# DOCUMENTACIÓN Y EVOLUCIÓN DE COMPONENTES REUSABLES

Contratos de Reuso con Semántica de Comportamiento

**Roxana Silvia Giandini**

**Director**: Lic. Gabriel Baum

**Tesis presentada al Dpto. de Informática de la Universidad Nacional de La Plata como parte de los requisitos para la obtención del título de Magister en Ingeniería de Software.**

**La Plata, Junio de 1999**

**Departamento de Informática**
**Facultad de Ciencias Exactas**
**Universidad Nacional de La Plata - Argentina**

# INDICE

# Capitulo 1: Introducción

El objetivo de este trabajo es brindar una metodología para documentación estructurada de componentes reusables que mantenga consistencia en la evolución de estas componentes y que muestre en que casos esta consistencia pueda ser alterada. El trabajo está dirigido particularmente a poder expresar, mediante esta metodología, *semántica de comportamiento* de estas componentes.
Este capítulo presenta en la sección 1.1 las motivaciones generales del trabajo, en la sección 1.2 los trabajos relacionados con la propuesta, en la 1.3 se presenta la propuesta y sus objetivos y por último se describe la forma en que se organiza el resto del trabajo.

## *1.1 Motivación*

A medida que en la ingeniería de software las aplicaciones se tornan más complejas y se pasa de construir sistemas simples a construir familias de sistemas, resultan necesarias nuevas tecnologías y herramientas de apoyo. Así, en el desarrollo de sistemas, surge la necesidad de construir componentes abstractas o semiacabadas que definan el comportamiento genérico de una familia de aplicaciones. Cada aplicación de la familia se puede obtener implementando porciones de comportamiento que le es especifico. A las componentes que reúnen esta característica se las denomina componentes reusables, las que frecuentemente son construidas sobre *frameworks*. Un framework está constituído por un conjunto de componentes que definen un diseño abstracto para soluciones de problemas de una familia de aplicaciones dentro de un dominio [JOH88]. Funciona como un *molde* que implementa el comportamiento genérico de un dominio de aplicación, dejando la implementación de aspectos específicos de cada aplicación para ser completado.
En la adaptación de aplicaciones basadas en el uso de componentes reusables, una técnica que utilizan frecuentemente los desarrolladores es descubrir puntos del framework que requerirán adaptación, por lo que deben mantenerse flexibles ( *hot spots*). Sin embargo, el recurso de completar *hot spots*, no resulta suficiente.
Los desarrolladores de software están confrontados a cambios constantes en el mercado. Así como evoluciona el negocio, el *framework* también debe hacerlo. Por esto un framework nunca se termina de construir. El objetivo de un framework es, además de permitir su adaptación mediante estos *hot spots*, consolidar el conocimiento adquirido del dominio durante los primeros proyectos de manera que pueda ser reusado en proyectos futuros para realizar un producto final. Así, un framework constituye una representación "siempre en evolución" en términos de variaciones y elementos en común de nuestro conocimiento del dominio del problema en un punto dado en el tiempo. Para que el reuso sea sistemático las componentes de un framework deben ser pre-planeadas y adecuadamente documentadas y su descubrimiento pasa a ser una meta principal en el proceso de desarrollo. Utilizar componentes reusables para la construcción de aplicaciones puede resultar costoso aún contando con documentación apropiada. La calidad de la documentación es básicamente importante para su mantenimiento y evolución. En la documentación de componentes reusables se presentan problemas de mayor complejidad por la naturaleza abstracta del diseño.
En el siguiente apartado se plantea el problema de lograr documentación estructurada para reuso; en las siguientes secciones se lo relaciona con nuestra propuesta.

### 1.1.1 Documentación para reuso.

Los ambientes de programación deberían proveer asistencia para actualizar a nuevas versiones las componentes reusables, basada en un entendimiento de la propagación de cambios. La ausencia de tales mecanismos es reconocida como un importante inhibidor para el reuso exitoso [GR95], [Pan95].
En la práctica, cuando el objetivo es el reuso de una componente o adaptación de parte de un framework, son necesarias conversaciones informales entre los reusadores y los diseñadores originales para clarificar temas de diseño que no pueden obtenerse de la documentación tradicional.
La clave para el desarrollo de software exitoso con frameworks es lograr una buena cooperación entre ingenieros de frameworks e ingenieros de aplicaciones [CHSV97]. Los ingenieros de frameworks tienen el importante rol de dar técnicas a los ingenieros de aplicaciones para incrementar los aspectos genéricos de sus diseños y código, guiar el proceso de adaptación y notificar sobre cómo reusar el framework. Los ingenieros de aplicaciones tienen que dar *feedback* sobre la adaptación del framework, de manera que éste pueda ser mejorado. Esta cooperación podría estar basada en contratos de reuso.
Un *contrato de reuso* es un conjunto de participantes relacionados que interactúan [LSM97, Lucas97b]. En los contratos de reuso cada participante lista los nombres de operaciones que son relevantes al diseño del framework y las cláusulas asociadas. La figura 1.1 muestra un contrato de reuso multi-clase, con dos participantes: CtaBanc y Titular; cada titular tiene asociado un conjunto de cuentas, mediante la relación de conocimiento *cuentas* y cada cuenta conoce a su titular mediante la relación de conocimiento *titular*. Cada participante lista sus operaciones (interfaz). La operación *variasExtrac*, que realiza un conjunto de extracciones simultáneamente, invoca a la operación *extraer*, del mismo participante, por lo que aparece en su cláusula de especialización. Las cláusulas de especialización listan los nombres de las operaciones invocadas por cada operación en la interfaz. Estas cláusulas proveen sólo información estática, que no es suficiente para determinar cuales operaciones deben ser redefinidas o refinadas. Al respecto, los *operadores de reuso*, que se aplican sobre contratos de reuso, documentan cómo se derivan contratos más especializados de otros. Proveen más información que simplemente una relación entre participantes, cubriendo las distintas formas en que todo el contrato o los participantes en particular pueden ser reusados.

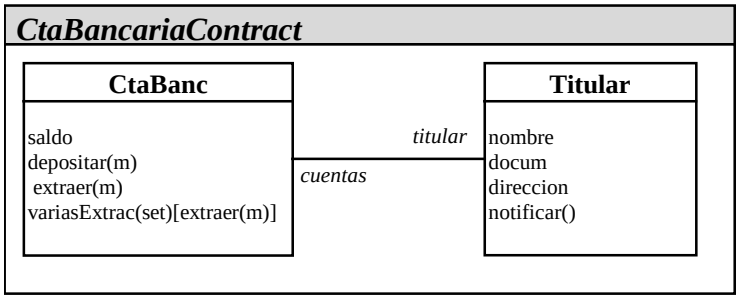


*Fig. 1.1. El contrato de reuso CtaBancaria*

**Conflictos en la evolución**

Además de documentar cómo un contrato es reusado por sus derivados, los operadores de reuso pueden ser usados para documentar la *evolución* de un contrato que ya tenía otro derivado. La evolución de componentes puede ocurrir en cualquiera de estas situaciones:

*Nuevos conocimientos sobre el dominio*. Luego de realizarse varias adaptaciones sobre las componentes, algunos conceptos específicos del dominio pueden tornarse generales y deben ser incorporados al framework.
*Complejidad de clases*. Al evolucionar el framework, las clases se tornan, en general, más complejas. Para reducir esta complejidad, el desarrollador del framework ha de considerar un rediseño del framework que puede variar desde introducir nuevas abstracciones (es decir, nuevas clases abstractas y la factorización de clases complejas) a utilizar patrones de diseño [GHJV94].
*Modificaciones en el diseño*. Algunos beneficios de diseño son descuidados u olvidados en la fase inicial de diseño del framework. Esto frecuentemente se torna en problemas que posiblemente afecten la *performance* durante la adaptación del framework. Como consecuencia, el framework deberá ser rediseñado para resolver estos problemas (por ejemplo, mejorar un algoritmo).

La figura 1.2 muestra dos modificaciones que se realizan sobre el participante CtaBanc del contrato de reuso presentado en la figura 1.1:

Evolución 1: **CtaBancNotifica**
El participante CtaBanc se modifica a fin de que sus instancias envíen un mensaje notificar() al titular cuando se realiza una extracción. Expresado en términos de operadores de reuso de [Lucas97b], es aplicar un operador de Refinamiento de Participante sobre la operación *extraer()* de CtaBanc, agregándole en su cláusula de especialización la operación titular.*notificar().*

Evolución 2: **CtaBancOptimizada**
El participante CtaBanc se modifica a fin de que la operación *variasExtrac*() haga todas las extracciones del conjunto directamente en vez de invocar a *extraer*(). Expresado en términos de operadores de reuso de [Lucas97b], es aplicar un operador de *Coarsening* de Participante sobre la operación *variasExtrac()* de CtaBanc, eliminando de su cláusula de especialización la operación *extraer().*

**Combinación de ambas modificaciones**

Ambas modificaciones en forma aislada son aplicables, pero su combinación conduce a un conflicto. Cuando se integran las dos evoluciones, ocurre comportamiento incorrecto ya que en un potencial participante derivado, los titulares no siempre serán notificados cuando se realicen extracciones en sus cuentas. En [Lucas97c], este conflicto es llamado de “operaciones inconsistentes” y ocurre cuando un modificador refina una operación mientras que el otro modificador la elimina de una cláusula de especialización.

