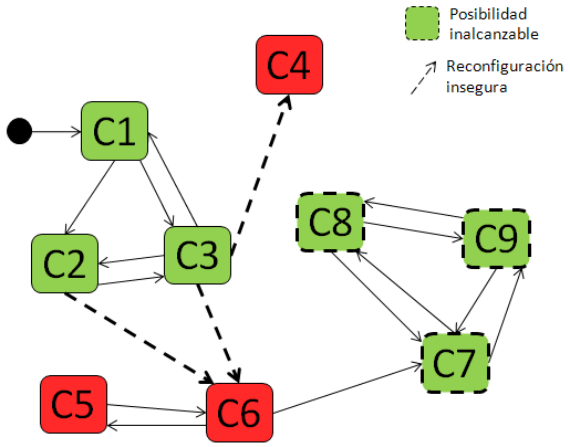


Figura 46 - Ejemplo espacio de posibilidades tras identificar errores

El objetivo de los refactorings no es solo identificar y marcar los errores sino también aplicar algún mecanismo que los resuelva.

Con la intención de proporcionar más información sobre ambos procesos en el próximo apartado se estudian por separado con un mayor nivel de detalle.

4.2 Primer Refactoring: Remove Unsafe Reconfigurations

Este primer refactoring consiste en la eliminación de las reconfiguraciones que disparan transiciones hacia posibilidades inválidas. Como se ha comentado estas reconfiguraciones se representarán mediante una línea discontinua en el diagrama.

Para comprender mejor el proceso tomamos un caso particular del ejemplo de la figura 46.

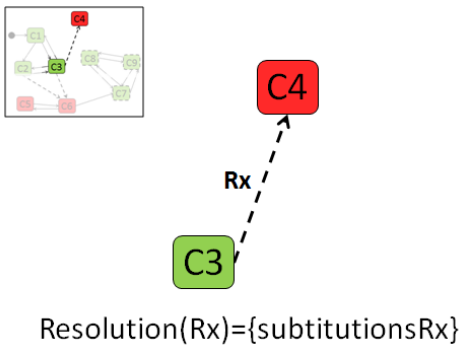


Figura 47 - Fragmento espacio de posibilidades

La figura 47 es un fragmento del ejemplo original, se centra en las posibilidades C3 y C4 del modelo y la resolución R_x que dispara la transición entre ellas.

La posibilidad C4, coloreada de rojo, indica que es una posibilidad que no cumple las restricciones del modelo y por lo tanto se ha marcado como inválida. Esto supone que para asegurar la fiabilidad del sistema no deberían llegar transiciones hacia dicha configuración. Las resoluciones que le alcanzan se marcan como inseguras, en este caso, R_x se representa de forma discontinua.

El objetivo ahora es informar a la posibilidad origen que no debe lanzar las resoluciones que generan transiciones hacia posibilidades inválidas. Una forma de hacer esto, consiste en aplicar una condición sobre la resolución afectada que impida que sea disparada cuando se encuentre en la posibilidad cuya transición sea inválida.

Para el caso presentado, consiste en aplicar una guarda sobre R_x que impida que sea lanzada cuando se encuentre en la posibilidad C3. De este modo, desaparece la transición que generaba esa posibilidad insegura como refleja la figura 48.

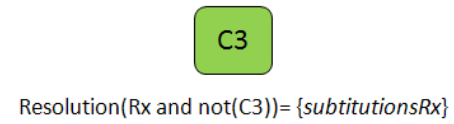


Figura 48 - Resultado primer refactoring

4.3 Segundo Refactoring: Unsafely Reachable Possibilities

Este segundo refactoring consiste en la creación de nuevas resoluciones que permitan que las posibilidades alcanzadas a través de posibilidades inválidas continúen siendo accesibles. Como se ha comentado estas posibilidades se representarán mediante una línea discontinua en el diagrama.

Las nuevas resoluciones deben contener como estado final de sus características el resultado de aplicar todas las resoluciones intermedias que se disparan entre posibilidades. Además debe contener una condición para que solo sea aplicable en la posibilidad que ha sido creada.

Las resoluciones intermedias que alcanzaban posibilidades inválidas seguirán el procedimiento explicado en el primer refactoring, anotándose la condición que le corresponda.

De forma análoga al primer refactoring se extrae un caso particular del ejemplo presentado en la figura 46 para facilitar la comprensión de este proceso.

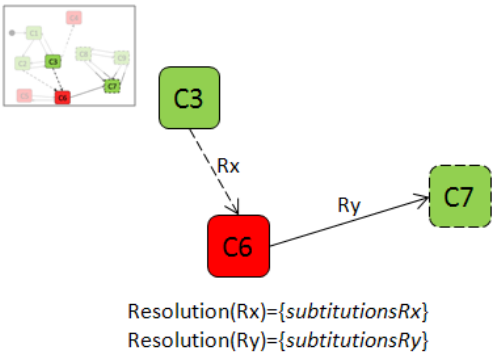


Figura 49 - Fragmento espacio de posibilidades

El fragmento mostrado en la figura 49 se centra en las posibilidades C3, C6 y C7 del modelo, enlazadas por las resoluciones R_x y R_y. La posibilidad C6 es inválida, por lo tanto la resolución que la alcanza, R_x, se marca como insegura. C6 a su vez lanza una resolución, R_y, que alcanza a la posibilidad C7.

La posibilidad C7 se representa con líneas discontinuas ya que ha sido alcanzada a través de una posibilidad inválida.

Si solo aplicamos el primer refactoring la posibilidad C7, que es válida, queda aislada del modelo y no es alcanzable. Consecuentemente tampoco lo sería el resto de posibilidades que ésta dispara.

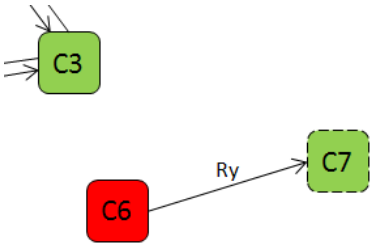


Figura 50 - Resultado parcial

La figura 50 ilustra como se ha eliminado la reconfiguración entre C3-C6. La posibilidad C7 y las generadas a partir de ella, ver figura 46, quedan separadas del resto del modelo.

El objetivo es crear una nueva resolución que disparada desde la posibilidad que lanza la resolución insegura lleve a una posibilidad que previamente era alcanzada por una posibilidad inválida. A continuación se explica más detalladamente sobre el ejemplo.

Para el caso ejemplo presentado, consistiría en crear una nueva resolución que desde C3 alcance la posibilidad C7. De este modo el resto de posibilidades continúan siendo alcanzables.

La nueva resolución R_x^R_y debe aglutinar las características de cada resolución y aplicar una condición en forma de guarda que solo permita su ejecución para la posibilidad C3. El estado de las características debe corresponderse con el proceso de lanzar una a una y de forma ordenada las resoluciones intermedias. En este caso, se aplica la resolución R_x sobre C3 y posteriormente la resolución R_y, la configuración resultante quedará plasmada en la posibilidad C7. La ejecución de esta nueva resolución queda limitada a la posibilidad C3, por ello, se guardará una condición que impida su lanzamiento en el resto de situaciones.

La figura 51 refleja como quedan enlazadas las posibilidades C3 y C7 tras crear la nueva resolución R_x^R_y. De este modo, C7 vuelve a ser accesible y consecuentemente todas las resoluciones que ésta lanza.

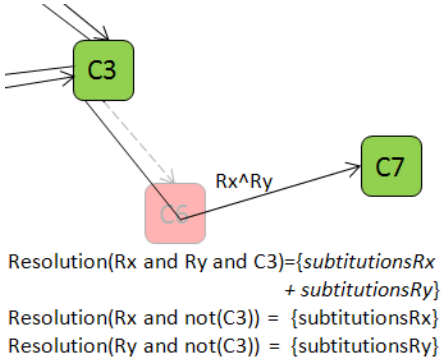


Figura 51 - Resultado del segundo refactoring

Al finalizar este proceso, las posibilidades contarán con unas condiciones que limitarán la ejecución de algunas de sus resoluciones. Según el primer refactoring, la ejecución de la resolución R_x estará inhabilitada para la posibilidad C3. Según el segundo refactoring, la resolución R_x^R_y almacenará una condición que impida su disparo en cualquier posibilidad diferente de C3.
Para el caso de ejemplo tomado al inicio de este apartado, el resultado de aplicar ambos procesos de refactoring es el indicado por la figura 52.

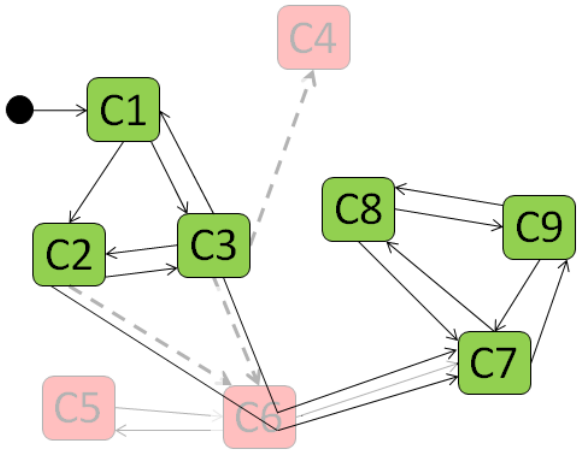


Figura 52 - Resultado deseado tras refactorings

Con la intención de presentar todo el proceso que atraviesa un espacio de posibilidades desde la identificación de los errores hasta conseguir un espacio totalmente libre de ellos se expone un ejemplo completo en el siguiente apartado.

4.4 EJEMPLO COMPLETO

En este apartado se presenta un ejemplo completo de ambos procesos de refactoring.

En primer lugar se efectúa el primer refactoring y se explica detalladamente cada uno de los pasos que se han ido siguiendo.

En segundo lugar y a partir del resultado del primer refactoring se procede al segundo refactoring siguiendo la misma metodología.

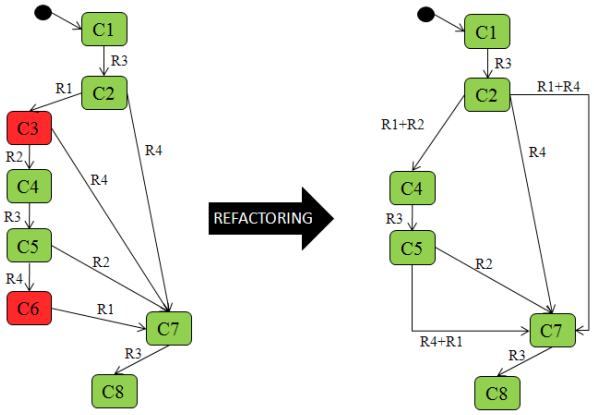
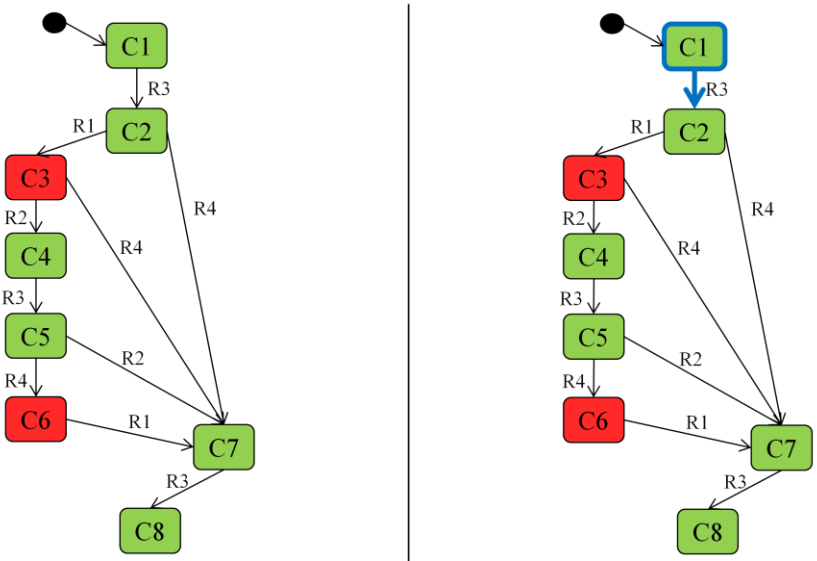


Figura 53 - Ejemplo resumido del proceso de refactoring

La figura 53 presenta de forma resumida el espacio de posibilidades original y el espacio de posibilidades libre de errores obtenido tras el proceso de refactoring.

A continuación se detalla el paso a paso del proceso:



RESOLUTION	CONDITION	GUARD	FEATURES	RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1		{(F1, T), (F2, F), (F3, T), (F4, F)}	R1	Condition1		{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}	R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}	R3	Condition3		{(F5, F)}
R4	Condition4		{(F4, F), (F5, T)}	R4	Condition4		{(F4, F), (F5, T)}

Figura 54 - Refactoring 1: paso 1-2

La figura 54 detalla los paso 1 y 2 del refactoring. El espacio de posibilidades representando a la izquierda se corresponde con el espacio antes de iniciar el proceso de refactoring. Está formado por 8 posibilidades (C1,…, C8) y 4 resoluciones (R1,…, R4).

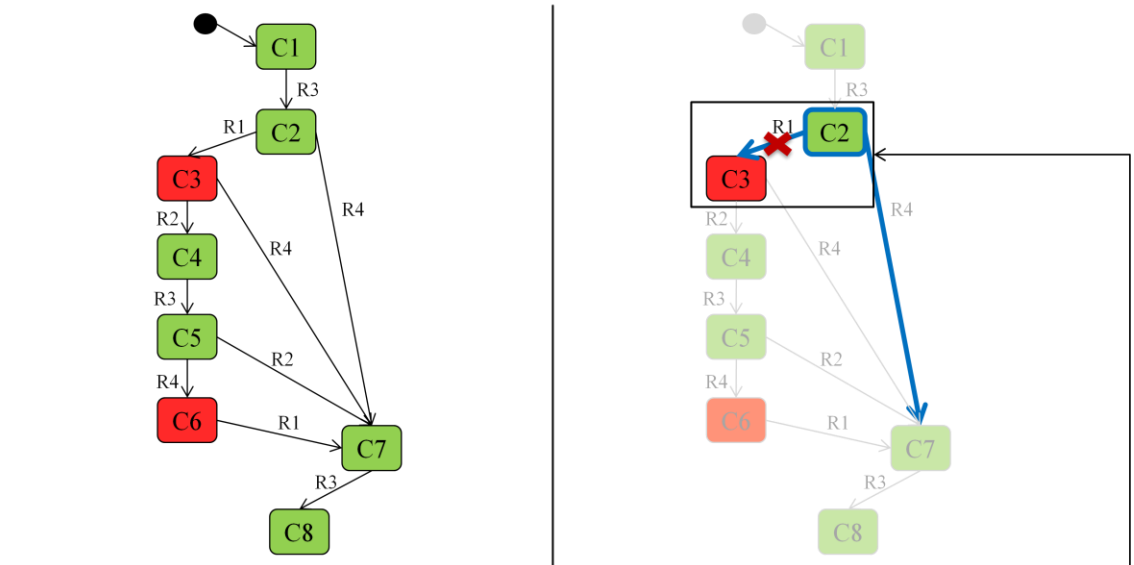
Como ya se ha comentado en apartados anteriores tras el proceso de validación las posibilidades quedan señaladas según la validez de su estado. Las posibilidades coloreadas de verde indican que son válidas, mientras que las posibilidades rojas se consideran inválidas. En este caso (C3 y C6) se consideran inválidas y el resto válidas.

