Una extensión de UML para modelar refinamientos

Roxana S. Giandini **Claudia F. Pons**

Universidad Nacional de La Plata, Facultad de Informática
LIFIA - Laboratorio de Investigación y Formación en Informática Avanzada
La Plata, Argentina, 1900
[giandini, cpons]@sol.info.unlp.edu.ar

Resumen

La técnica de refinamiento permite capturar la relación entre especificación e implementación en desarrollos de software. La documentación precisa de la relación de refinamiento facilita el rastreo de los requerimientos a través de los pasos de refinamiento. Cuanto más detalladamente se formula esta documentación, los requerimientos podrán ser rastreados más precisamente. La especificación detallada de la documentación para refinamientos no puede ser representada ni formalizada en UML, ya que este lenguaje carece de notación para ello. En este artículo presentamos una extensión del metamodelo de UML para modelar refinamientos, basada en la definición de nuevos estereotipos.

1. Introducción

El concepto de Abstracción [Dijkstra, 76] es la clave para manejar complejidad. La abstracción hace posible entender sistemas complejos y manejar las cuestiones principales antes de tomar en cuenta los detalles. Un modelo abstracto muestra información con no más detalle que el necesario, para luego poder ser refinado. Un refinamiento es una descripción más detallada que conforma a otra (su abstracción). Cada propiedad especificada en el modelo abstracto sigue siendo válida en el refinamiento, pero aquí posiblemente sean válidas nuevas propiedades.

Además de permitir manejar complejidad, la técnica de refinamiento captura la relación esencial entre especificación e implementación. El desarrollo por pasos de refinamiento, y en particular su documentación, permiten a los desarrolladores verificar si el código cumple con su especificación, controlar el impacto de cambios en los objetivos del negocio y ejecutar aserciones escritas en términos del vocabulario del modelo abstracto, traduciéndolas al vocabulario del modelo concreto.

La relación de refinamiento ha sido estudiada en varias notaciones formales tales como Z [Derrick - Boiten, 2001] y [Lano, 1996] y en diferentes contextos pero todavía, en lenguajes semi-formales como UML, existe una carencia de definiciones formales para este concepto. En este sentido, el artículo [Davies and Crichton, 2002] define una semántica formal para refinamiento y otro subconjunto de elementos de UML.

El lenguaje de modelado standard UML [OMG, 2004(b)] provee un artefacto llamado *Abstraction* (un tipo de *Dependency*) para especificar explícitamente relaciones entre elementos de modelado de UML. En el metamodelo de UML una *Abstraction* es una relación dirigida desde un cliente (*client*) a un proveedor (*supplier*) estableciendo que el cliente (el refinamiento) es dependiente del proveedor (la abstracción). El artefacto Abstraction tiene un meta-atributo llamado *mapping* para registrar los mapeos abstracción/ refinamiento, que es una documentación explícita de cómo las propiedades de un elemento abstracto son mapeados a sus versiones refinadas y, en la dirección opuesta, cómo elementos concretos pueden ser simplificados para ajustarse a una definición abstracta. Cuanto más formalmente se formula el mapping, los requerimientos serán rastreados más detalladamente a través de los pasos de refinamiento.

Existen trabajos recientemente publicados que analizan nociones básicas de las relaciones de dependencia en UML. Los artículos [Hnatkowska et al., Liu et al., 2004] presentan dos enfoques diferentes para la definición y uso de la relación de refinamiento, mientras que en [Kostrzewa, 2004] se describen algunos métodos formales de exploración automática de los conceptos de dependencias en el contexto de especificación de UML.

Aunque el artefacto *Abstraction* permite la documentación explicita de la relación de abstracción / refinamiento en modelos UML, resta especificar una importante cantidad de variaciones de esta relación por distintas causas; por ejemplo:

- por estar ocultas bajo otras notaciones
- por ser refinamientos compuestos de otros más elementales que no pueden ser representados ni formalizados en UML, ya que este lenguaje carece de notación para ello.

El primer caso fue tratado en el artículo [Pons and Kutsche, 2004], donde se estudian y analizan por ejemplo, artefactos UML tales como generalización, asociación compuesta, inclusión de casos de uso, entre otros, que implícitamente definen relaciones ocultas de abstracción/implementación. Con el fin de experimentar, fue creada una herramienta integrada en el ambiente Eclipse [IBM, 2003], llamada *PAMPERO* [Pons et al, 2003-4], basada en la definición formal de refinamiento. La herramienta soporta la documentación de refinamientos explícitos y el descubrimiento semi-automático y documentación de refinamientos ocultos. Además fue publicado un trabajo [Pons et al, 2004] con los últimos avances del proyecto *PAMPERO*.

En este artículo, nos centramos en el estudio del segundo caso, refinamientos que por estar compuestos por otros más elementales, carecen de precisión a la hora de definir sus *mappings*, ya que estos se expresan en términos de los refinamientos elementales. Aún en el caso más simple de refinamiento, por ejemplo el de clase, se presentan refinamientos compuestos, ya que al refinar una clase lo que se refina son sus componentes, por ejemplo sus atributos o sus operaciones.

La organización del artículo es la siguiente: en la sección 2 discutimos relaciones de refinamiento entre Clases y sus variantes, mientras que en la sección 3 expresamos estas variantes como composición de refinamientos más elementales; en la 4 presentamos una extensión del metamodelo de UML basada en la definición de nuevos estereotipos, formalizando la propuesta presentada en la sección 3. Por último presentamos conclusiones y líneas de trabajo futuro.

2. Refinamientos de clase

El método Catalysis [D´Souza and Wills, 98] menciona que los refinamientos entre clases (o Tipos) pueden ser realizados en dos formas diferentes: Refinamiento de atributos (o de modelo) y refinamiento de Operación. Estos refinamientos especifican que se pueden agregar nuevos atributos/operaciones a una clase o bien reemplazar atributos/operaciones existentes por otros, obteniendo así una clase nueva, refinamiento de la original.

