UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Proyecto Final de Carrera

Configuración de casas domoticas con la tecnología RFID

Diciembre de 2009, Valencia

Alfons Gómez Ferragud
algofer1@fiv.upv.es

Dirigido por: *Vicente Pelechano*
pele@dsic.upv.es

Carlos Cetina
cceina@dsic.upv.es

Índice

Índice de figuras

1. Introducción

Se entiende por domótica al conjunto de sistemas capaces de automatizar una vivienda, aportando servicios de gestión energética, seguridad, bienestar y comunicación, y que están integrados por medio de redes de comunicación, y cuyo control goza de cierta ubicuidad, desde dentro y fuera del hogar.

Hasta ahora la domótica se entendía como un sistema centralizado donde el usuario podía controlar de forma centralizada algunos servicios del hogar.

Pero cada vez más, los sistemas domóticos tienen que reconfigurarse automáticamente en respuesta a condiciones externas como: costumbres de los usuarios, condiciones meteorológicas, seguridad de los habitantes, instalación de nuevos dispositivos, etc.

Estos sistemas con multitud de servicios diferentes: multimedia, seguridad, iluminación, climatización etc. Y cada servicio con diferentes tecnologías, las cuales pueden generar conflictos de uso unas con otras. Todo esto hace que el diseño de sistemas de control para casas inteligentes pueda resultar muy complejo. Y mas complicado aun, es testear estos sistemas de forma efectiva, para que se pueda garantizar que cuando estos sistemas estén en funcionamiento en un casa inteligente, se reconfiguren correctamente ante cualquier estimulo externo.

Este es el problema que se quiere plantear en el proyecto que presentamos, obtener una herramienta para probar los diseños de estos sistemas reconfigurables [1] en tiempo de diseño. Para así poder depurar y eliminar los errores o deficiencias que puedan tener. De modo que cuando estos sistemas lleguen a instalarse en Smart Homes se tendrá la certeza de que serán robustos y fiables en cualquier situación.

Esto tipo de sistemas de control que en el futuro se van a usar en el hogar tendrán que ser capaces de reconfigurarse por si solos para adaptarse a diferentes situaciones.

Por ejemplo cuando la casa inteligente este habitada su prioridad deberá ser el confort de sus habitantes. El dueño de la casa podría configurar el sistema para que cuando llegue a casa, se encienda el servicio de hilo musical conectado al sistema de música mp3, se conecte el sistema de iluminación por presencia y se regule la temperatura de la casa a su gusto. Cuando el último ocupante de la casa la abandona y la casa se queda vacía la casa debería reconfigurar sus componentes para dar toda la prioridad a la seguridad de esta. Conectar la alarma silenciosa conectada a la central de seguridad, activar los sensores de presencia en la casa y en los exteriores.

La herramienta RecoAT tiene por objetivo ayudar a diseñar de forma sencilla sistemas que permitan a las casa inteligentes reconfigurarse por si mismas para adaptarse a diferentes contextos.

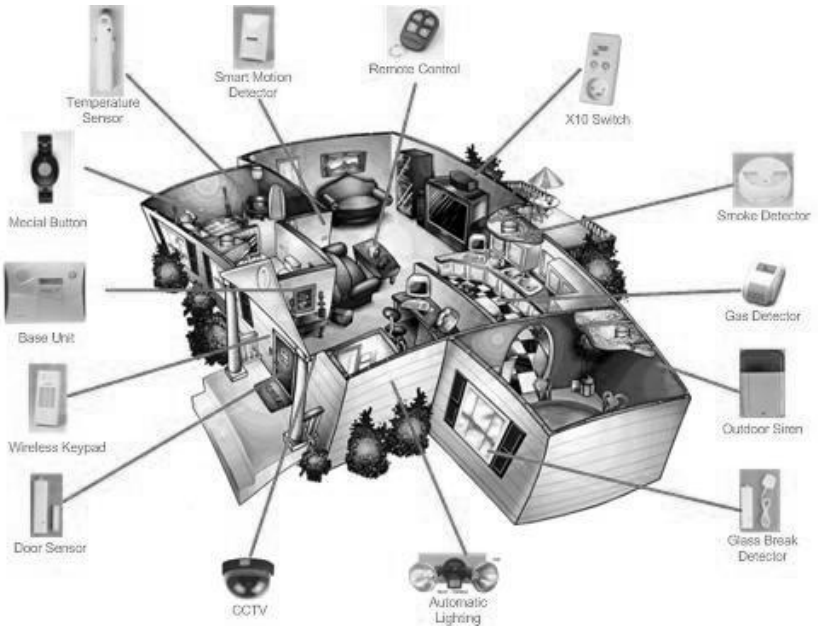


Figura 1. Dibujo de Smart Home.

El propósito de este proyecto es llevar la idea de sistema de control crítico y fiable al terreno de los sistemas domóticos. Y para ello necesitamos una aplicación que nos permita ejecutar estos sistemas en tiempo de diseño para poder depurar fallos y corregirlos rápidamente. Así poder conseguir sistemas dómoticos fiables que puedan controlar elementos críticos de los hogares como: seguridad ante intrusos, protección ante fugas de gas o agua, gestión eficiente de la energía, etc.
En los siguientes apartados de esta memoria se va a explicar la aplicación RecoAT, la cual se ha desarrollado, para llevar a cabo el propósito de testar el diseño de sistemas autoconfigurables para Smart Homes.

2. Contexto Tecnológico

En este capitulo se va a tratar de explicar las tecnologías utilizadas durante la realización del proyecto y en especial la requerida para la implementación de la herramienta Recoat.

La aplicación RecoAT se ha implementado con el entorno de desarrollo MOSkitt [7], el cual esta basado en el entorno eclipse.
La aplicación se ha desarrollado como un plugin del entorno de desarrollo MOSkitt (eclipse) [8], del cual hablaremos más adelante. El porque de integrar la aplicación dentro de esta plataforma es muy sencillo, la aplicación necesita de muchos componentes los cuales ya estaban implementados para esta plataforma. Así que se ha podido hacer uso de todos estos componentes sin necesidad de tenerlos que implementar.
Para la implementación del plugin se ha utilizado SWT (Standard Widget Toolkit), una herramienta integrada en eclipse la cual utiliza lenguaje de programación java y funcionalidad extra para construir nuevas herramientas integradas en eclipse.

Uno de estos componentes ya desarrollados para la plataforma eclipse es el analizador de modelos FAMA [4], uno de los componentes mas importantes del proyecto ya que nos ha permitido realizar validaciones de modelos.

Para poder utilizar este validado de modelos, se ha tenido que modelar las SmartHomes a modelos de características, los cuales son unos modelos con unas características especificas que se explicaran con mas detalle en un apartado del capitulo.

2.1 Entorno de desarrollo MOSkitt (Eclipse)

- **MOSkitt**

Modeling Software KIT (MOSKitt) es una herramienta CASE LIBRE, basada en Eclipse que está siendo desarrollada por la Conselleria de Infraestructuras y Transporte (CIT) para dar soporte a la metodología gvMétrica (una adaptación de Métrica III a sus propias necesidades). gvMétrica utiliza técnicas basadas en el lenguaje de modelado UML.
Su arquitectura de plugins la convierte no sólo en una Herramienta CASE sino en toda una **Plataforma de Modelado** en Software Libre para la construcción de este tipo de herramientas.
MOSKitt se desarrolla en el marco del proyecto gvCASE, uno de los proyectos integrados en gvPontis, el proyecto global de la CIT para la migración de todo su entorno tecnológico a Software Libre.

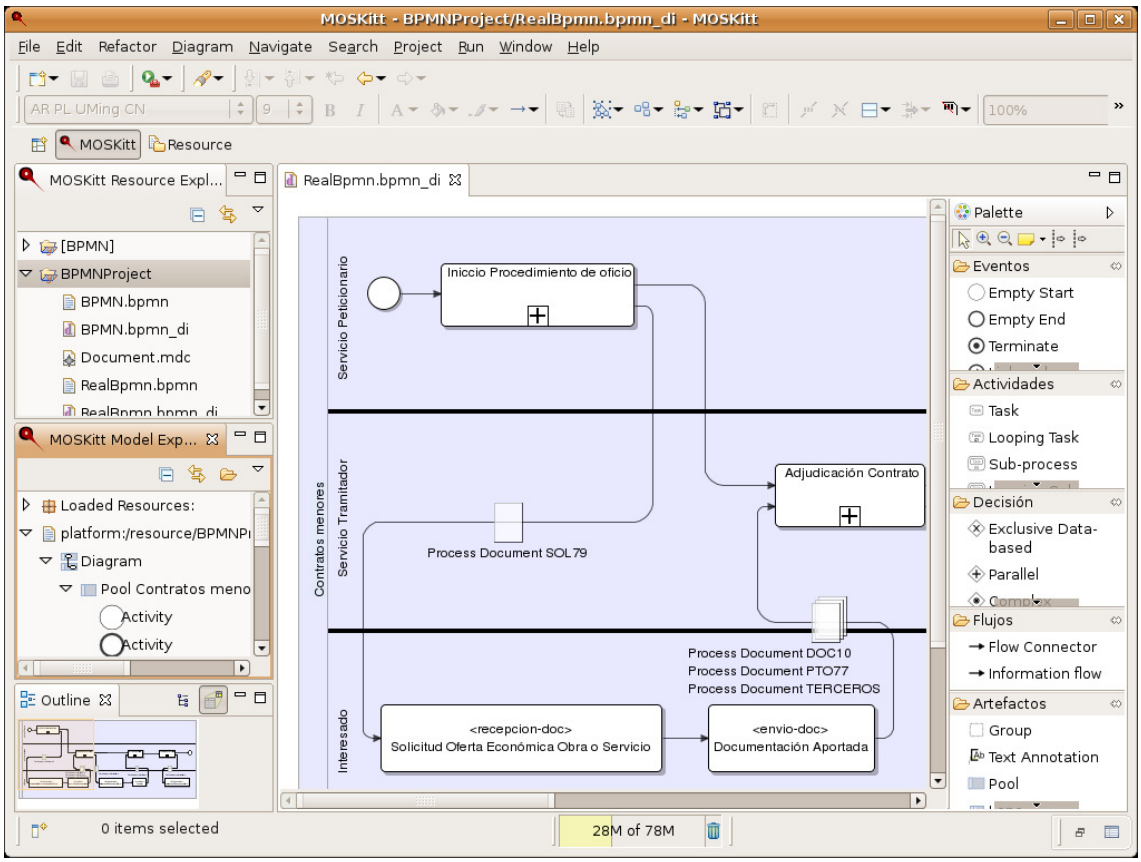


Figura 2. Vista de la herramienta MOSkitt.

- **Eclipse**

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Eclipse es también una comunidad de usuarios, extendiendo constantemente las áreas de aplicación cubiertas. Un ejemplo es el recientemente creado Eclipse Modeling Proyect, cubriendo casi todas las áreas de Model Driven Engineering.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

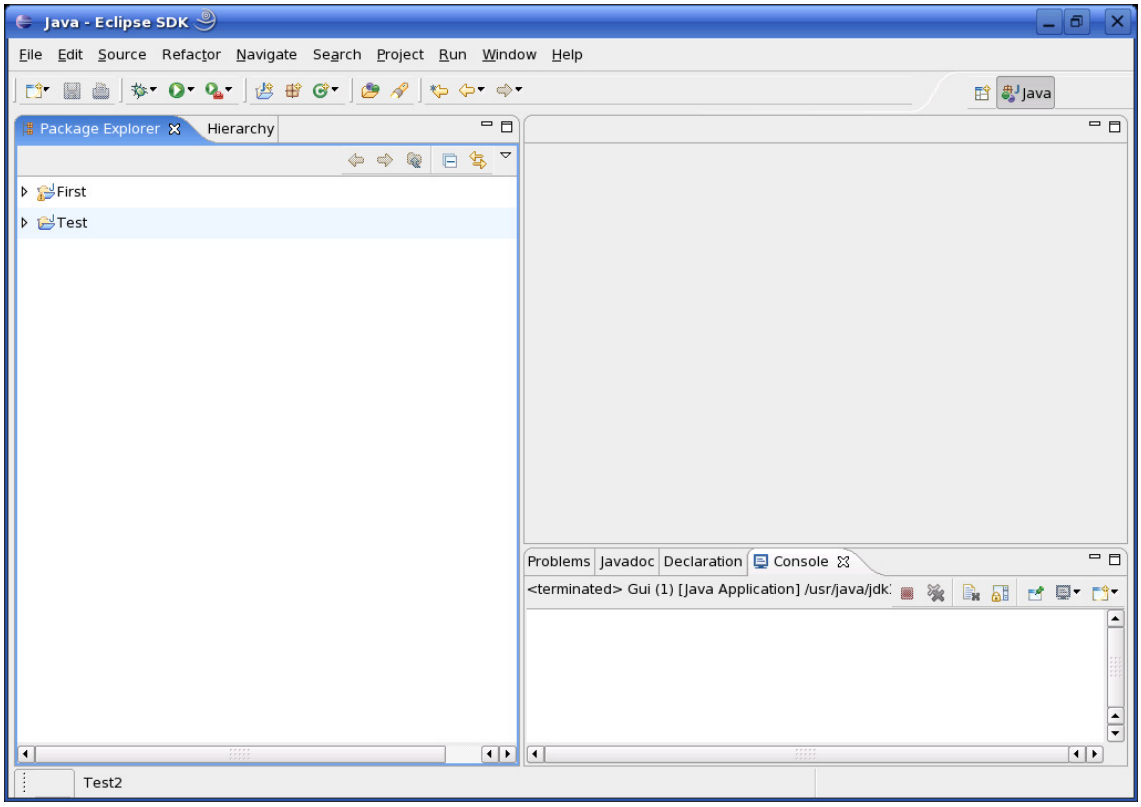


Figura 3. Vista del entorno de desarrollo Eclipse.

2.2 Analizador FAMA

FAMA-FW [5] [6], es un marco para el análisis automatizado de modelos. Cuenta con integración para las representaciones lógicas más comunes y validaciones propuestas en la literatura (BDD, SAT y CSP validadotes implementados).
FAMA es la primera herramienta que integra diferentes validadotes para análisis automático de modelos de características.

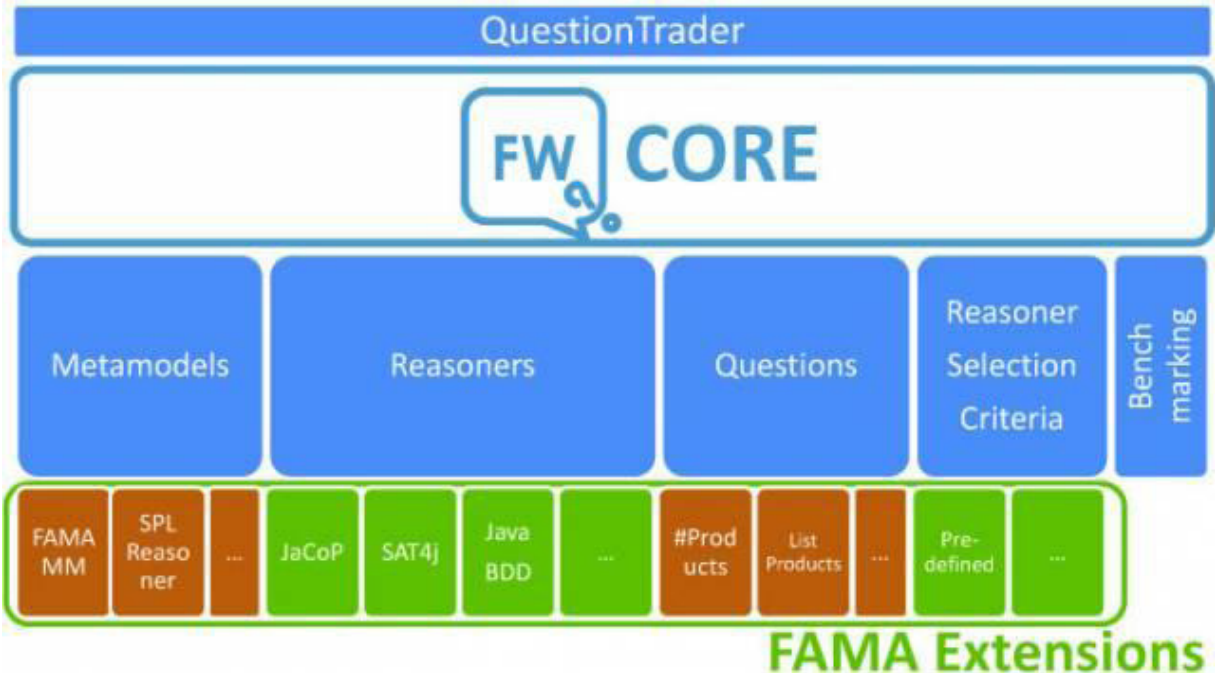


Figura 4. Esquema de los módulos del validador FAMA.

2.3 Modelo características

Un modelo características es una representación compacta de todos los productos de SPL en términos de características [2]. Los modelos son visualmente representados por medio de diagramas de características. Los modelos de características son ampliamente utilizados durante toda la línea de proceso de desarrollo de productos que habitualmente se utilizan como entrada para producir otros bienes, como los documentos, la definición de la arquitectura, o piezas de código.
Una línea de productos de software (SPL) es una familia de programas relacionados. Cuando las unidades de construcción del programa son características, se cuenta con incrementos en la funcionalidad del programa o de desarrollo, cada programa en un SPL es identificado por una única y legal combinación de características, y viceversa.

Los modelos de características se introdujo por primera vez en ***Feature-Oriented Domain Analysis method (FODA)*** por Kang en 1990. Desde entonces, el modelado de características ha sido ampliamente adoptado por la comunidad de línea de productos software y ha un gran numero de extensiones.

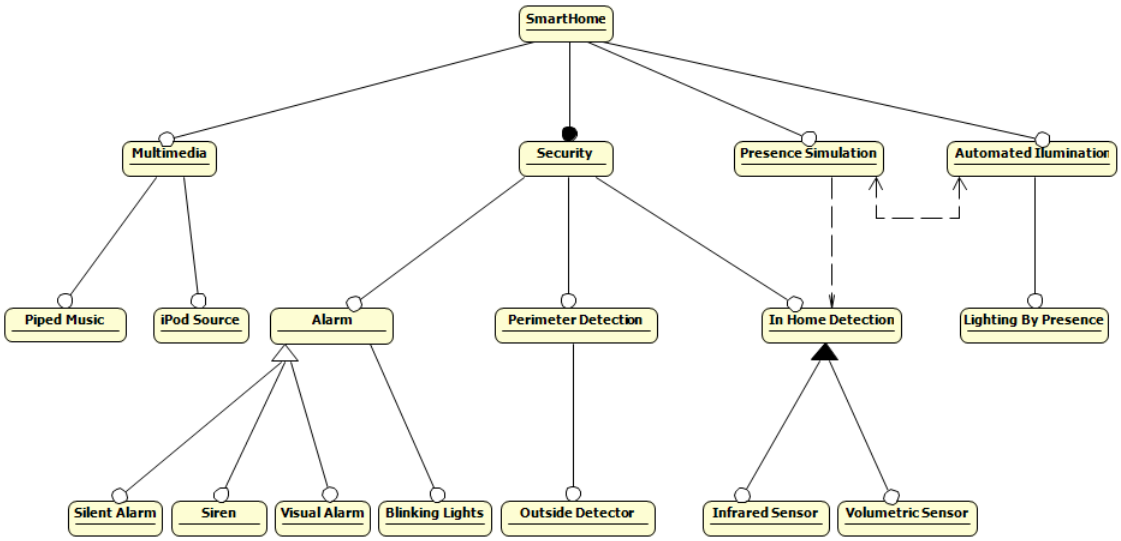


Figura 5. Ejemplo modelo de características.

3. Overview de la propuesta

En este capitulo se va a dar un idea general de todo lo que se desarrollara en el resto proyecto. La estructura que se va a seguir en el capitulo es la siguiente primero vamos a ver el contexto general de la aplicación, donde presentaremos los componentes principales y se ara una breve síntesis. Seguiremos por resumir y concretar el objetivo del proyecto. Y por ultimo detallaremos los pasos mas importantes que se han seguido para la realización del proyecto.

3.1 Contexto general de la aplicación

Para dar una explicación general de lo que se pretende hacer en este proyecto nos vamos a ayudar de la siguiente figura donde de forma muy esquemática tenemos todos los componentes del proyecto.

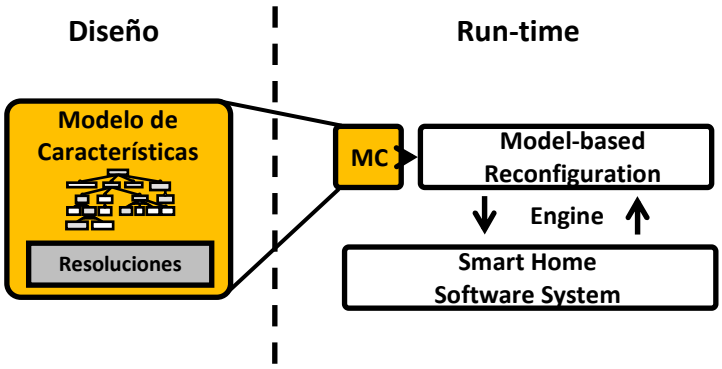


Figura 6. Esquema conceptual de la aplicación.

En la parte de diseño tenemos el modelo de características, lo cual se podía resumir como la representación compacta de los servicios que ofrece la Smart Home.
Y las resoluciones, cada resolución seria como un estado en el que se podría encontrar la casa. Y cada una esta compuesta listas de pares: servicio, estado en el que se encuentra el servicio. Cada resolución esta asociada a una condición de contexto y solo se activara si se cumple esta condición. En este ejemplo se ve más claro.

Resolución [Condición] = {(Servicio, estado)}

Resolución [Casa Vacía] = {(Alarma, conectada), (TV, desconectada)}

En la parte de Run-Time, se parte de los componentes de la parte de diseño y de la configuración actual del modelo, para generar el modelo basado en Reconfiguraciones o el espacio de posibilidades por las que puede pasar el sistema de la Smart Home.
Será necesario un controlador que sea el que según diferentes condiciones seleccione la mejor configuración del modelo basado en Reconfiguraciones. Y por ultimo esta última configuración que seleccione en cada momento el controlador será el sistema que se ejecutara en la Smart Home.

Una vez sabemos todos los componentes que intervienen en el sistema, se va a enumerar cuales serian los pasos mas importantes que se han seguido para la realización del proyecto. Estos pasos se explicaran en mayor detalle en las siguientes subsecciones.

- Primer paso: Validación de configuraciones.

Primero es necesario editar un modelo de características que sintetice los distintos tipos de sensores y actuadores de la Smart Home. En la siguiente figura vemos un ejemplo de lo que podría ser un modelo simplificado.

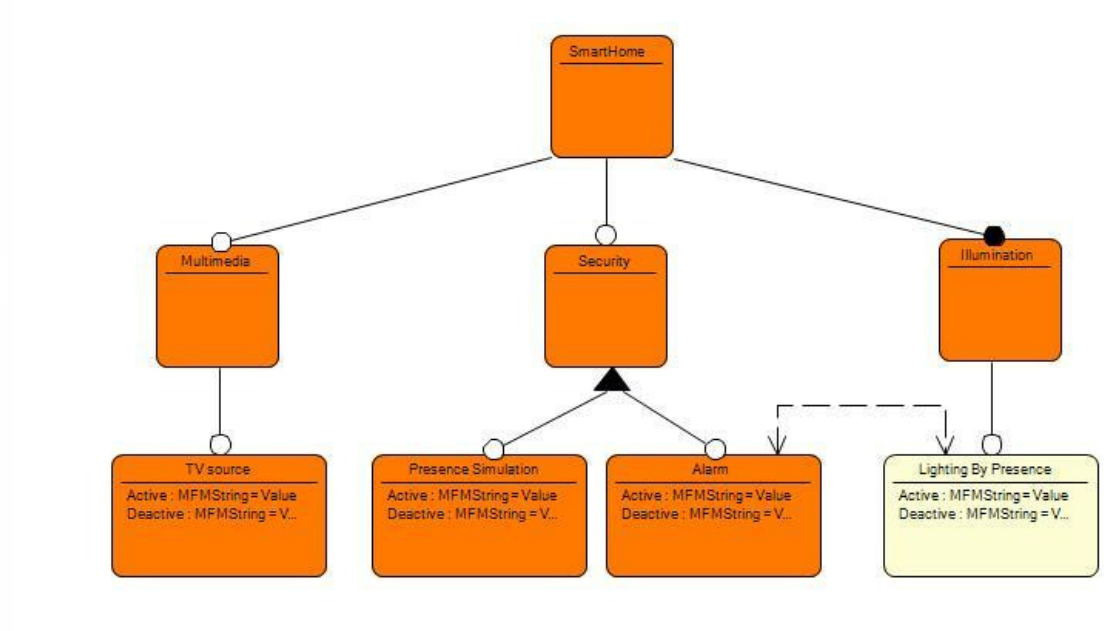


Figura 7. Modelo Características simplificado.