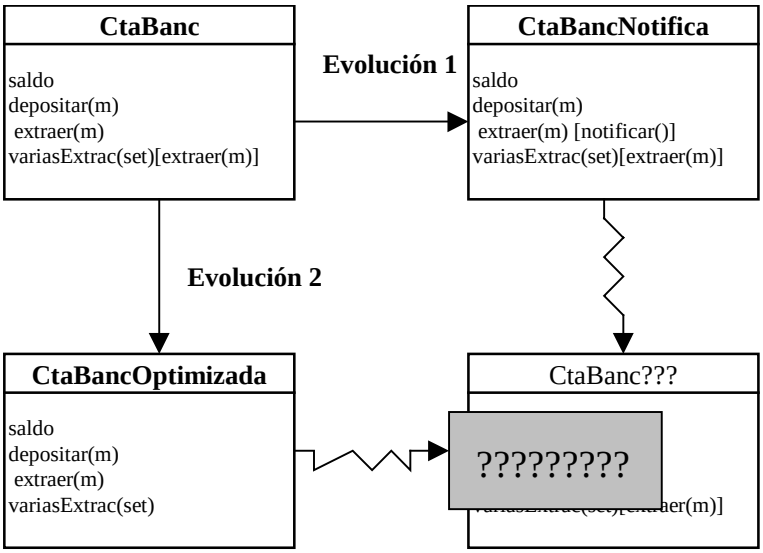


*Fig.1.2. Conflicto en la evolución de un participante del contrato*

Como se muestra en este ejemplo, la detección de *conflictos* puede ser realizada automáticamente por correlación de los operadores de reuso que documentan la primer derivación del contrato base con los que documentan la segunda. Pueden definirse reglas para señalar los posibles conflictos en contratos derivados existentes cuando se realizan nuevos cambios en el contrato base. Muchos de estos conflictos pueden ser expresados en términos de contratos de reuso y operadores de reuso en lugar que a nivel de interfaces y estructuras de llamadas. Este nivel de expresión permite a los desarrolladores razonar sobre cambios en forma más intuitiva. Además, estas reglas son la base para definir herramientas de desarrollo que puedan evaluar automáticamente el impacto de los cambios hechos al framework, prever cuáles conflictos podrían ocurrir y cuándo, guiar a los desarrolladores de aplicaciones en el entendimiento de dónde es necesario hacer pruebas y cómo reparar el conflicto.

## *1.2 Trabajos relacionados*

En los siguientes párrafos se citan algunos trabajos que guardan relación con la propuesta sobre diseño, documentación y evolución de componentes reusables construídas generalmente sobre *frameworks*. En este contexto, se incluyen los contratos de reuso ya citados en la sección anterior y los contratos de interacción. Ambos fueron utilizados como base para nuestra propuesta.

Sistemas de patrones:
Dentro de las técnicas propuestas para documentación de frameworks, podemos mencionar las *textuales*, como los *sistemas de patrones* propuestos por R. Johnson [Joh92]. Esta notación informal estructura la descripción como un conjunto de patrones organizados jerárquicamente. Cada patrón en la jerarquía explica que debe hacer el usuario para resolver un problema mediante el framework, señalando que clase especializar, por ejemplo. Estos sistemas se adecuan para describir la manera de utilizar un *framework* pero no proveen información suficiente para extender la funcionalidad.

Meta-patrones:
Los *meta-patrones* [Pre94] propuestos por W. Pree surgen como un complemento para documentar la estructura esencial de un framework [Pre95]. Básicamente, esta estructura está formada por la organización de métodos genéricos (*template*) y métodos redefinibles (*hook*), que al relacionarse permiten obtener un conjunto de combinaciones que determinan los meta-patrones. Basándose en esto, una clase de un framework es llamada clase *template* si contiene un método *template*, clase *hook* si contiene métodos *hook* y clase *template-hook* si cuenta con métodos de los dos tipos. Los meta-patrones brindan una forma sintética para documentar relaciones de clases esenciales en un framework, pero ofrecen estructuras de muy bajo nivel que no muestran claramente cómo utilizar un framework o cómo colaboran varios participantes. Sin embargo, su complementación con tarjetas de documentación de puntos de flexibilización (*Hot-Spot Cards)* [Pre96], brinda al usuario una guía interesante en la adaptación de un framework. Estas tarjetas tiene como objetivo describir la información necesaria para extender el framework en puntos de flexibilización previstos por el diseñador. Pree las clasifica en dos tipos: de función (que identifica una función del framework que debería mantenerse flexible y con que grado de flexibilidad: dinámica o adaptable por el usuario final, lo que definirá que meta-patrón se debe utilizar) y de datos (que identifica que elementos del dominio deberán generalizarse).

Interfaces de especialización:
Al construir y documentar librerías de clases y *frameworks*, la distinción entre componentes de "caja blanca" y componentes de "caja negra" [JF88] juega un papel crucial sobre la flexibilidad y extensibilidad. Las interfaces de componentes de caja blanca, están diseñadas para que los desarrolladores usen estas componentes, no para permitirles crear versiones específicas de una componente a fin de cumplir sus necesidades.
El problema principal es que no solamente el reusador invoca operaciones de componentes reusadas, sino que las componentes reusadas pueden también invocar operaciones provistas o redefinidas por el reusador. Si consideramos adaptar una componente heredando de ella (en orientación a objetos), cuando se redefine un método en una subclase, es posible que este método sea invocado por otros de la superclase, cuya estructura el reusador podría no entender. En forma similar, en sistemas de componentes, cuando un desarrollador inserta una componente dedicada en un sistema grande, no sólo esta componente invocará operaciones de otras componentes, estas otras invocarán también operaciones de la componente del usuario. Esto puede ocurrir en sistemas orientados a objetos a través de polimorfismo y envío de mensajes, como también a través de mecanismos de propagación de eventos como en JavaBeans [Sun97].
La primer propuesta en relación a este problema fue realizada por John Lamping, quien introdujo *interfaces de especialización* [Lam93]. En este trabajo se discute cómo los herederos de librerías de clase orientadas a objetos necesitan más información sobre la estructura de la clase que otros clientes. Específicamente, necesitan información de cuáles métodos invocan a cuáles otros en su implementación a fin de evaluar cuáles pueden ser heredados, cuáles necesitan redefinirse y cómo. Otros autores, como Ivar Jacobson sugieren controlar mediante *facades* (interfaces especiales) el acceso a sistemas de componentes [JGJ97]. Las *facades* proveen acceso público a sólo aquellas partes del sistema de componentes que ha sido elegido para hacer reuso, actuando como un tipo de interfaz pública para el sistema, minimizando dependencias y acotando cambios que el sistema de componentes permita. En JavaBeans [Sun97] se distinguen *interfaces en tiempo de diseño* de las *interfaces en tiempo de ejecución* (proceso de *customización*) Esta observación es válida no sólo en programación orientada a objetos; en general, el mismo problema se presenta en cualquier sistema donde se consideren diferentes versiones de una

componente que sean intercambiables e interactúen con su entorno. Este es el objetivo de los sistemas de componentes.

Contratos de interacción:
Los contratos de interacción en su forma declarativa [HHG90], brindan una herramienta en la programación orientada a objetos que puede ser usada como documentación estructurada.
Helm y Holland [HHG90] definen estos contratos como dependencias de comportamiento entre un conjunto de participantes que se comunican. Cada participante en un contrato debe realizar una secuencia de acciones cumpliendo ciertas condiciones al finalizar su realización; además el contrato permite expresar invariantes que los participantes cooperan en mantener. Estos contratos forman la primer propuesta que enfatiza sobre la importancia de incluir cooperación entre clases en documentación de sistemas orientados a objetos.

Diseño por contrato:
Otra propuesta similar a los contratos de interacción es presentada por Bertrand Meyer [Mey92]. Este trabajo presenta una visión del diseño de software basado en una metáfora: la construcción del software como una sucesión de decisiones contractuales documentadas. La idea es señalar pautas para que los programadores traten de construir software confiable (correcto y robusto), en particular muestra cómo el uso de contratos en el diseño clarifica los aspectos de herencia y *binding* dinámico en orientación a objetos, los cuales son fuente de confusión a menos que los desarrolladores comprendan su construcción en profundidad. La propuesta está descripta mediante mecanismos del lenguaje de programación orientado a objetos Eiffel, que permiten el uso de aserciones, manejo de excepciones y herencia.
Sin embargo, en ambos casos, los autores no brindan elementos de documentación que permitan expresar distintas formas de evolución del contrato ni de los participantes.

Evolución de componentes reusables:
En cuanto al problema de mantener la consistencia en la evolución de librerías de clases y *frameworks*, los trabajos de [KL92], [Lam93] y [Str86], enfatizan la importancia de contar con descripciones de interfaces claras. Los dos primeros resaltan la necesidad de la información estructurada. Presentan una visón general de algunos problemas pero no ofrecen soluciones particulares.
Con relación a este tema el trabajo de [KKS96] ayuda en el chequeo automático de validación de un número de restricciones de diseño.
El trabajo de Mezini [Mez97] ayuda además a resolver automáticamente algunos problemas que fueron especificados por diseñadores. Para esto propone un sistema de mantenimiento de consistencia automático, donde los diseñadores pueden formular propiedades del módulo base a ser propagadas a sus derivados durante la composición. Los cambios en el módulo base son monitoreados para filtrar alteraciones que pueden invalidar a los módulos derivados ya existentes. Las especificaciones sobre el diseño e implementación específicas del módulo base a ser propagadas son explícitamente formuladas por el diseñador en un lenguaje de descripción simple, llamado CCL (*cooperation contract language*). En este lenguaje puede especificarse que ciertos métodos son funcionales, por ejemplo, o que todos los métodos que modifican una cierta variable de instancia deberían además invocar a un cierto método. Se propone también una meta-programación para cambiar el comportamiento de constructores lingüisticos responsables en la evolución.

No existen mecanismos que asistan a los desarrolladores en la iteración sobre sus implementaciones. Cuando un desarrollador quiere hacer un cambio a un componente reusable es difícil evaluar dónde y cómo tal cambio influye en las aplicaciones construidas sobre esta componente. Similarmente, cuando un desarrollador de aplicaciones desea adaptar su aplicación a una nueva versión de los componentes que ha reusado, es difícil evaluar cuándo y dónde podrían ocurrir inconvenientes.