En este trabajo, consideramos estos refinamientos y proponemos otras posibles variantes:

- **Refinamiento de atributos**

Una clase se obtiene de otra reemplazando uno de sus atributos por su refinamiento, el cual puede consistir de uno o más nuevos atributos. Por ejemplo, la figura 1 a) muestra que el atributo *currentPosition* de la clase *Player* es refinado por los atributos *previousPosition* y *stepsToMove* a través del *mapping* <currentPosition = previousPosition + stepsToMove>. Los *mappings* están expresados en OCL [OMG, 2004(a)], lenguaje para especificar restricciones para objetos.

- **Refinamiento de Operación**

Consideramos dos variantes de este refinamiento:

- Refinamiento de Operación *atómico:* expresa que una operación refina su postcondición.

Una clase se obtiene de otra reemplazando la postcondición de una de sus operaciones por su refinamiento. La signatura de la operación no cambia, solamente cambia su postcondición. La postcondición refinada debe implicar la original. Por ejemplo, la figura 1 b) muestra que la operación *move* en la clase *Player* es refinada atómicamente, modificando su postcondición. Podemos observar en el diagrama que cuando no aparece el *mapping*, la operación se mapea a sí misma, por lo que correspondería la función identidad como *mapping*.

- Refinamiento de Operación *no-atómico*: Una clase se obtiene de otra reemplazando una de sus operaciones por su refinamiento, el cual puede consistir de una o más nuevas operaciones. Por ejemplo, la figura 1c) muestra que la operación *move* en la clase *Player* es refinada no-atómicamente por las operaciones *moveForwards* y *moveBackwards* a través del *mapping* <move= moveForwards; moveBackwards >.

- **Refinamiento de Invariante:** Una clase es obtenida de otra reemplazando su invariante (aserción que debe mantenerse verdadera para toda instancia de la clase) por su refinamiento. La clase refinada tendrá una nueva invariante que implique la original. Por ejemplo, la figura 1d) muestra a la clase *Player* donde su invariante es refinado por otro, obteniendo la clase *Player´*, cuyo invariante implica al original.

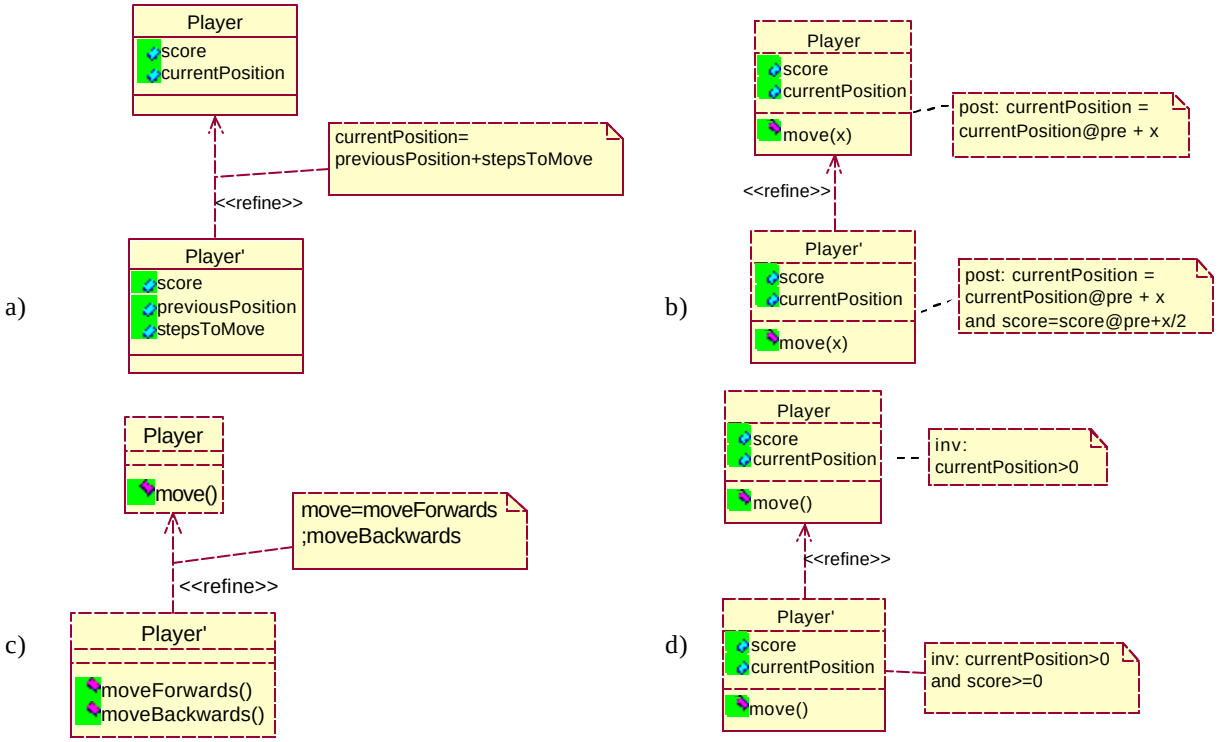

Figura 1. Distintos Refinamientos de clase: a) de atributo, b) de operación atómico, c) de operación no-atómico d) de Invariante