Una vez el modelo esta completado, se edita la configuración actual *CA* del modelo, que consiste en poner a cada característica del modelo uno de sus posibles estados. Vemos a continuación la *CA* del Modelo de características de la figura anterior:

{CA} = {(SmartHome, Activo), (Multimedia, Activo), (Segurity, Activo), (Ilumination, Activo), (TV source, Activo), (PresenceSimulation, Activo), (Alarm, Activo), (Lighting By Present, Inactivo)}

Figura 8. Formula configuración actual.

Antes de seguir adelante se tendría que validar esta *CA*, se valida porque hay que asegurarse que una vez esta configuración este en ejecución sea fiable. Para comprobar que las configuraciones son fiables, hay que comprobar una serie de restricciones que los modelos de características incluyen para que las configuraciones funcionen conforme con lo que se espera de ellas. Por ejemplo ciñéndonos al modelo de características de la figura 2, se podría poner una restricción por la cual si la SmartHome esta activa, siempre tenga que estar activa la característica de seguridad.
Y siguiendo con el ejemplo anterior vemos unas flechas discontinuas que unen las características Alarm y Ligting By Presence, esto es una restricción que impide que estas dos características estén activas a la vez. En el *subapartado 3.5* se verán detalladamente todos los tipos de restricciones posibles.
Para realizar las validaciones de las configuraciones utilizamos un analizador de modelo de características [3] (FAMA)

- Segundo paso: Análisis de las resoluciones.

Seguimos por analizar las resoluciones, para ello se tendrá en cuenta las características de la Smart Home como: los dispositivos instalados, sensores, accionadotes, etc. Y sobre todo las preferencias de sus habitantes en el uso de los servicios de la casa.
Después de finalizar este análisis, las resoluciones se aplicaran una a una a la configuración actual y se validaran con el analizador FAMA.

- Tercer paso: Calculo del espacio de posibilidades.

El siguiente paso a seguir seria ya el cálculo del modelo basado en reconfiguración, o espacio de posibilidades. En resumen este proceso consiste en realizar todas las posibles combinaciones de resoluciones aplicándolas a la CA del modelo. Cada una de estas combinaciones resultantes seria un estado del modelo de reconfiguración.
En el transcurso de este proceso también se calculan las transiciones entre estos estados que son simplemente las condiciones que hacen que se pase de un estado a otro.

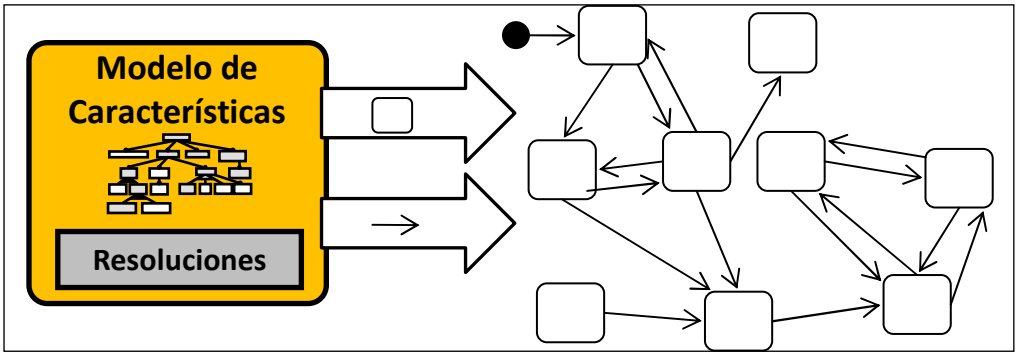


Figura 9. Esquema de paso a espacio posibilidades.

Como se ve en la figura siguiente partiendo del modelo de características y del conjunto de resoluciones, en tiempo de ejecución se calculan los estados (cuadrados) y las transiciones entre estados (flechas). Con el fin de obtener el modelo de reconfiguración.

Se va a explicar detalladamente através de un ejemplo del siguiente espacio de posibilidades, para que resulte más fácil de entender. La posibilidad inicial seria la CA del modelo de características. En función de eventos de contexto como podría ser, la entrada de un habitante en la casa, se avanzara en este espacio de la posibilidad inicial a la posibilidad que corresponda por el evento de contexto sucedido. Si el siguiente evento fuese que se hiciese de noche, pasaríamos a otra posibilidad donde se aplicarían las resoluciones pertinentes. Otro evento de contexto como que se hiciese de día otra vez, haría volver a la posibilidad anterior en dentro del espacio de posibilidades.
Vemos en la siguiente figura este ejemplo simple de espacio de posibilidades que hemos estado comentando anteriormente.

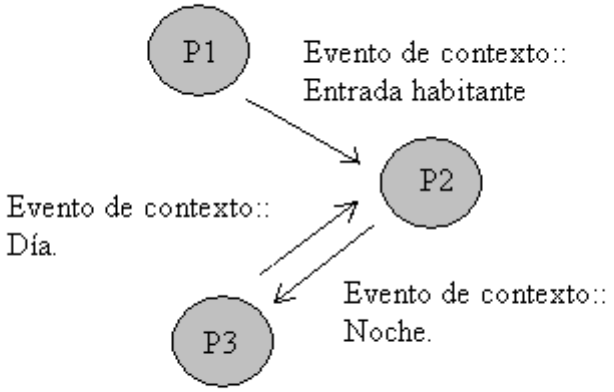


Figura 10. Ejemplo espacio de posibilidades simple.

- Cuarto paso: Validación de las posibilidades.

El último paso antes de obtener el modelo de reconfiguración final seria validar cada posibilidad individualmente por el mismo analizador utilizado anteriormente. Cada posibilidad una vez aplicadas las nuevas resoluciones, se obtendrá su configuración, a la cual el analizador FAMA [5] dará por valida o por inválida.

Siguiendo con el ejemplo de la figura 5 y utilizando el modelo de características de la figura 2, vamos a ver como se podría dar una posibilidad invalida. A continuación tenemos las resoluciones que se aplicarían a cada posibilidad en función de los eventos de contexto.

Resolución [Entrada habitante] = {(Ligting By Presence, conectada), (Alarm, desconectada)}

Resolución [Día] = {(TV Source, conectada)}

Resolución [Noche] = {(Alarm, conectada)}

Al aplicar la resolución ***R[Entrada habitante]*** a la P1 transitaríamos a P2, la cual seguiría siendo una posibilidad valida ya que cumpliría todas las restricciones. Pero al transitar de P2 a P3 con la resolución ***R[Noche]*** incumpliríamos la restricción de exclusión entre *Ligting By Presence* y *Alarm*. Por lo que esta posibilidad seria invalida.

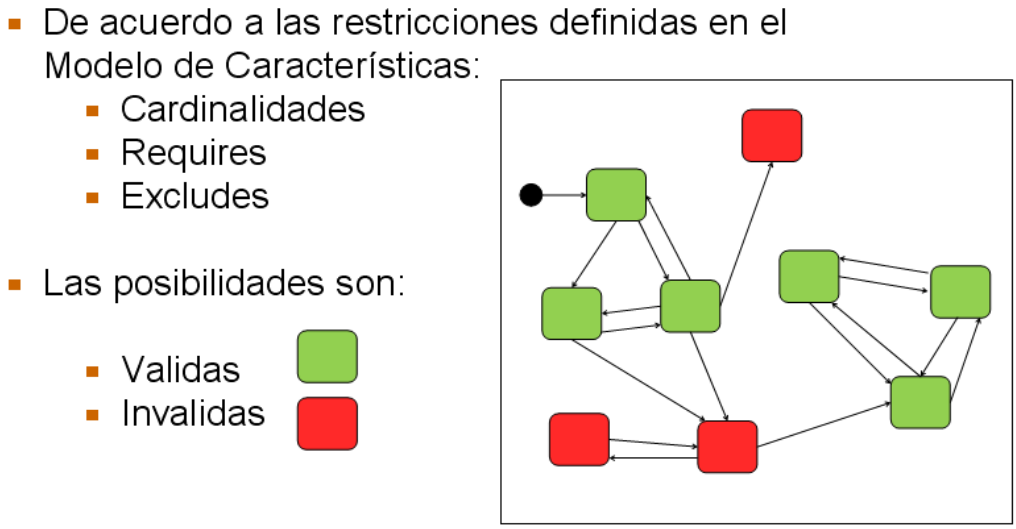


Figura 11. Ejemplo espacio posibilidades validado.

El diagrama que vemos a la derecha de la figura 6 es una representación de cómo quedaría un modelo de reconfiguración, después de que sus estados pasaran por el analizador. Se ve como los estados están coloreados en verde o rojo según si son validos o no y como las flechas actúan de transiciones entre unos estados y otros.

- Quinto paso: Pasar del espacio de posibilidades al espacio de posibilidades abstracto.

Por ultimo se va a explicar como se pasa el modelo de reconfiguración o el espacio de posibilidades a un modelo abstracto. Este nuevo modelo abstracto a nivel de computación no tiene ningún efecto, pero a nivel de entendimiento visual del modelo, puede ser muy útil. En el modelo abstracto simplemente se unen los estados validos y no validos, los cuales se encuentran conectados entre si por transiciones.

- Sexto paso: Uso del espacio de posibilidades abstracto.

El resultado final del nuevo modelo, es un mucho más resumido y claro que el anterior, donde se puede apreciar de forma muy clara cuales son las transiciones (condiciones) que hacen el sistema valido y cuales lo hacen invalido. Lo cual lo hace mucho más cómodo a la hora de trabajar sobre el.

3.2 Objetivo del PFC.

En este subapartado vamos a intentar resumir el objetivo de la aplicación que se esta presentando.
Como podemos observar en la figura 7, después de construir el modelo de características, aplicar una configuración inicial y editar las resoluciones que reflejan los hábitos en la casa inteligente, es posible que el resultado final genere conflictos en las restricciones del modelo. Los cuales son difíciles de identificar en la etapa de diseño y que se pasaran directamente a la etapa de ejecución. Este tipo de errores en tiempo de ejecución pueden ser fatales para el correcto funcionamiento del sistema de la casa inteligente. Además una vez detectados, su solución requiere volver a la etapa de diseño para solucionarlos, lo cual es costoso tanto en tiempo como en dinero.

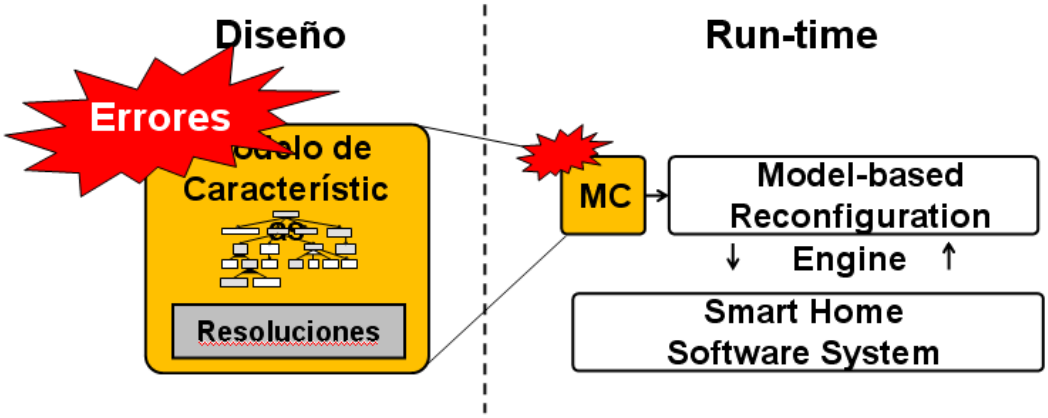


Figura 12. Esquema conceptual con errores.

Para intentar detectar y solucionar estos errores en tiempo de diseño es por lo que se ha pensado esta aplicación. Para ayudar a los diseñadores del sistema a detectar los errores en la fase de diseño. Así al implantar el sistema en un casa inteligente real, tener la certeza que su funcionamiento va a ser el esperado.
En la siguiente figura vemos un esquema de los pasos que habría que seguir para solucionar los posibles errores del sistema en fase de diseño.

Primero dado el modelo de características, la configuración inicial y las resoluciones se obtendría, el espacio de posibilidades. Con este podríamos ya analizar que posibilidades son erróneas y porque lo son. Lo cual haría mucho mas fácil analizar el porque de esos errores y buscar como solucionarlos.
Una vez solucionados los conflictos que generaban errores se tendría un modelo de características equivalente al anterior, pero seguro y fiable.

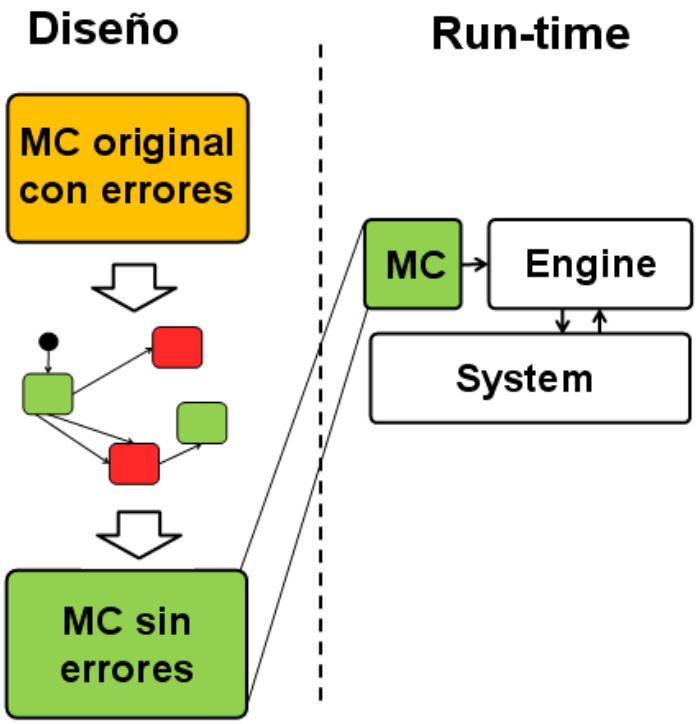


Figura 13. Esquema conceptual solución errores.

3.3 Validación de configuraciones

Las configuraciones se tienen que validar antes de poder ser usadas por el sistema de la Smart Home. Esta validación se tiene que hacer contra el modelo de características, este tipo de modelos de características tiene unas especificaciones. Al editar los modelos de características, podemos restringir ciertas opciones del modelo mediante restricciones.
Estas restricciones nos ayudan a que las configuraciones que se ejecutan en nuestro sistema estén dentro de lo previsto.

En el siguiente apartado vamos a explicar un poco las especificaciones de los modelos de características, centrándonos en las restricciones que se pueden aplicar a los modelos de características.

3.3.1 Restricciones de los modelos de características.

Primero vamos a hablar de cómo se organizan estos modelos de características. Podemos distinguir dos tipos de elementos: las características y las relaciones entre características. Y dentro de las relaciones entre características, dos tipos las relaciones entre padres e hijos y las relaciones entre iguales.

Primero vamos hablar de las relaciones entre padres e hijos.

Figura 14. Ejemplo relación padre-hijo.

El primer concepto a aclarar es que para poder activar la característica hija la característica padre tiene q estar activada.
Dentro de las relaciones padre-hijo simples tenemos dos opciones: la obligatoria y la opcional.

La primera relación de la figura 14 con una bola negra al final conectando con la característica hija, tenemos la relación **obligatoria**.
Esta relación se caracteriza por que si esta activada la relación padre, la relación hija también tiene que estarlo. Sino al validar obtendríamos un modelo inválido.

La relación **opcional,** con el círculo blanco al final, se define porque si esta activada la característica padre, la característica hija puede estar activada o no.

Ahora nos vamos a centrar en las relaciones padre-hijo múltiples, donde tenemos dos tipos la **n-alternativa** y la **n-disyuntiva**.

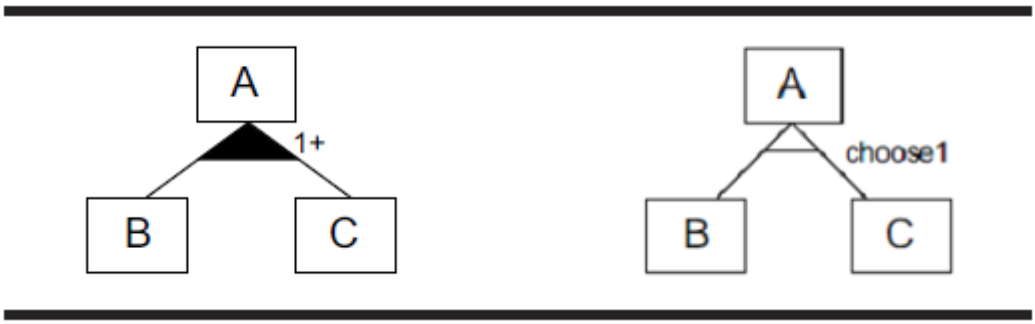


Figura 15. Ejemplo relación padre-hijo múltiple.

En la figura 15 la primera que vemos es la n-disyunción, con un triangulo negro entre las diferentes características hijas. Esta relación se define como que si el padre esta activo, por lo menos una de los hijos tiene que estar activo, además pueden estar activos 2 o todos los hijos.

La otra relación múltiple de la figura anterior seria la n-alternativa, esta se caracteriza por que cuando el padre esta activo, obligatoriamente uno y solo uno de los hijos tiene que estar activo. Si ningún o más de un hijo estuviesen activo, al validar la configuración contra el modelo de características esta fallaría.

Por ultimo vamos a presentar las dos últimas restricciones que nos ayudaran a modelizar sistemas demóticos del hogar. Estas dos restricciones no se utilizan entre pares e hijos como las anteriores se pueden utilizar entre dos características cualesquiera del modelo.

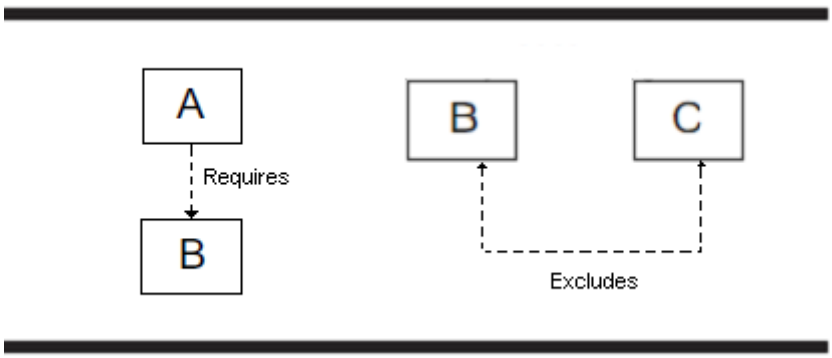


Figura 16. Ejemplo *Requires / Excludes*.

Primero observamos la restricción de requiere, la cual se caracteriza por que cuando A esta activa necesita de B para que la configuración mantenga la validez respecto al modelo.
La siguiente restricción seria la exclusividad, la cual se define como que B y C no pueden estar activas a la vez. Es decir que cuando una esta activa la otra no lo puede estar para mantener la validez.

3.3.2 Ejemplo de restricciones en un modelo de características.

En esta sección vamos a ver varios ejemplos de cómo una configuración de una Smart Home puede incumplir algunas restricciones de su correspondiente modelo de características.

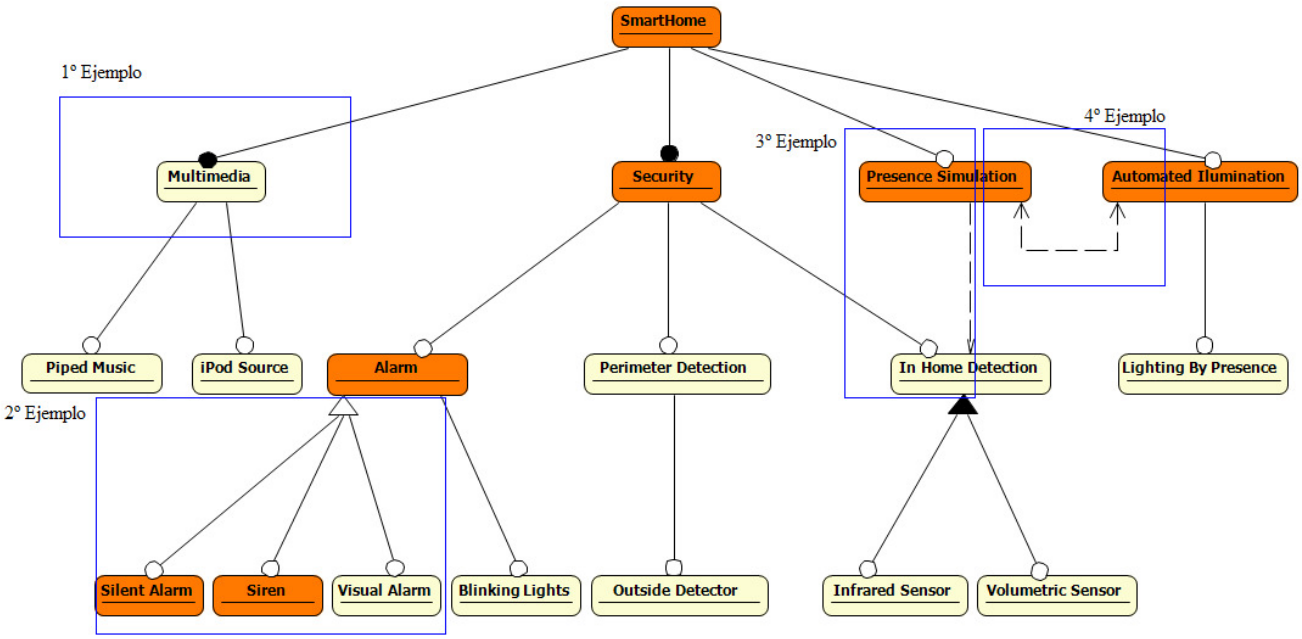


Figura 17. Modelo de características con restricciones.

A continuación se desglosa en 4 ejemplos algunas restricciones que no se cumplen y que por lo tanto harían esta configuración invalida.

Ejemplo 1:

En este ejemplo tenemos una relación padre-hijo obligatoria. Se ve como la característica *SmartHome* esta activa por lo cual la característica hijo *Multimedia* estaría obligada a estar activa, y como se ve en esta inactiva por lo cual incumple la restricción del modelo.

Ejemplo 2:

En este siguiente ejemplo tenemos una relación múltiple n-alternativa. Se tiene la característica padre *Alarm* activada y para que se cumpliera la restricción tendría que estar activa una y solo una de las características hijas. Y como se puede observar en la figura, dos de las características hijas están activadas, así que se invalidaría la configuración.

Ejemplo 3:

En este segundo ejemplo tenemos una relación de inclusión entre *Presence Simulation* y *In Home Detection.* Dado que la característica *Presence Simulation* esta activado, para que se cumpliera la restricción la característica *In Home Detection* también debería de estar activada. Como se puede observar no lo esta, por lo cual se invalidaría el modelo.

Ejemplo 4:

En el último ejemplo tenemos una relación de exclusión entre *Presence Simulation* y *Automated Ilumination.* Por lo que si las dos características se activasen a la vez esta restricción se incumpliría. Como vemos este es el caso de cuarto ejemplo, así que modelo invalido.