En la tabla adjunta a cada espacio se detallan los atributos propios de cada una de las cuatro resoluciones con las que se cuenta. Cada resolución está asociada con sus condiciones de disparo y con la lista de características sobre las que actúa.

En la tabla además se puede ver una columna *Guard* que inicialmente aparece vacía. El propósito de esta columna es almacenar para cada resolución las posibilidades origen donde no podrán ser lanzadas. De este modo, cuando una resolución conduzca a una posibilidad destino inválida, la posibilidad origen se marcará como false y no será ejecutada.

Conocidos los atributos que facilitarán el cálculo, el objetivo a partir de ahora es recorrer para cada posibilidad origen sus posibles destinos y determinar qué resoluciones conducen a posibilidad inválidas. El recorrido viene dado por el orden en el que han sido creadas, es decir, se empezará por C1 y terminará el proceso con C8.

La posibilidad de partida en este proceso es la que está marcada como inicial. A partir de la posibilidad C1 se estudian sus transiciones. Desde C1 únicamente existe una transición, que conduce a C2. Como C2 también es válida, hasta el momento, no se han detectado errores.



RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1		{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4		{(F4, F), (F5, T)}

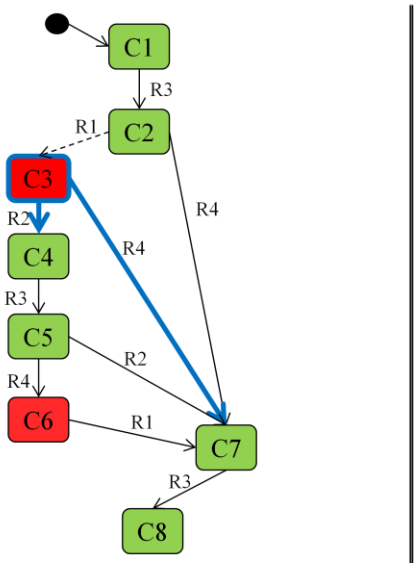
RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	**C2=False**	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4		{(F4, F), (F5, T)}

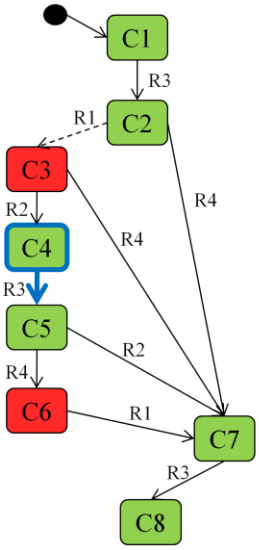
Figura 55 - Refactoring 1: paso 3-4

La figura 55 detalla los paso 3 y 4 del refactoring. A partir de C2 se estudian las transiciones de las que dispone. Esta posibilidad conduce a C3 mediante la ejecución de la resolución R1 y a C7 mediante R4. La posibilidad C3 es inválida por lo que la resolución que conduce a ella, R1, se marcará como inválida.

Marcar una resolución como inválida para una posibilidad supone que para dicha posibilidad no será ejecutada. Como se aprecia en la representación del espacio de la derecha, para la resolución R1 se ha añadido una guarda. En este caso para la posibilidad C2 no se ejecutará la resolución R1. Se añade C2=False en el campo de la guarda de R1.

Para identificar durante el resto del proceso las resoluciones inválidas que ya han sido estudiadas se marcan en la representación del espacio con una línea discontinua.





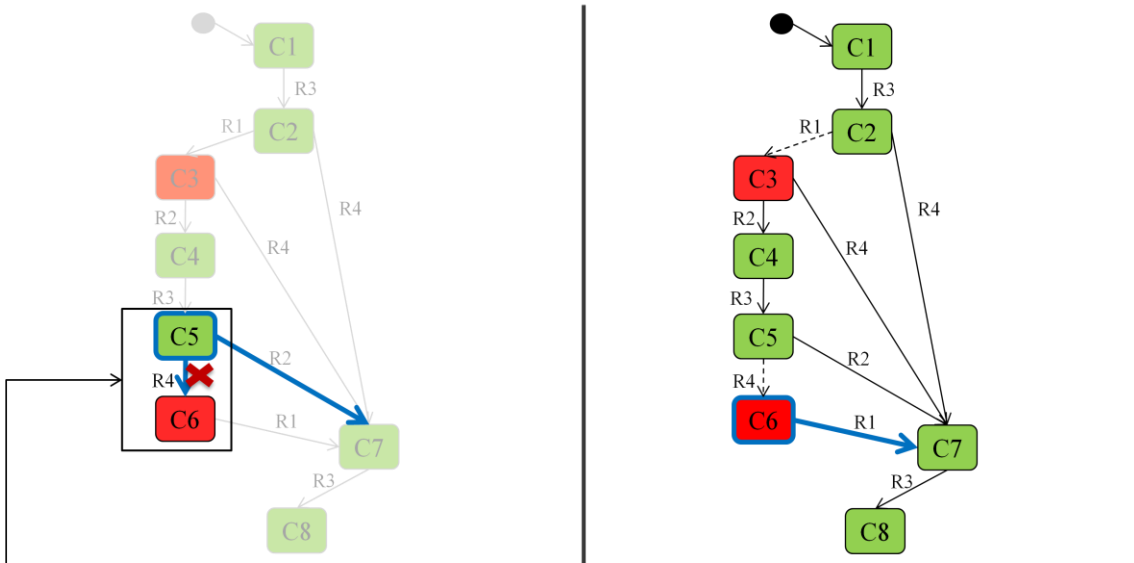
RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4		{(F4, F), (F5, T)}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4		{(F4, F), (F5, T)}

Figura 56 - Refactoring 1: paso 5-6

La figura 56 detalla los paso 5 y 6 del refactoring. La siguiente posibilidad a analizar es C3. A partir de ella se puede alcanzar C4 mediante la ejecución de R2 y C7 si se ejecuta R4. Como se aprecia en el espacio de la izquierda ambas posibilidades destino son válidas. Por lo tanto, ninguna de estas resoluciones incumple ninguna restricción y se siguen manteniendo como válidas.

A continuación se analiza la posibilidad C4. Solo dispone de una transición, ejecutando la resolución R3 conduce a la posibilidad C5. Tal como ocurría para C3 los destinos, en este caso solo uno, conduce a posibilidades válidas. Siendo así no se realiza ninguna operación.



RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	**C5=False**	{(F4, F), (F5, T)}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}

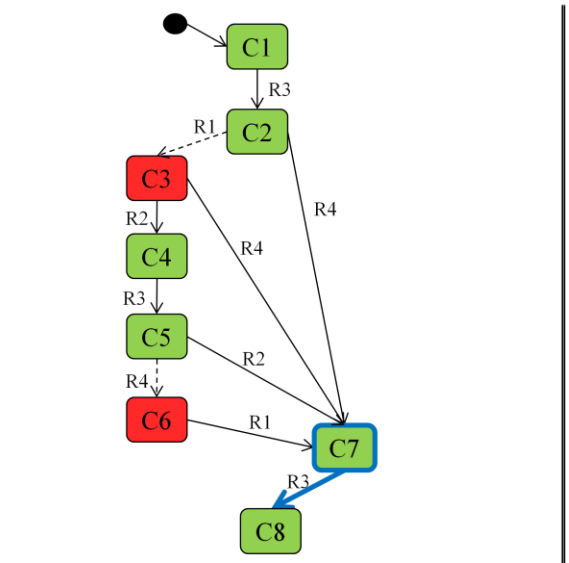
Figura 57 - Refactoring 1: paso 7-8

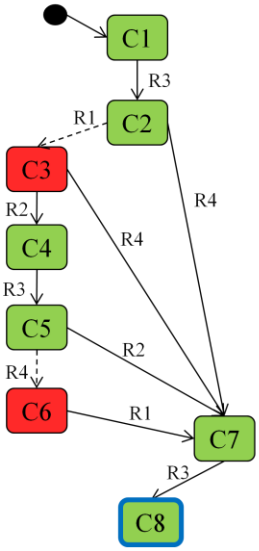
La figura 57 detalla los paso 7 y 8 del refactoring. La siguiente posibilidad bajo estudio es C5. Desde ella se puede alcanzar C6 ejecutando la resolución R4 y la posibilidad C7 mediante R2.

Como se aprecia, la posibilidad destino C6 es inválida. Esto supone que la resolución que la alcanza tendrá que marcarse como inválida. Del mismo modo que se ha procedido antes al detectar una resolución inválida, la resolución se representa con una línea discontinua y se añade la guarda. A la resolución R4 se le añade la guarda C5=false que impedirá que sea ejecutada en esta posibilidad.

En el espacio de la derecha, la resolución que se acaba de detectar ya ha sido identificada mediante una línea discontinua.

El paso siguiente es analizar los destinos alcanzables desde C6. Solo existe una transición que ejecuta R1 y conduce a la posibilidad C7. Dado que es una posibilidad válida el proceso puede continuar.





RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}

Figura 58 – Refactoring 1: paso 9-10

La figura 58 detalla los paso 9 y 10 del refactoring. A continuación se toma la posibilidad C7. Solo cuenta con una transición, ejecutando la resolución R3 alcanza C8. Dado que la posibilidad C8 es válida no se toma ningún tipo de medida.

Por último C8 es la posibilidad final, por lo que no actúa como posibilidad origen y no dispara ninguna resolución.

La representación del espacio tras aplicar el primer refactoring queda tal como el espacio de posibilidades de la derecha. En él se aprecian las resoluciones que se han marcado como inválidas durante el proceso. Además la tabla adjunta contiene las guardas asociadas a cada resolución.

La tabla que contiene información acerca de las guardas permite que para nueva creación del espacio se tengan en cuenta los errores identificados y de este modo evitar la ejecución de algunas resoluciones en determinadas posibilidades. Las resoluciones inseguras, representadas con una línea discontinua, dejarán de ejecutarse para las posibilidades que tienen anotadas en la guarda.

Teniendo en cuenta las situaciones detectadas como anómalas, el espacio tras el primer refactoring quedará como representa la figura 60.

Al observar la representación siguiente, resultado del primer refactoring, llama la atención como el espacio queda parcialmente desconectado. Es decir algunas posibilidades, hasta ahora, alcanzables dejan de estar accesibles.

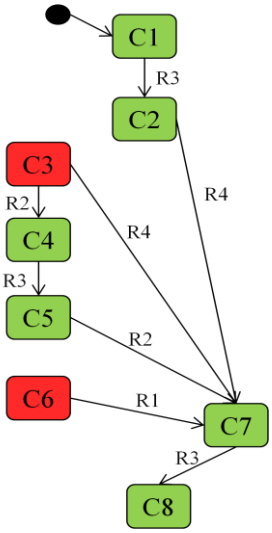


Figura 59 - Resultado del primer refactoring

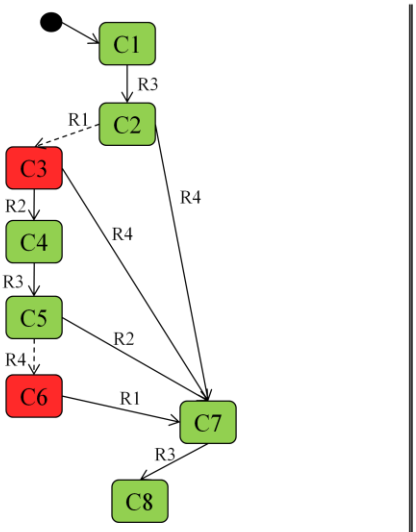
Ejemplo de ello es C3 y las posibilidades C4, C5, C6 que dependían de ella para su ejecución. Estas posibilidades eran únicamente alcanzadas a través de C2. La restricción añadida en la guarda de R1, impide que sea ejecutada y pueda alcanzar C3. Consecuentemente tampoco pueden ser alcanzadas las posibilidades que se generaban en esta rama.

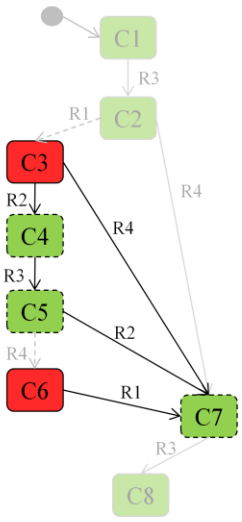
El propósito del segundo refactoring será permitir que de nuevo sean alcanzables.

Para el segundo refactoring se toma como partida el espacio de posibilidades resultante del primer refactoring. Se pretende identificar las posibilidades que son alcanzadas a través de posibilidades marcadas como inválidas.

El objetivo es crear nuevas resoluciones que permitan que las posibilidades alcanzables a través de inválidas continúen siendo alcanzables en el espacio.

En la figura 60 la representación de la izquierda coincide con el espacio resultante del primer refactoring. En este punto, se consultan las posibilidades que se alcanzan a través de las posibilidades C3 y C6, que son inválidas.





RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}

Figura 60 - Refactoring 2: paso 1-2

A partir de C3 se alcanzan las posibilidades C4, C5 y C7. Estas posibilidades se marcarán como posibilidades alcanzables a través de posibilidades inválidas. Es importante notar que la posibilidad C5 ha sido incluida ya que únicamente es posible alcanzarla pasando por C3, que es una de las inválidas. Como se aprecia en el espacio de la derecha se identifican con una línea discontinua.

Detectadas estas posibilidades, el objetivo es crear nuevas resoluciones que una vez eliminadas las posibilidades inválidas permitan que continúen siendo alcanzables.

En la figura anterior se puede ver como desde C2, pasando por C3, se alcanza C4. La resolución entre C2 y C3 es R1 y la resolución entre C3 y C4 es R2. Para poder seguir alcanzando C4, una vez sea eliminada C3 se debe crear una nueva resolución. La nueva resolución debe aplicar la activación y desactivación de las características tal y como hacían R1 en primera instancia y posteriormente R2.

La nueva resolución, denominada R1+R2, crea una transición directa entre C2 y C4. Es el resultado de aplicar de forma ordenada las modificaciones sobre las características que se definían en R1 y R2. Además esta nueva resolución solo debe lanzarse en la posibilidad C2. Para conseguirlo se añade una condición de guarda sobre esta posibilidad que habilite su disparo.

Resolution(R1 and R2 and C2) =
{*subtitutionsR1* + *subtitutionsR2*}

Resolution(R1 and R4 and C2) =
{*subtitutionsR1* + *subtitutionsR4*}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}
R1+R2	Condition5	C2=True	{(F1, F), (F2, T), (F3, F), (F4, T)}
R1+R4	Condition6	C2=True	{(F1, T), (F2, F), (F3, T), (F4, F), (F5, T)}

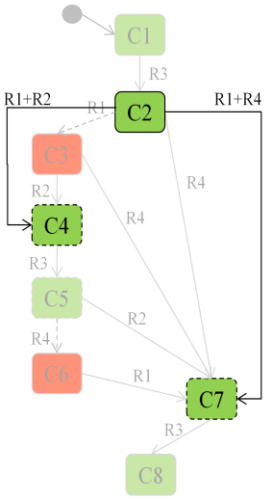


Figura 61 - Refactoring 2: paso 3-4

En la figura 61 ya se puede ver la nueva resolución R1+R2 que enlaza las posibilidades C2 y C4.