3. Refinamientos de clase expresados como composición de refinamientos más elementales

La dependencia *Abstraction* es imprecisa. Aún al ser estereotipada con el estereotipo <<refine>>, en el caso de refinamiento de clases no permite expresar si lo que se está refinando son los atributos, las operaciones de la clase, su invariante o bien una composición de ellos.
La figura 2 muestra la instancia del metamodelo de UML correspondiente al diagrama de la figura 1 a), donde el refinamiento está establecido entre clases, pero lo que específicamente se está refinando, es el atributo currentPosition. El *mapping* del refinamiento se expresa en función de atributos, aunque no queda claro que es un atributo lo que se está refinando. El metamodelo de UML es muy abstracto en este sentido, no permite instanciar con precisión refinamientos elementales.
Específicamente, es necesario poder expresar con más detalle los *mappings* del refinamiento de clases, para poder realizar el proceso de *traceability* en forma concreta y precisa.
Los *mappings* varían según que elemento se está refinando (atributos, operaciones o invariantes) por lo tanto *suppliers* y *clients* de la relación de dependencia también varían (son atributos, operaciones o invariantes de clase respectivamente). Podemos concluir que un refinamiento de clase se compone de cero o más refinamientos de atributos, cero o más refinamientos de operaciones y cero o más refinamientos de invariante.
En las siguientes subsecciones discutiremos cada uno de los refinamientos elementales que pueden componer un refinamiento de clase.

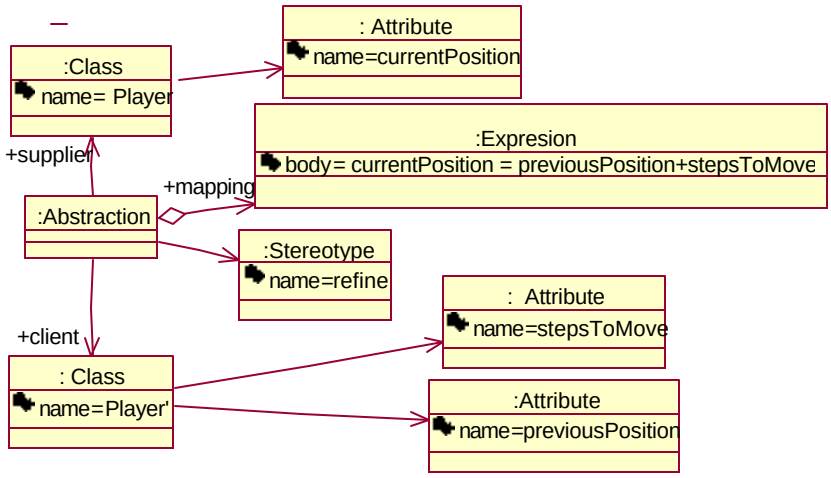

Figura 2. Diagrama de la figura 1 a) expresado como instancia del metamodelo

3.1 Refinamiento de Atributo

La figura 3 muestra el refinamiento de clase correspondiente a la figura 1a), donde se muestra explícitamente que está compuesto por un refinamiento de atributos.
Aparecen algunas cuestiones relacionadas al diagrama propuesto:

- UML no provee notación para mostrar un atributo en forma aislada, siempre lo muestra dentro del elemento (clase, por ejemplo) donde está definido.
- El metamodelo UML no permite expresar asociaciones entre dependencias. Es necesario poder denotar que un refinamiento se *compone* (la composición es una forma particular de la *asociación*) de otras abstracciones. Esto no es posible en las versiones de UML 1.x ni en UML 2.0 ya que las relaciones de dependencia (la abstracción es una de ellas) son *Relationship* (tienen una navegabilidad preestablecida, dirigida, tienen *client* y *supplier*) pero no son *Classifiers*, mientras que las asociaciones se establecen únicamente entre *Classifiers*.

Como vemos en el diagrama de la figura 3, una solución para esta situación de falta de expresividad es, por un lado, definir un nuevo estereotipo <<refinementComposition>>, para la metaclase *DirectedRelationship* (elegimos esta metaclase ya que, por lo que se explica previamente, en UML no sería válido extender la metaclase *Association*) para poder expresar composición entre refinamientos. Por otro lado, definir nuevos estereotipos para la metaclase *Abstraction*: <<classRefine>> que representa un refinamiento de clase compuesto, en este caso de uno más elemental: <<attributeRefine>>. Al extender el metamodelo de UML con estos nuevos estereotipos, estamos en condiciones de poder metamodelar este refinamiento de clase en forma más precisa, indicando qué elementos son los que realmente se están refinando (en este caso un atributo) y asociando el *mapping* a la abstracción correspondiente (entre atributos).

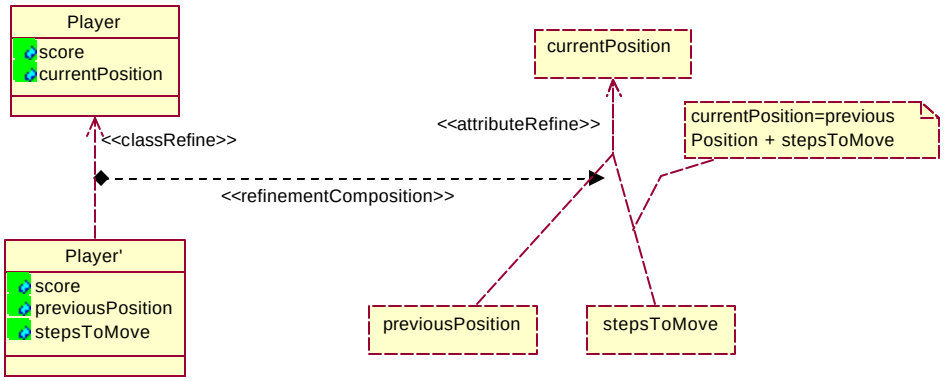

Figura 3. Refinamiento de clase compuesto por refinamiento de atributo.