3.4 Análisis de las resoluciones

En esta sección vamos a explicar como se consiguen las resoluciones para que el sistema de la casa domótica sea fiable y conforme a las preferencias de cada usuario.
Para ello vamos a analizar la obtención de resoluciones desde tres puntos de vista diferentes: situaciones cotidianas en la casa, situaciones extraordinarias y por ultimo preferencias de usuarios.
Para la realización de este análisis nos ceñiremos al modelo de características de la figura 18, lo cual hará más fácil el seguimiento de los ejemplos.

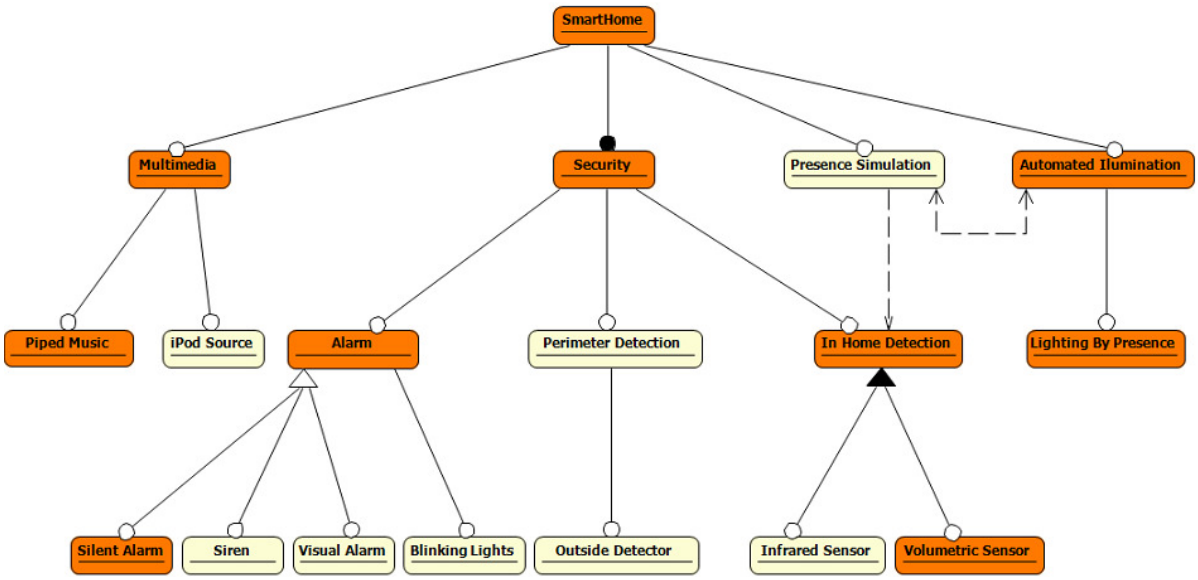


Figura 18. Modelo características (resoluciones).

3.4.1 Situaciones cotidianas

A Continuación analizaremos algunas situaciones cotidianas de la vida en una Smart Home, y razonaremos sobre que resoluciones se aplicarían para minimizar el gasto energético, obtener la mejor seguridad en cada momento, facilitar la vida a sus habitantes.

- La casa por el día:

> Durante las horas entre que sale y se pone el sol, seria poco eficiente tener funcionando la activación de luz por presencia, ya que la mayoría de las estancias de la casa se iluminan por luz solar. La condición que activase esta resolución podría ser un simple sensor fotoeléctrico

Resolución [casa de día] = {(Lightning by presencie, desconectada), (Automated Ilumination, desconectada)}

- La casa de noche:

En el momento que se pone el sol, se pondría en funcionamiento la activación automática de luz por presencia, este sistema es muy eficiente en cuanto a uso de energía, dado que se solo se iluminan las zonas donde están los habitantes de la casa. También es más cómodo para sus habitantes ya que no tiene que preocuparse de ir encendiendo y apagando las luces. La condición que activase esta resolución seria el mismo sensor fotoeléctrico usado en la sección anterior.

Resolución [casa de día] = {(Lightning by presencie, conectada), (Automated Ilumination, conectada)}

- La casa inhabitada:

Serian durante las horas en las los habitantes de la casa saliesen a trabajar, ha realizar sus actividades cotidianas, acudir a sus centros de estudio, etc. Este periodo de tiempo duraría solo unas horas así que se activaría la seguridad a un nivel medio. También se podría desactivar el sistema multimedia ya que nadie lo utilizaría y podría estar consumiendo energía.

Resolución [casa inhabitada] = {(Multimedia, desconectada), (Segurity, conectada), (Alarm, conectada), (Siren, conectada), (In Home Detection, conectada), (Volumetric Sensor, conectada)}

- La casa habitada:

Mientras algún habitante de la casa, este en ella. Se podría desactivar el sistema de alarma, conectar el sistema multimedia

Resolución [casa habitada] = {(Multimedia, conectada), (Segurity, desconectada)}

- La casa inhabitada durante un tiempo prolongado:

Esta situación se daría en periodos prolongados en que la casa va ha quedar vacía, como fines de semana o periodos vacacionales. Durante este periodo lo primordial es la seguridad por lo el nivel de seguridad debería de ser alto.

Resolución [casa inhabitada2] = {(Segurity, conectada), (Alarm, conectada), (Siren, conectada), (In Home Detection, conectada), (Volumetric Sensor, conectada), (Perimeter Detection, conectada), (Outside detector, conectada)}

3.4.2 Situaciones extraordinarias

A Continuación analizaremos algunas situaciones inusuales en una smart home, como desconexión o mal funcionamiento de los dispositivos de seguridad, detección de intrusos, etc.

- Desconexión o mal funcionamiento de la sirena.

En una configuración del sistema en que la sirena esta conectada, esta dejara de funcionar o se desconectar de forma inesperada, se activaría estar resolución para evitar dejar el sistema de alarma inefectivo.

Resolución [sirena desconectada] = {(Visual Alarm, conectada)}

- Desconexión o mal funcionamiento del sensor volumétrico.

Estando activada una configuración del sistema en que el sensor volumétrico estuviera activo, si este dejase de funcionar se aplicaría esta resolución para que el sistema siguiera funcionando, en la medida de lo posible

Resolución [sirena desconectada] = {(Infrared Sensor, conectada)}

- Detección de intrusos

Si los sensores de volumétricos o exteriores detectaran la presencia de intrusos, el sistema tendría que activar una resolución que asustara a estos y evitara la posibilidad de que se realizara algún robo o desperfecto en la casa.

Resolución [sirena desconectada] = {(Blinking Lights, conectada), (Siren, conectada)}

3.4.3 Preferencias de los usuarios

En función de que personas estén en la casa se aplicaran unas resoluciones, en función de sus gustos y preferencias.

- Preferencias Pedro.

Cuando Pedro llega a casa le gusta que se encienda el hilo musical con su emisora de radio favorita y que se desconecte la iluminación automática por presencia.

Resolución [Pedro] = {(Piped Music, conectada), (Automated Ilumination, conectada)}

- Preferencias Antonio.

A Antonio le gusta llegar a casa y que el ipod suene en su habitación.

Resolución [Pedro] = {(IPod Source, conectada)}

3.4.4 Ejemplo aplicación de resoluciones

En este apartado vamos a ver como quedaría la configuración actual aplicando una serie de las resoluciones generadas en los apartados anteriores a nuestro modelo de características.

Configuración Actual del modelo de características + **Resolución [casa inhabitada]** + **Resolución [sirena desconectada]**

La configuración final que resultaría en el sistema es la que vemos en el siguiente grafico

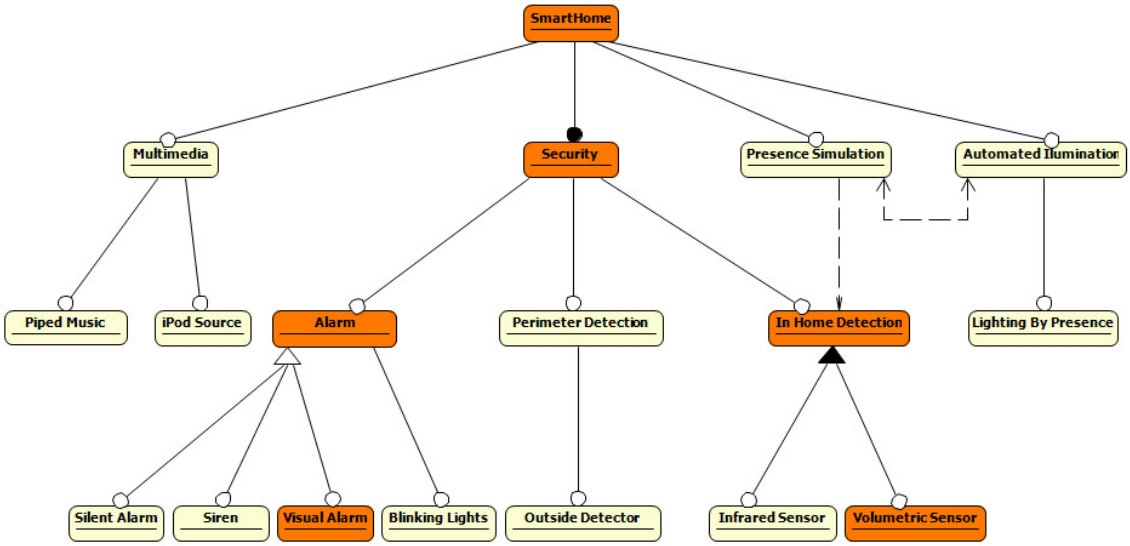


Figura 19. MC Configuración final.

3.5 Calculo del espacio de posibilidades.

3.5.1 Calculo de las posibilidades

Para la realizar el cálculo del espacio de posibilidades es necesaria la configuración actual (CA) del modelo de características de la casa inteligente y la lista de las resoluciones (R*x*).

Una vez tenemos estos dos componentes en el sistema, se puede comenzar con el proceso que nos llevara a la obtención del espacio de posibilidades. Para ello seguiremos estos pasos.

Primero se necesita una posibilidad origen o posibilidad 1. Esta se obtendrá al aplicar la configuración actual a la posibilidad origen.

El siguiente paso a seguir es, partiendo de la posibilidad inicial ir aplicando de una en una las resoluciones de la lista obtenida anteriormente. Cada resolución nueva que aplicamos es una transición a una nueva posibilidad y un nuevo camino en diagrama de posibilidades.

Seguiremos comprobando 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 comprobaran todos los pares (Servicio, estado) resultantes de la unión CC 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 un uno de los parámetros de alguno de los pares, tendremos una nueva posibilidad.

Cuando encontremos una nueva posibilidad, la guardaremos y lanzaremos como posibilidad inicial para que busque nuevos caminos aplicando nuevas resoluciones.

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

3.5.2 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. Ya que sino tendríamos un conjunto de posibilidades, pero no tendríamos las relaciones entre ellas. Que al fin lo que queremos obtener un diagrama donde tengamos unas posibilidades y las relaciones entre ellas.

La matriz de transiciones consiste en una lista de transiciones entre posibilidades.
A continuación vemos los campos por los que esta compuesta una transición.

Transición = ***{Posibilidad origen, posibilidad destino, resoluciones de transición, condición de la transición}***

Este proceso se tiene que 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 ya.
Se procede a crear una nueva transición donde:

La posibilidad origen será la posibilidad inicial utilizada.

La posibilidad destino será la nueva posibilidad, es decir la posibilidad inicial mas las resoluciones aplicadas.

Las resoluciones de transición serán las resoluciones aplicadas a la posibilidad inicial para crear la nueva posibilidad.

La condición de transición será el evento que tiene que darse para que se realice esta transición.

3.5.3 Ejemplo de espacio de posibilidades.

En la siguiente subsección vamos a ver un ejemplo para ayudar a aclarar los conceptos detallados anteriormente. Partiendo de las siguientes figuras se va a ver un espacio de posibilidades muy simple y su correspondiente matriz de navegación.

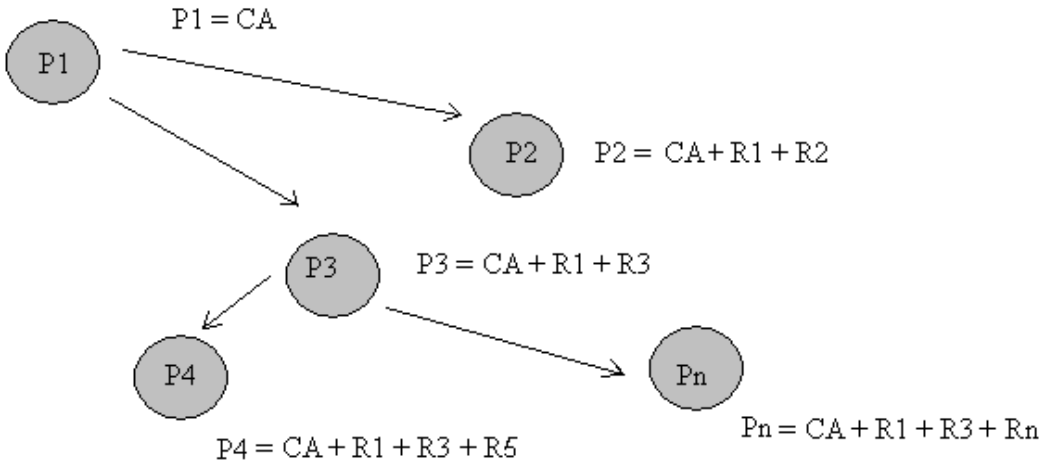


Figura 20. Ejemplo espacio posibilidades simplificado.

Como se puede observar en la figura el diagrama tiene cinco posibilidades y cuatro transiciones.
A la primera posibilidad se le ha aplicado solamente la configuración actual.
La segunda posibilidad parte de la posibilidad uno como origen y se le han aplicado las resoluciones uno y dos. Siguiendo el cálculo por ese camino no se han encontrado nuevas posibilidades.
En la tercera posibilidad se abre un nuevo camino, cogiendo como origen la posibilidad uno, y aplicándole las resoluciones uno y tres.
Siguiendo este camino, con la posibilidad tres como origen y aplicando la resolución cinco creamos la posibilidad cuatro. Partiendo con esta como origen no obtenemos ninguna nueva posibilidad.
Así que volvemos a la posibilidad tres y le aplicamos la resolución *n* para conseguí la nueva posibilidad *n.*

A continuación y siguiendo con el ejemplo anterior vamos a ver la matriz de navegación para el espacio de posibilidades anterior.

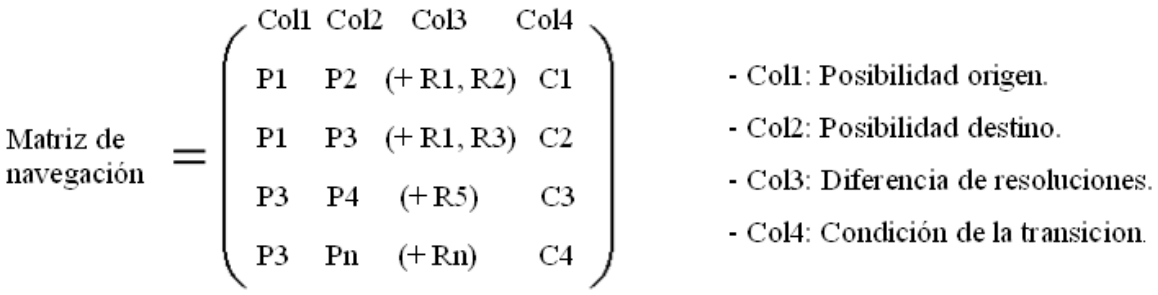


Figura 21. Ejemplo matriz de navegación.

Como se observaba en la figura del espacio de posibilidades teníamos 4 transiciones entre posibilidades. Esas serán las entradas que tendrá la matriz de navegación.

En la primera línea de la matriz de navegación vemos la transición entre la posibilidad uno y la posibilidad dos. Donde la las resoluciones de transición son R1 y R2, y la condición de transición es la condición uno.
La siguiente transición es la que va de la posibilidad uno a la tres, y las resoluciones añadidas son la R1 y la R3. Su condición de transición es la dos.
La transición que va de tres a cuatro tiene por resolución de transición la cinco y por condición la tres.
La última transición es la que enlaza la posibilidad tres con la *n* a la que se a añadido la resolución *n* y su condición de transición es la cuarta.

3.6 Validación de las posibilidades.

Para completar el espacio de posibilidades de nuestro sistema para una casa inteligente, es necesario conocer que posibilidades son validas para el sistema y cuales no.

Para ello es necesario validar cada posibilidad individualmente. Esto se consigue aplicando a la configuración asociada a cada posibilidad una validación de las restricciones del modelo de características del sistema de la casa inteligente.

Para realizar estas validaciones utilizamos un analizador de modelo de características [*] (FAMA).

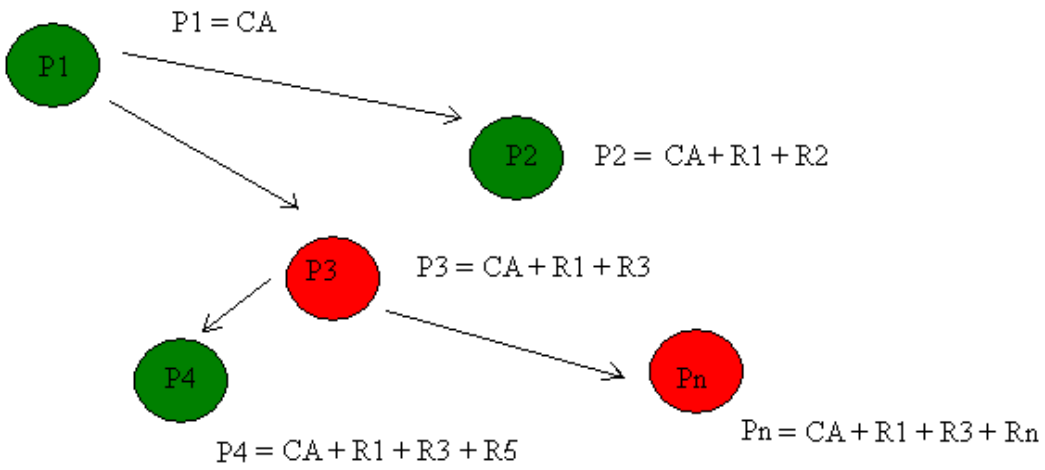


Figura 22. Ejemplo espacio de posibilidades validado.

Vemos en la figura anterior el espacio de posibilidades de la figura 21, pero con las posibilidades coloreadas en función de su validez. Las posibilidades con configuración valida están pintadas en color verde mientras que las inválidas están en color rojo.

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

3.7 Espacio de posibilidades al espacio de posibilidades abstracto.

En este subapartado se va ha tratar de cómo se realiza la transición del espacio de posibilidades al espacio de posibilidades abstracto.
La clave para realizar esta transformación es agrupar todas las posibilidades que tengan el mismo estado y que estén conectadas mediante transiciones. A continuación vamos a detallar como se realiza dicha tarea.

Se empieza 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 integraban.

Se seguiría por la transición que nos lleva a la posibilidad que tenia un estado diferente y recorreríamos todas las posibilidades de estado igual a esta, para así poder agruparlos como se hizo con la anterior. Y así sucesivamente hasta completar todas las posibilidades.

Como se hizo en el espacio de posibilidades tendremos que tener una matriz de navegación para saber las transiciones entre las nuevas posibilidades abstractas.
En estas transiciones que tenemos en la matriz de navegación, es muy importante el campo de la condición de transición que vimos en el capitulo 4.
Esta condición es un dato útil porque es el evento de contexto que hace que el sistema pase de un estado valido a uno inválido y viceversa.

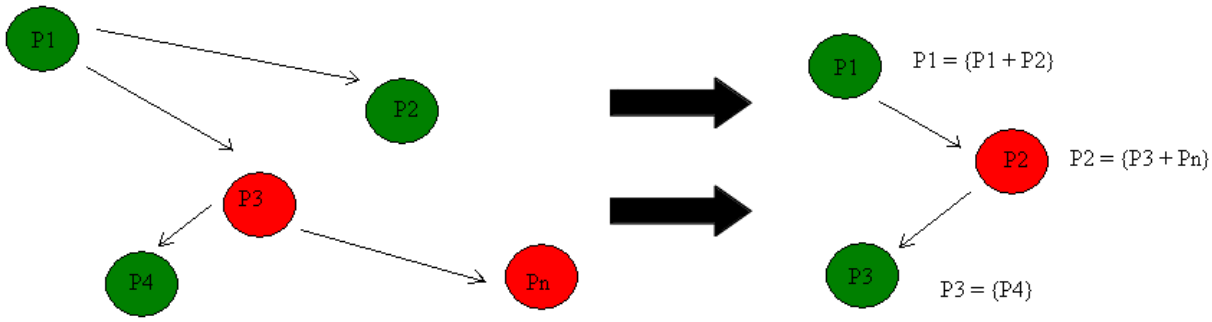


Figura 23. Ejemplo transición al espacio de posibilidades abstracto.

En la figura 23 vemos el ejemplo de espacio de posibilidades que se ha estado usando en secciones anteriores, y como este pasaría a espacio de posibilidades.
Se observa como pasa de tener cinco posibilidades a tener tres. Por el agrupamiento de las posibilidades validas {P1, P2} y las invalidas {P3, Pn}.

3.8 Uso del espacio de posibilidades abstracto.

En este subapartado vamos a explicar que uso va ha tener el espacio de posibilidades abstracto y como se va a utilizar.
Como se ha explicado anteriormente el espacio de posibilidades abstracto es una simplificación del espacio de posibilidades, esta simplificación se hace porque en los casos reales de casas inteligentes los espacios de posibilidades son enormemente grandes y es muy difícil trabajar sobre ellos.
Por ejemplo para un modelo de características muy simple como el de la figura 2, tendríamos 8 características con dos posibles estados, calculando el máximo número de posibilidades que se podrían generar se darían 20.922.789.888.000.
Luego no se generan todos, pero aun así estamos hablando de un orden de magnitud de billones. Por lo cual es una magnitud que no se puede manejar a la hora de trabajar sobre el espacio de posibilidades.
Por eso se simplifica el espacio de posibilidades en el abstracto que mantiene solo los estados que tiene transiciones a posibilidades en estado diferente, que es lo que se necesita para el análisis.
Una vez tenemos el espacio de posibilidades abstracto solo nos quedamos con la información que nos interesa. Lo cual facilita en gran medida el trabajo de los diseñadores, a la hora de detectar que transiciones son las que conducen a posibilidades del sistema inválidas. Y es mucho más fácil para estos el eliminar las transiciones de llevan a posibilidades invalidas. Por lo cual este trabajo será mucho más sencillo y rápido que si se trabajase sobre los espacios de posibilidades normales.

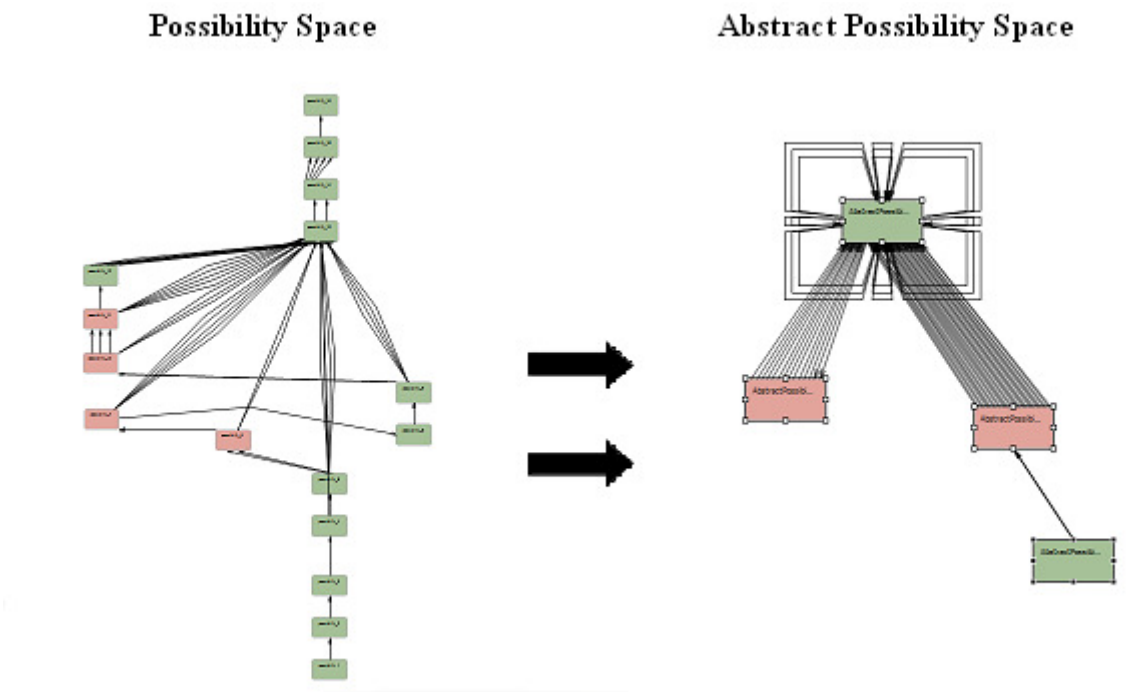


Figura 24. Transición al espacio de posibilidades abstracto real.

