Universidad Politécnica de Cataluña
Departamento de Lenguajes y Sistemas Informáticos
Máster en Computación

Tesis de Máster

# Redefinición de Asociaciones en UML: Semántica y Utilización

**Estudiante**: Pilar Nieto Soler
**Director(es)**: Dolors Costal Costa y Cristina Gómez Seoane

**Fecha**: 18 de Septiembre, 2008

| | |
|---|---|
| *Título:* | Redefinición de asociaciones en UML: Semántica y Utilización |
| *Autor:* | Pilar Nieto Soler |
| *Fecha:* | 18 de Septiembre, 2008 |
| *Resumen:* | Un inconveniente importante que presenta hoy en día UML es la falta de semántica formal. Existen muchos conceptos que no define con la suficiente precisión como para que puedan ser interpretados sin ambigüedades.<br>Uno de los principales objetivos de este trabajo es precisar la semántica de la redefinición de asociaciones, un constructor de UML que nos permite definir de manera más específica extremos de asociaciones. Así mismo, comparamos este constructor con conceptos similares, como el ***subsetting*** (de UML) o el refinamiento de asociaciones (de otros lenguajes de modelado), con el objetivo de mostrar claramente que se tratan de conceptos semánticamente diferentes. Todo ello ayudará al diseñador a hacer un uso correcto del constructor de la redefinición de asociaciones.<br>Otra contribución significativa de este trabajo es la de incorporar a UML la semántica del refinamiento de asociaciones. Para ello, creamos nuevos estereotipos que nos permitirán incorporar todos aquellos casos que podemos expresar con el refinamiento y que no quedan cubiertos por la redefinición de asociaciones. Finalmente, implementamos estos estereotipos en la herramienta CASE *PoseidonUML*. |
| *Palabras clave:* | Modelado conceptual, Semántica de asociaciones, Redefinición de asociaciones en UML. |

## Índice general

## Índice de figuras

## Índice de tablas

## 1. Introducción

El modelado conceptual es una actividad que consiste en realizar una descripción del conocimiento general de un sistema de información. El resultado de esta actividad es lo que se llama esquema conceptual. Los esquemas conceptuales se especifican en lenguajes llamados lenguajes de modelado conceptual, tales como el UML (*Unified Modeling Language*) [Oli07].

El modelado conceptual es una actividad crucial en el desarrollo de sistemas de información. Por tanto, es de vital importancia que el esquema conceptual describa con exactitud el sistema de información que se desea desarrollar. Para ello, es imprescindible conocer bien el lenguaje de modelado con el que se desea especificar el esquema conceptual. El diseñador, encargado de llevar a cabo la tarea de modelado, debe saber qué conceptos le ofrece el lenguaje de modelado y entenderlos con precisión.

Durante la última década, UML ha sido ampliamente utilizado en la industria y en el ámbito docente, contribuyendo a la mejora en el desarrollo de sistemas de información. Sin embargo, un inconveniente de UML que ha sido señalado con frecuencia es la falta de semántica formal.

Si bien el metamodelo de UML [OMG07] nos da información sobre su sintaxis abstracta, su semántica se describe en lenguaje natural. De manera que existen muchos conceptos que no tienen definiciones suficientemente precisas para que puedan ser interpretados sin ambigüedad. UML 2.0 ha hecho un paso importante hacia la precisión semántica de conceptos. Sin embargo, su intento de aumentar la expresividad del lenguaje ha introducido nuevas ambigüedades y aún existen cuestiones que permanecen abiertas. Esto es un obstáculo para un uso apropiado y efectivo del lenguaje y para la creación de modelos que describan claramente el conocimiento del sistema de información. Además, esto dificulta el hecho de que una máquina pueda procesar el lenguaje, realizar una ejecución automática, un razonamiento automático y generar automáticamente modelos subsecuentes como propone el *Model Driven Architecture* [OMG03].

Actualmente, existe una línea abierta de investigación (con trabajos como [CG06, COT01, GR02]) que gira entorno a precisar la definición de conceptos relacionados con las asociaciones, elementos estructurales fundamentales en UML.

Uno de los principales objetivos de este trabajo es hacer más comprensible y precisa la semántica de la redefinición de asociaciones en UML, un concepto introducido en UML 2.0. La redefinición de una asociación nos permite definir restricciones de integridad adicionales sobre dicha asociación, tales como restricciones de participación o cardinalidad.

En este trabajo proponemos una clasificación de los diferentes tipos de redefinición de asociaciones, en función de la restricción adicional que se desea definir. Para precisar la semántica de cada uno de estos tipos de redefinición utilizamos lo que denominamos capa básica de UML, un subconjunto de elementos básicos de UML de los que se tiene una definición precisa, junto a invariantes de OCL (*Object Contraint Language*) [OMG06].

Existen otros conceptos similares a la redefinición de asociaciones que permiten expresar también restricciones de integridad sobre asociaciones. Es el caso por ejemplo del ***subsetting*** de asociaciones. La especificación de UML 2.0 no describe con claridad la distinción entre la redefinición y el ***subsetting*** de asociaciones. Esto conlleva a la existencia de dificultades a la hora de escoger cuál de estos conceptos se adapta mejor a

unos determinados requisitos. En este trabajo se compara semánticamente la redefinición y el *subsetting* de asociaciones a través de la capa básica de UML. Como resultado de esta comparación, se identifican una serie de diferencias que ayudarán al diseñador a discernir entre ambos conceptos.

Otro de los conceptos con los que comparamos semánticamente la redefinición, es el refinamiento de asociaciones, un concepto usado en algunos lenguajes de modelado conceptual (como por ejemplo, Syntropy [CD94]), que hoy en día no ofrece el UML. Esta comparación se lleva a cabo también a través de la capa básica de UML. Como resultado de esta comparación, vemos que existen una serie de casos que pueden ser expresados a través del refinamiento y que no quedan cubiertos por la redefinición de asociaciones.

En último lugar y con el fin de incorporar toda la semántica del refinamiento de asociaciones a UML, realizamos una extensión de este lenguaje utilizando un mecanismo estándar llamado perfil de UML (en inglés, *UML profile*). Concretamente, extendemos el perfil creado en [CGQ+06] añadiendo nuevos estereotipos que implementamos a través de un *plugin* para la herramienta CASE *PoseidonUML*. Estos estereotipos nos permiten expresar en UML todos aquellos casos de refinamiento, que no quedan cubiertos por la redefinición de asociaciones.

Este trabajo se organiza de la siguiente manera. En la sección 2 veremos una descripción general de otros trabajos que estudian la semántica de las asociaciones en UML. La sección 3 está dedicada a explicar los conceptos básicos. En la sección 4 se precisa la semántica de la redefinición de asociaciones en UML. Las secciones 5 y 6 están dedicadas a explicar el significado del *subsetting*, el refinamiento de asociaciones y a comparar ambos conceptos con la redefinición de asociaciones en UML. En la sección 7 proponemos los estereotipos necesarios para incorporar a UML la semántica del refinamiento de asociaciones y finalmente, en la sección 8 mostramos las conclusiones y trabajo futuro.

## 2. Descripción general de la Investigación en la Semántica de las Asociaciones en UML

En la última década, se han realizado diversos trabajos [Fra99, GR02, RG98, Szl06] dedicados a formalizar los conceptos estructurales básicos de UML. Estos conceptos incluyen clases y asociaciones, así como conceptos relacionados con éstas, tales como multiplicidad, agregación y composición.

France [FRA99] utiliza el lenguaje de especificación Z para formalizar estos conceptos, mientras que [RG98, Szl06] optan por utilizar una notación matemática. Por su parte, Gogolla y Richters [GR02] transforman diagramas de clases en UML, que incluyen restricciones de cardinalidad, calificadores, clases asociativas, agregaciones, composiciones y generalizaciones en diagramas de clases equivalentes, utilizando sólo asociaciones binarias y restricciones OCL.

La asociación es uno de los elementos estructurales más importantes en UML. Por este motivo, existen muchos trabajos de investigación cuyo objetivo es aclarar y/o formalizar su semántica y la semántica de conceptos relacionados con ésta.

Génova y otros autores [GLM02] aclaran el significado de los valores de la multiplicidad en asociaciones n-arias y proponen una extensión de la notación de UML para precisar su significado. Stevens [Ste01, Ste02] estudia algunos temas que a menudo son objeto de diferentes interpretaciones, tales como las asociaciones estáticas versus las dinámicas, asociaciones derivadas, multiplicidad y múltiples enlaces (en inglés, *links*), etc. Barbier y otros autores [BHL+03] definen formalmente la semántica del concepto de relación de las partes con el todo (en inglés, *part-whole relationship)*, analizan su relación con la agregación y composición de UML y proponen una extensión de UML para cubrir este tipo de relación. [APFR+03] presenta una particular interpretación de la semántica de los conceptos de agregación y composición en UML. Esta interpretación se obtiene utilizando un *framework* multidimensional, que identifica un conjunto de propiedades (dimensiones) que nos permite caracterizar sin ambigüedades los conceptos de agregación y composición. Génova y otros autores [GLF04] estudian la distinción entre las asociaciones estáticas y dinámicas y proponen una distinción alternativa entre asociaciones estructurales y contextuales. Diskin y Dingel [DD06] presentan un *framework* formal, basado en conjuntos y *mappings*, que reconcilia las vistas estructural y operacional de asociaciones. Milicev [Mil07] describe los problemas que existen en la interpretación semántica de la unicidad de extremos de asociaciones y propone una interpretación alternativa. Además, formaliza a través del lenguaje Z, los conceptos de asociación y extremo de asociación, junto con los conceptos relacionados de multiplicidad, unicidad y ordenación. Algunos de estos trabajos [BHL+03, DD06, Mil07] proporcionan una formalización para describir la semántica de conceptos relacionados con las asociaciones, mientras que otros [APFR+03, GLM02, Ste01, Ste02, GLF04] proponen descripciones precisas que los clarifican.

Varios de estos últimos trabajos [GLM02, Ste02, GLF04] han sido utilizados por otros autores [AHM06, GRL03] para implementar el concepto de asociación utilizando el lenguaje de programación Java. [AHM06] desarrolla patrones de generación de código que permiten generar automáticamente código Java, a partir de diagramas de clases especificados en UML 2.0. Por su parte, Génova y otros autores [GRL03] presentan algunos principios para la implementación de asociaciones binarias de UML, prestando especial atención en la multiplicidad, navigabilidad y visibilidad.

Otro concepto relacionado con las asociaciones, sobre el que se han realizado pocos estudios de investigación, es la redefinición de asociaciones en UML. [CG06] describe cómo usar este concepto para declarar restricciones adicionales de integridad y cardinalidad para asociaciones. Además, analiza las interacciones entre restricciones taxonómicas y redefiniciones de asociaciones y establece una serie de condiciones que son necesarias para garantizar redefiniciones de asociaciones bien definidas.

La redefinición de asociaciones es un concepto que se encuentra estrechamente relacionado con el refinamiento de asociaciones. Olivé [Oli07] utiliza la redefinición de asociaciones en UML para describir los dos tipos de refinamiento más conocidos: el refinamiento de participantes y el refinamiento de restricciones de cardinalidad. [COT01] estudia estos dos tipos de refinamientos en modelos conceptuales con múltiple clasificación. Además, propone una notación gráfica y textual para su especificación, así como sus definiciones formales en términos lógicos. Finalmente, proporciona una serie de condiciones necesarias para garantizar que dado un conjunto de refinamientos, éste es válido. Otros trabajos que estudian y utilizan los refinamientos en modelos conceptuales son [BO92, BS95, CD94, MBW80].

El s*ubsetting* de asociaciones en UML es otro concepto, que al igual que la redefinición de asociaciones, ha recibido poca atención en el campo de la investigación. [Oli07] define el concepto de ***subsetting*** de asociaciones como una restricción de inclusión entre dos asociaciones, es decir, la población de una de las asociaciones debe estar incluida en la población de la otra asociación. Akehurst y otros autores [AHM07] presentan un patrón de generación de código que permite la generación autómatica de código Java, a partir de un ***subsetting*** de una asociación en UML.

Como hemos visto, se han realizado muchos trabajos entorno al concepto de asociación en UML y conceptos relacionados con ésta. No obstante, tal y como comenta Milicec [Mil07], son necesarias nuevas investigaciones para cubrir todos los aspectos importantes relacionados con las asociaciones en UML 2.0. Milicev señala además, que algunos de los conceptos que deben ser estudiados con mayor detalle son la redefinición o el ***subsetting*** de asociaciones en UML.

## 3. Conceptos Básicos

Uno de los principales objetivos de este trabajo es definir de manera precisa la semántica de la redefinición de asociaciones en UML. La definición del significado exacto de este concepto permite a los diseñadores usarlo apropiadamente o interpretarlo correctamente. Además, una semántica sin ambigüedades de este constructor es la base para la validación y verificación de éste.

Existe mucha confusión sobre lo que constituye la semántica para los constructores UML. En [HR04] se señalan diferentes ideas de lo que es la semántica y maneras erróneas de verla. Trabajos como [HR04, Rum00] tratan de aclarar algunos de los conceptos involucrados en la definición de la semántica de los constructores UML. Estos trabajos distinguen la notación del lenguaje del constructor, o sintaxis, de su significado, o semántica. Para precisar el significado de un constructor UML se debe definir su sintaxis y semántica.

En las siguientes subsecciones, explicamos el concepto de sintaxis y semántica y el enfoque que seguimos para definir de manera precisa la semántica de las redefiniciones de asociaciones en UML.

### *3.1 Sintaxis*

El término “sintaxis” se usa siempre que nos referimos a alguna notación. En principio, la sintaxis define un lenguaje *L* (que puede ser textual o gráfico) de declaraciones y afirmaciones bien formadas [Rum00].

Para definir la sintaxis de un constructor en UML, se debe declarar su sintaxis concreta y abstracta.

La sintaxis concreta proporciona las reglas para definir cómo el constructor debe aparecer cuando se escribe (es decir, su notación). UML es un lenguaje de modelado visual que usa elementos como líneas, cajas y flechas para expresar la sintaxis de sus constructores. La sintaxis concreta de UML se muestra con ejemplos en [BRJ99, RJB05, OMG07].

La sintaxis abstracta identifica los principales conceptos que describen su notación. En UML, la sintaxis abstracta se presenta como un modelo (el metamodelo de UML [OMG07]). Este metamodelo consiste en un diagrama de clases UML con una descripción en lenguaje natural y un conjunto de reglas sintácticas denominadas en inglés, ***well-formedness rules.*** Estas reglas se presentan utilizando un lenguaje formal (*Object Constraint Language* [OMG06]).

### *3.2 Semántica*

La semántica de un lenguaje *L* define el significado de cada constructor del lenguaje. Una definición semántica consiste en dos pasos: primero, definir un dominio semántico y segundo, proporcionar un ***mapping*** entre la sintaxis y el dominio semántico [Rum00]. Las siguientes subsecciones explican en detalle estos dos pasos.

### *3.2.1 Dominio Semántico*

La explicación de un constructor de un lenguaje *L* debe hacerse en términos de conceptos ya conocidos y que se entiendan bien. El dominio o el lenguaje donde estos conceptos se entienden bien, se llama dominio semántico o lenguaje semántico. Este dominio semántico, que nosotros denotamos como *S*, nos permite definir con precisión los conceptos de la sintaxis abstracta.

En general, diferentes notaciones o lenguajes pueden ser utilizados para describir el dominio semántico: lenguajes formales como términos matemáticos (como se hace en [Szl06]), el lenguaje Z (como en [ZM04]) o incluso el mismo UML (como en [GR02]). En este último caso, sólo un subconjunto de elementos básicos de UML con una definición precisa de su semántica, junto con invariantes OCL (llamado capa de UML) se suele considerar como dominio semántico.

Nosotros usamos una capa básica de UML como dominio semántico, ya que uno de los principales objetivos de este trabajo es definir claramente el concepto de redefinición de asociaciones y hacer comprensible esta definición para cualquier usuario de UML.

Los elementos que consideramos en nuestra capa básica de UML son: clases, asociaciones binarias, multiplicidades, generalizaciones/especializaciones y restricciones generales en OCL. A continuación, se explica intuitivamente la semántica de cada uno de estos elementos. Su formalización precisa y completa puede encontrarse en varios trabajos como [OMG06 (Appendix), RG98, Szl06]:

- *Clase*. Una clase proporciona una descripción común para un conjunto de elementos que comparten las mismas propiedades. El dominio de una clase es el conjunto de objetos que se pueden crear para esta clase y para todas las clases hijo (véase generalización/especialización). Los objetos son referenciados a través de identificadores únicos para cada objeto. Cada objeto se determina únicamente por su identificador y viceversa.
- *Asociación binaria*. Las asociaciones describen relaciones estructurales entre clases. Una asociación binaria describe una relación entre dos clases. Nosotros sólo consideramos asociaciones binarias y además no recursivas, es decir, asociaciones que relacionan dos clases diferentes. El dominio de una asociación es el producto cartesiano del conjunto de identificadores de los objetos de las clases que participan. Un *link* que indica una conexión entre objetos es un elemento de ese producto cartesiano. Suponemos sin pérdida de generalidad la unicidad de las asociaciones.
- *Extremos de asociación*. Una asociación binaria tiene dos extremos de asociación, cada uno de los cuales está conectado a una clase. Nosotros consideramos extremos de asociación no derivados.
- *Multiplicidad*. La multiplicidad se especifica para los extremos de asociaciones. Ésta restringe el número de *links* en los que un objeto puede formar parte.
- *Generalización/especialización*. Una generalización es una restricción taxonómica entre dos clases (nosotros no consideramos la generalización de asociaciones en esta capa básica). Esta restricción especializa una clase general (clase padre) en una clase más específica (clase hijo). Las especializaciones y generalizaciones son dos puntos de vista del mismo concepto. Las relaciones de generalización forman una jerarquía sobre un conjunto de clases. Una jerarquía de generalización induce una relación de subconjunto en el dominio semántico de clases. El conjunto de los

identificadores de objetos de una clase hijo es un subconjunto de los identificadores de objetos de sus clases padre.

- *Restricciones generales en OCL*. Una restricción general es una condición o una restricción expresada en algún lenguaje con el propósito de declarar algún aspecto de la semántica de un elemento. Nosotros consideramos que las restricciones generales son expresadas en OCL. En [OMG06] se define formalmente la sintaxis y semántica del OCL.

### *3.2.2 Mapping Semántico*

Dado un lenguaje *L* y un dominio semántico *S*, el segundo paso de la definición semántica es establecer un *mapping M* (es decir, $M$: $L \rightarrow S$) entre los conceptos sintácticos definidos en *L* y los conceptos del dominio semántico *S* [Rum00].

Un *mapping* entre un lenguaje *L* y un dominio semántico es una función que proporciona para cada constructor del lenguaje *L* una explicación de este constructor en el dominio semántico *S*. Esta explicación se da a través de los conceptos que se comprenden bien en el dominio semántico *S*.

Los *mappings* semánticos pueden ser definidos de diferentes maneras. Una manera es establecer la correspondencia entre el lenguaje y el metamodelo del dominio semántico, es decir, entre la sintaxis abstracta del constructor y los elementos adecuados del dominio semántico. Otra manera alternativa es definir el *mapping* algorítmicamente, es decir, traduciendo las expresiones en la sintaxis abstracta a expresiones en el lenguaje semántico [KER99]. En este trabajo utilizamos la primera alternativa.

## *3.3 El enfoque del Meta-modelado*

Existen diferentes enfoques generales para formalizar constructores del modelado orientado a objetos en UML, que definen estos constructores estableciendo un dominio semántico y un *mapping* entre la sintaxis y el domino semántico (tal como se ha explicado en subsecciones anteriores). La mayoría de estos enfoques han sido identificados por el *Precise UML group* (PUML) en [And00, FELR98].

Uno de los enfoques usados para formalizar constructores de UML, cuando se considera como dominio semántico una capa básica de UML, es el enfoque del meta-modelado. Este enfoque fue propuesto inicialmente en [KER99].

En este trabajo se utiliza el enfoque del meta-modelado para formalizar el concepto de la redefinición de una asociación en UML. Usamos este enfoque porque proporciona a los diseñadores una semántica precisa de los constructores (a través de su formalización) utilizando el mismo UML, sin necesidad de aprender otro lenguaje formal de especificación.

Los pasos básicos de este enfoque (extraídos de [KER99]) son los siguientes:

1. Desarrollar la sintaxis y semántica del núcleo del lenguaje de meta-modelado que se selecciona como dominio semántico. En nuestro caso, este lenguaje corresponde a la capa básica de UML presentada anteriormente. Su sintaxis concreta y abstracta se define de manera completa en [OMG06, OMG07, RBJ05] y su semántica se presenta utilizando como base teoría de conjuntos. Ésta última puede encontrarse en [OMG06, RG98, Szl06].
2. Transformar la sintaxis de los constructores UML en el metamodelo de UML, es decir, en una sintaxis abstracta. Este paso ya se ha hecho para el caso de redefinición de asociaciones. Las sintaxis concreta y abstracta para este

constructor son proporcionadas por OMG y explicadas en detalle en [OMG07, RJB05]. Una descripción general de ambas sintaxis para el constructor se muestran en las siguientes secciones.

3. Establecer un ***mapping*** entre la sintaxis abstracta del constructor (es decir, entre la parte del metamodelo que representa este constructor) y la descripción del metamodelo de la semántica (es decir, el metamodelo de la capa básica de UML) como se hace, por ejemplo, en [GR98]. Para entender mejor este ***mapping***, lo definimos como una traducción entre un esquema general usando el constructor y otro esquema general usando sólo elementos definidos en la capa básica de UML.

En las siguientes secciones mostramos, usando este enfoque, cómo transformar redefiniciones de asociaciones en elementos de la capa de UML.

## 4. Redefinición de Asociaciones en UML

La redefinición de una asociación binaria nos permite definir un extremo de asociación de manera más específica [OMG07, RJB05]. Diferentes redefiniciones pueden ser especificadas para cada extremo de una asociación y los dos extremos de una asociación binaria pueden ser redefinidos.

En las siguientes subsecciones se presenta una descripción general de la sintaxis concreta y abstracta de las redefiniciones de asociaciones en UML y se define una semántica precisa de este constructor.

### *4.1 Sintaxis*

La sintaxis concreta {redefines <nombre-extremo>} situada cerca de un extremo de asociación (el *redefining end*) indica que este extremo redefine a otro llamado <nombre -extremo> (el *redefined end*). La figura 4.1 muestra esta notación. Esta figura describe una asociación binaria ***R*** con un extremo ***b*** que es redefinido por otro extremo $b_1$. En la figura 4.1 a) el *redefining end* $b_1$ está conectado a la misma clase del *redefined end b* mientras que en la figura 4.1 b) el *redefining end* $b_1$ está conectado a un descendiente directo o indirecto de la clase conectada al *redefined end b*.