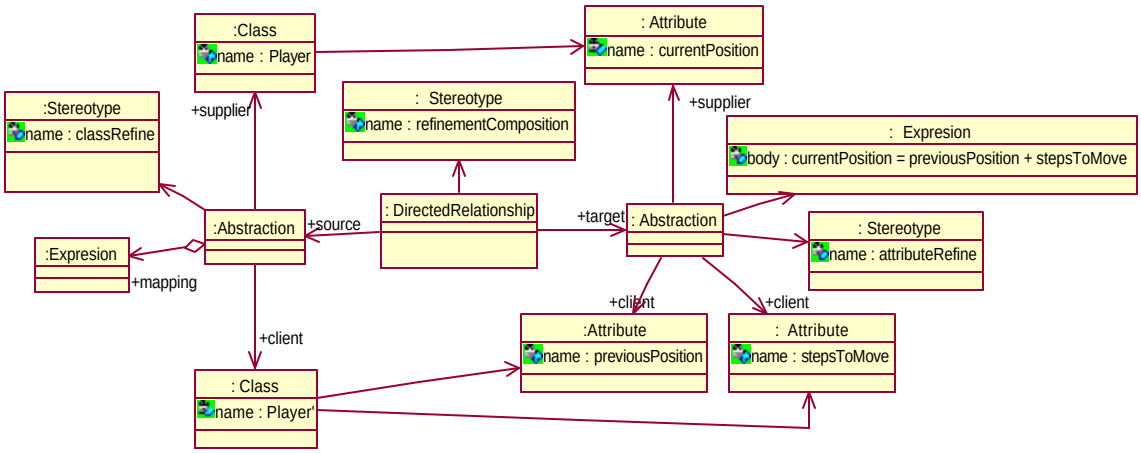

Figura 4. Diagrama de la figura 3 expresado como instancia del metamodelo propuesto

La figura 4 muestra la instanciación del metamodelo propuesto, para el diagrama de la figura 3. En este diagrama puede verse explícitamente la instanciación de la relación de refinamiento entre atributos. Una instancia de *DirectedRelationship* tiene un extremo *source* y un extremo *target*. En este caso, tiene el estereotipo refinementComposition (para representar la composición de refinamientos). El *source* es el refinamiento de clase y el *target* el refinamiento de atributo.

3.2 Refinamiento de Operación

En el caso de refinamiento de operación, sucede algo similar que al refinar atributos.
Cuando el refinamiento de operación es **atómico**, el *mapping* es la identidad. La figura 5 muestra el refinamiento de clase correspondiente a la figura 1b), donde se muestra explícitamente que está compuesto por un refinamiento de operación atómico. La operación sólo refina su postcondición. Aparece un nuevo estereotipo para *Abstraction*, <<atomicOpRefine>>. La figura 6 muestra la instanciación del metamodelo para este ejemplo. La relación de composición es instancia de la metaclase *DirectedRelationship*. El *source* es el refinamiento de clase y el *target*, el refinamiento de operación atómico. El *supplier* y el *client* del refinamiento atómico son conjuntos unitarios que representan la operación en cuestión con su postcondición (instancia de la metaclase *Constraint* con estereotipo *postCondition*) cambiada.

Cuando el refinamiento de operación es **no atómico**, el *mapping* relaciona a la operación refinada con las operaciones más concretas. La figura 7 muestra el refinamiento de clase correspondiente a la figura 1c), donde se muestra explícitamente que está compuesto por un refinamiento de operación no atómico. Los problemas son similares a los planteados anteriormente. Aparece por lo tanto un nuevo estereotipo para *Abstraction*: <<non-atomicOpRefine>>. Por la semejanza estructural de este refinamiento de operación con el de atributos, la instanciación del metamodelo es semejante también, por lo que no la incluimos.

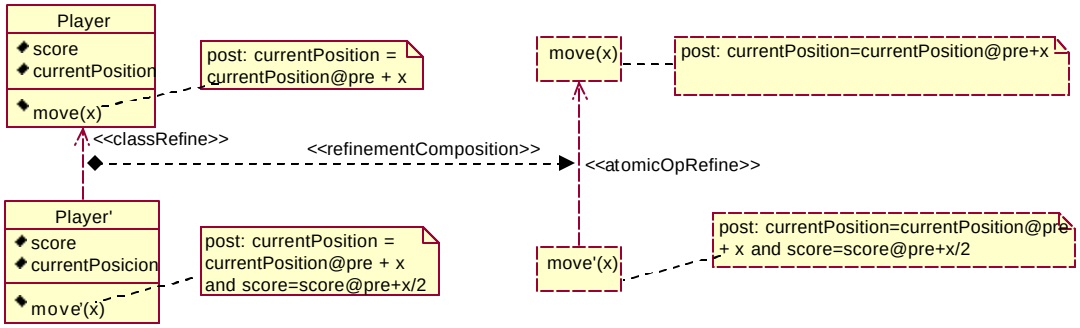

Figura 5.Refinamiento de clase compuesto por refinamiento de operación atómico.