4. Editor de Resoluciones

Introducción general

A lo largo de este capitulo vamos a ver como la aplicación que se ha desarrollado, realiza los pasos presentados en el capitulo anterior.

Esta aplicación esta dividida en cuatro perspectivas. En este capitulo se van a ver en detalle las dos primeras.

La primera perspectiva con nombre *General*, aparte de cargar el modelo de características y presentar sus detalles, se realiza la funcionalidad de validar la configuración actual, paso detallado en el capitulo anterior.

La segunda perspectiva *Resolutions,* esta enfocada principalmente a la edición de las resoluciones, explicado en el apartado overview de la propuesta.

La forma en la que se va ha estructurar el capitulo va a ser la siguiente, de las dos perspectivas que se presentan, primero se vera la parte de la presentación y luego se verán algunos detalles interesantes la de implementación.

La parte de la presentación de las perspectivas se mostrara mediante capturas para ayudar a los futuros usuarios a familiarizarse con ellas. Y apoyándonos en estas se vera en detalle para que se utiliza y como se usa cada elemento de la aplicación.

De la parte de implementación se detallaran los puntos más importantes e interesantes. Para ello se mostraran partes de código claves y se explicara su funcionamiento y sus particularidades.

4.1 Edición del modelo de características.

Antes de comenzar ha utilizar nuestra aplicación es necesario editar un modelo de características con las características de la casa para la cual se va a diseñar el sistema domótico. El la síguete figura tenemos un ejemplo de casa domótica: con sus sensores y actuadores domoticos interconectados.

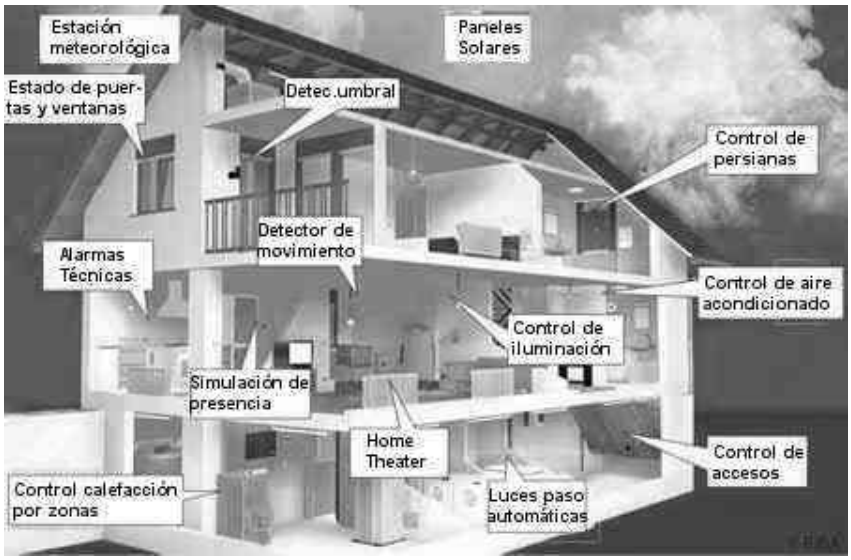


Figura 25. Dibujo Smart Home 2.

Para editar el modelo de características que utilizaremos para nuestra aplicación se utilizara la aplicación FeatureModel Diagram Editing de eclipse/Moskkit.
Como vemos en la figura 26, esta aplicación es fácil de utilizar simplemente hay que ir cogiendo los componentes que se quiera añadir e ir arrastrándolo a nuestro modelo.
En la parte derecha de la figura vemos la paleta donde podemos encontrar los diferentes componentes que podemos utilizar en el modelo, los componentes del modelo de características se explicaron anteriormente en *3.3 Validación de configuraciones*.

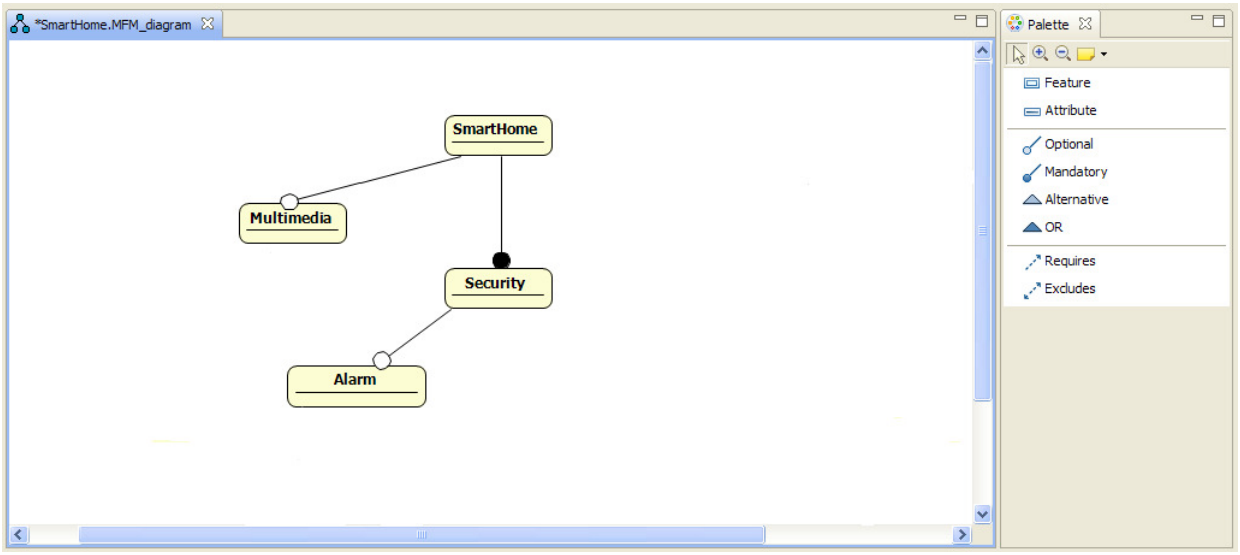


Figura 26. Editor de Modelos de características.

4.2 *Perspectiva General.*

4.2.1 Presentación perspectiva *Genera.*

Lo primero que observamos al arrancar la aplicación *recoAT*, es la pantalla de inicio *General*. Los campos se encuentran vacíos y vemos un botón *Load Model.*

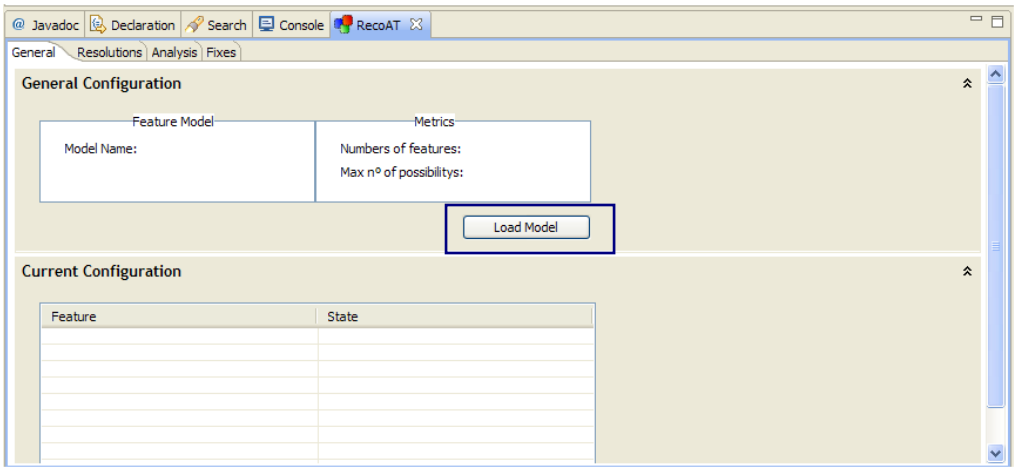


Figura 27. Pantalla principal vacía.

Para utilizar la aplicación tenemos que abrir un modelo desde el explorador de la izquierda y luego cargarlo desde el botón *Load Model.*

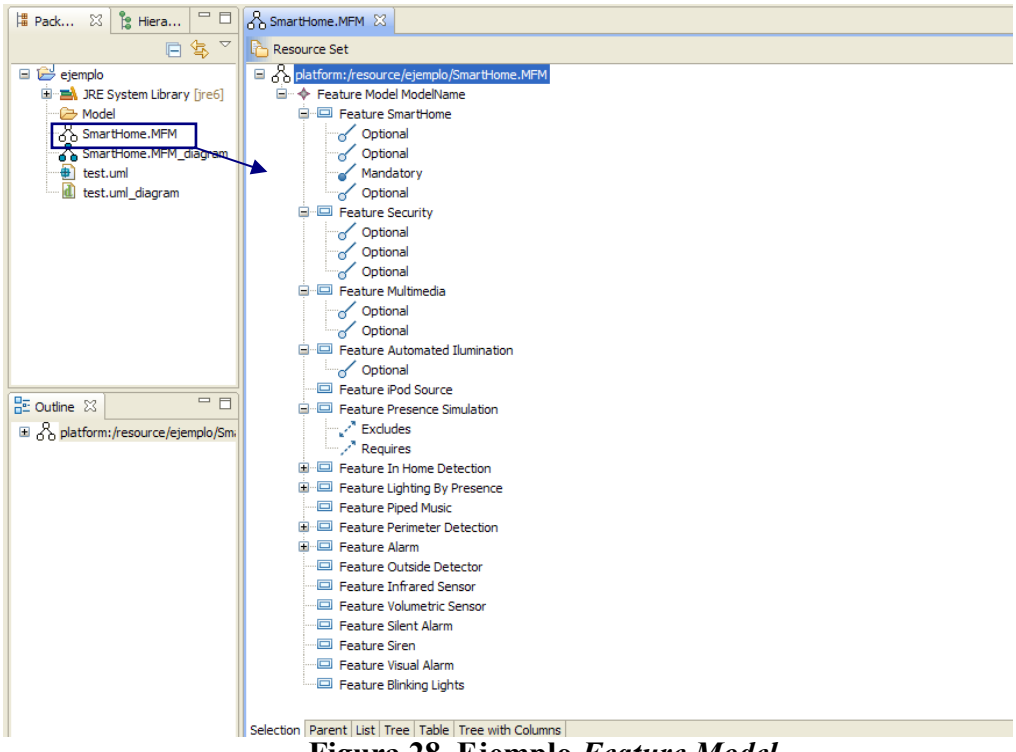


Figura 28. Ejemplo *Feature Model.*

Una vez cargamos el modelo, ya se puede comenzar ha utilizar la aplicación. Lo primero que se observa es que se rellenan los campos que estaban vacíos con información del modelo que se ha cargado.

En la siguiente figura se observa como quedaría la pantalla con el modelo ya cargado, en la parte superior vemos el modelo ya cargado. En la perspectiva *General*, en la parte superior vemos algunos datos extraídos del modelo de características, destaca por estar coloreado el estado de validez del modelo. En la parte inferior de la pantalla se observa una tabla con la configuración actual desglosada en característica y estado.

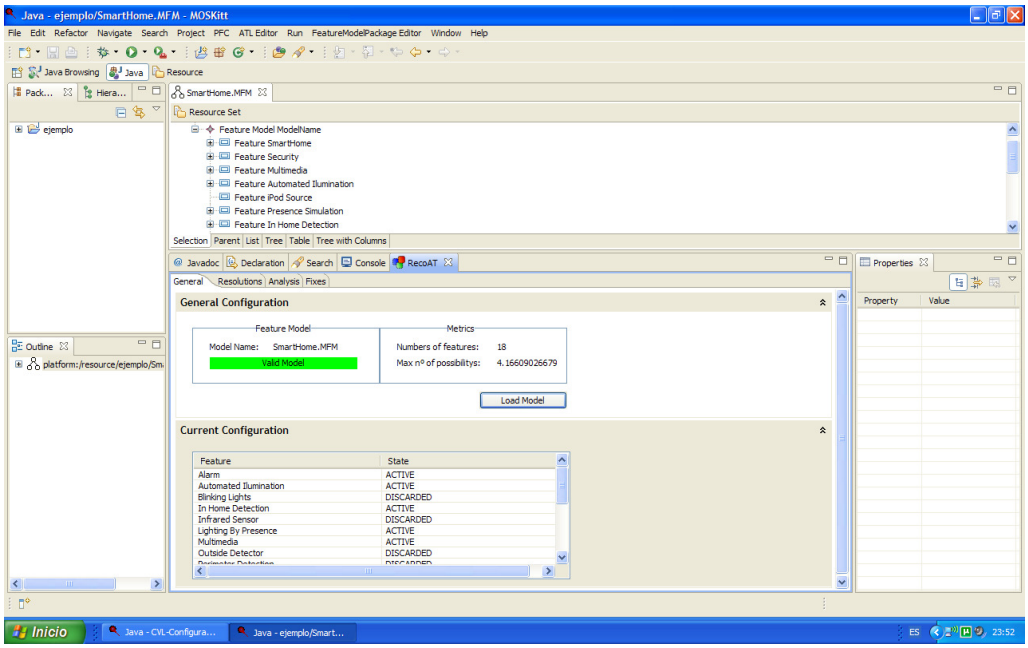


Figura 29. Pantalla inicial completada.

A continuación vamos a ver en detalle que todos los campos y apartados de la perspectiva *General.*

General Configuration:

Model name: Nombre del modelo de características usado.

Number of features: Número de características del modelo utilizado.

Valid Model / Invalid Model: Después de hacer una validación del modelo nos informara si el modelo es valido o no.

Max nº of possibilitys: El número máximo de posibilidades que se podrían generar con este modelo.

Current Configuration:

En esta sección de la pantalla, tenemos la configuración actual del modelo de características cargado en la parte de arriba de la pantalla. Como podemos observar en siguiente figura se presenta en forma de tabla con dos columnas:

Feature: Nombre de la característica.

State: Estado en el que se encuentra la característica.

Es de mucha utilidad tener la configuración actual de forma clara y visible durante el uso de la aplicación. Por esta razón se ha situado en la primera pantalla, de forma que siempre que se quiera consultar resulte fácil y cómodo.

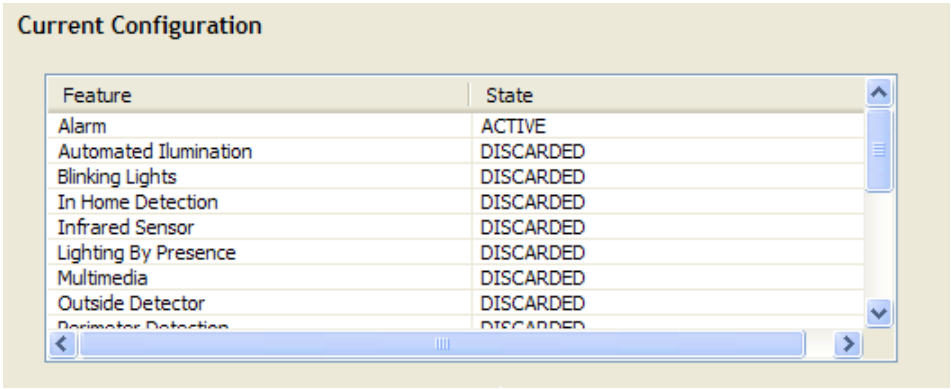


Figura 30. Pestaña General *(Current Configuration).*

4.2.2 Implementación perspectiva *General.*

En este apartado vamos explicar algunos detalles interesantes de la implementación de la perspectiva *General* de la aplicación.
Se va a ver como se extraen del modelo de características la configuración inicial y la lista de características. Además se explicara las transformaciones que se realizan al modelo de características para poder realizar las validaciones con el analizador FAMA.

Para ello se va ha seguir el código que realiza esta tarea, el cual vemos en la figura siguiente.

```
    private void Load_model(String Path){

        mfm = new MFM(Path);

        setCurrentConfiguration(mfm);

        Configuration configuration = new Configuration();

        FeatureModel fm = setConfiguration2MFMmodel(mfm.getFeatureModel(), configuration);
        boolean result1 = checkConfiguration(fm);

        if(result1){
            label11.setText("                    Valid Model");
            label11.setBackground(display.getSystemColor(SWT.COLOR_GREEN));

        } else{
            label11.setText("                    Invalid Model");
            label11.setBackground(display.getSystemColor(SWT.COLOR_RED));
    }
}

    private void setCurrentConfiguration(MFM mfm){
        List featureList = mfm.getFeatureModel().getFeatures();

        for (int i = 0;i<featureList.size();i++){

            FeatureModelPackage.Feature featue= (FeatureModelPackage.Feature) featureList.get(i);
            String featureName = featue.getName();
            FeatureConfiguration featureState = featue.getSelection();

        currentConfiguration.setFeature(featureName, featureState);
    }
}
```

Figura 31. Código método *Load model.*

Al observar el código lo primero que observamos es la clase Load_model(), a la cual le pasamos la ruta de donde se encuentra el modelo de características con el que vamos a trabajar.
En la primera línea vemos como se crea una instancia de la clase MFM() pasándole la ruta del modelo.
En la siguiente línea llamamos a la clase setCurrentConfiguration() pasándole la clase MFM.

Pasamos al cuerpo de esta clase que se puede observar en la parte inferior de la figura.
En esta clase vemos como primero almacenamos en una lista, las características del modelo que se extraen del modelo de características, el cual obtenemos de la clase MFM.
El siguiente paso seria iterar sobre la lista de características, y en cada característica obtenemos el nombre de la misma y su estado, y esto se guarda va guardando en una lista de pares (característica, estado) el cual es la configuración actual del sistema.

Se vuelve a la función Load_Model() la línea siguiente a la llamada a setCurrentConfiguration(), se crea una configuración vacía, la cual junto con el modelo obtenido de la clase mfm, es necesaria para en la línea siguiente obtener una instancia de la clase FeatureModel. Esta instancia de Feature model es la que pasaremos como parámetro a la función CheckConfiguration(). Esta función es la que analizara si la configuración actual del modelo de características con el que estamos trabajando es valida o no.
Esta función nos devolverá un booleano con validez o no de la configuración. Con este resultado la aplicación imprimirá en la parte de presentación si la configuración actual es valida o invalida.

4.3 Perspectiva *Resolutions.*

4.3.2 Presentación perspectiva *Resolutions.*

Al abrir la segunda pestaña de la aplicación, lo primero que se observa son una serie de tablas y campos vacíos. Podemos comenzar a trabajar de dos formas, una es comenzar a editar resoluciones desde cero o si ya se ha trabajado anteriormente con este modelo de características, se puede cargar un archivo con resoluciones guardadas anteriormente.

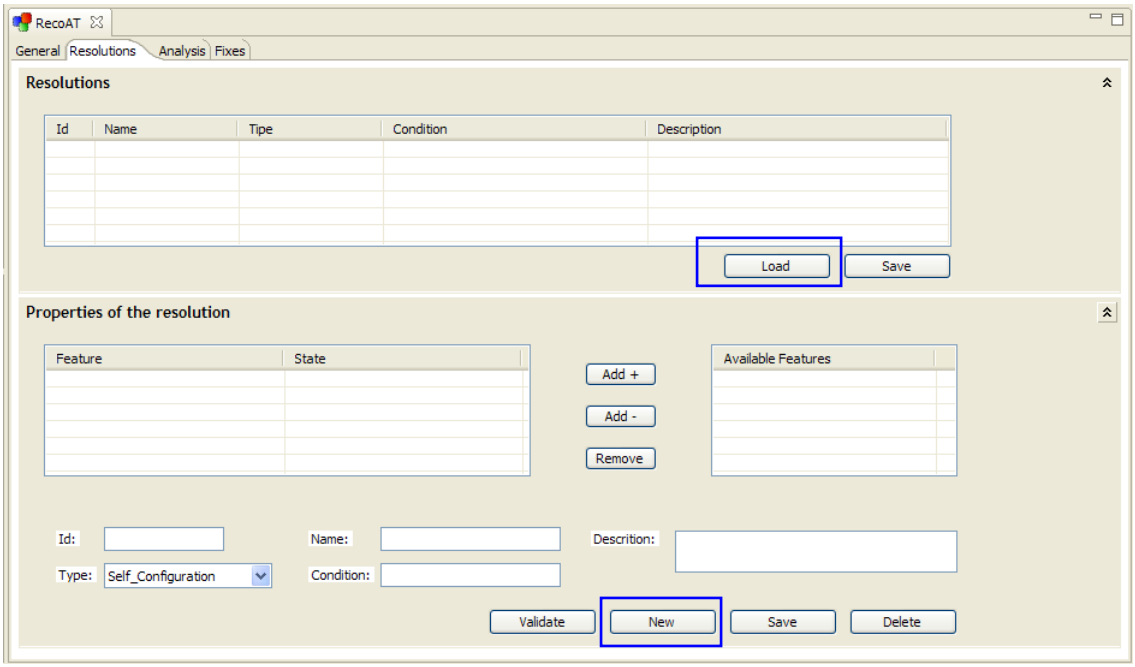


Figura 32. Pestaña *Resolutions* vacía.

Si tenemos guardadas en un archivo resoluciones del mismo modelo de características, se pueden cargar en el programa fácilmente, simplemente se tiene que presionar el botón Load que vemos enmarcado en azul en la figura. Una vez presionado nos aparecerá una ventana con un explorador en el que se tendrá que buscar el archivo con las resoluciones seleccionarlo y nos aparecerán las resoluciones en la tabla que sale en la parte superior de la figura.

Si no tenemos resoluciones del modelo que estamos trabajando se tendrá que empezar a crearlas de nuevo. Para ella lo primero es pulsar el botón *New* que vemos en la parte inferior de pestaña en un cuadrado. Una vez presionemos todos los campos excepto el *Id* se convertirán en editables y podremos comenzar a configurar resoluciones. Una vez este configurada podemos validarla, eliminarla o guardarla.

