CONVERSOR DE CVL A MODELO FAMA

CÓDIGO P.F.C. : DSIC-84

Memoria

Javier de la Fuente Sales

1. Introducción

1.1.Motivación

Durante los últimos años los sistemas software cada vez tienden a soportar una mayor variabilidad es decir la cualidad de variar o de poder variar. Esta variabilidad se debe principalmente a dos causas:

- A las demandas de los usuarios de sistemas más adaptables a sus necesidades (sistemas altamente configurables) .

- A la presión del mercado, que hace que los productos software se agrupen en familias para poder reutilizar la mayor parte posible de sus artefactos software (familias de productos software) .

De esta manera, el campo de la variabilidad está obteniendo una creciente importancia.

La variabilidad en los sistemas puede ser expresada con el modelo de variabilidad (Variability Model) estos permiten definir la variabilidad de una línea de productos, mostrando los puntos de variación de esta línea. Estos modelos pueden ser utilizados en cualquiera de la distintas etapas del desarrollo software siendo la incorporación de este al desarrollo software tradicional una tarea pendiente.

Durante este proyecto se hablará de dos de sus implementaciones, modelos de características (Feature Model) y modelos CVL que está en proceso de estandarización con la OMG (organización sin ánimo de lucro que promueve el uso de tecnología orientada a objetos mediante guías y especificaciones para las mismas).

Actualmente, existe una herramienta muy potente de análisis de modelos de características llamada FaMa (Análisis automático de modelos). Este es un Framework (conjunto de clases ya implementadas) que analiza estos modelos y permite al usuario realizar preguntas a este que le indiquen si es un modelo válido, cuantos productos tiene, si tiene algún error entre otras preguntas que a continuación se explicaran y se ejemplificaran.

Como consecuencia del auge y necesidad de análisis de estos modelos, y debido tanto a la existencia de un leguaje de modelado tan potente como es CVL en proceso de estandarización y de una herramienta de análisis completamente implantada y en funcionamiento, por todos estos motivos se ha decidido elaborar este proyecto.

1.2.Objetivos

El objetivo principal es elaborar un plugin para eclipse, que añadido al plugin ya existente de FaMa permita analizar modelos CVL como actualmente ya se puede hacer

con los modelos de características. Los objetivos que se han planteado son los siguientes:

- Permitir que la herramienta FaMa pueda realizar el análisis de modelos CVL.

- Realizar la transformación de los modelos CVL, siendo esta transparente para el usuario.

- Interpretar resultados obtenidos con el análisis FaMa.

2. Contexto tecnológico

En este apartado se hablara sobre las distintas tecnologías utilizadas en el transcurso del proyecto. Hablaremos del leguaje de implementación, de la plataforma sobre la que se ha realizado, de los dos tipos de modelos sobre los que se ha trabajado y de los lenguajes de estos.

2.1.Java

Lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Vale la pena destacar tres de sus principales características:

- La orientación a objetos se basa en diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables
- Independencia de plataforma significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo.
- El recolector de basura el problema de las fugas de memoria se evita en gran medida gracias a la recolección de basura. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto.

2.2.Common Variability Language (CVL)

Common Variability Language es un genérico y separado lenguaje para modelar los modelos de variabilidad en cualquier dominio especifico del lenguaje (DSL) el cual este basado en Meta Object Facility (MOF). Surge debido a la necesidad de un lenguaje genérico para poder expresar la variabilidad en un modelo.

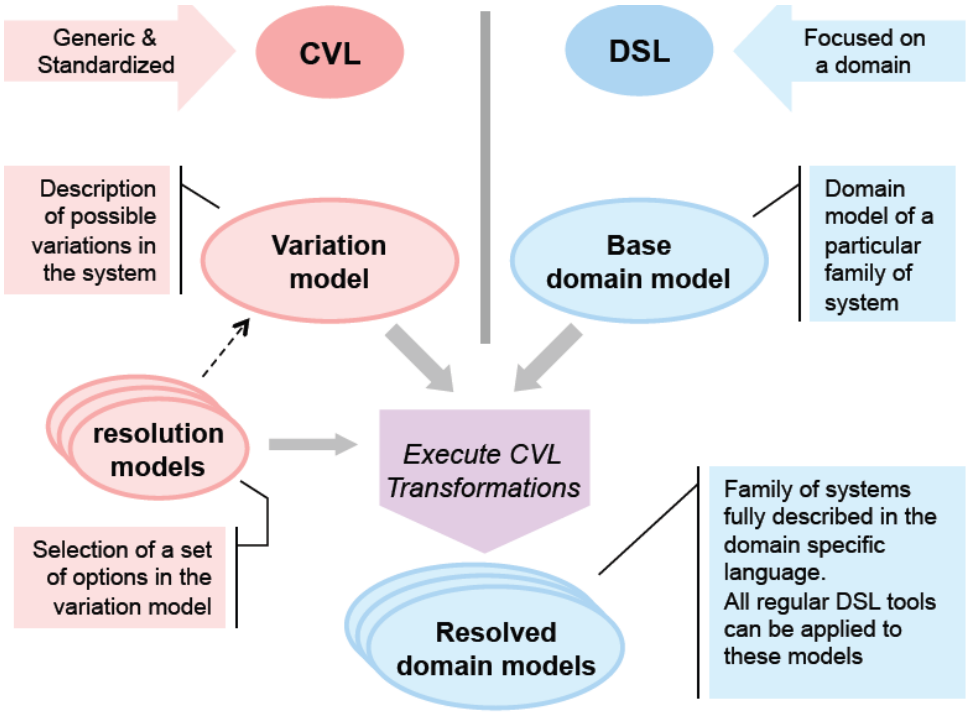
En CVL tenemos tres modelos:

- Modelo base: un modelo descrito en DSL.

- Modelo de variabilidad: define la variabilidad en el modelo base.

- Modelo de resolución: define la manera de resolver el modelo de variabilidad para crear un nuevo modelo en el DSL base.

La imagen que se muestra a continuación es interesante ya que se puede ver la interacción entre los tres modelos explicados anteriormente.



Un modelo base puede tener varios modelos de la variabilidad y cada modelo de variabilidad puede tener varios modelos de resolución. Con el modelo de la variabilidad y el modelo de resolución adecuadamente definidas, el usuario puede ejecutar genéricas transformaciones CVL de modelo a modelo para generar nuevos modelos resueltos que se ajusten a el DSL base.

2.3.Domain Specific Language (DSL)

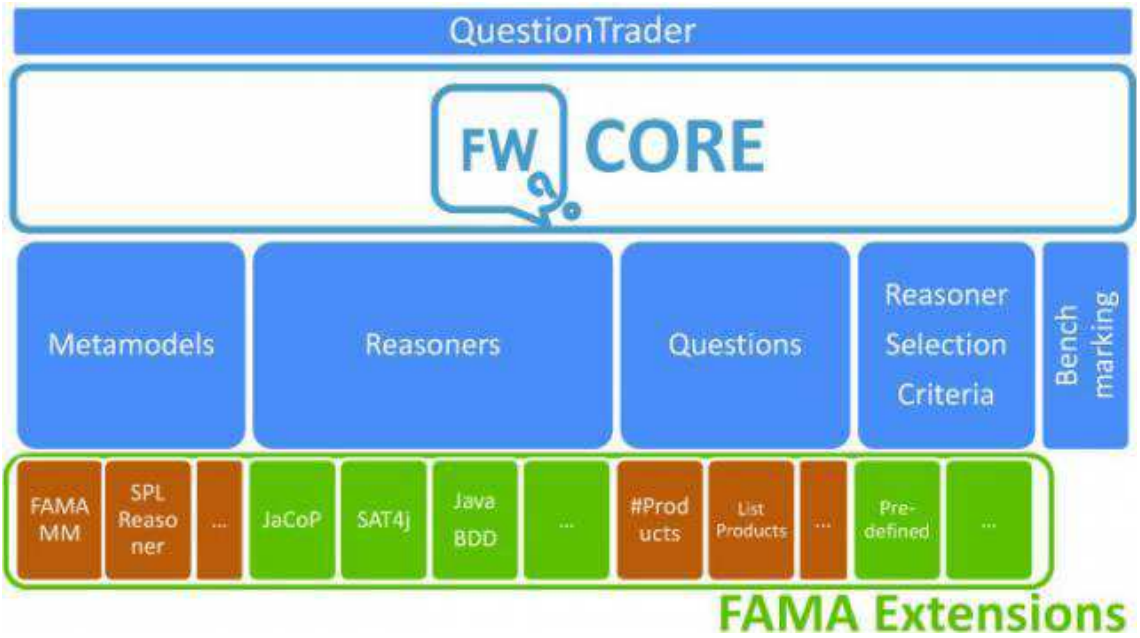
Se considera como DSL cualquier lenguaje que esté especializado en modelar o resolver un conjunto específico de problemas. Este conjunto específico de problemas es el llamado dominio de aplicación o de negocio. La mayor parte de los lenguajes de programación no se pueden considerar DSL ya que no están diseñados para resolver un conjunto específico de problemas, sino para resolver cualquier tipo de problema.

2.4.Análisis Automático de Modelos de Características (FaMa)

Herramienta desarrollada por un equipo de la Universidad de Sevilla, liderado por David Benavides. Permite el análisis automatizado de los modelos de características integrando algunos de los resolutores más común-mente propuestos (BDD, SAT y CSP), siendo esta una característica poco común. Dispone de un plugin de Eclipse gráfico desarrollado en EMF bajo licencia Open-Source .

Siendo la primera herramienta que integra soluciones para el análisis automático de modelos de características.

La arquitectura de FaMa es la siguiente:



2.5.BDD

En el campo de la informática, un diagrama de decisión binaria (BDD) es una estructura de datos que se utiliza para representar una función booleana. En un nivel más abstracto, BDDs puede ser considerado como una representación comprimida de los conjuntos o las relaciones. A diferencia de otras representaciones comprimido, las operaciones se realizan directa-mente en la representación comprimida, es decir, sin descompresión.

2.6.SAT

Consiste en decidir cuando una fórmula proposicional (expresión consistente en un conjunto de variables booleanas conectados por operadores lógicos) dada es satisfacible, por ejemplo, unos valores lógicos pueden ser asignados a sus variables de una forma que haga la fórmula verdadera.

2.7.CSP

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

2.8.Entorno Eclipse

Es un entorno de desarrollo integrado (programa informático compuesto por un conjunto de herramientas de programación) de código abierto multiplataforma para desarrollar software .

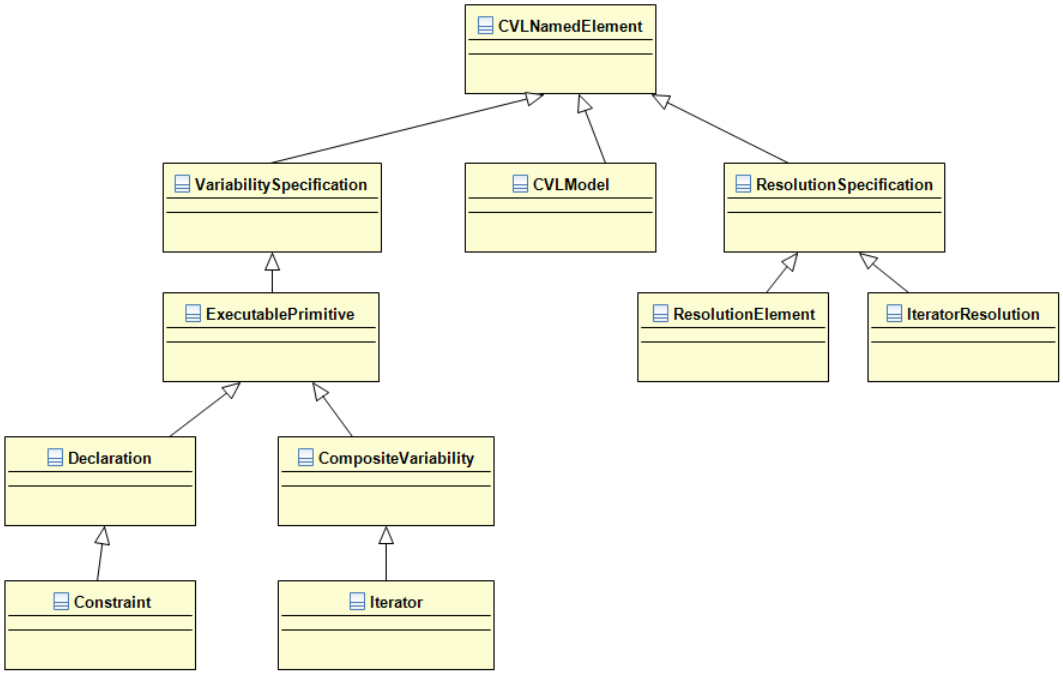
Concretamente , se ha utilizado a lo largo de este proyecto Moskitt que es una herramienta CASE LIBRE, basada en Eclipse que está siendo desarrollada por la Conselleria de Infraestructuras y Transporte (CIT). Destaca por su arquitectura de plugins que la convierte no sólo en una Herramienta CASE sino en toda una Plataforma de Modelado de Software Libre para la construcción de este tipo de herramientas.

3. Traducción de modelos CVL a modelos FaMa

3.1.Conceptos de los modelos CVL

Toda clase de CVL como cualquier clase en un lenguaje orientado a objetos hereda de otra y si se asciende en esta herencia se llega a object.

El siguiente esquema muestra la herencia de los distintos elementos de CVL que son relevantes para la comprensión del proceso de transformación que se explicara a continuación, no se muestra toda la herencia hasta object porque no es necesario para la explicación.



El primer elemento **CVLNamedElement** , de el heredan el resto de elementos que componen el leguaje CVL, todos sus atributos y métodos serán accesibles por el resto, como la herencia indica. Esta clase contiene el atributo name, que será accedido a el repetidas veces a lo largo del proceso de transformación.