Contratos de reuso:
Como una solución a los problemas de documentación y consistencia de componentes reusables surgen los contratos de reuso [Lucas97a]. Un *contrato de reuso* es un conjunto de participantes que interactúan y sobre el que pueden aplicarse *operadores de reuso*. Estos operadores describen cómo un contrato se deriva de otro mediante la aplicación de un modificador.
Los contratos de reuso brindan documentación estructurada de componentes reusables y asisten al ingeniero de software en la adaptación de componentes para necesidades particulares. Proveen un vocabulario y notación para documentar reuso. En la evolución, los contratos de reuso ayudan a evaluar cuánto trabajo de actualización es necesario antes de construir aplicaciones, dónde y cómo realizar pruebas y cómo ajustar estas aplicaciones. Los contratos de reuso ayudan a derribar las barreras entre productores de componentes reusables y reusadores de componentes, reduciendo así la dicotomía existente entre la ingeniería de frameworks y la ingeniería de aplicaciones [GR95].
Dado que el reuso de frameworks se centra en el reuso de clases (abstractas) simples y en el reuso de la interacción entre clases, los contratos de reuso han sido definidos para estas dos formas [CHSV97]. Los contratos de reuso simple-clase [SLMH96] están basados en la noción de Lamping de interfaz de especialización [Lam93]. Los contratos de reuso multi-clase [LSM97 y Lucas97b] están basados en los contratos de interacción de Helm y Holland.
En [Lucas97b] se definen los contratos de reuso y los operadores que documentan y restringen su evolución, pero se deja de lado la verificación semántica de condiciones que sus participantes deben cumplir al desplegar ciertas acciones. Esta verificación es considerada en los contratos de interacción ya citados.
La noción de contrato de reuso es utilizada también en [MLS98a], donde se propone extender UML [BRJ97] a fin de brindar semánticas precisas para reuso de especificaciones y modelos de diseño expresados en ese lenguaje gráfico. Esto permitiría también detectar conflictos de reuso automáticamente. En [MLS98b] los autores logran la extensión del metamodelo UML e introducen una notación para expresar reuso y evolución de cualquier elemento dentro de UML.

## *1.3 Nuestra propuesta: Contratos de reuso con semántica de comportamiento*

El ejemplo de la sección 1.1 muestra que sobre un contrato de reuso y mediante el análisis de los operadores de reuso se pueden expresar conflictos relacionados con la evolución estructural de las componentes. Nada podemos decir sobre ciertas condiciones que deba cumplir un participante al finalizar la ejecución de una secuencia de acciones o sobre aserciones (invariantes) que todos los participantes del contrato cooperen en mantener.
Volviendo al ejemplo, sería interesante poder expresar que luego de ejecutarse la operación *extraer(m)* en cada cuenta, se debe cumplir como post-condición que el saldo haya sido decrementado en *m* (*saldo:= saldo – m*). En forma similar se podrían expresar post-

condiciones para el resto de las operaciones en el contrato. Por otro lado, sería útil poder expresar la restricción semántica de que el *saldo* debe mantenerse mayor que 0 y que la cantidad de extracciones se mantenga menor que un tope, para todas las instancias del participante CtaBanc. Esto podría constituir el invariante del contrato. Consecuentemente, al agregar elementos semánticos, estos deberían poder ser refinados, redefinidos y debilitados como sucede con los elementos sintácticos del contrato, para poder expresar evolución en la semántica de comportamiento, tanto a nivel participante como del contrato en general. Por ejemplo, si la política del banco ahora permitiera que sus clientes giren en descubierto hasta $1000, será necesario redefinir el invariante del contrato para expresar que el saldo de sus cuentas debe mantenerse mayor que -1000.

Del análisis de los trabajos presentados en la sección 1.2, surge la idea de *integrar* los contratos de reuso de Lucas, que brindan documentación estructurada para evolución, con la propuesta de Helm que trata la semántica de la composición de comportamiento entre los participantes de un contrato, pero no provee elementos para documentar evolución. El hecho de integrar semántica de comportamiento a nivel participante y a nivel contexto y proveer operadores para evolución semántica, los *operadores de reuso semántico,* resulta altamente interesante ya que esto permitirá a los desarrolladores poder expresar y documentar tanto evolución estructural de componentes como también evolución semántica del comportamiento de sus participantes. Además, ayuda a que los desarrolladores de aplicaciones puedan conseguir un mayor entendimiento tanto de la estructura y comportamiento operacional del software como de su evolución.
En la evolución de componentes, como vimos en la sección 1.1, pueden surgir problemas al producirse cambios sobre diferentes partes del sistema que se relacionan y al propagarse los cambios. Estos problemas se clasifican en conflictos de distintos tipos. La aplicación de los operadores de reuso sobre un contrato representa la evolución de componentes, por lo que pueden generarse dichos conflictos [Lucas97c]. En consecuencia, este trabajo también incluye el *análisis de los posibles conflictos* que pueden surgir al aplicar los nuevos operadores de reuso semántico.
A continuación, ante la necesidad de poder enunciar *propiedades de aplicabilidad* en la evolución, se define un *modelo matemático* para estudiar contratos de reuso. En términos de este modelo, es posible expresar y demostrar dichas propiedades.

Específicamente, los objetivos del trabajo son:

1. Proveer un modelo matemático para contratos de reuso con semántica de comportamiento y sus operadores de reuso sintáctico y semántico:

    1.1. Integrar elementos semánticos a un contrato de reuso (*post-condiciones* en las operaciones de un participante e *invariantes* para el contrato).

2. Documentar en forma estructurada la evolución de contratos de reuso con semántica de comportamiento:

    2.1. Definir *operadores de reuso semántico de contexto,* para obtener el refinamiento, redefinición o debilitamiento del invariante del contrato

    2.2. Definir *operadores de reuso semántico de participante*, para obtener el refinamiento, redefinición o debilitamiento de una operación que refina, redefine o debilita su post-condición).

3. Definir posibles conflictos que se generan al integrar los nuevos operadores en la evolución de un contrato.

4. En términos del modelo matemático, expresar características de la evolución:

Definir y demostrar propiedades de aplicabilidad que surgen del estudio de casos específicos en los que no ocurren conflictos al combinar operadores.

### 1.3.1 Publicaciones

Los siguientes artículos publicados son algunos de los resultados obtenidos respecto al tema de esta tesis:

- *Evolución de Contratos de Reuso multi-Clase con semántica de Comportamiento* [GPB98a]
  En este trabajo hemos presentado una primer integración de los contratos de interacción con los contratos de reuso, a fin de reducir la brecha existente entre la descripción operacional estática del contrato y la semántica de la composición de comportamiento entre sus participantes, introduciendo los operadores de reuso semántico con el fin de documentar la evolución tanto de una operación que refina o cambia su comportamiento como del contexto del contrato.

- *Manejando Formalmente Evolución de Contratos de Reuso con Semántica de Comportamiento* [GPB98b]
  Como una continuación de la metodología presentada en [GPB98a], en este artículo presentamos un análisis de posibles conflictos que pueden surgir como consecuencia de la aplicación de los nuevos operadores de reuso semántico sobre diferentes partes del sistema que se relacionan y al propagarse estos cambios.

- *Precise Semantics of Model Evolution* [PG99]
  Este artículo introduce una semántica formal basada en UML y lógica dinámica para conflictos que surgen en la evolución de componentes reusables.

### 1.3.2 Organización del Trabajo

Este trabajo de tesis está organizado de la siguiente forma: el Capítulo 2 introduce las definiciones básicas de contrato de reuso con semántica de comportamiento y de operadores de reuso, incluyendo post-condiciones para operaciones e invariantes para el contrato. El Capítulo 3 presenta las definiciones para los operadores de reuso sintáctico [Lucas97b], enriquecidos con post-condiciones para operaciones e invariantes para el contrato, mientras que el Capítulo 4 introduce las definiciones de los nuevos *operadores de reuso semántico*. En el capítulo 5 se ejemplifica la evolución de un contrato de reuso con semántica de comportamiento. El sexto plantea, clasifica y ejemplifica los conflictos que generan los nuevos operadores en la evolución. El Capítulo 7 presenta el *modelo matemático* para estudiar el dominio de los contratos de reuso y sus operadores, mientras que en el octavo se definen *propiedades de aplicabilidad* en la interacción de operadores sobre un contrato que pueden ser expresadas y probadas sobre la base de dicho modelo. El

Capítulo 9 presenta un Caso de Estudio con el fin de mostrar la utilidad de estos contratos en la evolución de componentes reusables. Por último se presentan conclusiones y se discute la dirección que seguirá el trabajo futuro.

# Capítulo 2: Contratos de Reuso con Semántica de Comportamiento

En la primera sección de este capítulo presentamos una descripción ampliada de la noción de Contrato de Reuso [Lucas97b] a la que denominamos Contrato de Reuso con Semántica de Comportamiento, incluyendo post-condiciones para operaciones e invariantes para el contrato e introducimos la propiedad de buena formación para contratos de reuso**.** Presentamos también una extensión de la integración presentada por [Lucas97d] del diagrama de clases de UML con los contratos de reuso y por último mostramos un ejemplo de contrato de reuso bien formado.
En la segunda sección se describe en forma general el concepto de operador de reuso. En los capítulos siguientes se definen en particular los operadores de reuso sintáctico y los operadores de reuso semántico y se muestra la evolución de contratos. En el Capítulo 7 presentamos un modelo matemático para contratos y operadores de reuso con el agregado de post-condiciones para operaciones e invariantes para el contrato. Sobre esta notación matemática probaremos luego propiedades de aplicabilidad de los operadores.