Del mismo modo, se repite el proceso entre C2 y C7. La posibilidad C7, además de ser alcanzada directamente por C2, también es alcanzada a través de la posibilidad inválida C3. Esto supone crear una nueva resolución, llamada R1+R4, que conecta las dos posibilidades válidas C2 y C7.

El proceso continua con el resto del espacio sobre las posibilidades alcanzadas a través de la otra posibilidad inválida, C6. Desde C6 solo existe una transición que conduce a alguna posibilidad. La resolución R1 lanzada desde ella alcanza C7.

Como ha ocurrido antes se debe crear una nueva resolución que las conecte. Esta nueva resolución, etiquetada como R1+R4, crea una transición directa entre C6 y C7.

Resolution(R4 and R4 and C5) =
{*subtitutionsR4* + *subtitutionsR1*}

RESOLUTION	CONDITION	GUARD	FEATURES
R1	Condition1	C3=False	{(F1, T), (F2, F), (F3, T), (F4, F)}
R2	Condition2		{(F1, F), (F2, T), (F3, F), (F4, T)}
R3	Condition3		{(F5, F)}
R4	Condition4	C5=False	{(F4, F), (F5, T)}
R1+R2	Condition5	C2=True	{(F1, F), (F2, T), (F3, F), (F4, T)}
R1+R4	Condition6	C2=True	{(F1, T), (F2, F), (F3, T), (F4, F), (F5, T)}
R4+R1	Condition7	C5=True	{(F1, T), (F2, F), (F3, T), (F4, F), (F5, T)}

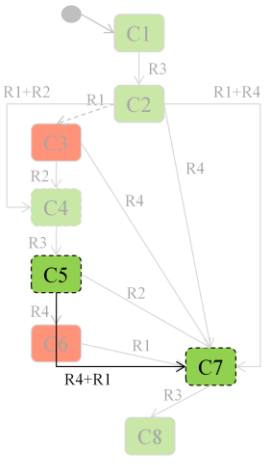


Figura 62 - Refactoring 2: paso 5-6

En la figura 62 se puede ver la nueva resolución creada como resultado de aplicar ordenadamente las resoluciones R4 y posteriormente la R1. A ella se le añade la condición de guarda que permita ser únicamente ejecutada desde la posibilidad C5.

El espacio de posibilidades resultante tras aplicar el primer y segundo refactoring queda como refleja la siguiente figura:

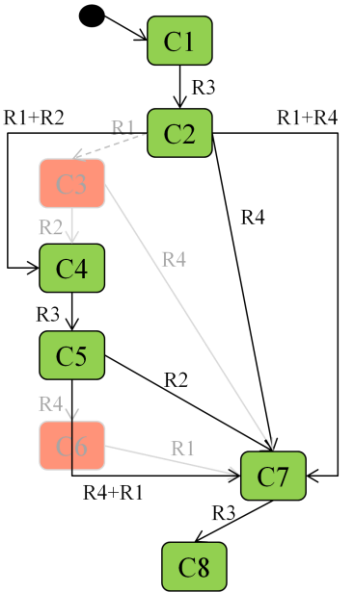


Figura 63 - Espacio de posibilidades libre de errores

Como se puede apreciar en la figura 63, la representación final, ya no existen posibilidades etiquetadas como inválidas y, consecuentemente, tampoco resoluciones inseguras.

Además las posibilidades que aparecían tras el primer refactoring como desconectadas ya pueden ser disparadas a través de las nuevas resoluciones creadas.

El resultado final es un espacio de posibilidades conectado y completamente válido. De este modo, cualquier configuración posible del espacio al aplicar las resoluciones conducirá a situaciones correctas sobre el sistema.

Para representar este es espacio de posibilidades se deben tener en cuenta las restricciones anotadas en la tabla durante el proceso de refactoring.

5. HERRAMIENTA DE SOPORTE A LA PROPUESTA

En este apartado se comenta como la herramienta desarrollada da soporte para trabajar con los elementos presentados en los apartados anteriores y como lleva a cabo los procesos descritos.

Esta herramienta se integra dentro de la plataforma de desarrollo MOSKitt[1]. MOSKitt es una plataforma libre de modelado basada en Eclipse [59] que está siendo desarrollada por la Consellería valenciana de Infraestructuras y Transporte. MOSKitt Feature Modeler (MFM) es el editor de modelos de características de código libre que incorpora MOSKitt. MFM permite la especificación de un modelo de variabilidad en términos de características y las relaciones entre ellas. Ver figura 64.

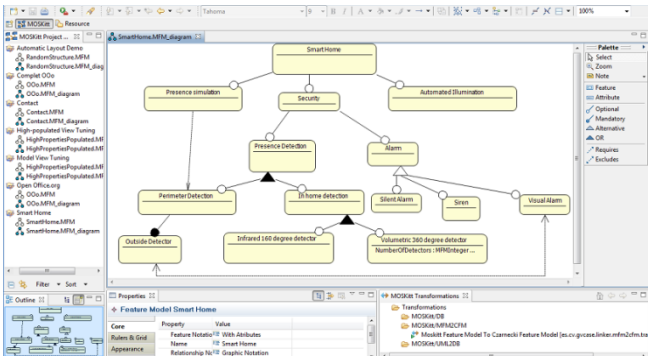


Figura 64 - MOSKitt: Feature Modeler

La herramienta desarrollada está organizada en cuatro perspectivas que ofrecen de forma estructura las diferentes funcionalidades y detalles acerca de cada uno de los componentes.

- Primera perspectiva, *General*

  Se utiliza principalmente para cargar en la herramienta y validar un modelo de características creado previamente. Además una vez cargado ofrece información acerca del estado inicial de las características que componen el modelo. Es decir, la configuración de partida.

[1] http://www.moskitt.org/eng/moskitt0/

- Segunda perspectiva, *Resolutions.*

  Aporta información detallada sobre las resoluciones. También permite la edición de sus propiedades, así como la inserción de nuevas resoluciones y su eliminación. Se utiliza principalmente como editor de resoluciones.

- Tercera perspectiva, *Analysis.*

  Refleja información relativa al espacio de posibilidades. Por una parte, para cada posibilidad se muestran la lista de resoluciones con sus características y su estado, que se pueden aplicar para la posibilidad dada. Por otra parte, la configuración resultante de lanzar las anteriores resoluciones.

- Cuarta perspectiva, *Refactorings*.

  Proporciona la funcionalidad de visualizar de forma gráfica el espacio de posibilidades generado y su versión abstracta. Además permite aplicar los procesos de refactoring que se han explicado anteriormente.

Mediante esta herramienta es posible, entre otras cosas, diseñar el entorno operacional, definir las condiciones de contexto, la lista de resoluciones, validar las configuraciones y representar en forma de espacio de posibilidades las características del sistema.

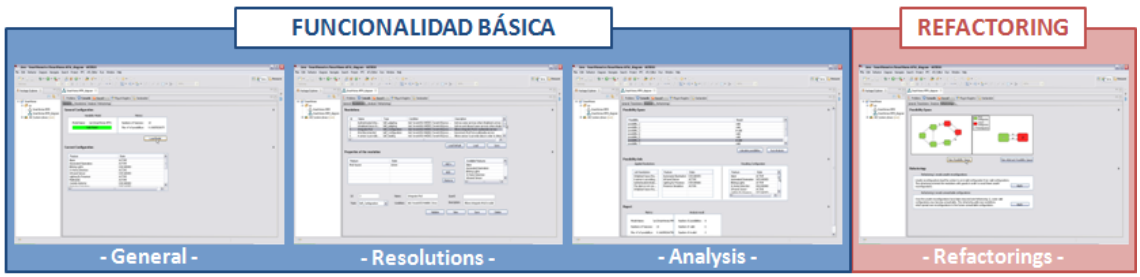


Figura 65 - Organización de las perspectivas de la herramienta

Como indica la figura 65, de forma conjunta se explican las tres primeras perspectivas que cubre la funcionalidad básica y por separado se explica la cuarta perspectiva, que hace referencia al proceso de refactoring.

5.1 FUNCIONALIDAD BÁSICA

En este apartado se detallan las capacidades de la herramienta y se proporcionan unas nociones básicas sobre su utilización. Concretamente se presenta la funcionalidad asociada a las tres primeras perspectivas: *General*, *Resolutions* y *Analysis*. Para acceder a las funciones de cada una de estas perspectivas se han organizado en diferentes pestañas dentro de la herramienta.

Antes de comenzar a utilizar la aplicación es necesario representar en forma de modelo de características el sistema con el que se desea trabajar. Existe a nuestra disposición un amplio abanico de propuestas para representar modelos de variabilidad y, en concreto, modelos de características. En este trabajo se ha elegido la propuesta de MOSKitt, el MOSKitt Feature Modeler.

Como se ha comentado anteriormente, el MOSKitt Feature Modeler (MFM) es un editor de modelos de características de código libre. Se ha optado por esta propuesta porque, además de ser de código abierto y de proporcionar una interfaz intuitiva para representar modelos, soporta el razonamiento sobre características.

A partir de la interfaz proporcionada por el editor de modelos de MOSKitt se representa el modelo de características del sistema deseado. Para ello la herramienta proporciona unos elementos gráficos que simbolizan las características y las relaciones entre ellas. Como viene siendo habitual, una figura en forma de rectángulo redondeado se utiliza para las características y unos conectores en forma de línea para las relaciones.

Como se aprecia en la figura 66 la edición del modelo se realiza de forma ágil arrastrando cada uno de los componentes deseados a partir de la paleta. La paleta, colocada en la parte derecha, contiene todos los elementos necesarios para dibujar nuestro modelo. Además de representar las características, este editor permite, definir de forma gráfica las restricciones asociadas a las relaciones entre ellas. Esto se puede realizar haciendo uso de los conectores adecuados (optional, mandatory, alternative, or, requires y excludes) que también se proporcionan a través de la paleta.

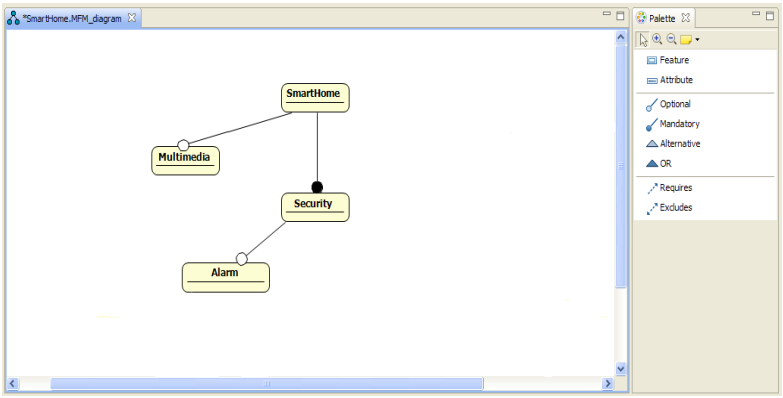


Figura 66 - Editor de modelos de características

Una vez completado el modelo de características la herramienta permite que sea almacenado para posteriores usos. De este modo nos evita tener que dibujarlo repetidas veces y facilita las tareas de modificación para probar diferentes combinaciones.

Finalizado nuestro modelo ya es posible utilizar la herramienta de soporte diseñada donde se proporcionará el modelo como datos de entrada.

5.1.1 PERSPECTIVA GENERAL

La perspectiva *General*, primera pestaña, consiste en un breve resumen de la configuración más genérica del modelo.

La figura 67 es una captura que muestra como queda la perspectiva *General* tras la carga del modelo de características.

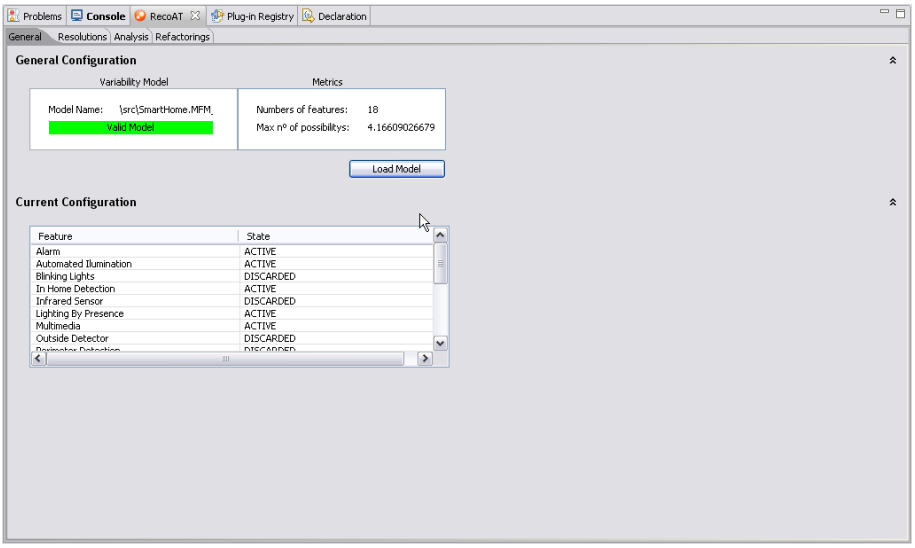


Figura 67 - Perspectiva General

Esta perspectiva permite la carga de un modelo de características que haya sido creado previamente, por ejemplo, mediante el editor de MOSKitt. Al cargar el modelo la herramienta comprueba la configuración que se ha definido y la valida de acuerdo a las restricciones impuestas. Además ofrece un breve informe sobre el número de características y el número máximo de posibilidades que su combinación podría generar.

En la parte inferior se acompaña una tabla que muestra el estado de cada una de las características definidas en el modelo y que constituyen la configuración actual.

5.1.2 RESOLUTIONS

La perspectiva *Resolutions*, segunda pestaña, permite la gestión de las resoluciones. Esta herramienta proporciona un editor de resoluciones el cual hace uso del modelo de características y de las condiciones de contexto.

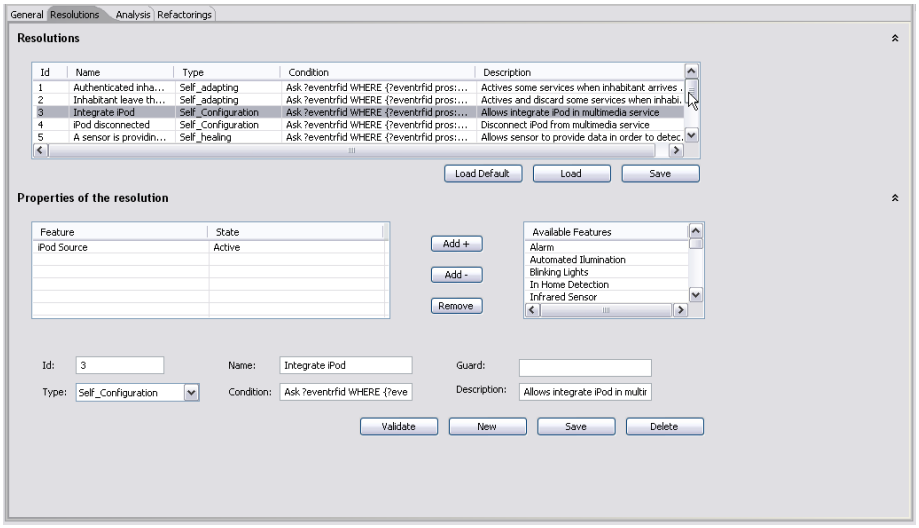


Figura 68 - Perspectiva Resolutions

La figura 68 es una captura que muestra como queda distribuida la perspectiva *Resolutions*.