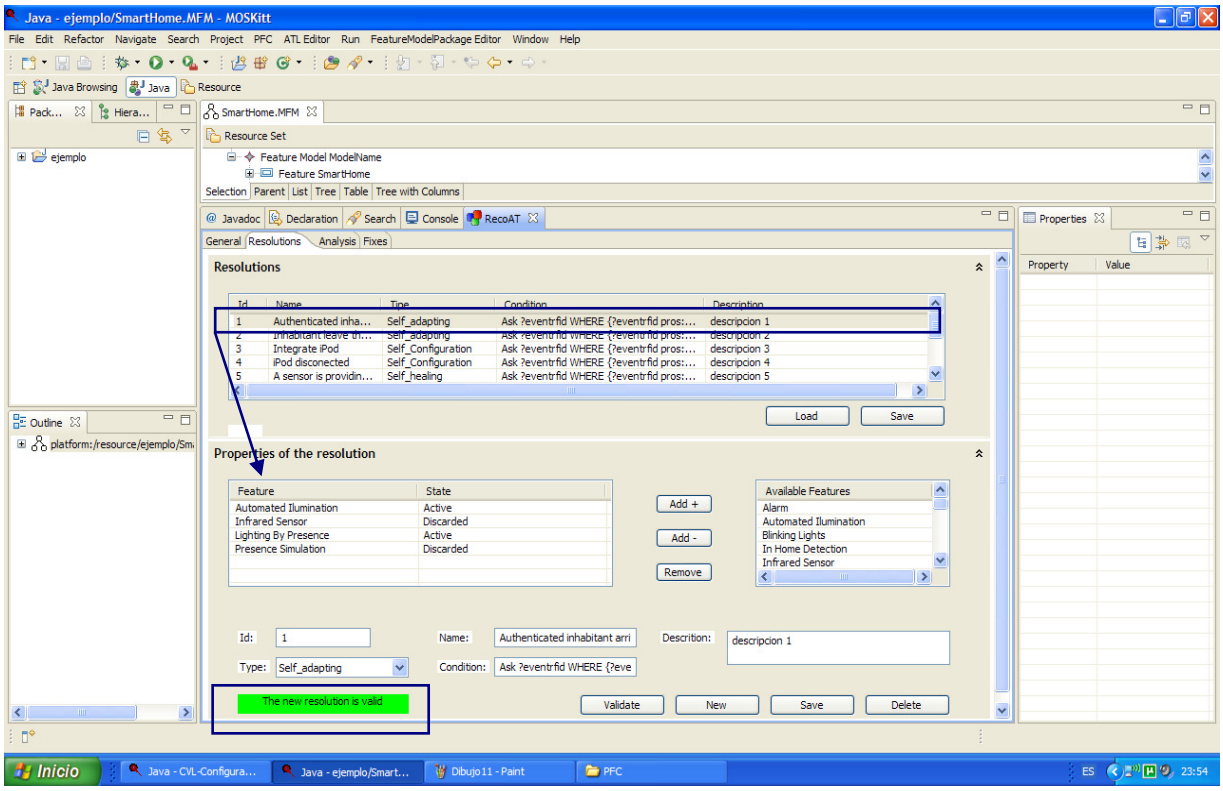


Figura 33. Pestaña *Resolutions.*

Una vez tenemos una lista de resoluciones, seleccionado alguna de las resoluciones modificar todos los campos de la resolución, excepto el *Id* que es automático.
En *Avaible Features* se cargan todas las características del modelo, así podemos cargar en la lista de características de la resolución cualquiera de las características del modelo tanto con estado activo como inactivo. También podemos eliminar las ya incluidas.
Lo mas practico de este editor es que podemos analizar si la resolución que estamos creando o modificando, es valida o no, gracias al botón *Validate* situado en la parte baja de la pantalla. Para este análisis no es necesario guardar la resolución se analizaran las características que tengamos en ese momento en el editor.
Por ultimo cuando ya tengamos la resolución totalmente creada o modificada, y sabiendo que es valida la podemos guardar con el botón *Save* que encontramos también en la parte baja de la pantalla.

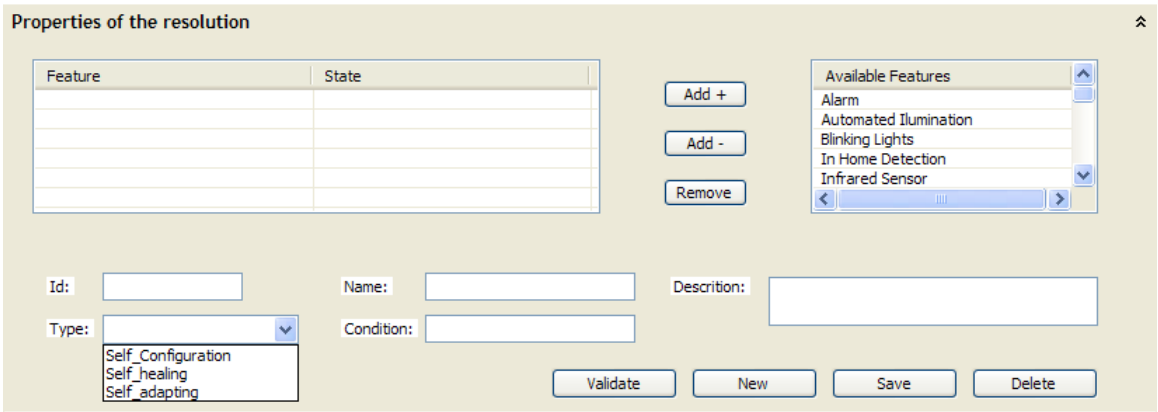


Figura 34. Plantilla de edición de las resoluciones.

A continuación se va a ver con detalle todos los elementos de la perspectiva *Resolutions*:

Lista Feature/State:

En esta tabla se encuentran las características con sus respectivos estados, si se carga desdés archivo tendremos las características pertenecientes a la resolución cargada, sino estará vacía. En ambos casos podremos modificar esta lista con los botones que tenemos inmediatamente a la derecha.

Botón Add +:

Con este botón se puede añadir la característica seleccionada, con estado *Activo*, de la lista Avaible Features a la lista de características de la resolución.

Botón Add -:

Con este botón se puede añadir la característica seleccionada, con estado Ina*ctivo*, de la lista Avaible Features a la lista de características de la resolución.

Botón Remove:

Con este botón se puede eliminar la característica seleccionada en la lista de características de la resolución.

Lista Avaible Features:

Es una lista de todas las características que incluye el modelo cargado previamente.

Id:

Es el numero de identificación de la resolución no se puede modificar ni editar ya que es un identificador que asigna automáticamente la aplicación.

Name:

Nombre que tiene la resolución cargada o que se le dará a la nueva resolución.

Descripción:

Algunos detalles más concretos de la resolución, los cuales no se han podido redactar en los otros campos.

Condition:

Es el suceso o condición que haría que esta resolución se activase.

Opciones del combo box Tipe:

Self-configuring:
Nuevos tipos de dispositivos pueden ser incorporados al sistema. 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 de control de iluminación automáticamente pueden hacer uso de el sin necesidad que el usuario lo tenga que configurar.

Self-healing:
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 fallada.

Self-adapting:
Las necesidades del usuario son diferentes dependiendo 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.

4.3.2 Implementación perspectiva *Resolutions.*

En este apartado vamos explicar algunos detalles interesantes de la implementación de la perspectiva *Resolutions* de la aplicación.

El objetivo de esta pantalla es la creación y modificación de resoluciones de forma fácil y rápida. Para ello se crearon dos zonas diferenciadas una con la lista de resoluciones creadas y otra con una plantilla de edición.

Para que resultara sencillo manejar estas dos zonas diferentes tenia que ser muy dinámico su funcionamiento. Así que estas dos zonas se tenían que sincronizar automáticamente.
Por ello cuando se selecciona una resolución en la lista de resoluciones, automáticamente se tienen que tener sus datos en la plantilla de edición listos para ser modificados.
Igualmente cuando se modifica alguna característica o a campo de la resolución y esta se guarda, automáticamente este cambio tiene que actualizarse en la lista de resoluciones.

Para llevar a cabo estas sincronizaciones se utilizaron los manejadores de eventos que proporciona la mayoría de clases java.

En concreto en el caso de lista de resoluciones mostrada en forma de tabla, se utilizo el manejador de selección de una tabla, que nos devuelve que línea de la tabla que esta seleccionada, sabiendo esta posición solo se tenía que recorrer la lista de resoluciones hasta la posición deseada. Con los datos de esta resolución se rellenan los campos editables y la tabla de características y estados de estas.

```
table.addListener (SWT.Selection, new Listener () {
    public void handleEvent (Event event) {

        aux_table = table.getSelectionIndex();
        r_select =  resolucion_list.get(aux_table);
        conditionsList = r_select.getConfigurationR().HashMap2Array();
        tabla1Refresh();

        text_id.setText(String.valueOf(r_select.getId()));
        text_name.setText(r_select.getName());
        text_condition.setText(r_select.getCondition());
        combo1.setText(r_select.getType().name());
        text_descrip.setText(r_select.getDescription());

    }
});
```

Figura 35. Manejador de elemento seleccionado en una tabla.

A continuación en la figura 36, vemos como se a implementado este manejador en la aplicación. En la primera línea de código del manejador se obtiene el índice de la línea de la tabla que esta seleccionada, en la siguiente recuperamos la resolución que se encuentra en esa posición. Seguimos por obtener la lista de características con la que se actualizara la tabla *Feature/Stat*, por ultimo actualizamos todos los campos edición con los datos de la resolución seleccionada.

En el caso de la modificación de algún elemento de las resoluciones y la posterior actualización de la tabla que muestra las resoluciones. Se utilizo el manejador que se acciona al pulsar un botón. Cuando se acciona el botón de guardar la resolución modificada, esta manejador modifica la resolución en la lista donde se almacenan todas las resoluciones y refresca la vista de la tabla con la lista de resoluciones actualizada con los nuevos datos.

```
Button save = new Button (group2, SWT.PUSH);
save.setBounds(590, 235, 90, 22);
save.setText ("Save");
save.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {

        r_select.setId(Integer.parseInt(text_id.getText().trim()));
        r_select.setName(text_name.getText());
        r_select.setDescription(text_descrip.getText());
        r_select.setCondition(text_condition.getText());

        if(combo1.getText().compareTo("Self_adapting")==0)
            r_select.setType(Type.Self_adapting);
        else if(combo1.getText().compareTo("Self_Configuration")==0)
            r_select.setType(Type.Self_Configuration);
        else
            r_select.setType(Type.Self_healing);



        if(new_r==true){
            resolucion_list.add(r_select);
            new_r = false;
        }else
            resolucion_list.set(aux_table, r_select);

        tablaRefresh();


    }
});
```

Figura 36. Manejador de activación de un botón.

En la figura anterior vemos como el manejador del botón *save* guarda todos los componentes de la resolución en una resolución auxiliar y finalmente actualiza la lista de resoluciones con los datos modificados de la nueva resolución.

Otra funcionalidad que se necesitaba en esta pestaña era poder validar resoluciones de forma rápida y fácil. Para ello se necesitaba poder realizar estas validaciones mientras se modificaban las resoluciones y antes de guardarlas en el sistema.
Para poder realizar estas validaciones en cualquier momento, se hubo que mantener una copia de la resolución de que se esta modificando y así cuando los usuarios quieran validar la resolución. Se validara la configuración asociada la copia de la resolución que se esta modificando y no la resolución original que se tiene guardada.

5. Análisis de las reconfiguraciones.

Introducción general

En este quinto capitulo vamos a ver como la aplicación que se ha desarrollado, realiza los últimos pasos presentados en el capitulo de overview. En este capitulo se van a ver en detalle las dos ultimas perspectiva de la aplicación.

La primera perspectiva que vamos a ver en este capitulo, de nombre *Possibility Space*, realiza la funcionalidad del calculo del espacio de posibilidades y la posterior validación de estas posibilidades

Por ultimo en la perspectiva *Fixes,* se vera el paso que se explico en la overview, transformación de espacio de posibilidades a espacio de posibilidades abstracto.
También se pueden aplicar los fixes, que consisten en marcar o eliminar todas las resoluciones que produzcan configuraciones invalidas.

La forma en la que se va ha estructurar el capitulo va a ser la siguiente, de las dos perspectiva que se presentan primero se vera la parte de la presentación y luego se verán algunos detalles interesantes la de implementación.

La parte de la presentación de las perspectiva se mostrara mediante capturas para ayudar a los futuros usuarios a familiarizarse con ellas. Y apoyándonos en estas se vera en detalle para que se utiliza y como se usa cada elemento de la aplicación.

De la parte de implementación se detallaran los puntos más importantes e interesantes. Para ello se mostraran partes de código claves y se explicara su funcionamiento y sus particularidades.

5.1 Perspectiva *Possibility Space.*

5.1.1 Presentación perspectiva *Possibility Space.*

Lo primero que observamos al desplegar esta ventana es una tabla de *Possibility–Result* vacía y dos botones en la parte inferior derecha.
Si accionamos el botón *Calcúlate posibilites,* nos aparecerán todas las posibilidades posibles de combinar todas las resoluciones asociadas al modelo cargado. Y posteriormente si accionamos el botón *Run Analysis* serán analizadas por el analizador FAMA una por una todas las posibilidades generadas anteriormente y en la comuna *Result* de la tabla nos aparecerá al lado de cada posibilidad si esta es valida o invalida.

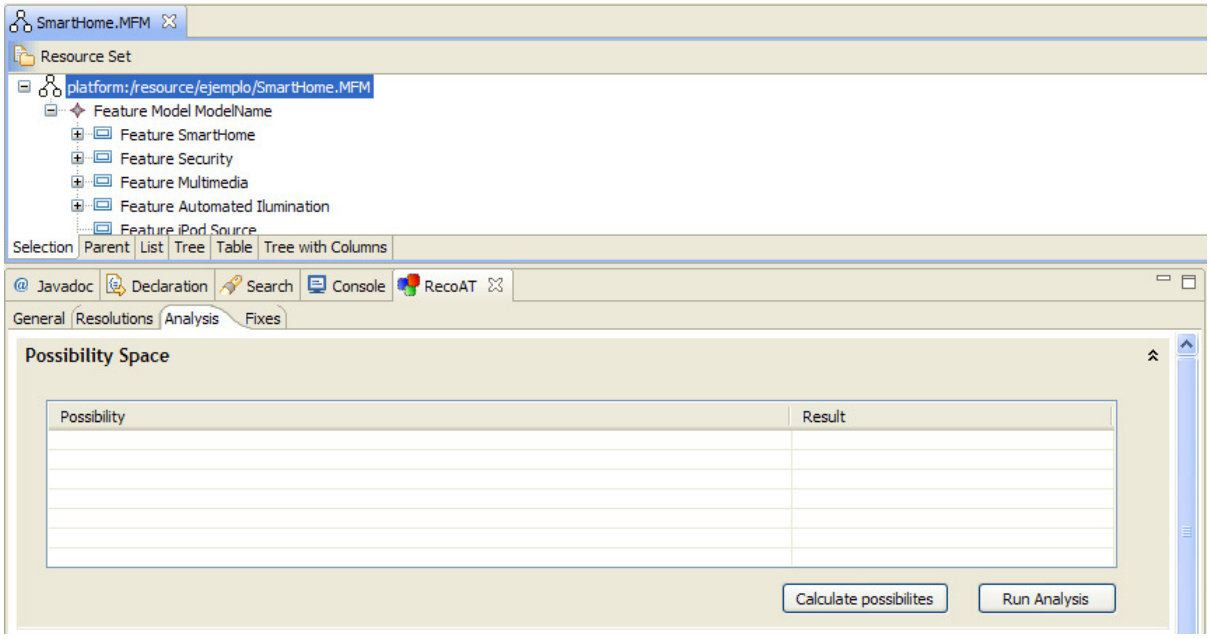


Figura 37. Pestaña *Analysis.*

- Possibility Space

A continuación vamos a detallar los elementos más importantes de este desplegable:

Lista *Possibility – Result*:

En esta lista nos aparecerán en la primera columna los nombres de las posibilidades generadas en función del modelo y de las resoluciones cargadas. En la segunda columna aparecerá si estas posibilidades generadas son validas o invalidas al validarlas con el modelo cargado.

Botón *Calcúlate posibilites*:

Al accionar este botón calculara todas las combinaciones posibles de las resoluciones cargadas con el modelo seleccionado.

Botón *Run Analysis*:

Al accionar este botón validara todas las posibilidades generadas anteriormente con el modelo seleccionado y nos dirá si la posibilidad es valida o invalida.

- Possibility Info

Al seleccionar una posibilidad de la lista se rellenaran los datos automáticamente del desplegable *Possibility Info*.

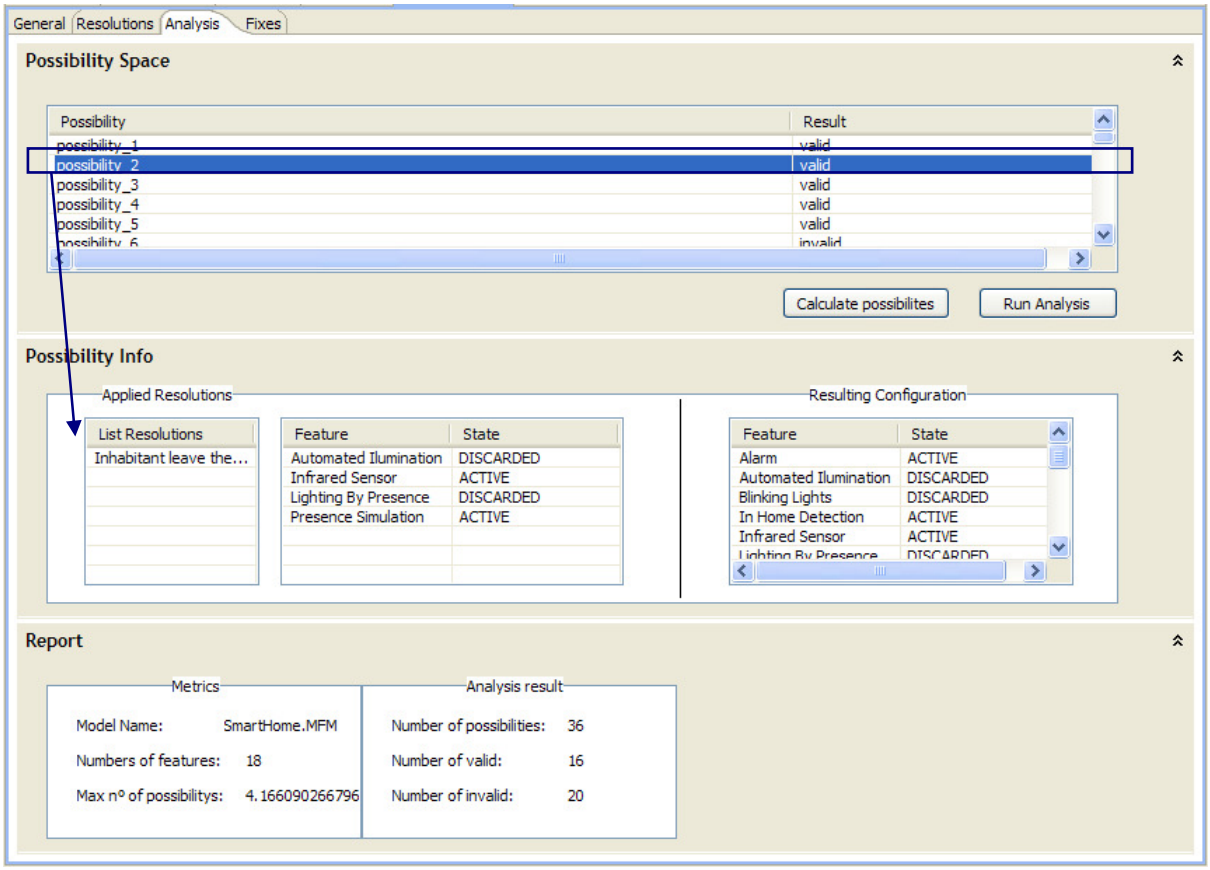


Figura 38. Pestaña *Analysis* completada.

Appled Resolutions

- ***List Resolutions:***

  En esta lista aparecerán los nombres de todas las resoluciones aplicadas a la posibilidad seleccionada.

- **Lista *Feature – State:***

  En esta lista se detallan las caracteristicas y el estado de la resolucion seleccionada en *List Resolutions*.

Resulting Configuration

Lista *Feature – State:*

En esta lista se detallan las características y el estado que presentan en la configuración de la posibilidad seleccionada. Esta configuración es el resultado de fusionar las características de las resoluciones de la posibilidad con las del modelo seleccionado.

- Report

El ultimo desplegable de la ventana es un informe que nos ayudara a entender el resultado de del calculo y la validación de las posibilidades. A continuación se detallan los campos de este informe:

Metrics

Model name: Nombre del modelo de características usado.

Number of features: Número de características del modelo utilizado.

Max nº of possibilitys: El número máximo de posibilidades que se podrían generar en este modelo.

Analysis result

Number of Possibilitys: Número de posibilidades generadas.

Number of valid: Número de posibilidades validas.

Number of invalid: Número de posibilidades inválidas.

5.1.2 Implementación perspectiva *Possibility Space.*

En este apartado vamos explicar algunos detalles interesantes de la implementación de la perspectiva *Possibility Space* de la aplicación.

El objetivo de esta pantalla es cálculo de todas las posibles posibilidades creadas a partir del modelo de características cargado y las resoluciones creadas y su posterior validación. También poder consultar las características, datos y estado de cada posibilidad.

A continuación en la figura 39, podemos ver el código de la clase *PossibilitySpace* que crea y amacena el espacio de posibilidades.

```
package org.eclipse.ui.articles.views;

import java.util.ArrayList;


public class PossibilitySpace {

    private ArrayList<Possibility> possibilityList = new ArrayList<Possibility>();
    private ArrayList<Transition> navegation = new ArrayList<Transition>();

    public ArrayList<Possibility> getPossibilityList(){
        return this.possibilityList;
    }

    public ArrayList<Transition> getNavegation(){
        return this.navegation;
    }

    public void calculate(Configuration currentConfiguration, ArrayList<Resolution> resolutionList){
        //...
    }

    private void populatePossibilitySpace(Possibility possibility,  ArrayList<Resolution> resolutionList){
        //...
    }

    private boolean exitsPossibility(Possibility possibility){
       //...
        return false;

    }
}
```

Figura 39. Código simplificado de la clase *PossibilitySpace.*

Como podemos observar en la declaración de la clase java de la figura, tenemos dos atributos: *possibilityList* y *navegación*. Y cinco métodos: get*PossibilityList, getNavegation, calculate*, *populatePossibilitySpace y existPossibility.*

En el atributo lista *possibilityList* almacenamos todas las posibilidades generadas.
Y en el atributo lista *navegación* almacenamos las transiciones que nos ayudaran a generar el espacio de posibilidades.

Lo siguiente que encontramos son los métodos get*PossibilityList y getNavegation*, estos métodos serán los que nos servirán para poder obtener estas listas desde fuera de la clase.
Lo siguiente que se observa son los métodos *calculate* y *populatePossibilitySpace* que serán los encargados de llenar las listas *possibilityList* y *navegation* con las correspondientes posibilidades y transiciones. Estos métodos se verán más detalladamente a continuación.
El método *existPossibility* ayudara en el proceso de obtención de las posibilidades, ya que cada vez que se cree una nueva posibilidad, se deberá comprobar si es realmente nueva o ya tenemos en el sistema una equivalente.

```
public void calculate(Configuration currentConfiguration, ArrayList<Resolution> resolutionList){

    Possibility initialPossibility = new Possibility(currentConfiguration);
    initialPossibility.setName("possibility_" + String.valueOf(possibilityList.size()+1));
    possibilityList.add(initialPossibility);
    populatePossibilitySpace(initialPossibility, resolutionList);
}
```

Figura 40. Código del método *calculate* clase *PossibilitySpace.*

El método calculate es el que se invoca para el cálculo del espacio de posibilidades, se le pasan la configuración actual del modelo de características y la lista de resoluciones.
Con esos parámetros lo que hace es obtener la posibilidad inicial, aplicando únicamente la configuración actual. Esta se guarda en la *possibilityList* e invoca al método *populatePossibilitySpace* con la posibilidad inicial y la lista de resoluciones anterior.