## *2.1 Contrato de Reuso con Semántica de Comportamiento*

Un contrato de reuso con semántica de comportamiento es un conjunto de participantes relacionados que interactúan y además un *invariante.*
El conjunto de participantes en un contrato junto con las relaciones de conocimiento entre ellos constituye el *contexto* del contrato.
Cada participante en el contrato consta de:
1. un *nombre p* que es único dentro del contrato;
2. una *cláusula de conocimiento*, que consta de un conjunto de relaciones de conocimiento de la forma *a.q*. El significado intuitivo es que *p* conoce al participante *q* a través de la relación de conocimiento denotada por *a*.
3. una *interfaz*, es decir, un conjunto de especificaciones de operaciones cada una formada por:

a) un *nombre* de operación que es único dentro de la interfaz;

b) una *cláusula de especialización*, que consta de una secuencia ordenada de invocaciones de operaciones *a.m*, asociando un nombre de conocimiento *a* con un nombre de operación *m.* El operador ";" entre invocaciones indica que se ejecutan en secuencia. El operador "||"entre invocaciones indica que se ejecutan en paralelo, por lo tanto entre ellas no interesa el orden de ejecución.

b) una *condición* asociada que establece relaciones entre valores del dominio de la aplicación anteriores y posteriores a la ejecución de la operación.

El conjunto de nombres de operaciones de una interfaz constituye la interfaz del participante.

### 2.1.1 Propiedad de Buena Formación

Debido a que en un contrato de reuso no es deseable encontrar referencias no resueltas, como una invocación a una operación que no existe, el contrato necesita cumplir con algunas condiciones de buena formación.

Un contrato de reuso R está bien formado si :

1.para todo participante *p* valen las siguientes condiciones :

a. para cada relación de conocimiento *a.q* en la cláusula de conocimiento de *p*, existe un participante con nombre *q* en R ;

b.para cada invocación de operación *a.m* en una cláusula de especialización en *p* :

b.1. *a* es un nombre de conocimiento en la cláusula de conocimiento de *p*;

b.2. *m* es el nombre de una operación en la interfaz del participante referido por la relación de conocimiento *a*.

c. para cada operación *m*, su post-condición es un conjunto de expresiones OCL bien formadas que involucran operaciones y participantes de R. Además, la oparación *m* preserva al invariante del contrato.

2. el invariante del contrato involucra operaciones y participantes que existen en R y es una expresión booleana OCL bien formada,

donde OCL (Object Constraint Language Specification) [RMH97] es un lenguaje de especificación formal integrado al UML. Las invariantes en un contrato, son expresiones OCL (properties) libres de efectos laterales.

En las post-condiciones pueden aparecen variables *primadas*[1] que representan el valor de la variable en el estado siguiente a la ejecución de la operación; esto puede verse en el ejemplo del apartado 2.1.3.

El contrato de reuso puede expresarse en UML, como veremos en el próximo apartado.

### 2.1.2 Integración de Contratos de Reuso con UML

Los contratos de reuso no son enteramente una nueva metodología sino que enriquecen metodologías existentes; la idea de integrar contratos de reuso en UML [BRJ97] surge para poder aplicarlos al campo de librerías de clases y *frameworks* orientados a objetos.

De los diversos diagramas que presenta el UML, resultan relevantes el diagrama de clases y el de colaboraciones. El primero se centra en cómo las clases se combinan estructuralmente, mientras que el segundo muestra como colaboran diferentes objetos que se conocen. Nuestra definición de Contrato de Reuso permite realizar una extensión de la integración presentada por [Lucas97d] del diagrama de clases de UML con los contratos de reuso, de la que surgen los *contratos de reuso multi-clase*. En estos contratos, participantes son mapeados a clases, operaciones a métodos, relaciones de conocimiento a asociaciones y nombres de conocimiento a nombres de roles. Sólo falta un ítem de la notación básica de contrato de reuso: invocación de operaciones; en sistemas orientados a objetos, esto naturalmente es mapeado a pasaje de mensajes. En UML, el pasaje de mensajes no se modela en el diagrama de clases, sino en diagramas de colaboraciones o en diagramas de secuencia. Lucas permite, por resultar útil, modelar pasaje de mensajes entre clases. Esta necesidad aparece también al describir patrones de diseño, donde frecuentemente se ligan

[1] Las variables *primadas* son una extensión de la seudovariable "*result*" de OCL.

fragmentos de código a asociaciones [GHJV94]. Así se extiende la notación de diagrama de clases con pasaje de mensajes ligado a asociaciones UML, como en los contratos de reuso originales.
Nuestra propuesta consiste en la integración de post-condiciones para operaciones e invariante para el contrato a fin de reflejar la semántica del comportamiento entre participantes. Como dijimos en el apartado 2.1.1, post-condiciones e invariante son expresados en lenguaje OCL (Object Constraint Language Specification) [RMH97], lo cual permite su verificación.

### 2.1.3 Ejemplo de Contrato de Reuso bien-formado

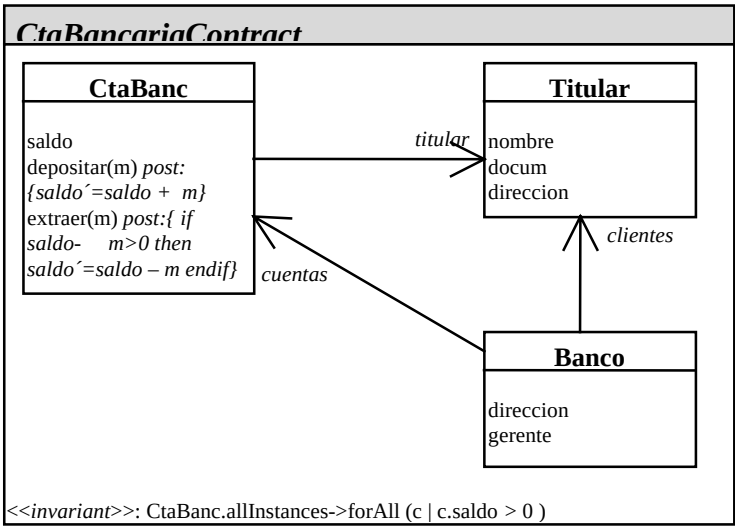


*Fig. 2.1. El contrato de reuso CtaBancaria*

La Fig.2.1 muestra el contrato de reuso con semántica de comportamiento bien-formado *CtaBancaria* que representa un participante (*Banco*) relacionado con sus cuentas (relación de conocimiento *cuentas* con el participante *CtaBanc)* y con sus clientes (relación de conocimiento *clientes* con el participante *Titular*). A su vez, CtaBanc se relaciona con Titular mediante la relación de conocimiento *titular*. Cada CtaBanc debe mantener siempre su saldo mayor que cero*;* esto aparece en la sección *<<invariant>>* expresado en OCL al igual que la post-condición de las operaciones *extraer*() y *depositar*(). La post-condición de *extraer*() especifica que si antes de ejecutarse esta operación el valor de saldo es mayor que m, entonces en el estado siguiente a la ejecución (*saldo'* ) el valor de *saldo* se decrementó en *m.*

## *2.2 Operadores sobre Contratos de Reuso*

Nuestro trabajo se centra en la integración de los contratos de interacción con los contratos de reuso, a fin de reducir la brecha existente entre la descripción operacional estática de un contrato y la semántica de la composición de comportamiento entre sus participantes. Como consecuencia de esta integración, en la sección anterior fue presentada una adaptación de la definición de contrato de reuso, mientras que en esta sección se presenta una adaptación de los operadores de reuso *sintáctico* de [Lucas97b], y la introducción de nuevos operadores: *los operadores de reuso semántico.*

Los *operadores de reuso sintáctico* son adaptaciones de los operadores de [Lucas97b], ya que continúan realizando el mismo tipo de operaciones pero en su definición intervienen también los nuevos elementos del contrato que expresan semántica de comportamiento: post-condiciones para operaciones e invariante para el contrato.
Los *operadores de reuso semántico* son nuevos operadores introducidos en este trabajo, y surgen de la necesidad de poder expresar ciertas restricciones sobre la semántica del comportamiento composicional entre los participantes de un contrato. Estos operadores documentan la evolución tanto de una operación que refina o cambia su comportamiento (es decir refina o redefine su post-condición), como del contrato (es decir que refina o redefine el invariante). Para los refinamientos semánticos existen también las operaciones inversas: *coarsening* semántico de participante y de contexto.
Los desarrolladores de aplicaciones pueden conseguir, a través de estos operadores, un mayor entendimiento de la semántica del comportamiento de los elementos intervinientes en un contrato.

Un *operador de reuso* describe cómo un contrato de reuso es derivado de otro mediante la aplicación de un modificador. Un modificador de reuso consta de los elementos que participan en la modificación del contrato base. Cada operador se describe por medio de tres definiciones:

- *Una definición de modificador*: describe los elementos que participan en la modificación del contrato base para este operador en particular;
- *Una definición de aplicabilidad*: propiedades que debe cumplir un modificador a fin de que el operador sea aplicable a un contrato de reuso particular;
- *Una definición del resultado*: describe cómo se determina el resultado de aplicar el operador.

Además un operador debe preservar la *propiedad de buena-formación*: la aplicación de un operador a un contrato de reuso bien formado resulta en otro contrato de reuso bien formado.

Los modificadores de reuso se muestran al lado de la línea de asociación UML, con un estereotipo para el nombre y una nota UML para la descripción, como se verá en ejemplos del próximo capítulo. Los operadores de refinamiento se expresan a través de la relación de generalización.

En los capítulos 3 y 4 respectivamente, se definen y ejemplifican los operadores de reuso sintáctico y los operadores de reuso semántico. En el Capítulo 7 se presenta la notación matemática que define cada operador sobre contratos de reuso. Con esta notación pueden enunciarse y probarse propiedades de aplicabilidad de operadores, como se verá en los últimos capítulos.

