

Meta-modelo para el Análisis de Variabilidad Guiado por Metas

Bruno González-Baixauli
Universidad de Valladolid
bbaixauli@infor.uva.es

Mayo de 2006

Versión 0.5

En este artículo se presenta un meta-modelo para dar soporte al análisis de variabilidad. El meta-modelo se basa en dos enfoques: el enfoque intencional (orientado a metas [29]) y el enfoque orientado a aspectos [14] (denominado de aspectos tempranos o *early-aspects* [21] cuando se aplica a la fase de requisitos). El enfoque intencional permite un estudio de la variabilidad desde un grado de abstracción más alto, permitiendo el estudio de un mayor espacio de variabilidad y, además permite, aplicando el *NFR Framework* [3] estudiar la variabilidad desde un punto de vista de los requisitos no funcionales (RNF), un aspecto en el que fallan la mayoría de las técnicas de análisis de la variabilidad existentes. El segundo enfoque, la orientación a aspectos nos sirve para solucionar el problema de integrar la parte funcional y no funcional en los modelos de metas. Además, nos permite aprovecharnos de las ventajas de este tipo de enfoques, obteniendo una mejor separación de intereses y, relacionado con la variabilidad, una mayor escalabilidad, permitiendo seleccionar los aspectos de interés para cada caso concreto. Por tanto, el meta-modelo debe incluir los conceptos de los enfoques intencionales (en este caso los propuestos en [30]) y conceptos de *early-aspects* como la separación de intereses, las relaciones entre intereses (relaciones transversales o *crosscutting*) y la composición de intereses. En este artículo se muestra dicho meta-modelo como primera fase para la definición de estrategias para el análisis de variabilidad basado en metas y aspectos.

Índice

1. Introducción

2

2. Trabajos Relacionados	5
2.1. Análisis de Variabilidad	6
2.2. NFR Framework y Aspectos	6
3. Requisitos Iniciales y Decisiones de Diseño	9
4. Descripción del Meta-modelo	11
4.1. Proyecto y Modelos	12
4.1.1. Clases	12
4.1.2. Relaciones	14
4.1.3. Restricciones	14
4.1.4. Decisiones de Diseño	15
4.2. Elementos	16
4.2.1. Clases	16
4.2.2. Relaciones	18
4.2.3. Restricciones	19
4.2.4. Decisiones de Diseño	19
4.3. Relaciones	20
4.3.1. Clases	20
4.3.2. Relaciones	23
4.3.3. Restricciones	23
4.3.4. Decisiones de Diseño	24
5. Reglas de Composición	25
5.1. Reglas para la Creación de la Vista	26
5.2. Reglas para la Creación de Referencias a Elementos	27
5.3. Reglas para la Creación de Relaciones a partir de Relaciones Aspectuales .	28
5.4. Reglas para las Relaciones OR	32
5.5. Reglas para la Creación de Referencias a Relaciones	35
6. Ejemplo	37
7. Conclusiones	41
Tabla Resumen	43

1. Introducción

Durante los últimos años ha aumentado el interés sobre la variabilidad de los sistemas software. Este interés se debe a varios factores: por un lado, los sistemas software tienden a integrar una funcionalidad cada vez mayor, debido por la necesidad de ofrecer aplicaciones con un mayor valor añadido frente a los competidores. Esta mayor funcionalidad permite una mayor flexibilidad a los usuarios, pero también puede producir una sobrecarga sobre los usuarios con un gran número de funcionalidades que nunca van a utilizar.

Otro grupo de factores se debe a la necesidad de abaratar costes, por ello se busca agrupar productos con un núcleo similar en grupos de productos, de forma que se pueda reutilizar la mayor parte y que la diferencia entre un producto u otro venga dada por extensiones y adaptaciones. El primer caso es el que se estudia desde el desarrollo de sistemas personalizables [15], donde, por lo general, la funcionalidad se decide configurando el producto en tiempo de ejecución. El segundo caso, normalmente a mayor escala, es estudiado por el desarrollo de líneas de producto software (LP) [1] o familias de productos software, donde el énfasis está en la reutilización. En estos casos, la funcionalidad se decide, por lo general, antes de la entrega del producto, ofreciendo un catálogo de productos según los clientes esperados. Por simplificar, en este documento se hablará en general de *software con variabilidad* para referirse a ambos tipos de software.

La diferencia principal del software con variabilidad con respecto a la idea tradicional del software es que es necesario analizar cual va a ser esa variabilidad y cual va a ser la parte común para todas las instancias de ese software. Este problema se ha abordado tradicionalmente desde la fase de diseño como la búsqueda de mecanismos para permitir una funcionalidad variable [23], pero se ha prestado poco interés a la necesidad de dicha variabilidad. El estudio de los requisitos de la variabilidad trata de detectar esa necesidad bajo la premisa de que cuanto antes se detecte, mejor se podrá tratar. La mayoría de los trabajos en este área se pueden descomponer en tres grandes grupos: los que utilizan modelos de *features* [13], los que utilizan casos de uso [12] o los que integran ambos [8]. Estos grupos se estudian en profundidad en la sección 2.1, pero en general, no son suficientes para el análisis de los requisitos de variabilidad: los primeros requieren un gran conocimiento del dominio y están más enfocados hacia la definición de la arquitectura. Los segundos están más enfocados hacia el análisis y elicitación de requisitos, pero fallan en la representación global de la variabilidad. Por último, los enfoques que integran ambas ideas logran un mejor análisis de requisitos con los casos de uso y una representación global con las *features*, pero fallan, al igual que las otras propuestas, en un concepto fundamental, el análisis de los requisitos no funcionales (RNF).

Esta limitación es muy importante, puesto que los RNF son un factor clave en la existencia de la variabilidad, por ejemplo, al proporcionar distintos grados de seguridad, o eligiendo una mayor facilidad de uso frente a una mayor rapidez de entrada, etc. La razón principal de esta situación es que la mayoría de técnicas que estudian los requisitos de variabilidad han evolucionado a partir de las técnicas de diseño y de las prácticas tradicionales de la ingeniería del software, con una visión más funcional de los sistemas software. Frente a esta situación, *nuestra propuesta es aplicar mecanismos y técnicas de la Ingeniería de Requisitos (IR) al estudio de la variabilidad*. En este sentido, una de las propuestas más exitosas es la IR Orientada a Metas (OM) o *goal-oriented*, donde las metas proporcionan una visión intencional, que permite a los interesados expresar sus necesidades de una manera más natural, centrándose en lo que quieren, es decir sus objetivos, frente a la manera de alcanzarlos o requisitos, que se pueden derivar a partir de los objetivos. Este mecanismo ha demostrado varias ventajas como su mayor estabilidad, la posibilidad de analizar distintas alternativas y verificar la compleción de un conjunto de requisitos con respecto a los objetivos planteados [29], o un tratamiento de los RNF y los conflictos más natural aplicando el *NFR Framework* [3]. Además, su aplicación al

campo de la variabilidad ya ha sido explorada en trabajos como [11, 7, 15].

No obstante, al explorar dicha línea en trabajos anteriores [7, 6], hemos observado problemas al integrar modelos de metas funcionales con los modelos de metas no funcionales (modelos de *softgoal* o *Softgoal Interaction Graphs (SIG)* en el *NFR Framework*). Estos problemas se deben a que las soluciones de los SIG (operacionalizaciones en el *NFR Framework*) modifican el modelo funcional, pero no tienen porqué estar dentro de él. Esto provoca que, en muchos casos se utilice un único modelo con todas las metas funcionales, tareas y softgoal juntas, lo que provoca un cierto enredo de las distintas partes del sistema. Este es un problema muy parecido al que tratan los enfoques orientados a aspectos (OA) [14], que propugnan una mayor separación de intereses, uno de los principios sobre los que se basa la ingeniería del software. De esta forma, la OA busca maneras de aplicar esa separación de intereses a partes del sistema que por sus características están muy entrelazadas con otros intereses y es complejo de separar en módulos. La solución ideada es localizar dichos intereses, extraerlos del código y establecer reglas de composición (*weaving*) donde se especifica dónde y cómo se debe insertar de nuevo el código antes de compilar. La misma idea es aplicada a los requisitos en los enfoques de aspectos tempranos (*early-aspects*) [21].

Nuestra propuesta trata de aplicar las ideas de la OA a los modelos de metas para solucionar el problema de integrar los distintos tipos de modelos y para mejorar la separación de intereses. Este documento se enmarca dentro de nuestra propuesta para el desarrollo de un marco de trabajo para el análisis de la variabilidad desde un enfoque de requisitos. Esta propuesta se basa en tres partes:

1. Definición de un meta-modelo que de soporte al análisis. Este meta-modelo debe permitir modelar la variabilidad utilizando metas e intereses, y la composición automática de dichos intereses.
2. Definición de técnicas para el análisis de la variabilidad. Estas técnicas deben permitir explorar la variabilidad a partir de los objetivos de los usuarios y seleccionar la parte del espacio de variabilidad que necesita el sistema con variabilidad.
3. Definición de técnicas de configuración del sistema con variabilidad que permitan decidir entre las variantes del sistema con variabilidad según cada usuario.

Las ventajas que esperamos que proporcione este marco de trabajo para la variabilidad son:

1. Permitir aumentar el espacio de variabilidad estudiado, al incrementar el nivel de abstracción del análisis de variabilidad. El análisis se inicia desde un nivel de abstracción mayor (los objetivos de los usuarios) y no desde maneras establecidas de alcanzar dichos objetivos (la funcionalidad), pudiéndose hallar nuevas formas de alcanzarlos y por tanto aumentando el espacio de variabilidad.
2. Permitir un estudio más profundo de cada interés, al realizarse su estudio por separado.

3. Permitir la aplicación del NFR Framework al análisis de requisitos de variabilidad no funcionales, ayudando al análisis de conflictos. Además, es posible utilizar catálogos de RNF definidos utilizando este enfoque.
4. Ayudar a restringir la explosión combinatoria, pudiendo realizar selecciones en los puntos de variabilidad antes de su combinación, o seleccionando solo los intereses que necesite el usuario.
5. Proporcionar un mecanismo más objetivo para estudiar la interacción de intereses complementarios o contradictorios gracias a la combinación automática de intereses.

En este documento nos centramos en la primera parte del marco de trabajo, presentando un meta-modelo para el modelado de la variabilidad y las reglas de composición que permiten crear e integrar varios modelos. Las características básicas del meta-modelo son:

- Utilización de un enfoque OM que permita modelar intereses, es decir aspectos particulares, de los requisitos mediante modelos de metas. Estos intereses pueden ser tanto requisitos funcionales como no funcionales. Por tanto debe incluir los elementos de modelado de las ideas orientadas a metas, fundamentalmente la idea de meta, softgoal y tarea y las posibles relaciones entre ellas
- Aplicación de un enfoque OA para permitir relacionar modelos funcionales y no funcionales. La utilización de un enfoque OA no busca encontrar aspectos candidatos como en las propuestas de aspectos tempranos, sino utilizar las ideas OA para mejorar la separación de intereses y para permitir la integración de modelos. Por tanto, debe incluir distintos modelos para la separación de intereses y relaciones entre intereses.
- Integración automática de modelos en vistas integradoras. Las vistas permiten ver varios intereses a la vez, pero también realizar operaciones sobre ellos, eliminando elementos sin modificar los modelos iniciales.

La estructura del documento es: en la próxima sección se describe el trabajo relacionado, centrado en aquellas que relacionan el *NFR Framework* con aspectos. En la sección 3 se discuten los requisitos iniciales del meta-modelo y las decisiones de diseño generales que se tomaron en función de los requisitos. La sección 4 contiene la descripción del meta-modelo, mientras que en la siguiente se describen las reglas de composición entre modelos. Estas ideas son mostradas en la sección 6 aplicadas a un ejemplo y por último se presentan las conclusiones del trabajo en la sección 7.

2. Trabajos Relacionados

Para estudiar los trabajos relacionados, los hemos dividido en dos grandes grupos: por un lado aquellos que definen metamodelos para el análisis de requisitos de variabilidad y por el otro aquellos que integran las ideas del *NFR Framework* con las ideas de la OA y de aspectos tempranos. En las siguientes sub-secciones describiremos y discutiremos las distintas propuestas.

2.1. Análisis de Variabilidad

En el análisis de variabilidad existen dos grandes familias de propuestas: los que se basan en *features* y los que se basan en casos de uso, aunque también hay propuestas que integran ambos enfoques.

En cuanto a las primeras, utilizan *features*, que se definen como características o conceptos destacados y distinguidos visibles a varios interesados [13]. Estas *features* se organizan en diagramas jerárquicos And-Or, donde los nodos And dan la parte común y los Or la variable. De esta forma, son utilizados para definir la arquitectura de referencia del sistema y los componentes reutilizables instanciables durante el desarrollo de aplicaciones con variabilidad [13]. El problema de este enfoque es que requiere un gran conocimiento del dominio y que está más enfocada hacia la definición de la arquitectura que a la elicitación y definición de requisitos, por lo que son difíciles de aplicar directamente. En la mayoría de estas propuestas, los RNF aparecen como *features* de alto nivel, aunque posteriormente no se describe como afectan al resto de las *features*. [Ahondar en el tema de los RNF]

Las propuestas que utilizan casos de uso están más enfocadas hacia la elicitación de los requisitos. Por ejemplo, Jacobson [12] utiliza el mecanismo *extend* de los casos de uso tradicionales para representar gráficamente la variabilidad, describiendo los detalles de los puntos de variación en la descripción. En cambio, Halmans y Pohl [10] proponen indicar explícitamente los puntos de variación y las distintas variantes en el modelo de casos de uso para obtener una mejor visión de la variabilidad, para lo que extienden su notación. En ambos casos, las técnicas facilitan la elicitación de requisitos representando la variabilidad en cada caso de uso, pero fallan en la representación global de la variabilidad al estar contenida en diversos documentos (en cada caso de uso). En cuanto a los RNF, existen propuestas para su estudio dentro de los casos de uso, pero no están muy extendidas y no han sido aplicadas al estudio de la variabilidad.

También existen otras técnicas que unen ambos enfoques como [8]. En este caso, los casos de uso permiten elicitar la variabilidad a partir de los deseos de los interesados y se mantiene la variabilidad global en modelos de *features*, obteniéndose las ventajas de ambos enfoques. En cualquier caso, siguen sin estudiar la variabilidad derivada de los RNF.

2.2. NFR Framework y Aspectos

Existen varias propuestas que proponen integrar el NFR Framework con las ideas de aspectos. Aquí es importante separar entre las propuestas que utilizan modelos de metas como la base para modelar los requisitos funcionales y los que utilizan otros modelos. A nosotros nos interesan los que utilizan modelos de metas puesto que se pueden utilizar para estudiar la variabilidad al utilizar una estructura jerárquica con partes obligatorias (descomposición AND) y opcionales (descomposición OR), aunque en este apartado se describen también los segundos.

Entre las propuestas que utilizan otros modelos está la de Sousa et al. [22] que utiliza casos de uso para la parte funcional. Para los RNF utiliza el *NFR Framework* y las

Cuadro 1: Tabla de composición usada en [22]

CROSSCUTTING REQUIREMENT: # N <NAME>			
AFFECTED ARTIFACT	CONDITION (OPTIONAL)	COMPOSITION RULE OPERATOR	AFFECTED POINT
UC #N<name>	condition of the composition	{overlap.after overlap.before override}	Step of the Scenario

considera las operacionalizaciones de los modelos de *softgoal* como requisitos transversales (aspectos). La relación entre ambos se realiza por medio de una tabla que define, para cada operacionalización dónde y cómo se componen como se muestra en la tabla 1. En este caso se define como se relacionan, pero no un mecanismo automático para la integración de la parte funcional y la no funcional.