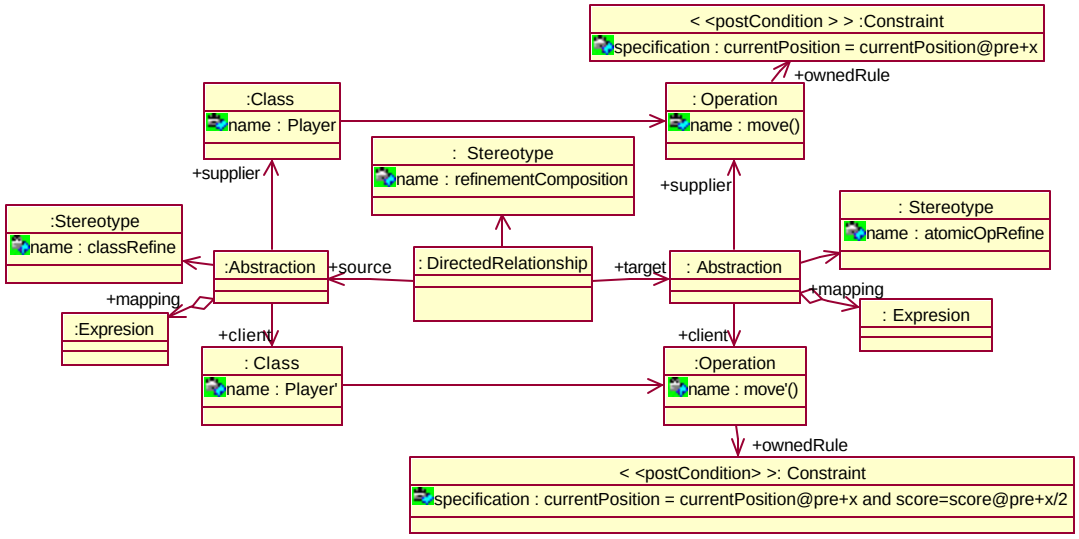

Figura 6. Diagrama de la figura 5 expresado como instancia del metamodelo propuesto

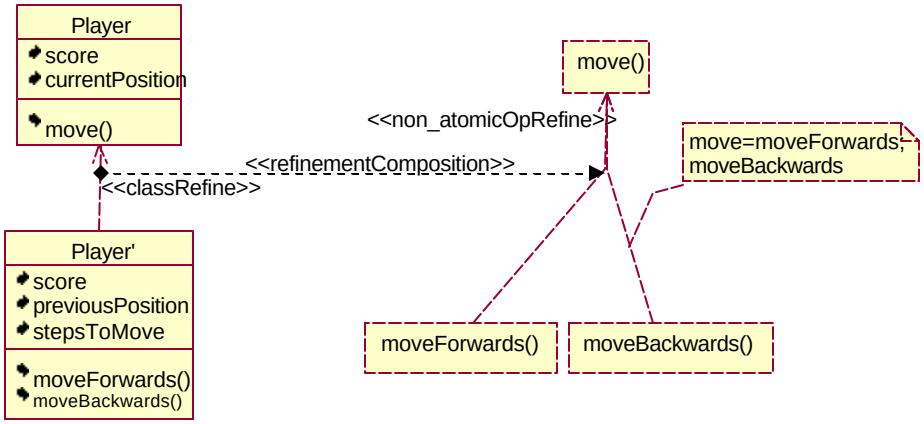

Figura 7. Refinamiento de clase compuesto por refinamiento de operación no-atómico

3.3 Refinamiento de Invariante

En el caso de refinamiento de invariante, el *mapping* es la identidad, ya que el refinamiento se realiza sobre la misma invariante de clase. La figura 8 presenta el refinamiento de clase correspondiente a la figura 1d), donde se muestra explícitamente que está compuesto por un refinamiento de invariante (un invariante es instancia de la metaclase *Constraint*). Por lo tanto, aparece la necesidad de definir un nuevo estereotipo para *Abstraction*, <<invariantRefine>>, donde *client* y *supplier* son *constraints*. La instanciación del metamodelo para este diagrama es similar a las anteriores.

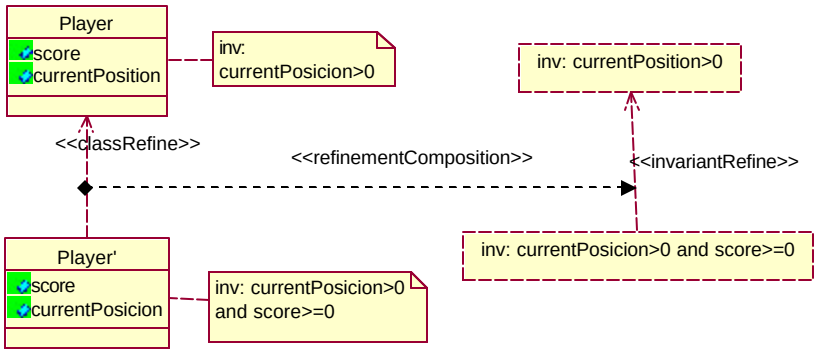

Figura 8. Refinamiento de clase compuesto por refinamiento de invariante.

4. Una extensión de UML para expresar refinamientos de clase

Para poder modelar refinamientos de clase más elementales en UML, como discutimos en la sección anterior, es necesario extender el metamodelo de UML. En las siguientes subsecciones presentamos una definición formal de los nuevos estereotipos mencionados previamente, indicando a que metaclase (cual es su base) estereotipan y las restricciones, expresadas en OCL [OMG, 2004(a)], que deben cumplirse al utilizarlos.

4.1 La composición entre refinamientos

En esta sección introducimos la extensión para metamodelar composición de refinamientos. Como dijimos previamente, expresamos esta composición específica definiendo un estereotipo para la metaclase *DirectedRelationship* (Figura 9).
Basándonos en el metamodelo de UML 2.0 [OMG, 2004(b)] definimos formalmente al estereotipo refinementComposition.

Figura 9. El estereotipo <<refinementComposition>> con base en DirectedRelationship.