# Capítulo 3: Operadores de Reuso Sintáctico

Respecto a su sintaxis, un contrato de reuso tiene cuatro elementos: participantes y relaciones de conocimiento, que juntos forman el contexto, y operaciones e invocaciones de operaciones. Cada uno de estos elementos puede ser agregado o eliminado de un contrato, lo cual lleva a ocho operadores básicos:

| Operador | Descripción | Significado |
|---|---|---|
| Extp | Extensión de Participante | agregar nuevas operaciones |
| Canp | Cancelación de Participante | eliminar operaciones |
| Refp | Refinamiento de Participante | agregar nuevas invocaciones de operaciones |
| Coap | *Coarsening* de Participante | eliminar invocaciones de operaciones |
| Extc | Extensión de Contexto | agregar nuevos participantes |
| Canc | Cancelación de Contexto | eliminar participantes |
| Refc | Refinamiento de Contexto | agregar nuevas relaciones de conocimiento |
| Coac | *Coarsening* de Contexto | eliminar relaciones de conocimiento |

En las siguientes secciones presentamos la definición completa de estos operadores. En el Capítulo 7 se encuentra la notación matemática para las definiciones de cada operador.

## *3.1 Extensión sintáctica de participante: Extp*

El objetivo de la extensión de participante es agregar nuevas operaciones a uno o más participantes en un contrato. Esta operación se aplica usualmente con el fin de adicionar nueva funcionalidad a un participante y, como consecuencia, al contrato.
La extensión de participante debe contenerse a sí misma: las nuevas operaciones agregadas no pueden referir a operaciones existentes en el contrato. Así, una extensión es completamente independiente del contrato que extiende. Esto podría ser no deseado frecuentemente en la práctica, pero se pretende que los operadores básicos sean, en lo posible, ortogonales. Utilizando un refinamiento extendido de participante, que combina ambas operaciones (refinamiento y extensión de participante) se pueden agregar dependencias entre las operaciones agregadas por la extensión y las que ya existían en el contrato.
La extensiones sintácticas, de participante y de contexto, no admiten la inclusión de elementos semánticos. Si es necesario explicitarlos, se hará aplicando un operador de reuso semántico. En ausencia de invariantes se asume la expresión OCL *true* y en ausencia de post-condiciones se asume el conjunto vacío de expresiones.

La figura 3.1 muestra una Extensión de los participantes Banco y CtaBanc en el contrato *CtaBancaria;* las operaciones *atender()* y *procesar(),* son agregadas a cada participante respectivamente, con el fin de que el banco atienda requerimientos que serán procesados en la cuenta correspondiente.
La extensión es auto-contenida: las operaciones agregadas no refieren a operaciones existentes en el contrato. Por supuesto, las nuevas operaciones pueden referenciarse entre ellas, como muestra el ejemplo, donde *atender*() invoca a *procesar*().

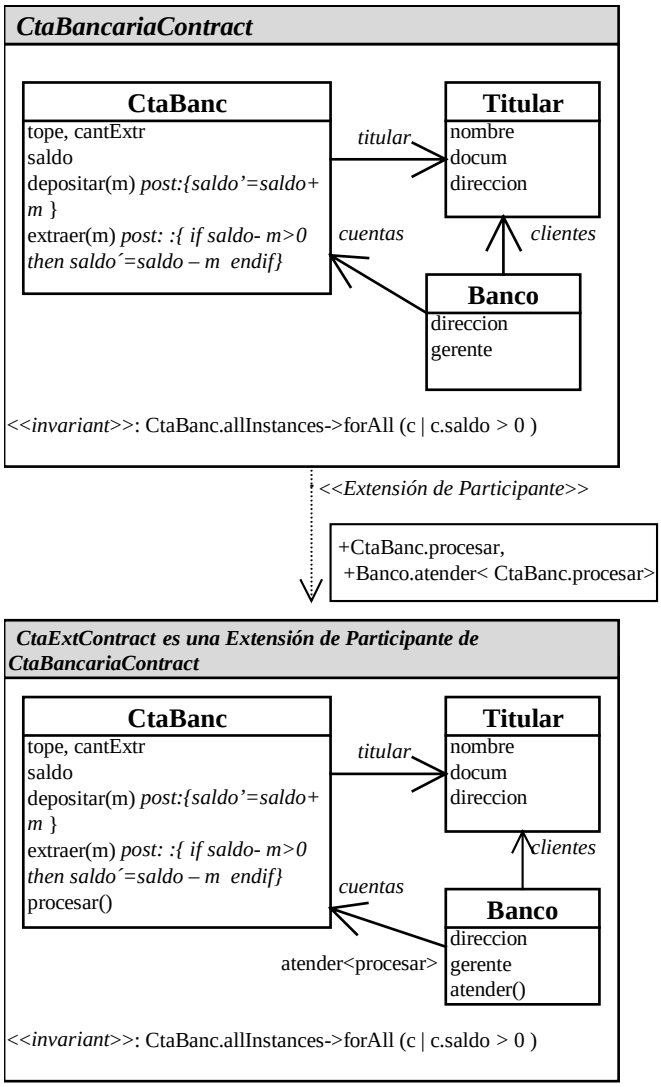


*Fig. 3.1 Un ejemplo de Extensión de Participante*

*Definición 1. Modificador*

Un Modificador de Extensión de Participante M es un conjunto de pares (p,int) cada uno consistente de un nombre de participante *p* y una interfaz int.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es extendible de participante por un modificador de extensión de participante M si para cada par (*p*,*int*) en M:

1. *p* es un nombre de participante en R
2. ningún nombre de operación en *int* aparece en la interfaz del participante *p* en R;
3. para cada invocación de operación *a.m* en una cláusula de especialización en int:
   (a) *a* es un nombre de conocimiento en la cláusula de conocimiento de *p* en R;
   (b) si *a* sobre *p* refiere a q entonces m es una operación en la interfaz de q en M.

*Definición 3. Resultado de la Extensión de Participante*

Si un contrato de reuso R es extendible de participante por un modificador M, entonces el contrato de reuso R' es la extensión de participante de R por M y escribimos
R'= Extp(R, M), donde:
1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada (*p, int*) en M: R' contiene un participante con el mismo nombre y cláusula de conocimiento que *p* en R y que contiene todas las operaciones de p, más las mencionadas en int.
3. El invariante de R' es el invariante de R.

### *3.2 Extensión sintáctica de contexto: Extc*

Así como se agregan operaciones a participantes existentes, pueden introducirse nuevos participantes al contrato.
La extensión de contexto, como la de participante, debe ser auto-contenida. Las cláusulas de conocimiento de estos nuevos participantes pueden referir solamente a participantes que son agregados a través de la misma extensión. Un modificador de extensión de contexto es un contrato de reuso disjunto del contrato que está extendiendo. Para agregar dependencias entre estas dos partes se debe aplicar refinamiento sintáctico de contexto.

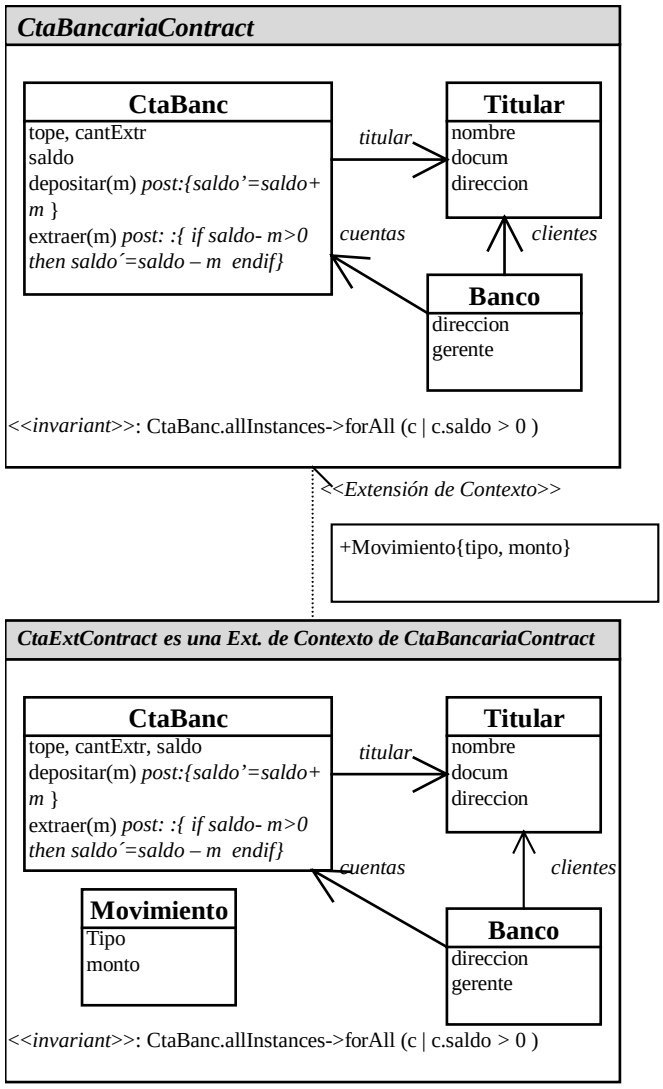


*Fig.3.2 Un ejemplo de Extensión de Contexto*

La figura 3.2 muestra una Extensión de Contexto del contrato *CtaBancaria*. El participante Movimiento es agregado al contrato, con la finalidad de poder registrar las transacciones que se realizan en cada cuenta.

*Definición 1. Modificador*

Un Modificador de extensión sintáctica de Contexto M es un conjunto de participantes.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es extendible sintácticamente de contexto por un modificador de extensión sintáctica de contexto si para cada participante *p* en M:
*p* es un nombre diferente a todos los nombres de participantes en R.