Por otra parte, Brito et al. [2], dentro de las propuestas de aspectos-tempranos (*early-aspects*), utilizan el *NFR Framework* como una de las posibles formas de especificar intereses (*concerns*). Los intereses son considerados como “aspectos candidatos” durante la fase de requisitos. De esta forma, en su propuesta primero se identifican los intereses, que son especificados de distintas formas dependiendo de su naturaleza (por ejemplo utilizando el *NFR Framework* si son RNF). Al utilizar diferentes tipos modelos, el análisis entre intereses se realiza mediante tablas como la mostrada en la tabla 2, sin poder aplicar las ideas del NFR Framework entre distintos intereses. Por tanto, necesitan un mecanismo distinto para estudiar las interrelaciones, que se basa en indicar si un interés afecta (positiva o negativamente) a los demás dentro de la tabla. Además, no proporcionan un mecanismo automático de composición, que tiene que ser definido para cada tipo de modelos.

Estos enfoques tienen la característica común de que no utilizan modelos de metas para la parte funcional, lo que es necesario en nuestro caso para definir la variabilidad. Un enfoque que sí utiliza modelos de metas funcionales es el propuesto por Navarro et al. [16] dentro de la propuesta para la parte de requisitos de ATRIUM . En este caso, todos los intereses se definen en el mismo modelo de softgoals, incluyendo la parte funcional, que se describe utilizando un enfoque similar a KAOS [25]. El problema de este enfoque es que no se diferencian los distintos intereses en distintos modelos, obteniéndose una menor separación de aspectos como se muestra en la Figura 1.

Por último, Yu et al. [30] exploran la detección de aspectos a partir de modelos de metas ya existentes. Para ello presenta un modelo simplificado (el modelo V-Graph) y un algoritmo. En cuanto al modelo en si, no aplica ideas OA al modelado de metas, sino que solo busca la detección de los aspectos, integrando todos los intereses (funcionales o no funcionales) en un mismo modelo. Un ejemplo de un modelo de este tipo se puede observar en la Figura .

Cuadro 2: Tabla de composición usada en [2]

Table 1: A template to describe concerns

Name	The name of the concern.
Source	Source of information, e.g. stakeholders, documents, domain, catalogues and business process.
Stakeholders	People that need the concern in order to use the system.
Description	Explains the intended behaviour of the concern.
Classification	Helps the selection of the most appropriate approach to specify the concern. For example: functional, non-functional, goals.
Contribution	Represents how a concern can be affected by other concern. This contribution can be positive (+) or negative (-)
Priority	Expresses the importance of the concern for the stakeholders. It can take the values: <i>Very Important, Important, Medium, Low</i> and <i>Very Low</i> .
Required concerns	List of concerns needed by this concern.

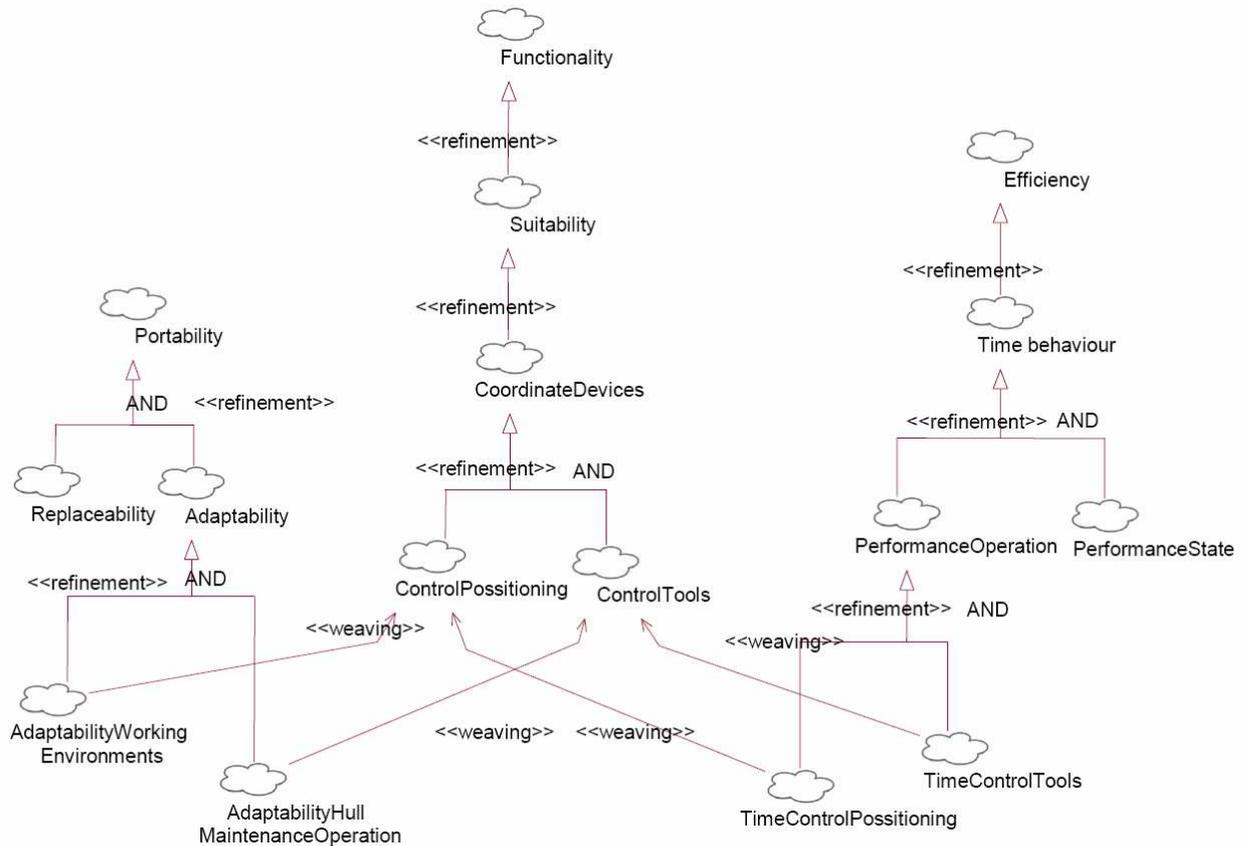


Figura 1: Parte de un modelo utilizando el enfoque de Navarro et al en [16].

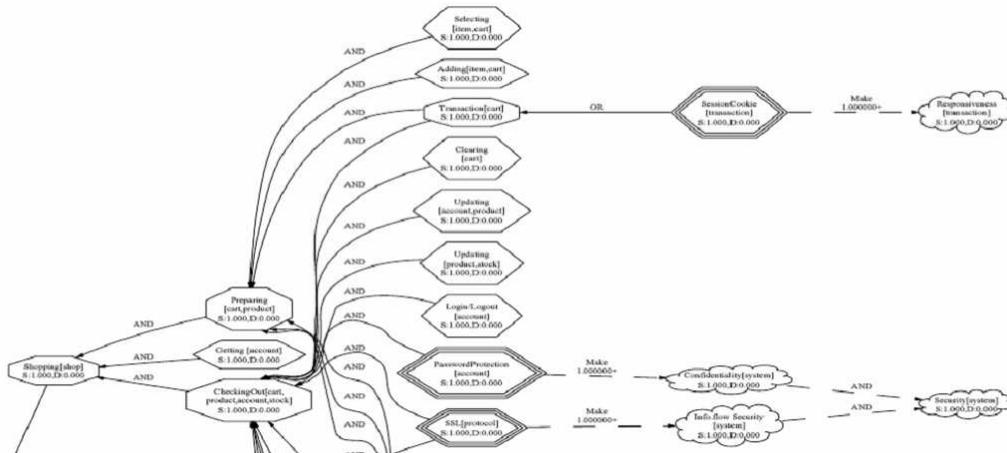


Figura 2: Ejemplo parcial de un modelo utilizando las ideas de Yu et al. mostrado en [30].

3. Requisitos Iniciales y Decisiones de Diseño

Los requisitos iniciales que debe cumplir el meta-modelo son permitir:

1. Modelar los elementos básicos de la orientación a metas.
2. Relacionar la parte funcional y la parte no funcional.
3. Modularizar los modelos utilizando dos mecanismos:
 - a) Modularización clásica: una parte de un modelo se puede modelar por separado en otro modelo y el modelo “padre” mantiene una referencia.
 - b) Modularización aspectual: una parte de un modelo afecta a otro modelo, modificando sus elementos.
4. Crear vistas que permitan integrar varios modelos por medio de la composición de módulos (modelos) tanto clásicos como aspectuales. En estas vistas se debe poder eliminar elementos para habilitar la selección entre alternativas, pero sin modificar los modelos base de las vistas.
5. Establecer relaciones de restricción entre tareas, que especifiquen su compatibilidad, como en los modelos de features [13].

Siguiendo estos requisitos iniciales, se decidieron una serie de decisiones de diseño generales al meta-modelo (en la descripción del meta-modelo también se muestran decisiones más específicas). Estas decisiones de diseño se explican para cada punto de los requisitos iniciales:

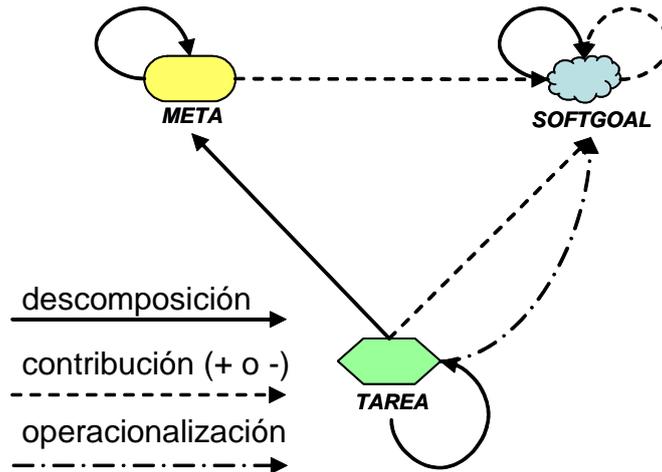


Figura 3: Esquema de los elementos OM y las posibles relaciones entre ellos. Modificado a partir de [30].

1. Para el primer punto nos hemos basado en el modelo V-Graph [30], que simplifica los enfoques tradicionales OM, definiendo solo tres tipos de elementos (metas, tareas y softgoals), y las relaciones entre ellos. Estos elementos son suficientes para el análisis de la variabilidad, de forma: los modelos funcionales, formados por estructuras jerárquicas de metas y tareas, permiten definir el espacio de variabilidad del sistema de forma similar a los modelos de features [13]. Esto se debe a que tienen solo dos posibles valores (alcanzado o no para las metas y presente o no para las tareas) y la estructura jerárquica, con descomposiciones And/Or permite definir la parte común y variable, respectivamente. Por otra parte, los modelos no funcionales, están formados por jerarquías de softgoals que representan los distintos RNF, y tareas, que representan soluciones que satisfacen en cierto grado los softgoals (y se relacionan por medio de la relación operacionalización). Los elementos funcionales (metas y tareas) se relacionan con los softgoals por medio de la relación contribución, indicando de que forma los afectan. Esta relación nos permite estimar la satisfacción de los RNF para cada variante, y por tanto seleccionar la óptima como se hace en [7].

Un esquema de los elementos del modelo y las relaciones posibles entre ellos se muestra en la Figura 3. Es importante señalar, que respecto al modelo V-Graph, no se permiten relaciones de contribución entre elementos funcionales, esto se debe al carácter binario de su valor de satisfacción (se satisface o no). Así, aunque a priori, parece que una tarea o meta puede ayudar a la satisfacción de otra, al no existir valores parciales en este tipo de elementos (que sería el resultado en este tipo de relaciones), no tiene sentido permitirlos, y en el caso de ser necesaria, se tendría que utilizar un softgoal, que sí que permite valores parciales.

La mejora respecto al modelo V-Graph es la separación de intereses en modelos independientes (denominados modelos de interés), y la correspondiente composición en vistas. Esta separación permite realizar el modelado de forma independiente y, si son suficientemente genéricos, la posibilidad de su reutilización (por ejemplo, utilizando catálogos de RNF), pero es necesario un mecanismo para integrar los distintos modelos, como se describe en el próximo punto.

2. Para relacionar modelos funcionales y no funcionales, definimos una relación, inspirada en los enfoques aspectuales, llamada relación aspectual. Esta relación indica que un RNF (un *softgoal*) afecta o puede afectar a un requisito funcional (meta o tarea), de forma que sus operacionalizaciones modifican la manera de alcanzar el requisito funcional. Así, al realizar la composición, la parte funcional del aspecto no funcional se añadirá a la jerarquía funcional utilizando una relación OR, pudiéndose seleccionar una o varias. Además, la fuerza de la operacionalización indica como contribuye a la satisfacción de los RNF, como en las relaciones de contribución positivas. Aquí nos aprovechamos de la abstracción de los elementos intencionales, que no requieren una descripción detallada.
3. Modularizar partes de los modelos en nuevos modelos: al ser los modelos de interés jerárquicos, el nodo raíz representa a todo el modelo. Esto permite implementar fácilmente la modularidad que hemos llamado *clásica* utilizando las relaciones generales, pero con un elemento que pertenece a otro modelo (y que es su raíz). La aspectual se implementa mediante la relación aspectual anteriormente explicada.
4. La creación de vistas obliga a definir un nuevo tipo de modelo (vista), cuyos elementos son referencias a los elementos y relaciones de otros modelos. Estas referencias nos permiten eliminar elementos en las vistas, sin modificar los elementos de los intereses. Hay que señalar que, en general, las vistas sólo contienen referencias, pero es posible crear nuevas relaciones debido a las relaciones aspectuales, que pertenecerán a la nueva vista.
5. Por último, se crea un nuevo tipo de relaciones para indicar las relaciones tipo restricción. Estas relaciones solo existen entre tareas, puesto que son los elementos que están o no en la variante, frente a las metas y softgoals que se calcula a partir de ellas. En el caso de los softgoals, las relaciones de contribución tienen los mismos efectos, por ejemplo *make* implica que si una se satisface, la otra también; y *break*, que si una se satisface, la otra no. En cuanto a las metas, se podrían calcular indirectamente a partir de las tareas.

4. Descripción del Meta-modelo

El meta-modelo se basa en cuatro tipos de elementos principales: proyecto, modelos, elementos y relaciones. El *proyecto* organiza un análisis de variabilidad, incluyendo los modelos de dicho análisis. Los *modelos* son la base de la separación de intereses y es donde se incluyen el resto de elementos. Los *elementos* son los componentes principales

de modelado, es decir, metas, tareas y *softgoals*. Por último, las *relaciones* conectan elementos utilizando las asociaciones típicas de los modelos de metas (descomposición), las específicas de modelos de *softgoals* (contribución y operacionalización), pero también las relacionadas con los enfoques orientados a aspectos y las restricciones propias del análisis de variabilidad (requiere y mutuamente exclusivas). La descripción del meta-modelo se realizará en función de estos tres tipos de elementos.

4.1. Proyecto y Modelos

Un *proyecto* agrupa el modelado de un sistema con variabilidad, que se compone de modelos y de relaciones aspectuales entre elementos del modelo. Un *modelo* es una representación de parte del sistema, que puede ser tanto funcional o no funcional. Se compone de componentes y relaciones entre dichos componentes. Existen dos tipos de modelos, los modelos de interés (*Concern*) que representan una parte de interés del sistema, y son la base de su definición, puesto que es donde se modelan sus elementos; y las vistas (*View*) o modelos generados, que son el resultado de la composición de otros modelos, y la base del análisis de variabilidad. Las vistas no incluyen nuevos elementos, sino referencias y solo pueden incluir nuevas relaciones como resultado de las relaciones aspectuales.

En la figura 4 se muestra la parte del meta-modelo correspondiente. Un modelo está definido por su nombre y muestra un conjunto de componentes y relaciones entre dichos componentes.