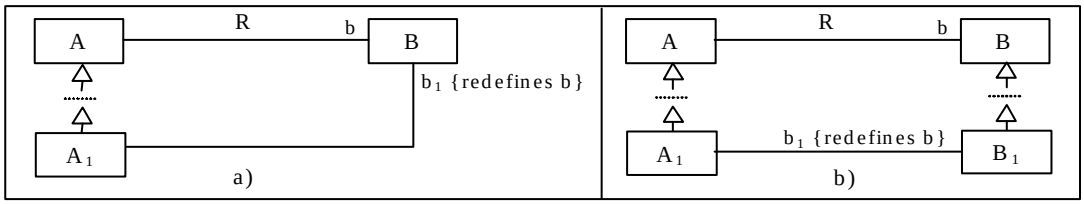



**Fig. 4.1.** Notación de la redefinición

Consideremos, por ejemplo, la redefinición que se muestra en la figura 4.2. El extremo *jeProyecto* para los empleados juniors define de manera más específica el extremo *proyecto* para estos empleados.

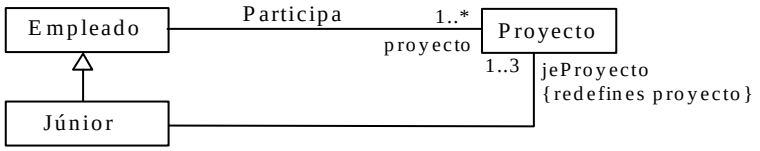



**Fig. 4.2.** Ejemplo de redefinición

La sintaxis abstracta identifica los conceptos principales que describen su notación. En UML, esta sintaxis se presenta a través del metamodelo de UML. La figura 4.3 muestra un fragmento del metamodelo de UML que contiene todos los conceptos que participan en la redefinición de extremos de asociaciones.

En general, un elemento que se puede redefinir (representado en el metamodelo de UML como una instancia de la metaclase *RedefinableElement*) es un elemento que, cuando es definido en el contexto de un clasificador, redefine de manera más especifica o diferente otro elemento en el contexto de otro clasificador que generaliza (directa o indirectamente) el contexto del clasificador. Los elementos que están siendo redefinidos son representados por el rol *redefinedElement.*

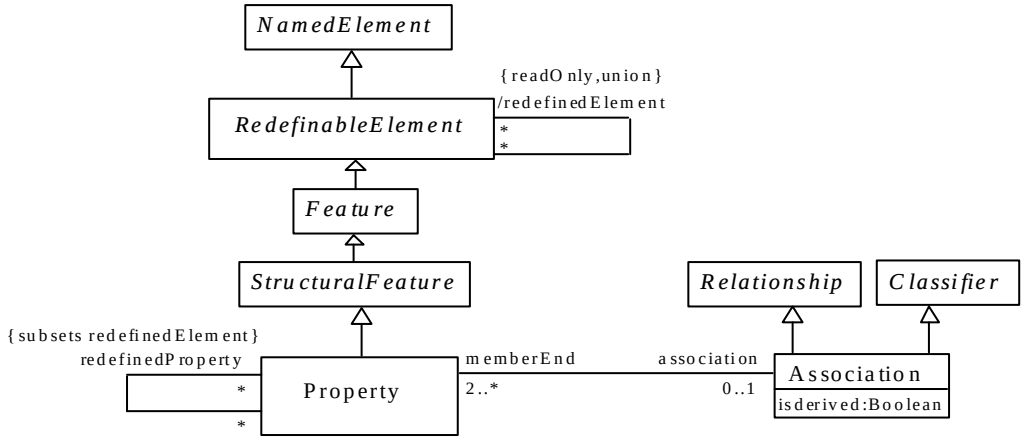



**Fig. 4.3.** Fragmento del metamodelo de UML que describe redefiniciones de extremos de asociaciones

La redefinición de extremos de asociaciones es un caso particular de la redefinición de propiedades. En general, las características de una propiedad que pueden ser redefinidas son el nombre, tipo (que puede ser especializado), valor por defecto, tipo de derivación, visibilidad, multiplicidad y restricciones en los valores. Para las propiedades que corresponden a extremos de asociación (es decir, propiedades con un rol *association* no vacío, véase figura 4.3) deducimos que sólo pueden ser definidas redefiniciones de nombre, tipo, visibilidad y multiplicidad. En estos casos, el rol *redefinedProperty* del extremo de una asociación, que es un elemento que se puede redefinir, contiene el extremo de asociación que está siendo redefinido por este extremo.

En este trabajo, nos centramos sólo en redefiniciones de asociaciones que restringen la especificación original (es decir, redefiniciones de tipo, multiplicidad y del tipo de derivación). Además, estudiamos las redefiniciones de nombre ya que éstas se combinan frecuentemente con otros tipos de redefiniciones.

Según una de las *well-formedness rules* definidas en el metamodelo de UML [OMG07], el extremo opuesto al *redefining end* debe estar siempre conectado a una clase que sea descendiente de la clase conectada al extremo opuesto del *redefined end*. Consideremos los diagramas descritos en la figura 4.1, donde $A_1$ es la clase conectada al extremo opuesto del extremo de $b_1$ y $A$ la clase conectada al extremo opuesto del extremo $b$. Entonces, $A_1$ debe ser un descendiente de $A$ (como en cada uno de los ejemplos de la figura). Como consecuencia, la redefinición de un extremo de asociación debe combinarse siempre con una relación de generalización/especialización.

Otra *well-formedness rule* establece que el *redefining end* puede estar conectado a la misma clase que el *redefined end* o a uno de sus descendientes [OMG07]. En el ejemplo de la figura 4.1 a), el extremo $b_1$ está conectado a la misma clase (es decir, la clase $B$) a la que está conectado el extremo $b$, mientras que en el ejemplo de la figura 4.1 b) $b_1$ está conectado a un descendiente de la clase conectada a $b$ (es decir, la clase $B_1$).

## *4.2 Semántica*

El efecto semántico de una redefinición se aplica sobre un subconjunto de las instancias de la asociación redefinida (es decir, la asociación con el *redefined end*). Llamamos a este subconjunto *instancias afectadas* de la redefinición. Para cualquier tipo de redefinición de asociación, las instancias afectadas son los *links* o instancias de la asociación redefinida, que conecta instancias de la clase del extremo opuesto del *redefining end* con otras instancias. Consideremos los ejemplos que se muestran en la figura 4.1, donde $R$ es una asociación redefinida que relaciona $A$ y $B$, y $A_1$ es la clase

conectada al extremo opuesto del *redefining end*. En este caso, las instancias afectadas son instancias de *R* que asocian instancias de $A_1$ con otras instancias.

En las siguientes subsecciones se explica la semántica y sintaxis abstracta específica de cada característica redefinida (nombre, tipo, multiplicidad y tipo de derivación) a través de traducciones a nuestra capa básica de UML. La última subsección muestra ejemplos de posibles combinaciones entre características redefinidas.

### *4.2.1 Redefinición de Nombre*

Decimos que una redefinición es una ***redefinición de nombre*** cuando el ***redefining end*** tiene un nombre diferente del ***redefined end***.

El efecto de una redefinición de nombre es dar un nuevo nombre a la propiedad del ***redefined end*** para las instancias afectadas de la redefinición.

Como consecuencia, el antiguo nombre del ***redefined end*** no puede ser usado por estas instancias. Las redefiniciones de nombre se combinan frecuentemente con la redefinición de otras características en una única redefinición de asociación aunque, según el metamodelo de UML, es posible especificar una redefinición que sólo redefine el nombre.

La figura 4.4 muestra un ejemplo de una redefinición que sólo redefine la característica del nombre. El nombre ***proyecto*** es redefinido por el extremo ***jeProyecto***. Su efecto es que el nombre ***jeProyecto*** se utilizará para referirse a los proyectos en los que participan empleados juniors.

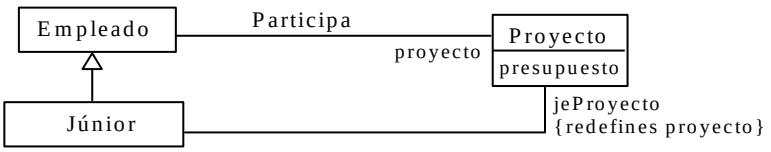



**Fig. 4.4.** Ejemplo de redefinición de nombre

Para precisar la semántica de la redefinición de nombre, indicamos en la figura 4.5 cómo traducirla a nuestra capa básica de UML. En la traducción, la asociación que redefine (es decir, la asociación con el ***redefining end***) es eliminada. La razón de esta eliminación es que todas sus instancias son instancias de la asociación redefinida y pueden ser deducidas de éstas. Esto no sorprende ya que el objetivo de una redefinición no es definir una nueva asociación, sino mejorar la definición de una asociación ya existente.

De esta traducción, se puede observar fácilmente que el efecto de una redefinición de nombre es sólo sintáctico y no tiene un efecto semántico en la asociación que se redefine. Esto es así porque la redefinición de nombre raramente se utiliza sola, sino que se combina normalmente con la redefinición de otras características.

Como la asociación que redefine ha desaparecido en la traducción del diagrama, las restricciones OCL sobre el diagrama que hacen referencia a $b_1$, deben ser rescritas para ser evaluadas en el diagrama traducido. Concretamente, '$b_1$' debe ser reemplazado por '*b*'. En la figura 4.5, ***expsOCL*** denota un conjunto de expresiones OCL sobre el diagrama original y ***nuevasExpsOCL*** denota el conjunto de expresiones obtenidas de reemplazar '$b_1$' por '*b*' en ***expsOCL***.

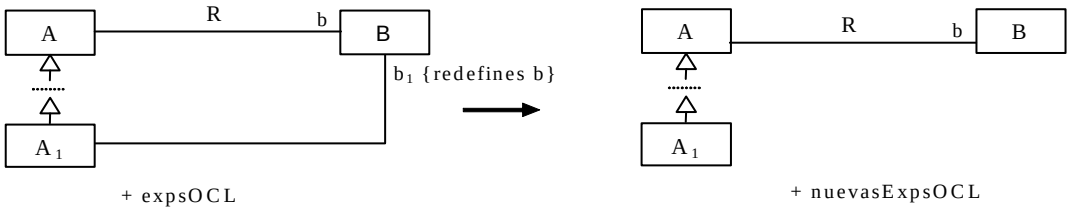



**Fig. 4.5.** Traducción de la redefinición de nombre

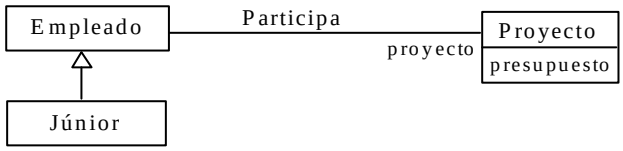
La traducción que se muestra en la figura 4.5 sólo representa aquellos elementos que deben ser cambiados del diagrama original, para obtener el diagrama equivalente en nuestra capa básica de UML. Otros elementos tales como atributos o la multiplicidad de la asociación redefinida no se muestran, ya que éstos seguirán siendo los mismos en el diagrama traducido.

Consideremos el ejemplo de la figura 4.4 y la siguiente expresión OCL (invariante) definida sobre él:

```
context Júnior inv:
self.jeProyecto->select(p | p.presupuesto>100000) -> size()<=1
```

Ésta restricción prohíbe que empleados juniors participen en más de un proyecto con un presupuesto mayor que 100000:

La figura 4.6 muestra la traducción del diagrama y su expresión OCL.




```
context Júnior inv:
self.proyecto->select(p | p.presupuesto>100000) -> size()<=1
```

**Fig. 4.6.** Traducción de la redefinición de nombre de la fig. 4.4

### *4.2.2 Redefinición de Tipo*

En general, el *redefining end* de una redefinición puede estar conectado a la misma clase a la que está conectado el *redefined end* o a alguno de sus descendientes [OMG07]. Decimos que una redefinición es una *redefinición de tipo* cuando el *redefining end* está conectado a un descendiente de la clase del *redefined end* ya que, claramente, el tipo no es redefinido cuando el *redefining end* está conectado a la misma clase que el *redefined end*.

El efecto de una redefinición de tipo es establecer una restricción de participación adicional sobre la asociación redefinida. Concretamente, se indica que las instancias afectadas de la redefinición deben asociar instancias de la clase opuesta al *redefining end* con instancias de la clase conectada al *redefining end*.

En el ejemplo de la figura 4.7, el extremo *ncProyecto* redefine el tipo del extremo *proyecto* ya que la clase *NoCrítico* es un descendiente de la clase *Proyecto*. El efecto es que los empleados juniors sólo pueden participar en proyectos no críticos. Nótese que no se impone la condición inversa, es decir, los proyectos no críticos pueden tener como participantes cualquier tipo de empleado. La razón es que sólo el extremo *proyecto* es redefinido pero no el extremo *empleado*.

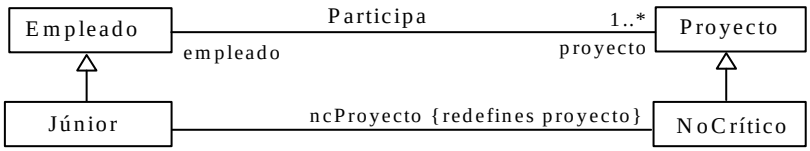



**Fig. 4.7.** Ejemplo de redefinición de tipo

La traducción de una redefinición de tipo a nuestra capa básica de UML, consiste en sustituir la redefinición por una restricción de participación que especifica de manera precisa su efecto sobre la asociación redefinida. La figura 4.8 indica la regla de equivalencia para esta traducción.

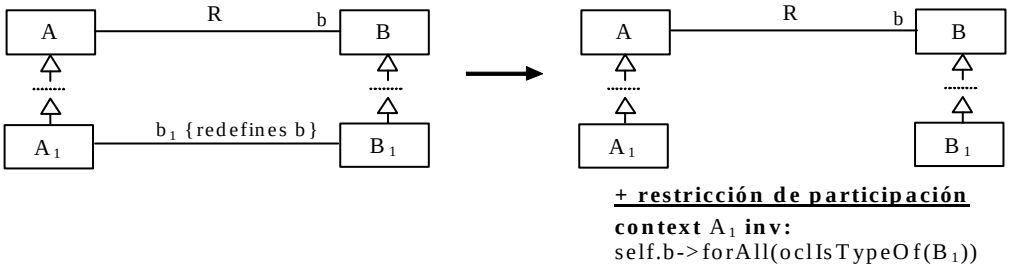



**Fig. 4.8.** Traducción de la redefinición de tipo

La restricción de participación es necesaria para asegurar que cada instancia del extremo *b* asociado con una instancia de $A_1$ debe ser instancia de $B_1$. Esta restricción puede ser expresada como un invariante de OCL tal y como se muestra en la figura 4.8.

Aunque la redefinición de la figura 4.8 redefine el tipo y el nombre, por claridad, la traducción propuesta se centra en el efecto de la redefinición de tipo.

La figura 4.9 muestra la traducción del ejemplo de la figura 4.7. El invariante OCL asegura que todos los proyectos de un empleado júnior son proyectos no críticos.

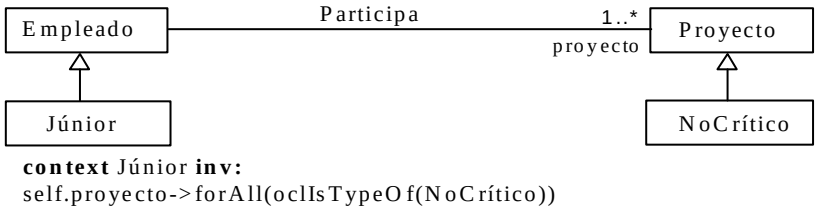



**Fig. 4.9.** Traducción de la redefinición de tipo de la fig. 4.7

### *4.2.3 Redefinición de Multiplicidad*

Según el metamodelo de UML, la multiplicidad del *redefining end*, si se especifica, debe ser igual o más restrictiva que la multiplicidad especificada en el *redefined end* [OMG07]. Decimos que una redefinición es una *redefinición de multiplicidad* cuando se especifica una multiplicidad en el *redefining end* y ésta es más restrictiva que la del *redefined end*. De lo contrario, la multiplicidad no sería redefinida.

El efecto de una redefinición de multiplicidad es establecer una restricción de cardinalidad adicional sobre la asociación redefinida. Básicamente, ésta restringe la multiplicidad permitida para las instancias afectadas de la redefinición.

En el ejemplo de la figura 4.10, el extremo *jeProyecto* redefine la multiplicidad del extremo *proyecto*. El efecto de esta redefinición es que los empleados juniors no pueden participar en más de tres proyectos. Nótese que la multiplicidad del *redefinig end*, *1..3*, es más restrictiva que la multiplicidad redefinida, *1..**.

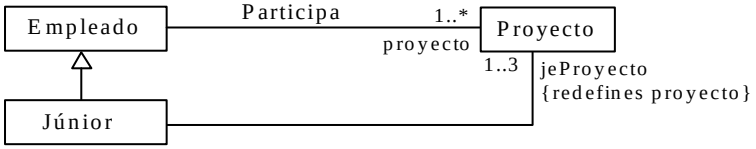



**Fig. 4.10.** Ejemplo de redefinición de multiplicidad

La figura 4.11 indica cómo traducir una redefinición de multiplicidad a nuestra capa básica de UML, con el fin de hacer más preciso su significado semántico. En la traducción, la redefinición es sustituida por una restricción de cardinalidad que especifica su efecto sobre la asociación redefinida.

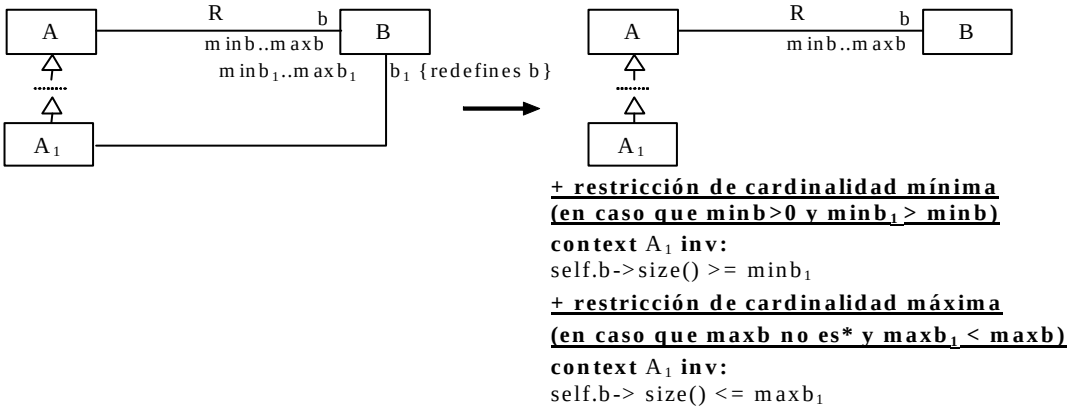



**Fig. 4.11.** Traducción de la redefinición de multiplicidad

Las restricciones de cardinalidad son necesarias para asegurar que cada instancia de la clase $A_1$ está asociada a través de la asociación *R* a un número $minb_1$ de instancias como mínimo y a un número $maxb_1$ de instancias como máximo. En general, estas restricciones pueden ser expresadas en OCL tal y como se muestra en la figura 4.11.

La figura 4.12 muestra la traducción del ejemplo de la figura 4.10. El invariante OCL garantiza que los empleados juniors no pueden participar en más de tres proyectos.

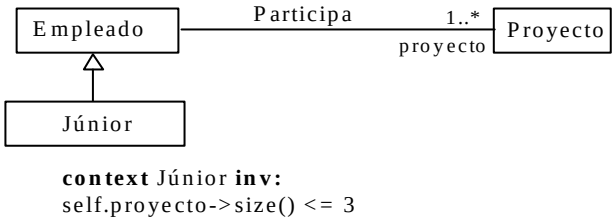


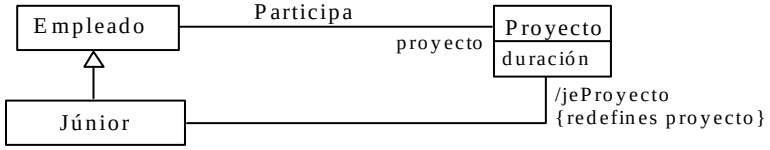

**Fig. 4.12.** Traducción de la redefinición de multiplicidad de la figura 4.10

### *4.2.4 Redefinición del tipo de derivación*

Según se indica en [OMG07] sólo están permitidas redefiniciones de propiedades no derivadas en derivadas. Decimos que una redefinición es una *redefinición del tipo de derivación* cuando una barra (/) se sitúa delante del nombre del *redefining end*.

El efecto de la redefinición del tipo de derivación es cambiar el tipo de derivación (es decir, la manera que utiliza el sistema para conocer la población) de la propiedad del *redefined end* para las instancias afectadas de la redefinición. Concretamente, la redefinición del tipo de derivación indica que las instancias afectadas de la redefinición deben ser deducidas o calculadas por la regla de derivación (nosotros suponemos reglas de derivación expresadas en OCL).

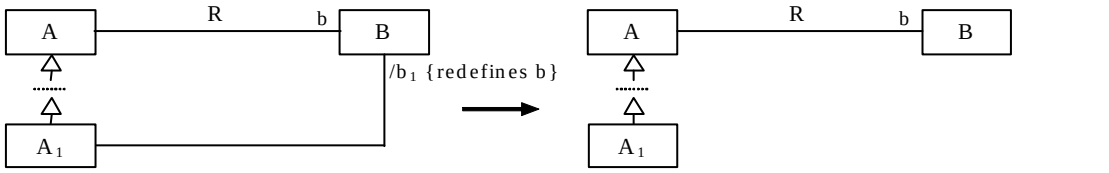
Consideremos el ejemplo de la figura 4.13. El extremo *jeProyecto* redefine el tipo de derivación del extremo *proyecto*. El efecto de esta redefinición es que los empleados juniors participaran en proyectos (calculados por la regla de derivación) con un máximo de ocho semanas de duración.




**context** Júnior :: jeProyecto: Set (Proyecto)
**derive:** Proyecto.allInstances->select(p | p.duración <= 8)

**Fig. 4.13.** Ejemplo de una redefinición del tipo de derivación

Para hacer más precisa la semántica de la redefinición del tipo de derivación, mostramos en la figura 4.14 cómo traducirla a nuestra capa básica de UML. En la traducción, la redefinición es eliminada y la regla de derivación para $b_1$ es sustituida por una restricción sobre el extremo *b* para las instancias afectadas. Esta restricción establece que las instancias de *B* que están asociadas con cada instancia de $A_1$ se obtienen a través de la regla de derivación para $b_1$.