- STEREOTYPE **refinementComposition**

Base: DirectedRelationship
Constraints:
1. Una relación << refinementComposition>> se debe establecer entre refinamientos (los extremos deben estar estereotipados con estereotipos de la jerarquía de *refine*)

```
self.source.type.oclIsKindOf(Abstraction) and
self.target.type.oclIsKindOf(Abstraction) and
self.source.type.stereotype.allParents -> includes (<<refine>>) and
self.target.type.stereotype.allParents -> includes (<<refine>>)
```

2. El extremo *source* de la relación debe ser un refinamiento compuesto (por ejemplo, no puede ser un refinamiento elemental de atributo, ni de operación, ni de invariante)

```
self.source.type.stereotype <> <<attributeRefine>> and
self.source.type.stereotype <> <<operationRefine>> and
self.source.type.stereotype <> <<invariantRefine>>
```

3. El extremo *target* de la relación debe pertenecer a la jerarquía de refinamientos elementales y no debe pertenecer a otra relación de composición entre refinamientos (composición fuerte).

```
self.target.type.stereotype.parent -> includes (<<elementalRefine>>) and
(self.owningPackage.ownedMember -> select (r: DirectedRelationship |
r.stereotype= << refinementComposition >> and r <> self ))-> forAll (r |
r.target <> self.target)
```

4.2 Estereotipos para modelar refinamientos de clase elementales

En la figura 10 presentamos un metamodelo para poder expresar refinamientos de clase. La subclase ClassRefine de la metaclase *Abstraction* modela refinamientos de clase y se compone de refinamientos elementales (subclases de ElementalRefine) que también son subclases de *Abstraction*. Consideramos los refinamientos elementales concretos de atributo, de operación y de invariante pero, por supuesto, la jerarquía queda abierta para análisis futuro.

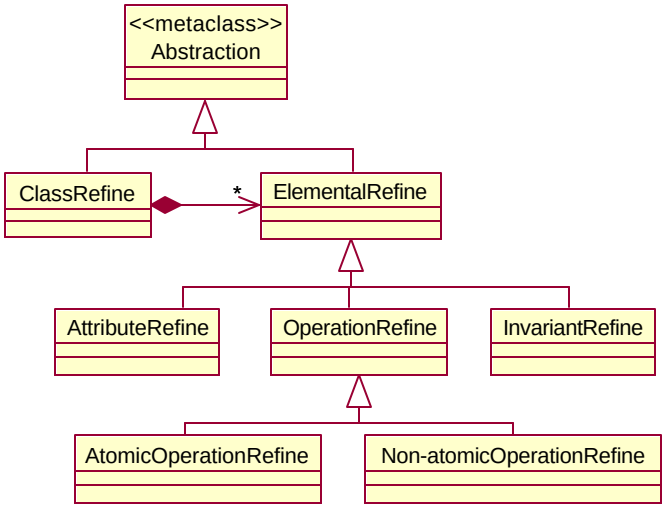

Figura 10. Metamodelo para expresar refinamientos de clase.

Para implementar esta idea en UML respetando la forma estándar de extensión al metamodelo (el documento de especificación de UML define como mecanismos de extensión la creación de nuevos estereotipos, valores etiquetados y restricciones sobre elementos ya existentes), decidimos basarnos en el estereotipo <<refine>> de *Abstraction* ya existente en el metamodelo de UML.

Concretamente, la extensión al metamodelo de UML que proponemos para expresar el diagrama de la Figura 10, consiste en definir una jerarquía de estereotipos con raíz en <<refine>> (Figura 11).
En el diagrama presentamos a igual nivel los estereotipos <<classRefine>> y <<elementalRefine>>. Más adelante, incluimos reglas de buena formación (*Constraints*) para expresar que un refinamiento con estereotipo <<classRefine>> se compone de refinamientos elementales, o sea refinamientos con estereotipo <<attributeRefine>>,<<operationRefine>> o <<invariantRefine>>, subestereotipos de <<elementalRefine>>.
A su vez, el refinamiento de operación se subclasifica con estereotipos más específicos: <<atomicOperationRefine>> y <<non_atomicOperationRefine>>, permitiendo así el modelado de los refinamientos de operación que introdujimos en la sección 2.

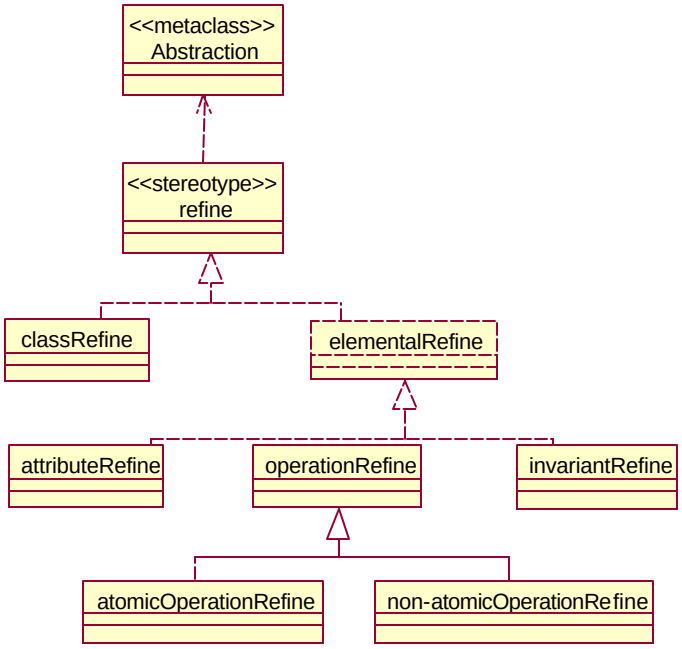

Figura 11. Estereotipos propuestos como extensión del metamodelo de UML.

4.2.1 Definición de los estereotipos

En este apartado definimos los estereotipos que forman la jerarquía descrita en la figura 11 cuya raíz es el estereotipo *refine*, con base en la metaclase *Abstraction*.