4.1.1. Clases

Project (Proyecto) proyecto de análisis de un sistema con variabilidad. Está compuesto por un conjunto de modelos (*Model*) y por las relaciones aspectuales entre dichos modelos. Atributos: *name:string* (nombre del proyecto).

Model (Modelo) clase abstracta que define un modelo compuesto por un conjunto de elementos (*element*) y de relaciones OM (*GO_Relationship*) o de referencias (*ElementRef* y *RelationshipRef*). Atributos: *name:string* (nombre del modelo).

Concern (Interés) interés del sistema, es decir una parte del sistema que nos interesa estudiar por separado. Puede ser funcional, en cuyo caso estará compuesto por componentes *hard* (metas y tareas) o no funcional, en cuyo caso por lo general estará formado por *softgoals* y soluciones a dichas *softgoals*, es decir tareas (ver sección 4.2). Todo interés tiene una (y solo una) raíz (*root*), que es el elemento inicial de la estructura de jerarquía definida por relaciones de jerarquía (ver relación *hierarchy* en sección 4.3) .

View (Vista) modelo generado a partir de otros modelos, que pueden ser modelos de interés u otras vistas. Es decir, es un modelo donde se han aplicado las reglas de composición para obtener una nueva

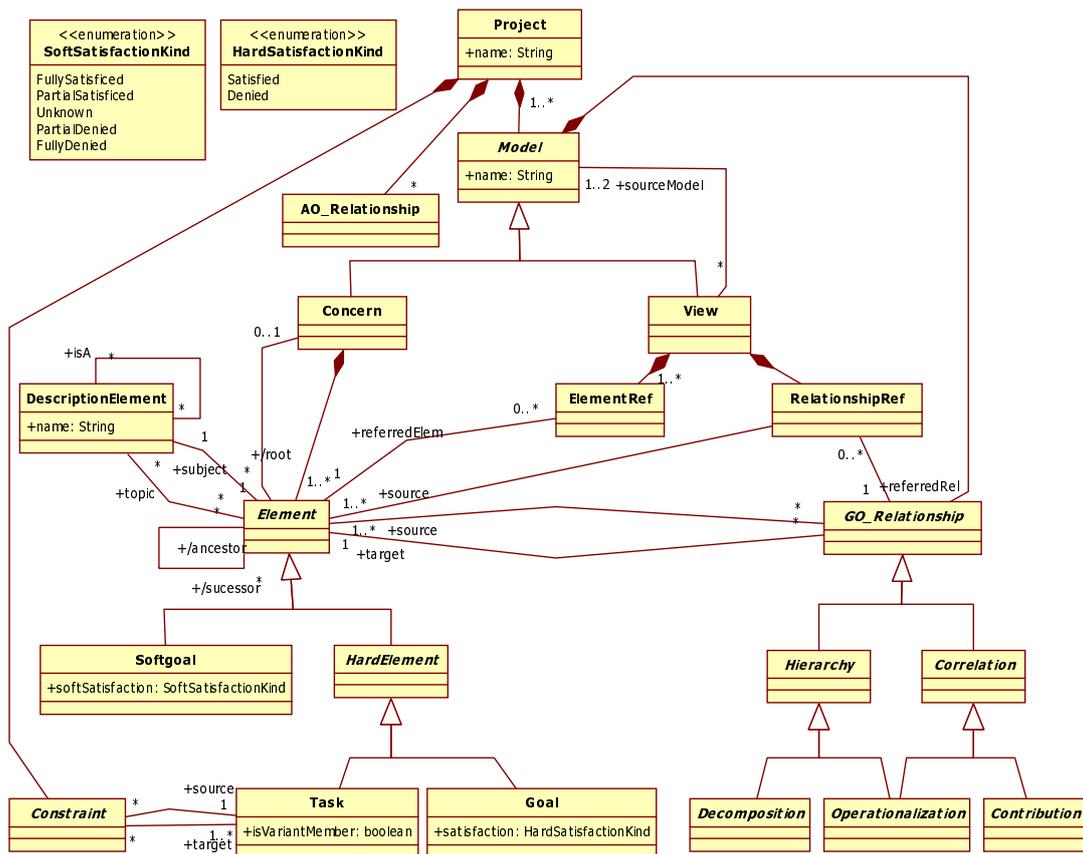


Figura 4: Parte del meta-modelo que representa la definición del proyecto y sus modelos.

vista del sistema que incluye referencias a los elementos y relaciones de los modelos compuestos, así como nuevas relaciones a generadas a partir de las relaciones aspectuales. Este tipo de modelos difieren de los modelos de interés en que solo hay referencias a elementos (y no elementos) y en que pueden existir más de una raíz (varias estructuras de descomposición). Las referencias permiten la eliminación de elementos y relaciones en la vista sin afectar a los modelos de interés iniciales.

4.1.2. Relaciones

`modelOwned` modelos que son parte del proyecto de análisis de la variabilidad.

`AORelationshipOwned` relaciones aspectuales en el proyecto de análisis de la variabilidad.

`constraintOwned` relaciones de restricción entre tareas parte del proyecto.

`sourceModel` modelos a partir de los cuales se ha obtenido una vista.

`elementOwned` elementos que son parte del modelo de interés. Es importante señalar que sólo se pueden definir elementos en los modelos de interés.

`relationshipOwned` relaciones que son parte del modelo. Esta relación puede aparecer en modelos de interés, pero también se pueden generar como resultado de la composición de varios modelos.

`root` relación derivada que representa el componente raíz del concern. Se define como el último antecesor (sin padre) de la estructura de jerarquía definida en el concern.

`elementRefOwned` referencias a elementos que pertenecen a la vista.

`relationshipRefOwned` referencias a relaciones que pertenecen a la vista.

4.1.3. Restricciones

1. Todo proyecto debe tener al menos un modelo (representado por la relación *modelOwned*).
2. Todo modelo de interés debe contener al menos un elemento (representado por la relación *elementOwned*).
3. Toda vista debe contener al menos una referencia a elemento (representado por la relación *elementRefOwned*).
4. Todo interés debe tener un y solo un elemento raíz (representado por la relación *root*).

- a) El nombre del interés coincide con el nombre del subject del elemento raíz.
5. Toda vista debe estar formada a partir de uno o dos modelos (representado en la relación *origin*).
 6. Las relaciones OM definidas en un modelo de interés (es decir, no generadas) pertenecen al interés al que pertenezca el elemento *source* (ver sección 4.3).
 7. Las relaciones OM definidas en un modelo de interés no están asociadas por la asociación *origin* a relaciones aspectuales.
 8. Toda relación OM generada en una vista está relacionada con al menos una relación aspectual por medio de la asociación *origin*.

4.1.4. Decisiones de Diseño

- Los modelos de interés son los modelos base, a partir de los cuales se definen las diferentes vistas del sistema. Por tanto, es donde se definen los elementos. Para definir dicho interés se descompone un interés inicial (representado por el elemento raíz) en nuevos elementos que son parte del interés.
 - Para representar que el componente raíz representa el interés, se define la restricción de que el nombre del interés debe ser igual al nombre del subject del componente raíz.
- Se utilizan referencias en las vistas para no duplicar los elementos base definidos en los intereses y para permitir su eliminación sin modificar los intereses iniciales. Esto nos va a permitir seleccionar parte de una relación OR en el análisis de la variabilidad.
- Una vista es el resultado de aplicar las reglas sobre uno o dos modelos.
 - Se permiten vistas a partir de un modelo para que se pueda modificar un interés antes de componerlo. Por ejemplo, si queremos seleccionar parte del interés, se crea la vista, y se eliminan las referencias que no sean útiles.
- Permitir relaciones OM entre elementos dentro de intereses distintos se permite aunque se obtenga una menor separación de intereses para simplificar el modelo. De esta forma obtenemos los dos tipos de modularidad, la modularidad clásica, similar a una llamada a procedimiento (un interés incluye a otro) y la modularidad aspectual, definida por las relaciones aspectuales. La otra opción sería considerar toda relación entre modelos (inter-modelos) como aspectual, pero al ser posible que todas las relaciones OM se den entre modelos y dentro de los modelos (intra-modelos) habría una duplicación de las relaciones y la necesidad de crear nuevas relaciones intra por cada relación inter en la composición. Para simplificar se optó por no diferenciar.

- Al tener que pertenecer a alguno de los modelos, se decidió que pertenecieran al elemento *target* por ser el que siempre tiene una cardinalidad de uno. Además, tiene un significado para todas los tipos de relación: en las descomposiciones la relación pertenece al modelo padre frente al sub-modelo; en las contribuciones al modelo de softgoals, obteniendo todas las contribuciones en el modelo de softgoals; y en las operacionalizaciones, también al padre (al modelo de *softgoals*), indicando las posibles soluciones.
- En cuanto a las relaciones aspectuales, pertenecen al proyecto para indicar que son independientes de los modelos de interés, al no tener el mismo problema de que puedan existir relaciones tanto dentro como fuera de los modelos de interés y para conseguir una mayor independencia.

4.2. Elementos

Elementos que describen un determinado interés. Un interés es descrito por un modelo de interés, que contiene un elemento raíz con el mismo nombre que el interés y por sus sub-elementos.

En la figura 5 se describen los elementos y sus características. Un elemento se define por un tema (*subject*) que lo describe y cero o más tópicos (*topic*) que refinan el subject (ambos instancias de la meta-clase descriptor o *DescriptionElements*).

4.2.1. Clases

Element (Elemento) clase abstracta que describe los elementos base de modelado. Un elemento se define por, al menos, un descriptor (*DescriptionElement*) que es su tema (*subject*). También se puede incluir cero o más tópicos (*topic*) que refinan el tema. Los descriptores forman el nombre del elemento con el formato *Tema/Tópicos*. Tiene dos subtipos: *HardElement* y *Softgoal*.

HardElement (Elemento Hard) clase abstracta que define aquellos elementos para los cuales se puede establecer de forma clara si se satisfacen o no o pertenecen a la variante. Son, por lo general, elementos funcionales. Se descompone en dos sub-tipos: Goal y Task.

Goal (Meta) representa un estado que se quiere alcanzar (o satisfacer) en el sistema. Atributos: *satisfaction:HardSatisfactionKind* (establece si el componente está satisfecho o no).

Task (Tarea) representa un proceso, operación, algoritmo, estructura, restricción, decisión de diseño, ... que puede ser claramente establecido como perteneciente al sistema. Una tarea puede ser parte de la solución de una meta o de la solución de un softgoal, en cuyo caso también se denomina operacionalización. Atributos: *isVariantMember:boolean* (indica si es parte o no de una determinada variante).

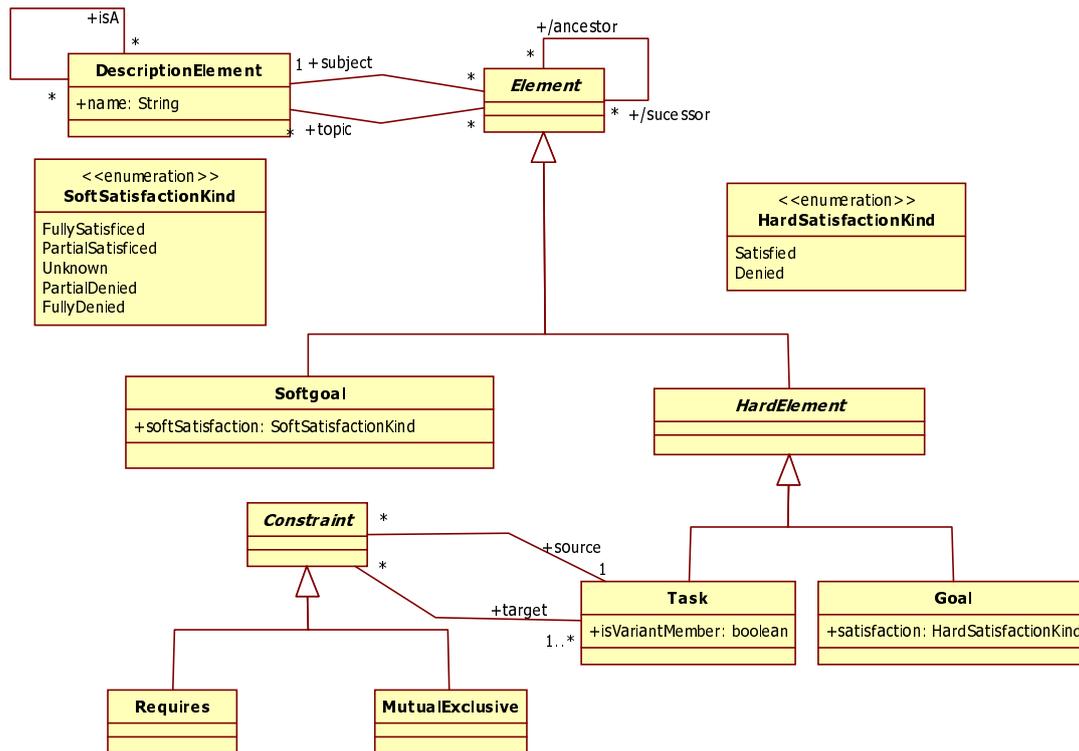


Figura 5: Definición de los elementos de modelado

Softgoal (Softgoal) metas que no tienen necesariamente un criterio claro y a priori de satisfacción. Este tipo de metas son normalmente utilizadas para representar RNF debido a su naturaleza subjetiva y relativa. Atributos: *softSatisfaction:SoftSatisfactionKind* (grado de satisfacción del componente. Debido a la naturaleza de este tipo de componentes se permite un posible rango de valores desde suficientemente satisfecho a claramente no satisfecho pasando por valores parciales).

DescriptionElement (Descriptor) describen un componente por medio de su nombre y las relaciones establecidas con otros elementos de descripción. Pueden ser relacionados entre ellos por medio de la relación *isA* estableciendo una estructura de herencia. Atributos: *name: string* (nombre que define al elemento de descripción).

Constraint (*Restricción*) clase abstracta que define restricciones sobre la presencia de varias tareas a la vez en una determinada variante. Se descompone en dos sub-tipos: *Requires* y *MutuallyExclusive*.

Requires (Requiere) restricción que indica que si una tarea (*source*) pertenece a la variante, las tareas relacionadas también deben pertenecer a la variante.

MutuallyExclusive (Excluyentes) restricción que indica que un conjunto de tareas no pueden estar a la vez en la misma variante. En este caso no importa que tarea es el fuente (*source*) y cuales son los destinos (*target*).

HardSatisfactionKind tipo enumerado que describe los posibles valores del atributo *satisfaction* de la clase *HardGoal*. Los posibles valores son: *Satisfied* (satisfecho) y *Denied* (no satisfecho).

SoftSatisfactionKind tipo enumerado que describe los posibles valores del atributo *softSatisfaction* de la clase *Softgoal*. Los posibles valores son: *FullySatisfied* (claramente satisfecho), *PartialSatisfied* (más satisfecho que no satisfecho), *Unknown* (desconocido), *PartialDenied* (más no satisfecho que satisfecho) y *FullyDenied* (claramente no satisfecho).

4.2.2. Relaciones

- subject todo elemento tiene un tema de tipo *DescriptionElement* que lo describe
- topic *DescriptionElement* que refina el tema de un elemento. Puede haber 0, 1 o varios.
- ancestor relación derivada que indica los elementos antecesores al elemento. Se calcula siguiendo las relaciones de jerarquía (ver clase *Hierarchy*) de hijo a padre (*child* a *parent*) para el elemento en cuestión y sus padres recursivamente.

successor	relación derivada que indica los elementos sucesores al elemento. Como la relación <i>ancestor</i> se calcula siguiendo las relaciones de jerarquía, pero en sentido opuesto, es decir de padre a hijo (<i>parent</i> a <i>child</i>).
isA	relación de herencia entre descriptores.
source	tarea fuente de la relación de restricción.
target	tarea destino de la relación de restricción.