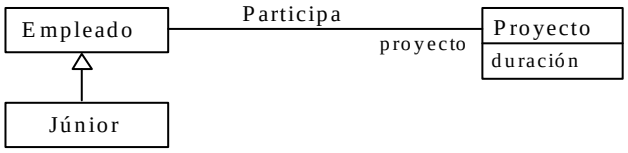

**+ regla de derivación para $b_1$**
**context** $A_1$ :: $b_1$: Set(B)
derive: regla de derivación para $b_1$

**+ restricción general deducida de la regla de derivación para $b_1$**
**context** $A_1$ **inv:**
self.b = expresión OCL de la regla de derivación para $b_1$

**Fig. 4.14.** Traducción de la redefinición del tipo de derivación

Obsérvese que el diagrama original y traducido son equivalentes con respecto a las instancias y *links* que éstos contienen (es decir, las instancias de *A*, $A_1$ y *B* y los *links* de *R* son las mismos en ambos diagramas) aunque la manera que utiliza el sistema para conocerlas es diferente. En el diagrama original los *links* de la asociación redefinida afectados por la redefinición son calculados automáticamente por la regla de derivación, mientras que en el diagrama traducido estos *links* deben ser proporcionados explícitamente por el usuario (y deben cumplir la restricción definida sobre el extremo *b*).

En la figura 4.15 mostramos cómo traducir el ejemplo de la figura 4.13. El invariante OCL garantiza que los proyectos en los que participan empleados juniors son proyectos con una duración máxima de ocho semanas.




**context** Júnior **inv:**
self.proyecto = Proyecto.allInstances->select(p | p.duración <= 8)

**Fig. 4.15.** Traducción de la redefinición del tipo de derivación de la fig. 4.13

### *4.2.5 Visión global*

En las anteriores subsecciones se ha definido de manera precisa la semántica de la redefinición de asociaciones, a través de la traducción de cada una de las características redefinidas a nuestra capa básica de UML. Todas estas traducciones se encuentran resumidas en la tabla 4.1. Mediante su análisis, es posible obtener una visión global del significado de la redefinición de asociaciones en UML.

El principal objetivo de una redefinición de asociación no es añadir una nueva asociación, sino mejorar la definición de una asociación existente. Por este motivo, la asociación que redefine no aparece en la traducción de ninguna redefinición (véase tabla 4.1). Además, las instancias de la asociación que redefine pueden ser siempre deducidas de las de la asociación redefinida.

En segundo lugar, la definición de la asociación redefinida se mejora en todos los casos al restringir el conjunto de instancias que están permitidas para ésta, con la única excepción de aquellos casos que solo redefinen la característica de nombre. Los tipos de restricciones que se imponen dependen de las características que están siendo redefinidas. Concretamente, las restricciones de tipo establecen restricciones de participación adicionales, las redefiniciones de multiplicidad establecen restricciones de cardinalidad mínima y máxima y las redefiniciones del tipo de derivación imponen restricciones que dependen de la regla de derivación proporcionada por el diseñador. La tabla 4.1 resume estas restricciones. La especificación precisa de estas restricciones se ha realizado a través de expresiones OCL en subsecciones anteriores.

Las redefiniciones de nombre no siguen este patrón y sólo tienen el efecto sintáctico de dar un nuevo nombre a la propiedad redefinida para las instancias afectadas de la redefinición. Esto es así, porque la traducción de la redefinición de nombre simplemente consiste en la sustitución de ese nombre en las expresiones OCL definidas en el diagrama original (véase tabla 4.1).

Para concluir, debemos señalar que las redefiniciones del tipo de derivación tienen una particular implicación adicional que no es tratada por la traducción propuesta, tal y como se explica en la sección 4.2.4, es decir, las instancias afectadas de la redefinición son calculadas automáticamente por la regla de derivación en vez de ser proporcionadas por los usuarios del sistema, tal y como ocurre con el resto de instancias de la asociación redefinida.

| | | **Asociación que redefine** | ***Redefining end*** | **Restricciones** |
|---|---|---|---|---|
| **Característica redefinida** | **Nombre** | Eliminada | Sustituido por el *redefined end* en las expresiones OCL | |
| | **Tipo** | Eliminada | | -Restricción de participación |
| | **Multiplicidad** | Eliminada | | -Restricción de cardinalidad mínima<br>-Restricción de cardinalidad máxima |
| | **Tipo de derivación** | Eliminada | | -Restricción general deducida de la regla de derivación para $b_1$ |

**Tabla 4.1.** Resumen de las traducciones propuestas para redefiniciones de extremos de asociaciones

### *4.2.6 Combinaciones*

Diferentes redefiniciones pueden ser especificadas para un extremo de una asociación y diferentes características pueden ser redefinidas por una única redefinición. En general, cualquier combinación de características puede ser redefinida. La traducción de estas combinaciones se obtiene traduciendo por separado cada una de las características redefinidas y combinando el resultado.

En esta sección, presentamos dos ejemplos particulares que muestran algunas posibles combinaciones. El primero de ellos, corresponde a dos redefiniciones de tipo especificadas para una única asociación, una por cada extremo de la asociación. El segundo, se compone de redefiniciones que redefinen varias características simultáneamente: nombre, tipo y multiplicidad.

#### *4.2.6.1 Combinación de dos redefiniciones de tipo*

Consideremos la asociación definida en la figura 4.16 que relaciona personas con sus billetes para viajar. Algunas personas son niños, tal y como indica la especialización de la clase *Persona*. De manera similar, algunos billetes son billetes infantiles.

Suponemos que queremos indicar que los niños sólo pueden viajar con billetes infantiles y los billetes infantiles sólo están permitidos para niños. Para ello, se necesitan dos redefiniciones, una para cada extremo de la asociación. Ambas redefinen únicamente la característica de tipo.

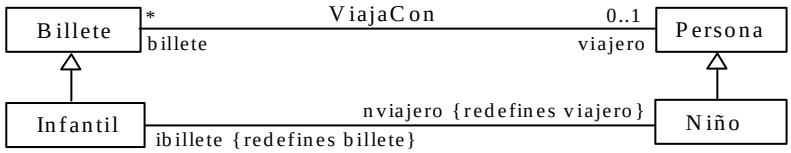



**Fig. 4.16.** Ejemplo de dos redefiniciones de tipo para dos extremos de una sola asociación

La figura 4.17 proporciona la traducción del ejemplo anterior.


```
context Infantil inv:
self.viajero->forAll(oclIsTypeOf(Niño))

context Niño inv:
self.billete->forAll(oclIsTypeOf(Infantil))
```

**Fig. 4.17.** Traducción de las redefiniciones de la fig. 4.16

#### *4.2.6.2 Redefiniciones de nombre, tipo y multiplicidad*

Explicamos este caso usando el ejemplo de la figura 4.18 que hace referencia a un fragmento del ejemplo mostrado en [RJB05]. El diagrama de clases define cuentas y entidades legales en un banco. Las cuentas tienen como mínimo una entidad legal como propietario y pueden tener diversos agentes. Las entidades legales tienen como mínimo una cuenta y pueden actuar como agentes de varias cuentas. Las cuentas son especializadas en cuentas personales y corporativas. Las entidades legales son especializadas en las clases *Persona* y *Compañía*

Se han especificado tres redefiniciones. Dos de ellas, redefinen el nombre, tipo y multiplicidad y una de ellas redefine el tipo y multiplicidad. Como podemos observar, dos de ellas redefinen el mismo extremo, es decir, el extremo *agente*.

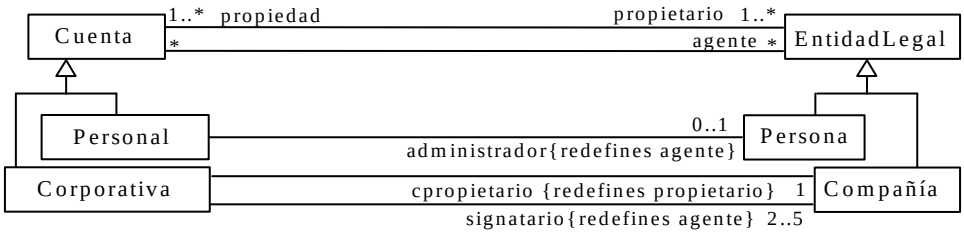



**Fig. 4.18.** Ejemplo de redefiniciones de nombre, tipo y multiplicidad de un extremo de asociación

La traducción de las redefiniciones anteriores se muestra en la figura 4.19. Como podemos ver en la figura, 'administrador' y 'signatario' han sido reemplazados por 'agente' en las expresiones OCL sobre el diagrama original.


```
context Personal inv:
self.agente->forAll(oclIsTypeOf(Persona)) and
self.agente->size() <= 1

context Corporativa inv:
self.propietario->forAll(oclIsTypeOf(Compañía)) and
self.propietario->size() = 1 and
self.agente->forAll(oclIsTypeOf(Compañía)) and
self.agente->size() >= 2 and
self.agente->size() <= 5
```

**Fig. 4.19.** Traducción de las redefiniciones de la fig. 4.18

## 5. Subconjunto de Asociaciones en UML: Comparación con la Redefinición de Asociaciones

### *5.1 Subconjunto de una asociación en UML*

El subconjunto (en inglés, ***subsetting***) de una asociación binaria nos permite expresar que la colección de instancias de un extremo de una asociación (el ***subsetting end***), asociada a una determinada instancia, debe estar incluida (o ser igual) a la colección de instancias de un extremo de otra asociación (el ***subsetted end***), asociada a la misma instancia [RJB05, OMG07]. El ***subsetting*** es un constructor de UML que se define para las colecciones representadas por extremos de asociaciones y no para las asociaciones en sí.

En las siguientes subsecciones se presenta la sintaxis concreta y abstracta del ***subsetting*** de asociaciones en UML y se define de manera precisa la semántica de este constructor.

#### *5.1.1 Sintaxis*

La sintaxis concreta {subsets <nombre-extremo>} que se sitúa cerca del extremo de una asociación (***subsetting end***), indica que la colección de dicho extremo, asociada a una determinada instancia, es un subconjunto de la colección de un extremo de otra asociación (***subsetted end***), asociada a dicha instancia.

La figura 5.1 muestra su notación para los diferentes escenarios que se pueden dar a la hora de definir un ***subsetting***. En esta figura, vemos que el extremo ***b*** de la asociación binaria ***R*** corresponde al ***subsetted end*** y el extremo $b_1$ de la asociación $R_1$ al ***subsetting end***.

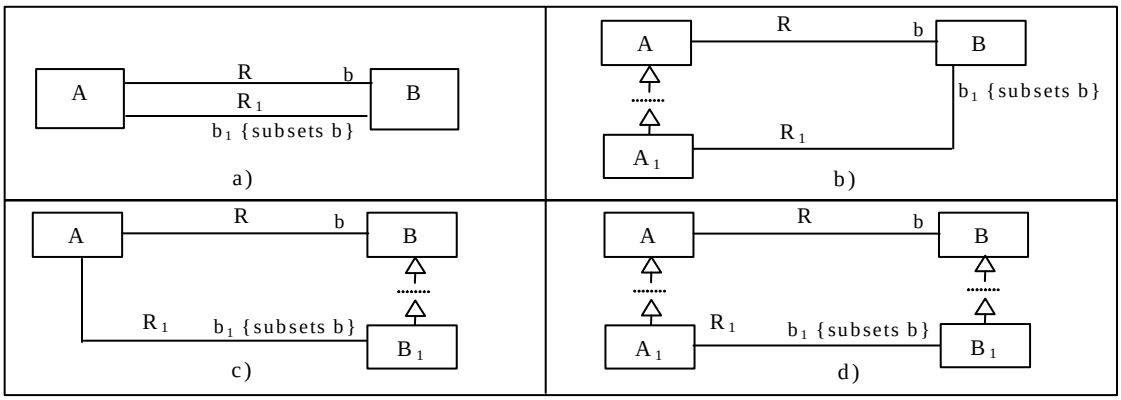



**Fig. 5.1.** Notación del ***subsetting***

La figura 5.2 muestra un ejemplo en el que se ha definido un ***subsetting***. Dicho ***subsetting*** indica que la colección del extremo ***liProyecto***, asociada a una determinada instancia de la clase ***Senior***, es un subconjunto de la colección del extremo ***proyecto***, asociada a la misma instancia de ***Senior***. En otras palabras, un empleado senior sólo puede ser líder de los proyectos en los que participa.

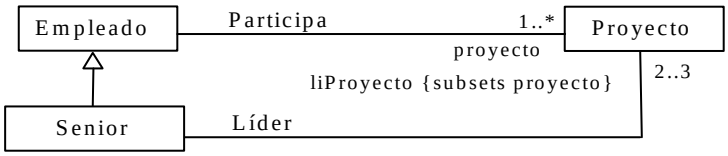



**Fig. 5.2.** Ejemplo de *subsetting*

La sintaxis abstracta del *subsetting* de asociaciones identifica los conceptos principales que describen su notación. En UML, esta sintaxis se presenta a través del metamodelo de UML. La figura 5.3 muestra el fragmento del metamodelo de UML relacionado con el *subsetting* de extremos de asociaciones.

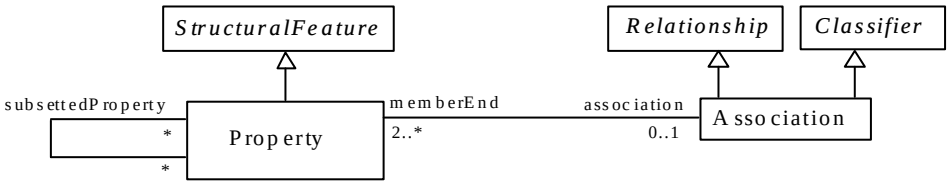



**Fig. 5.3.** Fragmento del metamodelo de UML que describe el *subsetting* de extremos de asociación

El *subsetting* de un extremo de asociación es un caso particular de *subsetting* de propiedad. Éste corresponde a los casos en los que la propiedad no tiene el rol de *association* vacío y por tanto es un extremo de asociación. En estos casos, el rol *subsettedProperty* referencia a otro extremo de asociación del que es un subconjunto.

Según las *well-formedness rules* definidas en el metamodelo de UML [OMG07], el *subsetting* de extremos de asociaciones debe garantizar que: (1) el *subsetting end* puede estar conectado a la misma clase a la que está conectado el *subsetted end* o a alguno de sus descendientes. (2) De manera similar, el extremo opuesto al *subsetting end* puede estar conectado a la misma clase a la que está conectado el extremo opuesto al *subsetted end* o alguno de sus descendientes. (3) Si se especifica la multiplicidad en los extremos *subsetting* y *subsetted*, la cardinalidad máxima del *subsetting end* debe ser menor o igual a la cardinalidad máxima del *subsetted end*. (4) Finalmente, el nombre del *subsetting end* ha de ser diferente al del *subsetted end*.

### *5.1.2 Semántica*

El efecto de definir un *subsetting* es el de establecer una restricción de inclusión entre los extremos *subsetting* y *subsetted*. Dicha restricción indica que la colección de instancias del *subsetting end*, asociada a una determinada instancia, es igual o está incluida en la colección de instancias del *subsetted end*, asociada a dicha instancia.

Para precisar la semántica del *subsetting* de asociaciones, mostramos en la figura 5.4 cómo traducir este constructor a nuestra capa básica de UML. Concretamente, se muestran los diagramas traducidos para cada uno de los cuatro escenarios descritos en la figura 5.1.

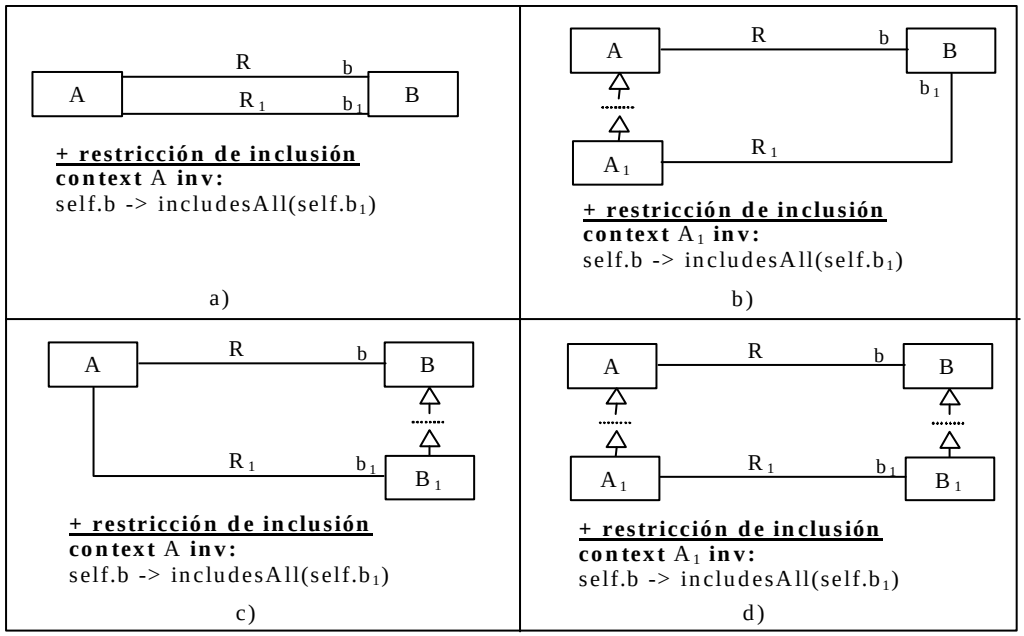



**Fig. 5.4.** Traducción del *subsetting*

Como podemos observar en la figura anterior, al realizar la traducción desaparece la indicación del ***subsetting*** (es decir, la palabra clave ***subsets***) y se añade una restricción de inclusión que define su efecto sobre el diagrama traducido. Dicha restricción puede ser expresada como un invariante de OCL, tal y como se muestra en la figura 5.4. Nótese que la asociación con el ***subsetting end*** se mantiene en el diagrama traducido. La razón de que se mantenga esta asociación, es que las instancias de ésta no pueden deducidas a partir de las instancias de la asociación con el ***subsetted end***.

La figura 5.5 muestra el diagrama traducido para el ejemplo de la figura 5.2. La restricción OCL que se añade asegura que una instancia de la clase ***Senior*** sólo puede ser líder de los proyectos en los que participa.

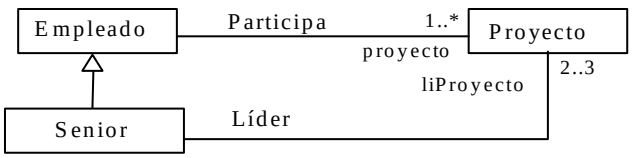



**Fig. 5.5.** Traducción del *subsetting* de la fig. 5.2

## *5.2 Comparación con la Redefinición de Asociaciones*

El propósito de esta sección es comparar el constructor del ***subsetting*** de UML con los diferentes tipos de redefinición de asociaciones que se han explicado en la sección 4. Esta comparación se realiza desde un punto se vista sintáctico y semántico.

Como resultado, veremos que existen tanto diferencias sintácticas como semánticas entre el ***subsetting*** y la redefinición de asociaciones. Estas diferencias ayudarán al diseñador de UML a decidir, dada una situación del mundo real, cuál de los dos conceptos se adapta mejor a las necesidades de esa situación.

### *5.2.1 Sintáctica*

Recordemos que según el metamodelo de UML [OMG07], la definición de la redefinición de asociaciones garantiza que, el extremo opuesto al ***redefining end*** debe estar conectado siempre a un descendiente de la clase a la que está conectado el extremo opuesto al ***redefined end***. En el caso concreto de la redefinición de tipo, además el ***redefining end*** debe estar conectado siempre a un descendiente de la clase conectada al ***redefined end***. En cualquier caso, la redefinición de asociaciones debe combinarse siempre con una relación de generalización/especialización. Por el contrario, recordemos que el metamodelo de UML no exige que al definir un ***subsetting***, deba existir una generalización/especialización en el diagrama.

En el caso de la redefinición de multiplicidad, recordemos que según el metamodelo de UML, la multiplicidad especificada para el ***redefining end*** debe ser más restrictiva que la especificada en el ***redefined end***. Esto significa que la cardinalidad mínima del ***redefining end*** debe ser mayor o igual que la cardinalidad mínima del ***redefined end*** y la cardinalidad máxima del ***redefining end*** debe ser menor o igual que la cardinalidad máxima del ***redefining end***. Para el caso del ***subsetting***, el metamodelo de UML exige que si se especifica multiplicidad en los extremos ***subsetting*** y ***subsetted***, la cardinalidad máxima del ***subsetting end*** debe ser menor o igual a la cardinalidad máxima del ***subsetted end***. No obstante, no restringe el valor de la cardinalidad mínima, pudiendo darse el caso de que esta cardinalidad del ***subsetting end***, sea menor que la cardinalidad mínima del ***subsetted end***.

### *5.2.2 Semántica*

El objetivo de esta subsección es comparar semánticamente la redefinición y el ***subsetting*** de asociaciones. Para ello, traducimos ambos conceptos a nuestra capa básica de UML. La idea es comparar los diagramas traducidos de ambos constructores y demostrar formalmente qué restricciones OCL generadas en dicha traducción, quedan garantizadas por la redefinición y el ***subsetting*** de asociaciones.

#### *5.2.2.1 Comparación de los diagramas traducidos*

Como hemos visto en la sección 4, el efecto semántico de la redefinición se aplica sobre un subconjunto de instancias de la asociación con el ***redefined end.*** Estas instancias, que hemos llamado instancias afectadas, pueden ser deducidas a partir de las instancias de la asociación con el ***redefined end***. Por este motivo, al traducir la redefinición de asociaciones a nuestra capa básica de UML, la asociación con el ***redefining end*** desaparece y en su lugar se añade una restricción OCL que define su efecto sobre el diagrama traducido. Recordemos que el objetivo de la redefinición no es definir una nueva asociación, sino mejorar la definición de una asociación existente. Por tanto, ambas asociaciones deben tener la misma semántica.

Recordemos el ejemplo 4.7 de la sección 4. En este ejemplo, redefiniendo la asociación *Participa* se quiere expresar que un empleado júnior sólo puede participar en proyectos que no sean críticos. El propósito de esta redefinición no es crear una nueva asociación sino definir de una manera más específica la asociación *Participa*.

En el caso del ***subsetting***, a diferencia de la redefinición, las asociaciones con los ***subsetting*** y ***subsetted end*** pueden ser semánticamente diferentes. Cada una de estas asociaciones tiene sus propias instancias y las instancias de la asociación con el ***subsetting end*** no se pueden deducir de las instancias de la asociación con el ***subsetted***

*end*. Por este motivo, como hemos comentado anteriormente, al traducir el ***subsetting*** a nuestra capa básica de UML, la asociación del ***subsetting end*** no se elimina en el diagrama traducido.

La figura 5.2 muestra un ejemplo en el que se ha definido un ***subsetting*** entre dos extremos de las asociaciones *Participa* y *Líder*. La primera de ellas expresa que un empleado puede participar en proyectos y la segunda que un empleado senior puede ser líder de proyectos. Ambas asociaciones son semánticamente diferentes y las instancias de la asociación *Líder* no pueden ser deducidas de las instancias de la asociación *Participa*.