Esta perspectiva contiene toda la información asociada a las resoluciones. En la parte superior aparecen listadas las resoluciones definidas con sus atributos: nombre, tipo, condición y descripción. Al seleccionar cualquier resolución se detallan en los campos inferiores la información de sus atributos y el estado de sus características asociadas.

Funciona principalmente como editor de resoluciones, donde se permite crear, modificar y eliminar. Para crear nuevas resoluciones pulsaremos sobre el botón *New*, esto dejará en blanco los campos asociados a los datos de las resoluciones. Una vez se hayan introducidos todos los datos es posible almacenarla en el sistema mediante el botón *Save*. Para modificar cualquiera de las resoluciones ya creadas será suficiente con hacer clic sobre ella, editar los campos requeridos y almacenarla.

Este editor consulta el modelo de características con el propósito de mostrar la lista de características disponibles. Además es posible fijar el estado de cada característica para establecer una configuración parcial del sistema. Una configuración parcial, es un conjunto de características del sistema que describen el efecto de una resolución cuando su condición de contexto se cumple.

Se pretende que la creación y manipulación de resoluciones se haga de una forma rápida y simple.

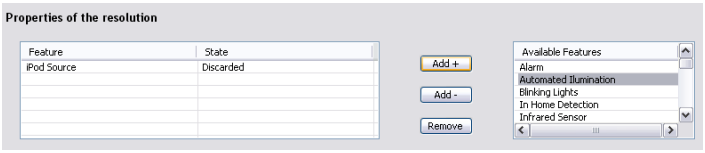


Figura 69 - Añadir/eliminar características

En la figura 69 se muestra con más detalle el proceso de añadir o eliminar características a una determinada resolución y de fijar el estado en el que quedará. El botón *Add+* añade la característica seleccionada como activada, mientras que el botón *Add-* añade la misma característica pero en estado inactivo. Obviamente, el botón *Remove* sirve para eliminar una característica previamente añadida a la resolución que se está editando.

Un aspecto a destacar es la capacidad de analizar si la nueva resolución, es válida o no, gracias al botón *Validate* situado en la parte baja de la pantalla. La validación consiste en comprobar si una determinada resolución es coherente, es decir, cumple las restricciones y no es contradictoria respecto al resto. Esta validación está basada en el análisis realizado por el framework FaMa [20] para determinar si una determinada configuración es válida o no de acuerdo a las restricciones de variabilidad.

Cabe destacar el campo etiquetado como *Guard* que se utilizará posteriormente para anotar las restricciones que pueden tener las resoluciones.

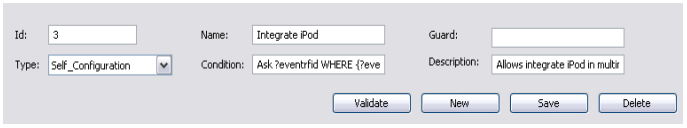


Figura 70 - Campo guard

La figura 70 muestra con más detalle los atributos de una resolución. Concretamente hace referencia a la resolución cuyo id es 3.

El desplegable *Type* permite definir el tipo de la resolución de acuerdo a los tipos definidos en apartados previos.

- Self-configuring

  Este tipo de resolución sirve en el caso de que nuevos tipos de dispositivos pueden ser incorporados al sistema. Por ejemplo, cuando un nuevo sensor de presencia se agrega al sistema, los diferentes servicios del hogar inteligente, como la seguridad o el control de iluminación, automáticamente pueden hacer uso de él sin necesidad de que el usuario lo tenga que configurar.

- Self-healing

  Este tipo de resolución es útil cuando un dispositivo se retira o falla, el sistema debería adaptarse para ofrecer sus servicios de forma alternativa, y así reducir el impacto de la pérdida del dispositivo. Por ejemplo, si una alarma falla, la casa inteligente puede encender y apagar las luces como una alternativa a la alarma fallida.

- Self-adapting

  Este tipo de resolución se utiliza en el caso de que las necesidades del sistema varíen en función del usuario y momento dado. El sistema deberá ajustar sus servicios para cumplir con las preferencias del usuario. Por ejemplo, cuando un usuario sale de casa, los servicios del hogar deben ser reorganizados para dar prioridad a la seguridad.

5.1.3 ANALYSIS

La perspectiva *Analysis*, tercera pestaña, proporciona información acerca de las reconfiguraciones.

Cuando se define una resolución para la activación/desactivación de un conjunto de características, se está expresando la transición entre dos configuraciones del sistema. Esta herramienta también proporciona un mecanismo que da soporte al análisis de la reconfiguración. Este mecanismo está basado en el análisis operaciones del framework FaMa[60] para determinar si una configuración dada es válida o inválida de acuerdo a las restricciones. Para todo lo anterior se utiliza la siguiente perspectiva.

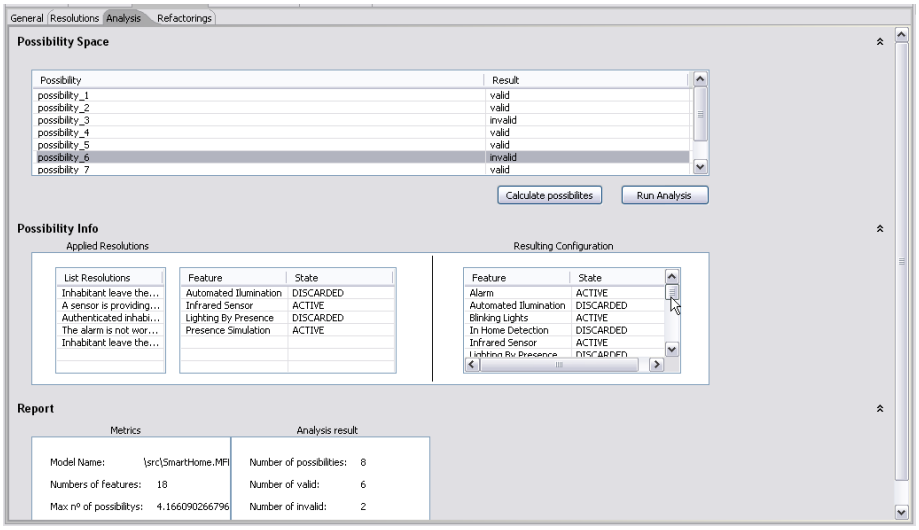


Figura 71 - Perspectiva analysis

La figura 71 es una captura que muestra como queda distribuida la perspectiva *Analysis*.

Esta perspectiva proporciona información acerca de las posibilidades. El primer paso es calcular las posibilidades que genera el modelo de características. A partir del botón *Calculate possibilies* se obtienen las combinaciones posibles de las resoluciones cargadas con el modelo.

Para calcular el espacio de posibilidades la herramienta tiene en cuenta el cumplimiento, no sólo de una condición de contexto, sino también las de varias posibilidades.

A partir de las posibilidades se procede al análisis. *Run Analysis* estudia la validez de todas las posibilidades generadas anteriormente y para cada una de ellas indica si su estado es válido o inválido. Para la validación se recurre al framework FaMa. Este indica no solo la validez de cada configuración sino también las razones por las que una determinada configuración es invalida.

La tabla superior, como se aprecia en la figura 72, muestra el nombre y estado de cada posibilidad tras el análisis.

Figura 72 - Resultado del análisis

En la figura 73 se presenta la información de las posibilidades. Concretamente en la parte izquierda, seleccionando cualquiera de las posibilidades se pueden obtener las resoluciones aplicadas junto a las características y su estado. La tabla derecha permite conocer el resultado de fusionar las características de las resoluciones de cada posibilidad con las del modelo. De igual modo, se listan las características junto a su estado actual.

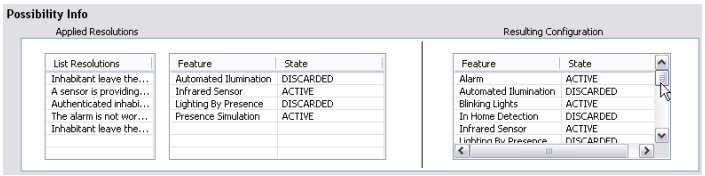


Figura 73 - Información de las posibilidades

La parte baja de esta perspectiva, figura 74, proporciona un breve informe sobre el número de características, el número de posibilidades y aquellas que han resultado válidas e inválidas.

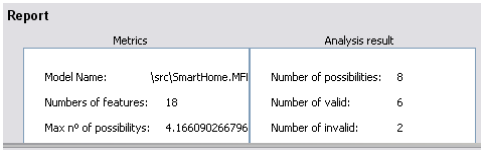


Figura 74 - Informe

5.2 PROCESO DE REFACTORING

La perspectiva *Refactorings*, cuarta y última pestaña, como su propio nombre indica proporciona los mecanismos para aplicar el proceso de Refactoring sobre el modelo. También su posterior representación en términos de espacio de posibilidades.

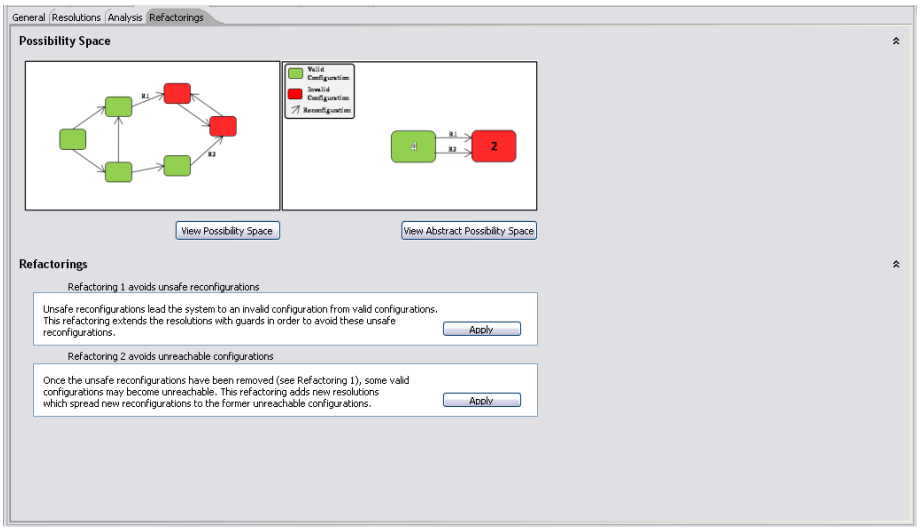


Figura 75 - Perspectiva refactorings

La figura 75 es una captura que muestra como queda distribuida la perspectiva *Refactorings*.

Esta perspectiva se dedica a la generación de los espacios de posibilidades para obtener una visualización gráfica. Además se implementan los procesos de refactoring que dejarán el modelo y su correspondiente representación libre de errores.

En la parte superior destacan dos imágenes que pretenden explicar de una manera intuitiva la generación del árbol de posibilidades y su respectiva representación abstracta.

En la parte inferior, figura 76, se describen brevemente los procesos de refactoring asociados a cada botón. Mediante el botón superior se eliminan las reconfiguraciones inseguras (primer refactoring) y el inferior elimina las configuraciones alcanzables a través de posibilidades inválidas (segundo refactoring).

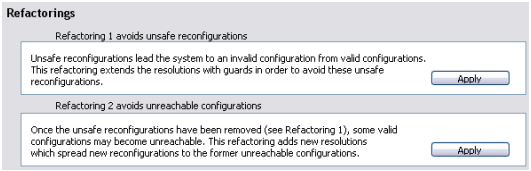


Figura 76 – Descripción refactoring

Después de realizar estos refactorings podemos visualizar la nueva representación del espacio de posibilidades.

Mediante el botón *View Possibility Space* obtendremos la representación del modelo en un árbol de posibilidades y mediante el botón *View Abstract Possibility Space* se calculará y representará su versión en notación abstracta.

6. CASOS DE ESTUDIO

Figura 77 - Alcance del capítulo 6

En este apartado se aplican sobre dos casos de estudio todos los conceptos, técnicas y procedimientos explicados en apartados anteriores.

El objetivo es comprobar el correcto funcionamiento de la herramienta realizando un proceso completo y al mismo tiempo demostrar su aplicación sobre un problema real. Para ello, se toma como caso de estudio una Smart Home.

La estructura de este apartado queda como sigue. En primer lugar, se presenta una visión general del contexto asociado al caso de estudio. Se comentan las características principales y las razones que justifican la elección de una Smart Home como ámbito de trabajo.

En segundo lugar, se introducen los detalles bajo los cuales se tratará el caso y las funcionalidades que formarán parte de la Smart Home. Además estas funcionalidades se describen y se presentan de una forma gráfica.

En tercer y último lugar, se aplica la descripción del caso de estudio sobre la herramienta de soporte desarrollada. En este punto se especifican por separado las características de cada uno de los dos casos que se van a tratar. El primer caso pretende ofrecer una visión completamente detallada. Mientras que el objetivo del segundo es demostrar la escalabilidad de esta propuesta.

6.1 PANORÁMICA DEL CASO DE ESTUDIO

El contexto de ambos casos de estudio se centra en una Smart Home, también conocida como hogar inteligente. La elección de un hogar digital como caso de estudio está fundamentada en dos razones principales. En primer lugar, la propia naturaleza de una casa donde intervienen diferentes perfiles de personas. Cada persona tiene sus propias preferencias y necesidades dentro del hogar y éste debe ajustarse de la mejor forma posible a cada una de ellas. En segundo lugar, las preferencias de estos habitantes cambian de acuerdo a la actividad llevada a cabo. Por ejemplo, son diferentes las necesidades del usuario cuando está trabajando que cuando está viendo la televisión.

Este dominio es ideal para las técnicas de modelización de la variabilidad por el alto grado de similitud entre los diferentes sistemas y también por las capacidades de computación autónoma que puede abordar algunas de las limitaciones del dominio. Como un apoyo mínimo para la evolución de las nuevas tecnologías emergentes o como un tipo de aplicación madura [19, 61].

La Smart Home cuenta con un conjunto de elementos que dotan al hogar de una serie de funcionalidades especiales. Este entorno debe ser capaz de adaptarse a las cambiantes exigencias de los usuarios, añadir y retirar dispositivos de una forma autónoma y proporcionar diferentes servicios.

El tipo de entorno inteligente referido aquí está formado por numerosos dispositivos físicos tales como sensores (por ejemplo, de presencia o de intensidad de la luz), actuadores (por ejemplo, la iluminación o la alarma) y multimedia (por ejemplo, el hilo musical o la TDT). La funcionalidad de estos dispositivos se ve aumentada por el alto nivel de servicios. Estos servicios coordinan la interacción entre los dispositivos para realizar tareas específicas. Por ejemplo, un servicio de simulación de presencia disuade a los ladrones, actuando como si no hubiera gente en el hogar. Para lograr este objetivo, el servicio de simulación de presencia coordina las luces, el reproductor multimedia, la televisión y las persianas.