De CVLNamedElement cuelgan tres elementos:

CVLModel es la clase que representa al modelo en si. De el cuelgan tanto el modelo de variabilidad, como las resoluciones. Está formado por numerosos atributos y métodos, de ellos vale la pena recalcar dos debido a su uso repetido en el proceso de transformación.

- getVariabilitySpecification() -> Devuelve el primer elemento del modelo de variabilidad.

- getResolutionSpecification() -> Devuelve una lista con todas las resoluciones del modelo

VariabilitySpecification es la clase que representa al modelo de variabilidad, la subclases que heredan son los elementos con los que se construye el modelo.

Como subclases de VariabilitySpecification: esta **ExecutablePrimitive** que es utilizado para especificar sobre que elementos actúa una restricción. Mas abajo heredando de esta se tiene **CompositeVariability** y **Declaration**.

CompositeVariability elemento principal del modelo, con el que se expresa la variabilidad del modelo, la selección de un conjunto de los CompositeVariability es una resolución. Sobrescribe el método:

- getVariabilitySpecification() -> Devuelve los elementos que se encuentran inmediatamente por debajo del CompositeVariability.

Del **CompositeVariability** hereda el **Iterator** otro de los elementos principales que indican la relación entre CompositeVariability. Además de todos los métodos que hereda del resto, incorpora dos nuevos atributos.

- Lista de tipo **CompositeVariability**, indican como su nombre indica que **CompositeVariability** tiene como destino.
- Cardinalidad , indica cuantos elementos de la lista pueden ser seleccionados.

También tiene un método:

- addDestination() que permite añadir elementos a esta lista.

Declaration engloba una serie de elementos que comparten una serie de características y métodos, de ellos el que interesa comentar el **Constraint** el tipo de elemento con el que se elaboran las restricciones. Tiene dos atributos que indican a que grupo de **CompositeVariability** afecta.

Por otro lado esta **ResolutionSpecification**, representa al propio modelo de resolución y de el heredan los dos elementos que componen la resolución.

ResolutionElement, es el equivalente al CompositeVariability en el Modelo de Variabilidad, consta de un atributo que indica a que CompositeVariability del Modelo de Variabilidad hace referencia. Tiene un método:

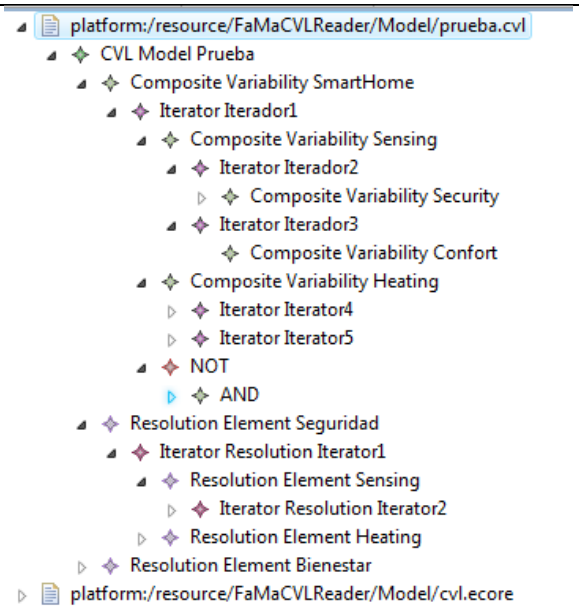
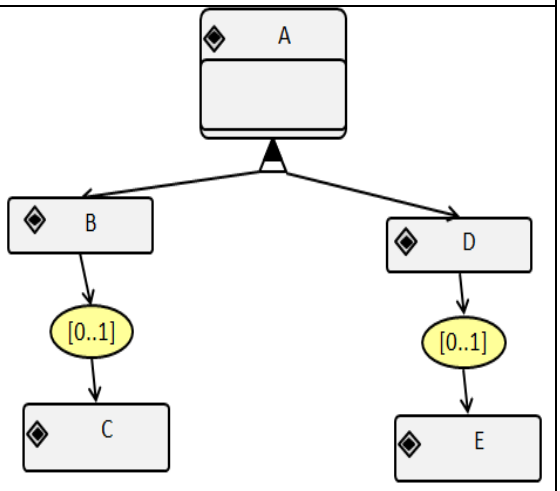
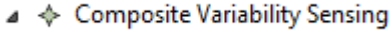
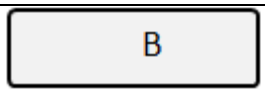
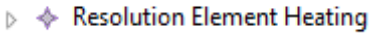
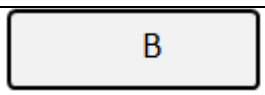
- getResolution() -> Devuelve los **IteratorResolution** que se encuentran inmediatamente por debajo.

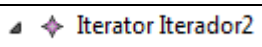
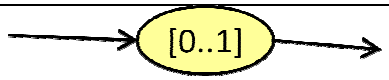
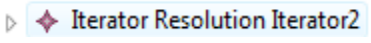
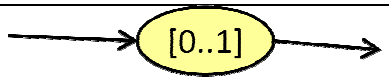
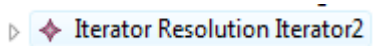
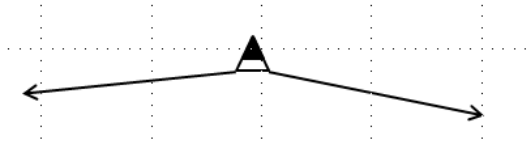
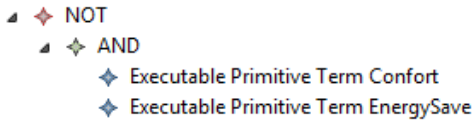
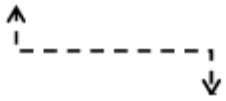
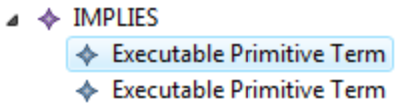
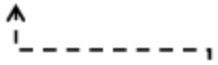
IteratorResolution, equivalente al elemento Iterator del Modelo de Variabilidad, tiene una variable que señala al Iterator hace referencia. Tiene un método similar al anterior:

- getChoice() -> Devuelve los ResolutionElement que se encuentran inmediatamente por debajo de el.

Todos los elementos de CVL que se utilizan a lo largo del proyecto, han sido explciados, ahora se verá su representación en el modelo.

Primero de todo, hay que diferenciar dos términos diagrama y modelo, un diagrama es la representación grafica de un modelos, creado con el objetivo de facilitar tanto la comprensión del modelo como su construcción.

<table>
<tr><td>Modedo CVL</td><td>Representación grafica de un Modelo CVL</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>CompositeVariability (Modelo)</td><td>(Diagrama)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td>ResolutionElement (Modelo)</td><td>(Diagrama)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td></td><td></td></tr>
</table>

Iterator (Modelo)	(Diagrama)
	
IteratorResolution (Modelo)	(Diagrama)
	
Su representación el modelo es igual a la anterior. 	Un iterador tambien se puede representar  Es un Iterator donde todos sus elementos pueden ser seleccionados. Si por ejemplo tiene dos asociados, su cardinalidad seria: [0...2].
Constraint (Restricción de exclusión) (Modelo)	(Diagrama)
	
Constraint (Restricción de inclusión) (Modelo)	(Diagrama
	

3.2.Conceptos de los modelos FaMa

Los modelos FaMa son mas sencillos, tiene dos tipos de modelos al igual que CVL el modelo de características y modelo de producto. Un modelod e cacrteristicas está formado por:

FeatureModel, es el propio modelo de características que contiene todos los elementos. Tiene tres atributos:

- name que identifica el nombre del modelo
- Un atributo con la **Feature** principal es decir la primera de todo el modelo
- Lista de **Dependency**.

Feature, elemento principal del modelo de cacracteristicas y de los productos.Consta de:

- Atributo llamado name que identifica la **Feature**.
- Una lista de **Relation** que tienen como origen esta **Feature**.

Relation, Otro de los elementos principales del modelo que indica las relaciones entre las distintos elementos **Feature** Consta de:

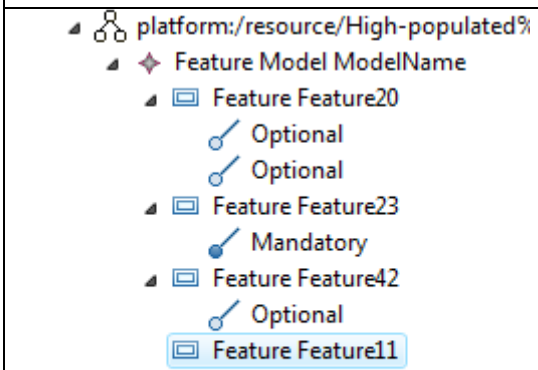
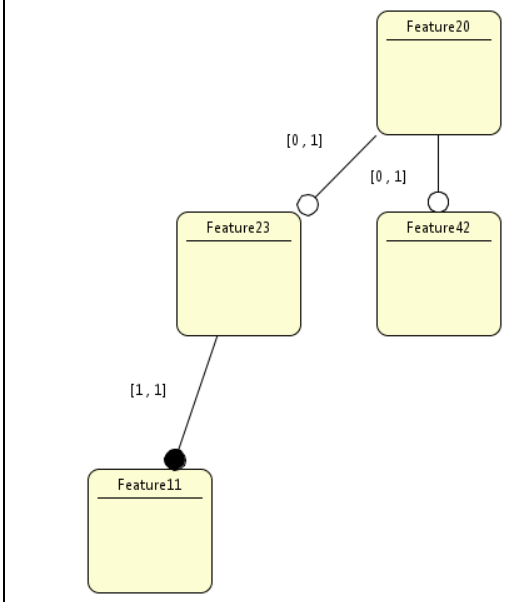
- Cardinalidad
- Lista de **Feature** destino.

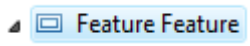
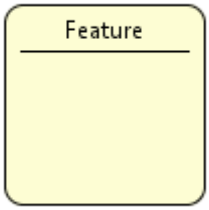
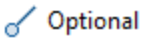
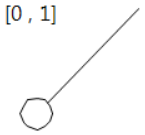
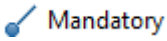
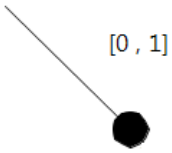
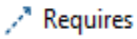
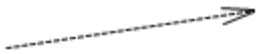
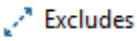
RequiresDependency, es una de las restricciones del modelo de características. Tiene dos atributos que indican sobre que dos elementos **Feature** actúa. Un atributo contiene la referencia a la **Feature** origen y otro a la destino, si la origen es seleccionada en un **Product**, el destino tiene que existir. Es una restricción de implicación, si uno existe tiene que existir el otro, pero no de forma inversa.

ExcludesDependency, otra de las restricciones del modelo de características, tiene dos atributos que indican sobre que elementos Feature actúa. No hay diferencia de entre los dos atributos, si cualquiera de los dos elementos Feature existe el otro no puede existir. Es una restricción de exclusión.

Product, es una instancia concreta de un modelo de características. Esta formado únicamente por elementos de tipo **Feature**.

Al igual que en CVL se tiene un modelo y una representación grafica, un diagrama.

Modelo	Diagrama
	

Feature (Modelo)	(Diagrama)
	
Relation	
Optional La selección de la Feature que une es opcional en cualquier producto (Modelo)	(Diagrama)
	
Mandatory La selección de la Feature que enlaza es obligatorio en cualquier producto (Modelo)	(Diagrama)
	
RequiresDependency (Modelo)	(Diagrama)
	
ExcludesDependency (Modelo)	(Diagrama)
	

3.3.Correspondencia entre modelos CVL y modelos de FaMa

CVL	**FaMa**	**OBSERVACIONES**
CVLModel	FeatureModel	La correspondencia nos es exacta y se pierden alguna información.

ResolutionEspecification	Product	Ambos cumplen la misma función y son muy similares.
CompositeVariability	Feature	Los CompositeVariability permiten atributos los Feature no, habría que bajar en la herencia hasta los FeatureAttribute incorporados en la ultima versión de Fama.
Iterator	Relation	Están implementados de forma distinta pero tienen la misma función y su transformación es exacta.
Constraint	Dependency	No son exactamente iguales las, Constraint de CVL permiten mas restricciones que las de FaMa.

3.4. Traducción entre modelos de variabilidad y modelos de características

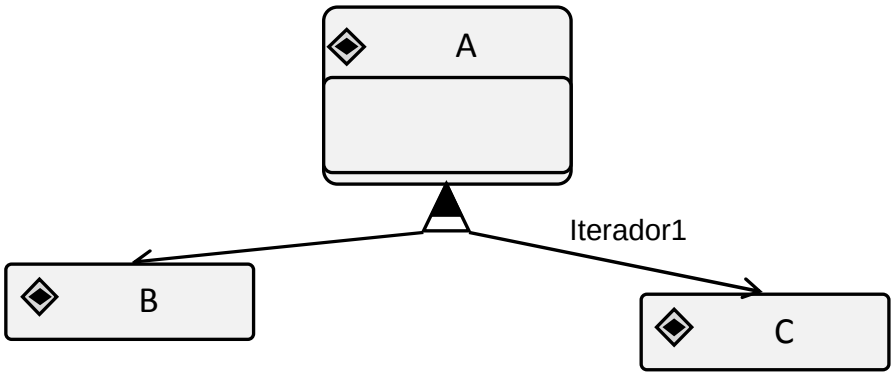
Para entender esta transformación primero es necesario recalcar algunos aspectos concretos del modelo CVL:

- Un elemento CompositeVariability se conecta con otro del mismo tipo a través de un elemento de tipo Iterator, no pueden conectarse sin este.

- El primer elemento de un modelo de variabilidad siempre es de tipo CompositeVariability.

- De igual modo el último elemento es de tipo CompositeVariability.

En primer lugar se explicara como se ha realizado la transformación con un ejemplo pequeño, después se explicaran los métodos implementados para ello y su funcionamiento, en último lugar se realizara una traza de como se producen las llamadas entre los métodos para la transformación de un modelo para el mismo ejemplo.