En la siguiente figura se va a ver el código del método *populatePossibilitySpace.* Lo primero que se realiza es obtener la configuración fuente de la posibilidad inicial que le pasamos como parámetro. Lo siguiente es iterar sobre todas las resoluciones de la lista de resoluciones, realizando la unión de la configuración inicial con la de la configuración perteneciente a cada resolución de la lista y lo guardamos en una nueva configuración. Se seguiría por la creación de una nueva lista de resoluciones donde añadiríamos las resoluciones asociadas a la posibilidad y la resolución seleccionada en esta iteración de la lista de resoluciones.
Teniendo los dos componentes de la posibilidad crearíamos la nueva posibilidad, se le pondría un nombre, se le añadiría la nueva configuración y la nueva lista de resoluciones.
Por ultimo se comprobaría si la posibilidad que se acaba de crear es nueva o existe una equivalente en el sistema para ello se usa el método de la clase *existPossibility* que nos devolverá *true* si ya existe la posibilidad y *false* si realmente es una nueva posibilidad.
En el caso de ser una nueva posibilidad se llamara recursivamente a la misma función con la nueva posibilidad creada y la lista e resoluciones usada anteriormente.

```
private void populatePossibilitySpace(Possibility possibility,  ArrayList<Resolution> resolutionList){

    Configuration sourceConfiguration = possibility.getConfiguration();

    for (int i=0;i<resolutionList.size();i++){
        //newPossibilityConfiguration
        Resolution resolution = resolutionList.get(i);
        Configuration targetConfiguration = resolution.getConfigurationR();
        Configuration newPossibilityConfiguration =
            Configuration.join(sourceConfiguration, targetConfiguration);

        //newPossibilityResolutionList
        ArrayList<Resolution> newPossibilityResolutionList = new ArrayList<Resolution>();
        newPossibilityResolutionList.addAll(possibility.getResolucions());
        newPossibilityResolutionList.add(resolutionList.get(i));

        //newPossibility
        Possibility newPossibility = new Possibility();
        newPossibility.setName("possibility_" + String.valueOf(possibilityList.size()+1));
        newPossibility.setConfiguration(newPossibilityConfiguration);
        newPossibility.setResolucions(newPossibilityResolutionList);

        //check if newPossibility already exits
        boolean exits = exitsPossibility(newPossibility);
        //add newPossibility to possibilityList
        if (exits==false){
            possibilityList.add(newPossibility);}
        //add navegation
        Transition transition = new Transition(possibility.getName(), newPossibility.getName(),
                resolution.getCondition(), String.valueOf(resolution.getId()));
        navegation.add(transition);

        if (exits==false)
            populatePossibilitySpace(newPossibility, resolutionList);}
    }
```

Figura 41. Código método *populatePossibilitySpace* de la clase *PossibilitySpace.*

5.2 Perspectiva *Fixes.*

5.2.1 Presentación perspectiva *Fixes.*

Lo primero que se observa en esta al desplegar esta última perspectiva son dos dibujos, que intentan explicar de forma visual la funcionalidad de los dos botones que los acompañan.
Lo siguiente que se puede ver es en el desplegable inferior son dos cuadros de texto, donde nos explica la funcionalidad de los botones que incluye el mismo cuadro.

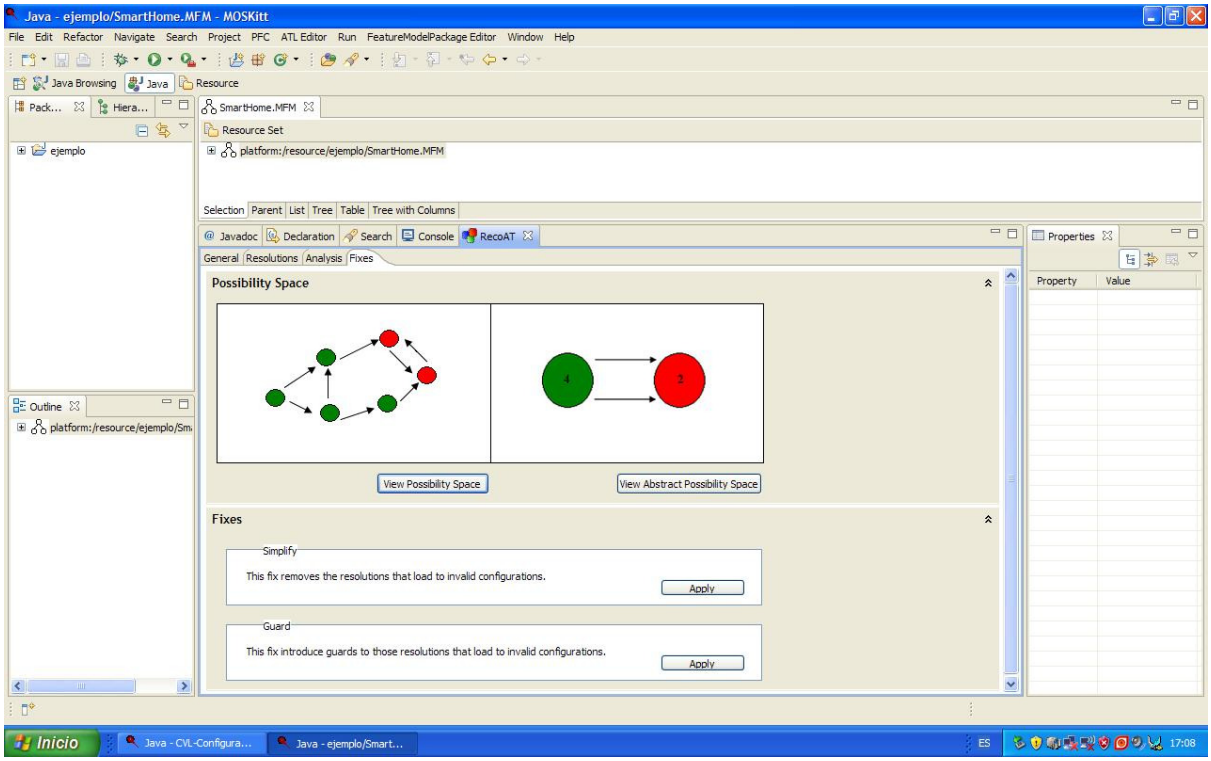


Figura 42. Pestaña *Fixes.*

A continuación se va a explicar los componentes que componen esta pantalla:

- Possibility Space

Botón View Possibility Space

Al accionar este botón se generara un archivo con el espacio de posibilidades asociado a al modelo de características cargado y a las resoluciones de este.

Botón *View Abstract Possibility Space*

Al accionar este botón se creara un archivo con el espacio de posibilidades abstracto construido simplificando el espacio de posibilidades generado anteriormente.

En la siguiente figura tenemos el desplegable *Fixes,* en el cual podemos aplicar dos diferentes ayudas a la hora de refinar las resoluciones que hacen que algunas posibilidades del espacio de posibilidades sean inválidas.

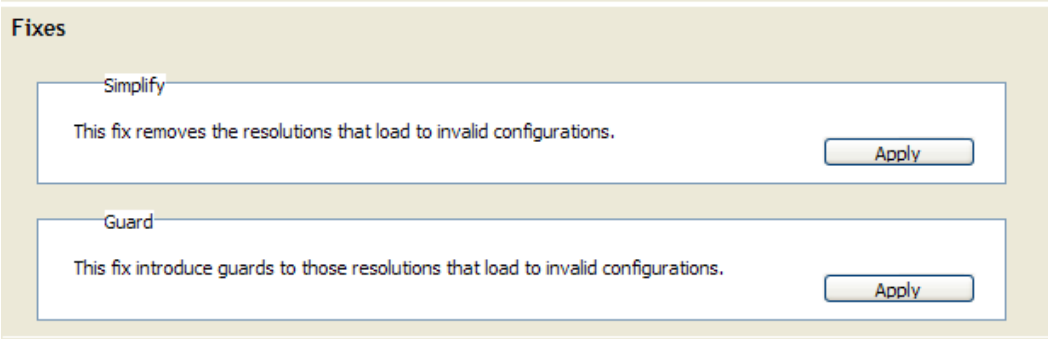


Figura 43. Vista del desplegable *Fixes*.

A continuación detallaremos que ayuda nos aporta cada uno de los *Fixes* al presionar sobre sus respectivos botones de *Apply*.

- Fixes

Simplify

Este fix lo que realizara será borrar de la lista de resoluciones cargadas en el sistema, todas aquellas que conduzcan a posibilidades que tengan configuraciones del sistema invalidas.

Guard

Lo que realizara fix, será marcar con unas guardas todas las resoluciones que produzcan posibilidades que tengan configuraciones del sistema inválidas. Mediante este mecanismo será más fácil volver al paso de editar las resoluciones y modificar solo aquellas resoluciones que estén dando problemas en algunas configuraciones del sistema.

5.2.2 Implementación *Fixes.*

En este apartado vamos explicar algunos detalles interesantes de la implementación de la perspectiva *Fixes* de la aplicación.
El objetivo de esta perspectiva es cálculo del espacio de posibilidades normal y abstracto, en función de las posibilidades creadas a partir del modelo de características cargado y las lista de resoluciones editada. Y la posterior aplicación de fixes que ayuden a los diseñadores a solucionar posibles problemas de invalidez de algunas posibilidades, marcando o eliminando aquellas resoluciones que provoquen configuraciones del sistema invalidas.

Se va a analizar el código del manejador de eventos del botón *View Possibility Space.* En este manejador se crea el diagrama *space.recoatpossibilityspace,* el cual albergara el espacio de posibilidades. Una vez creado el diagrama vamos añadiendo las posibilidades y transiciones que se han calculado anteriormente.

```
final UML2StateMachine stateMachine =
    new UML2StateMachine("C:/workspace/CVL-ConfigurationsSpace/Model/space.recoatpossibilityspace");

Button view = new Button (group41, SWT.PUSH);
view.setBounds(180, 190, 120, 23);
view.setText ("View Possibility Space");
view.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent e) {

        ArrayList<Possibility> possibilityList = possibilitySpace.getPossibilityList();
        ArrayList<Transition> navegation = possibilitySpace.getNavegation();


         if (possibilityList.size()>0)
         {
            Possibility firstPosibility = possibilityList.get(0);
            stateMachine.addInitState(firstPosibility.getName(), firstPosibility.getName().substring(12),
                    firstPosibility.getState());
         }

         for (int i=1;i<possibilityList.size();i++){

            Possibility posibility = possibilityList.get(i);
            stateMachine.addState(posibility.getName(), posibility.getName().substring(12),posibility.getState());
        }

        for (int pos=0;pos<navegation.size();pos++){
            Transition transition = navegation.get(pos);
            stateMachine.addTransition(transition.getSource(), transition.getTarget(),
                    transition.getCondition(), transition.getResolution());

        }
    }
});
```

Figura 44. Código manejador del botón *View possibility Space.*

Como se puede observar en el código, lo primero que se ha de realizar para obtener el espacio de posibilidades es crear una instancia de la clase *UML2StateMaquine* de nombre *stateMachine*, para ello pasamos al constructor la ruta donde se guardara.

Se obtienen las listas de posibilidades y transiciones de la clase *possibilitySpace*, las cuales se han calculado anteriormente.
Antes de empezar el proceso de dibujado del diagrame comprobamos que la lista de posibilidades no esta vacía. Si es así comenzamos por dibujar en el diagrama el estado inicial para lo cual tendremos que llamar al método *addInitState* de *stateMachine,* instancia que se ha creado de la clase *UML2StateMaquine,* al cual le pasaremos el nombre de la posibilidad inicial y el estado en el que se encuentra.

A continuación iteraríamos sobre toda las lista de posibilidades y llamando al método *addState* de *stateMachine,* pasándole el nombre y el estado de las sucesivas posibilidades. Este método ira dibujando los diferentes estados en el espacio de posibilidades.

Por ultimo se tendrán que recorrer todas las transiciones de la lista obtenida anteriormente, para poder dibujarlas en el diagrama y así completarlo. Para ello se utiliza el método *addTransition* de *stateMachine*, al cual le pasamos la posibilidad origen, la de destino, la condición y la resolución.

En este momento podemos se puede abrir el espacio de posibilidades que se habrá guardado en la ruta antes asignada.

5.3 Resultado final: Possibility Space y Abstract Possibility Space

Para finalizar este capitulo, en esta sección se va a ver el resultado final de todo el proceso, después seguir los todos los pasos detallados en el capitulo de overview se va ha obtener el espacio de posibilidades y el espacio de posibilidades abstracto.
Con el espacio de posibilidades obtenemos una simulación en la etapa de diseño del comportamiento que tendría el sistema en uso, que es lo necesitamos para poder refinar el sistema, poder garantizar su correcto funcionamiento en todo momento.

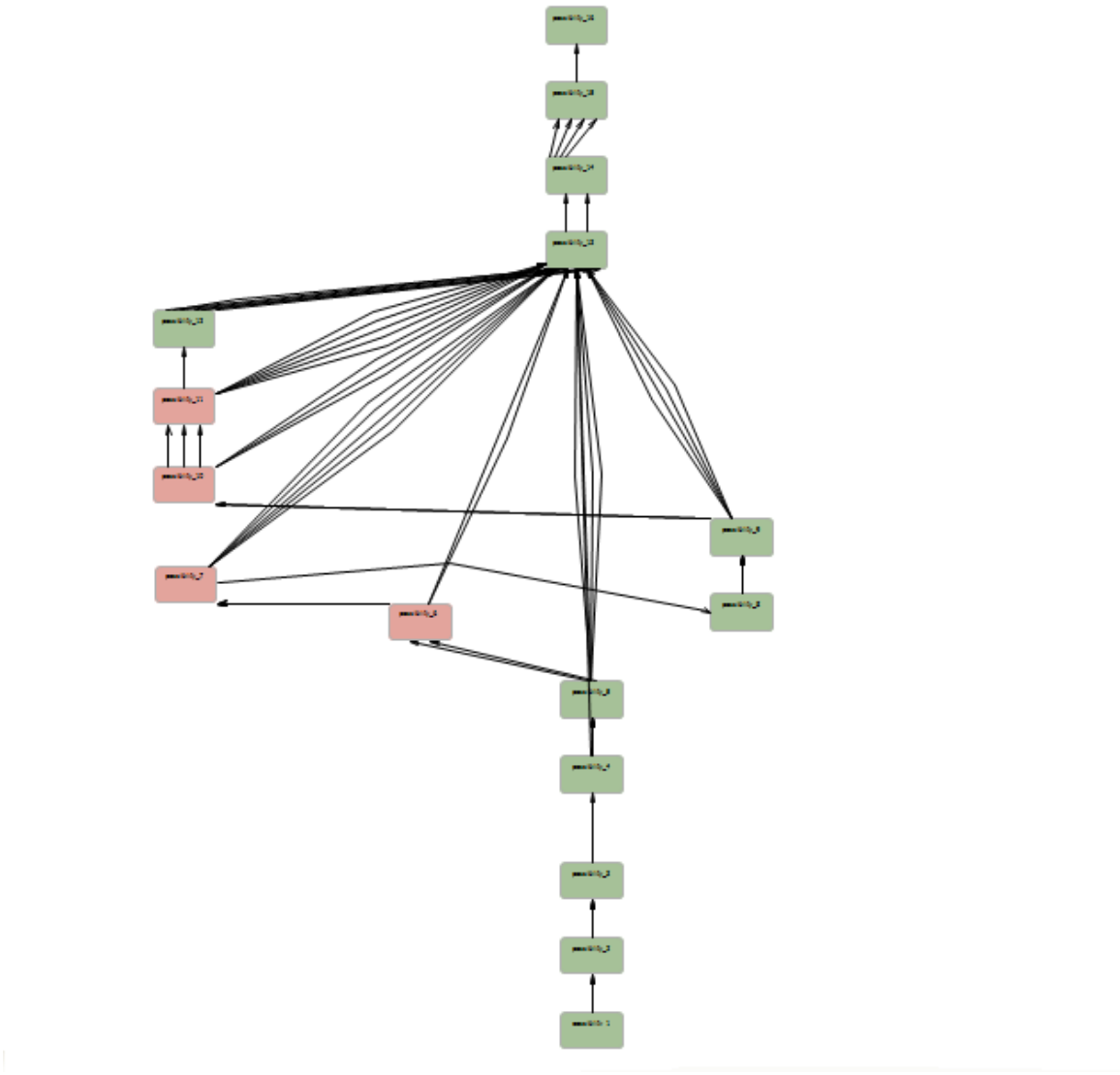


Figura 45. Ejemplo de espacio de posibilidades generado por la aplicación.

Como podemos observar en la figura 45 el diagrama resultante, se obtiene una sucesión de estados validos e inválidos que transitan de unos a otros en función de unas condiciones.

Se ha buscado un ejemplo de espacio de posibilidades bastante simple para que se pueda seguir fácilmente y para que se pueda observar la correspondencia con el diagrama de posibilidades abstracto de la figura siguiente. También se han omitido las condiciones de cada transición para que resultase mas claro de ver el diagrama.

Este espacio de posibilidades corresponde a un sistema domótica muy simple para que resulte sencillo de entender, pero aun así tiene 18 posibilidades y unas cuantas más transiciones. Si se tuviera que analizar que resoluciones han sido las que han producido las configuraciones inválidas no seria una cosa trivial se tendría que analizar y repasar todas las transiciones que llevan hacia posibilidades inválidas.
Por esa razón, también construimos el espacio de posibilidades abstracto, con el cual resultara mas cómodo y sencillo trabajar.
Si se estuviese trabajando con un sistema real, el espacio de posibilidades resultante seria imposible de tratar.

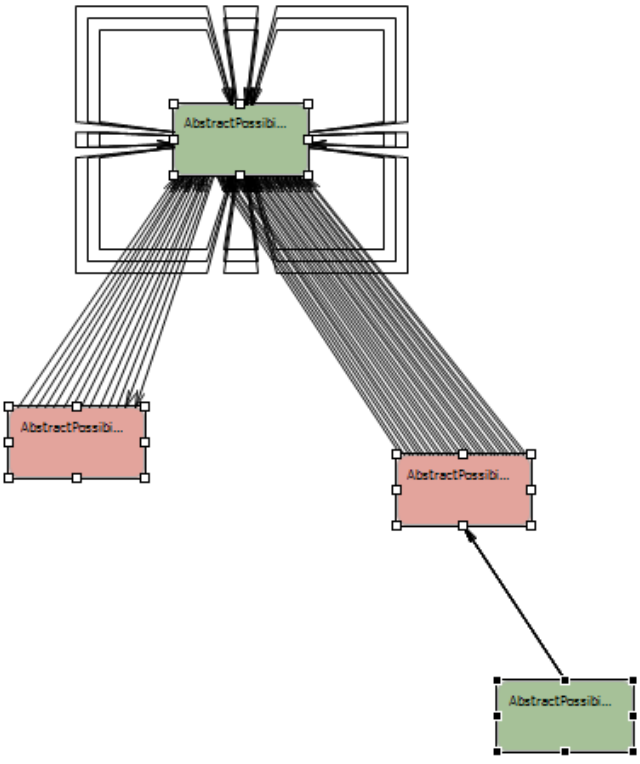


Figura 46. Ejemplo de espacio de posibilidades abstracto.

Como vemos en el espacio de posibilidades abstracto de la figura 46, se ha simplificado bastante respecto al de la anterior figura, en el anterior teníamos 16 estados mientras que en este solo 4.
Pues si esto lo extrapolamos a las grandes magnitudes con las que se trabaja en sistemas domoticos reales, la simplificación del trabajo de los diseñadores de sistemas puede ser muy grande.

5.4 Diagrama Implementación.

Como introducción a la aplicación RecoAT se va a dar un vistazo a su diagrama de clases, como se ve en la figura anterior se han diferenciado por colores las diferentes zonas del diagrama para que sea más claro de leer. A continuación se va ha explicar brevemente el diagrama, a lo largo de los sucesivos capítulos se irán explicando mas detalladamente los elementos de la aplicación.

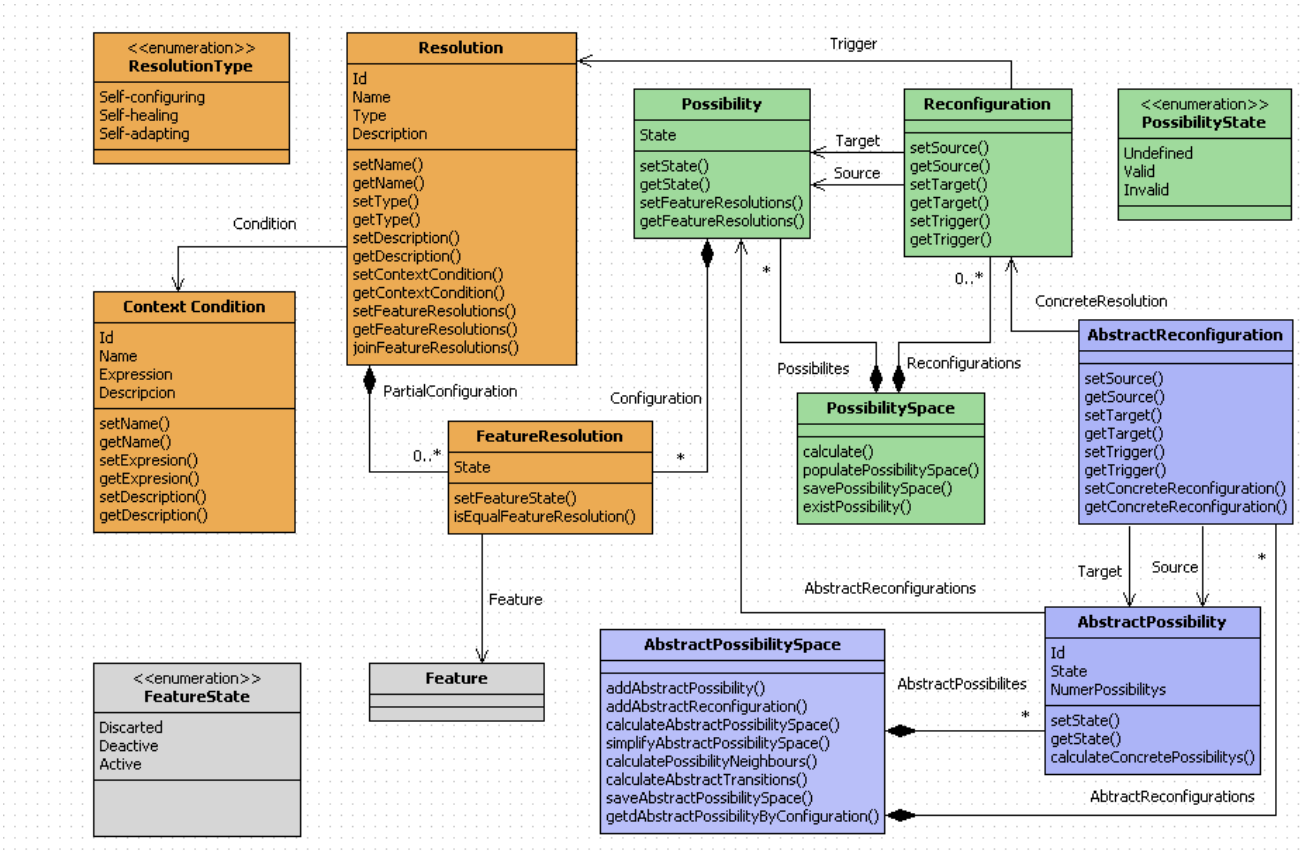


Figura 47. Diagrame implementación.

Lo primero que vemos arriba a la izquierda es la zona naranja, donde el elemento principal es la resolución, la cual tiene unos atributos y unas funciones. De las clases adyacentes, podemos resaltar la condición de contexto, las opciones del atributo tipo y el conjunto de pares característica, estado.

Bajo de la anterior vemos la zona gris donde la característica es la clase principal, también encontramos una enumeración con los posibles estados de la resolución característica.
En la zona verde el elemento principal es el espacio de posibilidades, el cual se forma por conjuntos de posibilidades y reconfiguraciones o transiciones entre estados, además tenemos las opciones del atributo estado de la posibilidad.

Por ultimo la zona azul con el espacio de posibilidades abstracto, que esta formado por conjuntos de posibilidades abstractas y reconfiguraciones abstractas.

6. Caso de estudio

6.1 Preparación

En este capitulo vamos a seguir todos los pasos de la aplicación pero utilizando un caso de estudio parecido a sistema real de una casa integente pero mas simplificado.

Los objetivos del capitulo son: comprovar el correcto funcionamiento de la aplicación realizando un proceso completo, y dar un ejemplo de uso a los futuros usuarios de la aplicación seguiendo el proceso de la aplicación de principio a fin siguiendo el mismo caso de estudio.

Para la preparacion de este capitulo se ha buscado un modelo de caracteristicas lo suficientemente completo para asemejarse a un sistema real. Y a la vez sencillo para ser facil de seguir en las explicaciones de su uso.