Con el tiempo los dispositivos y los servicios que utiliza el usuario cambian. Por ejemplo, un usuario puede comprar inicialmente una lámpara y más tarde puede comprar un sensor de luz. La idea es que pueda conectarlo a la Smart Home y ésta a su vez pueda configurar el servicio de iluminación para que sea proactivo. Este mismo comportamiento es el que se desea con todo tipo de dispositivos, como un iPod. Si el usuario compra un iPod se espera que pueda volver a configurar el servicio multimedia para poder acceder a la biblioteca de música o para escuchar la música a través de los altavoces del hogar.

6.2 FUNCIONALIDAD DE LA SMART HOME

La Smart Home definida para el caso de estudio contará con 4 características o servicios principales sobre los que se recogerán el resto de funcionalidades del sistema.

Las principales características son:

- Multimedia

  El propósito de esta característica es agrupar los servicios multimedia que puedan implementarse en el hogar. Concretamente esta propuesta cuenta con hilo musical y la posibilidad de utilizar el Ipod como fuente de datos.

- Seguridad

  Bajo esta característica se recogen los elementos que ofrecen su funcionalidad para proporcionar servicios de seguridad. En este caso, cuenta con un sistema de alarma, detección de presencia en el perímetro y en el hogar. Además el sistema de alarma está formado por alarma silenciosa, sirena, alarma visual y luces parpadeantes.

- Simulador de presencia

  Esta característica recoge el servicio que simula la presencia de personas en el hogar con el objetivo de disuadir a posibles intrusos.

- Iluminación automática

  Esta característica recoge el servicio de iluminación inteligente que enciende o apaga las luces basándose en la detección de presencia.

Además de estas caracteríticas, el sistema cuenta con unas restricciones:

- En la Smart Home se le da mucha importancia a la seguridad del sistema, por lo cual siempre que el sistema esté encendido el sistema de seguridad obligatoriamente tendrá que estar encendido.
- El sistema de alarma gestiona diferentes dispositivos, pero solo uno de ellos estará activo en un instante dado.
- Para que el sistema de simulación de presencia pueda funcionar requiere necesariamente que el sistema detección en el hogar esté conectado.
- Para que los sistemas de simulación de presencia y iluminacion automática no se vean afectados por conflictos no pueden conectarse simultáneamente.

Estas restricciones deben quedar reflejadas en el modelo. Para ello se hace uso de la notación de restricciones que se ha comentado en el apartado 3.2.1.

La siguiente figura representa la funcionalidad de la Smart Home y sus posibles variantes haciendo uso de la notación de modelos de características. Los diferentes cuadrados muestran las funciones disponibles en el hogar. De color gris se identifican las características activas y de color blanco las inactivas.

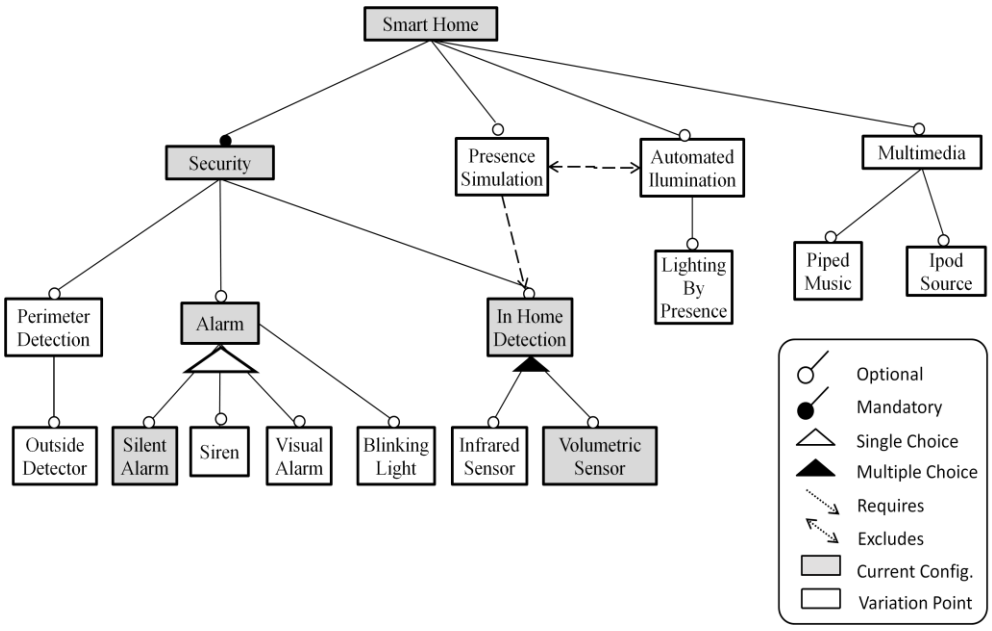


Figura 78 - Modelo de características

El propósito del modelo de características es representar las diferentes características que la Smart Home implementa para proporcionar unos servicios que ajusten la estancia a las necesidades de la persona residente.

Las características del modelo cambian, activándose o desactivándose, de acuerdo a la situación de cada momento. Estas variedad de configuraciones del hogar permiten identificar diferentes escenarios. De este modo, las características grises indican la situación actual y las características blancas variantes posibles que pueden ser activadas en el futuro.

El modelo de características de la figura 78 describe una Smart Home con las siguientes características: Multimedia, Seguridad, Simulación de presencia e iluminación automática. Estas características están conectadas jerárquicamente en una estructura tipo árbol a través de relaciones de variabilidad, como opcional, obligatoria, única opción o múltiple opción.

Como se ha comentado en la descripción de la Smart Home, además del conjunto de servicios y dispositivos, el sistema debe ser capaz de tomar de forma autónoma ciertas decisiones para adaptar su comportamiento conforme cambien las condiciones.

La descripción de esos cambios se define mediante una serie de resoluciones que llevarán a cabo unas determinadas acciones que modificarán el comportamiento de la Smart Home.

A modo de ejemplo, se espera que el sistema sea capaz de detectar la entrada o salida de los residentes en el hogar. La detección de los habitantes llevará al sistema a una nueva configuración de la casa. Esta nueva configuración supone que algunos de los servicios pasen a estar activos o inactivos. La llegada de cualquier residente implica activar los servicios de iluminación automática y de iluminación por presencia. Mientras que supone desactivar el simulador de presencia y los sensores infrarojos.

El ejemplo anterior es solo una muestra del tipo de cambios que se espera que la Smart Home sea capaz de llevar a cabo sin la intervención humana.

6.3 HERRAMIENTA DE SOPORTE

En este apartado se aplica el caso de estudio descrito anteriormente sobre la herramienta de soporte desarrollada. La herramienta se va a poner a prueba sobre dos casos de estudio. El primero pretende dar una visión profunda sobre cada uno de los elementos y métodos llevados a cabo. Con este propósito el problema elegido es de una complejidad menor y cada uno de los procesos se detallarán paso a paso. El segundo, de una envergadura considerable, ejemplifica la capacidad de aplicar estos procedimientos sobre problemas complejos.

Para ello, se toma la descripción del caso para representarla en términos del lenguaje utilizado por la herramienta.

El primer paso para aplicar este caso de estudio sobre la herramienta consiste en identificar todos los dispositivos y concretar todas las características con las que contará la Smart Home. En función de esto se establecen las relaciones entre ellos y se definen todos los cambios que será capaz de gestionar.

De acuerdo a la descripción presentada en el apartado anterior, para el caso de estudio se definen las siguientes funcionalidades:

- Multimedia: agrupa todos los servicios multimedia: música, TV, cine, etc.
    - Piped Music: Hilo musical de la casa.
    - Ipod Source: Contenido multimedia del reproductor de música Ipod.

- Security: Agrupa todos los servicios de seguridad de la casa.
    - Alarm: Sistema que gestiona los diferentes sistemas de alarmas de la casa.
        - Silent Alarm: Sistema de aviso a policía, seguridad privada etc.
        - Siren: Accionador de fuertes sonidos disuasorios.
        - Visual Alarm: Accionador de fuertes luces disuasorias.
        - Blinking Lights: Accionador de luces parpadeantes disuasorias.

    - Perimeter Detection: Sistema de perímetro de detección.
        - Outside detector: Sistema de sensores exteriores de presencia.

    - In Home Detection: Sistema de detección del hogar.
        - Infrared Sensor: Sistema de sensores de presencia interior por infrarrojos.
        - Volumetric Sensor: Sistema de sensores de presencia por volumen.

- Presence Simulation: Sistema de simulación de presencia, simula la presencia de habitantes en la casa siguiendo una rutina de encendido de dispositivos.

- Automated Ilumination: Sistema de iluminación automática del hogar.
    - Lighting By Presence: Sistema de iluminación basado en sensores de presencia.

Todos estos elementos quedarán englobados bajo la característica Smart Home.

El siguiente paso, y haciendo uso del editor de la aplicación se genera la representación de la SmartHome en términos de modelo de características. La figura 79 muestra el modelo tras tener en cuenta los dispositivos disponibles, los servicios deseados y las restricciones que deben cumplir.

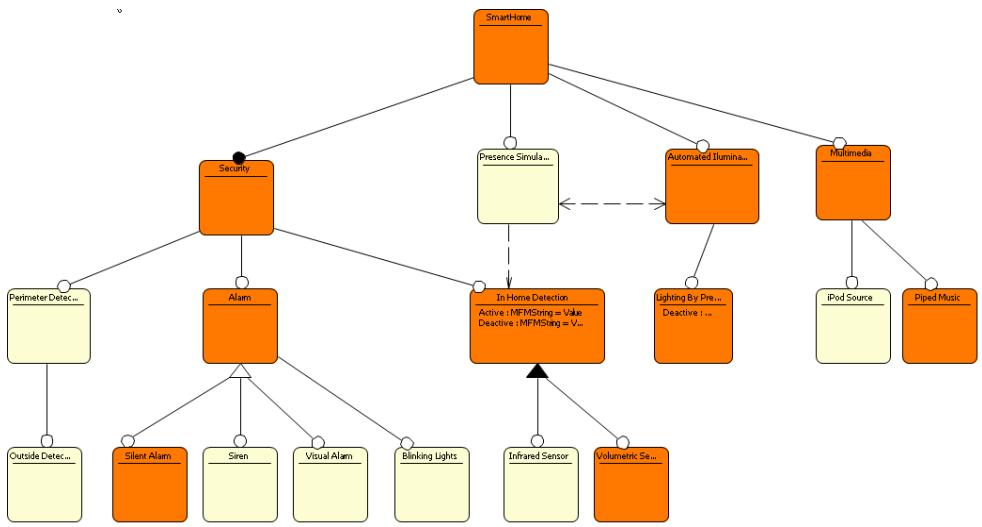


Figura 79 - Representación de la Smart home en la herramienta

Las características coloreadas de naranja representan aquellos servicios que están activos, por el contrario, las características amarillas aquellos servicios inactivos.

El modelo de la figura 79 ya representa la configuración inicial del sistema. Como se puede apreciar en la figura, los servicios principales son Security, Automated Ilumination, Presence Simulation y Multimedia. El sistema multimedia activa el hilo musical (Piped Music). El sistema de iluminación automática (Automated Ilumination) mantiene activo el servicio de iluminación por presencia (Lighting by presence). Por su parte, el servicio de simulación de presencia (Presence Simulation) inicialmente estará inactivo. Por último, el sistema de seguridad está activo habilitando la alarma (Alarm), concretamente la version silenciosa (Silent Alarm) y también hace uso de la detección de presencia (In Home Detection) en el hogar mediante el sensor volumetrico (Volumetric Sensor).

Creado el modelo de características de la Smart Home ya puede ser cargado en la herramienta a través de la primera perspectiva, como queda reflejado en la figura 80.

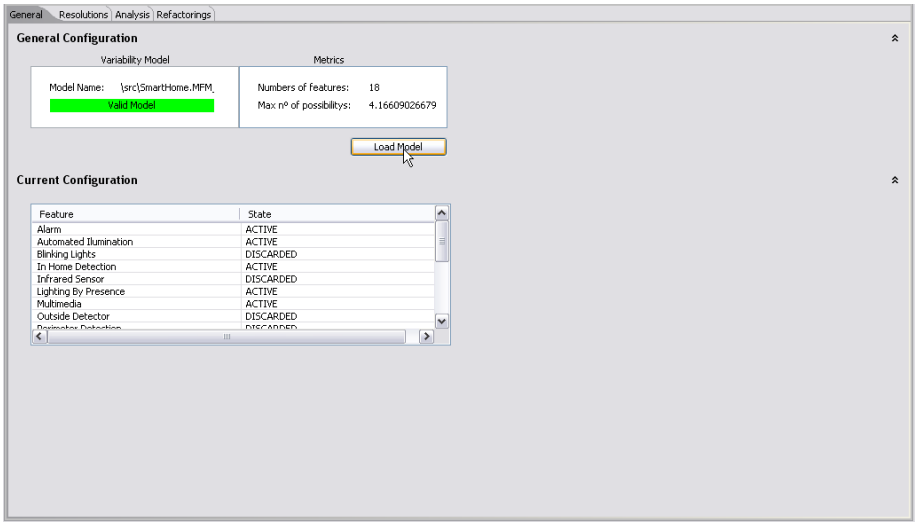


Figura 80 - Carga del modelo en la herramienta

Se puede apreciar que la tabla inferior, *Current Configuration,* refleja la configuración inicial detallada en el punto previo.

El siguiente paso consiste en definir el conjunto de resoluciones con las que contará el sistema. Estas resoluciones se han diseñado pensando en los cambios que pueden darse en la Smart Home.

La tabla 1 resume algunas de las resoluciones:

RESOLUCIÓN	DESCRIPCIÓN	CARACTERÍSTICAS	ESTADO
Resolution 1	Authenticated inhabitant arrives home	Automated Ilumination	Active
		Infrared Sensor	Discarded
		Lighting by presence	Active
		Presence Simulation	Discarded
Resolution 2	Inhabitant leave the house	Automated Ilumination	Discarded
		Infrared Sensor	Active
		Lighting by presence	Discarded
		Presence Simulation	Active
Resolution 3	Integrate iPod	iPod Source	Active
Resolution 4	iPod disconnected	iPod Source	Discarded
Resolution 5	A sensor is providing data	In Home Detection	Discarded
		Volumetric Sensor	Discarded
Resolution 6	The alarm is not working	Blinking Lights	Active
Resolution 7	Infrared detector is replaced	Infrared Sensor	Discarded
		Volumetric Sensor	Active
Resolution 8	New presence detector sensor is plugged	Volumetric Sensor	Active

Tabla 1 - Listado de resoluciones

Estas resoluciones se insertan en la perspectiva *Resolutions*, como se aprecia en la figura 81.