4.2.3. Restricciones

1. Todo elemento tiene un y solo un tema (representada por la relación *subject*).
2. Toda restricción tiene un y solo un fuente (representada por la relación *source*).
3. Toda restricción tiene al menos un destino (representada por la relación *target*).

4.2.4. Decisiones de Diseño

- Las relaciones *ancestor* y *successor* son útiles para calcular el elemento *root* de los Concern (modelos de interés).
- *DescriptionElement* es necesario para permitir relacionar los subjects y topics de diferentes elementos. Esto nos permitirá relacionar catálogos de RNF con RF y obtener automáticamente relaciones aspectuales candidatas. Con este mismo objetivo, tanto *subject* como *topic* son de este tipo (y por tanto en algunos elementos actuarán como *subject* y en otros como *topic*).
- *Goal* y *Task* se han agrupado en un componente de mayor nivel al tener una naturaleza similar, aunque uno sea de mayor nivel de abstracción que el otro. Esta clasificación permite simplificar las restricciones de los *targets* y *source* de los distintos tipos de relaciones (ver siguiente sección). A pesar de ser similares, se utiliza un valor enumerado en las metas y un valor lógico en las tareas para representar que unos se calculan a partir de la pertenencia o no de las tareas en las variantes.
- Se han diferenciado los valores de satisfacción de los elementos meta y *softgoal*. Se podría considerar la existencia de una relación de herencia entre ellos (siendo *Satisfied* igual a *FullySatisfied* y *Denied* igual a *FullyDenied*), pero se ha preferido mantenerlos independientes debido a la diferencia de significado entre *satisfy* (satisfecho) y *satisfice* (suficientemente satisfecho).
- *Constraint* es necesaria para representar relaciones entre las tareas. Se podría considerar la idea de que las restricciones se dan porque afectan de forma diferente a las softgoals, pero esto no nos permite representar que dos tareas nunca pueden estar a la vez en la misma variante por razones técnicas, o que dos soluciones (operacionalizaciones) son incompatibles por su naturaleza.

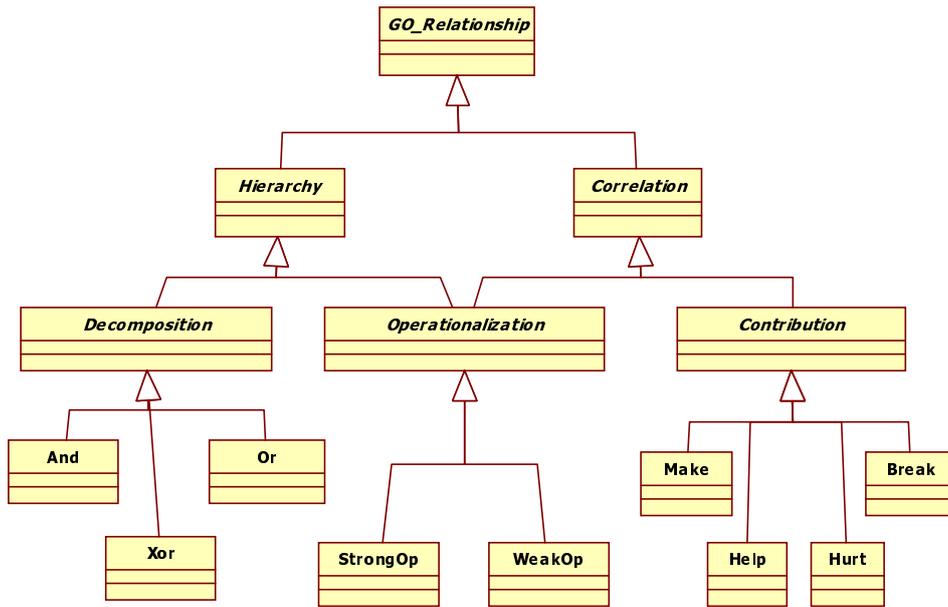


Figura 6: Tipos de relaciones entre elementos

4.3. Relaciones

Un interés se define por sus elementos, pero también por las relaciones que existen entre ellos. Sin dichas relaciones no se podrían descomponer, ni relacionar con otros elementos. Un esquema de las relaciones OM y los elementos que pueden relacionar se presentó anteriormente en la Figura 3.

En las siguientes figuras se describen los diferentes tipos de relaciones (figura 6), tanto orientadas a metas, como la relación aspectual y como se relacionan con los elementos (posibles fuentes y destinos). En la figura 7 se muestran los tipos finales de relaciones orientadas a metas posibles.

4.3.1. Clases

AO_Relationship (Relación Aspectual) clase que define las relaciones aspectuales que asocian modelos de interés no funcionales con funcionales, indicando que el *softgoal* puede modificar a la parte funcional. El resultado es que las operacionalizaciones de los softgoals se incluyen como descomposición opcional en la parte funcional.

GO_Relationship (Relación OM) clase abstracta principal que define una relación orientada a metas como un elemento de modelado con uno o más elementos fuente (*source*) y un elemento destino (*target*).

Hierarchy (Jerarquía) clase abstracta que define relaciones de tipo jerárquicas

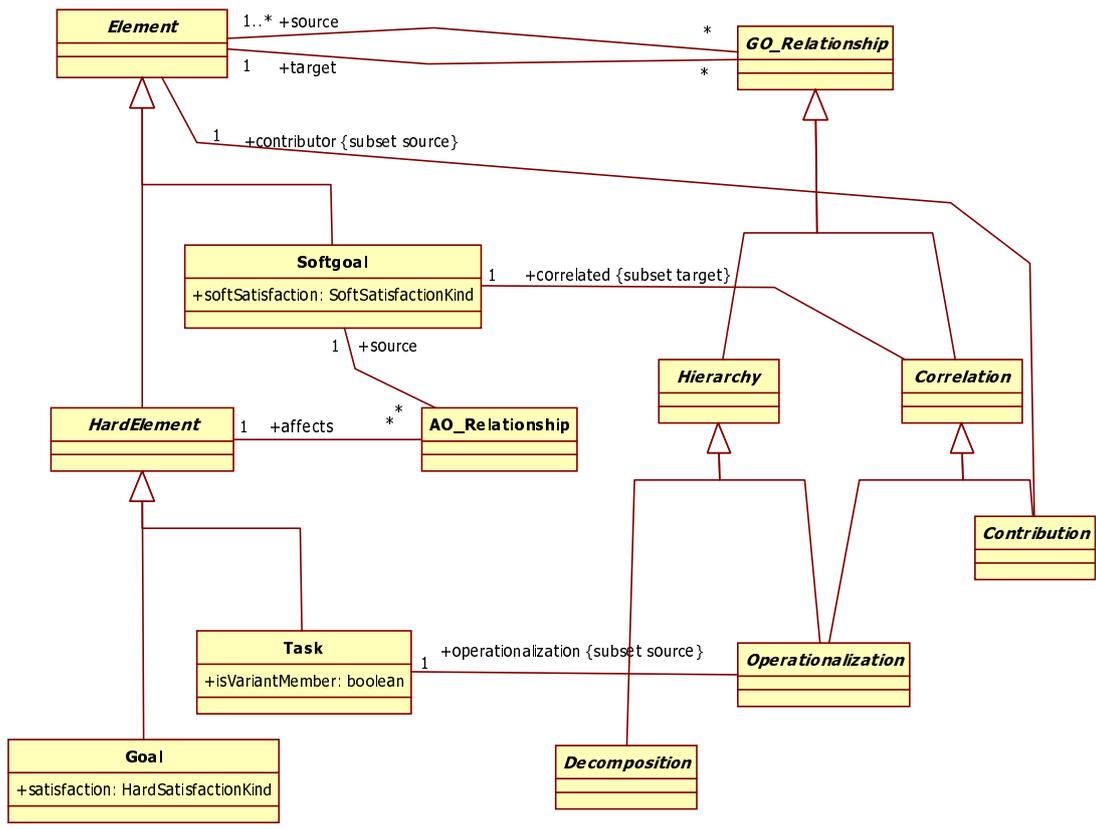


Figura 7: Tipos de relaciones orientadas a metas.

entre uno o más elementos que descomponen a otro elemento (padre). Este tipo de relaciones tiene dos sub-tipos: *Decomposition* y *Operationalization*.

Correlation (Correlación) clase abstracta que define relaciones que muestran como afecta un elemento de cualquier tipo a un *softgoal*. Tiene dos subtipos: *Operationalization* y *Contribution*.

Decomposition (Descomposición) clase abstracta que define como descomponen uno o más elementos a otro elemento. Tiene tres subtipos: *And*, *Or* y *Xor*.

And (And) clase que define que los elementos fuente (*source*) descomponen al destino y que deben pertenecer todos a la variante (o satisfacerse) para que el destino pertenezca (o se satisfaga).

Or (Or) clase que define que los elementos fuente (*source*) descomponen al destino y que, al menos uno debe pertenecer a la variante (o satisfacerse) para que el destino pertenezca (o se satisfaga).

Xor (Xor) clase que define que los elementos fuente (*source*) descomponen al destino y que debe pertenecer uno (y solo uno) a la variante (o satisfacerse) para que el destino pertenezca (o se satisfaga).

Operationalization (Operacionalización) clase abstracta de tipo jerarquía y correlación que define una (y solo una) posible solución (una tarea) para un softgoal dado. La fuerza de la operacionalización, es decir, si es suficiente para la satisfacción del softgoal o solamente ayuda, viene dada por el subtipo. Subtipos: *StrongOp* y *WeakOp*.

StrongOP (Operacionalización Fuerte) clase que define que la tarea fuente es una posible solución para el softgoal destino y que la solución es lo bastante buena como para satisfacer dicho softgoal.

WeakOP (Operacionalización Débil) clase que define que la tarea fuente es una posible solución para el softgoal destino, pero la solución no es lo bastante buena como para satisfacer dicho softgoal, solamente ayuda a su satisfacción.

Contribution (Contribución) clase abstracta que define como afecta un elemento a la satisfacción de un softgoal. El elemento puede afectar de forma positiva o negativa y en distintos grados (ver atributo *type*). La diferencia entre una operacionalización y una contribución positiva es que las primeras se conciben como soluciones para el *softgoal*, mientras que las segundas son efectos colaterales del

componente. Además, la fuente de una operacionalización debe ser una tarea, mientras que para una contribución también podría ser otro softgoal. Subtipos: Make, Help, Hurt y Break.

Make (Positiva Fuerte) clase que define que el elemento fuente afecta al valor de satisfacción del elemento destino de una forma positiva, y lo suficientemente fuerte como para que si no hay otras contribuciones negativas, el softgoal se satisfaga completamente.

Help (Positiva Débil) clase que define que el elemento fuente afecta al valor de satisfacción del elemento destino de una forma positiva.

Hurt (Negativa Débil) clase que define que el elemento fuente afecta al valor de satisfacción del elemento destino de una forma negativa.

Break (Negativa Fuerte) clase que define que el elemento fuente afecta al valor de satisfacción del elemento destino de una forma negativa, y lo suficientemente fuerte como para que si no hay otras contribuciones positivas, el softgoal no se satisfaga completamente.

4.3.2. Relaciones

source representa el elemento fuente de la relación. Puede ser uno o varios dependiendo del tipo de relación. La relación va del fuente(s) al destino.

contributor especializa a *source* como el componente fuente de una relación tipo *Contribution*. Restringe la cardinalidad a un único elemento.

solution especializa a *source* como la tarea fuente de una relación tipo *Operationalization*. Restringe el tipo de componente (a *Task*) y la cardinalidad a un único elemento.

target representa el elemento destino de la relación. Debe ser un único componente por relación.

correlated especializa a *target* como el *softgoal* destino de una relación tipo *Correlation*. Restringe el tipo del elemento a *Softgoal*.

source softgoal fuente de una relación tipo aspectual.

affects elemento *hard* destino de una relación tipo aspectual.

4.3.3. Restricciones

- Toda relación OM tiene al menos un elemento fuente (representado por la relación *source*).

- Toda relación de tipo *Contribution* tiene un y solo un elemento fuente llamado *contributor* (representado por la relación *Contribution*).
 - Toda relación de tipo *Operationalization* tiene un y solo un elemento fuente llamado *solution* de tipo *Task* (representado por la relación *Operationalization*).
- Toda relación OM tiene un y sólo un elemento destino (representado por la relación *target*).
 - Toda relación de tipo aspectual tiene un y solo un elemento fuente (representado por la relación *source*).
 - Toda relación de tipo aspectual tiene un y solo un elemento afectado (representado por la relación *affects*).
 - En toda relación de tipo *Decomposition* el elemento *target* de la relación debe ser del mismo tipo que el del elemento *source*, excepto si el *target* es de tipo *Goal* y los *source* son de tipo *Task*, que también está permitido.
 - Toda relación OM pertenece al modelo que posee el elemento destino (*target*) de la relación, sin importar si los elementos *target* y *source* pertenecen a distintos modelos.

4.3.4. Decisiones de Diseño

- Se permiten relaciones OM entre elementos de distintos modelos de interés. Esto provoca una menor independencia entre intereses, pero simplifica mucho el modelo, puesto que todas las relaciones OM pueden existir entre elementos del mismo interés (intra) o de distintos intereses (inter). Por tanto, su separación obligaría a duplicar todas las relaciones, además de requerir crear nuevas relaciones intra por cada inter.
- Los subtipos finales se han definido como meta-clases y no como tipos enumerados porque sus valores no van a cambiar (como ocurre con los valores de las metas y softgoals).
- Es necesario mantener la relación entre las relaciones OM y las aspectuales para que solo se generen una vez.
- En general, se ha preferido crear especializaciones de las asociaciones *source* y *target* para restringir los tipos de elementos y la cardinalidad puesto que es más visual y más fácil de entender. En los casos donde no ha sido posible si crear nuevas subclases (relación *decomposition*) se da una restricción OCL.

5. Reglas de Composición

En esta sección se describen los pasos necesarios para crear nuevas vistas a partir de modelos existentes (intereses o vistas ya creadas). La creación de vistas (o composición de modelos) es fundamental por dos razones: para obtener un modelo global del sistema, puesto que los intereses permiten centrarse en un problema concreto, pero no dan la visión global; y para dar visiones particulares a partir de varios modelos, que permitan decidir entre una alternativa u otra.

En nuestro caso, la creación de una vista se realiza a partir de uno o dos modelos. Una vista sobre un único modelo tiene sentido si queremos modificar un modelo de interés antes de componerlo con otros (por ejemplo para seleccionar una alternativa). Como se ha explicado, las vistas contienen referencias a los elementos y relaciones de los intereses, para poder modificarlos sin alterar dichos intereses, pero además pueden tener nuevas relaciones debido a las relaciones aspectuales.

Los pasos necesarios para crear una nueva vista se resumen en cinco:

1. Creación de la vista: se crea una instancia de la meta-clase *View* y enlaces a los modelos que se están componiendo (asociación *sourceModel*), para mantener la trazabilidad de la vista.
2. Creación de las referencias a elementos: se crean instancias de la meta-clase *ElementRef* para cada elemento en los modelos origen, que referencian a los elementos originales (asociación *ReferredElem*). Hay que señalar que se referencia siempre a los elementos, aunque la referencia se cree a partir de otra referencia, como ocurre cuando componemos una vista.
3. Creación de relaciones a partir de relaciones aspectuales: si existe una relación aspectual entre un *softgoal* de un modelo (fuente) y un elemento funcional del otro (destino), si no existe una descomposición OR para el elemento destino se crea (nueva instancia de la meta-clase *OR*) y en ambos casos, se incluyen las operacionalizaciones del *softgoal* (elemento fuente) dentro de la descomposición OR (nuevas asociaciones *source* a la descomposición OR).
4. Creación de relaciones OR: si los dos modelos tienen descomposiciones OR diferentes (que se pueden dar debido a las relaciones aspectuales), se crea una nueva descomposición OR (instancia de la meta-clase *OR*) con los elementos de las dos relaciones para tener una única relación OR. En el caso de que ya se haya creado en el paso anterior, simplemente se añaden los elementos que no estuvieran. En ambos casos supone la creación de nuevas asociaciones *source* a la descomposición OR.
5. Creación de referencias a relaciones: se crean instancias de la meta-clase *RelationshipRef* para cada relación de los modelos que se están componiendo, excepto para las relaciones OR que ya se hayan creado en los pasos anteriores. Para cada referencia creada, se crea una asociación (*ReferredRel*) a la relación original. Al igual

que con las referencias a elementos, siempre se referencia a relaciones, nunca a referencias.