PASOS:

- Se tiene como entrada la ruta del modelo representado por el diagrama que se muestra arriba. Con esta ruta se carga el modelo obteniendo una variable de tipo CVLModel.

- Utilizando el método getVariabilitySpecification () de esta variable, se obtiene el primer elemento del modelo, que como se explica en el punto 2 de aspectos a recalcar del modelo, este será de tipo CompositeVariability. En este caso concreto este elemento es A.

- Este elemento tiene un método muy similar en funcionamiento y con el mismo nombre al mencionado en el punto 2, pero con la diferencia que este devuelve una lista de todos los elementos inmediatamente inferiores a este. Con ese método se obtiene una lista de VariabilitySpecification con un elemento que es Iterador1, sabemos que después de un CompositeVariability no puede ir otro elemento del mismo tipo, por lo que este solo puede ser o un Iterator o una Constraint. En este caso será un Iterator.

- Una vez que se sabe que es un Iterator, con el mismo método utilizado en el punto 3 se obtiene una lista con dos elementos B y C, habrá que comprobar si son CompositeVariability o Constraint, en este caso son CompositeVariability.

- Primero se pregunta al primer elemento de las lista que es B con el mismo método por lo siguientes elemento, pero devuelve una lista vacía, esto quiere decir que esta rama ha terminado.

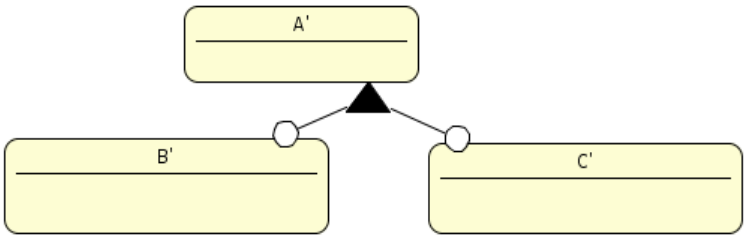
- Se pasa al segundo elemento de la lista que es C, se repite el mismo procedimiento del punto 5 y da el mismo resultado, esa rama también ha terminado.
- Se transforman B y C a sus equivalentes en el modelo FeatureModel, estos se transformaran en Features.

- Una vez convertido B y C hay que trasformar el iterator 1 en su equivalente en el FeatureModel que es una Relation. Al terminar esta transformación se añaden como Features destino de esta Relation B y C.

- Tras transformar todo los elementos posteriores a A, el primer elemento, se transforma este , igual que B y C y se le añade todas las Relation, en este caso solo una Iterador 1.

- Creamos un modelo FeatureModel vacío y se añade como root A.

El resultado obtenido es un modelo equivalente, cuya representación en un diagrama es:



Los métodos implementados para realizar la transformación expuesta anteriormente son los siguientes:

1.public GenericFeatureModel CVL2GenericFeatureModel(String cvlPPath)

Es el método encargado de encapsular el proceso de transformación, tiene como entrada la ruta de donde se encuentra el modelo, y como salida el FeatureModel es decir el modelo de FaMa equivalente al de CVL.

2.private GenericFeatureModel cvlModelToFeactureModel(String cvlPPath)

Se podrían diferenciar claramente dos fases en este método:

En la primera fase llama al método 3 pasándole directamente la ruta, coge el resultado de la llamada que es un CVLModel y con ayuda de los métodos ya implementados de esta clase busca el primer elemento del modelo, que por lo que se hablo anteriormente es un CompositeVariability.

En la segunda llama al método 4 pasándole el primer elemento encontrado, el resultado a la llamada al método 4 será el modelo trasformado FeatureModel.

3.private CVLModel getCvlModel(String cvlPPath)

Carga el modelo que se encuentra en la ruta que se le pasa como entrada y lo devuelve como salida.

4.private Feature convertCompositeVariability(CompositeVariability c)

Tiene como entrada un elemento CompositeVariability del modelo CVL y te devuelve un elemento Feature del modelo FaMa.

5.private Relation convertIterator(cvl.Iterator i)

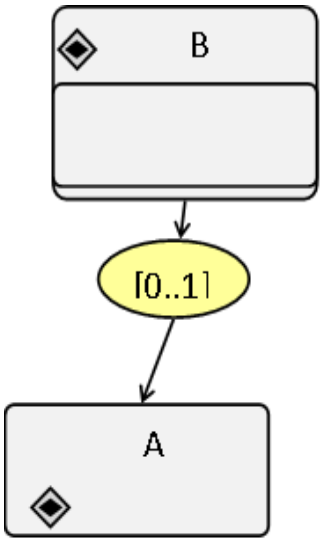
Similar al método anterior, pero en este caso se le pasa un iterator, tipo de elemento de CVL del cual se ha hablado en profundidad con anterioridad y devuelve como salida su equivalente en el modelo FaMa una Relation.

6.private FAMAFeatureModel process(CompositeVariability c)

En primer lugar crea un modelo FaMa vacío, después convierte el elemento que le pasan como entrada con el método 4. Luego llama al método 7 que devuelve una lista de relaciones. Añade estas relaciones al elemento transformado y agrega este elemento como root (Feature principal) al modelo FaMa. Por ultimo devuelve el modelo.

7.private ArrayList<Relation> processAux(CompositeVariability c)

Este método es uno de los mas complicados de entender, ya que es recursivo es decir se llama a si mismo un número indefinido de veces hasta llegar a la solución. Por este motivo su explicación se hará mediante el uso de un ejemplo.



- Se llama a esta función con la llamada processAux(B)
- Se busca los siguientes elementos de B que como se puede ver en la imagen solo hay uno, un iterator se le llamara iterador1.
- Se obtiene los elementos/elemento que están inmediatamente después del iterator, este es A.
- Se vueleve a llamar al propio método, pero esta vez con B la llamada será processAux(A), aun no se ha terminado la ejecución de processAux(B) pero esta tiene que esperar al resultado de processAux(A).
- Comienza a ejecutarse processAux(A), busca los elementos inferiores pero no encuentra ninguno así que nos devuelve una lista vacía de relaciones.

- Con el resultado obtenido, processAux(B) puede continuar

- Convierte A a una Feature obtiene A'

- Añade el resultado a un atributo de A' que contiene todas las relaciones que tiene, en este caso es una lista vacía, por lo que no añade ninguna, pero en otro caso podría ser distinto.

- Convierte el iterador1 a relación, obtenemos iterador1'

- Se modifica el valor del atributo destino de iterador1' con A'.

- Agrega esta relación a una lista de relaciones, y como en este caso ya no hay mas relaciones, devuelve una lista con iterador1'.

En pocos palabra, este método coge un elemento de tipo CompositeVariability del modelo, transforma todo el modelo CVL que se encuentra después de este elemento, conectando los elementos entre ellos, y nos devuelve una lista con todas las relaciones que salen de el. Después será el método process, que con esta información terminara de transformar el modelo.

3.5.Traducción entre resoluciones y productos

Al igual que en el apartado anterior, primero se pondrá un ejemplo sencillo y se explicara como se hace esta transformación, después se detallaran los métodos necesarios para llevar a cabo la transformación y cual es su función en este proceso.

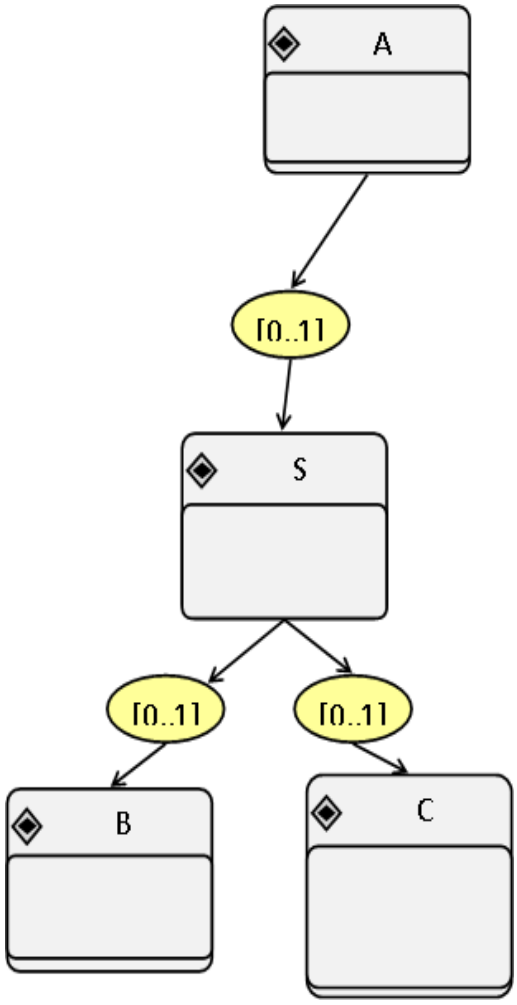
Algunos detalles a tener en cuenta antes de comenzar, los Resolution de CVL y los productos de FaMa no tiene los mismos elementos, el primero utiliza IteratorResolution y ResolutionElement y los productos utilizan solo Features, es decir un Producto es una lista de Features.

PASOS

- Se tiene como entrada la ruta del modelo. Con esta ruta se carga el modelo obteniendo una variable de tipo CVLModel.

- Utilizando el método getSpecification() de esta variable, se obtiene una lista de ResolutionSpecification, en este caso solo hay uno, pero podría haber mas de uno.

- Se coge el primer elemento de la Resolucion A y se le piden con el método getResolution() que nos devuelve una lista de IteratorResolution concretamente siguiendo el ejemplo de la derecha esta lista estará formada por un único Iterator.



- Por cada elemento de la lista hay que pedirle los ResolutionElement que tiene por debajo con el método getChoice() esto devuelve S.

- Repetimos el procedimiento del punto 3 obteniendo una lista con dos iterator.

- Utilizando el método getChoice()como en el punto 4, para cada uno de los iterator obtenemos B y C.

- Ahora se tiene que coger los nombres de todos los ResolutionElement y buscar su correspondiente Feature en el Feature Model que ya habrá sido transformado como se ha explicado en el apartado anterior.

- Una vez se tiene todas las Features que formaran el producto, se crear un producto vacío y se le añaden todas la Features especificando el FeatureModel.

- En caso de que hubiera mas Resoluciones habría que repetir este procedimiento por cada Resolucion.

Los métodos implementados para realizar la transformación expuesta anteriormente son los siguientes:

1. public ArrayList<Product> cvlResolutionToFamaVariability (String cvlPPath)

Encapsula la funcionalidad de la aplicación, el usuario llamara a este método y obtendrá como resultado una lista con todas las resoluciones del modelo que se encuentra en la ruta que le pasamos como entrada, claro esta transformadas en su equivalente en el modelos FaMa es decir productos.

2. public GenericFeatureModel CVL2GenericFeatureModel (String cvlPPath)

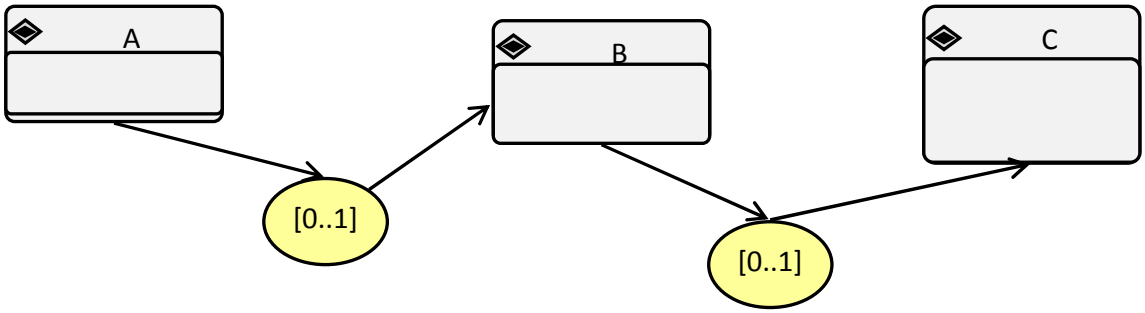
Este método se explico en el apartado anterior, es el método que encapsula la funcionalidad de la transformación del CVL Model, se le llama para obtener esta trasformación ya que es necesario para la transformación de las resoluciones.

3. private CVLModel getCvlModel (String cvlPPath)

También ha sido explicado anteriormente, se le pasa una ruta y carga el modelo, en este se encuentran tanto el modelo principal como las resoluciones que gracias a un método ya definido podremos recuperarlas.

4. private ArrayList<ResolutionElement> processResolution (ResolutionElement resolutionElement)

Es el método recursivo encargado de recorrer la resolución que se le pasa y devolver una lista con todos los resolutionElement, para mostrar este proceso se pondrá un ejemplo:



- Se inicializa una lista vacía, de ResolutionElement, esta será la lista que se devolverá cuando se finalice la ejecución.

- La función tiene como entrada A, se le pide los iterator que tiene por debajo con la función ya implementada getResolution().

- Esta función nos devuelve una lista con todos los iterator por debajo, en este caso como se puede ver solo uno.

- Se recorre la lista de iterator, y por cada iterator se le piden los ResolutionElement que tiene por debajo con getChoice().

- En este caso se obtendrá B, ahora se vuelve a llamar a processResolution con B, y se resuelve esta nueva llamada, mientras la llamada de processResolution de A se queda esperando a la respuesta de esta.

- La llamada de B tendrá el mismo procedimiento explicado anteriormente y realizara otra llama con C y se quedara esperando la respuesta de este.

- La llamada con C al no tener mas elementos por debajo termina y devuelve una lista con C.

- La llamada con B recibe C y devuelve una lista con B y C.

- Y por último la llamada con A recibe B y C y devuelve A, B y C.

5.private ArrayList<GenericFeature> getFeatureOfResolution(GenericFeatureModel fm,ArrayList<ResolutionElement> arrayList)

Como se puede ver tiene varias entradas, la primera es el modelo FaMa transformado anteriormente, y el segundo es el resultado de la llamada al método anterior, una lista de ResolutionElement.

- Inicializa una lista vacía de Feature.