*Definición 3. Resultado de la Extensión de Contexto*

Si el contexto de un contrato de reuso R es extendible por un modificador M, entonces el contrato de reuso R' es la extensión de contexto de R por M y escribimos R' = Extc(R, M), donde:
1.R' contiene todos los participantes de R y todos los participantes de M.
2. El invariante de R' es el invariante de R.

### *3.3 Cancelación sintáctica de participante: Canp*

La operación opuesta a la extensión es la cancelación. Como ocurre con las extensiones, las cancelaciones deben contenerse a sí mismas. Esto significa que cuando una operación se cancela, todas las operaciones referidas en su cláusula de especialización necesitan ser canceladas. Cuando se desea cancelar una operación, pero no las operaciones que ella refiere, previamente debe realizarse un *coarsening*. Por otro lado, la operación en cuestión no debe aparecer en las post-condiciones ni invariante del contrato para poder ser removida.

La figura 3.3 muestra una Cancelación de Participante del contrato *CtaBancaria*. Supongamos que queremos eliminar la operación *procesar*(). Como *procesar*() es invocada por *atender*(), ésta debe también eliminarse, para que el operador pueda aplicarse.

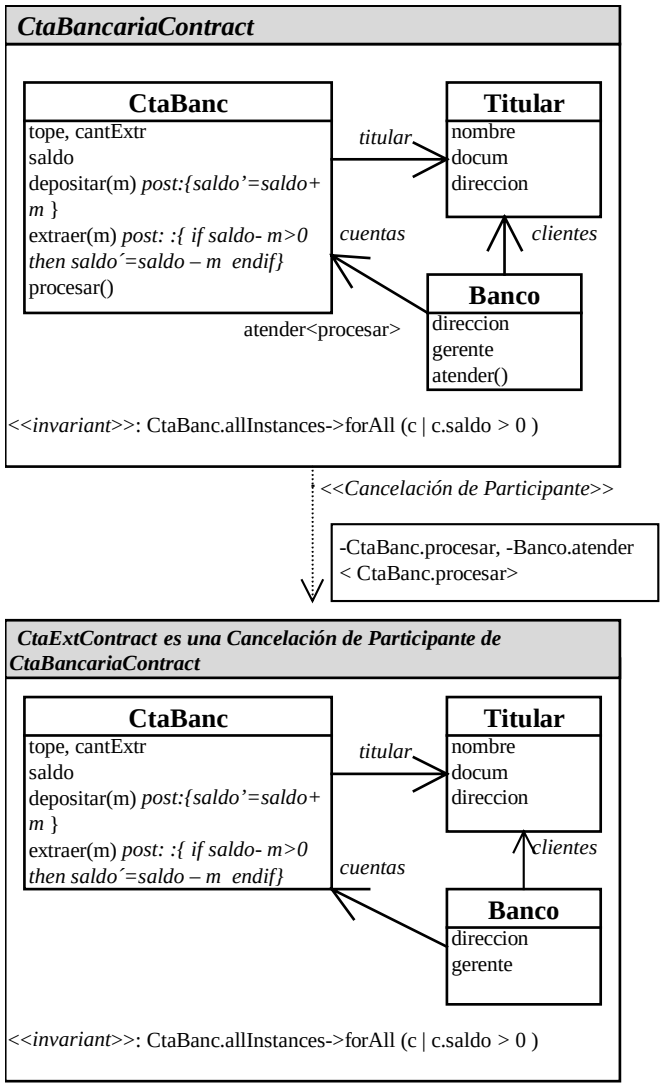


*Fig.3.3 Un ejemplo de Cancelación de Participante*

*Definición 1. Modificador*

Un Modificador de cancelación sintáctica de Participante M es un conjunto de pares (*p,int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.* La interfaz *int* consistirá de un conjunto de nombres de operaciones y una cláusula de especialización asociada cada una.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es cancelable sintácticamente de participante por un modificador de cancelación sintáctica de participante M si para cada par (*p,int*) en M:
1. *p* es un nombre de participante en R y cada operación en *int* es idéntica a una operación de este participante en R.
2. para todas las operaciones *m*, *n* y todos los participantes *q* en R, tales que *m* de *q* invoca a *n* de *p*: si *n* aparece en *int* , entonces *m* aparece en *int* asociada a *q.*
3. para toda operación *n,m* y todo participante *q* en R, si *n* aparece en *int* y ocurre en la post-condición de *m* entonces *m* aparece en *int.*

4. ninguna operación en *int* de *p* ocurre en el invariante de R.

*Definición 3. Resultado de la Cancelación sintáctica de Participante*

Si un contrato de reuso R es cancelable de participante por un modificador M, entonces el contrato de reuso R' es la cancelación de participante de R por M y escribimos R'=Canp(R,M) donde:

1. R' contiene todos los participantes de R que no estan mencionados en M;
2. para cada (*p*, *int*) en M : R' contiene un participante con el mismo nombre y cláusula de conocimiento que *p* en R y que contiene todas las operaciones de *p* con la misma cláusula de especialización y post-condición en R, excepto aquellas que están en *int*.
3. El invariante de R' es el invariante de R.

### *3.4 Cancelación sintáctica de contexto: Canc*

Nuevamente, como ocurre con la extensión de contexto, la cancelación debe contenerse a sí misma. Esto significa que debemos verificar que no se remuevan participantes que son referenciados por otros. Por otro lado, el participante en cuestión no debe aparecer en las post-condiciones ni invariante del contrato para poder ser removido.

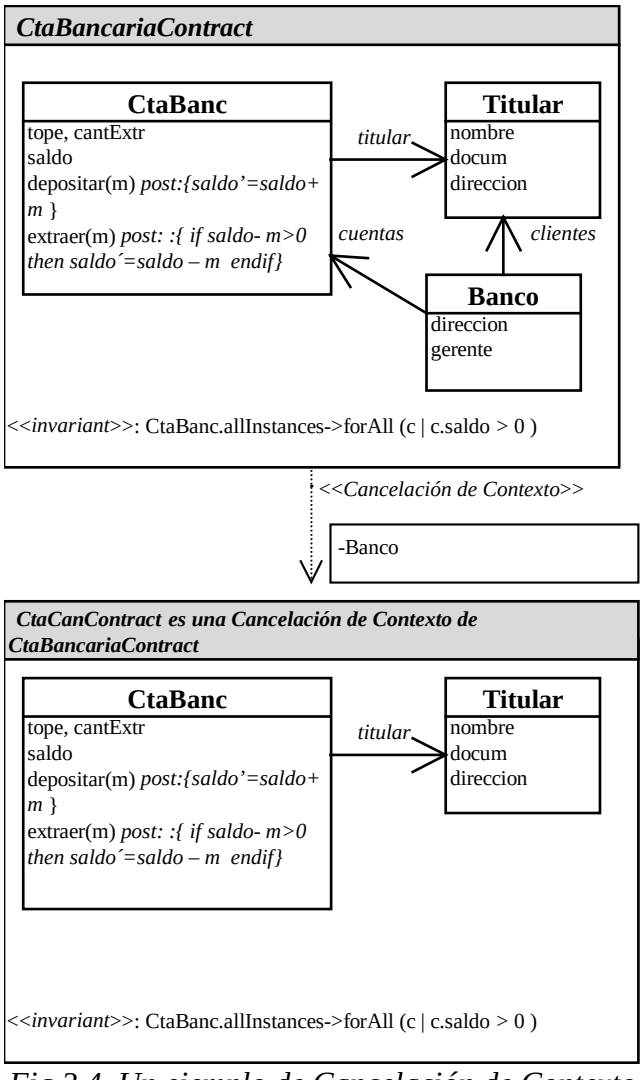


*Fig.3.4 Un ejemplo de Cancelación de Contexto*

La figura 3.4 muestra una Cancelación de Contexto del contrato *CtaBancaria.* Supongamos que queremos eliminar al participante Banco debido a que ya no deseamos que el control de las transacciones y los clientes estén a su cargo. Como parte del efecto de esta cancelación, las relaciones de conocimiento *cuentas* con CtaBanc y *clientes* con Titular son también canceladas.

*Definición 1. Modificador*

Un Modificador de cancelación sintáctica de Contexto M es un conjunto de participantes.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es cancelable sintácticamente de contexto por un modificador de cancelación sintáctica de contexto M si para cada participante *p* en M:

1. *p* es un nombre de participante en R.
2. *p* no ocurre en la cláusula de conocimiento de un participante en R que no esté en M.
3. Para todo participante *q* y toda operación *n* en R, si *p* ocurre en la post-condición de *n* entonces *q* aparece en M.
4. *p* no ocurre en el invariante de R

*Definición 3. Resultado de la Cancelación sintáctica de Contexto*

Si un contrato de reuso R es cancelable de contexto por un modificador M, entonces el contrato de reuso R' es la cancelación de contexto de R por M y escribimos R'= Canc(R, M), donde:

1. R' contiene todos los participantes de R que no estan mencionados en M;
2. El invariante de R' es el invariante de R.

## *3.5 Refinamiento sintáctico de Participante: Refp*

Cuando resulta necesario agregar dependencias entre los elementos de un contrato, aparecen los operadores de *refinamiento***.** El operador Refp agrega nuevas invocaciones a operaciones ya existentes en el contrato, en una determinada posición, o en paralelo a una posición.

En la figura 3.5 se muestra un Refinamiento de Participante en el contrato *CtaBancaria.* Las operaciones *atender()* y *procesar()* fueron previamente agregadas al contrato pero no se habían relacionado. Aquí se agrega una invocación de la operación *procesar*() a la operación *atender*().