Resumiendo, hemos visto que los diagramas traducidos para la redefinición y el ***subsetting*** de asociaciones son diferentes: en la redefinición, la asociación con el ***redefining end*** desaparece y en su lugar se añade una restricción OCL que define su efecto sobre el diagrama. En cambio en el ***subsetting***, la asociación con el ***subsetting end*** se mantiene en el diagrama traducido.

#### *5.2.2.2 Satisfacción de restricciones OCL*

En esta subsección, demostramos formalmente qué restricciones OCL, generadas al traducir la redefinición y el ***subsetting*** de asociaciones a nuestra capa básica de UML, quedan garantizadas por cada uno de estos constructores. La tabla 5.1 resume estas restricciones.

| | **Restricción OCL** | | | | |
|---|---|---|---|---|---|
| | **Participación** | **Cardinalidad mínima** | **Cardinalidad máxima** | **General (deducida de una regla de derivación)** | **Inclusión** |
| **Redefinición** | **P** (redefinición de tipo) | **P** (redefinición de la multiplicidad) | **P** (redefinición de la multiplicidad) | **P** (redefinición del tipo de derivación) | **P** |
| ***Subsetting*** | **O** | **P** | **O** | **O** | **P** |

**Tabla 5.1.** Restricciones OCL (generadas en la traducción) que quedan garantizadas por la redefinición y el ***subsetting*** de asociaciones

Como podemos ver en la tabla anterior, todas las restricciones OCL quedan garantizadas por la redefinición: la restricción de participación queda garantizada por la redefinición de tipo, la de cardinalidad mínima y máxima por la redefinición de multiplicidad, la restricción general (deducida de una regla de derivación) por la redefinición del tipo de derivación y finalmente, la restricción de inclusión, generada en la traducción del ***subsetting***, queda garantizada por cualquier tipo de redefinición.

En cambio, sólo dos restricciones OCL quedan garantizadas por el ***subsetting***: la restricción de inclusión y la de cardinalidad mínima, generada en la traducción de la redefinición de multiplicidad.

A continuación, demostramos formalmente a través de cinco teoremas que: (a) la restricción de participación no queda garantizada por el ***subsetting***, (b) la restricción de cardinalidad mínima queda garantizada por el ***subsetting***, (c) la restricción de cardinalidad máxima no queda garantizada por el ***subetting***, (d) la restricción general (deducida de una regla de derivación) no queda garantizada por el ***subsetting*** y (e) la restricción de inclusión queda garantizada por cualquier tipo de redefinición.

*La restricción de participación no queda garantizada por el subsetting*

Tal y como explicamos en la sección 4, la restricción de participación se genera al traducir la redefinición de tipo a nuestra capa básica de UML. Recordemos que para definir este tipo de redefinición, el ***redefining end*** debe estar conectado a una de las clases descendientes de la clase del ***redefined end***. Por este motivo, para demostrar que la restricción de participación no queda garantizada por el ***subsetting***, partimos del escenario descrito en la figura 5.6.

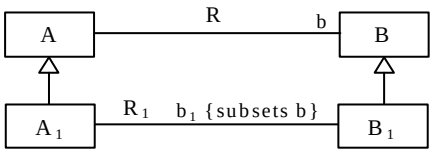



**Fig. 5.6.** La restricción de participación no queda garantizada por el ***subsetting***

**Teorema 5.1**: *La restricción de participación no queda garantizada por el subsetting*
Sean *A* y *B* dos clases y *R* una asociación binaria entre ellas. Sea *b* el extremo de la asociación que conecta *R* a la clase *B*. Sea $A_1$ una subclase de *A*, $B_1$ una subclase de *B* y $R_1$ una asociación binaria entre ellas. Sea $b_1$ el extremo que conecta $R_1$ a $B_1$. Suponemos que el extremo $b_1$ se declara como un ***subsetting*** del extremo *b*. Entonces, la restricción de participación expresada como **`context`** $A_1$ **`inv`**: `self.b->forAll(oclIsTypeOf(`$B_1$`))` no queda garantizada por el ***subsetting*** (véase figura 5.6).

**Demostración.** (1) La traducción del ***subsetting*** declarado a nuestra capa básica de UML requiere la restricción de inclusión expresada como **`context`** $A_1$ **`inv:`** `self.b->includesAll(self.`$b_1$`)`. La figura 5.4 d) muestra el esquema traducido.
(2) Suponemos una base de información (BI) donde:
(2.1) *x* e *y* son instancias de *B* y *x* es también una instancia de $B_1$.
(2.2) *z* es una instancia de *A* y $A_1$.
(2.3) *z* está ***R-relacionada*** con *x* y con *y* y *z* está ***$R_1$-relacionada*** con *x*.
(3) La BI descrita en (2) satisface todas las restricciones del diagrama traducido descrito en (1), es decir, ésta satisface la restricción textual de la inclusión junto con las restricciones gráficas definidas sobre el diagrama traducido.
(4) La BI descrita en (2) no satisface la restricción de participación porque, según esta restricción, las instancias *x* y *y* deben ser de tipo $B_1$, y por el paso (2) sólo *x* es una instancia de $B_1$.
(5) Por tanto, por (3) y (4) la restricción de participación no queda garantizada por el ***subsetting***.

Consideremos el ejemplo de la figura 5.7 que relaciona estudiantes y asignaturas. Un estudiante tiene inicialmente una serie de asignaturas que son de su interés. Una vez el estudiante ha sido aceptado y se ha decidido que asignaturas serán ofrecidas, éste puede matricularse de dichas asignaturas, siempre y cuando sean de su interés. Para este ejemplo, vemos que la restricción de participación, expresada como **`context`** `Aceptado` **`inv`**: `self.asignatura->forAll(oclIsTypeOf(Ofrecida))` no se satisface por el diagrama. Según esta restricción, todas las asignaturas que son de interés para un estudiante aceptado deben ser finalmente ofrecidas. En cambio, vemos que el diagrama no satisface esta restricción ya que según éste, un estudiante puede tener asignaturas que sean de su interés y que finalmente no sean ofrecidas.

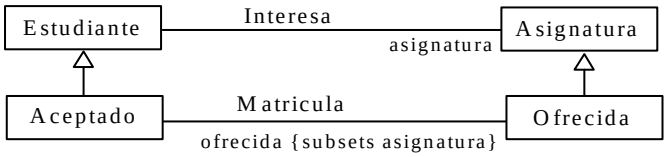
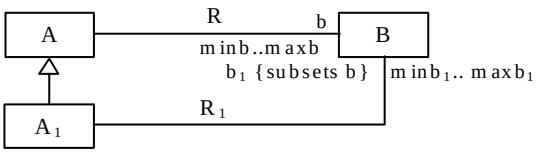



**Fig. 5.7.** Ejemplo de *subsetting* con diferentes subclases

*La restricción de cardinalidad mínima queda garantizada por el subsetting*

Recordemos que la restricción de cardinalidad mínima se genera al traducir la redefinición de multiplicidad a nuestra capa básica de UML. Para definir este tipo de redefinición, el *redefining* y el *redefined end* deben estar conectados a la misma clase (suponiendo que no se combina con una redefinición de tipo). Por este motivo, para demostrar que la restricción de cardinalidad mínima queda garantizada por el *subsetting*, consideramos el escenario descrito en la figura 5.8.




**context** $A_1$ **inv**: self.b->size() >= $minb_1$
*(garantizada)*

**Fig. 5.8.** La restricción de cardinalidad mínima queda garantizada por el *subsetting*

**Teorema 5.2**: *La restricción de cardinalidad mínima queda garantizada por el subsetting*

Sean *A* y *B* dos clases y *R* una asociación binaria entre ellas. Sea *b* el extremo que conecta *R* a la clase *B*. Sea $A_1$ una subclase de *A* y $R_1$ una asociación binaria entre $A_1$ y *B*. Sea $b_1$ el extremo que conecta $R_1$ a *B* y $minb_1$ la multiplicidad mínima para el extremo $b_1$. Suponemos que el extremo $b_1$ se declara como un *subsetting* del extremo *b*. Entonces, la restricción de la cardinalidad mínima expresada como **context** $A_1$ **inv**: self.b->size() >= $minb_1$ queda garantizada por el *subsetting* (véase figura 5.8).

**Demostración.** (1) La traducción del *subsetting* declarado a nuestra capa básica de UML requiere la restricción de inclusión expresada como **context** $A_1$ **inv:** self.b -> includesAll(self.$b_1$). La figura 5.4 b) muestra el esquema traducido.

(2) Como *self.b* contiene todos los elementos de $self.b_1$ el tamaño de *self.b* debe ser mayor o igual que el tamaño del $self.b_1$. Entonces, self.b->size() >=self.$b_1$->size().

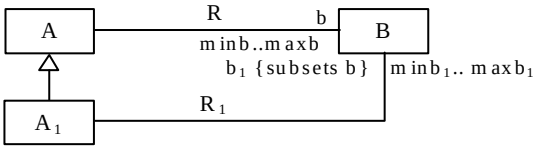
(3) Como $minb_1$ es la cardinalidad mínima del extremo $b_1$ el tamaño del $self.b_1$ debe ser mayor o igual a $minb_1$. Entonces, self.$b_1$->size() >= $minb_1$.

(4) Por tanto, por (2) y (3) la restricción de cardinalidad mínima queda garantizada por el *subsetting*.

En el ejemplo de la figura 5.2, la restricción de cardinalidad mínima se expresa como **context** Senior **inv**: self.proyecto->size() >= 2. Según esta restricción un empleado senior debe participar en como mínimo dos proyectos. Por otro lado, el diagrama indica que un empleado senior debe ser líder de como mínimo dos proyectos en los que participa, por tanto dicho empleado debe participar en como mínimo dos proyectos. Así pues, vemos claramente que el diagrama satisface dicha restricción.

*La restricción de cardinalidad máxima no queda garantizada por el subsetting*

Como vimos en la sección 4, para definir una redefinición de multiplicidad, el *redefining* y el *redefined end* deben estar conectados a la misma clase (suponiendo que no se combina con una redefinición de tipo). Por este motivo, para demostrar que la restricción de cardinalidad máxima, generada en la traducción de la redefinición de multiplicidad, no queda garantizada por el *subsetting*, consideramos el escenario descrito en la figura 5.9.




**context** $A_1$ **inv**: self.b->size() <= $maxb_1$
*(no garantizada)*

**Fig. 5.9.** La restricción de cardinalidad máxima no queda garantizada por el *subsetting*

**Teorema 5.3**: *La restricción de cardinalidad máxima no queda garantizada por el subsetting*

Sean *A* y *B* dos clases y *R* una asociación binaria entre ellas. Sea *b* el extremo que conecta *R* a la clase *B* y *maxb* la multiplicidad máxima del extremo *b*. Sea $A_1$ una subclase de *A* y $R_1$ una asociación binaria entre $A_1$ y *B*. Sea $b_1$ el extremo que conecta $R_1$ a *B* y $maxb_1$ la multiplicidad máxima del extremo $b_1$. Suponemos que el extremo $b_1$ está declarado como un *subsetting* del extremo *b*. Entonces, la restricción de cardinalidad máxima expresada como **context** $A_1$ **inv**: self.b->size() <= $maxb_1$ no queda garantizada por el *subsetting* (véase figura 5.9).

**Demostración.** (1) La traducción del *subsetting* declarado a nuestra capa básica de UML requiere la restricción de inclusión expresada como **context** $A_1$ **inv:** self.b -> includesAll(self.$b_1$). La figura 5.4 b) muestra el diagrama traducido.

(2) Suponemos una base de información (BI) donde:

(2.1) *x* e *y* son instancias de *B*.

(2.2) *z* es una instancia de *A* y $A_1$.

(2.3) maxb = 2 y $maxb_1$ = 1.

(2.4) *z* está *R-relacionada* con *x* y con *y* y *z* está $R_1$*-relacionada con x*.

(3) La BI descrita en (2) satisface todas las restricciones del diagrama traducido descrito en (1), es decir, ésta satisface la restricción textual de inclusión junto con las restricciones gráficas definidas sobre el diagrama traducido.

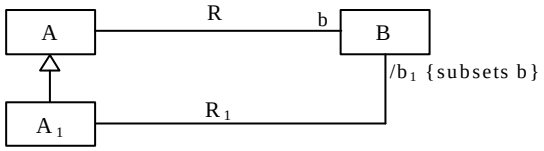
(4) La BI descrita en (2) no satisface la restricción de cardinalidad máxima porque según esta restricción, *z* debe estar *R-relacionada* como mucho con una instancia de tipo *B* y por el paso (2) *z* está *R-relacionada* con *x* y con *y*.

(5) Entonces, por (3) y (4) la restricción de cardinalidad máxima no queda garantizada por el *subsetting*.

En el ejemplo de la figura 5.2, vemos que la restricción de cardinalidad máxima, expresada como **context** Senior **inv**: self.proyecto->size() <= 3 no se satisface por el diagrama. Según esta restricción, un empleado senior puede participar en como máximo tres proyectos. No obstante, la restricción gráfica de cardinalidad máxima * indica que un empleado puede participar en más de tres proyectos, lo que contradice lo expresado por la restricción OCL.

*La restricción general (deducida de una regla de derivación) no queda garantizada por el subsetting*

Recordemos que la restricción general (deducida de una regla de derivación) se genera al traducir la redefinición del tipo de derivación a nuestra capa básica de UML. Para aplicar la redefinición del tipo de derivación, el *redefining* y el *redefined end* deben estar conectados a la misma clase (suponiendo que no se combina con una redefinición de tipo). Por tanto, para demostrar que la restricción general no queda garantizada por el *subsetting*, partimos del escenario descrito en la figura 5.10.




**context** $A_1$ :: $b_1$: Set(B)
derive: regla de derivación para $b_1$

**context** $A_1$ **inv**:
self.b = *expresión OCL de la regla de derivación para* $b_1$
*(no garantizada)*

**Fig. 5.10.** La restricción general (deducida de una regla de derivación) no queda garantizada por el *subsetting*

**Teorema 5.4**: *La restricción general (deducida de una regla de derivación) no queda garantizada por el subsetting*

Sean *A* y *B* dos clases y *R* una asociación binaria entre ellas. Sea *b* el extremo que conecta *R* a la clase *B*. Sea $A_1$ una subclase de *A* y $R_1$ una asociación binaria entre $A_1$ y *B*. Sea $b_1$ el extremo que conecta be $R_1$ a *B*. Suponemos que el extremo $b_1$ está declarado como un *subsetting* del extremo *b*. Entonces, la restricción general expresada como **context** $A_1$ **inv**: self.b = *expresión OCL de la regla de derivación de* $b_1$ no queda garantizada por el *subsetting* (véase figura 5.10).

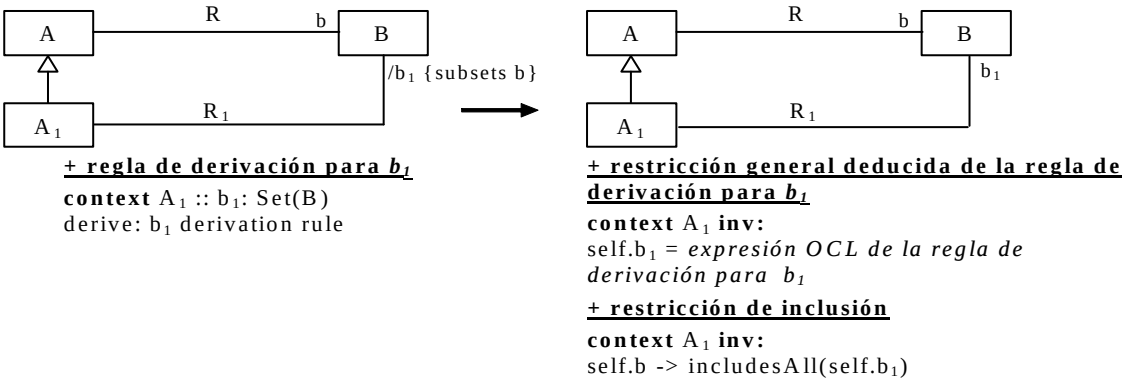



**Fig. 5.11.** Traducción del *subsetting* con el *subsetting end* derivado

**Demostración.** (1) La traducción del *subsetting* declarado a nuestra capa básica de UML requiere la restricción de inclusión expresada como **context** $A_1$ **inv:** self.b -> includesAll(self.$b_1$) y la restricción general deducida de la regla de derivación para $b_1$: **context** $A_1$ **inv**: self.$b_1$ = *expresión OCL de la regla de derivación para* $b_1$.

La figura 5.11 muestra el diagrama traducido. Nótese que en la traducción, se elimina la indicación de derivación (es decir, la /) y se añade un invariante OCL que sustituye la

regla de derivación inicial. Evidentemente, desaparece también la palabra clave ***subsets*** y se añade la restricción de inclusión.

(2) Suponemos una base de información (BI) donde:

(2.1) *x* es una instancia de *B*.

(2.2) y es una instancia de *A* y $A_1$.

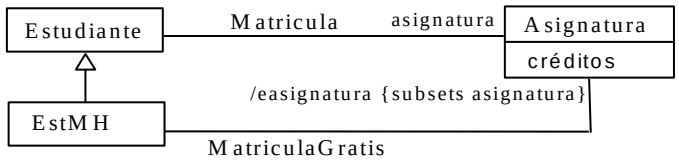
(2.3) y está ***R-relacionada*** con *x*.

(3) La BI descrita en (2) satisface todas las restricciones del diagrama traducido descrito en (1), es decir, ésta satisface la restricción textual de inclusión y la restricción deducida de la regla de derivación para $b_1$, junto con las restricciones gráficas definidas sobre el diagrama traducido.

(4) La BI descrita en (2) no satisface la restricción general expresada como **`context`** $A_1$ **`inv`**: `self.b =` *`expresión OCL de la regla de derivación de`* $b_1$ (Nótese que en la restricción general nos referimos al conjunto ***self.b*** y en la restricción deducida de la regla de derivación al conjunto ***self.*$b_1$**). Según esta restricción general, se deduce que conjunto ***self.b*** es igual a ***self.*$b_1$** ya que la regla de derivación es la misma, y por (2) ***y*** está ***R-relacionada*** con ***x*** pero no está ***$R_1$-relacionada con x***.

(5) Entonces, por (3) y (4) la restricción general expresada como **`context`** $A_1$ **`inv`**: `self.b =` *`expresión OCL de la regla de derivación de`* $b_1$ no queda garantizada por el ***subsetting***.

Consideremos el ejemplo de la figura 5.12 en el que se relacionan estudiantes y asignaturas. Además, se ha definido una regla de derivación que calcula las asignaturas, en las que un estudiante con M.H podría matricularse de forma gratuita.




**+ regla de derivación para *easignatura***

**context** EstMH :: easignatura: Set(Asignatura)
derive: self.asignatura->select(créditos>=9)

**Fig. 5.12.** Ejemplo de subsetting con el ***subsetting end*** derivado

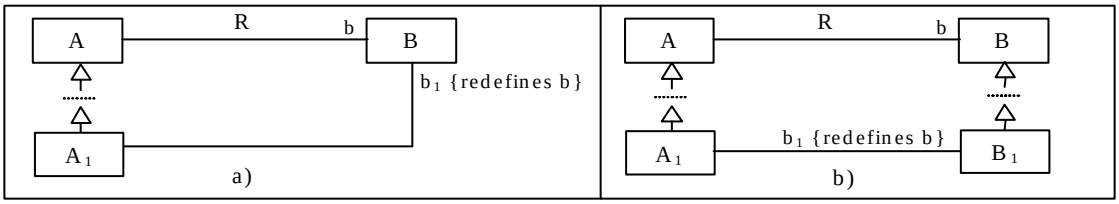
Para este ejemplo, la restricción general deducida de la regla de derivación para ***easignatura***, se expresa como **`context`** `EstMH` **`inv`**: `self.asignatura = self.asignatura -> select (créditos>=9)`. Nótese que la parte derecha del '=' corresponde a la regla de derivación para ***easignatura***.

Como podemos ver, esta restricción general no queda garantizada por el diagrama. Según esta restricción, un estudiante con M.H solo podría matricularse de asignaturas que tengan nueve o más créditos. Si nos fijamos en la regla de derivación definida, vemos que estas asignaturas corresponderían a las asignaturas en las que un estudiante con M.H puede matricularse gratis.

No obstante, el diagrama de la figura no impide que un estudiante con M.H pueda matricularse de asignaturas con menos de nueve créditos. Por tanto, vemos que la restricción general expresada como **`context`** `EstMH` **`inv`**: `self.asignatura = self.asignatura –> select (créditos>=9)` no queda garantizada por el diagrama.

*La restricción de inclusión queda garantizada por cualquier tipo de redefinición*

Como vimos en la sección 4, para redefinir una asociación, el extremo opuesto al *redefining end* debe estar siempre conectado a un descendiente de la clase, a la que está conectado el extremo opuesto al *redefined end*. Dependiendo del tipo de redefinición, se pueden dar dos casos: el *redefining end* puede estar conectado a (1) la misma clase del *redefined end* o (2) a uno de sus descendientes. Por tanto, para demostrar que la restricción de inclusión queda garantizada por cualquier tipo de redefinición, consideramos los dos escenarios descritos en la figura 5.13. El escenario a) corresponde al caso (1) y el escenario b) al caso (2).




**context** $A_1$ **inv**: self.b->includesAll(self.$b_1$)
*(garantizada)*

**Fig. 5.13.** La restricción de inclusión queda garantizada por cualquier tipo de redefinición

**Teorema 5.5**: *La restricción de inclusión queda garantizada por cualquier tipo de redefinición (es decir, redefinición de nombre, redefinición de tipo, redefinición de multiplicidad y la redefinición del tipo de derivación)*

Sean *A* y *B* dos clases, *R* una asociación binaria entre ellas y $A_1$ una subclase de *A*. Sea *b* el extremo de la asociación que conecta *R* con la clase *B*. Suponemos que $b_1$ es una redefinición del extremo *b* y que su extremo opuesto está conectado a $A_1$. Entonces, la restricción de inclusión expresada como **context** $A_1$ **inv**: self.b->includesAll(self.$b_1$) queda garantizada por la redefinición (véase figura 5.13).

**Demostración.** (1) Distinguimos dos casos:

(1.1) el extremo $b_1$ también redefine el nombre del extremo *b*. Entonces, la traducción de la redefinición a nuestra capa básica de UML sustituye $b_1$ por *b* en las expresiones OCL y la restricción de inclusión se traduce como **context** $A_1$ **inv**: self.b->includesAll(self.b).

(1.2) el extremo $b_1$ no redefine el nombre, por tanto el nombre $b_1$ es igual a *b*. Entonces, la restricción de inclusión puede también ser formulada como **context** $A_1$ **inv**: self.b->includesAll(self.b).

(2) Como *self.b* contiene todos los elementos de *self.b* la restricción de inclusión se satisface para la redefinición en ambos casos.