El primer paso realizado durante el caso de estudio es dadas unas caracteristicas de la SmartHome y unas restricciones editar un modelo de caracteristicas. Se ha seguido por hacer un estudio de las condiciones externas y de las costumbres de los habitantes para generar unas resoluciones adecuadas al modelo de la casa.
El paso siguiente, y ayudandonos de la aplicación RecoAT se ha generado el espacio de posibilidades. Seguido de un analisis de este espacio de posibilidades, posteriormente se ha optimizado este, para obtener un espacio de posibilidades mas simpre pero manteniendo las caracteristicas que nos interesaban. En el ultimo apartado se ha comprovado como con ayuda del espacio de posibilidades abstrasto, resulta sencillo detectar las resoluciones que provocan incumplimientos de las condiciones del modelo. Y se ha reailzado un ejemplo de cómo detectar una de estas resoluciones y modificarla para que cumpla siempre las condiciones del modelo.

6.2 Obtención del modelo de características

Para obtener el modelo de caracteristicas partimos de la configuracion de una casa inteligente con sus diferentes dispositivos (sensores, accionadores, servicios, etc) y restricciones.
Acontinuacion vamos a ver la descripcion de los diferentes elementos y las restricciones de uso para la casa inteligente que se va a usar en el caso de estudio.

SmartHome: La raíz del modelo, la cual engloba todas las funcionalidades de la casa inteligente.

Contiene las siguientes funcionalidades:

- **Multimedia:** Agrupa todas los servios multimedia: música, TV, cine, etc.

  Piped Music: Hilo musical de la casa.
  Ipod Source: Fuente de sonido 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 por sensors de presencia.

Restricciones que hay que aplicar al modelo:

- Se quiere dar mucha importancia a la seguridad del sistema, por lo cual siempre que el sistema este encendido el sistema de seguridad obligatoriamente tendra que estar encendido.
- En el sistema de alarma gestiona diferentes dispositivos, pero se quiere de entre Silent Alarm, Siren y Visual Alarm solo uno de estos dispositivos este encendido a la vez.
- Para que el sistema de simulacion de presencia pueda funcionar necesita necesariamente que el sistema deteccion en el hogar este conectado.
- Para que los sistemas de simulación de presencia y iluminacion automatica no tengan conflictos entre ellos no deberian conectarse a la vez.

Siguiendo la metodologia para la contrunstruccion de modelos de caracteristicas y ciniendonos a la descripcion anterior y a las restricciones correspondientes. Se ha obtenido el siguiente modelo, el cual se va a usar para llevar a cabo el caso de estudio.

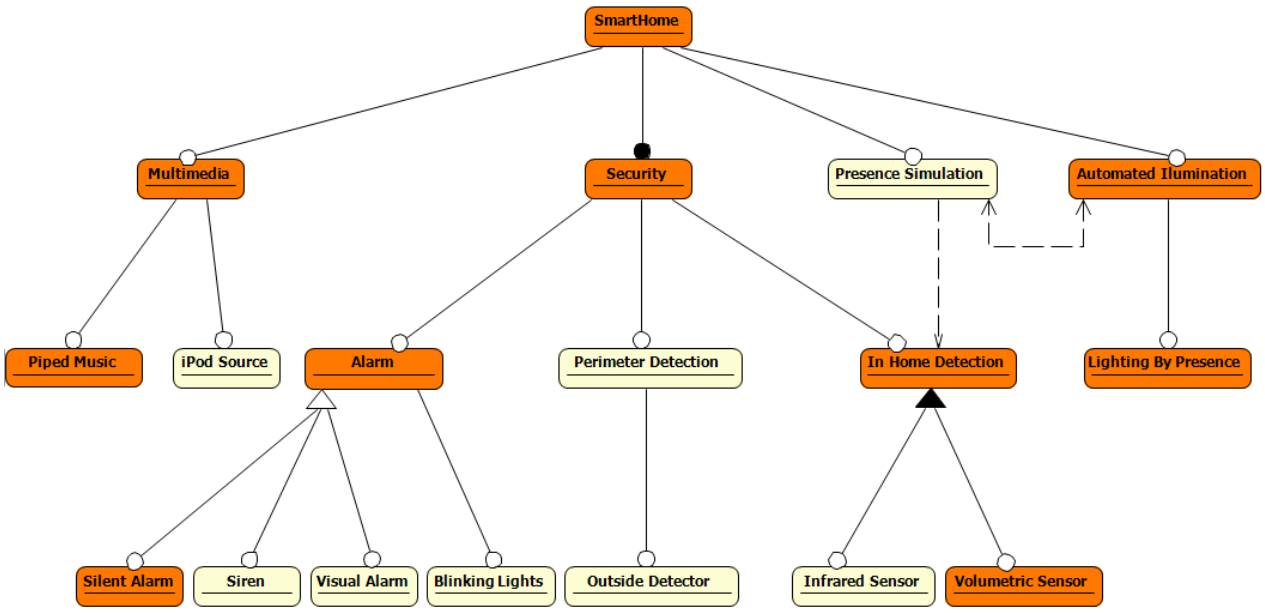


Figura 48. Modelo de características usado en el caso de estudio.

El siguiente paso una vez ya tenemos el modelo de caracteristicas es aplicar la configuracion inicial. Esta configuracion sera la que tendremos por defecto en el sistema.
Para nuestro caso se ha elegido la siguiente, el sistema principal *Smart Home* activado. El sistema multimedia tambien conectado con el con el hilo musical activo. Pasamos al sistema de seguridad que estara activo, con la alarma conectada y usando la version silenciosa. Siguiendo con el sistema de seguridad se tendra conectado el sensor volumetrico del sistema de deteccion del hogar.
Por ultimo el sistema de iluminacion automatica haciendose servir de el sistema de sensores de iluminacion por presencia.

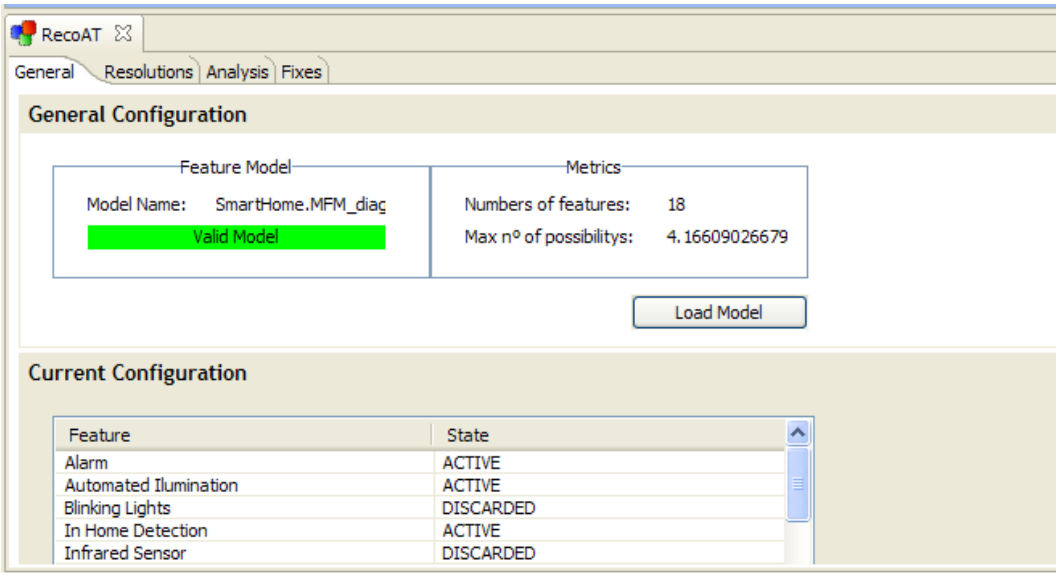


Figura 49. Pantalla inicial con la configuracion y el modelo validado.

Por ultimo antes de finalizar este primer paso de estudio se tiene que validar la configuracion incial que se ha editado anteriormente.
Para ello tendremos que cargar el modelo en la aplicación por medio del boton *Load Model* e indicar en una ventana de exploracion la ruta exacta donde se encuentra el modelo. Una vez el modelo esta cargado automaticamente la aplicación nos indicara si la configuracion inicial elegida para el modelo de caracteristicas correspondiente, es valida o no.
Como vemos en la anterior figura para el caso de estudio que se esta realizando, la aplicación nos dice que el modelo es valido.

6.3 Análisis de las resoluciones

En esta secccion vamos a ver como se aplicaria el analisis de resoluciones al caso de estudio concreto y utilizando la aplicación RecoAT.

Se va a detallar como se editaria la primera restriccion con la aplicación (ver Figura 50), primero editamos los campos en la parte inferior de la pantalla *Resolutions*. Los campos a editar son los siguientes: *Name, Description, Type, Condition*. Todos menos el campo *Id* que lo genera automaticamente la aplicación.
Se seguiria por añadir la caracteristicas que se quiere que tenga la resolucion. Para ello se seleccionan de la lista *Available Features* y se añaden con el estado deseado con el boton *Add* + con estado activo y con el boton *Add* - con estado inactivo.
Una vez añadidas las caracteristicas se puede modificar su estado con los botones correspondientes o eliminar la caracteristica con el boton *Remove*.
El siguiente paso seria validar la resolucion accionando el boton *Validate* si el resultado es valido se puede guardar y seguir con la siguiente resolucion, si es inavalido se modifica hasta conseguir que la validacion sea positiva.

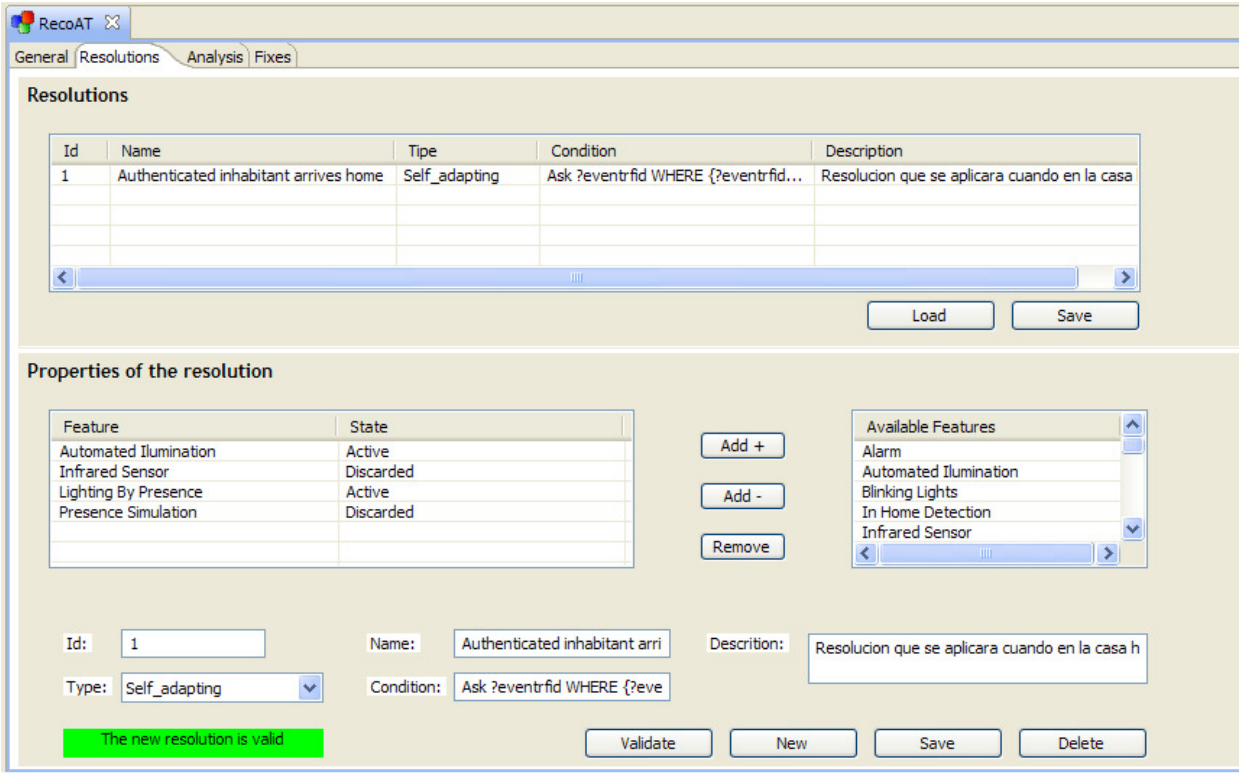


Figura 50. Pestaña *Resolutions* completada con una *Resolucion 1*.

A continuacion vamos a ver las descripciones de todas las resoluciones resultantes del analisis de la casa inteligente usada para nuestro caso de estudio. Las cuales usaremos en los siguientes pasos del estudio.

Resolucion 1:

Nombre: Authenticated inhabitant arrives home			
Id: 1	**Tipo:** Self_adapting		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Carlos\\) && REGEX(?event, \\add\\))	Resolución que se aplicaría cuando en la casa hay algún habitante.	Lighting By Presence	Active
		Automated Ilumination	Active
		Presence Simulation	Discarded
		Infrared Sensor	Discarded

Resolucion 2:

Nombre: Inhabitant leave the house			
Id: 2	**Tipo:** Self_adapting		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Carlos\\) && REGEX(?event, \\remove\\))	Resolución que se aplicaría cuando la casa esta vacía.	Lighting By Presence	Discarded
		Automated Ilumination	Discarded
		Presence Simulation	Active
		Infrared Sensor	Active

Resolucion 3:

Nombre: iPod Source			
Id: 3	**Tipo:** Self_Configuration		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\iPod\\) && REGEX(?event, \\add\\))}	Resolución que se aplicaría cuando la fuente de sonido *Ipod* estuviese en conectada.	iPod Source	Active

Resolucion 4:

Nombre: iPod disconected			
Id: 4	**Tipo:** Self_Configuration		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\iPod\\) && REGEX(?event, \\ remove \\))}	Resolución que se aplicaría cuando la fuente de sonido *Ipod* estuviese en desconectada.	iPod Source	Discarded

Resolucion 5:

Nombre: A sensor is providing data			
Id: 5	**Tipo:** Self_healing		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Segurity\\) && REGEX(?event, \\add\\))}	Resolución que se aplicaría cuando el sensor esta proporcionando datos.	In Home Detection	Discarded
		Volumetric Sensor	Discarded

Resolucion 6:

Nombre: The alarm is not working			
Id: 6	**Tipo:** Self_healing		
Condición:	**Descripción:**	**Configuración:**	
Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Segurity\\) && REGEX(?event, \\remove\\))}	Resolución que se aplicaría cuando la alarma no esta funcionando bien.	Blinking Lights	Active

Resolucion 7:

<table>
<tr><td colspan="4">Nombre: Infrared detector is replaced</td></tr>
<tr><td>Id: 7</td><td colspan="3">Tipo: Self_Configuration</td></tr>
<tr><td>Condición:</td><td>Descripción:</td><td colspan="2">Configuración:</td></tr>
<tr><td rowspan="2">Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Segurity\\) && REGEX(?event, \\add\\))}</td><td rowspan="2">Resolución que se aplicaría cuando el detector de infrarrojos se esta remplazando.</td><td>Infrared Sensor</td><td>Discarded</td></tr>
<tr><td>Volumetric Sensor</td><td>Active</td></tr>
</table>

Resolucion 8:

<table>
<tr><td colspan="4">Nombre: New presence detector sensor is plugged</td></tr>
<tr><td>Id: 8</td><td colspan="3">Tipo: Self_Configuration</td></tr>
<tr><td>Condición:</td><td>Descripción:</td><td colspan="2">Configuración:</td></tr>
<tr><td>Ask ?eventrfid WHERE {?eventrfid pros:name ?name ;pros:event ?event .FILTER (REGEX(?name, \\Segurity\\) && REGEX(?event, \\remove\\))}</td><td>Resolución que se aplicaría cuando se ha conectado un nuevo detector de presencia.</td><td>Volumetric Sensor</td><td>Active</td></tr>
</table>

6.4 Calculo del espacio de posibilidades

En esta sección se va ha explicar como se realizaría el cálculo de posibilidades para el caso de estudio en el que se esta trabajando.
Una vez están cargados y validados el modelo de características y la lista de resoluciones asociada a este, el siguiente paso es calcular las todas las posibles posibilidades resultantes de combinar estos dos.
Para realizar este paso con la aplicación, se desplegaría la pantalla *Analysis,* y una vez en esta se presionaría el botón *Calculate posibilites* instantáneamente aparecería una lista con todas las posibilidades.
Si seleccionamos una de las posibilidades, en es desplegable *Possibility Info* se puede consultar los datos de la posibilidad. Tanto la lista de posibilidades como la configuración final de combinar la configuración final con la lista de resoluciones.

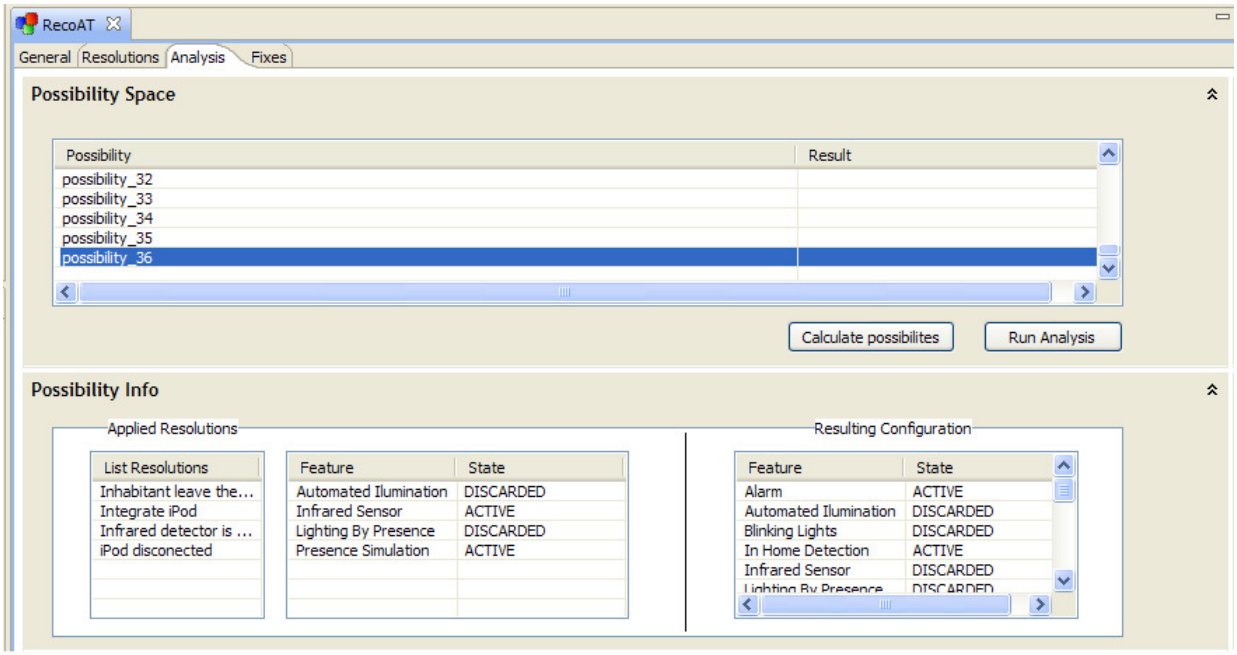


Figura 51. Pestaña *Analysis* con las posibilidades calculadas.

Se puede observar en la figura anterior cual es el resultado final del calculo de posibilidades del caso de estudio que se esta llevando a cabo. Podemos ver como se han creado 36 posibilidades diferentes, con las cuales posteriormente se creara el espacio de posibilidades del estudio.
También podemos observar los detalles de la posibilidad nº 36, que esta seleccionada en la (figura 51).

Se puede observar en la *List Resolutions* como las resoluciones que forman la posibilidad junto con la configuración inicial son las siguientes: *Inhabitant leave the house, Integrate ipod, Infrared detector is replaced* y *iPod desconected.*

En la siguiente tabla están listadas las características con su respectivo estado, de la primera resolución de la posibilidad seleccionada.

Por ultimo en la última tabla *Resulting Configuration,* se ve la configuración resultante de combinar las cuatro resoluciones pertenecientes a la posibilidad seleccionada con la configuración actual del modelo de características.

Esta configuración es la que se almacena asociada a la posibilidad, y será la que validaremos posteriormente contra el modelo de características para conocer la validez o no validez de esta posibilidad

6.5 Validación de las posibilidades

Una vez se tienen todas las posibilidades calculadas, el siguiente paso es mandar a validar cada posibilidad individualmente al analizador FAMA. A cada posibilidad se le aplicara la configuración inicial y se obtendrá su propia configuración, la cual el analizador validara.
Para realizar esto con la aplicación se tiene que presionar el botón *Run Analysis*, y cuando se haya completado el análisis de todas las posibilidades, aparecerá en la columna de *Result* el estado de cada posibilidad.
En la siguiente figura vemos las cinco últimas posibilidades del caso de estudio con sus respectivos estados, después de realizar la validación.

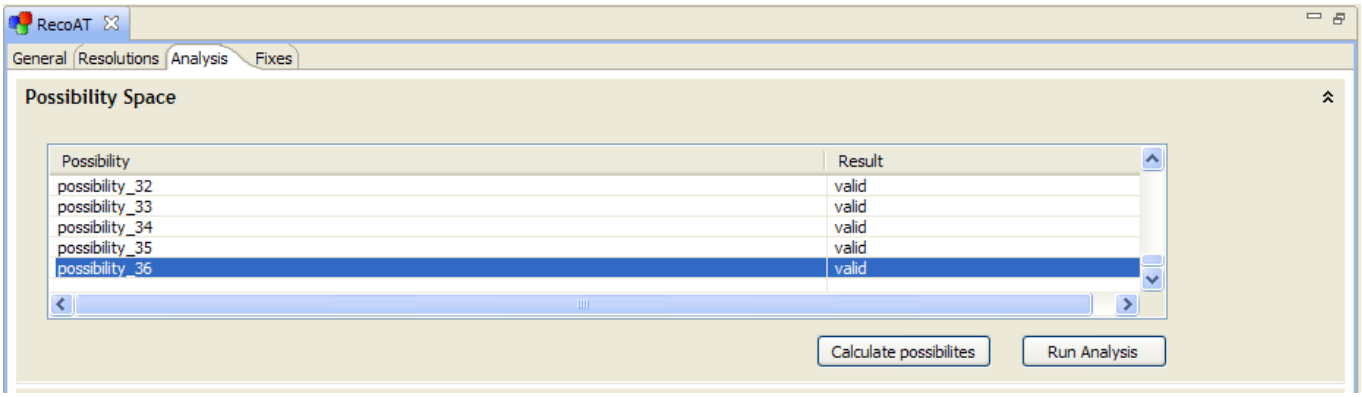


Figura 52. Pestaña *Analysis* con las posibilidades validas.

A la vez que se realizan las validaciones, la aplicación genera un informe donde podemos ver aparte de datos del modelo cargado, las posibilidades totales generadas y un desglose de las validas y las invalidas.
Se puede observar el informe generado para el caso de estudio en la figura siguiente.

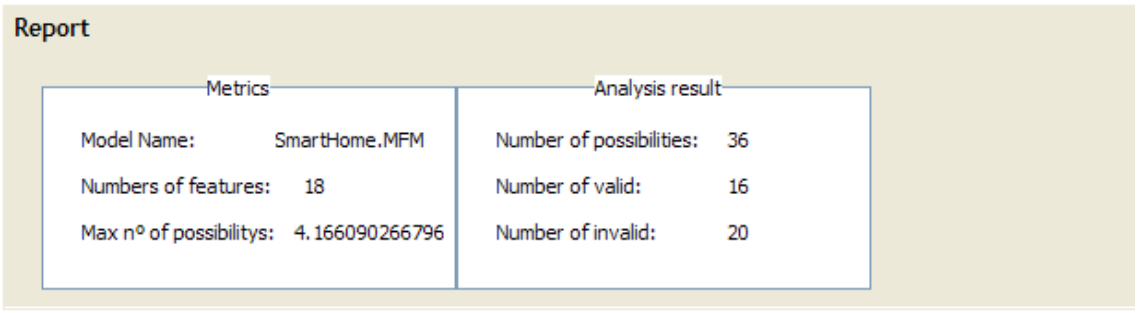


Figura 53. Informe generado por la pestaña *Analysis*.