A la hora de definir los pasos anteriores, no hemos encontrado trabajos en la literatura que especifiquen una manera de hacerlo. En general, se definen de forma textual o mediante técnicas de transformación de modelos. El problema de estas técnicas es que tienen como entrada un único modelo y no dos modelos como en la composición y no existe un estándar para su representación, aunque existe una propuesta del OMG (QVT [17]) bastante compleja y poco utilizada. En nuestro caso, puesto que los modelos están definidos dentro del meta-modelo es posible representar cada paso como una (o varias) transformaciones de modelos cuyo origen y destino son instancias del mismo meta-modelo. El modelo resultante será igual al modelo inicial con la nueva vista.

En las siguientes sub-secciones se presentan las reglas para cada paso. Es importante que las reglas se deben ejecutar siguiendo el orden de cada paso, para no crear relaciones OR duplicadas.

5.1. Reglas para la Creación de la Vista

Estas reglas permiten crear una nueva vista a partir de uno o dos modelos (reglas 1.1 y 1.2 respectivamente).

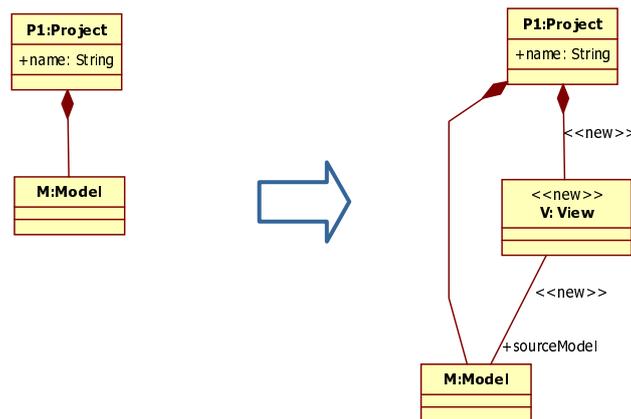


Figura 8: Regla 1.1: Creación de una vista a partir de un modelo.

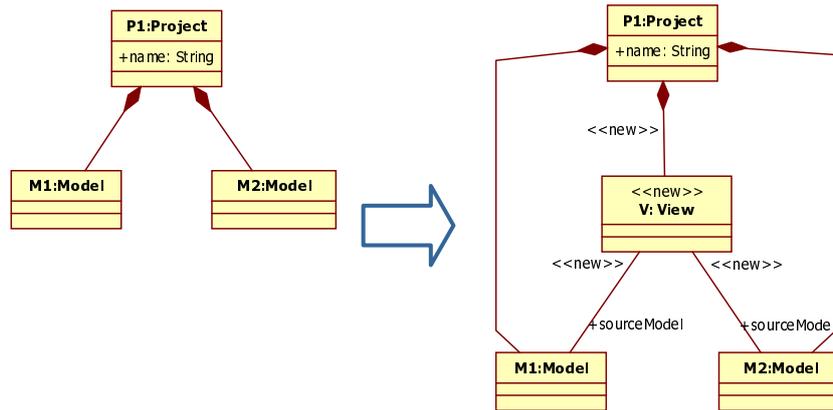
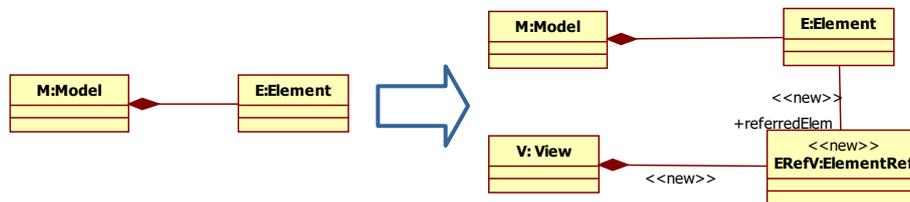


Figura 9: Regla 1.2: Creación de una vista a partir de dos modelos.

5.2. Reglas para la Creación de Referencias a Elementos

En este caso, las Reglas 2.1 y 2.2 crean las referencias a elementos dentro de las vistas. La diferencia entre una y otra radica en que la primera crea las referencias a partir de elementos (el modelo es un interés), y la segunda las crea a partir de referencias a elementos (el modelo origen es una vista). En el segundo caso, la referencia se crea sobre el elemento y no sobre la referencia como se comentó en la sección anterior. La restricción asegura que no se creen varias referencias para un mismo elemento.



$$\neg \exists \text{refE} : \text{ElementRef} / \text{refE.owner} = \text{V} \wedge \text{refE.referredElem} = \text{E}$$

Figura 10: Regla 2.1: Creación de referencias a elementos a partir de elementos.

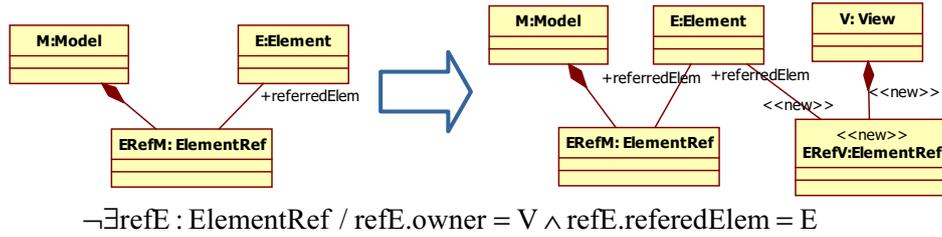


Figura 11: Regla 2.2: Creación de referencias a elementos a partir de referencias a elementos.

5.3. Reglas para la Creación de Relaciones a partir de Relaciones Aspectuales

Las relaciones aspectuales causan que las operacionalizaciones del softgoal (elemento fuente o *source*) se añadan como descomposiciones alternativas al elemento funcional (elemento destino o *target*). En este caso son necesarias dos tipos de reglas: a) las que crean la relación OR si no existía ninguna que descomponga al elemento funcional (E_Target) de la relación aspectual. Estas reglas crean la descomposición utilizando una de las operacionalizaciones como *source* (Reglas 3.1 a 3.4); y b) las que crean nuevas asociaciones para la descomposición OR de las operacionalizaciones (asociaciones *source*), creando el resto de las asociaciones para cada operacionalización (Reglas 3.5 a 3.8).

La restricción de las reglas del primer tipo asegura que no se crea una nueva descomposición OR si ya existe una en la vista, mientras que la del segundo tipo asegura que no se vuelve a añadir un elemento a una descomposición donde ya esté (ya sea un *source* de la descomposición OR).

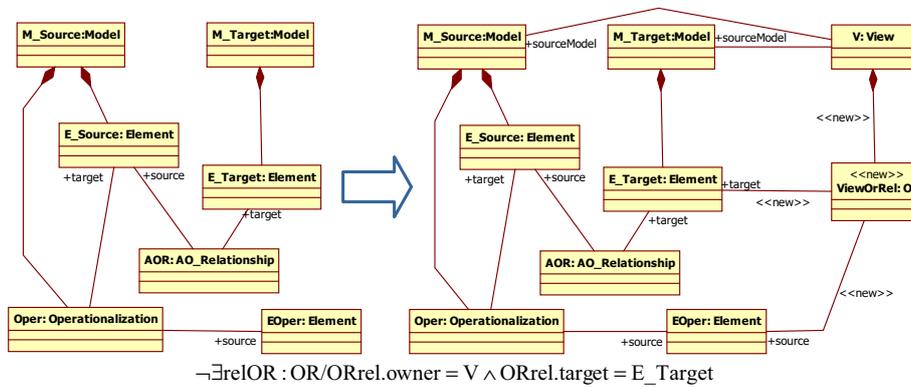


Figura 12: Regla 3.1: creación de una relación OR debido a una relación aspectual entre dos intereses.

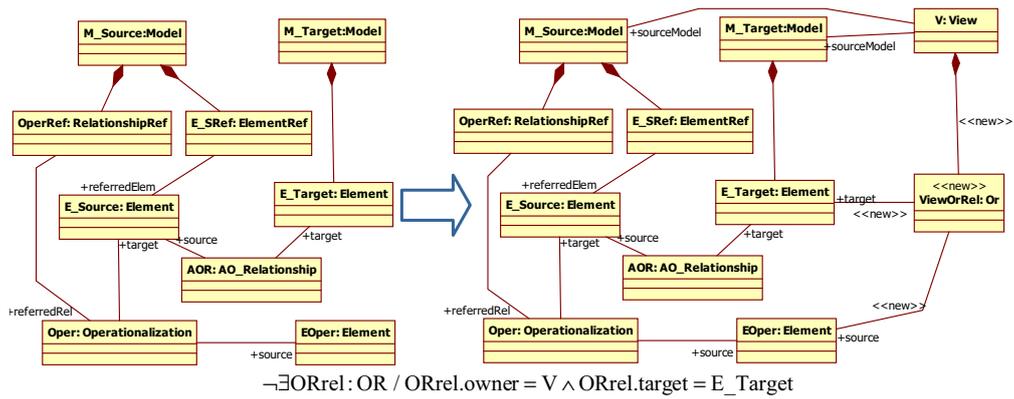


Figura 13: Regla 3.2: creación de una relación OR debido a una relación aspectual donde el fuente es una referencia y el destino un elemento.

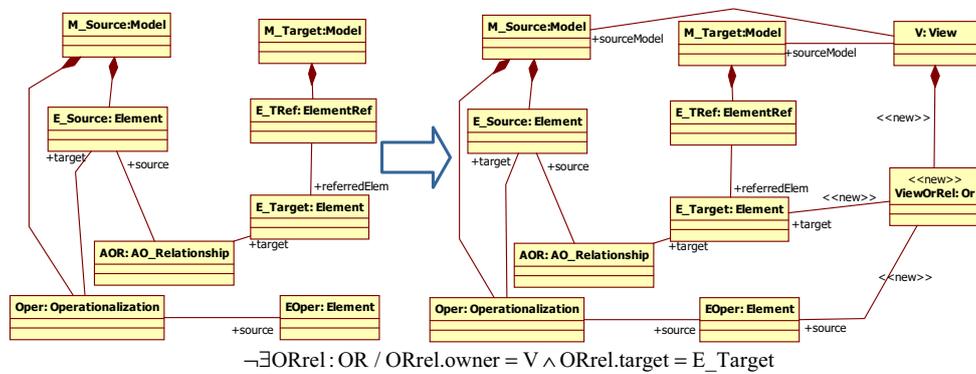


Figura 14: Regla 3.3: creación de una relación OR debido a una relación aspectual donde el fuente es un elemento y el destino una referencia.

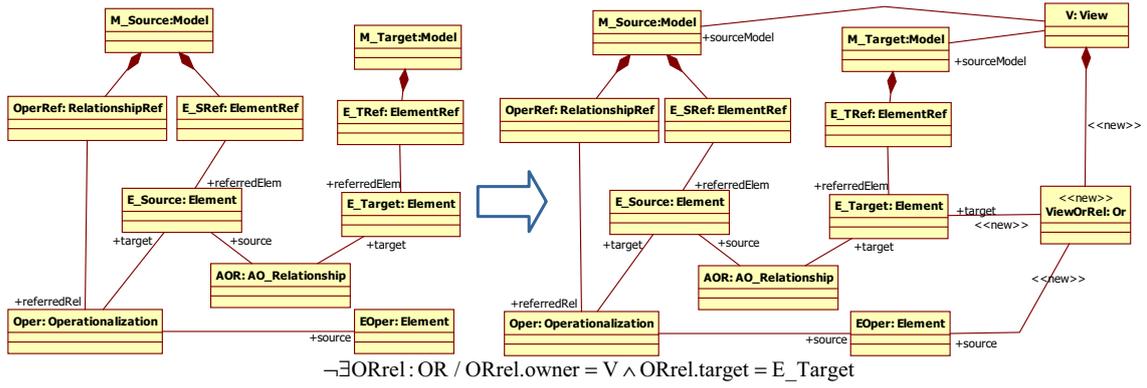


Figura 15: Regla 3.4: creación de una relación OR debido a una relación aspectual entre dos modelos cuyos fuentes y destino están representados por referencias.

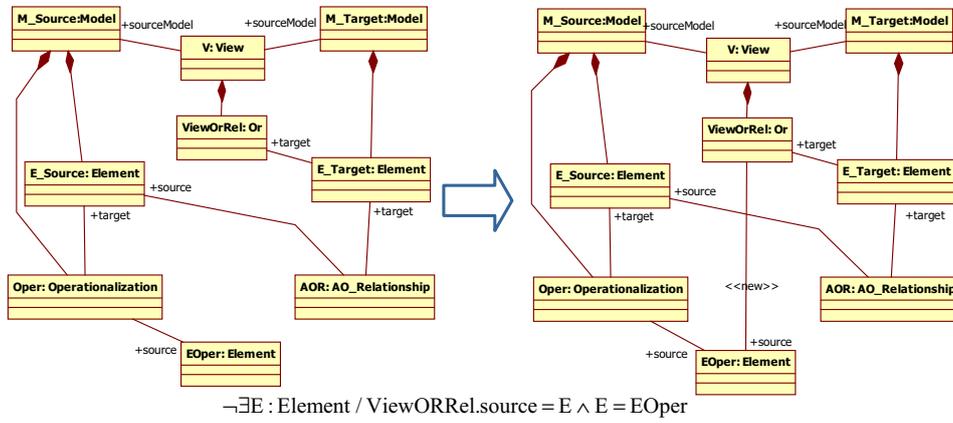


Figura 16: Regla 3.5: adición de nuevos fuentes a una relación OR debido a una relación aspectual entre dos intereses.

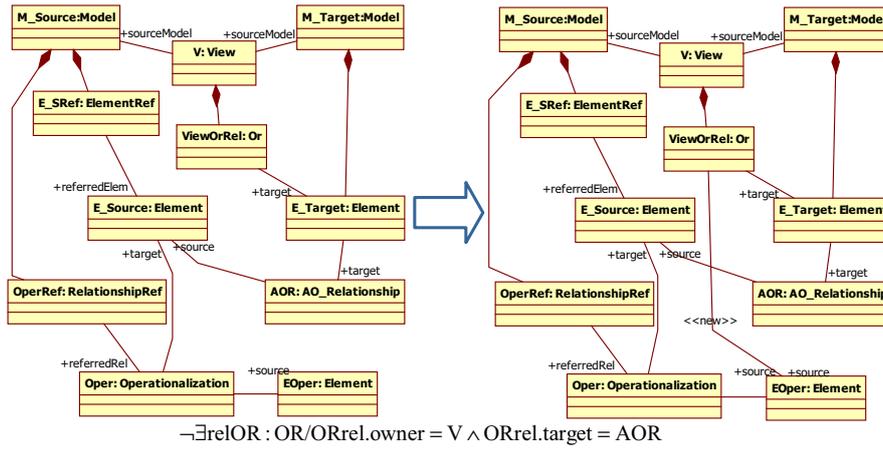


Figura 17: Regla 3.6: adición de nuevos fuentes a una relación OR debido a una relación aspectual en la cual el fuente es una referencia y el destino es un elemento.