- STEREOTYPE **classRefine ,** subclase del estereotipo *refine*

Base: Abstraction
*Constraint*s:
1. Una Abstraction <<*classRefine*>> se debe establecer entre clases.

```
self.supplier.oclIsKindOf(Class) and self.client.oclIsKindOf(Class)
```

2. El *mapping* entre clases es una relación de correspondencia entre las propiedades *(features)* de la clase *supplier* y las propiedades de la clase *client*.

```
self.mapping.body = [s  (<p1, p2, ..,pn>) = < f(p1), f(p2), .. f(pn)>]
```

La especificación de una clase puede definirse por la lista de propiedades (sus atributos, operaciones, etc.) que describen a los objetos de la clase [Pons and Kutsche, 2004].
En este caso la lista de propiedades <p1, p2,..,pn> especifica a la clase *supplier* (`self.supplier`); s es la función que aplicada a la clase supplier (a su especificación), retorna su refinamiento (`self.client`); y *f* es la función que describe el *mapping* para cada propiedad.

s : Class -> Class

f: Feature -> Feature

Nota: Utilizamos una representación abstracta para expresar las funciones asociadas al *mapping*. Algunos de los *mappings* entre *features* no pueden ser expresados en OCL (por ejemplo composición de operaciones), por lo que es necesario definir un lenguaje simple, como trabajo futuro, para poder precisar esta representación.

- STEREOTYPE **elementalRefine ,** subclase del estereotipo *refine.* Es un estereotipo abstracto. Se subclasifica en estereotipos para refinamientos elementales concretos.

Base: Abstraction

*Constraint*s:
1. Una Abstraction <<*elementalRefine*>> no puede existir aislada, debe ser "parte de" sólo un refinamiento compuesto (de clase), o sea:

debe ser el extremo *target* de una relación <<refinementComposition>> entre abstracciones, donde el extremo *source* es una única abstracción <<classRefine>>

```
self.composite -> size = 1 and self.composite->stereotype= << classRefine >>
```

donde la operación *composite* retorna el conjunto de refinamientos compuestos de los cuales el refinamiento elemental (*self*), es parte:

composite: Abstraction -> Set(Abstraction)

```
composite = (self.owningPackage.ownedMember -> select (r: DirectedRelationship
| r.stereotype= << refinementComposition >> and
r.target = self ))  -> collect (r: DirectedRelationship | r.source )
```

- STEREOTYPE **attributeRefine,** subclase del estereotipo *elementalRefine*

Base: Abstraction
Constraints:
1.Una Abstraction <<*attributeRefine*>> se debe establecer entre atributos.

```
self.supplier.oclIsKindOf(Attribute) and self.client.oclIsKindOf(Attribute)
```

2. Los atributos que forman el refinamiento deben estar definidos en las clases del refinamiento que se está componiendo. Más precisamente:

2.1 los atributos *clients* deben pertenecer a la clase *client* de la abstracción <<classRefine>> (refinamiento compuesto, obtenido aplicando la operación *composite)*.

```
self.client-> forAll(c |self.composite.client.ownedAttribute -> includes(c))
```

2.2 el atributo *supplier* debe pertenecer a la clase *supplier* de la abstraccion <<classRefine>> (refinamiento compuesto, obtenido aplicando la operación *composite)*

```
self.composite.supplier.ownedAttribute -> includes (self.supplier)
```

- STEREOTYPE **operationRefine,** subclase del estereotipo *elementalRefine*

Este estereotipo es abstracto, ya que como vimos en la sección 2, un refinamiento de operación puede establecerse sobre la operación misma, que se refina con nuevas operaciones o sobre la postcondición de la operación y en este caso, la operación se mantiene pero con su postcondición refinada. Por lo tanto, este refinamiento se subclasifica en refinamiento de operación atómico y refinamiento de operación no atómico.
Base: Abstraction
Constraints:
1. Una Abstraction <<*operationRefine*>> se debe establecer entre operaciones.

```
self.supplier.oclIsKindOf(Operation) and self.client.oclIsKindOf(Operation)
```

2. Las operaciones que forman el refinamiento deben estar definidas en las clases del refinamiento que se está componiendo. Más precisamente:

2.1 las operaciones *clients* deben pertenecer a la clase *client* de la abstraccion <<classRefine>> (refinamiento compuesto, obtenido aplicando la operación *composite)*

```
self.client-> forAll(c |self.composite.client.ownedOperation -> includes(c))
```

2.2 la operación *supplier* deben pertenecer a la clase *supplier* de la abstracción <<classRefine>> (refinamiento compuesto, obtenido aplicando la operación *composite)*

```
self.composite.supplier.ownedOperation -> includes (self.supplier)
```

- STEREOTYPE **atomicOperationRefine** subclase del estereotipo *operationRefine*

Base: Abstraction
Constraints:
1. Una operación se refina con una sola operación.

```
self.client -> size = 1
```

2. La postCondición de la operación *client* implica la postCondición de la operación *supplier*

```
(self.client.ownedRule->detect (r| r.stereotype=<<postCondition>>)) implies
(self.supplier.ownedRule->detect (r| r.stereotype=<<postCondition>>))
```

- STEREOTYPE **non-atomicOperationRefine,** subclase del estereotipo *operationRefine*

Base: Abstraction
Constraints:
1. Una operación se refina con más de una operación.

```
self.client -> size > 1
```

- **STEREOTYPE invariantRefine** subclase del estereotipo *elementalRefine*

Base: Abstraction
Constraints:
1. Se debe establecer entre invariantes de clase, es decir *client* y *supplier* son *constraints* con estereotipo <<invariant>>

```
self.supplier.oclIsKindOf(Constraint) and self.client.oclIsKindOf(Constraint) and
self.supplier.stereotype=<<invariant>> and self.client.stereotype = <<invariant>>
```

2. Los invariantes que forman el refinamiento deben ser los invariantes de las clases de la relación que se está componiendo. Más precisamente:

2.1 el invariante *client* debe ser invariante de la clase *client* de la abstraccion <<classRefine>>, (refinamiento compuesto, obtenido aplicando la operación *composite)*

```
self.composite.client.ownedRule -> includes (self.client)
```

2.2 el invariante *supplier* debe ser el invariante de la clase *supplier* de la abstraccion <<classRefine>>, (refinamiento compuesto, obtenido aplicando la operación *composite)*

```
self.composite.supplier.ownedRule -> includes (self.supplier)
```

3. Un invariante se refina con un solo invariante.

```
self.client -> size = 1
```

4. El invariante *client* debe implicar al invariante *supplier*.

```
self.client implies self.supplier
```

5. Conclusiones y Trabajo Futuro

Cuanto más detalladamente se formula la documentación de refinamientos (*mapping*) en el desarrollo de software, los requerimientos podrán ser rastreados más precisamente a través de los pasos de refinamiento.