Recordemos el diagrama del ejemplo 4.16 de la sección 4, en el que se definen dos redefiniciones de tipo. En la figura 4.17 se muestra su diagrama traducido, junto a las restricciones de participación generadas en dicha traducción. Según estas restricciones, todas las personas que viajen con un billete infantil deben ser niños y todos los billetes con los que viaja un niño deben ser billetes infantiles.

Para este ejemplo, las restricciones de inclusión se expresarían de la siguiente manera:

```
context Infantil inv:
self.viajero->includesAll(self.nviajero)
```

```
context Niño inv:
self.billete->includesAll(self.ibillete)
```

Según estas dos restricciones de inclusión, todos los niños que viajan con billetes infantiles deben estar incluidos en las personas que viajan con dichos billetes y todos los billetes infantiles con los que viaja un niño deben estar incluidos en los billetes con los que éste viaja. Partiendo de esto y de las restricciones de participación generadas en la traducción, vemos claramente que las dos restricciones de inclusión quedan garantizadas por el diagrama.

## 6. Refinamiento de Asociaciones: Comparación con la Redefinición de Asociaciones

En las secciones anteriores, se ha explicado la sintaxis y semántica de la redefinición y el *subsetting* de asociaciones, dos constructores que ofrece UML y que nos permiten definir restricciones de integridad, tales como restricciones de cardinalidad o de inclusión.

El refinamiento de asociaciones es un concepto utilizado en algunos lenguajes de modelado (como por ejemplo, Syntropy [CD94]) que, a diferencia de la redefinición o el *subsetting* de asociaciones, no ofrece el UML. A continuación, se explica la semántica de este concepto a través de su traducción a nuestra capa básica de UML y se compara con la redefinición de asociaciones.

### *6.1 Refinamiento de una asociación*

El refinamiento de una asociación binaria ***R***, nos permite definir restricciones de integridad adicionales, cuando alguna de las instancias de las clases que participan en ***R***, es a su vez instancia de otras clases [Oli07].

Existen diferentes tipos de refinamiento, pero nosotros nos centraremos en los dos casos más comunes: el refinamiento de participantes y el refinamiento de cardinalidad.

En las siguientes subsecciones se define de manera precisa la semántica de cada uno de estos tipos de refinamiento, así como la notación (o sintaxis concreta) utilizada para su especificación. Dicha notación es la que se propone en [COT01].

Como hemos comentado anteriormente, el refinamiento de asociaciones es un concepto que no ofrece el UML. Así pues, al no disponer del metamodelo de UML, no se explica la sintaxis abstracta para este concepto.

#### *6.1.1 Refinamiento de Participantes*

La definición del refinamiento de participantes sobre una asociación permite restringir las posibles clases que pueden participar en dicha asociación. A continuación, se muestra la notación y semántica para este tipo de refinamiento.

##### *6.1.1.1 Notación*

La notación utilizada para definir el refinamiento de participantes es la que se muestra en la figura 6.1.

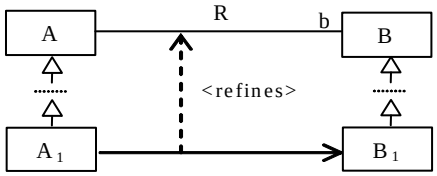



**Fig. 6.1**. Notación del refinamiento de participantes

La palabra clave *refines* junto a una flecha discontinua que apunta a ***R*** y la flecha que une las clases $A_1$ y $B_1$, indica que se ha definido un refinamiento de participantes sobre la asociación ***R***. Nótese que se ha especificado una dirección en dicho refinamiento utilizando una flecha que une las clases $A_1$ (clase origen) y $B_1$ (clase destino). Esta

dirección indica que se restringe la participación de la clase $A_1$ en la asociación *R*, pero no la participación de la clase *B*. Si no se establece la dirección del refinamiento no es posible saber sobre qué clase se aplica la restricción de participación.

Es importante destacar que la flecha que une las clases $A_1$ y $B_1$ no se trata de una nueva asociación, sino la indicación de que se ha realizado un refinamiento de la asociación *R*.

La figura 6.2 muestra un ejemplo de refinamiento de participantes sobre la asociación *Participa*. Dicho refinamiento restringe la participación de la clase *Júnior* en dicha asociación, es decir, indica que los empleados júnior solo pueden participar en proyectos que no sean críticos.

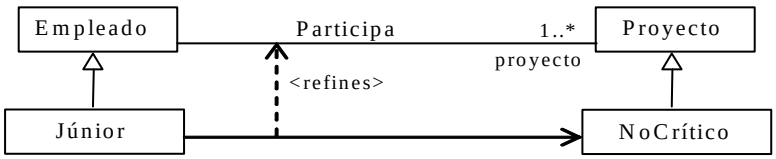



**Fig. 6.2.** Ejemplo de refinamiento de participantes

En los ejemplos anteriores, hemos visto el refinamiento de participantes en el caso que existe una única clase origen y destino. No obstante, este tipo de refinamiento también se puede definir en el caso que tengamos un conjunto de clases origen y un conjunto de clases destino. La figura 6.3 muestra la notación para este caso.

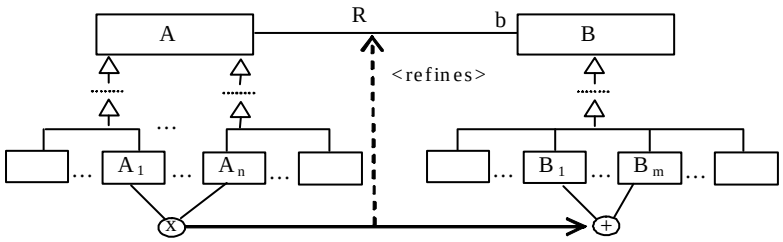



**Fig. 6.3.** Notación del refinamiento de participantes con un conjunto de clases origen y destino

Nótese que una instancia de la clase *A* puede ser al mismo tiempo instancia de $A_1,...,A_n$, subclases que pertenecen o no a generalizaciones/especializaciones diferentes.

De manera similar, una instancia de la clase *B* puede ser al mismo tiempo instancia de $B_1,\ldots,B_m$, subclases que en este caso, suponemos que pertenecen siempre a la misma generalización/especialización.

La notación gráfica de la figura 6.3 indica que se ha definido un refinamiento sobre la asociación *R*. El significado de este refinamiento es que, dada una instancia de la asociación *R*, que conecta dos instancias de las clases *A* y *B*, si la instancia de *A* es a su vez instancia de todas las subclases $A_1,\ldots,A_n$, entonces la instancia de *B* debe ser instancia de alguna de las subclases $B_1,...,B_m$.

Por ejemplo, el refinamiento definido en la figura 6.4 indica que los empleados juniors que tienen una posición temporal, sólo pueden participar en proyectos de corta o media duración.

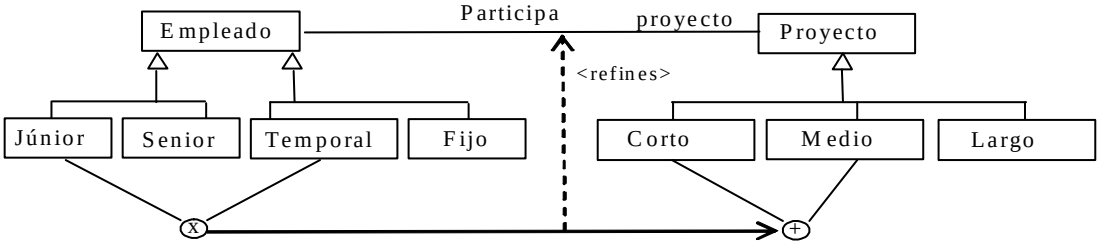
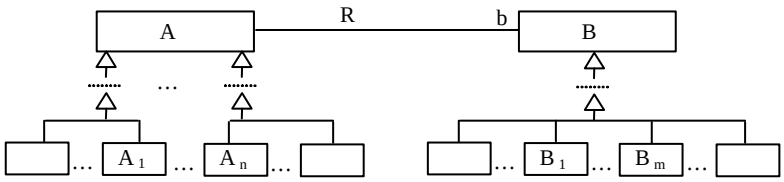



**Fig. 6.4.** Ejemplo de refinamiento de participantes con un conjunto de clases origen y destino

### *6.1.1.2 Semántica*

El efecto del refinamiento de participantes es establecer una restricción de participación adicional sobre la asociación refinada.

Para precisar la semántica de este tipo de refinamiento, mostramos en la figura 6.5 cómo es su traducción a nuestra capa básica de UML. Se toma como diagrama original el de la figura 6.3.

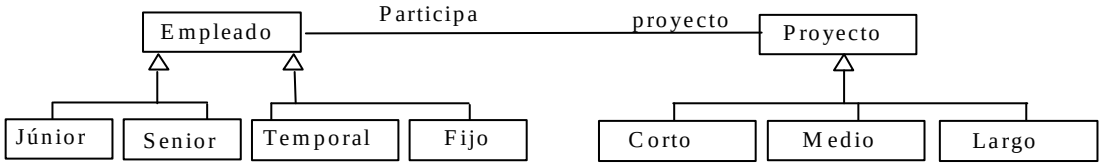



**context** A **inv:**
self.oclIsTypeOf($A_1$) and ... and self.oclIsTypeOf($A_n$) **implies**
self.b->forAll(b | b.oclIsTypeOf($B_1$) or ... or b.oclIsTypeOf($B_m$))

**Fig. 6.5.** Traducción del refinamiento de participantes con un conjunto de clases origen y destino de la fig. 6.3

Como podemos observar, en la traducción se elimina el refinamiento de la asociación ***R*** y en su lugar se añade una restricción que define su efecto sobre el diagrama traducido. Dicha restricción puede expresarse como un invariante de OCL, tal y como se indica en la figura 6.5. Cabe decir que esta restricción se obtiene al traducir al OCL, la expresión lógica definida en [COT01] para el refinamiento de participantes.

La figura 6.6 muestra el diagrama traducido para la figura 6.4.




**context** Empleado **inv:**
self.oclIsTypeOf(Junior) and self.oclIsTypeOf(Temporal) **implies**
self.b->forAll(b | b.oclIsTypeOf(Corto) or b.oclIsTypeOf(Medio))

**Fig. 6.6.** Traducción del refinamiento de participantes de la fig. 6.4

La restricción OCL generada garantiza que los empleados juniors que tienen una posición temporal, sólo pueden participar en proyectos de corta y media duración.

Se definen dos casos particulares de refinamiento de participantes, en función de si alguno de los conjuntos de las clases origen y destino está vacío.

Para explicar cada uno de ellos, consideramos el escenario descrito en la figura 6.3. Suponemos además, que existe una instancia de la asociación ***R*** que conecta dos instancias ***a*** y ***b*** de las clases ***A*** y ***B***. Partiendo de esta situación, los dos casos particulares son:

- El conjunto de clases origen $A_1,...,A_n$ está vacío. En este caso, se ha de cumplir que la instancia ***b*** es a su vez instancia de alguna de las subclases $B_1,...,B_m$. La traducción de este caso a nuestra capa básica de UML, genera la siguiente restricción OCL:

  ```
  context A inv:
  self.b->forAll(b|b.oclIsTypeOf(B1) or ... or b.oclIsTypeOf(Bm))
  ```

- El conjunto de clases destino $B_1,...,B_m$ está vacío. Para este caso, se debe garantizar que la instancia ***a*** no sea instancia simultáneamente de diferentes subclases $A_1,...,A_n$. Al traducir este caso a nuestra capa básica de UML, se genera la siguiente restricción OCL:

  ```
  context A inv:
  self.oclIsTypeOf(A1) and ... and self.oclIsTypeOf(An) implies
  self.b->isEmpty()
  ```

### *6.1.2 Refinamiento de Restricciones de Cardinalidad*

La definición del refinamiento de restricciones de cardinalidad de una asociación permite definir restricciones de cardinalidad adicionales sobre dicha asociación. A continuación, se muestra la notación y semántica para este tipo de refinamiento.

#### *6.1.2.1 Notación*

La notación utilizada para definir el refinamiento de restricciones de cardinalidad es la que se muestra en la figura 6.7. Concretamente, se muestran todos los posibles escenarios que se pueden dar a la hora de definir este tipo de refinamiento.

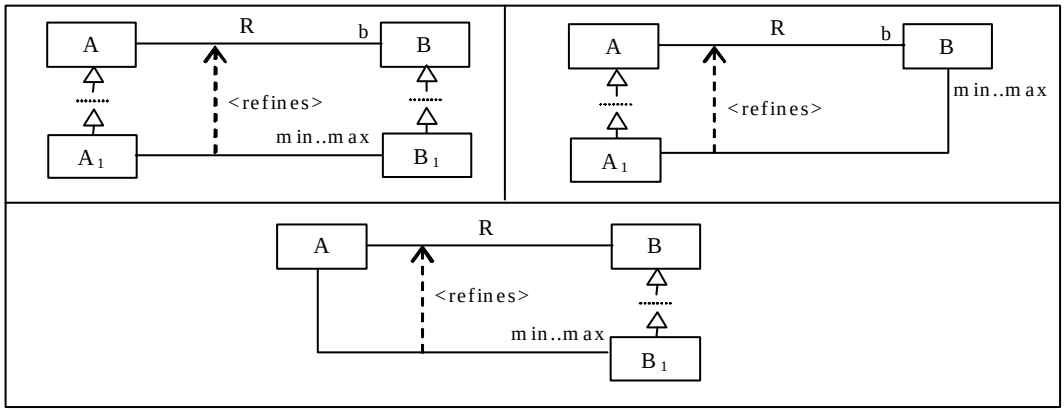



**Fig. 6.7.** Notación del refinamiento de cardinalidad

La palabra clave *refines* junto a una flecha discontinua que apunta a ***R*** y las cardinalidades *min..max*, indica que se ha definido un refinamiento de restricciones de cardinalidad sobre la asociación ***R***. El efecto de este refinamiento es restringir el número mínimo y máximo (*min..max*) de instancias de la subclase $B_1$ con las que puede estar asociada una instancia de la subclase $A_1$. Llamamos a $A_1$ clase origen y $B_1$ clase destino. Es importante destacar que las cardinalidades *min..max* deben ser más restrictivas que las cardinalidades de la asociación que se refina.

Nótese que a diferencia del refinamiento de participantes, no es necesario definir la dirección del refinamiento, ya que la situación en el diagrama de las cardinalidades *min..max*, ya nos indica la clase sobre la que se aplica la restricción de cardinalidad.

Es importante destacar que, de la misma manera que en el refinamiento de participantes, la línea que conecta las subclases $A_1$ y $B_1$ no se trata de una nueva asociación sino que indica el refinamiento de la asociación *R*.

La figura 6.8 muestra un ejemplo de refinamiento de restricciones de cardinalidad. El refinamiento de la asociación *Participa* restringe el número de proyectos no críticos en los que puede participar un empleado júnior. Las cardinalidades *1..3* indican que un empleado júnior puede participar en como mínimo 1 y como máximo 3 proyectos no críticos, pudiendo participar en otro tipo de proyectos.

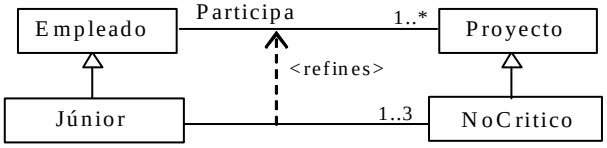



**Fig. 6.8.** Ejemplo de refinamiento de cardinalidad

En los ejemplos anteriores, hemos visto el refinamiento de restricciones de cardinalidad para el caso en el que la restricción de cardinalidad se aplica sobre una única clase origen. No obstante, es posible definir también este tipo de refinamiento en el caso que tengamos un conjunto de clases origen. La figura 6.9 muestra un ejemplo de notación para este caso. Nótese que, a diferencia del refinamiento de participantes, sólo puede existir una clase destino.

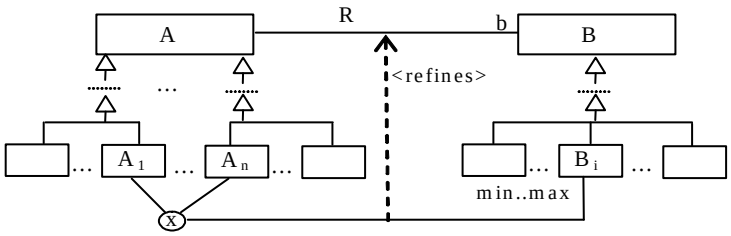



**Fig. 6.9.** Ejemplo de notación del refinamiento de cardinalidad con un conjunto de clases origen

La notación gráfica de la figura 6.9 indica que dada una instancia de la clase *A* que es a su vez instancia de las subclases $A_1,\ldots,A_n$, dicha instancia puede estar asociada a como mínimo *min* y como máximo *max* instancias de la clase *B*.

Hemos definido dos casos particulares de refinamiento de restricciones de cardinalidad. Para explicar cada uno de ellos, nos ayudaremos del escenario descrito en la figura 6.9.

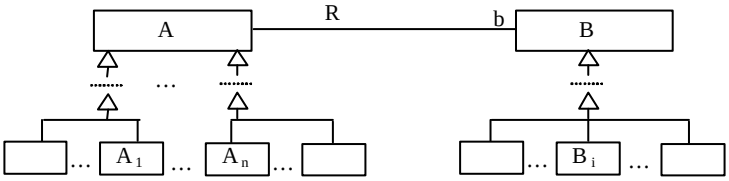
El primero de ellos, se da cuando la clase origen es la clase *A*. En este caso, el refinamiento de *R* restringe el número mínimo *min* y máximo *max* de instancias de $B_i$ con las que puede estar asociada una instancia de la clase *A*.

El segundo de los casos se da cuando la clase destino es la clase *B*. En este caso, el refinamiento de *R* restringe el número mínimo *min* y máximo *max* de instancias de *B* con las que puede estar asociada una instancia de la clase *A*, que es a su vez instancia de las subclases $A_1,\ldots,A_n$.

### *6.1.2.2 Semántica*

El efecto del refinamiento de restricciones de cardinalidad es establecer una restricción de cardinalidad adicional sobre la asociación refinada.

Para precisar la semántica de este tipo de refinamiento mostramos a continuación, cómo sería su traducción a nuestra capa básica de UML. La figura 6.10 muestra el diagrama traducido, tomando como diagrama original el de la figura 6.9.




en el caso que min>0

**context** A **inv:**
self.oclIsTypeOf($A_1$) and ... and self.oclIsTypeOf($A_n$) **implies**
self.b->select (b | b.oclIsTypeOf($B_i$)) -> size()>=min

en el caso que max no es *

**context** A **inv**:
self.oclIsTypeOf($A_1$) and ... and self.oclIsTypeOf($A_n$) **implies**
self.b->select (b | b.oclIsTypeOf($B_i$)) -> size()<=max

**Fig. 6.10.** Traducción del refinamiento de cardinalidad con un conjunto de clases origen

Como podemos observar, en la traducción desaparece el refinamiento de la asociación y en su lugar se añaden dos restricciones que definen su efecto sobre el diagrama traducido. Dichas restricciones pueden expresarse como invariantes de OCL, tal y como se indica en la figura 6.10. Cabe decir que estas restricciones se obtienen al traducir al OCL, la expresión lógica definida en [COT01] para el refinamiento de restricciones de cardinalidad.

La figura 6.11 muestra un ejemplo de este tipo de refinamiento.

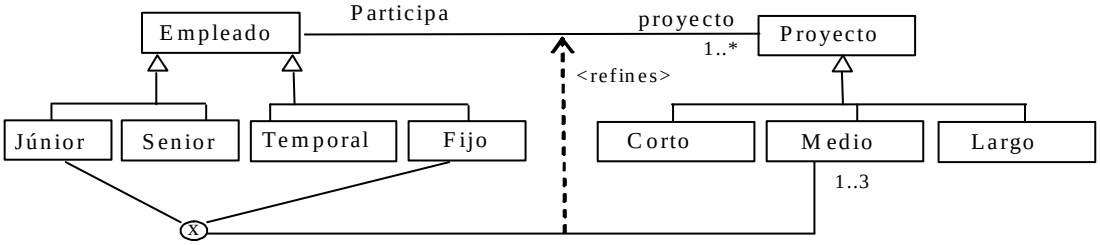
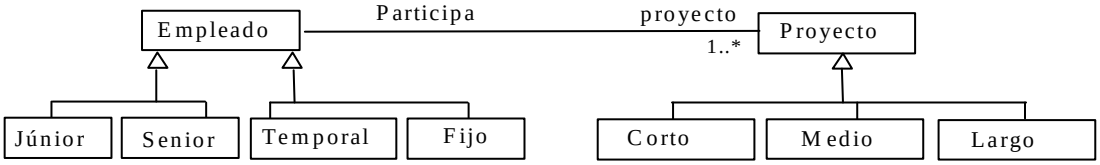



**Fig. 6.11.** Ejemplo de refinamiento de cardinalidad con un conjunto de clases origen

El refinamiento de la asociación *Participa* indica que los empleados juniors que tienen una posición fija, deben participar en como mínimo uno y como máximo en tres proyectos de duración media, pudiendo participar también en otros tipos de proyecto. La figura 6.12 muestra la traducción del diagrama para la figura 6.11.




en el caso que min>0

**context** Empleado **inv:**
self.oclIsTypeOf(Junior) and self.oclIsTypeOf(Fijo) **implies**
self.proyecto->select (b | b.oclIsTypeOf(Medio)) -> size()>=1

en el caso que max no es *

**context** Empleado **inv:**
self.oclIsTypeOf(Junior) and self.oclIsTypeOf(Fijo) **implies**
self.proyecto->select (b | b.oclIsTypeOf(Medio)) -> size()<=3

**Fig. 6.12.** Traducción del refinamiento de cardinalidad de la fig. 6.11

Como hemos comentado anteriormente, existen dos casos particulares de refinamiento de restricciones de cardinalidad. Recordemos que el primero de ellos se da cuando la clase origen es la clase *A* y el segundo cuando la clase destino es la clase *B*

(véase la figura 6.3). Las restricciones OCL para ambos casos se expresan de la siguiente manera:

- Clase origen *A*:

  *en el caso que min>0:*

  ```
  context A inv:
  self.b->select (b | b.oclIsTypeOf(Bi)) -> size()>=min
  ```

  *en el caso que max no es *:*

  ```
  context A inv:
  self.b->select (b | b.oclIsTypeOf(Bi)) -> size()<=max
  ```

- Clase destino *B*:

  *en el caso que min>0:*

  ```
  context A inv:
  self.oclIsTypeOf(A1) and ... and self.oclIsTypeOf(An) implies
  self.b->size()>=min
  ```

  *en el caso que max no es *:*

  ```
  context A inv:
  self.oclIsTypeOf(A1) and ... and self.oclIsTypeOf(An) implies
  self.b->size()<=max
  ```

## *6.2 Comparación con la Redefinición de Asociaciones*

El refinamiento y la redefinición de asociaciones son dos conceptos que están estrechamente ligados. El refinamiento de participantes es similar a la redefinición de tipo y el refinamiento de cardinalidad similar a la redefinición de multiplicidad. No obstante, veremos que existen casos de refinamiento de participantes y de cardinalidad que no pueden ser expresados a través de la redefinición de asociaciones.