- Hace un recorrido de la lista que le pasan como entrada, cogiendo para cada elemento su nombre y buscándolo en el FeatureModel un elemento con ese nombre, una vez lo encuentra lo añade a la lista creada antes de Features.

- Devuelve la lista de Features.

3.6.Traducción entre restricciones CVL y restricciones del modelo de características

El proceso de transformación de restricciones se lleva a cabo al mismo tiempo que la transformación del modelo CV. Este proceso es mas sencillo que los anteriores, el método que incluye la mayor parte de lógica es:

1.private void addConstraint(FAMAFeatureModel fm)

Básicamente accede a la lista donde se encuentran todas las restricciones de CVL y las transforma en restricciones FaMa, para ello primero tiene que diferenciar entre los distintos tipos de restricciones.

Este proyecto solo soporta dos:

- Inclusión -> En CVL se representa con AND

- Exclusión -> En FaMa se representa con NOT AND

Una vez analizada la restricción se decide si se trata de inclusión o de exclusión se llama a los siguientes métodos respectivamente:

2.private void addExcludes(FAMAFeatureModel fm, Constraint c)

Sabiendo que es una restricción de exclusión, necesitamos crear su equivalente en FaMa que es una ExcludesDependency, para terminar se tendrá que obtener de la Constraint de CVL los datos que nos indican que Feature tendrá como origen y cual como destino la nueva Dependencia y añadirla a el modelo ya transformado Feature-Model.

3.private void addRequieres(FAMAFeatureModel fm, Constraint c)

Sabiendo que es una restricción de inclusión, necesitamos crear su equivalente en Fa-Ma que es una RequiresDependency, para terminar al igual que en el método anterior se tendrá que obtener de la Constraint de CVL los datos que nos indican que Feature tendrá como origen y cual como destino la nueva Dependencia y añadirla a el modelo ya transformado FeatureModel.

Para terminar con esta explicación falta explicar como se encuentran las restricciones en el modelo CVL y como se almacena.

Esto se hace en el método processAux explicado en el apartado A, cada vez que se buscan los siguientes elementos tanto de un iterador como de un compositeVariability se si el elemento es de tipo Constraint (Tipo de las restricciones en CVL) y en caso de que lo sea se almacena en un array global donde se almacenan todas las restricciones del modelo y del que luego se sacaran para ser analizadas como se ha explicado ante-riormente.

Después de explicar la cuestiones que se pueden realizar con FaMa se mostraran ejemplos de restricciones y se les hará preguntas para comprobar que después de la transformación estas siguen funcionando.

4. Operaciones de Análisis FaMa sobre los modelos CVL

FAMA ofrece dos funciones principales: Edición modelo visual creación y análisis del modelo automatizado. Sobre esta última función es sobre la que se hablará en este apartado.

4.1.Operaciones sobre Modelos de Variabilidad

A continuación se explicaran las distintas pregunta que permite realizar FaMa Framework sobre el modelo de variabilidad, y para cada una de estas preguntas se mostraran pequeños casos practico que permitirán no solo una mejor comprensión de estas cuestiones sino la comprobación de la correcta transformación del modelo CVL al modelo FaMa. Como se podrá ver las respuestas a las preguntas hacen referencia a los modelos de FaMa porque para él, se está trabajando con modelos FaMa, no obstante la respuesta se puede aplicar a CVL sin ningún problema.

4.1.1. Products

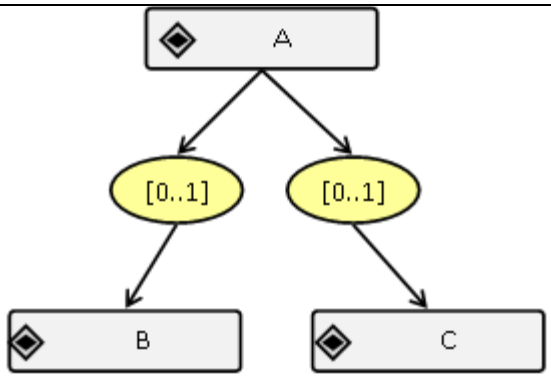
Esta pregunta te permite acceder al conjunto de productos representados por el Modelo, no los que se han hecho manualmente sino todos los que se podrían hacer. Una vez que hemos hecho la pregunta a FAMA, se dispone de un método que permite obtener una lista con todos los productos:

- getAllProducts() ->Devuelve una colección con todos los posibles productos del modelo.

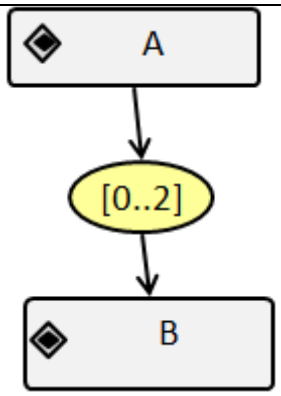
4.1.2. Number of Products

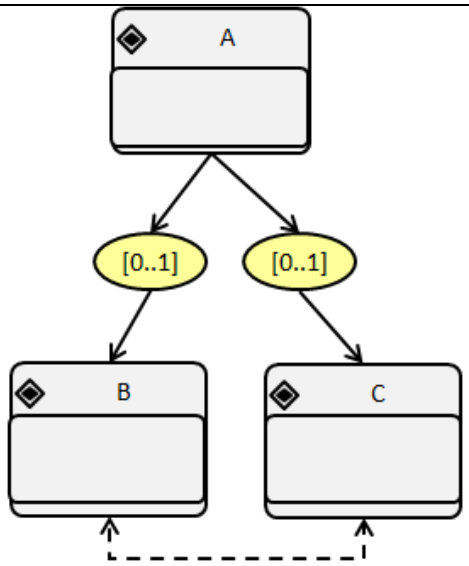
Solo tiene un método:

- getNumberOfProfucts() -> Devuelve el numero de productos del modelo.

Caso práctico 1:

Resultado de aplicar la pregunta al modelo:
"The number of products is: 4.0"
Conclusiones:

Esto es correcto ya que podrían existir cuatro productos: Feature A /Feature A y B /Feature A y C /Feature A, B y C

Caso práctico 2:

Resultado de aplicar la pregunta al modelo:
"The number of products is: 2.0"
Conclusiones:
En CVL este resultado es incorrecto ya que para este pueden existir tres resoluciones: Feature A /Feature A y B /Feature A, B y B Pero para FaMa según este modelo solo se permite: Feature A y /Feature A y B Esta es una de las inconsistencias encontradas entre los modelos CVL y FaMa, será comentada en conclusiones.

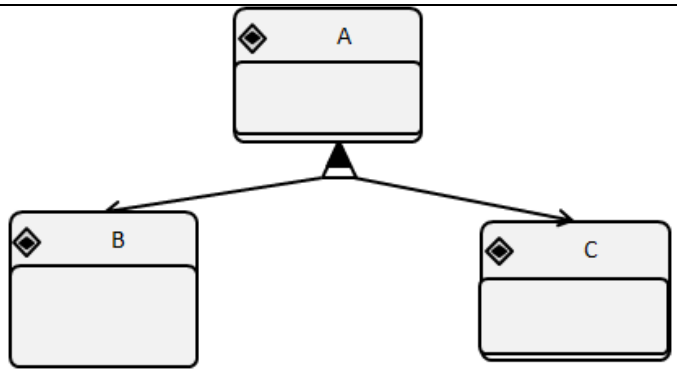
Caso práctico 3:

Resultado de aplicar la pregunta al modelo:
"The number of products is: 3.0"

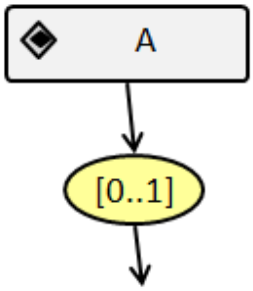
Conclusiones:
El resultado es el esperado, por lo que se comprueba la correcta transformación de la restricción de tipo exclusión. Los productos que se podrían formar con este nuevo modelo serian: Feature A /Feature A y B /Feature A y C.

4.1.3. Valid

Determina si un modelo es válido, es una de las preguntas mas importantes que nos permite realizar FaMa saber si un modelo es valido sintácticamente nada mas terminarlo puede ahorrar muchos problemas futuros, tiene un solo un método:

- isValid() -> Devuelve un boolean (True o False) indicando si el modelos es valido o no.

Caso práctico 1:

Resultado de aplicar la pregunta al modelo:
"Your feature model is valid"
Conclusiones:
El resultado es correcto, el modelo es sintácticamente correcto.

Caso práctico 2:

Resultado de aplicar la pregunta al modelo:
"Your feature model is not valid"
Conclusiones:

El resultado es correcto, un Iterator en CVL o una Relation en FaMa siempre tiene que unir dos o mas CompositeVariability o Feature. Este modelo no es válido.
Comentario:
Utilizando la herramienta grafica, que se ha usado habitualmente para hacer estos diagramas, hacer este diagrama es imposible, te obliga a que como mínimo un Iterator conecte dos CompositeVariability. Este error solo se podria cometer si no se utiliza la herramienta grafica y se crea o modifica directamente en el árbol del modelo.

4.1.4. Commonality

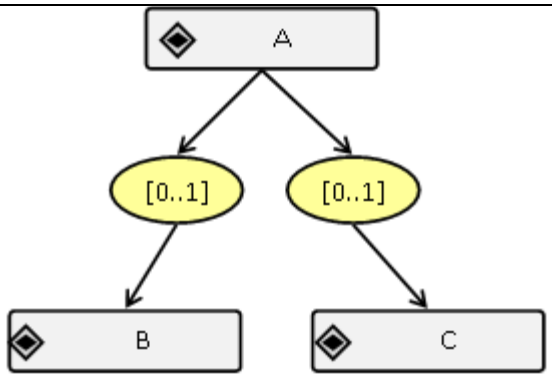
El objetivo de esta pregunta es determinar la frecuencia de una característica, es decir el numero de productos donde podría aparecer.

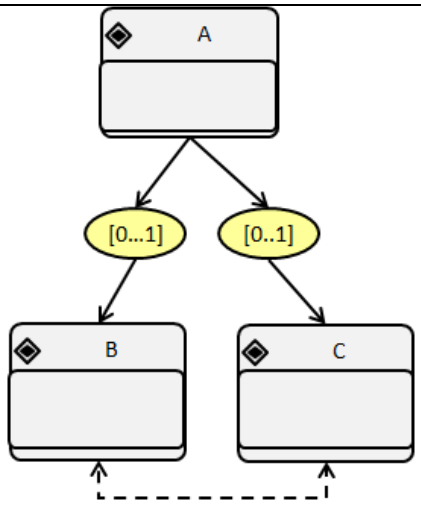
Antes de preguntar esta pregunta es necesario especificar la característica sobre la cual queremos realizar la pregunta. Para ello utilizamos el método:

- setFeature(Feature f) -> con este método especificamos la característica sobre la que queremos preguntas.

Una vez especificada la característica, se llama a este método:

- getCommonality() -> que devuelve la frecuencia de aparición de la característica especificada.

Caso práctico 1:

Resultado de aplicar la pregunta al modelo:
Si se pregunta por A -> "Commonality of the selected: 4" Si se pregunta por C -> "Commonality of the selected: 2"
Conclusiones:
El resultado es correcto, A puede aparecer en los productos formados por las Features A/ A,B /A,C /A,B,C respectivamente y C puede aparecer en los productos formados por las Features A,C/ A,B,C respectivamente.

Caso práctico 2:

Resultado de aplicar la pregunta al modelo:
Si se pregunta por A -> "Commonality of the selected: 3" Si se pregunta por C -> "Commonality of the selected: 1"
Conclusiones:
El resultado es correcto, A puede aparecer en los productos formados por las Features A/ A,B /A,C respectivamente y C puede aparecer en el producto formados por las Features A,C .

4.1.5. Filter

Permite especificar un filtro en el modelo, añadiendo o eliminando características con las siguientes operaciones:

- addValue(VariabilityElement ve, int value) ->Permite añadir un elemento al modelo, que solo existirá durante la duración de la pregunta.

- removeValue(VariabilityElement ve) ->Permite eliminar un elemento del modelo que solo existirá al igual que con el método anterior mientras dure la pregunta.

Para poder apreciar el efecto, se debe utilizar el método setQuestion que se explicará a continuación.

4.1.6. Set Question

Con SetQuestion, podemos componer un conjunto de preguntas, y preguntar una tras otra.

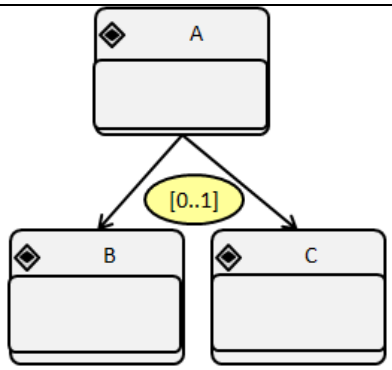
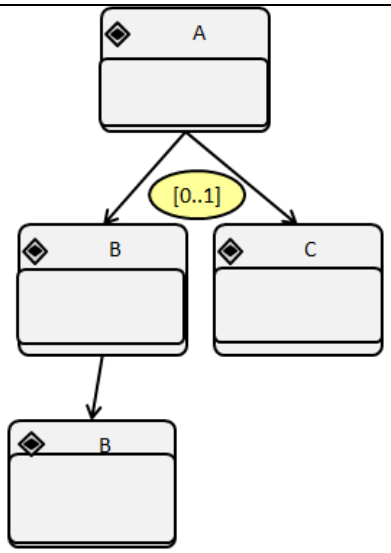
- *addQuestion (Pregunta q)* -> Permite añadir pregunta.

Después habrá que preguntar por la setQuestion.

4.1.7. Variability

Es muy simple, devuelve la variabilidad (Las medidas de dispersión, también llamadas medidas de variabilidad, muestran la variabilidad de una distribución, indicando por medio de un número, si las diferentes puntuaciones de una variable están muy alejadas de la media. Cuanto mayor sea ese valor, mayor será la variabilidad, cuanto menor sea, más homogénea será a la media. Así se sabe si todos los casos son parecidos o varían mucho entre ellos.) del modelo de característica.
Tiene un único método:

- *getVariability()* -> Devuelve un float (numero real) con el valor de la variabilidad.