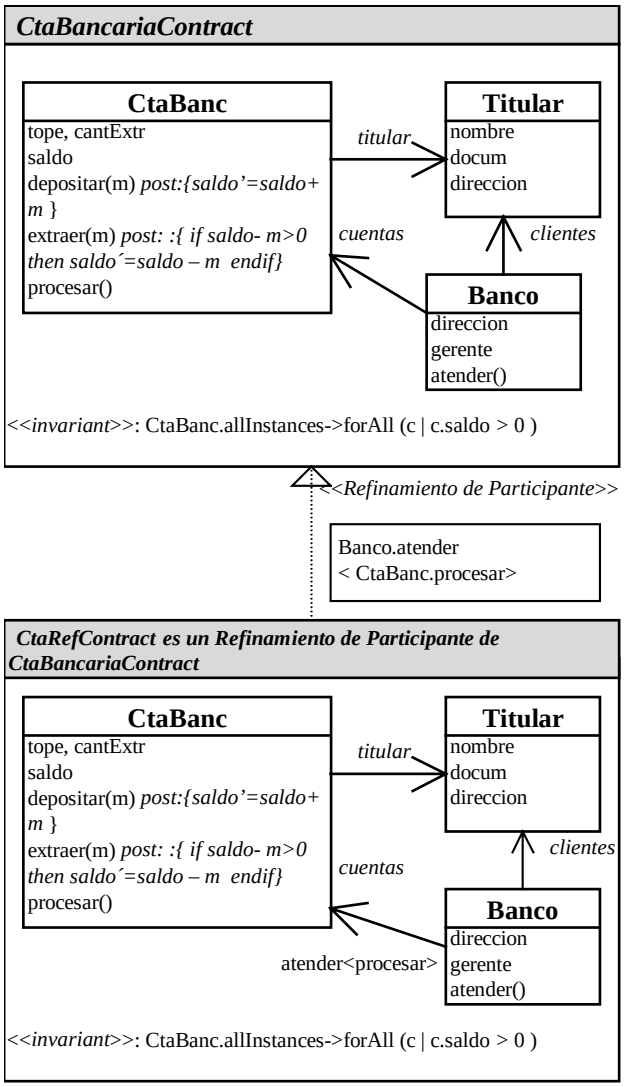


*Fig.3.5 Un ejemplo de Refinamiento de Participante*

*Definición 1. Modificador*

Un Modificador de Refinamiento sintáctico de Participante M es un conjunto de pares (*p,int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.*
Una interfaz es un conjunto de nombres de operación con una cláusula de especialización asociada, que es la nueva cláusula (refinada) para la operación.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es refinable sintácticamente de participante por un modificador de refinamiento de participante M si para cada par (*p,int*) en M:
1. *p* es un nombre de participante en R
2. para cada nombre de operación *m* en *int*: *m* aparece en la interfaz del participante *p* en R, cada operación en la cláusula de *m* en *int* está en la interfaz de participantes relacionados con *p* y la cláusula de especialización de *m* en *int* refina a la cláusula de especialización de *m* en *p* de R.

El significado intuitivo de *refina* es que la nueva cláusula refina a la original *si* le agrega una operación en secuencia o paralelo respetando el orden entre las operaciones que ya existían.

*Definición 3. Resultado del Refinamiento sintáctico de Participante*

Si un contrato de reuso R es refinable sintácticamente de participante por un modificador M, entonces el contrato de reuso R' es el refinamiento de participante de R por M y escribimos R' = Refp(R, M) , donde:

1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada (*p, int*) en M: R' contiene un participante
(a) con nombre *p* y la misma cláusula de conocimiento que *p* en R
(b) que contiene todas las operaciones de *p* en R no mencionadas en *int*.
(c) que contiene todas las operaciones de *int* con la cláusula de especialización dada en *int* y como post-condición, la existente en R.
3. El invariante de R' es el invariante de R.

### *3.6 Coarsening sintáctico de Participante: Coap*

La operación inversa al refinamiento es llamada *coarsening* y tiene como finalidad eliminar dependencias entre elementos del contrato**.** El operador Coap elimina invocaciones a operaciones.

Si queremos, por ejemplo que la operación *atender*() resuelva sus requerimientos directamente sin invocar a la operación *procesar*(), debemos eliminar su invocación. La figura 3.6 muestra un *Coarsening* de Participante en el contrato *CtaBancaria;* la operación *procesar()* es eliminada de la cláusula de especialización de *atender*().

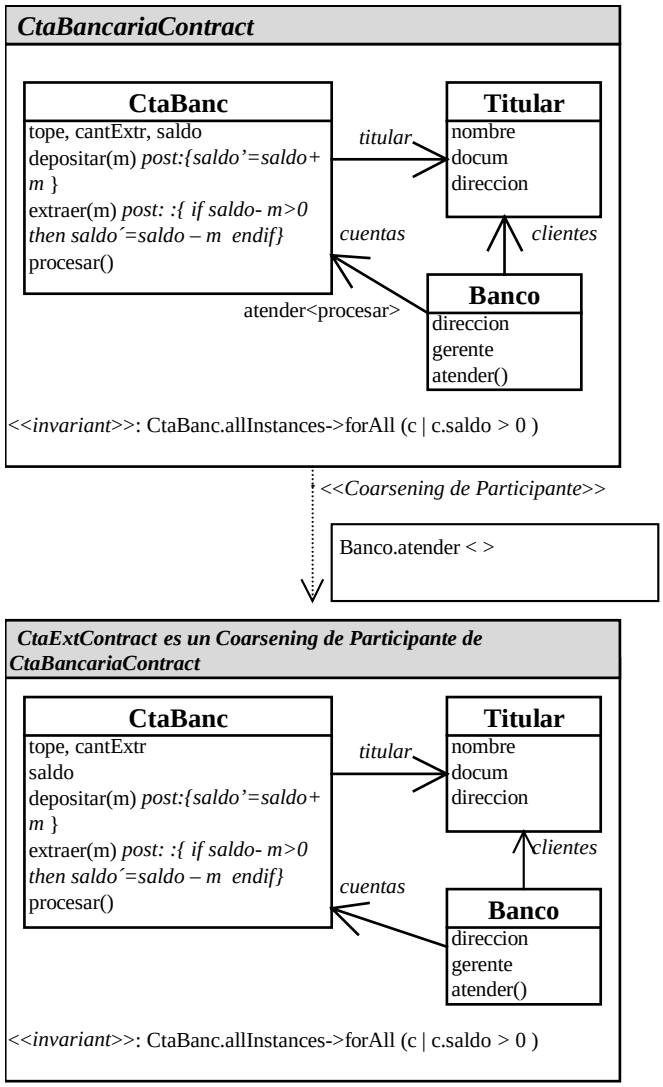


*Fig.3.6 Un ejemplo de Coarsening de Participante*

*Definición 1. Modificador*

Un Modificador de Coarsening sintáctico de Participante M es un conjunto de pares (*p,int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.*
Una interfaz es un conjunto de nombres de operación con una cláusula de especialización asociada, que es la nueva cláusula de especialización (debilitada) para la operación.

*Definición 2. Aplicabilidad*

Si un contrato de reuso R es *coarsenable* sintácticamente de participante por un modificador de *coarsening* sintáctico de participante M si para cada par (*p,int*) en M:
1. *p* es un nombre de participante en R
2. para cada nombre de operación *m* en *int*: *m* aparece en la interfaz del participante p en R y la cláusula de especialización de *m* de p en R *refina* a la cláusula de especialización de *m* en *int*.

*Definición 3. Resultado del Coarsening sintáctico de Participante*

Un contrato de reuso R es *coarsenable* sintácticamente de participante por un modificador M, entonces el contrato de reuso R' es el coarsening sintáctico de participante de R por M y escribimos R' = Coap(R, M), donde:

1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada (*p*, *int*) en M: R' contiene un participante
(a) con nombre *p* y la misma cláusula de conocimiento que *p* en R
(b) que contiene todas las operaciones de *p* en R no mencionadas en *int*.
(c) que contiene todas las operaciones de *int* con cláusula de especialización dada en *int* y como post-condición, la existente en R.
3. El invariante de R' es el invariante de R.

### *3.7 Refinamiento sintáctico de Contexto: Refc*

Así como un contrato puede ser refinado agregando nuevas invocaciones a operaciones, puede serlo también adicionando nuevas relaciones de conocimiento. El refinamiento sintáctico de contexto agrega nuevas relaciones de conocimiento entre participantes ya existentes en el contrato.

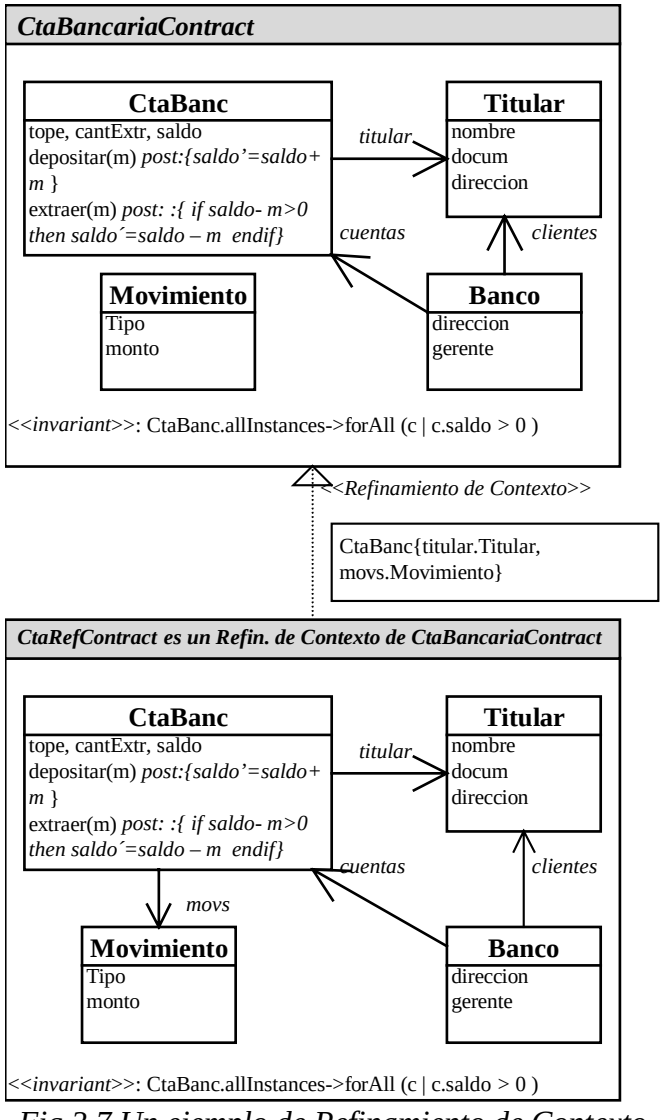


*Fig.3.7 Un ejemplo de Refinamiento de Contexto*

En la figura 3.7 se muestra un Refinamiento de Contexto del contrato *CtaBancaria;* los participantes CtaBanc y Movimiento se ligan a través de la relación de conocimiento *movs*, por la cual cada cuenta registrará y conocerá sus transacciones.