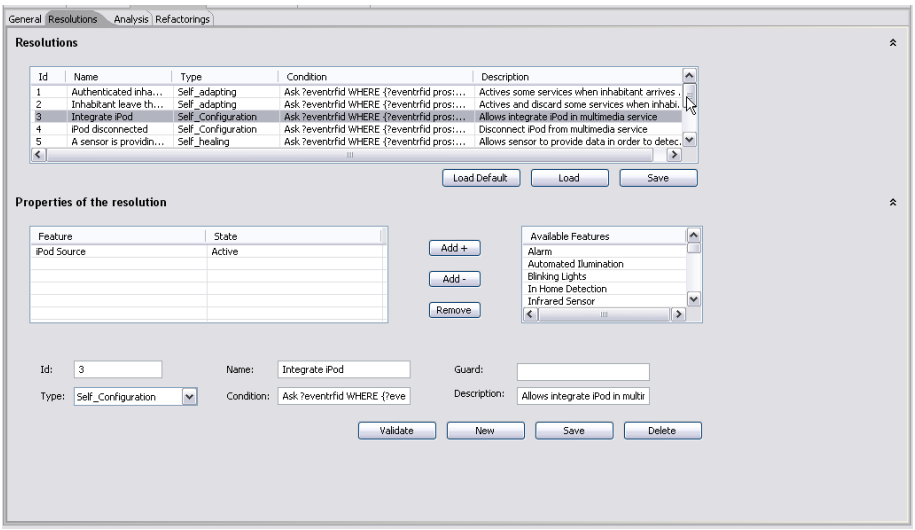


Figura 81 - Inserción de resoluciones

Toda la información presentada hasta ahora es común a los dos casos de estudio. A partir de este punto se van a concretar los elementos específicos que intervendrán en cada uno de ellos.

6.4 CASO 1: EJEMPLO DETALLADO

Para este primer estudio el número de resoluciones disponibles en el sistema se van a reducir a 5 con la intención de limitar el tamaño del espacio de posibilidades. De forma aleatoria se han elegido las resoluciones R1, R2, R4, R5 y R6.

6.4.1 FUNCIONALIDAD BÁSICA

Una vez está cargado y validado el modelo de características e introducida la lista de resoluciones, el siguiente paso es calcular toda la variedad de posibilidades resultantes de combinarlos.

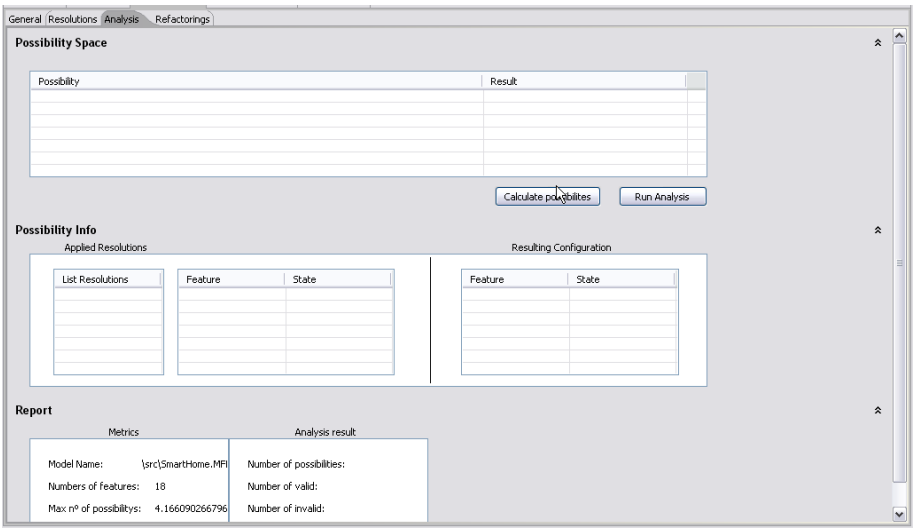


Figura 82 - Cálculo del espacio de posibilidades

Para realizar este paso haciendo uso de la herramienta se despliega la perspectiva *Analysis*, como se aprecia en la figura 82. A partir de ella se pueden calcular el espacio de posibilidades.

Calculadas las posibilidades se puede lanzar el análisis para comprobar la validez de cada una de ellas. Van acompañadas de las características asociadas y la configuración resultante, como indica la figura 83.

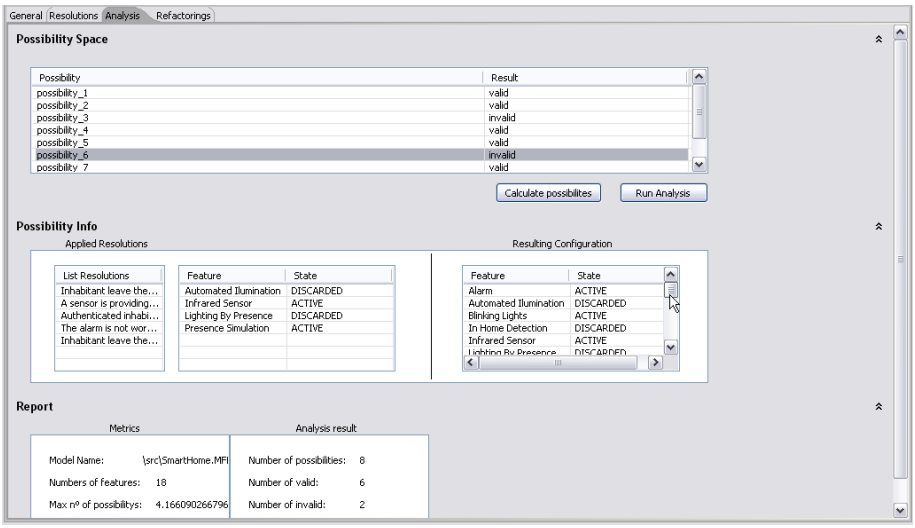


Figura 83 - Validación de las posibilidades

Como resultado de ambos procesos obtenemos un pequeño informe. Podemos observar, a través de la figura 84, que el numero de posibilidades es 8, de las cuales 6 han resultado válidas y 2 inválidas.

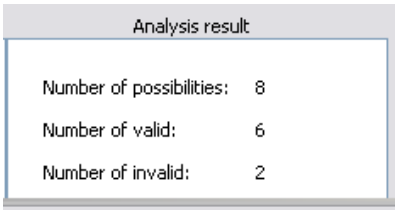


Figura 84 - Informe tras el análisis

A través de la perspectiva *Refactorings*, figura 85, se puede obtener la visualización del espacio de posibilidades que, en definitiva, el modelo de características ha generado.

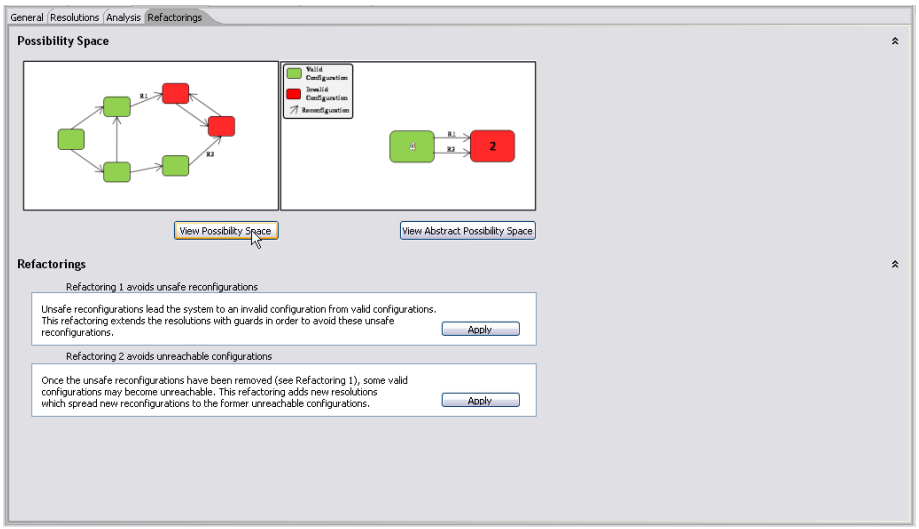


Figura 85 - Visualizar el espacio de posibilidades

La representación gráfica nos permitirá identificar de forma visual el estado de las posibilidades y las resoluciones lanzadas en cada situación. El espacio de posibilidades queda como indica la figura 86.

Como se había previsto en el informe generado por la herramienta el espacio consta de 8 posibilidades, 2 de las cuales permanecen en un estado inválido: posibilidades C3 y C6. Además podemos apreciar como las transiciones están etiquetadas con las resoluciones que se habían insertado en la herramienta: resoluciones R1, R2, R4, R5 y R6.

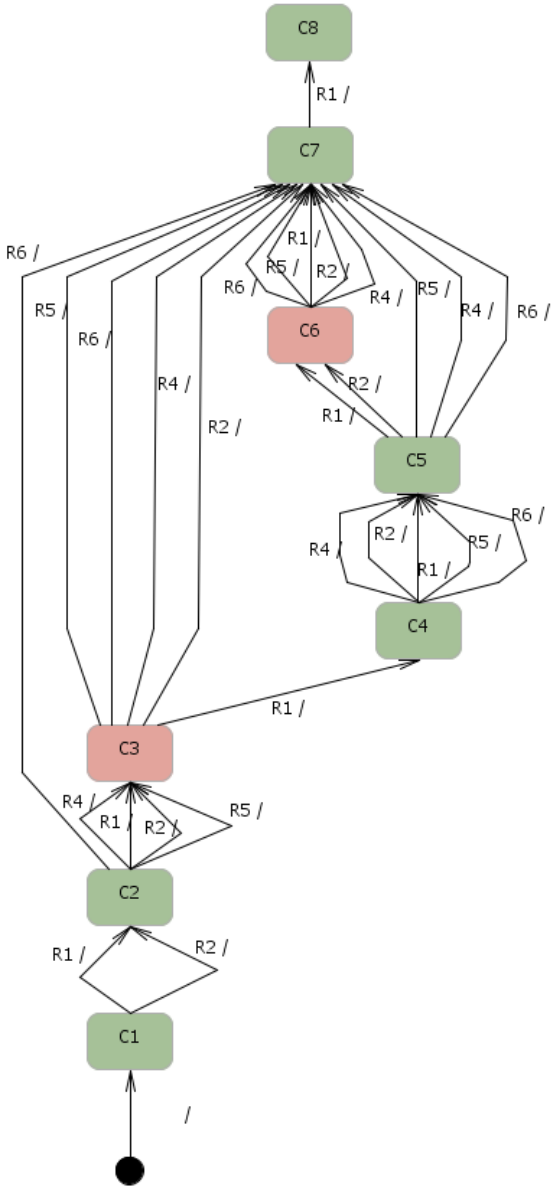


Figura 86 - Espacio de posibilidades

6.4.2 PROCESO DE REFACTORING

El espacio de posibilidades generado contiene dos posibilidades que incumplen las propiedades del sistema y que por lo tanto son inválidas. Mediante el proceso de refactoring se pretende dejar el sistema libre de errores. Como se ha explicado en el capítulo 4 este proceso está compuesto por dos refactorings.

A través de la perspectiva *Refactorings*, figura 87, podemos ejecutar cada uno de ellos.

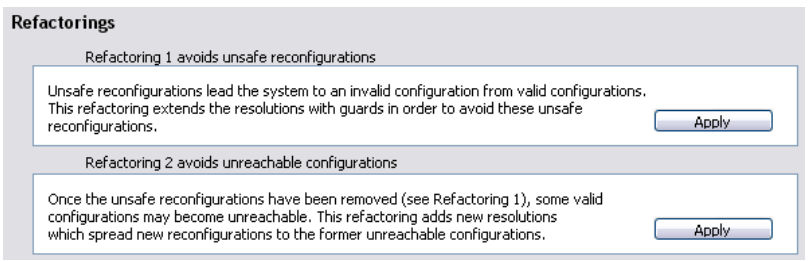


Figura 87 - Sección para aplicar los refactorings

6.4.2.1 PRIMER REFACTORING (R1)

El objetivo del primer refactoring consiste en no permitir que se lancen resoluciones hacia posibilidades que han resultado inválidas tras el análisis. Esto se consigue añadiendo una condición de guarda en las resoluciones impidiendo que sean ejecutadas en determinadas posibilidades.

Tras aplicar el primer refactoring, en la perspectiva *Resolutions*, se puede ver como el campo *Guard*, hasta ahora vacío, ha sido rellenado para algunas resoluciones. Concretamente se ha rellenado para las resoluciones que alcanzaban posibilidades inválidas.

En el recuadro marcado de la figura 88 se puede apreciar como la resolución 1 ha sido complementada con unas condiciones de guarda. Esta resolución a través de las posibilidades C2 y C5 alcanzaba a las posibilidades inválidas C3 y C6 respectivamente. Con estas condiciones en el campo *Guard* se impide que la resolución 1 las pueda alcanzar.

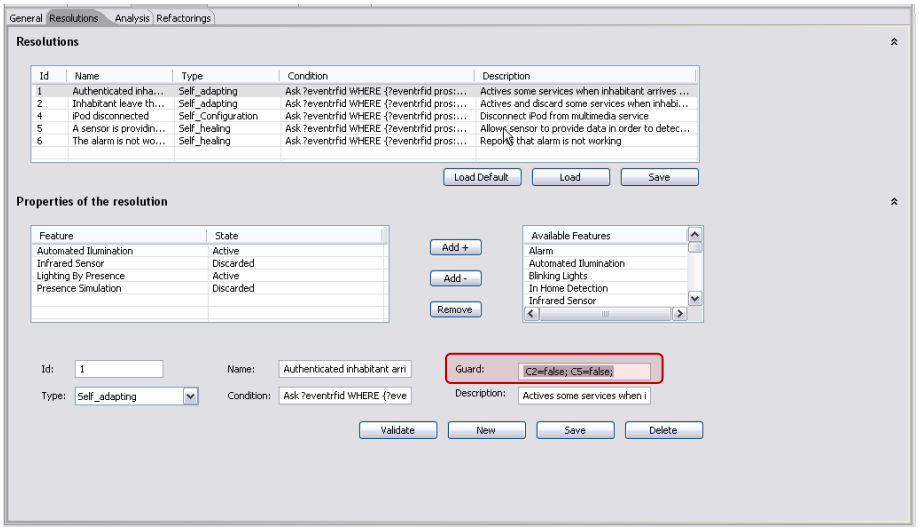


Figura 88 - Condiciones de guarda en las resoluciones tras aplicar el primer refactoring

Si tras este primer refactoring se visualiza el nuevo espacio de posibilidades llama la atención como no hay resoluciones con posibilidades destino inválidas.

La figura 89 refleja cómo queda el espacio de posibilidades tras el primer refactoring.

Tal y como se puede apreciar en el espacio de posibilidades generado tras el primer refactoring debido a las restricciones impuestas en algunas resoluciones han quedado posibilidades válidas que no son accesibles. Por ejemplo, la posibilidad C4 siendo válida no cuenta con ninguna resolución que le alcance, quedándose aislada y consecuentemente todas las posibilidades que dependen de ella. Resolver estas situaciones será el objetivo del segundo refactoring.