<table>
<tr><td colspan="2">Caso práctico 1:</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Resultado de aplicar la pregunta al modelo:</td></tr>
<tr><td>"The variability is: 0.33333334"</td><td>"The variability is: 0.42857143"</td></tr>
<tr><td colspan="2">Conclusiones:</td></tr>
<tr><td colspan="2">El resultado es el esperado, cuanto mas bajo es el numero, mas variabilidad tiene el modelo.</td></tr>
</table>

4.2.Operaciones sobre Modelos de Resolución

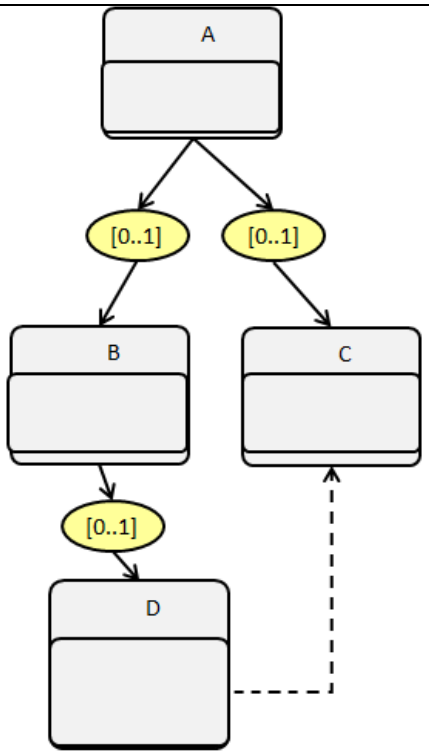
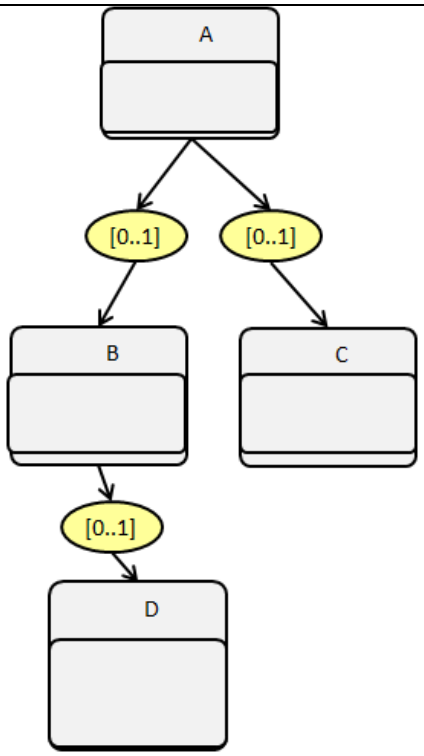
A continuación se explicaran las distintas pregunta que permite realizar FaMa Framework sobre los modelos de resolución, y para cada una de estas preguntas al igual que en el apartado anterior se mostraran pequeños casos practico que permitirán no solo una mejor comprensión de estas cuestiones sino la comprobación de la correcta transformación de los modelos de resolución de CVL a el producto de FaMa.

4.2.1. Valid Product

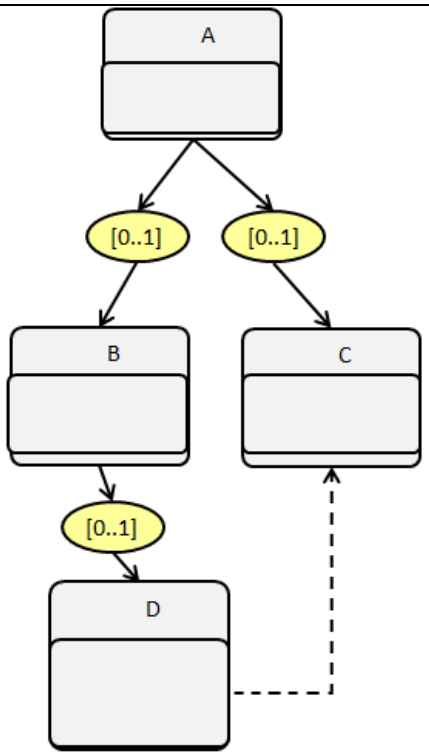
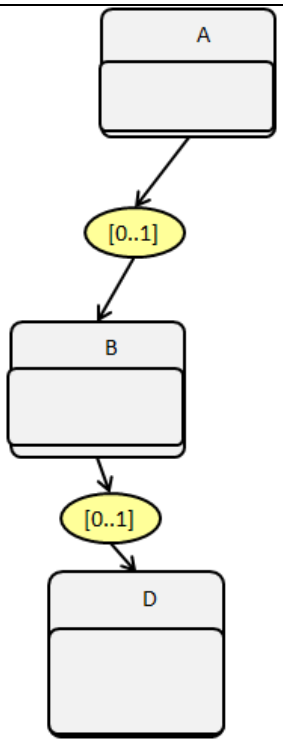
El objetivo de esta pregunta es determinar si un producto es valido respecto al modelo, un modelo establece que elementos podrían existir en un producto, como y en que medida y algunas restricciones sobre ello. Esta pregunta informa de si el producto cumple estas restricciones y no tiene ningún elemento no valido.

Para poder realizar esta pregunta se ofrecen dos métodos:

- setProduct(Product p) -> Permite elegir el producto sobre el que se preguntará.
- isValid() -> Permite saber si el producto seleccionado previamente es válido similar al método ya explicado para modelos de variabilidad.

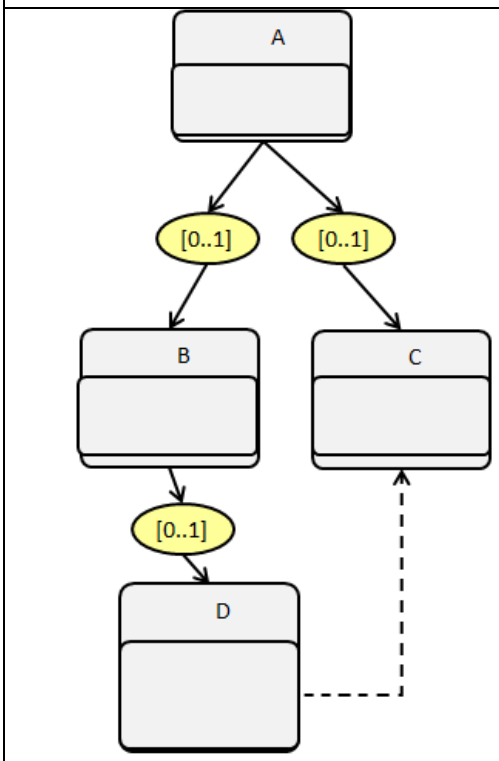
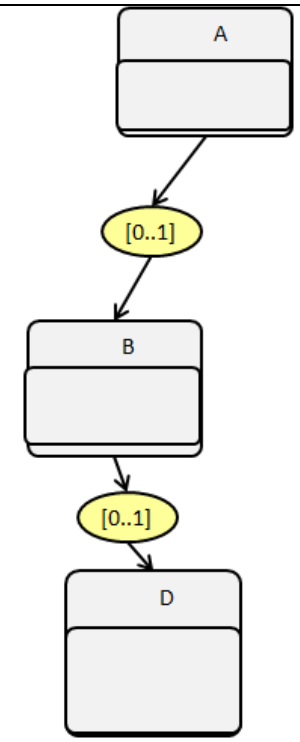
<table>
<tr><td colspan="2">Caso práctico 1:</td></tr>
<tr><td>Modelo (Diagrama)</td><td>Resolución (Diagrama)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Resultado de aplicar la pregunta a la resolución:</td></tr>
<tr><td colspan="2">"This product is Valid"</td></tr>
</table>

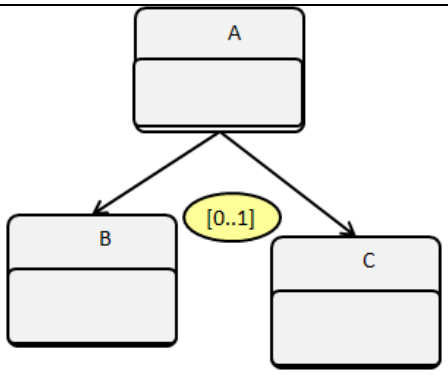
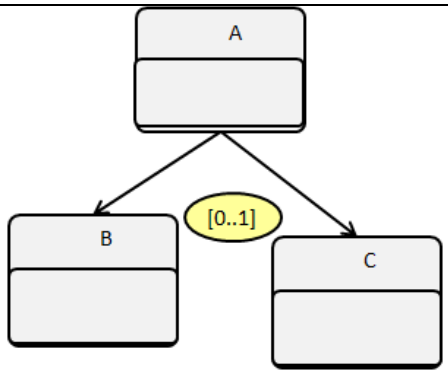
Conclusiones:
Esto es correcto, la resolución no incumple ninguna de las restricciones que incorpora el modelo.
Comentarios:
Es la primera vez que se pone un ejemplo con restricción de tipo implicación, se recuerda que esto obligaba a que si D se encuentra en la resolución, también este C.

Caso práctico 2:	
Modelo (Diagrama)	Resolución (Diagrama)
	
Resultado de aplicar la pregunta a la resolución:	
"This product is not Valid"	
Conclusiones:	
El resultado es correcto, como se ha comentado antes, debido a la restricción de implicación, si existe D en la resolución tiene que existir C y como esto no se cumple la resolución no es valida.	

4.2.2. Valid Configuration

Muy similar a la anterior, solo que esta no evalúa el producto terminado, sino una temporal configuración antes del producto terminado. Los métodos son los mismos que en la cuestión anterior.

Caso práctico 1:	
Modelo (Diagrama)	Resolución (Diagrama)
	
Resultado de aplicar la pregunta a la resolución:	
"This product is Valid"	
Conclusiones:	
El resultado es correcto, utilizando el mismo modelo y la misma resolución que en la anterior pregunta obtenemos un resultado opuesto y correcto, esto se debe a que esta pregunta evalúa una resolución que aun no está terminada, por lo que solo detecta errores que ya no tendría solución, en cambio el error que produce la restricción de implicación puede ser solucionado añadiendo C.	

Caso práctico 2:	
Modelo (Diagrama)	Resolución (Diagrama)
	

Resultado de aplicar la pregunta a la resolución:
"This product is not Valid"
Conclusiones:
El resultado de la pregunta es el esperado, el Iterator del modelo obliga a que solo puede ser seleccionado una de las CompositeVariability, y en la resolución se han seleccionado ambas. Por lo que la resolución es no valida.

4.3.Operaciones de explicación de error

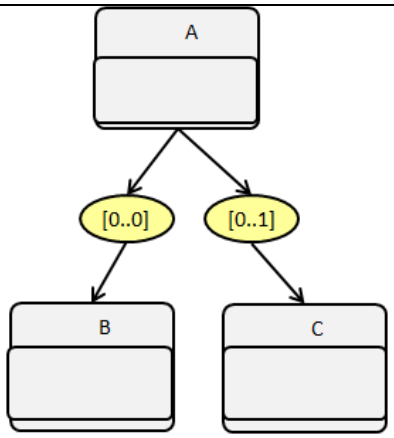
4.3.1. Sobre Modelos de Variabilidad

4.3.1.1. Detect Errors

Esta es una de las preguntas mas útiles de FaMa, coge un modelo de FaMa y te dice si tiene algún error. Algunas de las preguntas de las que se ha hablado anteriormente permitían saber si era valido, pero ninguna te permitía localizar el error.

- setObservations(Collection<Observation> observations) -> Permite introducir algunas observaciones antes de preguntar por los errores.

- getErrors() -> Devuelve los errores del modelo.

Caso práctico 1:

Resultado de aplicar la pregunta al modelo:
" Dead Feature: B"
Conclusiones:
Correcto, B nunca podrá ser alcanzado.

4.3.1.2. Explain Errors Question

Una vez detectado el error con la anterior pregunta, se puede utilizar esta para obtener una explicación de las posibles soluciones de este problemas. Para ello se facilitan los siguientes métodos:

- setErrors (Collection<Error> colErrors) -> Con este método se le pasan los errores obtenidos con al pregunta anterior.

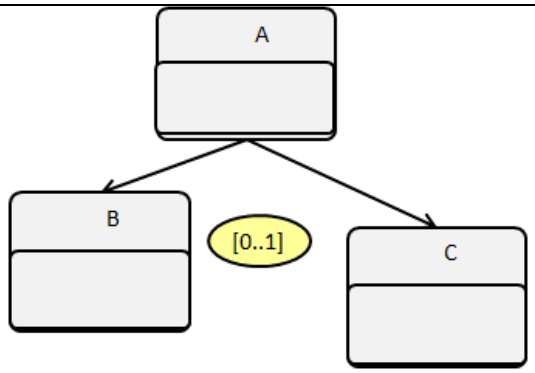
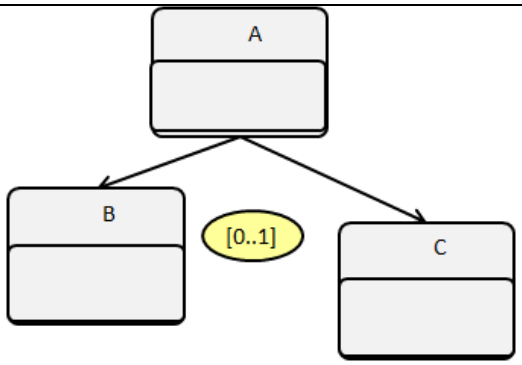
- getErrors() -> Devuelve los errores, una vez realizada la pregunta.