6.6 Análisis del espacio de posibilidades

En esta sección vamos a analizar el espacio de posibilidades resultante del caso de estudio. Nos centraremos en la estructura y en las transiciones entre estados con diferente validez, lo que luego nos permitirá realizar optimizaciones sobre el espacio de posibilidades

Con el espacio de posibilidades obtenemos una simulación en la etapa de diseño del comportamiento que tendría el sistema de la casa inteligente que se ha estado analizando durante todo el caso de estudio. Esta simulación será la que nos permitida refinar el modelo de características, para que sea fiable ante cualquier circunstancia.

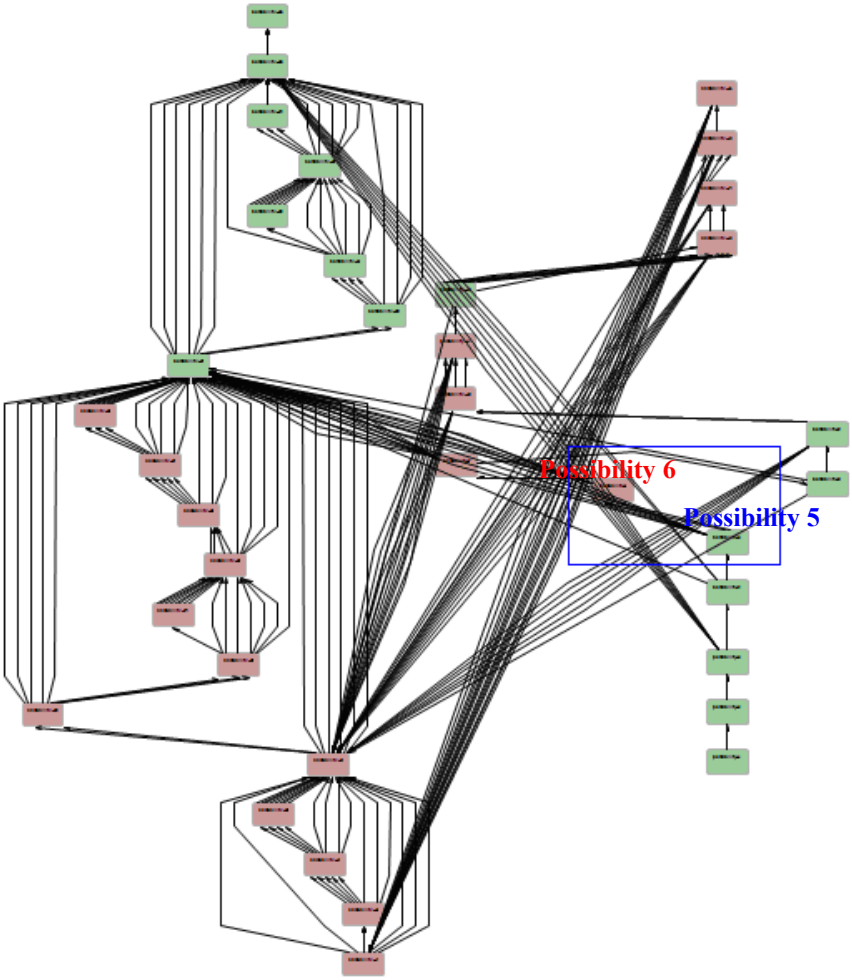


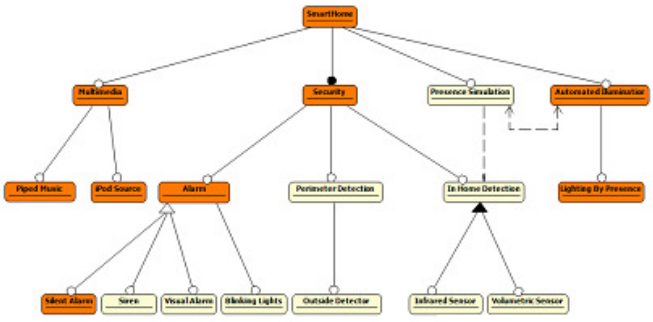
Figura 54. Espacio de posibilidades resultante del caso de estudio.

Podemos observar como en la figura anterior estamos ante un espacio de posibilidades, con 36 posibilidades de las cuales tenemos 16 validas y 20 invalidas.
Lo más interesante es como se distribuyen estas posibilidades validas e invalidas en el espacio de posibilidades. Se puede ver como las posibilidades con misma validación tienden a agruparse, con lo cual tenemos un diagrama con subgrupos de posibilidades con la misma validación. Esta característica nos será útil para crear el espacio de posibilidades abstracto que se vera en el siguiente apartado.
Pero para nuestro proyecto lo que nos interesan son las transiciones entre posibilidades de diferente validación. Por esa razón nos vamos a centrar en la transición entre la posibilidad 5 (que es valida) y la posibilidad 6 (que es inválida). En la siguiente figura podemos ver una comparación de ambas posibilidades, donde podemos ver tanto su configuración como su representación en diagrama de características.

Possibility 5 (Configuration)	
SmartHome	Active
Multimedia	Active
Piped Music	Active
iPod Source	Active
Segurity	Active
Alarm	Active
Silent Alarm	Active
Siren	Discarded
Visual Alarm	Active
Blinking Light	Discarded
Perimeter Detection	Active
Outside Detector	Discarded
Presence Simulation	Discarded
Automated Ilumination	Discarded
In Home Detection	Discarded
Infrared Sensor	Discarded
Volumetric Sensor	Discarded
Automatic Ilumination	Discarded
Lighting By Presence	Discarded

Possibility 6 (Configuration)	
SmartHome	Active
Multimedia	Active
Piped Music	Active
iPod Source	Active
Segurity	Active
Alarm	Active
Silent Alarm	Active
Siren	Discarded
Visual Alarm	Active
Blinking Light	Discarded
Perimeter Detection	Active
Outside Detector	Discarded
Presence Simulation	Active
Automated Ilumination	Discarded
In Home Detection	Discarded
Infrared Sensor	Active
Volumetric Sensor	Discarded
Automatic Ilumination	Discarded
Lighting By Presence	Discarded

Modelo características P.5.



Modelo características P.6.

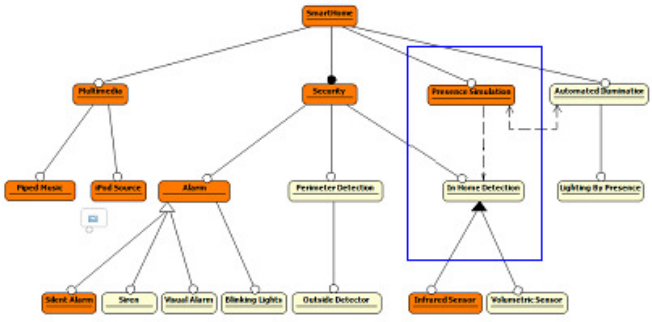


Figura 55. Comparación entre posibilidad 5 y posibilidad 6.

En las siguientes formulas vamos a ver las resoluciones incluidas en cada posibilidad:

Posibilidad 5 : ***{Inhabitant leave the house, Integrate iPod, Authenticate inhabitat, A sensor is Providing data}***

Posibilidad 6 : ***{Inhabitant leave the house, Integrate iPod, Authenticate inhabitat, A sensor is Providing data, Inhabitant leave the house}***

Se puede observar que la resolución que hace transitar de una Posibilidad a otra es *Inhabitant leave the house,* pese a estar repetida, se puede ver que esta en la primera posición también, pero ahora al estar en la ultima posición se trasladan todas las condiciones a la configuración final. Por ello vamos a ver en detalle las condiciones de esta condición a continuación

Resolution (Inhabitant leave the house): ***{(Lighting By Presence, Discarted), (Automated Ilumination, Discarted), (Presence Simulation , Active), (Infrared Sensor, Active) }***

Ya vistos los detalles de la transición entre la P5 y P6 vamos a ver porque la posibilidad 6, es errónea. Podemos ayudarnos de la figura 55 donde en la parte derecha inferior se puede ver el diagrama de características de dicha posibilidad.
Se puede observar, marcado con un cuadro azul, como se incumple la condicion de inclusión entre *Presence Simulation* y *In Home Detection.* . Dado que la característica *Presence Simulation* esta activada, para que se cumpliera la restricción la característica *In Home Detection* también debería de estar activada. Como se puede observar no lo esta, por lo cual se invalidaría la posibilidad.
Gracias al espacio de posibilidades generado se pueden encontrar estas incompatibilidades entre componentes, que se generarían en tiempo de ejecución en casas inteligentes

6.7 Espacio de posibilidades al espacio de posibilidades abstracto

En el siguiente apartado se van a explicar como se ha pasado del espacio de posibilidades al espacio de posibilidades abstracto. Para facilitar la comprensión de esta transformación nos ayudaremos de dos figuras: una el espacio de posibilidades agrupado y el espacio de posibilidades abstracto resultante.
A en la siguiente figura tenemos el espacio de posibilidades de caso de estudio agrupando las posibilidades en grupos. Estos grupos tienen la particularidad que todas sus posibilidades tienen la misma validación.

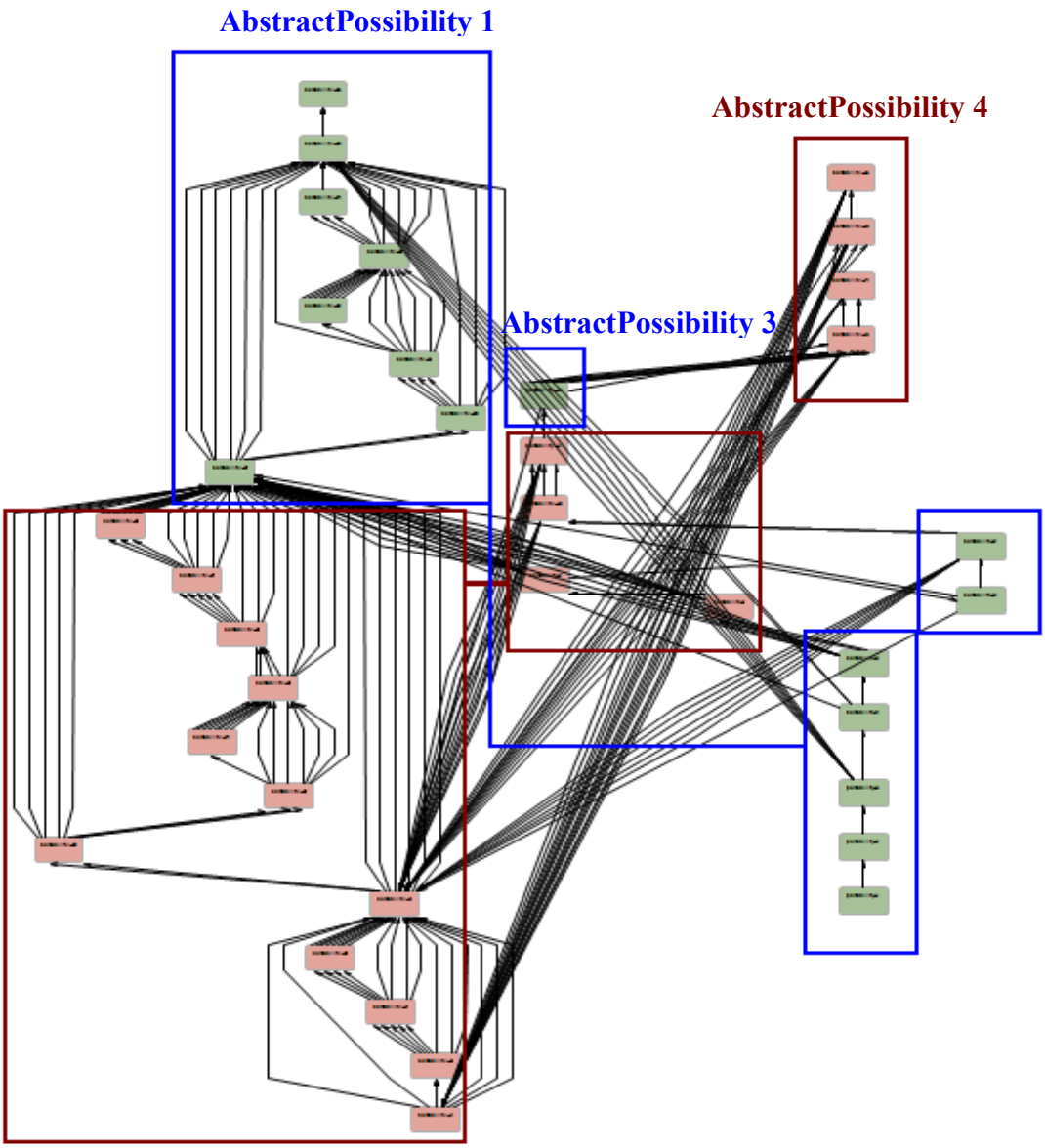


Figura 56. Espacio de posibilidades agrupado.

Para obtener el espacio de posibilidades abstracto, partiendo de la posibilidad inicial, se recorren todas las posibilidades accesibles mediante transiciones hasta encontrar una posibilidad con estado diferente. Todas estas posibilidades se agrupan y se crea una posibilidad en el nuevo diagrama, con la misma validación que las anteriores. Así hasta haber recorrido todo el espacio de posibilidades, y tener todas sus posibilidades agrupadas en nuevas posibilidades en el nuevo diagrama.

En el nuevo espacio de posibilidades mantenemos todas las transiciones del anterior, pero es más fácil de detectar las transiciones que nos interesan, las que van de estados validos a inválidos y nos invalidan el modelo de características.
Como vemos tenemos 3 transiciones que van de estados validos a inválidos: dos desde la posibilidad 1 a la posibilidad 2 y una de la posibilidad 3 a la posibilidad 4.
Por ultimo vamos a ver como la transición de la posibilidad 5 a la posibilidad 6 del espacio de posibilidades, explicada en el apartado anterior. A pasado a ser una de las dos posibilidades que transitan de a posibilidad 1 a la posibilidad 2.

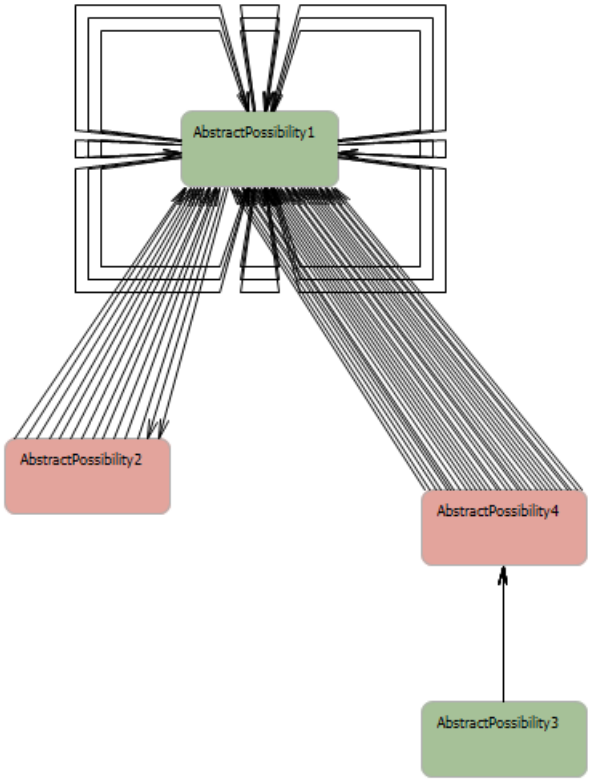


Figura 57. Espacio de posibilidades abstracto del caso de estudio.

6.8 Uso del espacio de posibilidades abstracto

En esta última sección del caso de estudio, vamos a trabajar con el espacio de posibilidades abstracto, para intentar eliminar de nuestro espacio de posibilidades las inválidas.
En la siguiente figura vemos el espacio de posibilidades abstracto, el resultado final de todo el proceso. En el de han marcado en azul las tres transiciones que nos llevan a posibilidades invalidas.

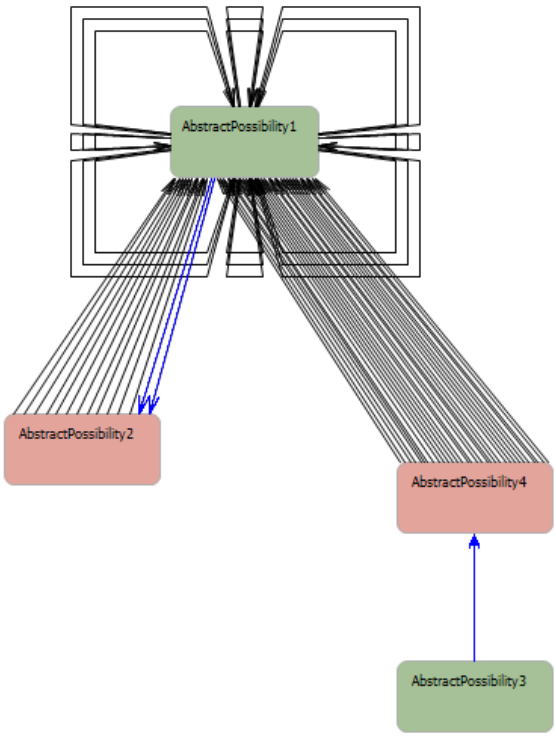


Figura 58. Espacio de posibilidades abstracto con tradiciones marcadas.

Como se vio en el capitulo anterior la aplicación RecoAT permite aplicar unos *Fixes* que ayudan a eliminar las resoluciones que provocan posibilidades invalidas. Con estos *Fixes* se pueden eliminar estas resoluciones o marcar para luego modificarlas.
En este ejemplo supondremos que se ha usado el *Fixe* de marcar las resoluciones que producen conflictos y la primera resolución que produce conflictos en la *Inhabitant leave the house,* la que vimos en el apartado 6.4 que provocaba la primera transición de posibilidades validas a invalidas

Resolution (Inhabitant leave the house): ***{(Lighting By Presence, Discarted), (Automated Ilumination, Discarted), (Presence Simulation , Active), (Infrared Sensor, Active) }***

Vamos a analizar el modelo de características asociado a la configuración de la posibilidad 6 y vamos a modificar la resolución *Inhabitant leave the house* para evitar que esta resolución vuelva a provocar resoluciones invalidas.
Como se explico en el apartado 6.4, esta resolución provoco que se generasen configuraciones invalidas, porque incluye la condición *(Presence Simulation , Active).*
La cual esta en una relación de inclusión con *In Home Detection*. Y la relación no asegura que ambas dos condiciones se den a la vez. Así que es una resolución mal diseñada, porque puede inducir a que se incumplan condiciones del modelo de características.
Por esta razón vamos a completar la resolución con la condición (*In Home Detection, Active)* lo cual nos asegurara, que al aplicar la resolución estas dos condiciones irán siempre juntas. Por ultimo vemos como quedaría la resolución después de revisarla

Resolution (Inhabitant leave the house): ***{(Lighting By Presence, Discarted), (Automated Ilumination, Discarted), (Presence Simulation , Active), (Infrared Sensor, Active),* (*In Home Detection, Active) }***

Este proceso se seguiría con las otras 2 transiciones que conducen a posibilidades inválidas. Una vez modificadas todas las resoluciones que provocan que se incumpla alguna condición del modelo. Se a ejecutar el espacio de características, y si este saliese solo con posibilidades validas, se habría conseguido sistema para gestionar Smart Homes fiable y que se reconfigure ante diferentes estímulos externos.

7. Conclusiones

Como conclusión al proyecto que se ha desarrollado, se puede decir que se han cumplido los objetivos y solucionado los problemas que se plantearon al comienzo de este. El objetivo del proyecto era diseñar y validar sistemas para Smart Homes de forma sencilla sin tener que desarrollar completamente el sistema real.

Para ello se ha desarrollado un método, por el cual los sistemas se diseñan mediante modelos de características. Esto nos permite una vez se ha terminado el diseño del sistema, simular el comportamiento de este sistema como si estuviese en funcionamiento. Gracias al cálculo del espacio de posibilidades del modelo del sistema, se pueden detectar posibles fallos que el sistema tendría en ejecución.

Una vez detectados fallos o conflictos en el sistema, se intentaran solucionar antes de que el sistema se ejecute en un sistema real. Así se obtendrá un sistema que cuando llegue a ejecución será capaz de reconfigurarse según diferentes condiciones de contexto.

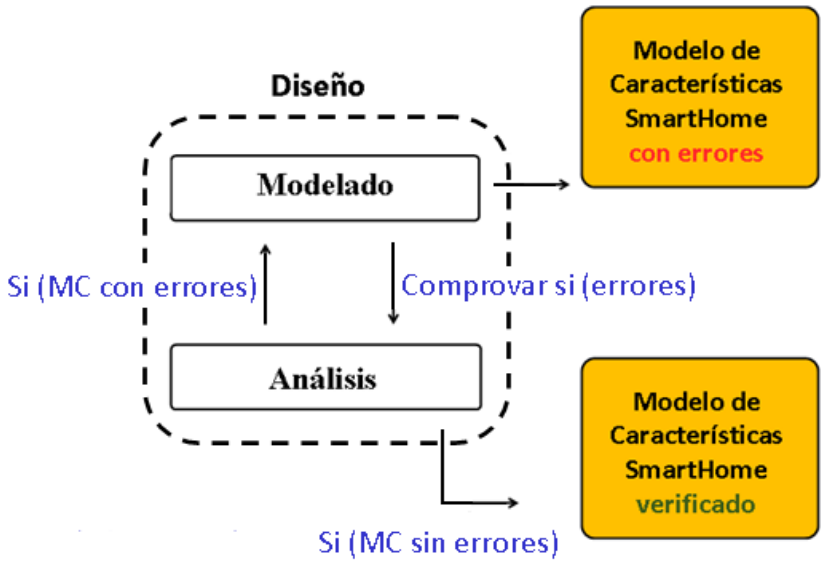


Figura 59. Esquema método de diseño.

Como el número de configuraciones puede ser muy alto, identificar los posibles problemas puede no ser posible con modelos tan complejos. Para ello se decidió calcular espacios de posibilidades optimizados para la tarea que se quería realizar, así que partiendo de los espacios de posibilidades, se crean nuevos espacios de posibilidades abstractos. Estos nuevos diagramas agrupan las posibilidades sin transiciones conflictivas, lo cual deja diagramas mucho más manejables, dado que nos calcula solo los estados y las transiciones que nos pueden provocar conflictos una vez en ejecución. Así que se desarrollado un método asistido por la aplicación RecoAT, que nos permite la creación de sistemas para Smart Homes fiables.

Después de haber realizado un caso de estudio completo, visto en el capitulo 6, en el que se han utilizado modelos que serian aplicables a hogares inteligentes reales. Y siendo el resultado de este estudio satisfactorio en todos los aspectos analizados, podemos decir que mediante el uso de este método se crearan sistemas para Smart Homes de forma más rápida y fiable.

También habría que destacar de este proyecto la buena experiencia obtenida del trabajo con modelos. El uso de modelos para este tipo de desarrollos podría parecer una cosa un poco conceptual. Pero en este proyecto se ha demostrado que pueden ser muy útiles para el modelado, validado y análisis de hogares inteligentes reales.

Pero el uso de este método y de la aplicación RecoAT se podría extrapolar a otros ámbitos, modificando los diagramas de contexto. Podríamos aplicar este método de diseño a la industria automovilística, al diseño de sistemas industriales, etc.

8. Referencias bibliográficas.

[1] 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.

[2] Kang, K.C. and Cohen, S.G. and Hess, J.A. and Novak, W.E. and Peterson, A.S., "Feature-oriented domain analysis (FODA) feasibility study", Technical Report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990

[3] D. Benavides, P. Trinidad and A. Ruiz-Cortés. The 17th Conference on Advanced Information Systems Engineering (CAiSE'05): "Automated Reasoning on Feature Models". Porto, Portugal. 2005.

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

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

[6] http://www.isa.us.es/fama

[7] http://www.moskitt.org

[8] http://www.eclipse.org/