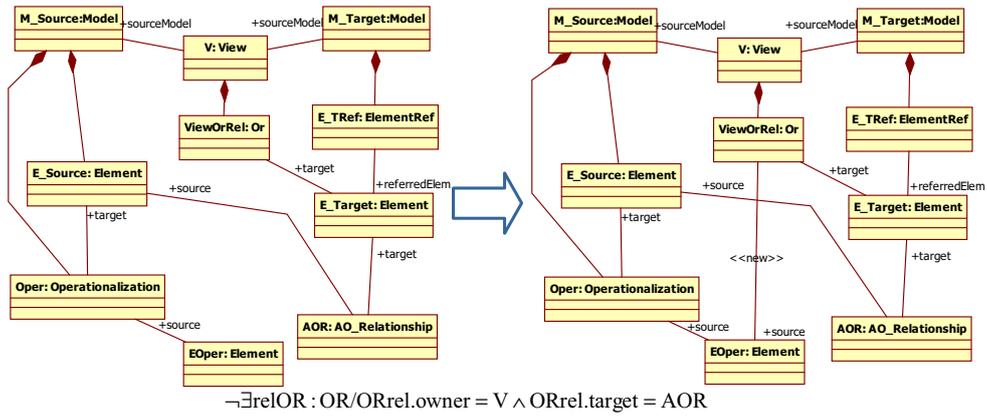


Figura 18: Regla 3.7: adición de nuevos fuentes a una relación OR debido a una relación aspectual en la cual el fuente es un elemento y el destino una referencia.

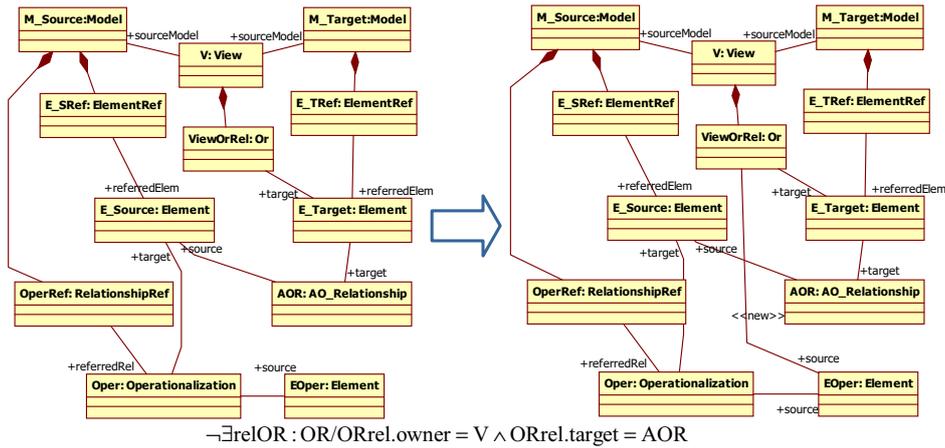


Figura 19: Regla 3.8: adición de nuevos fuentes a una relación OR debido a una relación aspectual entre dos modelos cuyos fuentes y destino están representados por referencias.

5.4. Reglas para las Relaciones OR

La creación de nuevas relaciones OR debido a las relaciones aspectuales, puede provocar que los dos modelos a componer tengan dos descomposiciones OR diferentes (o referencias a dichas relaciones) para el mismo elemento. Para que solamente exista como mucho una descomposición de cada tipo hay que comprobar si sucede esto. De esta forma las cuatro primeras reglas (de la 4.1 a la 4.4) de este paso buscan dos relaciones OR (o referencias) en los modelos, y si ocurre, crean una nueva descomposición OR (instancia de la meta-clase *OR*) a partir de uno de los elementos *source* de una de las descomposiciones. Las otras dos reglas (4.5 y 4.6) añaden el resto de elementos. Hay que señalar que si ya existía la descomposición OR porque ya se ha creado en el paso anterior, no se crea una nueva (restricción de las primeras reglas), sino que se aplican directamente las reglas 4.5 y 4.6 para añadir los nuevos elementos que descomponen a la descomposición ya existente.

En cuanto a las restricciones, como ya se ha comentado, la de las primeras sirve para no crear una nueva descomposición si ya existía una relación OR para el elemento. En la restricción de las segundas reglas, se comprueba que el elemento a añadir en la descomposición (*E_Source*) no esté ya en dicha descomposición.

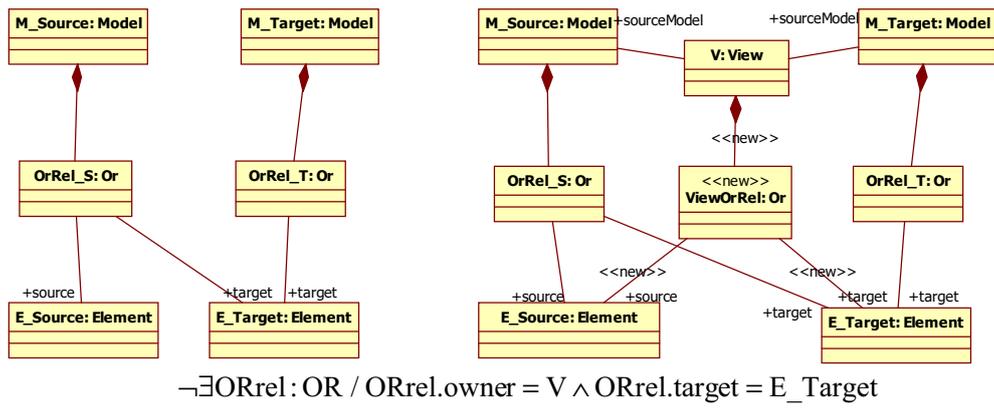


Figura 20: Regla 4.1: Creación de nueva relación OR a partir de dos relaciones OR.

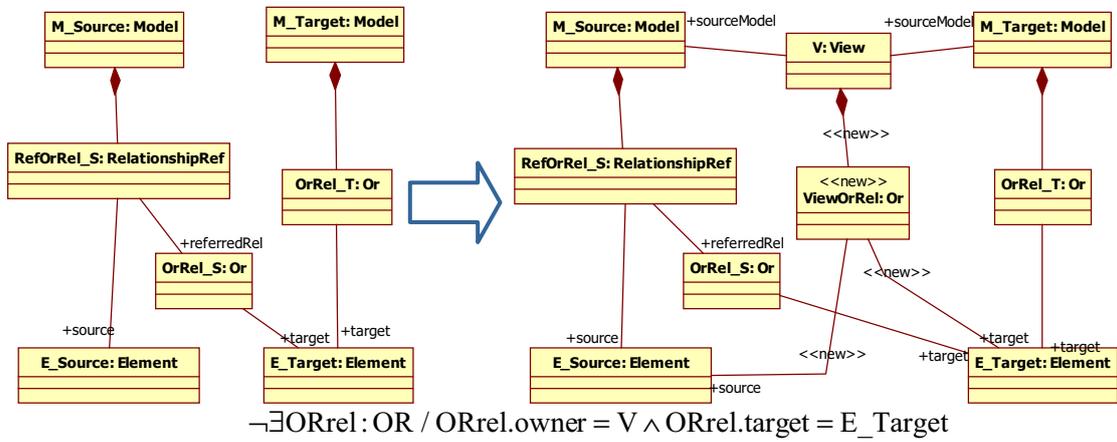


Figura 21: Regla 4.2: Creación de nueva relación OR donde el elemento del modelo fuente es una referencia y el del destino no.

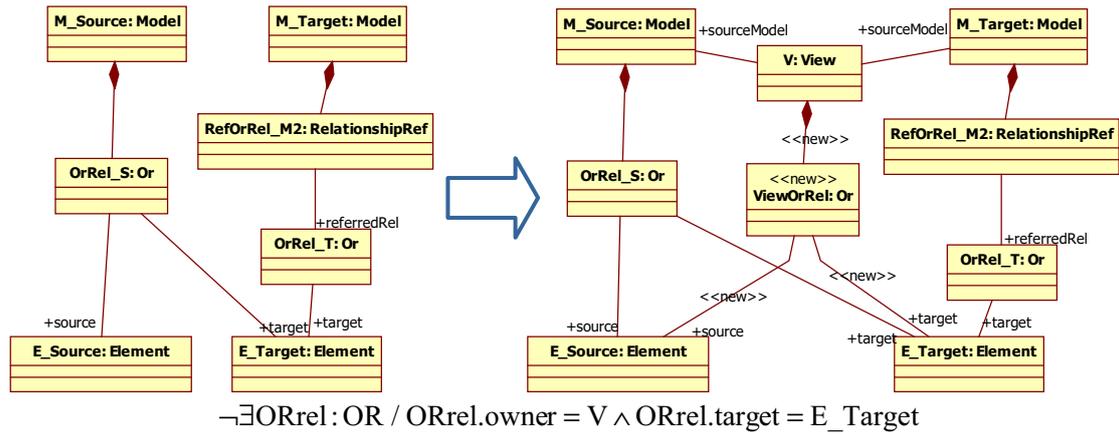


Figura 22: Regla 4.3: Creación de nueva relación OR donde el elemento del modelo destino es una referencia y el del fuente no.

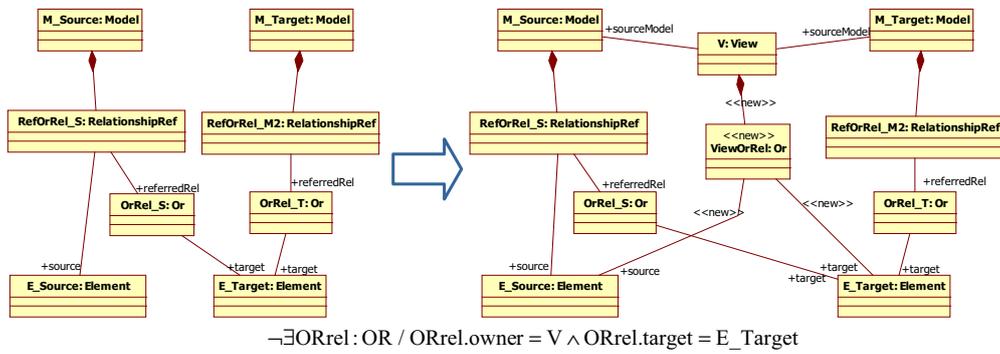


Figura 23: Regla 4.4: Creación de nueva relación OR donde tanto el elemento del modelo fuente como el del destino son referencias.

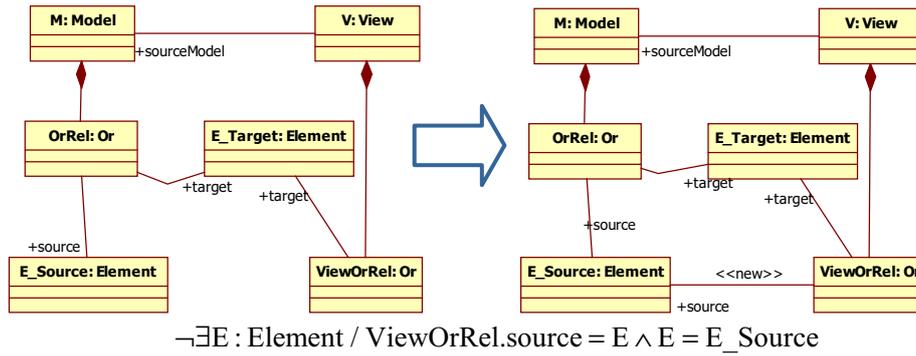


Figura 24: Regla 4.5: Adición de los elementos fuente de una relación OR si existe una relación OR en la vista con el mismo target (E_Target).

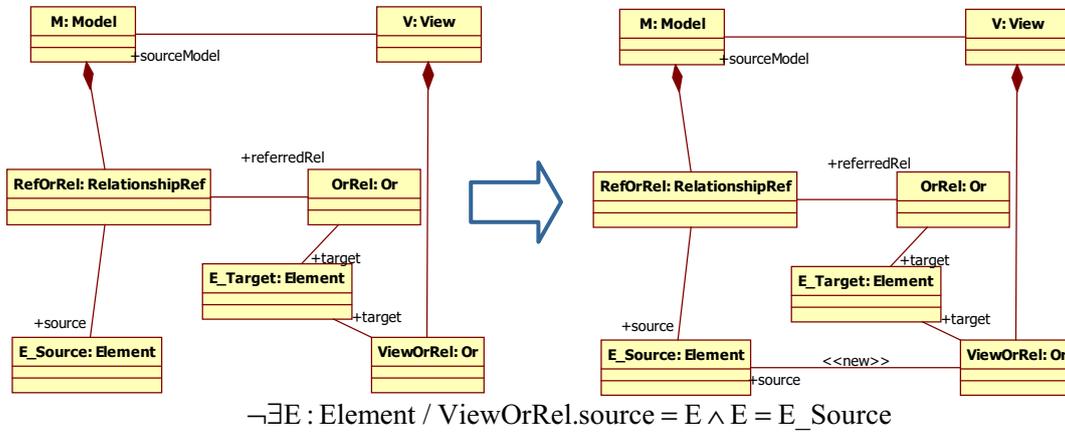


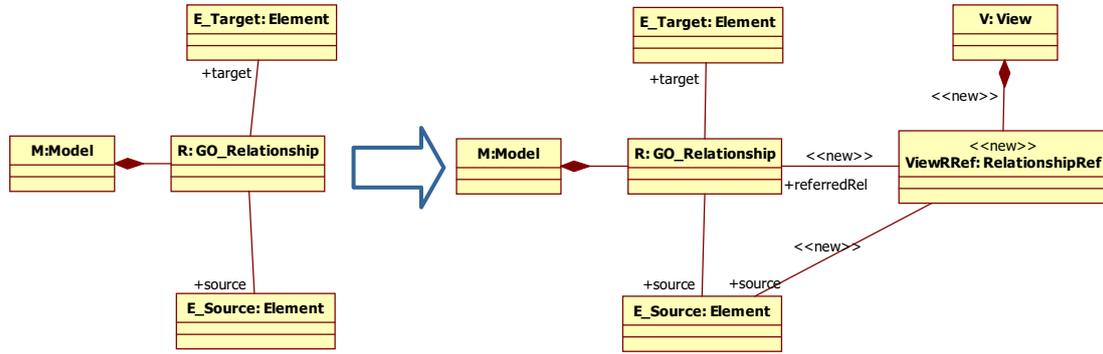
Figura 25: Regla 4.6: Adición de los elementos fuente de una relación OR desde una referencia a relación, si existe una relación OR en la vista con el mismo target (E_Target).

5.5. Reglas para la Creación de Referencias a Relaciones

Por último, se crean las referencias a las relaciones. Como en las reglas del paso 2, las referencias se hacen siempre a los elementos originales, no a otras referencias. Existen dos tipos de relaciones como en las reglas anteriores, unas (reglas 5.1 y 5.2) para crear las referencias (instancias de la meta-clase *RelationshipRef*), y otras para añadir nuevos elementos (reglas 5.3 y 5.4), en este caso *sources* a la referencia.