4.3.2. Sobre Modelos de Resolución

4.3.2.1. Valid Configuration Errors Question

Similar a la pregunta Detect Errors pero esta va dirigida a los productos. Solo tiene un método:

- setProduct (Product p) -> Permite seleccionar el producto sobre el que se realizara la pregunta

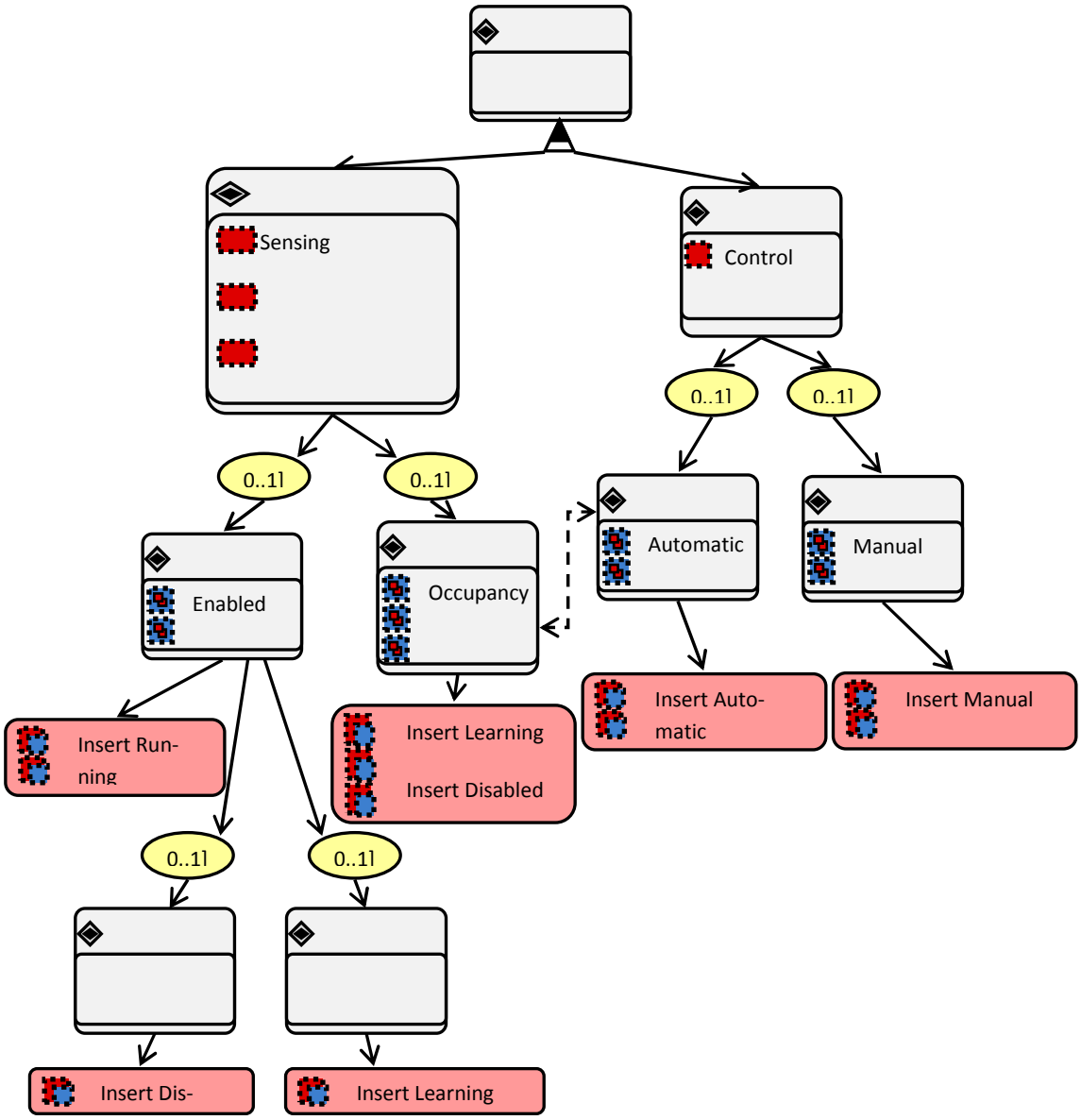
<table>
<tr><td colspan="2">Caso práctico 1:</td></tr>
<tr><td>Modelo (Diagrama)</td><td>Resolución (Diagrama)</td></tr>
<tr><td> 
</td><td> 
</td></tr>
<tr><td colspan="2">Resultado de aplicar la pregunta a la resolución:</td></tr>
<tr><td colspan="2">Solution 1:
"The feature B has to be deselected"
Solution 2:
"The feature C has to be deselected"</td></tr>
<tr><td colspan="2">Conclusiones:</td></tr>
<tr><td colspan="2">Indica, que hay dos posibles soluciones, deseleccionar B o C.</td></tr>
</table>

5. Smart Home

5.1. Introducción al Caso de estudio

En este apartado se expondrá un caso completo y cuya aplicación es real, se explicara tanto el proceso de transformación del modelo y de sus configuraciones CVL a modelos y productos FaMa como los resultados de aplicar las preguntas FaMa a este, y su interpretación.

El caso de estudio a tratar es el siguiente:



Básicamente, es un modelo que permite elegir distintas configuraciones de una casa inteligente. Este modelo se hizo para un sistema que dependiendo de unas condiciones se auto configuraba a si mismo.

En el modelo se distinguen dos ramas:

- Detección: Como su nombre indica, se centra en la detección de personas en la casa, como se vera en los siguientes subnodos esto puede ser tanto útil para la segurita y la detección de intrusos como para el bienestar del inquilino que se encuentra en la casa.

- Calefacción: Al igual que el anterior su nombre lo dice todo, indica como se tratara el tema de la calefacción.

Ambas pueden ser elegidas, su iterator es de tipo OR, por lo que puede ser elegido cualquiera de las dos o ambas al mismo tiempo.

En cuanto a la rama de detección, se distinguen dos subnodos:

- Seguridad: Nodo que define la seguridad de la casa, este a su vez permite elegir entre simulación y detección.

- Confort: Se entiende por confort, aquello que ofrece bienestar y comodidades.

Todos estos subnodos pueden ser elegidos al mismo tiempo con una única restricción que se explicara a continuación de explicar la ultima rama, ya que relaciona ambas.

En cuanto a la rama de calefacción, hay dos subnodos:

- Energy save (Ahorro de energía): Se centra en el ahorro de energía muy útil cuando no hay nadie en la casa.

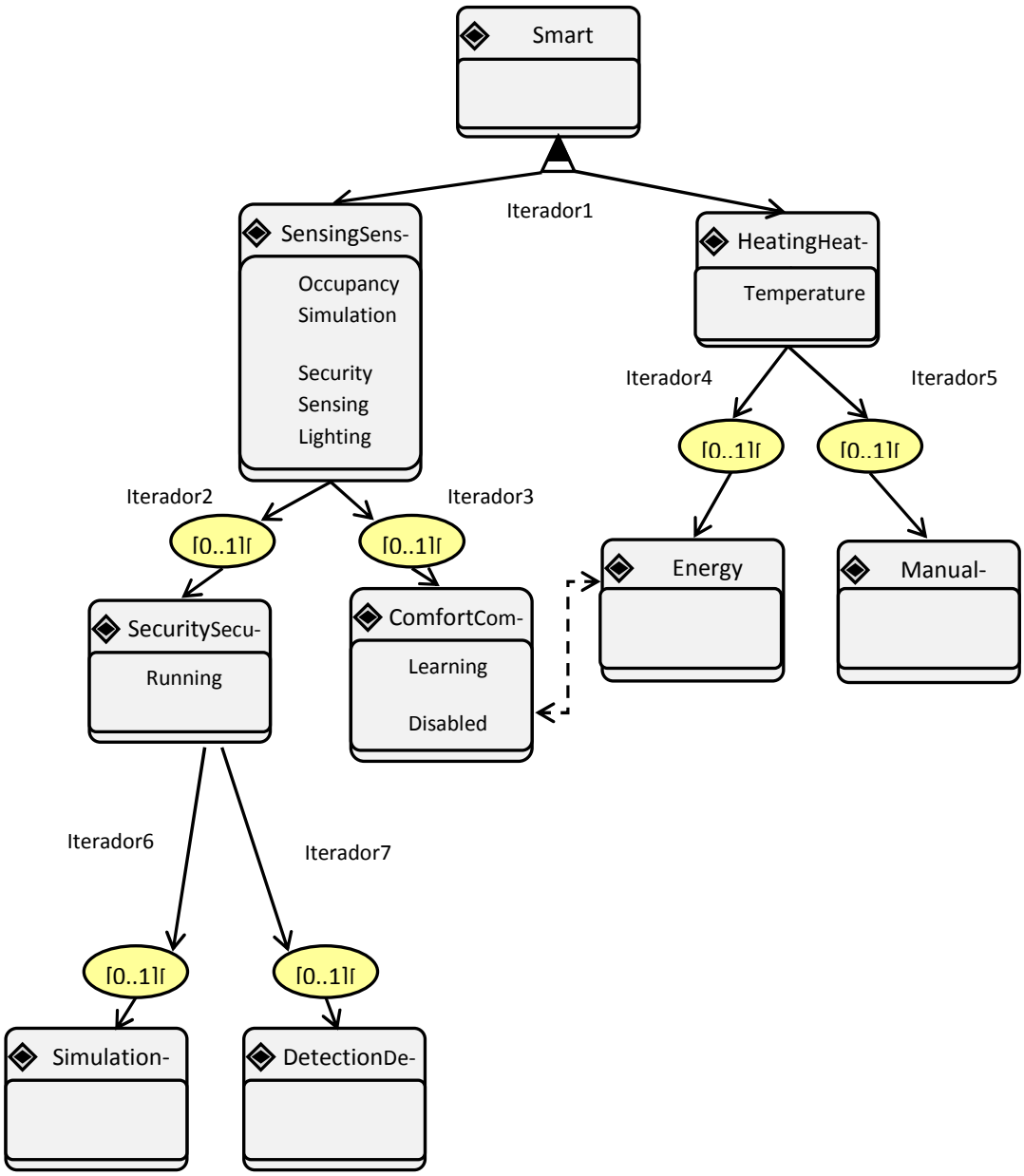
- Manual: Cuando el inquilino se encuentra en casa, puede en lugar de utilizar las configuraciones automáticas, puede decidir ahorrar energía o en caso de tener miedo dar prioridad a la seguridad.

Ambas pueden ser seleccionadas, pero existe una restricción en este modelo, si se ha elegido el nodo Confort antes este no puede ser seleccionado y viceversa.

5.1.1. Modelo de variabilidad del Smart Home

Como se puede ver en el modelo del caso práctico aparece unos elementos sobre los que hasta ahora nunca se había hablado, estos son atributos, en este proyecto no se han podido tener en cuenta a la hora de la transformación, se hablara de ellos en conclusiones y futuras ampliaciones.

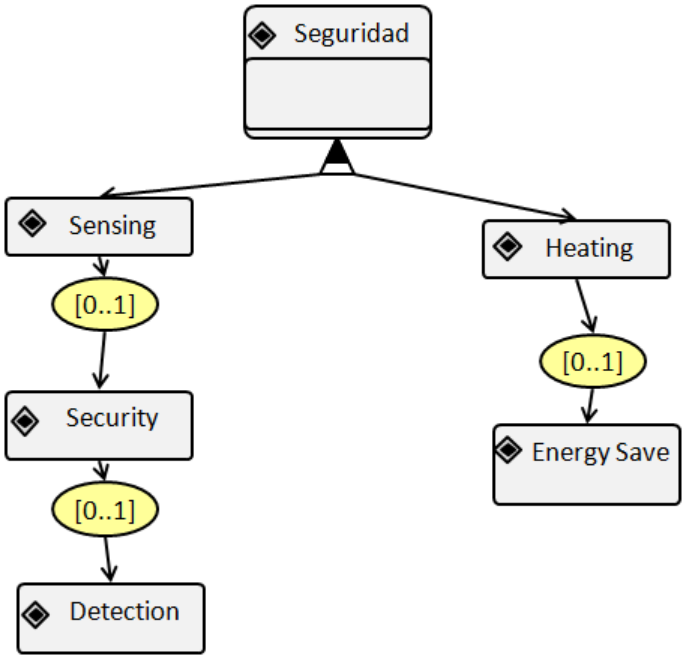
Así que lo que interesa del caso práctico serán los CompositeVariability, iterator y Constraint, el modelo entonces queda así:



5.1.2. Modelos de Resolución del Smart Home

La posibilidades de este modelo, permiten hacer numerosos productos, en esta sección se expondrán dos resoluciones que resultan interesantes:

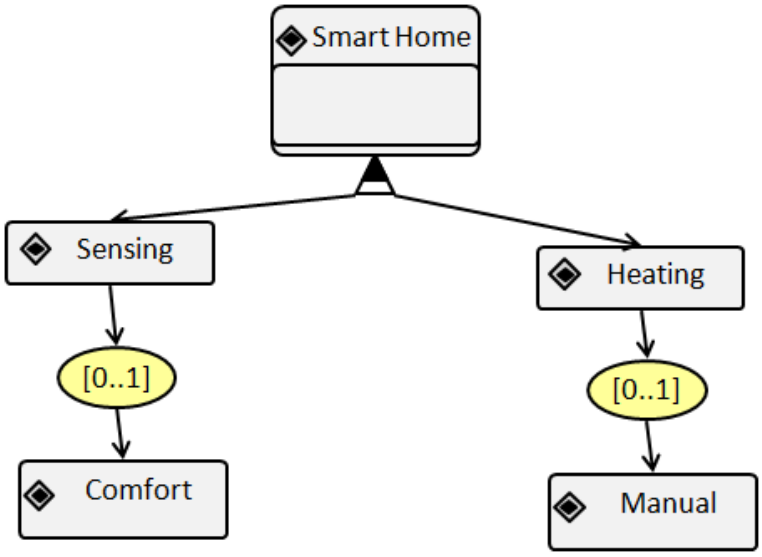
Resolución 1:



Este producto seria una configuración apropiada para una casa inteligente, cuando no hay ningún inquilino en su interior.

Se reduce el consumo de calefacción ya que no hay nadie en casa, y se aumenta el nivel de seguridad.