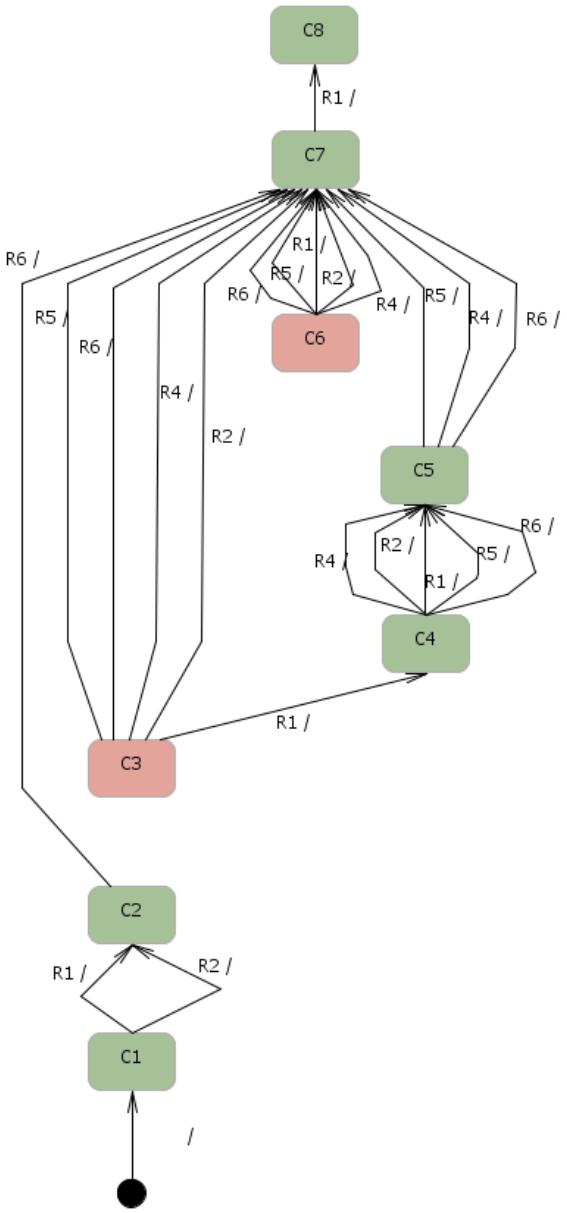


Figura 89 - Espacio posibilidades tras el primer refactoring

6.4.2.2 SEGUNDO REFACTORING (R2)

Este segundo refactoring consiste en la creación de nuevas resoluciones que permitan que las posibilidades alcanzadas a través de posibilidades inválidas continúen siendo accesibles. Por ejemplo, creando una resolución, que de acuerdo a las condiciones del espacio y de forma válida, alcance a C4.

Tras ejecutar el segundo refactoring, en la perspectiva *Resolutions*, se puede ver como aparecen nuevas resoluciones. Estas resoluciones son el resultado de aplicar ordenadamente varias resoluciones.

Además el campo *Guard*, hasta ahora vacío para estas resoluciones, ha sido completado con las posibilidades para las cuales será posible ejecutarlas.

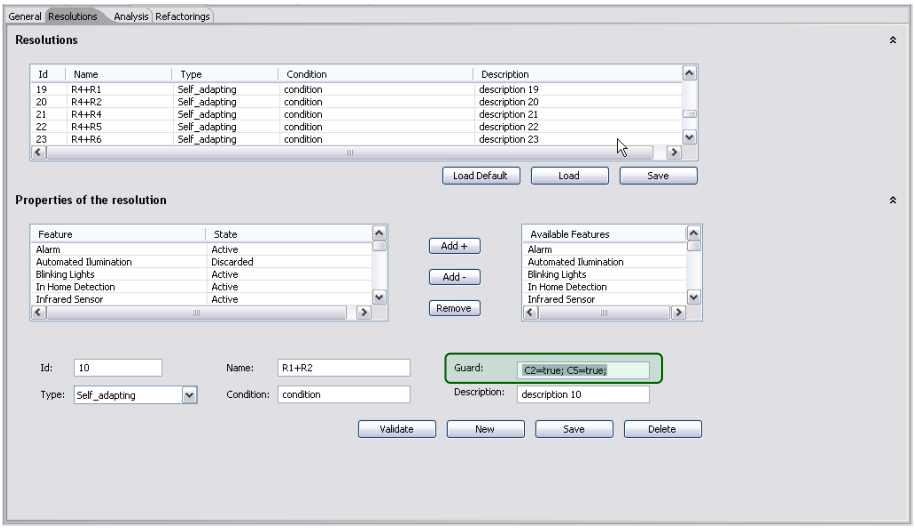


Figura 90 – Creación de nuevas resoluciones

En la figura 90 se pueden ver las nuevas resoluciones. En esta misma figura se puede apreciar marcado en un recuadro las guardas que se han añadido a las nuevas resoluciones.

Es oportuno recordar que se utilizarán para limitar la ejecución de las nuevas resoluciones. Estas nuevas resoluciones son las que permitirán alcanzar aquellas posibilidades válidas que se habían quedado desconectadas tras el primer refactoring.

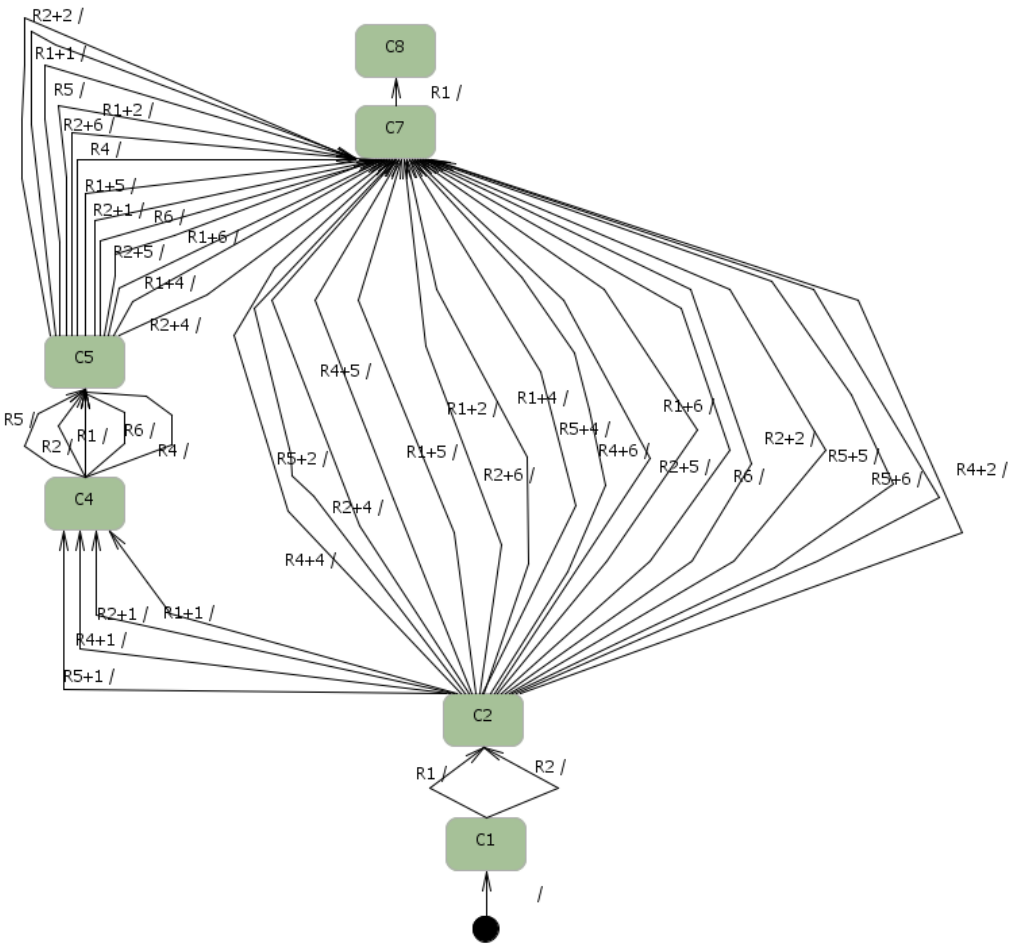


Figura 91 - Espacio de posibilidades tras segundo refactoring

Una vez ejecutados ambos procesos de refactoring se procede a visualizar el espacio de posibilidades, teóricamente, libre de errores. Como rápidamente se puede ver en la figura 91 el nuevo espacio no contiene ninguna posibilidad coloreada de rojo, por lo tanto, no hay posibilidades inválidas. Consecuentemente tampoco hay resoluciones inseguras.

El espacio resultante de este segundo refactoring no sólo debía proporcionar una representación libre de errores sino que su objetivo principal era hacer alcanzables aquellas posibilidades válidas desconectadas tras el primer refactoring. Siguiendo con el ejemplo tomado al principio del apartado, la posibilidad C4 aislada en el primer refactoring, vuelve a estar accesible a través de las nuevas resoluciones creadas.

La figura 92 pretende mostrar resumidamente los diferentes espacios por los que ha pasado el sistema a través del proceso de refactoring.

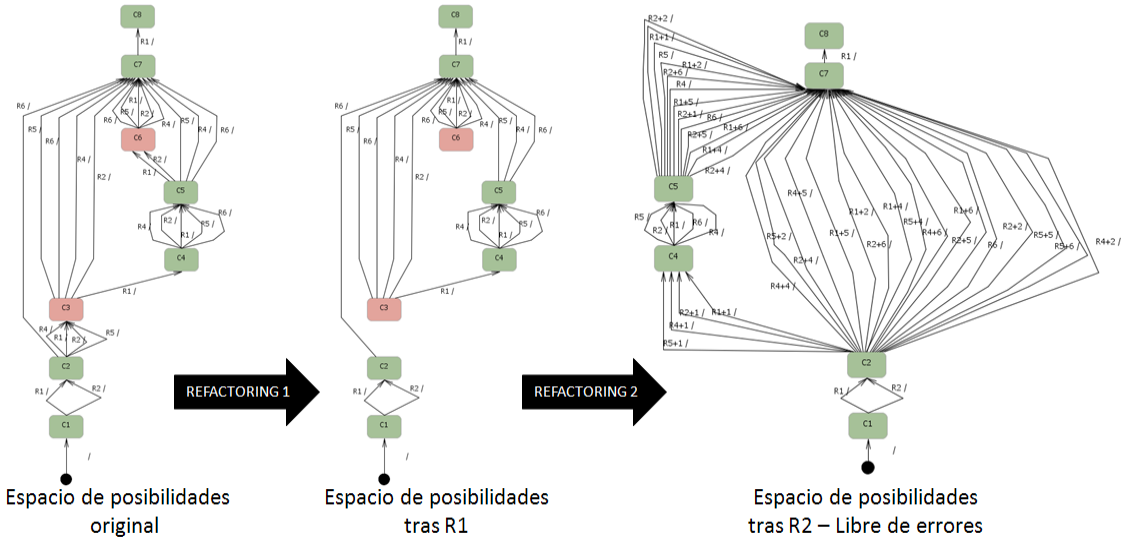


Figura 92 - Proceso completo de refactoring

El espacio inicial contiene posibilidades inválidas y desde ellas se lanzan resoluciones, conocidas como resoluciones inseguras. Al aplicar el primer refactoring sobre este espacio se obtiene una representación donde no aparecen resoluciones hacia estados inválidos, pero quedan posibilidades válidas alcanzadas a través de ellos. Por último, la aplicación del segundo refactoring resuelve el problema anterior y conduce a un espacio final libre de errores.

6.5 CASO 2: ESCALABILIDAD DE LA PROPUESTA

El objetivo de este segundo caso de estudio es demostrar la escalabilidad de la propuesta. Para ello se utiliza el mismo modelo de características pero con un número de resoluciones mayor. De este modo el número de posibilidades generadas se incrementará notablemente.

La descripción de este apartado se centra en el proceso de transformación del espacio de posibilidades a su representación abstracta. De ahí que la funcionalidad básica y los procesos de refactoring sean omitidos.

Como punto de partida se toma el modelo descrito en el apartado 6.2 y este caso se van a utilizar las 8 resoluciones propuestas en la tabla 1. Queda asumido que el modelo ya está cargado y que las resoluciones han sido insertadas en la perspectiva adecuada siguiendo un procedimiento equivalente al del primer caso de estudio.

Los resultados obtenidos tras el análisis de las posibilidades se pueden comprobar en el informe que proporciona la herramienta como se indica en la figura 93:

Metrics		Analysis result	
Model Name:	\src\SmartHome.MFI	Number of possibilities:	36
Numbers of features:	18	Number of valid:	16
Max nº of possibilitys:	4.166090266796	Number of invalid:	20

Figura 93 - Informe tras el análisis

El número de posibilidades ha pasado de las anteriores 8 aplicando 5 resoluciones, a las 36 actuales si se aplican las 8 resoluciones. Además en el informe se aprecia que se han detectado un total de 20 posibilidades inválidas.

El siguiente paso es representar el espacio de posibilidades de forma gráfica mediante la herramienta. Para ello, se recurre a la perspectiva de *Refactorings*, concretamente a *View Possibility Space*. El resultado queda reflejado en la figura 94. Como se aprecia, el espacio de posibilidades ha crecido notablemente al incrementar el número de resoluciones aplicadas. Trabajar con esta representación resulta costoso y poco comprensible.

Este espacio es sólo una muestra de la complejidad que este tipo de representaciones puede alcanzar cuando se aplica sobre casos donde el número de resoluciones se va incrementando.

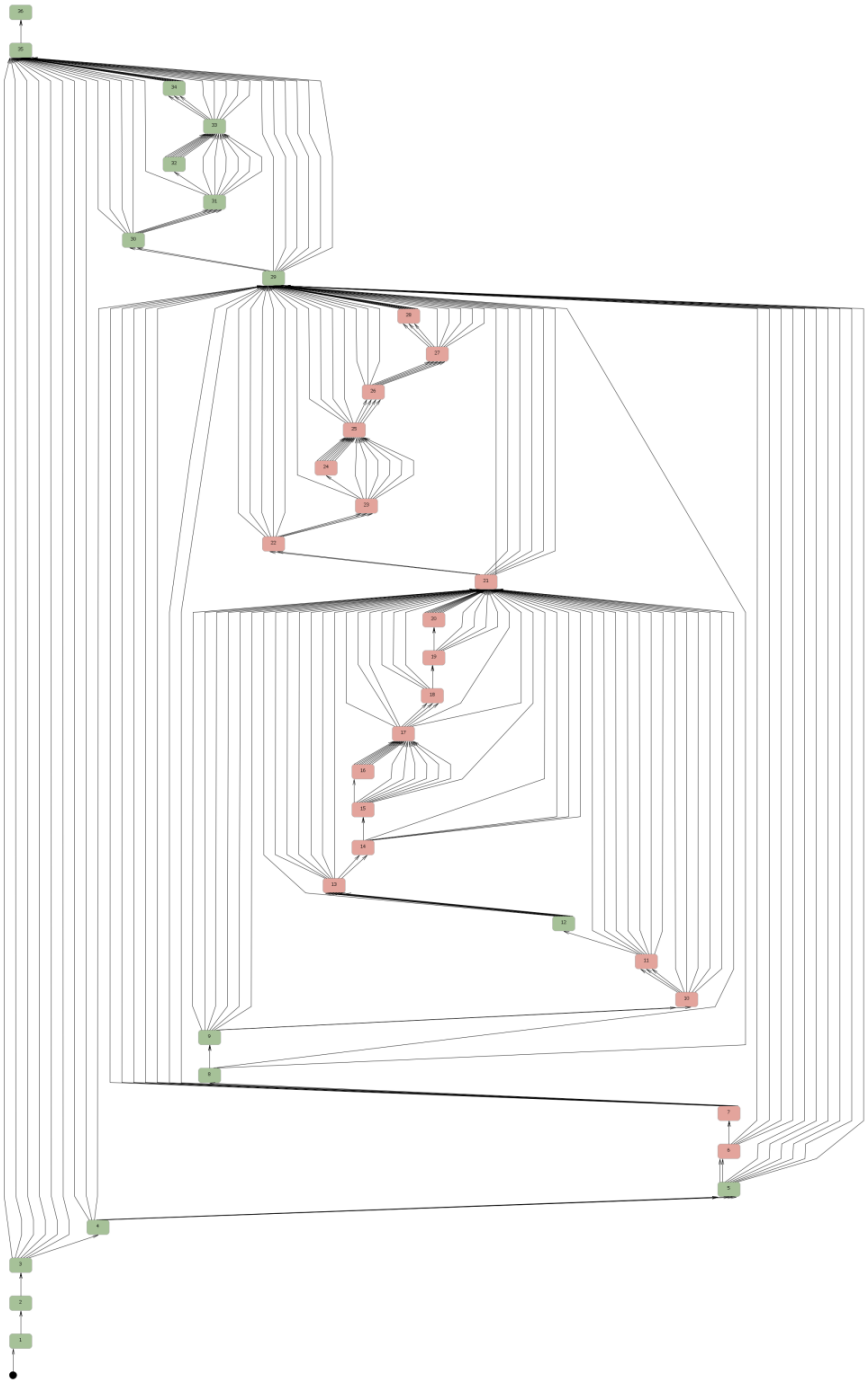


Figura 94 - Espacio de posibilidades: completo

Con el objetivo de mitigar esta complejidad se aplica el proceso de abstracción que dejará el espacio reducido a una representación que abstrae toda la información que resulta superflua. Esto es posible mediante el proceso de abstracción que implementa la herramienta.