El objetivo de esta sección es realizar una comparación entre los diferentes tipos de refinamiento y la redefinición de asociaciones. Concretamente, analizamos los casos que pueden ser expresados por cada uno de los tipos de refinamiento y vemos cuáles de estos casos quedan cubiertos (y cuáles no) por la redefinición de asociaciones.

### *6.2.1 Refinamiento de Participantes*

Recordemos que el refinamiento de participantes se define a partir de un conjunto de clases origen y un conjunto de clases destino. Estos conjuntos pueden contener una única clase o incluso estar vacíos. Si combinamos estas posibilidades, obtenemos todos los casos que se pueden expresar con la redefinición de participantes. La tabla 6.1 muestra cuáles de estos casos pueden ser expresados también a través de la redefinición de asociaciones. Nótese que para referirse al conjunto de clases origen y clase destino se han utilizado los nombres que se muestran en la figura 6.3.

| | | **Clase(s) origen** | | |
|---|---|---|---|---|
| | | **Una ($A_i$)** | **Más de una ($A_1$,…, $A_n$)** | **Ninguna** |
| **Clase(s) destino** | **Una ($B_i$)** | **P** (redefinición de tipo) | **O** | - |
| | **Más de una ($B_1$,…, $B_m$)** | **O** | **O** | **O** |
| | **Ninguna** | **O** | **O** | - |

**Tabla 6.1.** Casos del refinamiento de participantes cubiertos por la redefinición

Tal y como comentamos en la sección 4, al definir una redefinición, el extremo opuesto al *redefining end*, debe estar siempre conectado a una clase que sea descendiente de la clase conectada al extremo opuesto del *redefined end*. Además, en el caso concreto de la redefinición de tipo, el *redefining end* debe estar conectado a un descendiente de la clase conectada al *redefined end*. Por tanto, vemos que la redefinición de asociaciones se ha de definir siempre entre dos únicas clases.

De esta manera, tal y como se muestra en la tabla 6.1, sólo el caso del refinamiento en el que tenemos una única clase origen y destino puede ser expresado también a través de la redefinición de tipo. Es decir, para este caso, el refinamiento de participantes es semánticamente equivalente a la redefinición de tipo.

El caso en el que no existe ninguna clase origen y existe una única clase destino, la restricción de participación se puede expresar sin necesidad de utilizar el refinamiento o la redefinición de asociaciones. Finalmente, el caso en el que no existen clases origen ni clases destino, evidentemente no tendría sentido.

A continuación, demostramos de manera intuitiva que, para el caso en el que existe una única clase origen y una destino, el refinamiento de participantes y la redefinición son semánticamente equivalentes.

*Refinamiento de Participantes versus Redefinición de Tipo*

Recordemos que para definir el refinamiento de participantes, tanto la clase origen como la clase destino deben ser subclases de las clases conectadas a la asociación que se refina. Por este motivo, para demostrar que el refinamiento de participantes y la redefinición de asociaciones son semánticamente equivalentes consideramos el escenario descrito en la figura 6.13.

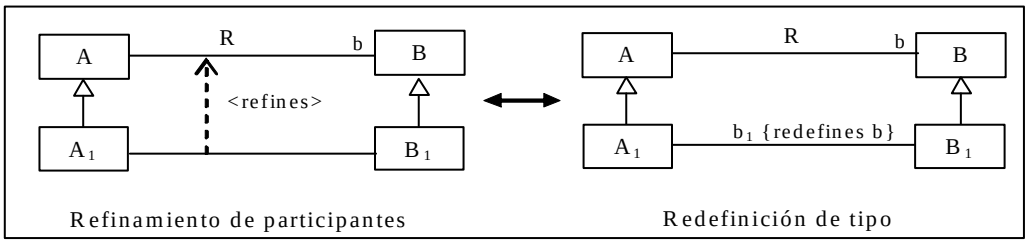



**Fig. 6.13.** Refinamiento de participantes versus redefinición de tipo

Consideremos el ejemplo 6.2, en el que se ha definido un refinamiento de participantes sobre la asociación *Participa*. Si en lugar de este refinamiento, se define una redefinición de tipo sobre *Participa*, el efecto es el mismo, es decir, indicar que los empleados juniors sólo pueden participar en proyectos no críticos.

Para demostrar que el refinamiento de participantes y la redefinición de tipo son semánticamente equivalentes, para el caso descrito en la figura 6.13, traducimos ambos conceptos a nuestra capa básica de UML. La idea es comparar los diagramas traducidos, junto a las restricciones OCL generadas en dicha traducción. La figura 6.14 muestra los diagramas traducidos junto a las restricciones OCL para la figura 6.13.

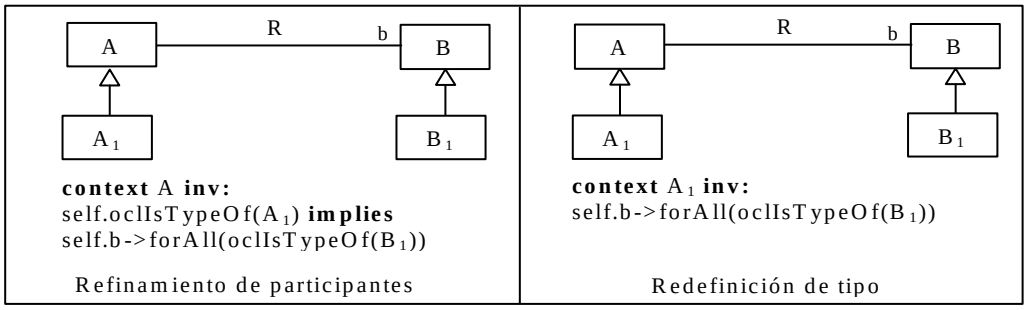



**Fig. 6.14.** Traducción del refinamiento de participantes y la redefinición de tipo de la fig. 6.13

Como podemos observar, en ambos casos, el diagrama traducido es el mismo y las restricciones OCL son equivalentes. Tanto en el refinamiento de participantes como en la redefinición de tipo, las restricciones OCL expresan que una instancia de la subclase $A_1$ sólo puede estar asociada con instancias de $B_1$. Por tanto, podemos afirmar que ambos conceptos son semánticamente equivalentes para el caso descrito en la figura 6.13.

### *6.2.2 Refinamiento de Restricciones de Cardinalidad*

Como hemos visto anteriormente, el refinamiento de restricciones de cardinalidad se define a partir de un conjunto de clases origen y una clase destino.

Recordemos que en el caso de tener una única clase origen, dicha clase puede ser la clase conectada a la asociación sobre la que se define el refinamiento o alguno(s) de sus descendientes. De manera similar, la clase destino puede ser la clase conectada a la asociación sobre la que se define el refinamiento o un descendiente de ésta.

Si combinamos estas posibilidades, obtenemos todos los casos que se pueden expresar con el refinamiento de restricciones de cardinalidad. La tabla 6.2 muestra cuáles de estos casos pueden ser expresados también con la redefinición de asociaciones. Nótese que para referirse al conjunto de clases origen y clase destino se han utilizado los nombres que se muestran en la figura 6.9.

| | | **Clase(s) origen** | | |
|---|---|---|---|---|
| | | **Una ($A_i$)** | **Más de una ($A_1$,…, $A_m$)** | **Clase asociada (A)** |
| **Clase destino** | **Clase asociada (B)** | **P** (redefinición de multiplicidad) | **O** | - |
| | **Subclase ($B_i$)** | **O** | **O** | **O** |

**Tabla 6.2.** Casos del refinamiento de restricciones de cardinalidad cubiertos por la redefinición

Recordemos que según el metamodelo de UML, al definir una redefinición, el extremo opuesto al *redefining end*, debe estar siempre conectado a una clase que sea descendiente de la clase conectada al extremo opuesto del *redefined end.* Por este motivo, los casos que se muestran en la tabla 6.2, en los que existe más de una clase

origen o la clase origen es *A*, no pueden ser expresados a través de la redefinición de asociaciones. El caso en el que la clase origen es *A* y la clase destino la *B*, se puede prescindir del refinamiento y utilizar una restricción de multiplicidad sobre la asociación que une ambas clases.

A continuación, analizamos los dos casos que faltan: (1) la clase origen es $A_i$ y la clase *B*, (2) la clase origen es también $A_i$ pero la clase destino es la subclase $B_i$.

Concretamente, demostramos de manera intuitiva, que para el caso (1), el refinamiento de restricciones de cardinalidad es semánticamente equivalente a la redefinición de multiplicidad y que el caso (2), no puede ser expresado por la redefinición, es decir, el refinamiento de restricciones de cardinalidad y la redefinición no son semánticamente equivalentes.

### *Refinamiento de Restricciones de Cardinalidad versus Redefinición de Multiplicidad*

A continuación, demostramos que el refinamiento de cardinalidad y la redefinición de multiplicidad son semánticamente equivalentes, para el caso de la tabla 6.2 en el que la clase origen es $A_i$ y la clase destino es *B*. Para ello, consideramos el escenario descrito en la figura 6.15.

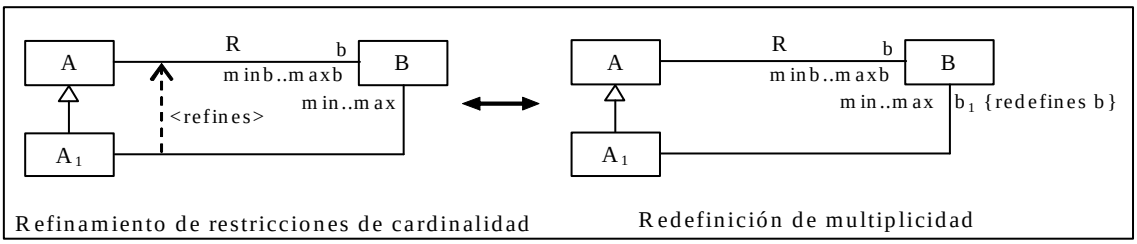



**Fig. 6.15.** Refinamiento de restricciones de cardinalidad versus la redefinición de multiplicidad

La idea es comparar los diagramas traducidos de ambos conceptos a nuestra capa básica de UML, junto a las restricciones OCL generadas en dicha traducción. La figura 6.16 muestra los diagramas traducidos y las restricciones OCL para la figura 6.15.

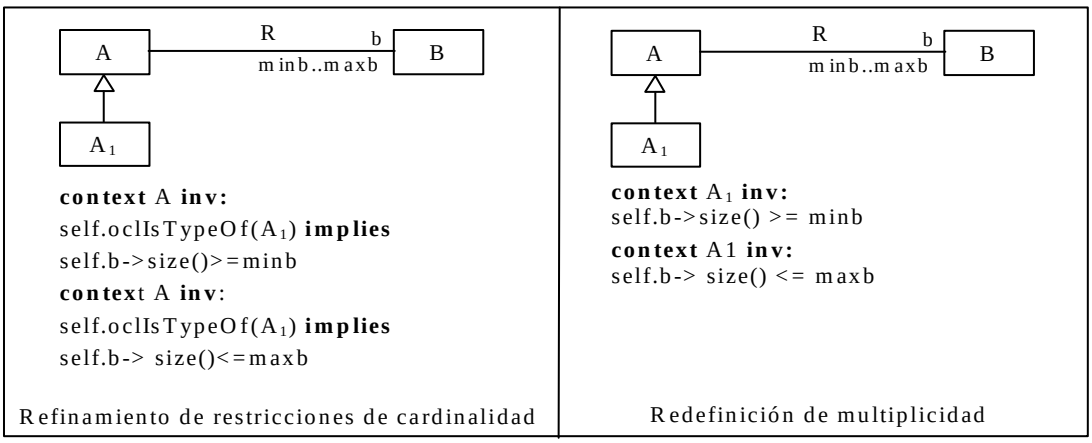



**Fig. 6.16.** Traducción del refinamiento de restricciones cardinalidad y la redefinición de multiplicidad de la fig. 6.15

Como podemos ver en la figura anterior, los diagramas traducidos son iguales y las restricciones OCL equivalentes. En ambos casos, las restricciones expresan que una instancia de la subclase $A_1$ debe estar asociada como mínimo a *min* y como máximo a *max* instancias de la clase *B*. Por tanto, podemos afirmar que el refinamiento de restricciones de cardinalidad y la redefinición de multiplicidad son semánticamente equivalentes, para el caso descrito en la figura 6.15.

### *El Refinamiento de Restricciones de Cardinalidad versus Redefinición de Tipo y Multiplicidad*

A continuación, demostramos que el refinamiento de cardinalidad y la redefinición de tipo y multiplicidad no son semánticamente equivalentes, para el caso de la tabla 6.2 en el que la clase origen es $A_i$ y la clase destino es $B_i$. Para ello, consideramos el escenario descrito en la figura 6.17.

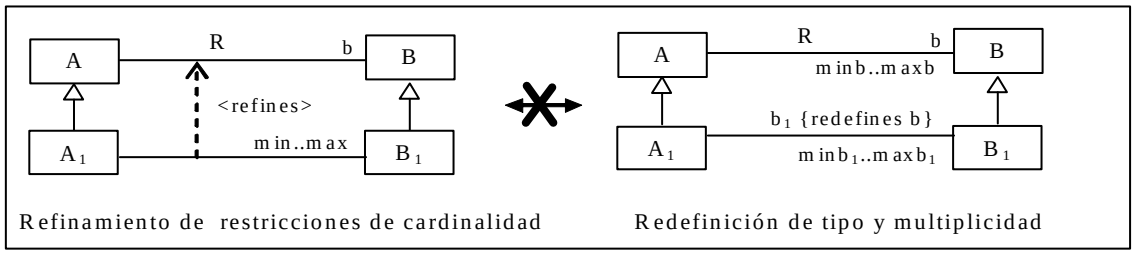



**Fig. 6.17.** Refinamiento de cardinalidad versus redefinición de tipo y multiplicidad

Para demostrar que ambos conceptos no son semánticamente equivalentes, comparamos los diagramas traducidos de ambos conceptos y las restricciones OCL generadas en dicha traducción. La figura 6.18 muestra los diagramas traducidos junto a las restricciones OCL para la figura 6.17.

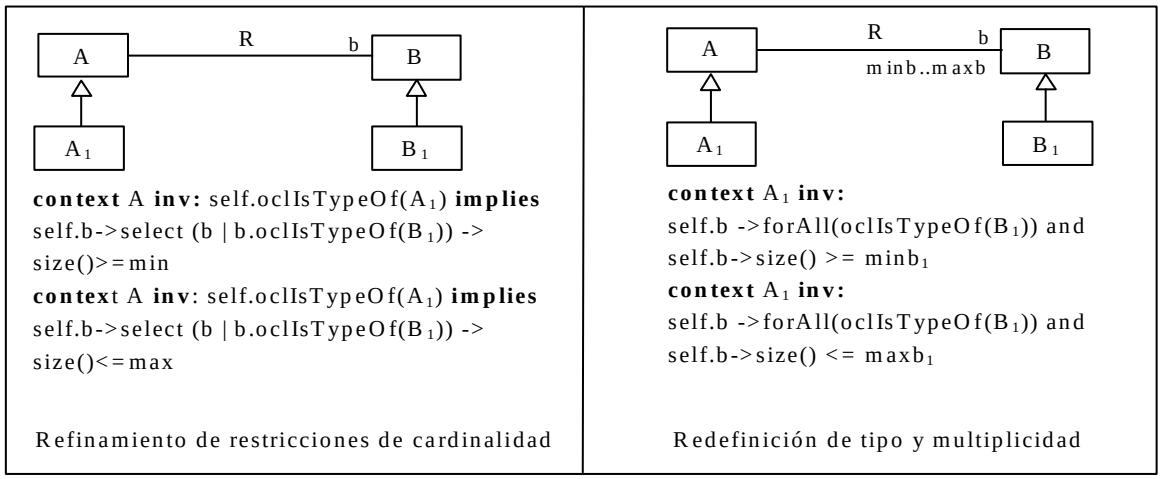



**Fig. 6.18.** Traducción del refinamiento de restricciones de cardinalidad y la redefinición de tipo y multiplicidad de la fig. 6.17

Como podemos observar, los diagramas traducidos son iguales. No obstante, vemos que no ocurre lo mismo con las restricciones OCL.

En el caso de la redefinición, las restricciones expresan que una instancia de la subclase $A_1$ puede estar asociada a como mínimo *minb* y como máximo *maxb* instancias de la clase $B_1$, no pudiendo estar asociada a ninguna instancia de otra subclase de *B*.

En el caso del refinamiento, sí que es posible que una instancia de $A_1$ pueda estar asociada también a instancias de otras subclases de *B*.

Por tanto, podemos afirmar que para el caso descrito en la figura 6.17, el refinamiento de restricciones de cardinalidad no es semánticamente equivalente a la redefinición de tipo y multiplicidad.

La figura 6.19 muestra un ejemplo para cada uno de estos conceptos.

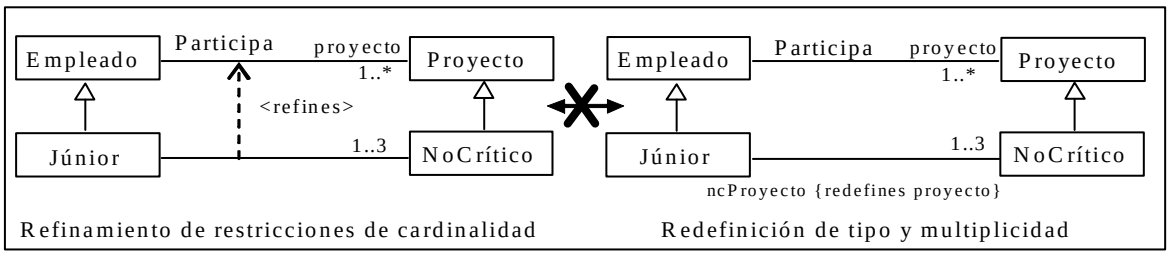



**Fig. 6.19.** Ejemplo de refinamiento de participantes y redefinición de tipo y multiplicidad

En el ejemplo de la figura anterior, la redefinición de multiplicidad definida sobre la asociación *Participa*, indica que un empleado júnior puede participar en como mínimo uno y como máximo tres proyectos no críticos. La redefinición de tipo definida indica además que los empleados juniors sólo pueden participar en proyectos no críticos.

Por otro lado, el refinamiento de restricciones de cardinalidad definido sobre la asociación *Participa*, indica también que un empleado júnior puede participar en como mínimo uno y como máximo tres proyectos no críticos. No obstante, a diferencia del caso del refinamiento, los empleados juniors pueden participar en otros proyectos que no sean críticos.

## 7. Extensión de UML para incorporar la semántica del Refinamiento de Asociaciones

Como hemos comentado en la sección anterior, el refinamiento de asociaciones es un concepto que hoy en día no ofrece el UML. El objetivo de esta sección es extender este lenguaje, para incorporar la semántica del refinamiento de asociaciones. De esta manera, se consiguen expresar todos aquellos casos del refinamiento que no quedan cubiertos por la redefinición de asociaciones.

Para realizar la extensión de UML, utilizamos un mecanismo que ofrece el propio lenguaje y que se conoce con el nombre de perfil (en inglés, *UML Profile*). Nuestra propuesta consiste en extender el perfil definido en [CGQ+06], creando dos nuevos estereotipos (uno por cada tipo de refinamiento). La figura 7.1 muestra estos estereotipos.

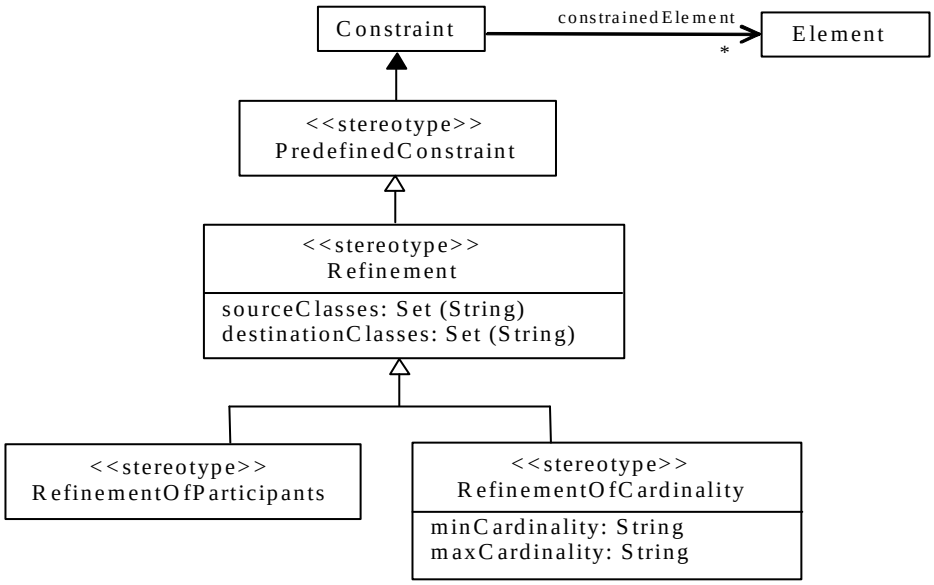



**Fig. 7.1.** Estereotipos para el refinamiento de asociaciones

Como podemos ver en la figura anterior, la metaclase *Constraint* está asociada a un conjunto de elementos llamado *constrainedElement*, es decir, el conjunto de elementos necesarios para evaluar la restricción (en inglés, *constraint*).

*PredefinedConstraint* es un estereotipo abstracto de la clase *Constraint* que define todas las características comunes a todas las restricciones predefinidas. El conjunto de las instancias de este estereotipo es la unión de las instancias de sus subtipos.

Nuestra propuesta consiste en crear un nuevo estereotipo abstracto *Refinement* que a su vez tiene dos subtipos: *RefinementOfParticipants* y *RefinementOfCardinality*.

En las siguientes subsecciones se explican con más detalle cada uno de estos estereotipos y se muestra la implementación que se ha llevado a cabo para incorporar dichos estereotipos a la herramienta CASE *PoseidonUML*.

## *7.1 Estereotipos definidos*

### *7.1.1 <<Refinement>>*

Recordemos que el refinamiento de una asociación ***R*** nos permite definir restricciones de integridad adicionales, cuando alguna de las instancias de las clases que participan en ***R***, es a su vez instancia de otras clases.

Esta restricción se puede expresar a través del estereotipo ***Refinement*** con dos atributos, ***sourceClasses*** y ***destinationClasses***, que representan las clases origen y destino que participan en el refinamiento de la asociación.