Resolución 2:



Este producto es una configuración apropiada para cuando la casa esta ocupada, donde no interesa que la alarma de seguridad se active cada vez que alguien se mueve por la casa, y si que interesa que el bienestar del inquilino sea el mejor posible.

5.2.Transformación del Modelo de Variabilidad

Para explicar esta transformación, se ha decidido a diferencia de otras explicaciones anteriores, no dar ningún detalle de implantación, facilitando así la compresión del proceso.

1. Se tiene como entrada la ruta del modelo de variabilidad representado por el diagrama que se muestra arriba.

2. Con esta ruta se carga el modelo obteniendo una variable con el modelo CVL.

3. Se busca en esta variable el primer elemento.

4. Se crea un modelo Fama vacío

5. Se pregunta por los elementos que son inferiores inmediatos a el, estos son:

Iterador1	Constraint Exclusión		

6. Se recorre esta lista, preguntando a cada elemento su tipo, hay dos posibilidades, que sea de tipo iterador o constraint.

6.1. El primer elemento es de tipo Iterador por lo que se pasa a pedir sus elementos inferiores inmediatamente.

Sensing	Heating		

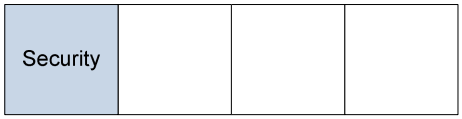
7.1. Se recorre la lista, preguntando a cada elemento por su tipo, pudiendo ser de dos tipos, constraint o composite.

7.1.1. El primer elemento es de tipo Composite, por lo que se repetira el procedimiento realizada con el primer elemento del modelo, y se buscaran sus elementos inferiores.

Iterador2	Iterador3		

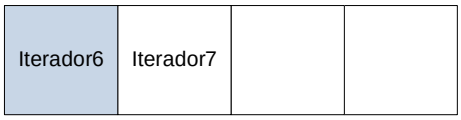
7.1.2. Se recorre preguntando por su tipo.

7.1.2.1. El primer elemento es de tipo iterador, se repite el mismo procedimiento y se le pregunta por sus elementos inferiores.



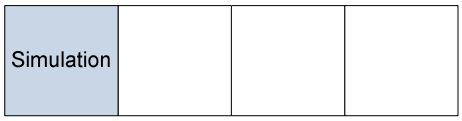
7.1.2.2. Se recorre preguntando por su tipo.

7.1.2.2.1 Su primer y único elemento, es de tipo composite, se pide los elementos inferiores.



7.1.2.2.2. Se recorre preguntando por su tipo.

7.1.2.2.2.1. El primer elemento es de tipo iterador, se obtiene sus elementos inferiores.



7.1.2.2.2.2. Se recorre preguntando por su tipo.

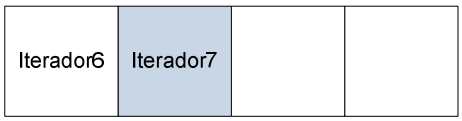
7.1.2.2.2.2.1. El primer y único elemento es de tipo composite, se obtienen sus elementos inferiores.

7.1.2.2.2.2.2. Como no hay ninguno, se pasa ha transformarlo en su equivalente en el modelo FaMa una feature.

7.1.2.2.2.3. Termina el recorrido

7.1.2.2.2.4. Ahora se pasa a transformar el Iterador6 en una relación equivalente en el modelo FaMa.

7.1.2.2.2.5. Se añade a la relación recién creado como destino la feature obtenida como resultado de la transformación de Simulation.



7.1.2.2.3. Se continua recorriendo de la esta lista, que se tenia pendiente.

7.1.2.2.3.1. El segundo elemento es también de tipo iterador, se pasan a obtener sus elementos inferiores.

Detection			

7.1.2.2.3.2. Se recorre preguntando por su tipo.

7.1.2.2.3.2.1. El primer y único elemento es de tipo composite, se obtienen sus elementos inferiores.

7.1.2.2.3.2.2. Como no hay ninguno, se pasa ha transformarlo en su equivalente en el modelo FaMa una feature.

7.1.2.2.3.3. Termina el recorrido no quedan mas elementos

7.1.2.2.3.4. Ahora se pasa a transformar el Iterador7 en una relación equivalente en el modelo FaMa.

7.1.2.2.3.5. Se añade a la relación recién creado como destino la feature obtenida como resultado de la transformación de Detection.

7.1.2.2.4. Se termina el recorrido no quedan mas elementos.

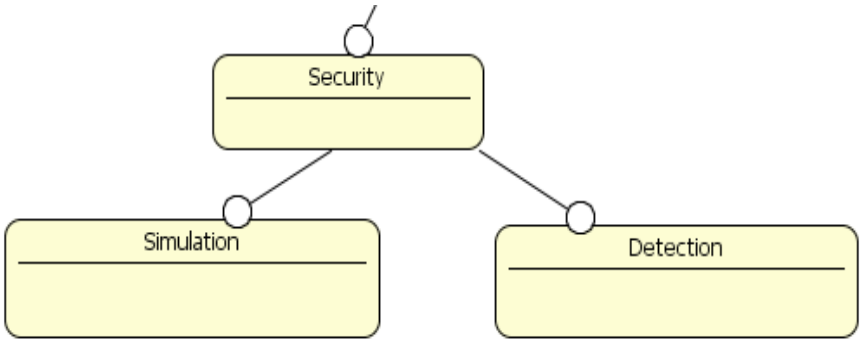
7.1.2.2.5. Se transforma Security en su Feature correspondiente.

7.1.2.2.6. Se le añade la relación 7 y 6

7.1.2.3. Se transforma el Iterador 2 en una relación.

7.1.2.3. A la relación2 resultado de la transformación del Iterador2 se le añade como feature destino la feature Security.

7.1.3. Las transformaciones realizadas hasta hora, equivaldría al siguiente diagrama que representa un modelo FaMa:



Iterador2	Iterador3		

7.1.4. Se continua recorriendo de la esta lista, que se tenia pendiente.

7.1.4.1. El segundo elemento es también de tipo iterador, se pasan a obtener sus elementos inferiores.

Confort			

7.1.4.2. Se recorre preguntando por su tipo.

7.1.4.2.1. El primer y único elemento es de tipo composite, se obtienen sus elementos inferiores.

7.1.4.2.2. Como no hay ninguno, se pasa ha transformarlo en su equivalente en el modelo FaMa una feature.

7.1.4.3. Termina el recorrido no quedan mas elementos en la lista.

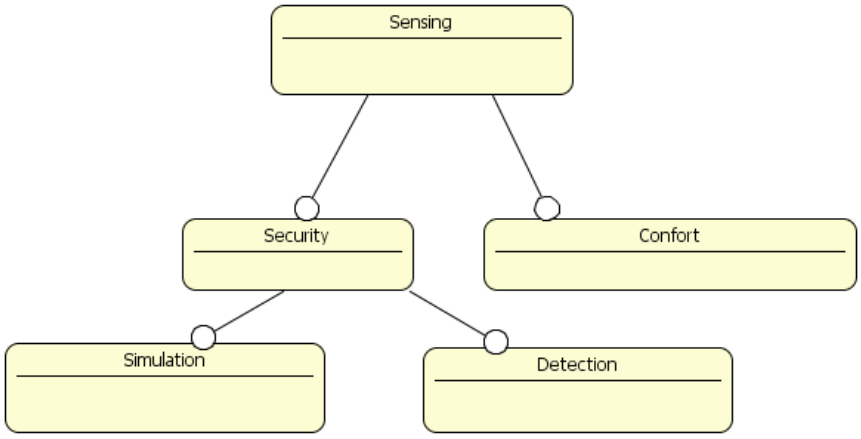
7.1.4.4. Ahora se pasa a transformar el Iterador3 en una relación equivalente en el modelo FaMa.

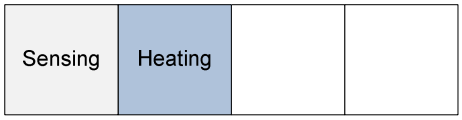
7.1.4.5. Se añade a la relación recién creado como destino la feature obtenida como resultado de la transformación de Confort.

7.1.5. Termina el recorrido, no quedan mas elementos.

7.1.6. Se transforma Sensing en su Feature correspondiente.

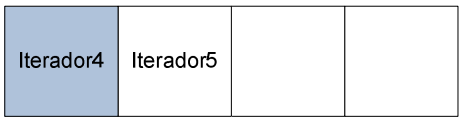
7.1.7. Se le añade la relación 2 y 3





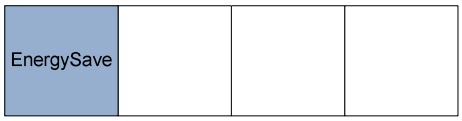
7.2. Se continua recorriendo preguntando por su tipo.

7.2.1. El segundo elemento es de tipo composite, se obtienen sus elementos inferiores.



7.2.2. Se recorre preguntando por su tipo.

7.2.2.1. El primer elemento es de tipo iterador, se obtiene sus elementos inferiores.



7.2.2.2. Se recorre preguntando por su tipo.

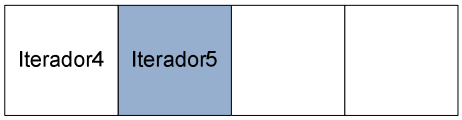
7.2.2.2.1. El primer y único elemento es de tipo composite, se obtienen sus elementos inferiores.

7.2.2.2.2. Como no hay ninguno, se pasa ha transformarlo en su equivalente en el modelo FaMa una feature.

7.2.2.3. Termina el recorrido

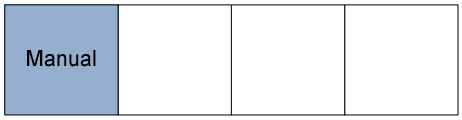
7.2.2.4. Ahora se pasa a transformar el Iterador4 en una relación equivalente en el modelo FaMa.

7.2.2.5. Se añade a la relación recién creado como destino la feature obtenida como resultado de la transformación de EnergySave.



7.2.3. Se continua con el recorrido.

7.2.3.1. El segundo elemento es de tipo iterador, se obtiene sus elementos inferiores.



7.2.3.2. Se recorre preguntando por su tipo.

7.2.3.2.1. El primer y único elemento es de tipo composite, se obtienen sus elementos inferiores.

7.2.3.2.2. Como no hay ninguno, se pasa ha transformarlo en su equivalente en el modelo FaMa una feature.

7.2.3.3. Termina el recorrido

7.2.3.4. Ahora se pasa a transformar el Iterador5 en una relación equivalente en el modelo FaMa.

7.2.3.5. Se añade a la relación recién creado como destino la feature obtenida como resultado de la transformación de Manual.

7.2.4. Se termina el recorrido.

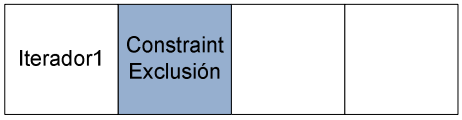
7.2.5. Se transforma Heating, en una feature.

7.2.6. Se le añaden las relaciones correspondientes al Iterador4 e Iterador5.

7.2.7. Termina el recorrido, no quedan mas elementos.

7.3. Se transforma el Iterador1 en una relación de FaMa.

7.4. Se le añade como features destino, Sensing y Heating.



8. Se continua recorriendo con el siguiente elemento.

8.1. El segundo elemento y ultimo es de tipo Constraint, por lo que se almacena en el array de tipo Constraint.

9. Se termina el recorrido.

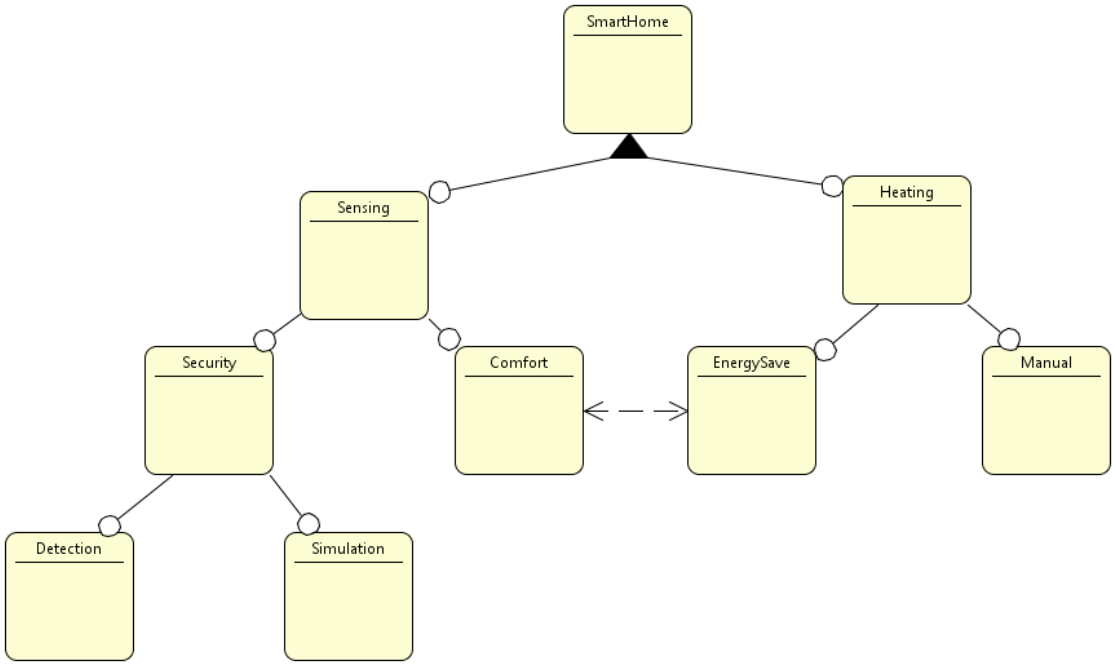
10. Se transforma el primer composite del modelo.

11. Se le añade la relación correspondiente a Iterador1.

12. Es agregado como root (Principal) al modelo FaMa vacío.

13. La lista de constraint es analizada, se identifica el tipo de restricción que es y es transformada en su equivalente en FaMa y por ultimo se agrega al modelo FaMa.