La figura 95 representa el espacio de posibilidades anterior en su versión abstracta:

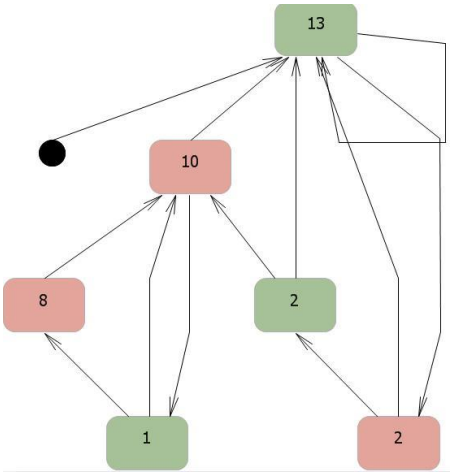


Figura 95 - Espacio de posibilidades abstracto: completo

Llama la atención como el espacio de posibilidades del caso completo en su versión abstracta ha sido simplificado. Esta representación resulta más fácil de manejar y permite analizar con más facilidad la información importante.

Es importante destacar que el proceso de abstracción no ha suprimido ninguna de las posibilidades del sistema, se mantienen las 36 (13+10+8+2+2+1), sino que se han agrupado de acuerdo a las propiedades que compartía.

7. CONCLUSIONES

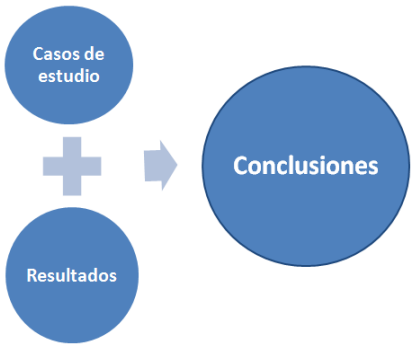


Figura 96 - Alcance del capítulo 7

A lo largo de este documento se han expuesto una serie de técnicas y métodos para la gestión de sistemas software en entornos altamente cambiantes.

El contexto del trabajo son entornos software formados por una serie de dispositivos que ofrecen sus servicios a unos usuarios. Estos entornos se caracterizan por la variedad de usuarios que intervienen y la necesidad de cubrir dinámicamente sus necesidades. Su funcionamiento dinámico implica que el sistema es capaz de adaptar su comportamiento sin la intervención de los usuarios.

Concretamente este trabajo se ha centrado en la detección y resolución de aquellas situaciones que conducían la configuración del sistema a un estado anómalo. Estas situaciones llevaban al sistema a un estado incoherente lo cual invalidaba la configuración. Para resolver estas situaciones se han estudiado las acciones que conducían hasta ellas y se han implementando unas técnicas que impiden alcanzarlas. Cabe destacar que la aplicación de estas técnicas no supone ningún tipo de pérdida de funcionalidad para el sistema.

Para alcanzar este propósito se han aprovechado las capacidades que ofrecen las técnicas de líneas de producto, el desarrollo dirigido por modelos y las bases de la computación autónoma. Haciendo uso de las ideas principales de estas metodologías se ha desarrollado una técnica que ha permitido que todas las configuraciones del sistema alcanzables sean válidas. Es decir, que cualquier cambio siempre deje al sistema en un estado coherente.

Con la intención de dar credibilidad y demostrar los beneficios de la propuesta presentada se han expuesto ejemplos completos de su aplicación en contextos reales. Concretamente dentro del marco de una Smart Home. Mediante el primer caso de estudio se ha demostrado con detalle el proceso de gestión llevado a cabo sobre las configuraciones inválidas. El segundo caso ha servido para comprobar la validez de la propuesta en términos de escalabilidad. Los resultados documentados prueban el éxito de aplicar esta propuesta.

Este trabajo también proporciona la implementación de una herramienta de soporte. La implementación de las técnicas sobre esta herramienta permite desarrollar, administrar y probar diferentes configuraciones de un problema antes de llevar a cabo su desarrollo en un entorno físico.

En definitiva, el objetivo de este trabajo en cuanto a la gestión y resolución eficiente de configuraciones inválidas dentro de la evolución dinámica de un sistema ha resultado satisfactorio. A pesar de ello, se debe seguir trabajando en este ámbito ya que son muchos los nuevos retos que la computación autónoma y los futuros dispositivos van a presentar.

8. TRABAJOS FUTUROS

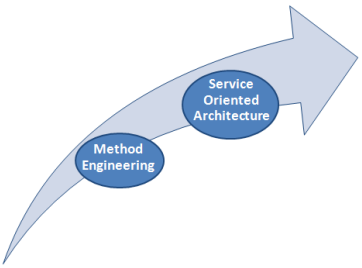


Figura 97 - Alcance del capítulo 8

En este trabajo la metodología y las técnicas propuestas se han aplicado en el ámbito de una Smart Home pero esto no significa que sus capacidades queden limitadas a este contexto. Además de su aplicación sobre dispositivos móviles o para dar soporte a sistemas de decisión es posible aprovechar su potencial en otros dominios, como por ejemplo, en una arquitectura orientada a servicios o en Method Engineering.

- La visión de una **arquitectura orientada a servicios**, también llamada SOA (Service Oriented Architecture) proporciona un ecosistema de servicios donde hay diferentes proveedores de los servicios ofertados para diferentes niveles de calidad y precios.

  Además es un ecosistema dinámico donde la oferta de servicios aparece y desaparece. Desde el punto de vista de los clientes, esto significa que hacen una selección dinámica de los servicios y se unen a los proveedores correspondientes. Desde el punto de vista de los proveedores, éstos proporcionan una atractiva oferta de servicios y sirven a un conjunto de clientes con necesidades diferentes.

  El rol que puede jugar este trabajo puede ser significativo para la implementación de propiedades auto-gestionables con el objetivo de administrar los acuerdos a nivel de servicio y para intervenir en la negociación entre los clientes y los proveedores en entornos SOA.

- **Method Engineering** es una disciplina de la ingeniería para diseñar, construir y adaptar métodos, técnicas y herramientas para el desarrollo de sistemas de información.

  No obstante, se ha centrado en la derivación eficiente de un método personalizable para satisfacer los requisitos de un proyecto particular, que una vez construido, permaneciese estático a lo largo de su ciclo de vida.

  Las ideas presentadas en este trabajo pueden permitir a esta metodología hacer frente también a problemas dinámicos, como la fluctuación de recursos humanos o la reestructuración de tareas para llevar a cabo métodos ingenieriles dinámicos.

9. BIBLIOGRAFÍA

[1] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37 (7):56–64, July 2004.

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

[3] 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.

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

[5] J. P. Sousa and D. Garlan. Aura: An architectural framework for user mobility in ubiquitous computing environments. En *In Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*, pages 29–43. Kluwer Academic Publishers, 2002.

[6] V. Bellotti and K. Edwards. Intelligibility and accountability: Human considerations in context-aware systems. *Human-Computer Interaction*,16:193–212, 2001.

[7] W. Sitou M. Fahrmair and B. Spanfelner. Unwanted behavior and its impact on adaptive systems in ubiquitous computing. *ABIS 2006: 14th Workshop on Adaptivity and User Modeling in Interactive Systems*, October 2006.

[8] P. Horn. Autonomic computing: Ibm's perspective on the state of information technology, 2001.

[9] S. Bhola, M. Astley, R. Saccone, and M. Ward. Utility-aware resource allocation in an event processing system. En *ICAC'06: Proceedings of the 2006 IEEE International Conference on Autonomic Computing*, pages 55–64, Washington, DC, USA, 2006.IEEE Computer Society.

[10] T. Zenmyo, H. Yoshida, and T. Kimura. A self-healing technique based on encapsulated operation knowledge. En *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 25–32, June 2006.

[11] M. L. Littman, N. Ravi, E. Fenson, and R. Howard. Reinforcement learning for autonomic network repair. En *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 284–285, May 2004.

[12] S. Kent. Model driven engineering. En *Proceedings of the Third International Conference Integrated Formal Methods* (IFM'2002), 2002.

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

[14] Carlos Cetina, Joan Fons and Vicente Pelechano. Applying software product lines to build autonomic pervasive systems. *12th International Software Product Lines Conference (SPLC)*. Limerick, Ireland. 2008

[15] Javier Muñoz, Vicente Pelechano, Carlos Cetina. Software Engineering for Pervasive Systems. Applying Models, Frameworks and Transformations. IEEE International Conference on Pervasive Services (ICPS). Istanbul,Turkey. 2007

[16] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *Software, IEEE*, 15(6):37–45, Nov/Dec 1998.

[17] G. Blair, N. Bencomo, and France. R.Models@run.time. *Computer*, pages 22–27, October 2009.

[18] N. Bencomo, G. Blair, and R. France. 3rd workshop on models@run.time at models 2008 (proceedings).Technical Report COMP-005-2008 Lancaster University.

[19] Carlos Cetina, Pau Giner, Joan Fons, Vicente Pelechano. Autonomic Computing through Reuse of Variability Models at Runtime: The Case of Smart Homes. IEEE Computer. vol. 42, no. 10, pp. 37-43, Oct. 2009.

[20] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura and A. Jiménez. Software Product Line Conference Tool Demonstrations (SPLC 08 Tools Demos): FAMA Framework. 2008

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

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

[23] J. O. Kephart and D. M. Chess.The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[24] 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.

[25] L. D. Paulson. Computer system, heal thyself. *Computer*, 35(8):20–22, 2002.

[26] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[27] A. Agrawal, T. Levendovszky, J. Sprinkle, F. Shi, and G. Karsai. Generative programming via graph transformations in the model-driven architecture. En, *In OOPSLA 2002 Workshop in Generative Techniques in the context of Model Driven Architecture*, 2002.

[28] S. J. Mellor and A. N. Clark and T. Futagami. Guest Editors' Introduction: Model-Driven Development. *IEEE Software*, 20(5):14–18, 2003.

[29] A. Sernadas, C. Sernadas,and H. D. Ehrich. Object-oriented specification of databases: Analgebraic approach. In VLDB'87:*Proceedings of the 13th International Conference on Very Large Data Bases*, pages 107–116, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[30] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL: a language for object-oriented specification of information systems. *ACM Trans. Inf. Syst.*, 14(2):175–211, 1996.

[31] M. Rohs and J. Bohn. Entry points into a smart campus environment "overview of the ethoc system. En ICDCSW'03: *Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 260, Washington, DC, USA, 2003. IEEE Computer Society.

[32] Object Management Group. Unified Modeling Language: Superstructure version 2.1.1. OMG Specification, February 2007.

[33] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003

[34] I. Kurtev. *Adaptability of Model Transformations*. phdthesis, IPA, 2005. ISBN 90-365-2184-X.

[35] J. Bézivin, N. Farcet, J. Jézéquel, B. Langlois, and D. Pollet. Reflective model driven engineering. Pages 175–189.Springer, 2003.

[36] J. Bézivin. In search of a basic principle for model driven engineering. *UP-GRADE*, 1:15–24, 2004.

[37] D. C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2): 25–31, 2006.

[38] J. Miller and J. Mukerji. Mda guide version 1.0.1, 2003. Letzte Änderung am 12. Jun. 2003, besucht am15. Mai 2008

[39] M. Douglas McIlroy. Mass-produced software components. In J. M. Buxton, Peter Naur, and Brian Randell, editors, *Software Engineering Concepts and Techniques (1968 NATO Conference of Software Engineering)*, pages 88–98. NATO Science Committee, October 1968.

[40] D. L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, SE-2(1):1–9, March 1976.

[41] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[42] F. van der Linden. *Lecture Notes in Computer Science*, volume 2290. Springer, Bilbao, Spain, October 3-5, 2001 2002.

[43] P. Donohoe. Number ISBN 0-7923-7940-3. Denver, Colorado, USA, August 28-31

[44] D. Batory, J. Neal Sarvela, and A. Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30:2004, 2003.

[45] L. Northrop. SEI's Software Product Line Tenets. *IEEE Software*, 19(4):32–40, July/August 2002.

[46] G. Chastek and J. D. McGregor. Guidelines for developing a product line production plan.Technical report, CMU/SEI, June 2002.

[47] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press, New York, NY, USA, 2000.

[48] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005

[49] P. Schobbens, J. C. Trigaux, P. Heymans and Y. Bontemps. Generic semantics of feature diagrams. *Comput. Netw.*, 51(2):456–479, 2007

[50] R. C. Linger, B. I. Witt, and H. D. Mills. Structured Programming; *Theory and Practice the Systems Programming Series*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[51] H. A. Schmid. Creating applications from components: A manufacturing framework design. IEEE *Softw.*, 13(6):67–75, 1996.

[52] K. Czarnecki. Overview of generative software development. In *Unconventional Programming Paradigms, International Workshop UPP*, pages 326–341, 2004.

[53] J. Greenfield, K. Short, S. Cook, and S. Kent. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools.Wiley, August 2004.

[54] D. Schmidt, A. Nechypurenko, and E. Wuchner.Workshop mdd for software product-lines: Factor fiction. International Conference on Model Driven Engineering Languages and Systems, 2005.

[55] G. Botterweck, I. Groher, A. Polzer,C. Schwanninger, S. Thiel, and M. Voelter. International workshop on model-driven approaches in software product line. 13th International Software Product Line Conference, 2009.

[56] S. Hallsteinsen, M. Hinchey, Sooyong Park, and K. Schmid. Dynamic software product lines. *Computer*, 41(4):93–95, April 2008.

[57] P. Schobbens, J. C. Trigaux P. Heymans, and Y. Bontemps. Generic semantics of feature diagrams. *Comput. Netw.* 51(2):456–479, 2007.

[58] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.

[59] Eclipse Modeling Framework (EMF) – Model Query website. http://www.eclipse.org/modeling/emf/?project=query

[60] User Guide FAMA. Framework for automated analyses of feature models.

[61] C. Cetina, P. Giner, J. Fons, and V. Pelechano. Using feature models for developing self-configuring smart homes. Fifth International Conference on Autonomic and Autonomous Systems (ICAS2009), April 2009.