Las restricciones asociadas a este estereotipo son las siguientes:

- El ***constrainedElement*** debe ser un elemento de tipo ***AssociationEnd***.
- Las clases de las ***sourceClasses*** no pueden ser disjuntas entre ellas si son de la misma generalización/especialización.

### *7.1.2 <<RefinementOfParticipants>>*

El refinamiento de participantes de una asociación ***R*** restringe las posibles clases que pueden participar en ***R***. Esta restricción puede ser expresada a través del estereotipo ***RefinementOfParticipants*** con los atributos ***sourceClasses*** y ***destinationClasses***.

La figura 7.2 muestra un ejemplo de uso de este estereotipo.

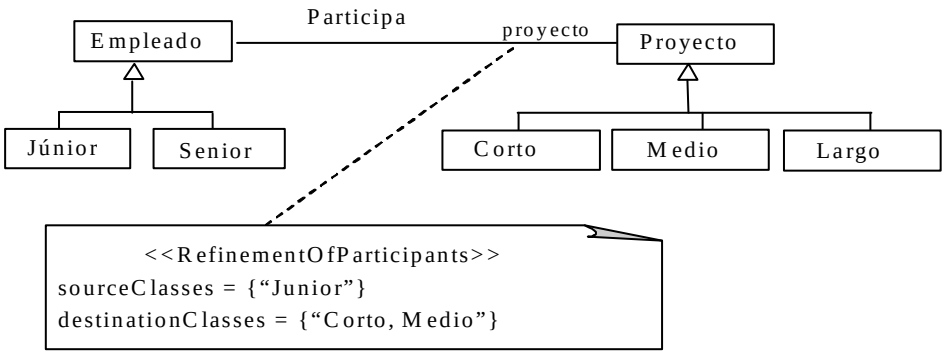



**Fig. 7.2.** Ejemplo de uso del estereotipo ***refinementOfParticipants***

El estereotipo definido en la figura 7.2, restringe la participación de un ***Empleado*** en la asociación ***Participa***. Concretamente, indica que un empleado ***Júnior*** sólo puede participar en proyectos de tipo ***Corto*** o ***Medio***.

Las instancias de este estereotipo tienen que cumplir las siguientes restricciones:

- Si el número de clases de ***sourceClassses*** es 0 entonces el número de clases de ***destinationClasses*** ha de ser más grande que 0.
- Si el número de clases de ***destinationClasses*** es 0 entonces el número de clases de ***sourceClasses*** ha de ser más grande que 0.
- Los valores del atributo ***sourceClasses*** deben ser clases descendientes de la clase conectada al extremo opuesto al ***constrainedElement***.
- Los valores del atributo ***destinationClasses*** deben ser clases descendientes (de la misma generalización/especialización) de la clase conectada al ***constrainedElement***.

### *7.1.3 <<RefinementOfCardinality>>*

Recordemos que el refinamiento de restricciones de cardinalidad de una asociación ***R***, permite definir restricciones de cardinalidad adicionales sobre ***R***. Para ello, es necesario definir nuevas cardinalidades que serán más restrictivas que las cardinalidades de la asociación R.

Esta restricción puede expresarse a través del estereotipo *RefinementOfCardinality* con cuatro atributos, *sourceClasses*, *destinationClasses*, *minCardinality y maxCardinality*. Estos últimos atributos indican los valores mínimo y máximo de las nuevas cardinalidades definidas.

La figura 7.3 muestra un ejemplo de uso de este estereotipo.

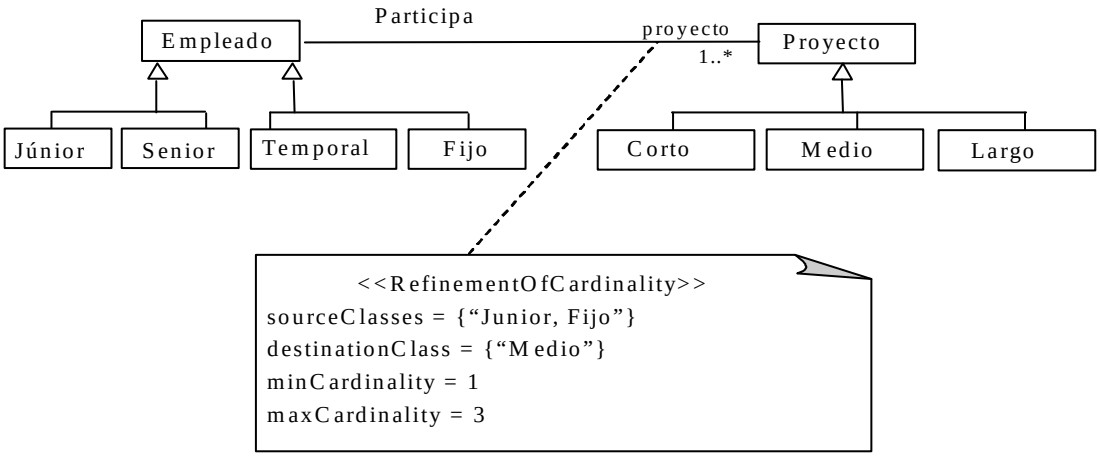



**Fig. 7.3.** Ejemplo de uso del estereotipo *refinementOfCardinality*

El estereotipo definido en la figura 7.3, indica que los empleados juniors que tienen una posición fija, pueden participar en como mínimo uno y como máximo tres proyectos de tipo *Medio*, pudiendo participar también en proyectos de tipo *Corto* y *Largo*.

Las instancias de este estereotipo tienen que cumplir las siguientes restricciones:

- El número de *sourceClasses* debe ser más grande que 0.
- El número de *destinationClasses* debe ser igual a 1 y puede ser la clase conectada al *constrainedElement* o un descendiente de ésta.
- Si el número de *sourceClasses* es 1 entonces dicha clase puede ser la clase conectada al extremo opuesto al *constrainedElement* o un descendiente de ésta.
- No se puede dar el caso que la clase origen sea la clase conectada al extremo opuesto del *constrainedElement* y la clase destino, la clase conectada al c*onstrainedElement*.
- Si el número de *sourceClasses* es más grande que 1 entonces éstas deben ser clases descendientes de la clase conectada al extremo opuesto al *constrainedElement*.
- Las cardinalidades *minCardinality* y *maxCardinality* deben ser más restrictivas que las cardinalidades definidas para el *constrainedElement.* Es decir, *minCardinality* debe ser mayor o igual a la cardinalidad mínima del *constrainedElement* y *maxCardinality* debe ser menor o igual a la cardinalidad máxima del *constrainedElement*. No se puede dar el caso que *minCardinality* sea igual a la cardinalidad mínima del *constrainedElement* y *maxCardinality* igual a la cardinalidad máxima de éste.

## *7.2 Implementación*

Los estereotipos que hemos definido para incorporar al UML la semántica del refinamiento de asociaciones han sido implementados en la herramienta CASE *PoseidonUML*. Esta herramienta ofrece un mecanismo para incorporar nuevas funcionalidades, conocido con el nombre de *plugin* y cuya programación debe realizarse en Java.

Nuestra implementación consiste en extender el *plugin* creado en [Vil07] con el nombre de *PredefinedConstraint*. Este *plugin*, nos permite definir y gestionar una serie de restricciones predefinidas (en inglés, *predefined constraints*) definidas en dicho trabajo. Concretamente, hemos creado dos nuevos estereotipos, con los que podemos expresar dos nuevas restricciones predefinidas: (1) el refinamiento de participantes, a través del estereotipo *RefinementOfParticipants* y (2) el refinamiento de restricciones de cardinalidad, con el estereotipo *RefinementOfCardinality*.

La extensión de este *plugin* se ha realizado para la versión estándar *PoseidonUML 6.0.* Puede descargarse de *http://www.lsi.upc.edu/~cristina/tesiMaster/PredefinedConstraints.zip* y se instala en la herramienta CASE como cualquier otro *plugin* desarrollado para la herramienta.

La figura 7.4 muestra un ejemplo de definición del estereotipo *RefinementOfParticipants* en *PoseidonUML*. El diseñador selecciona el extremo de asociación sobre el que desea aplicar el refinamiento y a continuación indica las clases origen y destino que participan en dicho refinamiento.

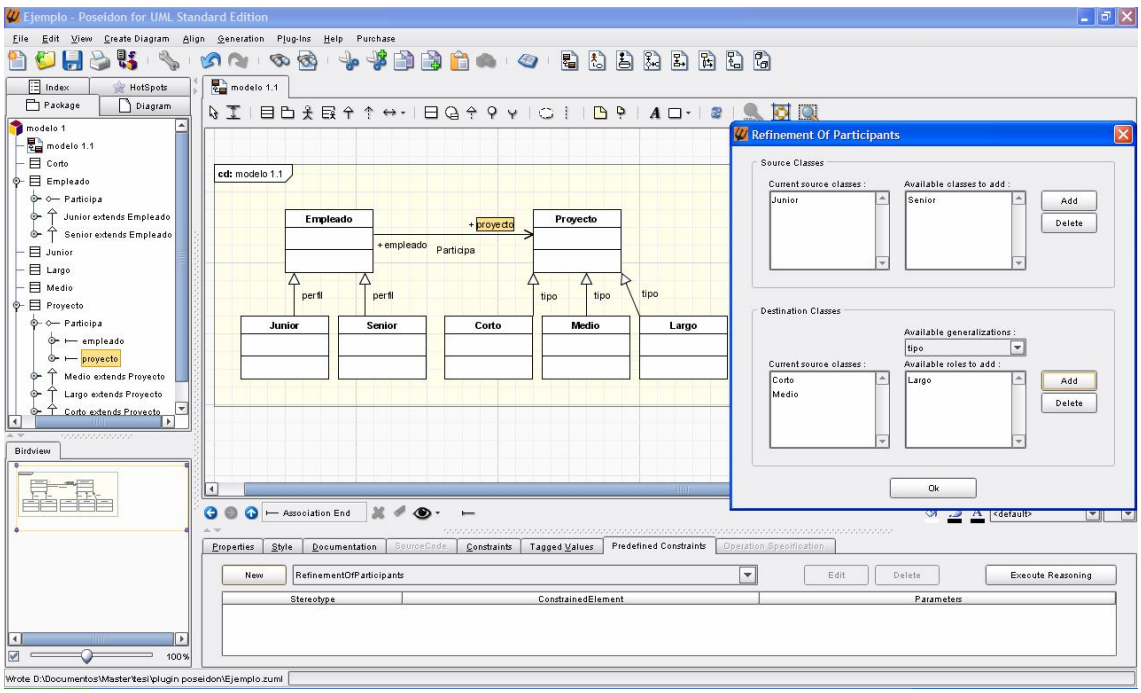



**Fig. 7.4.** Definición del refinamiento de participantes

Una vez definido el refinamiento, el esquema conceptual queda de la siguiente manera:

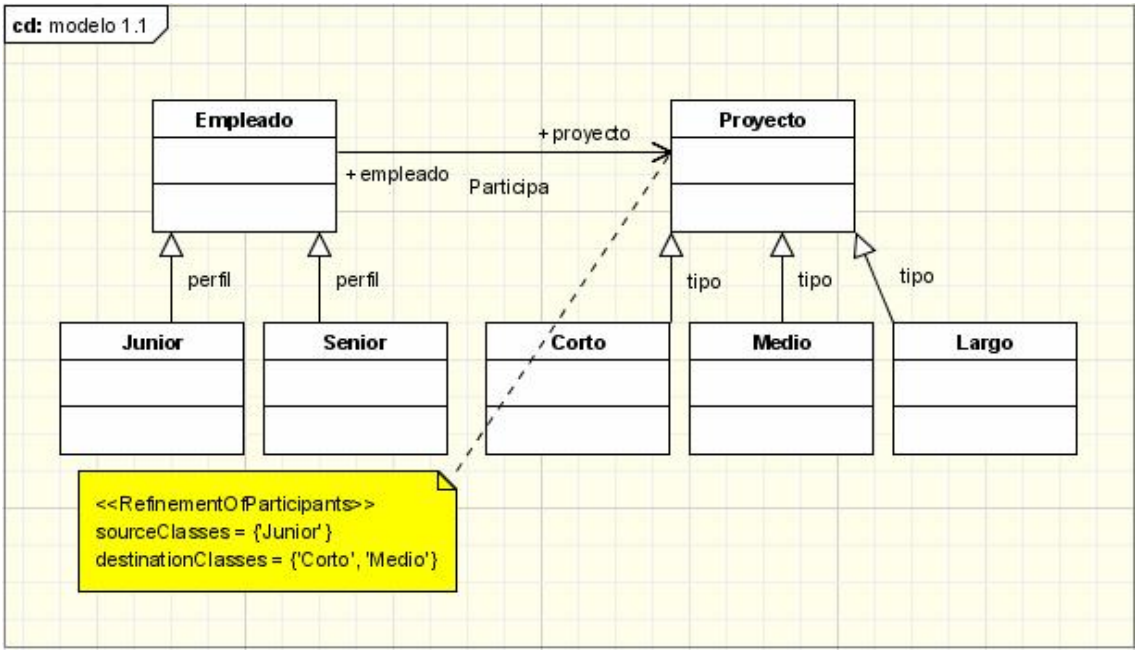



**Fig. 7.5.** Representación gráfica del refinamiento de participantes

En el anexo de este trabajo, se encuentra la guía de usuario para el *plugin* que hemos desarrollado. En ésta, se explica con más detalle cómo utilizar cada uno de los estereotipos implementados.

## 8. Conclusiones y trabajo futuro

En este trabajo hemos precisado la semántica de la redefinición de asociaciones en UML, a través de su traducción a nuestra capa básica de UML. Ésta consiste en un subconjunto de elementos básicos de UML con una definición precisa, junto a invariantes OCL. Así mismo, hemos comparado la redefinición de asociaciones con otros conceptos similares como el ***subsetting*** o el refinamiento de asociaciones. Traduciendo estos conceptos a nuestra capa básica de UML, hemos podido demostrar que la redefinición de asociaciones es semánticamente diferente al ***subsetting*** y el refinamiento de asociaciones.

Con el objetivo de incorporar la semántica del refinamiento de asociaciones al UML, hemos realizado una extensión de dicho lenguaje utilizando un perfil de UML (en inglés, *UML profile*). Concretamente, hemos creado nuevos estereotipos que nos permiten expresar en UML, todos aquellos casos de refinamiento que no quedan cubiertos por la redefinición de asociaciones. Finalmente, hemos implementado dichos estereotipos a través de un ***plugin*** para la herramienta CASE *PoseidonUML*.

Como trabajo futuro proponemos realizar una guía de usuario para orientar al diseñador de UML a la hora de definir redefiniciones de asociaciones. Otra propuesta, es comparar la redefinición de asociaciones con otros conceptos similares de UML, como la especialización de asociaciones. Tal y como sucede con el ***subsetting*** de asociaciones, el UML no deja claro la diferencia entre la redefinición y la especialización de asociaciones. La comparación de dichos conceptos, contribuiría a entender con mayor precisión la semántica de la redefinición de asociaciones en UML.

## Referencias


[AHM07] D. Akehurst, G. Howells, K. McDonald-Maier, “Implementing Associations: UML 2.0 to Java 5”, in: Journal on Software and Systems Modeling (2007) 6:3-35.

[And00] W. Andreopoulos, “Defining Formal Semantics for the Unified Modeling Language”, Research Report of Department of Computer Science, University of Toronto, (2000) http://www.cs.toronto.edu/~chechik/courses99/csc2108/projects/index.html.

[APFR+03] M. Albert, V. Pelechano, J. Fons, M. Ruiz, O. Pastor, “Implementing UML Association, Aggregation and Composition. A Particular Interpretation Based on a Multidimensional Framework”, in: Proceedings of CAISE 2003, LNCS 2681, pp 143-158.

[BHL+03] F. Barbier, B. Henderson-Sellers, A. Le Parc-Lacayrelle, J. M. Bruel, “Formalization of the Whole-Part Relationship in the Unified Modelling Language”. IEEE Transactions on Software Engineering 29 (2003) 459-470.

[BO92] S.E.Bratsberg, E.Odberg, “Relation Refinement in Object-Relation Data Models”. Nordic Workshop on Programming and Software Development Research, Tampere (Finland), 1992.

[BRJ99] G. Booch, J. Rumbaugh, I. Jacobson, “The unified modeling language user guide”, 2nd edn, (Addison-Wesley, 1999).

[BS95] R.J. Brachman, J.G. Schmolze, “An Overview of the KL-ONE Knowledge Representation System”. Cognitive Science 9 (2) (1995) 171-216.

[CD94] S.Cook, J.Daniels, “Designing Object Systems: Object-Oriented Modeling with Syntropy”. Prentice-Hall, 1994.

[CG06] D. Costal, C. Gómez, “On the Use of Association Redefinition in UML Class Diagrams”, in Embley D.W., Olivé, A., Ram, S. (Eds.), Conceptual Modeling, ER 2006, LNCS, vol. 4215, Springer, 2006, pp. 513-527.

[CGQ+06] D. Costal, C. Gómez, A. Queralt, R. Raventós, E. Teniente, “Facilitating the Definition of General Constraints in UML”. In Nierstrasz et al. (eds.), in: MoDELS 2006, LNCS 4199, pp. 260–274. Springer-Verlag, 2006.

[COT01] D. Costal, A. Olivé, E. Teniente, “Relationship Type Refinement in Conceptual Models with Multiple Classification”, in: 20th International Conference on Conceptual Modeling (ER’01), LNCS 2224, pp. 397-411.

[DD06] Z. Diskin, J. Dingel, “Mappings, Maps and Tables: Towards Formal Semantics for Associations in UML2”. In Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (Eds.), Model Driven Engineering Languages and Systems, MoDELS 2006, LNCS, vol. 4199 Springer, 2006, pp. 230-244.

[FELR98] R. France, A. Evans, K. Lano, B. Rumpe, “The UML as a Formal Notation”, in: The Unified Modeling Language (UML'98) - Beyond the Notation. First International Workshop, Lecture Notes in Computer Science, vol. 1618, (Springer-Verlag, 1998) 336-348.

[Fra99] R. B. France, “A Problem-Oriented Analysis of Basic UML Static Requirements Modeling Concepts”. In Proc. 1999 ACM SIGPLAN Conf. Object-Oriented Programming, Systems, Languages and Applications (OOPSLA’99), 1999, pp. 57-69.

[GLF04] G. Génova, J. Llorens, J.M. Fuentes, “UML Associations: A Structural and Contextual View”. Journal of Object Technology 3 (2004) 83-100.

[GLM02] G. Génova, J. Llorens, P. Martínez, “The meaning of multiplicity of n-ary associations in UML”. Software and Systems Modeling 1 (2002) 86-97.

[GR98] M. Gogolla, M. Richters, “Transforming Rules for UML Class Diagrams”, in: The Unified Modeling Language (UML'98) - Beyond the Notation, Lecture Notes in Computer Science, vol. 1618 (Springer-Verlag, 1998) 92-106.

[GR02] M. Gogolla, M. Richters, “Expressing UML Class Diagrams Properties with OCL”, in: Object Modeling with the OCL, Lecture Notes in Computer Science, vol. 2263 (Springer-Verlag, 2002) 85-114.

[GRL03] G. Génova, C. Ruiz del Castillo, J. Llorens, “Mapping UML Associations into Java Code”, Journal of Object Technology, 2(5):135-162, September-October 2003.

[HR04] D. Harel, B. Rumpe, “Meaningful Modeling: What's the Semantics of "Semantics"?”, in: IEEE Computer vol 37, n. 10, (IEEE Computer Society, 2004) 64-72.

[KER99] S. Kent, A. Evans, B. Rumpe, “UML Semantics FAQ”, in: European Conference on Object-Oriented Programming Workshops (ECOOP’99), Lecture Notes in Computer Science, vol. 1743, (Springer-Verlag, 1999) 33-56.

[Mil07] D. Milicev, “On the Semantics of Associations and Association Ends in UML”. IEEE Transactions on Software Engineering 33 (2007) 238-251.

[MBW80] J. Mylopoulos, P.A. Bernstein, H.K.T Wong, “A Language Facility for Designing Database-Intensive Applications”, TODS 5 (2) (1980) 185-207.

[OMG03] OMG, “MDA Guide Version 1.0.1”, OMG, omg/2003-06-01, 2003.

[OMG05] OMG, “UML 2.0 Superstructure Specification”, OMG Adopted Specification (2005)

[OMG06] OMG, “Object Constraint Language (OCL)”, Version 2.0, OMG Available Specification (formal/2006-05-01). (2006) http://www.omg.org/docs/ptc/03-10-14.pdf.

[OMG07] OMG, “Unified Modeling Language (OMG UML) Superstructure”, V2.1.2, OMG Available Specification without Change Bars (formal/2007-11-02). (2007).

[Oli07] A. Olivé, “Conceptual modeling of information systems”, (Springer-Verlag, 2007).

[RG98] M. Richters, M. Gogolla, “On Formalizing the UML Object Constraint Language OCL”, in: International Conference on Conceptual Modeling (ER'98), Lecture Notes in Computer Science, vol. 1507 (Springer-Verlag, 1998) 449 - 464.

[RJB05] J. Rumbaugh, I. Jacobson, G. Booch, “The unified modeling language reference manual”, 2nd edn, (Addison-Wesley, 2005).

[Rum00] B. Rumpe, “A Note on Semantics (with an Emphasis on UML)”, in: European Conference on Object-Oriented Programming Workshops (ECOOP’98), Workshop on Precise Behavioral Semantics (with an Emphasis on OO Business Specifications), vol. 1543 (Springer-Verlag, 1998).

[Ste01] P. Stevens, “On Associations in the Unified Modelling Language”, in: UML2001 Proceedings of the 4th International Conference (Springer LNCS 2185, 2001) 361 - 375

[Ste02] P. Stevens, “On the interpretation of binary associations in the Unified Modelling Language”. Software and Systems Modeling 1 (2002) 68-79.

[Szl06] M. Szlenk, “Formal Semantics and Reasoning about UML Class Diagram”, in: Proceedings of the International Conference on Dependability of Computer Systems (DEPCOS-RELCOMEX’06), (IEEE Computer Society, 2004) 51-59.

[Vil07] J. Vilà, “Extensió d’una CASE per facilitar la gestió de restriccions en UML” (2007).

[ZM04] X. Zhan, H. Miao, “An Approach to Formalizing the Semantics of UML Statecharts”, in: International Conference on Conceptual Modeling (ER'04), Lecture Notes in Computer Science, vol. 3288 (Springer-Verlag, 2004) 753-765.

## Anexo: Manual de usuario para el *plugin* de *PoseidonUML*

En este manual se explican las nuevas funcionalidades que se han incorporado al *plugin PredefinedConstraint*. Los pasos para realizar su instalación, así como las funcionalidades implementadas inicialmente, quedan explicados en el trabajo *Extensió d'una CASE per facilitar la gestió de restriccions en UML*, realizado en el 2007 por Joaquim Vilà Bou.