En cuanto a las restricción, debemos comprobar que no existe ya una referencia a la relación en la vista o al elemento en la referencia (primera restricción), y si la relación a referenciar es un OR, que no se haya creado una descomposición OR en la vista con el mismo target (segunda restricción). En ese supuesto, la relación ya se habría integrado

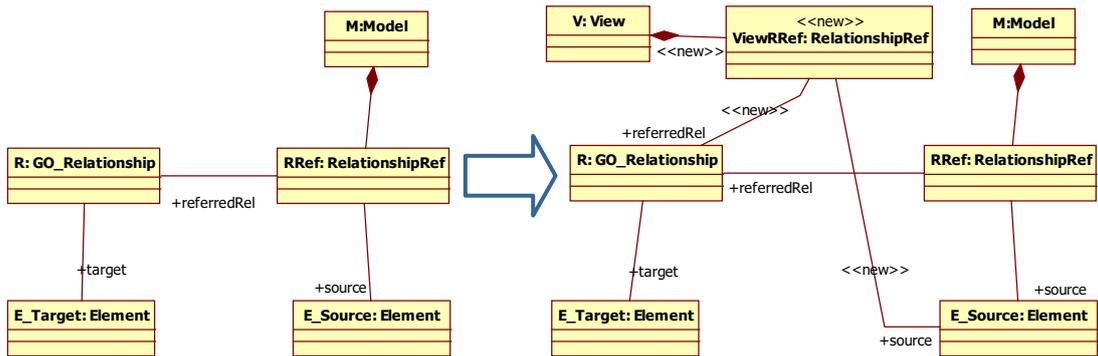
en la descomposición en el paso anterior, por lo que no sería necesario crear una nueva relación.



$$\neg \exists \text{refR} : \text{RelationshipRef} / \text{refR.owner} = \text{V} \wedge \text{refR.referredRel} = \text{R}$$

$$\neg (\text{R.type} = \text{OR}) \vee (\neg \exists \text{Rel} : \text{Relationship} / \text{Rel.owner} = \text{V} \wedge \text{Rel.target} = \text{E_Target})$$

Figura 26: Regla 5.1: Creación de una nueva referencia a relación a partir de una relación en el modelo origen.



$$\neg \exists \text{refR} : \text{RelationshipRef} / \text{refR.owner} = \text{V} \wedge \text{refR.referredRel} = \text{R}$$

$$\neg (\text{R.type} = \text{OR}) \vee (\neg \exists \text{Rel} : \text{Relationship} / \text{Rel.owner} = \text{V} \wedge \text{Rel.target} = \text{E_Target})$$

Figura 27: Regla 5.2: Creación de una nueva referencia a relación a partir de una referencia a relación en el modelo origen.

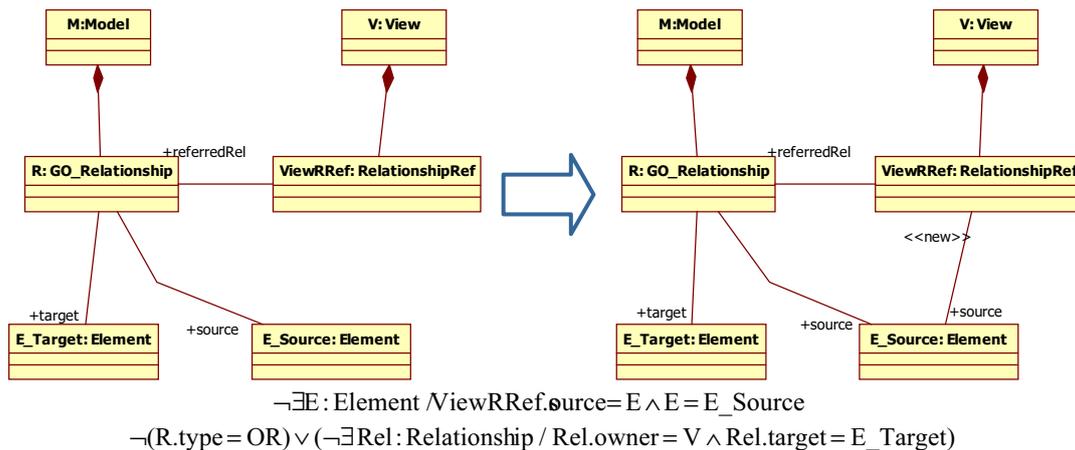


Figura 28: Regla 5.3: Creación de nuevos fuentes a una referencia a relación que ya ha sido creada en la vista por una relación en el modelo origen.

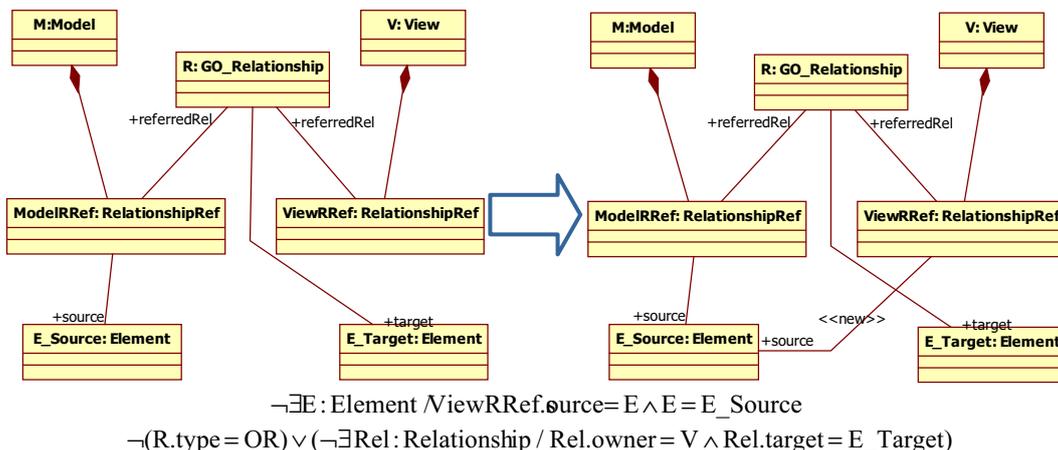


Figura 29: Regla 5.4: Creación de nuevos fuentes a una referencia a relación que ya ha sido creada en la vista por una referencia a relación en el modelo origen..

6. Ejemplo

En esta sección se utiliza el meta-modelo y las reglas propuestas en el modelado de un comunicador alternativo y aumentativo. La elección de este dominio de aplicación se debe a que tiene un alto grado de variabilidad debido a las distintas discapacidades sobre las que se puede actuar, y a que contamos con la colaboración del Centro de Educación Especial Obregón [18], que actúa como experto del dominio.

En la Figura 30 se muestra parte del modelo para el dominio. Cada elemento y relación

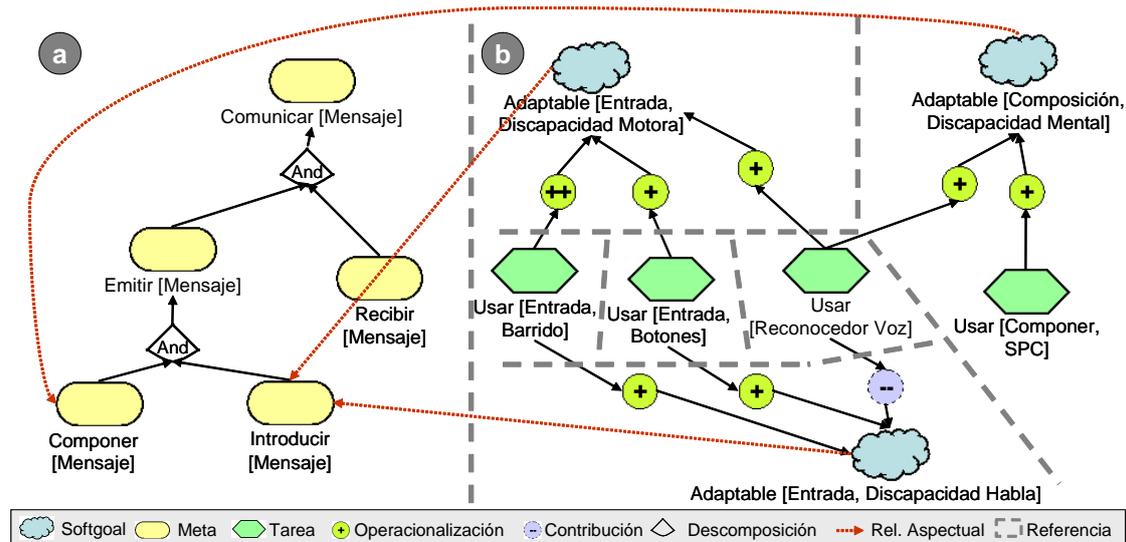


Figura 30: Ejemplos de modelos de interés (separados por líneas). La Figura a) describe la parte funcional y la b) algunos RNF y sus soluciones.

son instancias de las meta-classes definidas en la Sección 4, así como los modelos, que se muestran separados por líneas discontinuas. En la primera parte de la figura se describe la parte funcional que, en este ejemplo simplificado, está formada por la meta principal (Comunicar[Mensaje]), que a su vez se descompone en dos sub-metas: emitir y recibir el mensaje. Por último, se ha separado la emisión en otras dos sub-metas, componer e introducir. En la Figura 30b se muestran parte de los modelos de softgoals que definen la adaptabilidad a diferentes discapacidades y sobre distintos subsistemas. Estos modelos son sub-modelos del catálogo de RNF Adaptable[Sistema], no mostrado en la imagen, que contiene varios softgoals como Adaptable[Entrada], Adaptable[Composición], Adaptable[Visualización] de los cuales se han utilizado en nuestro dominio solo los dos primeros. Un detalle importante es que las distintas tareas, al ser soluciones de varios modelos a la vez, deben ser raíces de sus propios modelos (sólo se permiten relaciones de jerarquía entre elementos de distintos modelos si el fuente es una raíz). También se muestran como afectan estos softgoals a los elementos funcionales (en este caso metas) mediante relaciones aspectuales. Por ejemplo, Adaptable [Entrada, Discapacidad Motora] afecta a Introducir [Mensaje] y cuando se compongan ambos modelos, las soluciones del softgoal se añadirán a la meta.

La composición de modelos se puede hacer sobre modelos con relaciones aspectuales o no, por ejemplo se podrían componer los intereses de la Figura 30b obteniéndose una vista similar al resultado, pero eliminando las líneas discontinuas, y con referencias en vez de los elementos y las relaciones como se puede ver en la Figura 31. Esta vista nos permite analizar como afectan las distintas operacionalizaciones al conjunto de los RNF, puesto que inicialmente están separados en distintos modelos.

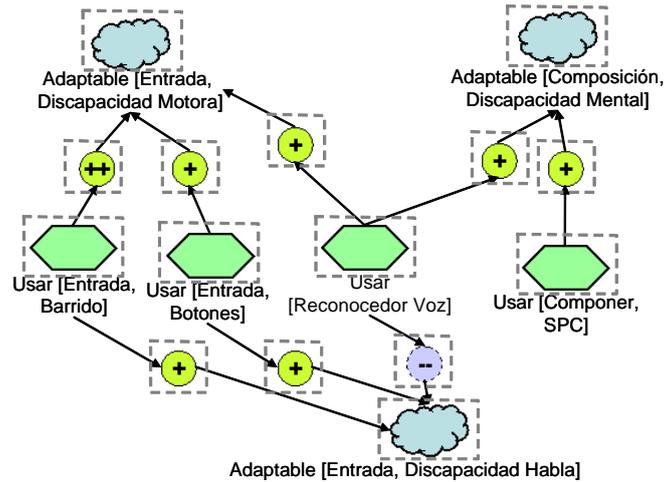


Figura 31: Ejemplo de vista que muestra los softgoals y como afectan las operacionalizaciones a cada una de ellas.

Otra opción es crear una vista para cada softgoal con todas sus operacionalizaciones, como se muestra en la Figura 32. Es importante señalar que la contribución entre Usar [Reconocedor Voz] y Adaptable [Entrada. Discapacidad Habla] no aparece puesto que no es una operacionalización y en la vista de Adaptable [Entrada. Discapacidad Habla] (Figura 32b) no está Usar [Reconocedor Voz]. También se podría analizar como interactúan varios softgoals, en ese caso se compondrían vistas del estilo a las mostradas en la Figura 32, es decir con sus operacionalizaciones.

Si hay relaciones aspectuales entre los modelos, se aplica el paso 3. La Figura 33 muestra el resultado de la componer la parte funcional de la Figura 30 con la vista resultado de componer el resto de intereses. En los primeros pasos, se crean la vista y las referencias y en el tercer paso se crean nuevas relaciones (en este caso las dos relaciones Or) para dar una visión global del sistema.

En este caso no es necesario utilizar el paso 4 (relativo a elementos que se descomponen en OR difentes), pero si la descomposición se hubiera hecho a partir del modelo funcional y Adaptable[Entrada, Discapacidad Motora] por un lado y del funcional y Adaptable[Entrada, Discapacidad Habla] (el resultado se muestra en la Figura) por el otro y luego compuestas ambas vistas, si que sería necesario.

Para representar los ejemplos anteriores, hemos utilizado GME (Generic Modeling Environment) [4], una herramienta de meta-modelado desarrollada en la Universidad de Vanderbilt. Esta herramienta permite obtener rápidamente un entorno de modelado a partir de un meta-modelo, y definir restricciones en OCL sobre dicho meta-modelo que deben cumplir los modelos. Los modelos desarrollados con esta herramienta nos han permitido observar como se comporta el meta-modelo en la creación de modelos de intereses y en la generación de vistas, puesto que permite exportar a formato XML [26].

En la Figura 35 se muestra un pantalla de la herramienta donde se define el meta-

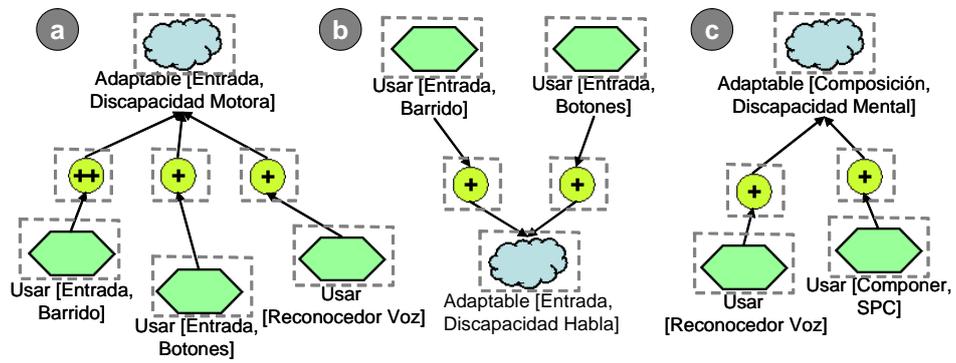


Figura 32: Ejemplos de vistas que muestran las operacionalizaciones para cada softgoal mostrada en la figura 30b.

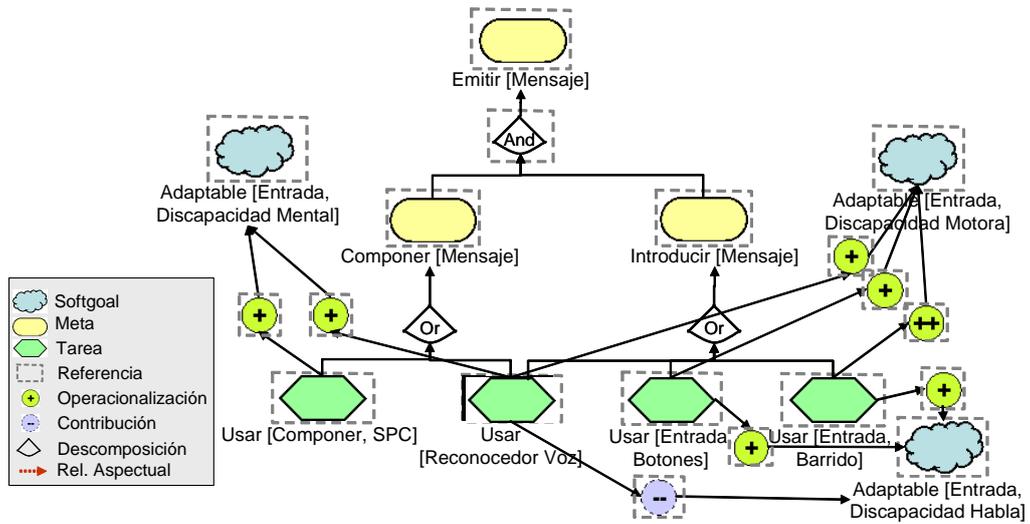


Figura 33: Ejemplo de vista, resultado de componer los modelos de la Figura 30.