Obteniendo así el equivalente en FaMa al modelo CVL que inicialmente se pasaba obteniendo como resultado el siguiente modelo:



5.3.Transformación de los Modelos de Resolución

Al igual que en la explicación del proceso de transformación entre modelos de variabilidad y modelos de características, esta explicación se realizara omitiendo al máximo posible detalles de implementación para facilitar su comprensión.

1. Se pasa la ruta del modelo, y modelo Fama ya transformado.

2. Se carga el modelo, y se pregunta por las resoluciones, obteniendo así una lista de resoluciones:

Seguridad	Bienestar		

3. Se recorre la lista.

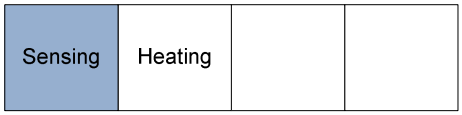
3.1. Se crea un producto vacío.

3.2. Se coge, la primera resolución y utilizando un método ya implementado se le piden sus iteradores.

Iterador1			

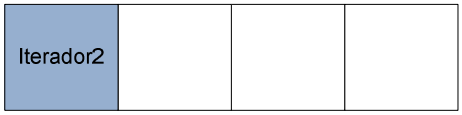
3.3. Se recorre la lista.

3.3.1. Se le piden los composites que tiene este iterador como destino.



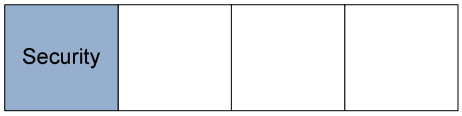
3.3.2. Se recorre la lista.

3.3.2.1. Al igual que en el caso anterior se le piden todos sus iteradores.



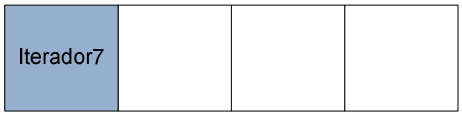
3.3.2.2. Se recorre la lista.

3.3.2.2.1. Se obtienen sus composites.



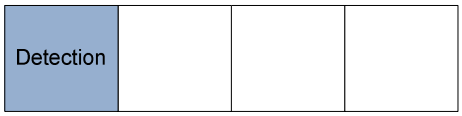
3.3.2.2.2. Se recorre la lista.

3.3.2.2.2.1. Se obtienes sus iteradores.



3.3.2.2.2.2. Se recorre la lista.

3.3.2.2.2.2.1. Se obtienen sus composites.



3.3.2.2.2.2.2. Se recorre la lista.

3.3.2.2.2.2.2.1. Se obtienen sus iteradores, como no tiene ninguno se almacena el composite en una lista.

3.3.2.2.2.2.3. Termina el recorrido.

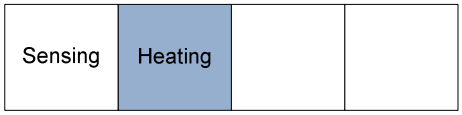
3.3.2.2.2.3. Termina el recorrido.

3.3.2.2.4. Se agrega Security a la lista de composites.

3.3.2.2.3. Termina el recorrido.

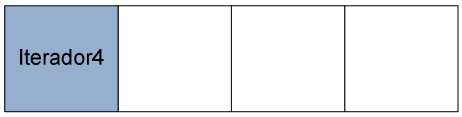
3.3.2.3. Termina el recorrido.

3.3.2.4. Se añade Sensing a la lista.



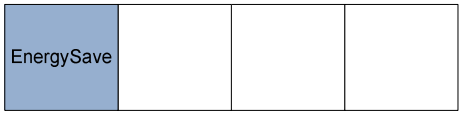
3.3.3. Se continua el recorrido.

3.3.3.1. Se obtienes sus iteradores.



3.3.3.2. Se recorre la lista.

3.3.3.2.1. Se obtienen sus composites.



3.3.3.2.2. Se recorre la lista.

3.3.3.2.2.1. Se obtienen sus iteradores, como no tiene ninguno se almacena EnergySave en una lista.

3.3.3.2.3. Se termina el recorrido.

3.3.3.3. Se termina el recorrido.

3.3.3.4. Se añade Heating a la lista

3.3.4. Se termina el recorrido.

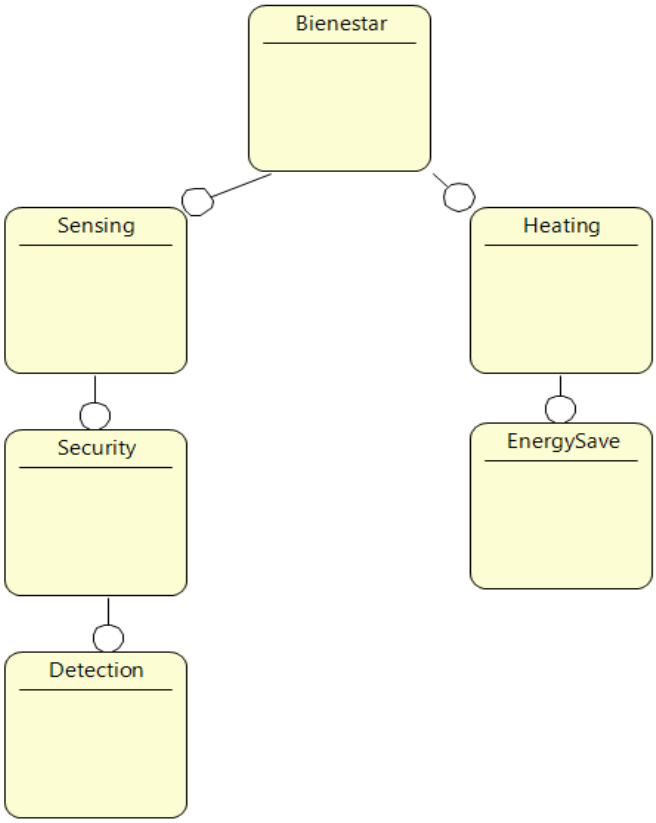
3.4. Se termina el recorrido

3.5. Se añade Bienestar a la lista

3.6. Ahora se recorre la lista de composites obtenida, cogiendo el nombre de cada composite y buscándolo en el modelo FaMa obtenido con la transformación del modelo. Creando así una lista con todos los Features que tendrá el producto.

3.7. Por ultimo se agregan todas estas Features al producto vacío.

Ya se tiene un producto equivalente a la primera resolución encontrada, que era la resolución de seguridad. El diagrama de este nuevo producto es:



Seguridad	Bienestar		

4. Se continua el recorrido, con la siguiente resolución la del Bienestar.

4.1. Se crea un producto vacio.

4.2. Se coge, la primera y se le piden sus iteradores.

Iterador1			

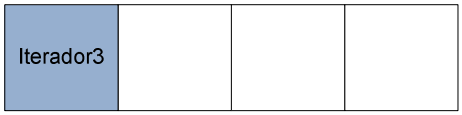
4.3. Se recorre la lista.

4.3.1. Se le piden los composites que tiene este iterador como destino.

Sensing	Heating		

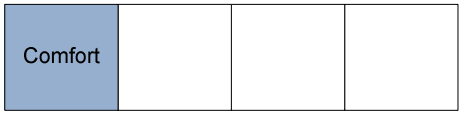
4.3.2. Se recorre la lista.

4.3.2.1. Se le piden todos sus iteradores.



4.3.2.2. Se recorre la lista.

4.3.2.2.1. Se obtienen sus composites.



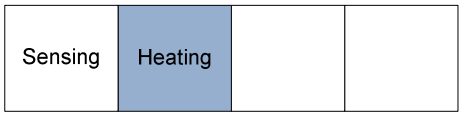
4.3.2.2.2. Se recorre la lista.

4.3.2.2.2.1. Se obtienes sus iteradores, como no hay ninguno se añade Comfort a la lista.

4.3.2.2.3. Se termina el recorrido.

4.3.2.3. Se termina el recorrido.

4.3.2.4. Se añade Sensing a la lista.

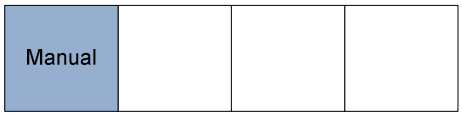


4.3.3. Se continua el recorrido.

4.3.3.1. Se obtienes sus iteradores.

4.3.3.2. Se recorre la lista.

4.3.3.2.1. Se obtienen sus composites.



4.3.3.2.2. Se recorre la lista.

4.3.3.2.2.1.Se obtienen sus iteradores, como no tiene ninguno se almacena Manual en una lista.

4.3.3.2.3. Se termina el recorrido.

4.3.3.3. Se termina el recorrido.

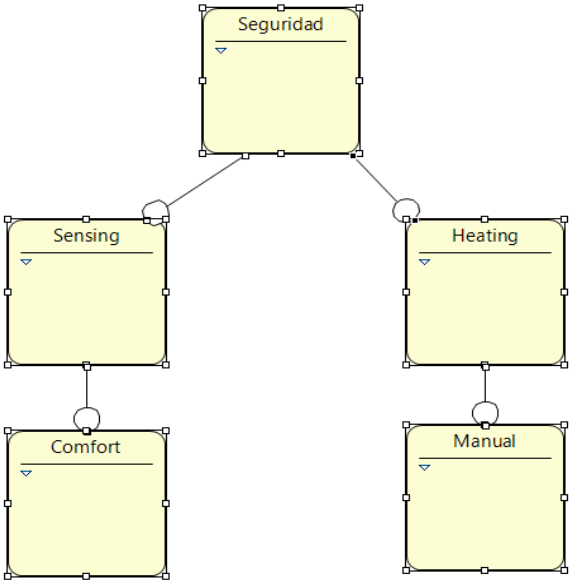
4.3.3.4. Se añade Heating a la lista

4.3.4. Se termina el recorrido.

4.4. Termina el recorrido.

4.5. Se añade Seguridad a la lista.

Al igual que antes, se recorrerá la lista, cogiendo sus nombres, realizando una búsqueda de estos en el modelo previamente transformado obteniendo así una lista de Features y se agregaran al producto vacío, teniendo el segundo producto y ultimo transformado. El diagrama que equivale al resultado de esta transformación es:



5.Se termina el recorrido.

Una vez transformados todos los productos se devuelve una lista de estos, el usuario ya podrá trabajar con ellos libremente, realizando todas la preguntas FaMa deseadas, siendo esta transformación trasparente para él.

5.4.Análisis del Smart Home con FaMa

Utilizando las preguntas que se han explicado en el apartado 3, se puede analizar los modelos del caso de estudio.

Sobre el modelo de variabilidad:

1.¿Es un modelo valido?

-El modelo es válido.

2.¿Cuantos productos validos se pueden hacer con el modelo?

-Un total de 44 productos.

3.¿En cuantos productos puede aparecer SmartHome, Security, Confort?

-SmartHome -> 44

-Security -> 32

-Confort -> 15

4.¿Cual es la variabilidad?

-Es de 0.086105675

5.¿Tiene algún error?

-No

Sobre el producto:

Hay dos productos:

Seguridad

1.¿Es la configuración valida?

-La configuración es valida

2.¿Es el producto valido?

-El producto es valido

3.¿Tiene algún error?

-No

Bienestar

1.¿Es la configuración valida?

-La configuración es valida

2.¿Es el producto valido?

-El producto es valido

3.¿Tiene algún error?

-No

6. Conclusiones y futuros trabajos

A lo largo de este documento, se ha hablado de una herramienta de transformación de modelos CVL en modelos FaMa.

Explicando en primer lugar la tecnología utilizada, los elementos tanto de CVL como de FaMa y sus equivalencias.

En segundo lugar se ha expuesto el proceso de transformación distinguiendo la transformación del modelo de variabilidad, de las resoluciones y el de las restricciones.

En tercer lugar las preguntas de FaMa, han sido explicadas, utilizando pequeños ejemplos e interpretadas.

En último lugar, ha sido expuesto un caso de estudio, practico y completo, detallando toda su transformación, desde el modelo a las resoluciones, concluyendo con una lista de preguntas y resultados, obtenidos al ejecutar las preguntas FaMa sobre el modelo ya transformado, comprobando que los resultados son los esperados y que la transformación realizada es correcta.

En conclusión, la transformación efectuada es correcta, pero quedan muchos aspectos en el aire que no se han resuelto. Se ha marcado la línea, faltaría profundizar en ella y conseguir una transformación más completa, es probable que no se puede llegar a una transformación total de todos los modelos CVL en modelos FaMa. Hay algunos que no se podrán transformar, en su totalidad. Aunque no se consiga la transformación total, esto seguirá resultando muy interesante ya que aunque sea un modelo parcial, poder realizar las preguntas FaMa puede ser muy útil.

Por último, los futuros trabajos, son muchos como se ha mencionado, solo se ha marcado la línea, entre ellos vale la pena destacar 4:

- Los atributos de CVL , y su transformación en los atributos FaMa ha quedado como tarea pendiente, debido a que han sido incorporados recientemente a FaMa y la información encontrada sobre ellos ha sido escasa. Pero es uno de los aspectos importantes, que aportan numerosas funcionalidades a los modelos que se encargan de representar la variabilidad.

- Restricciones, solo se han aceptado dos tipos de restricciones, de implicación y de exclusión, el resto han sido descartadas en el proceso de transformación. Tanto CVL como los modelos FaMa soportan otras restricciones, pero su conversión no es directa habría que investigar a fondo los funcionamientos de ambas. Debido a falta de tiempo no se ha podido llevas a cabo.

- Hay diferencias entre los modelos CVL y FaMa, por ejemplo la expuesta en una de las cuestiones FaMa como que CVL acepte características duplicadas y Fama no.

- Temas de efectividad en el recorrido y de optimización de la memoria utilizada, ya que en casos tan sencillos como los utilizados, la transformación es despreciable en términos de costes temporales, pero con modelos muy grandes este coste o el coste espacial podría ser alto, debido al uso de métodos recursivos.