### *A.1 Creación de refinamientos*

Para crear un nuevo refinamiento se deben realizar los siguientes pasos:

1. Seleccionar el extremo de asociación sobre el que se desea definir el nuevo refinamiento.

2. En el *comboBox* de la pestaña *Predefined Constraints* se muestran los dos tipos de refinamiento que se pueden crear: *RefinementOfParticipants* y *RefinementOfCardinality*. Seleccionamos uno de ellos y pulsamos el botón de *New* (véase figura A.1).

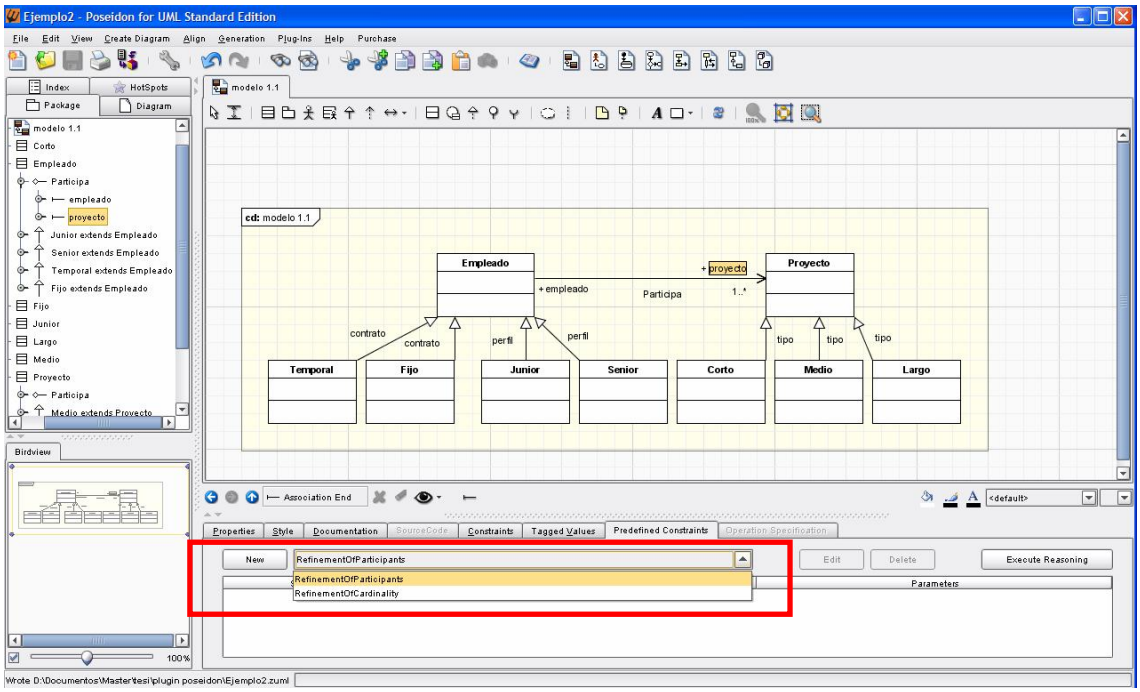



**Fig. A.1.** Selección del tipo de refinamiento

3. A continuación, se abre una ventana donde se deben introducir los parámetros necesarios para definir el refinamiento. Estos parámetros varían en función del tipo de refinamiento seleccionado.

    a. *RefinementOfParticipants*. Se deben especificar las clases origen y destino que participan en el refinamiento. Para las clases origen se utilizan las listas *available classes to add* y *current source class* del panel *Source Classes*. La primera de ellas indica las posibles clases origen que pueden participar en el

refinamiento y la segunda las finalmente escogidas. Los botones *Add* y *Delete* del panel *Source Classes* nos permiten añadir y borrar clases de la lista *current source class*. Para el caso de las clases destino, se utilizan las listas y botones del panel *Destination* Classes. En este caso, las clases destino deben pertenecer a la misma generalización/especialización. Por este motivo, para escoger las clases destino que participan en el refinamiento, previamente se debe seleccionar una generalización a través del *comboBox available generalizations*. En este combo, se muestran todas las generalizaciones en las que el padre es la clase conectada al extremo sobre el que se define el refinamiento (véase figura A.2).

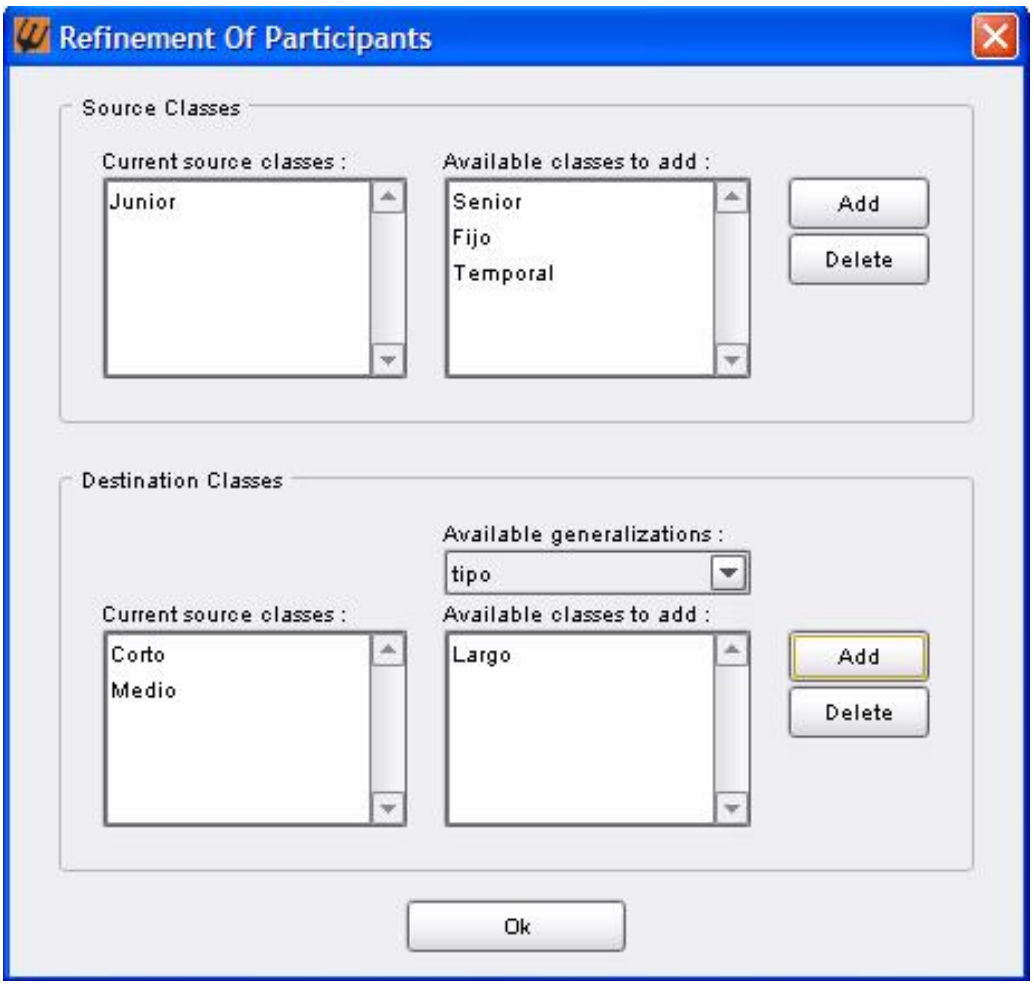



**Fig. A.2.** Introducción de los parámetros para la definición del refinamiento de participantes

b. *RefinementOfCardinality*. Se deben especificar las clases origen y destino que participan en el refinamiento, así como las nuevas cardinalidades. El funcionamiento para introducir las clases es el mismo que en el caso del refinamiento de participantes, pero con la diferencia de que en este caso no existe el *comboBox* de generalizaciones, ya que sólo puede existir una clase destino. Las cardinalidades mínima y máxima se especifican mediante los dos cuadros de texto del panel *Cardinalities* y deben ser más restrictivas que las cardinalidades del extremo sobre el que se define el refinamiento. La cardinalidad mínima debe ser un número natural y la máxima un número natural o * (véase figura A.3).

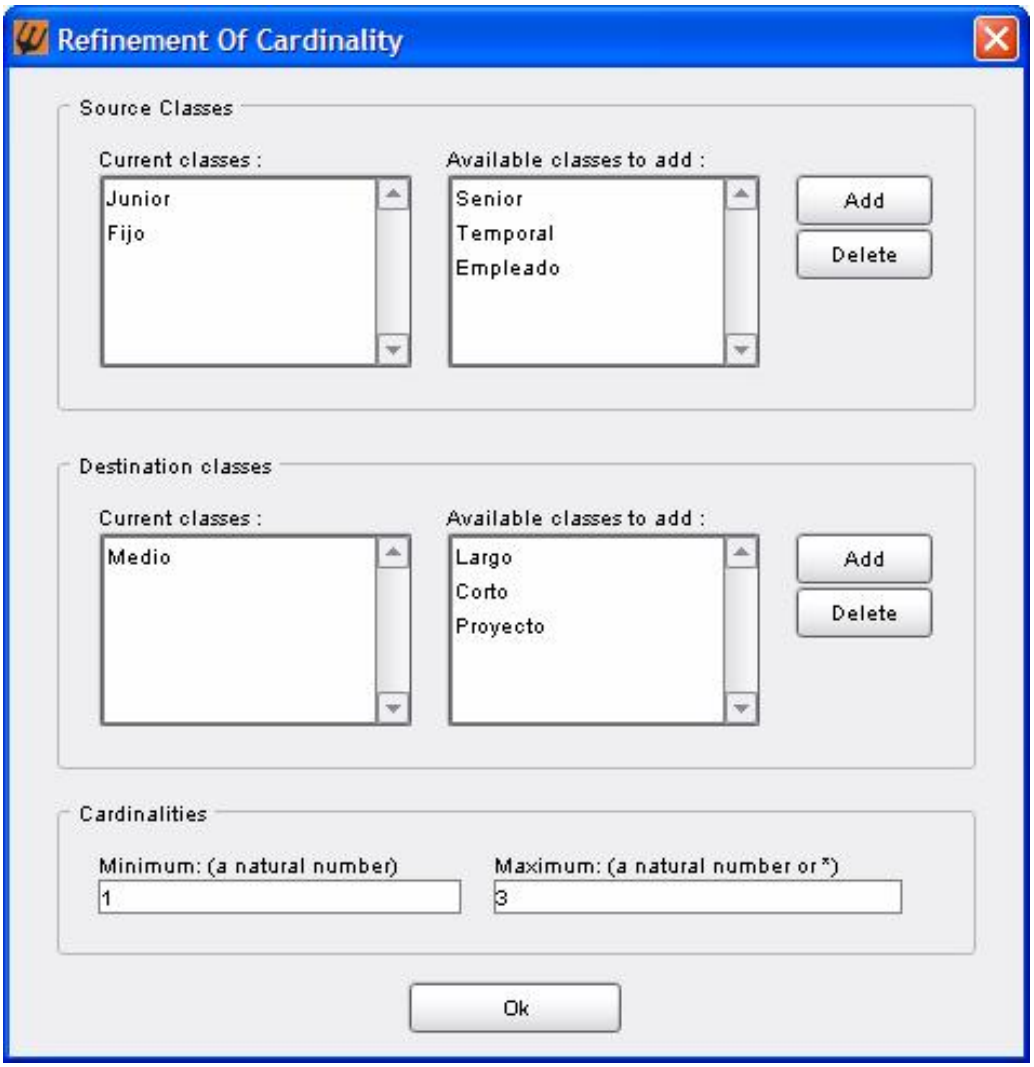



**Fig. A.3.** Introducción de los parámetros para la definición del refinamiento de cardinalidad

4. Una vez especificados los parámetros necesarios para la definición del refinamiento, pulsamos el botón O*k*. A continuación, se visualiza en el diagrama de clases el refinamiento definido mediante un comentario con ***stereotype*** y se actualiza la tabla de la pestaña *Predefined Constraints* con el nuevo estereotipo creado. Los valores del comentario varían en función del tipo de refinamiento.

a. *RefinementOfParticipants*.

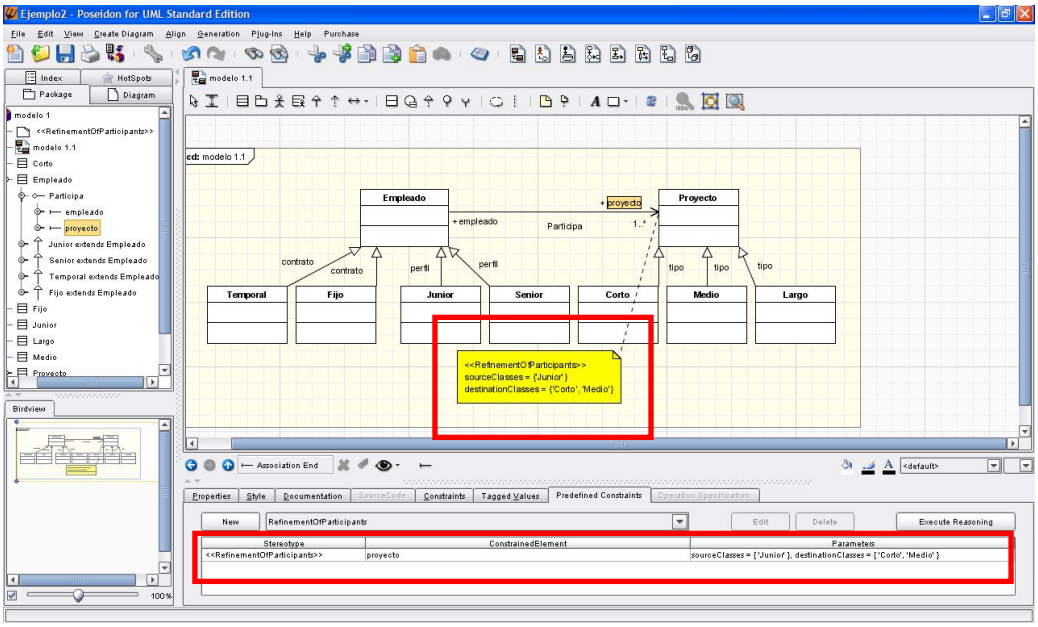



**Fig. A.4.** Representación del refinamiento de participantes

b. *RefinementOfCardinality*.

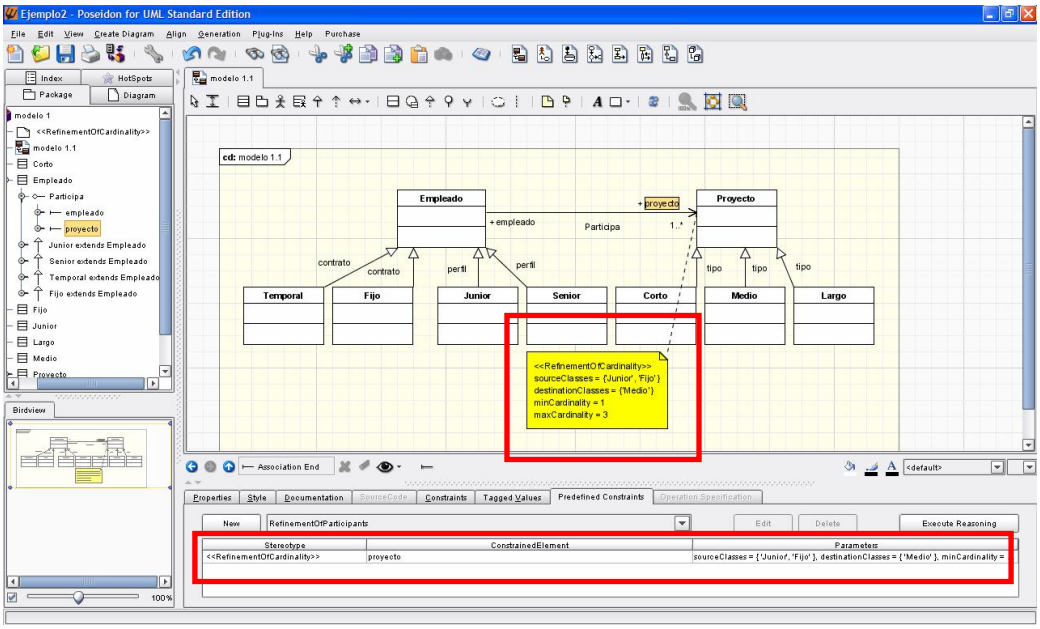



**Fig. A.5.** Representación del refinamiento de cardinalidad

## *A.2 Edición de refinamientos*

Los pasos a seguir para editar un refinamiento existente, es decir, modificar alguno de los valores de sus parámetros son los siguientes:

1. Seleccionar el extremo de asociación sobre el que se ha definido el refinamiento que deseamos editar.

2. En la tabla de la pestaña *Predefined Constraints* se muestran todos los refinamientos definidos para el extremo de la asociación. Seleccionamos uno de ellos y pulsamos el botón *Editar* (véase figura A.6).

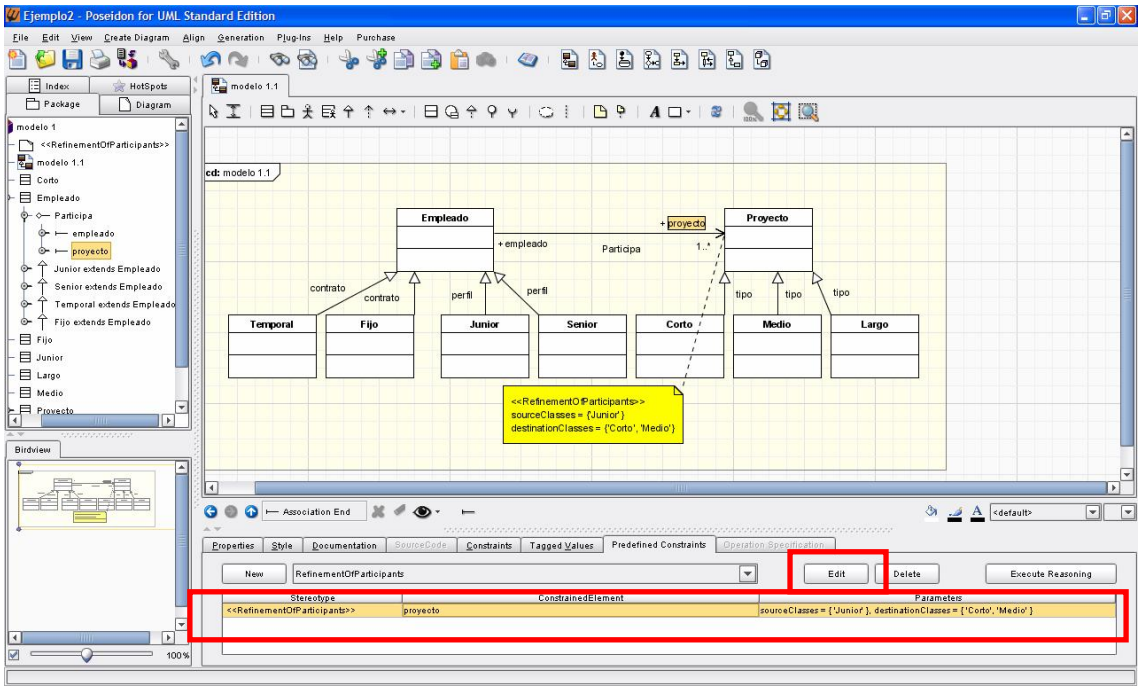



**Fig. A.6.** Edición de refinamientos

3. A continuación, se abre la ventana que se utiliza para introducir los parámetros, con los valores que tiene actualmente el refinamiento. Modificamos los valores que deseamos y pulsamos el botón *Ok*. El comentario del diagrama de clases y la tabla de la pestaña *Predefined Constraints* se actualizan con los nuevos valores.

## *A.3 Borrado de refinamientos*

Los pasos a seguir para borrar un refinamiento existente son los siguientes:

1. Seleccionar el extremo de asociación sobre el que se ha definido el refinamiento que deseamos borrar.

2. En la tabla de la pestaña *Predefined Constraints* se muestran todos los refinamientos definidos para el extremo de la asociación. Seleccionamos uno de ellos y pulsamos el botón *Delete* (véase figura A.7).

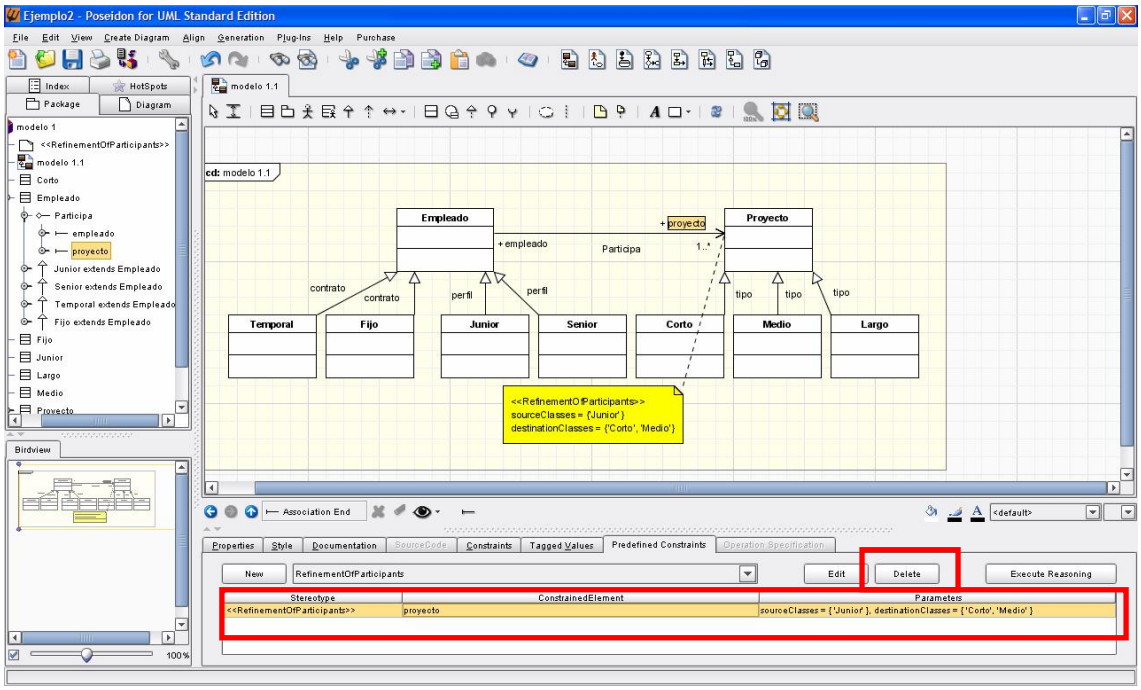



**Fig. A.7.** Borrado de refinamientos

3. A continuación, se borra el comentario del diagrama de clases y el estereotipo de de la tabla de la pestaña *Predefined Constraints*.