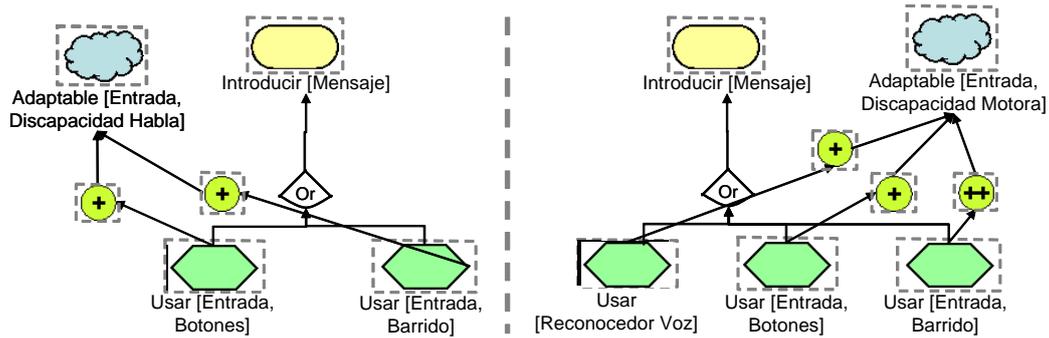


Figura 34: Posibles vistas resultado de componer la parte funcional con softgoals diferentes.

modelo, mientras que en la Figura 36 se muestra un ejemplo del modelado de un sistema utilizando nuestro meta-modelo.

7. Conclusiones

En este documento se ha presentado una propuesta para el soporte del modelado de la variabilidad de requisitos a partir de modelos de metas, donde se aplica un enfoque orientado a aspectos. Para ello se ha definido un meta-modelo basado en modelos de interés formados por elementos de la orientación a metas y vistas formadas por referencias a dichos elementos. Además, se han descrito las reglas para componer dichos modelos y generar nuevas vistas de forma automática.

Aunque el objetivo principal de la propuesta es constituir un marco de trabajo para el análisis de variabilidad, también se puede aplicar al modelado de requisitos orientado a metas en general, proporcionando un mecanismo para relacionar la parte funcional y no funcional. Otra ventaja es que permite una mejor estructuración y un análisis más preciso, separando el modelado en distintos intereses, que pueden ser compuestos de forma automática. Esta mejor estructuración permite también una mayor reutilización, de forma que intereses lo suficientemente genéricos pueden utilizarse en varios proyectos, por ejemplo utilizando catálogos de RNF como los propuestos en el NFR Framework [3].

Las vistas son una mejora substancial para el análisis de la variabilidad, puesto que permiten eliminar elementos, por ejemplo para seleccionar una variante de un modelo, antes de componer con otros disminuyendo el número de variantes a analizar y, por tanto, obteniéndose una mayor escalabilidad. Otro mecanismo para disminuir el número de variables y mejorar la escalabilidad del análisis se basa en componer sólo ciertos modelos, según los requisitos del usuario final.

En cuanto al soporte de herramientas, se ha utilizado GME para obtener una primera herramienta de modelado, que además, permite exportar los modelos creados a XML [26], aunque siguiendo su propio meta-modelo.

Por otra parte, las composiciones se han implementado mediante hojas de estilo XML

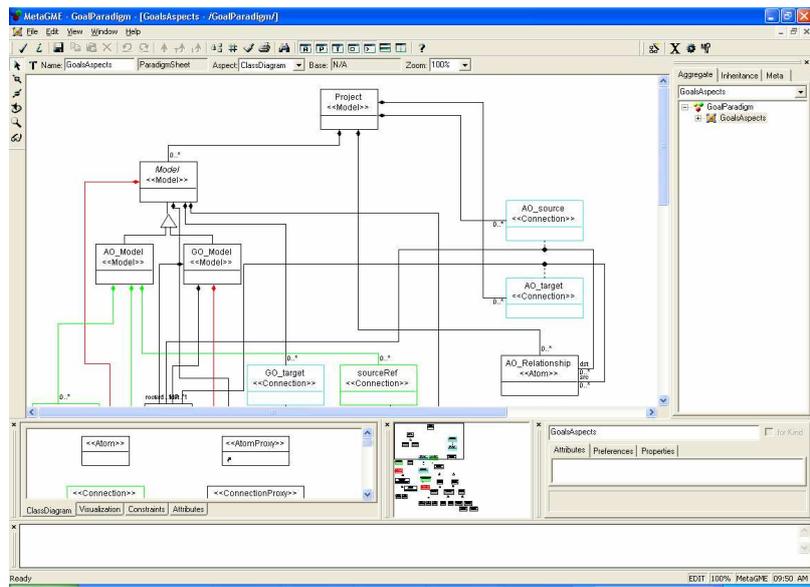


Figura 35: Pantalla de la herramienta GME donde se define nuestro meta-modelo.

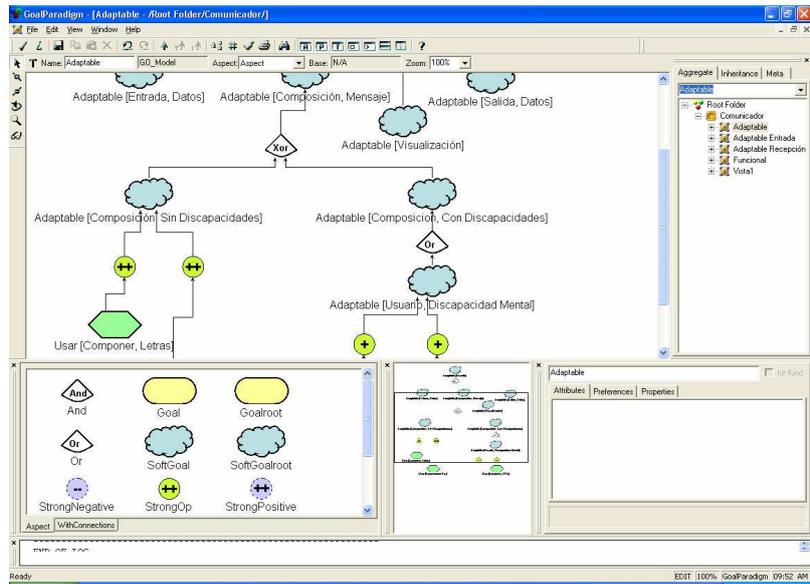


Figura 36: Pantalla de la herramienta GME donde se utiliza nuestro meta-modelo para describir un sistema con variabilidad.

(XSLT) [27]. La hoja de estilo recibe el proyecto en XML y el nombre de los dos modelos a componer y el de la nueva vista como parámetro, creando una nueva vista en el mismo proyecto como resultado de la composición. También se ha desarrollado un conversor del fichero XML obtenido por la herramienta al formato propio de nuestro meta-modelo.

En cuanto a trabajo futuro, se pretende definir técnicas que mejoren el análisis de la variabilidad analizando diferentes RNF y priorizando unos sobre otros. Estas técnicas se basarán en el meta-modelo definido y en las posibilidades que brindan las vistas generadas.

Asimismo, hay que adaptar el trabajo desarrollado para el análisis de la configuración [7] al nuevo meta-modelo y al enfoque orientado a aspectos. La forma más simple de obtener esta adaptación es componiendo todos los modelos y separando la parte funcional y no funcional en modelos diferentes y creando una tabla con las interrelaciones entre ellos. En cualquier caso, queremos explorar nuevas posibilidades como la aplicación de la técnica descrita en el artículo a los modelos antes de componerlos, o componer algunos modelos en función de las prioridades, ...

Por último, a pesar de que el prototipo desarrollado con GME nos ha permitido utilizar el meta-modelo con ejemplos simples, nuestro objetivo es la implementación de una herramienta completa de modelado. Esta herramienta incluirá la aplicación transparente e integrada de las composiciones definidas, y además, dará soporte a las técnicas de análisis y configuración de la variabilidad.

Tabla Resumen

Elemento [Meta-clase]	Descripción
Proyecto [Project]	Proyecto de variabilidad orientado a metas. Es una agrupación de modelos.
Modelo [Model]	Modelo del proyecto.
Modelo de Interés [Concern]	Una parte de interés del sistema a analizar. Es una estructura jerárquica de Elementos con un nodo raíz.
Vista [View]	Elemento [Meta-clase] Descripción
Elemento [Element]	Elementos del Modelo de Interés. Se describen por un tema (Subject) y cero o más tópicos (Topic).
Meta [Goal]	Objetivo del sistema. Se descomponen en sub-metas hasta llegar a las tareas. Valores: alcanzada / denegada.
Tarea [Task]	Forma de alcanzar una meta / satisfacer un softgoal. Puede ser un algoritmo, proceso, restricción, decisión de diseño Su presencia o no en una variante determina los valores de metas y softgoals.
Softgoal [Softgoal]	Meta que no tiene un criterio claro y definido de satisfacción. Valores desde claramente satisfecho a claramente insatisfecho.

Elemento [Meta-clase]	Descripción
Descriptor [DescriptionElement]	Unidad de descripción de un elemento. Se organizan en jerarquías de herencia que pueden utilizarse para sugerir relaciones aspectuales.
Relación OM [GO_Relationship]	Relaciones propias de los modelos orientados a metas.
Descomposición [Decomposition]	Relación de jerarquía entre elementos de mismo tipo o de tarea a meta. Subtipos: And, Or, Xor.
Operacionalización [Operationalization]	Indica una posible solución a un softgoal (representada como una tarea). Subtipos: Strong, Weak.
Contribución [Contribution]	Indica como contribuye un elemento a un softgoal. Subtipos: Strong Positive, Weak Positive, Weak Negative, Strong Negative
Relación Aspectual [AO_Relationship]	Indica que un softgoal puede afectar a un elemento funcional, modificando su comportamiento.
Restricción [Constraint]	Relaciona una o más tareas. Subtipos: Requires (1 tarea necesita otras tareas) y MutuallyExclusive (las tareas no pueden estar a la vez)
Referencia a Elemento [Element_Ref]	Referencia a un elemento.
Referencia a Relación [Relationship_Ref]	Referencia a una relación.

Referencias

- [1] Bosch, J.: Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach. Addison-Wesley (2000)
- [2] Brito, I., & Moreira, A. Integrating the NFR Framework in a RE Model. Early Aspects 2004: Aspect-Oriented Requirements Engineering and Architecture Design, Lancaster, UK. 2004
- [3] Chung, L., Nixon, B., Yu, E. and Mylopoulos, J. Non-Functional Requirements in Software Engineering Kluwer Academic Publishers 2000.
- [4] Generic Modeling Environment (GME 5) Universidad de Vanderbilt (Nashville). Disponible en <http://www.isis.vanderbilt.edu/projects/gme/> (Última visita en mayo de 2006).
- [5] Giorgini, P., Mylopoulos, J., Nicchiarelli, E., and Sebastián, R. Reasoning with goal models 21st Int. Conf. Conceptual Modeling (ER 2002), Tampere, Finland, Oct. 2002.
- [6] González-Baixaui, B., Laguna M., Leite, J.C.S.P. Aplicación de un Enfoque Intencional al Análisis de Variabilidad. Anais do WER05, Porto, Portugal, Junho, 2005. ISBN 972-752-079-0, pp:100-111

- [7] González-Baixauli, B., Leite J.C.S.P., and Mylopoulos, J. Visual Variability Analysis with Goal Models. Proc. of the RE2004. Sept. 2004. Kyoto, Japan. IEEE Computer Society, 2004. pp: 198-207.
- [8] Griss, M., Favaro, J., and d' Alessandro, M. Integrating feature modeling with the RSEB. Fifth International Conference on Software Reuse (ICSR), pages 76–85. IEEE Computer Society Press, 1998.
- [9] G.R.L.; Goal-oriented Requirement Language. University of Toronto, Canada. Disponible en <http://www.cs.toronto.edu/km/GRL/> (última visita en mayo de 2006).
- [10] Halmans, G., and Pohl, K., Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling 2, 1 2003, 15–36.
- [11] Hui, B., Liaskos, S., Mylopoulos, J., Requirements Analysis for Customizable Software: a Goals - Skills - Preferences Framework, RE'03, Monterey Bay, USA, Sep. 2003, pp.117-126
- [12] Jacobson I., Griss M., and Jonsson P.: Software Reuse. Architecture, Process and Organization for Business Success. ACM Press. Addison Wesley Longman (1997)
- [13] Kang, K. C., Cohen, S., Hess, J., Nowak, W. and Peterson, S. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, SEI (Carnegie Mellon), Pittsburgh,
- [14] Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C, Loingtier, J., and Irwin, J.: Aspect-Oriented Programming. In: Object-Oriented Programming 11th European Conference (ECOOP 97), LNCS 1241, Springer-Verlag, 1997. p. 220-242.
- [15] Liaskos, S., Lapouchnian, A., Wang, Y. , Yu, Y., and Easterbrook, S.: Configuring Common Personal Software: a Requirements-Driven Approach. International Conference on Requirements Engineering (RE'05). pp. 9-18, Paris, France. 2005
- [16] Navarro, E., Letelier, P., & Ramos, I.. Goals and Quality Characteristics: Separating Concerns. Early Aspects Workshop @ OOPSLA 04, 2004, Vancouver, Canada.
- [17] Object-Management-Group, QVT-Merge-Group: MOF 2.0 Query/View/Transformation Specification version 2.0 Final Adopted Specification. Object Management Group doc. ptc/05-11-01. (2005)
- [18] Obregón, Centro de Educación Especial. Página Web: <http://www.asprona-valladolid.es/obregon/> (última visita en mayo de 2006).
- [19] OpenOME tool project web page. <http://www.cs.toronto.edu/~yijun/OpenOME.html/>.
- [20] Parnas, D.L. "On the Design and Development of Program Families," IEEE Trans. on Soft. Eng. 2(1), Mar. 1976, pp: 1-9

- [21] Rashid, A., Sawyer, P., Moreira A., and Araujo J.: "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", Proceedings of IEEE Joint International Conference on Requirements Engineering (RE), 2002, IEEE Computer Society, pp. 199-202.
- [22] Sousa, G., & Castro, J.. Improving the Separation of Non-Functional Concerns in Requirements Artifacts. 12th IEEE International Requirements Engineering Conference (RE 04), Japan, 2004.
- [23] Svahnberg, M, Bosch, J. "Issues Concerning Variability in Software Product Lines", LNCS 1951, Jan 2000, pp:146-157
- [24] van Gorp, J., Bosch J. and Svahnberg, M. On the notion of variability in software product lines, Working Int. Conference on Software Architecture, 2001. pp: 45-54.
- [25] van Lamsweerde, A. and Letier E. "Handling Obstacles in Goal-Oriented Requirements Engineering", IEEE Transactions on Software Engineering, vol. 26, 2000, pp:978-1005.
- [26] XML 1.0 (Extensible Markup Language). World Wide Web Consortium (W3C). Disponible en <http://www.w3.org/XML/> (última visita en mayo de 2006).
- [27] XSLT 2.0 (Extensible Stylesheet Language Transformation). World Wide Web Consortium (W3C). Disponible en <http://www.w3.org/XML/http://www.w3.org/TR/xslt20/> (última visita en mayo de 2006).
- [28] Yu, E. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering, 3rd IEEE Int. Symp. on Requirements Engineering (RE'97) Jan 1997, USA. pp:226-235.
- [29] Yu, E., Mylopoulos, J., Why goal-oriented requirements engineering, Proceedings of the Fourth International Workshop on Requirements Engineering: Foundations of Software Quality, Pisa, Italy, June 1998, pp: 15-22.
- [30] Yu, Y., Leite, J.C.S.P., and Mylopoulos, J.: From goals to aspects: discovering aspects from requirements goal models. 12th IEEE International Requirements Engineering Conference (RE'04). Sep. 6-10, pp:38-47