En este artículo, nos centramos en el estudio de refinamientos que por estar compuestos por otros más elementales, carecen de precisión a la hora de definir sus *mappings*, ya que estos se expresan en términos de los refinamientos elementales. Aún en el caso más simple de refinamiento, por ejemplo el de clase, se presentan refinamientos compuestos, ya que al refinar una clase lo que se refina son sus componentes, por ejemplo sus atributos o sus operaciones. La composición de refinamientos no puede ser representada ni formalizada en UML, ya que este lenguaje carece de notación para ello. Como una solución para esta falta de expresividad, hemos presentado una extensión del metamodelo de UML para modelar refinamientos, basada en la definición de nuevos estereotipos.

La relación de refinamiento puede establecerse entre elementos del mismo tipo (por ejemplo entre clases) o de tipos diferentes (por ejemplo un modelo de casos de uso y un modelo de colaboración que lo realiza) [Pons et al. 2000; Giandini and Pons, 2002; Pons et al., 2003].

En este trabajo hemos estudiado una forma de refinamiento entre elementos del mismo tipo (el refinamiento de clases). Como línea de trabajo futuro, incluiremos el tratamiento de otros refinamientos entre elementos del mismo tipo. La idea es discriminar al subclasificar el estereotipo <<refine>> para Abstraction. Una de las ramas será bajo el discriminante sameTypeElement al que pertenecen el refinamiento de clases, de casos de uso, de colaboraciones, etc., y otra bajo el discriminante differentTypeElement, donde *Client* y *Supplier* de la relación de refinamiento son de distinto tipo. Por ejemplo: un caso de uso *realizado por* una colaboración; una colaboración *con base en* diagramas de clase, etc.

En cada caso, ya sea aplicados a elementos del mismo o diferente tipo, estos estereotipos agregarán nuevas restricciones y nuevas definiciones para *mappings* más detallados con el fin de poder realizar con mayor precisión el proceso de *traceability* entre dichos elementos.

Referencias

Davies, J., Crichton, C. Concurrency and refinement in the UML. Electronic Notes in Theoretical Computer Science, Volume 70, July 2002

Derrick, J. and Boiten,E. Refinement in Z and Object-Z. Foundation and Advanced Applications. FACIT, Springer, 2001

D´ Souza, Desmond and Wills, Alan. Objects, Components and Frameworks with UML.Addison-Wesley. 1998.

Dijkstra, E.W., A Discipline of Programming. Prentice-Hall, 1976.

Giandini, R., Pons, C., Pérez,G. Use Case Refinements in the Object Oriented Software Development Process. Proceedings of CLEI 2002, ISBN 9974-7704-1-6, Uruguay. 2002.

Hnatkowska B., Huzar Z., Tuzinkiewicz L. On Understanding of Refinement Relationship. Workshop in Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships at the 7th International Conference on the UML. Lisbon, Portugal, October 11, 2004

IBM, The Eclipse Project. Home Page. Copyright IBM Corp. and others, 2000-2003. http://www.eclipse.org/.

Kostrzewa M. Exploration and Refinement of the UML Dependency Concepts. Workshop in Consistency Problems in UML-based Software Development III at the 7th International Conference on the UML. Lisbon, Portugal, Oct 11, 2004

Lano,K. The B Language and Method. FACIT. Springer, 1996.

Liu Z., Jifeng H., Li X., Chen Y. Consistency and Refinement of UML Models. Workshop in Consistency Problems in UML-based Software Development III at the 7th International Conference on the UML. Lisbon, Portugal, Oct 11, 2004

OMG (a). The Object Constrain Lenguaje Specification – Version 2.0, for UML 2.0, revised by the OMG, http://www.omg.org, April 2004.

OMG (b). The Unified Modeling Language Specification – Version 2.0, UML Specification, revised by the OMG, http://www.omg.org, April 2004.

Pons, C., Giandini R. and Baum G. Specifying Relationships between models through the software development process, Tenth International Workshop on Software specification and Design, San Diego., IEEE Computer Society Press. November 2000.

Pons, C., Pérez,G., Giandini, R., Kutsche, R. Understanding Refinement and Specialization in the UML. and. 2nd International Workshop on MAnaging SPEcialization/Generalization Hierarchies (MASPEGHI). In IEEE ASE 2003, Canada.

Pons, C., Giandini, R, Pérez., G., Pesce, P., Becker, V., Longinotti,J., Cengia,J., Kutsche, R-D., The PAMPERO Project: “Formal Tool for the Evolutionary Software Development Process”. Home page: http://sol.info.unlp.edu.ar/eclipse., 2003-4

Pons C., Giandini R., Pérez G., Pesce P., Becker V., Longinotti J., Cengia J., Correa N. and Labaronnie P. PAMPERO: Precise Assistant for the Modeling Process in an Environment with Refinement Orientation. In "UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers". Lecture Notes in Computer Science number 3297. Springer, Oct., 2004

Pons, C., Kutsche, R. Traceability Across Refinement Steps in UML Modeling. 3rd Workshop in Software Model Engineering WiSME at the 7th International Conference on the UML. October 11th 2004. Lisbon, Portugal .