*Definición 1. Modificador*

Un Modificador de Refinamiento sintáctico de Contexto M es un conjunto de pares (*p,con*) cada uno consistente de un nombre de participante *p* y una cláusula de conocimiento (refinada) para el participante.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es refinable sintácticamente de contexto por un modificador de refinamiento de contexto M si para cada par (*p,con)* en M:
1. *p* es un nombre de participante en R
2. *con* incluye a la cláusula de conocimiento de *p* en R
3. *con* contiene relaciones de conocimiento *a.q*, donde *q* es un nombre de participante en R.

*Definición 3.Resultado del Refinamiento sintáctico de Contexto*

Si el contexto de un contrato de reuso R es refinable por un modificador M, entonces el contrato de reuso R' es el refinamiento sintáctico de contexto de R por M y escribimos R'=Refc(R, M) donde:
1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada par (*p, con*) en M: R' contiene un participante con el mismo nombre e interfaz que *p* en R y como cláusula de conocimiento, la dada en *con.*
3. El invariante de R' es el invariante de R.

## *3.8 Coarsening sintáctico de Contexto: Coac*

Así como en un contrato se pueden eliminar invocaciones a operaciones, pueden eliminarse también relaciones de conocimiento. El coarsening sintáctico de contexto elimina relaciones de conocimiento entre participantes ya existentes en el contrato.

Si ya no queremos que se registre cada transacción sobre las cuentas, podemos eliminar la relación entre CtaBanc y Movimiento. La figura 3.8 muestra un Coarsening de Contexto del contrato *CtaBancaria;* la relación de conocimiento *movs* entre los participantes Movimiento y CtaBanc es eliminada.

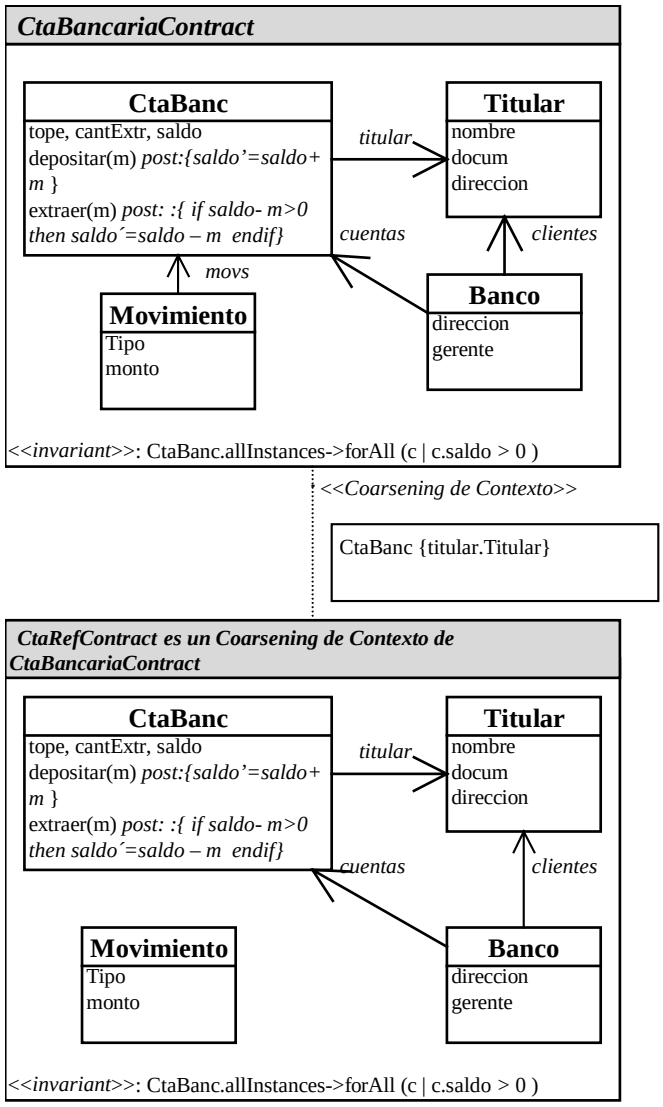


*Fig.3.8 Un ejemplo de Coarsening de Contexto*

*Definición 1. Modificador*

Un Modificador de Coarsening sintáctico de Contexto M es un conjunto de pares (*p,con)* cada uno consistente de un nombre de participante *p* y una cláusula de conocimiento (debilitada) para el participante.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es *coarsenable* sintácticamente de contexto por un modificador de coarsening de contexto M si para cada par (*p, con*) en M:
1. *p* es un nombre de participante en R
2. *con* está incluída en la cláusula de conocimiento de *p* en R
3. para toda relación de conocimiento *a.q* que esté en la cláusula de conocimiento de *p* en R *y* no esté en *con*:

- ninguna operación en *p* tiene a *a* en su cláusula de especialización.
- *a.q* no ocurre en la post-condición de ninguna operación de *p.*
- *a.q* no ocurre en el invariante de R.

*Definición 3. Resultado del Coarsening sintáctico de Contexto*

Si el contexto de un contrato de reuso R es *coarsenable* por un modificador M, entonces el contrato de reuso R' es el coarsening de contexto de R por M y escribimos R' = Coac(R, M), donde:
1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada par (*p, con*) en M: R' contiene un participante con el mismo nombre e interfaz que *p* en R y *con* como cláusula de conocimiento.
3. El invariante de R' es el invariante de R.

# Capítulo 4: Operadores de Reuso Semántico

Como vimos en el Capítulo 1, sección 1.3, resulta interesante integrar semántica de comportamiento a un contrato de reuso, como así también proveer mecanismos para mostrar su evolución. Estos nuevos operadores de reuso semántico que definimos en el presente capítulo, documentan la evolución semántica del contrato de reuso. Los operadores semánticos conservan la sintaxis del contrato de reuso, es decir no provocarán modificaciones estructurales de participantes, ni de relaciones de conocimiento, ni de operaciones e invocaciones entre ellas. Respecto a la semántica de comportamiento, un contrato de reuso tiene dos elementos: el invariante del contrato y la post-condición de sus operaciones. Cada uno de ellos puede ser refinado, debilitado o redefinido, lo cual lleva a seis operadores:

| Operador | Descripción | Significado |
|---|---|---|
| Red | Redefinición semántica de Participante | redefinir la post-condición de una operación del part. |
| Refsp | Refinamiento semántico de Participante | refinar la post-condición de una operación del part. |
| Coasp | *Coarsening* semántico de Participante | debilitar la post-condición de una operación del part. |
| Redc | Redefinición semántica de Contexto | redefinir el invariante del contrato de reuso |
| Refsc | Refinamiento semántico de Contexto | refinar el invariante del contrato de reuso |
| Coasc | *Coarsening* semántico de Contexto | debilitar el invariante del contrato de reuso |

Por definición de contrato de reuso con semántica de comportamiento bien formado enunciada anteriomente, cada operación en el contrato debe *preservar* el invariante. Como consecuencia, al aplicar operadores de reuso semántico sobre un contrato de reuso se debe verificar que todas las operaciones del contrato, mediante sus post-condiciones, continúen satisfaciendo el predicado invariante.

En las siguientes secciones se incluyen las definiciones completas de los operadores de reuso semántico; su notación matemática puede verse en el modelo para contratos de reuso y operadores presentado en el Capítulo 7.

## *4.1 Refinamiento semántico de participante: Refsp*

Cuando se necesita refinar comportamiento de una operación, puede enriquecerse su post-condición, adicionándole expresiones, de manera tal que la operación siga preservando el invariante del contrato. Al operador semántico que permite documentar este cambio lo llamamos Refinamiento semántico de participante. El refinamiento semántico de una operación no altera, sintácticamente, la definición del contrato de reuso base ya que no se

agregan ni se quitan invocaciones originales. Si esto fuera necesario, se hará mediante el refinamiento sintáctico.

Supongamos que con el objetivo de obtener cuentas específicas, por ejemplo cajas de ahorro, se aplica sobre el contrato *CtaBancaria* descripto en el capítulo 2, un operador de Extensión sintáctica de Participante y de Refinamiento sintáctico de Participante que agrega y refina operaciones del participante CtaBanc. Se genera así el contrato CAhorro.
En la Figura 4.1 se muestra un Refinamiento Semántico de Participante, donde se refina la post-condición de la operación *extraer*() de CAhorro para indicar que cada vez que se ejecute esta operación se debe cumplir que la cantidad de extracciones se mantenga menor a un tope. Este refinamiento es válido dado que la nueva post-condición permite que se siga preservando el invariante.
En esta figura y en las siguientes de este capítulo se muestra la parte del contrato que interesa a las modificaciones.

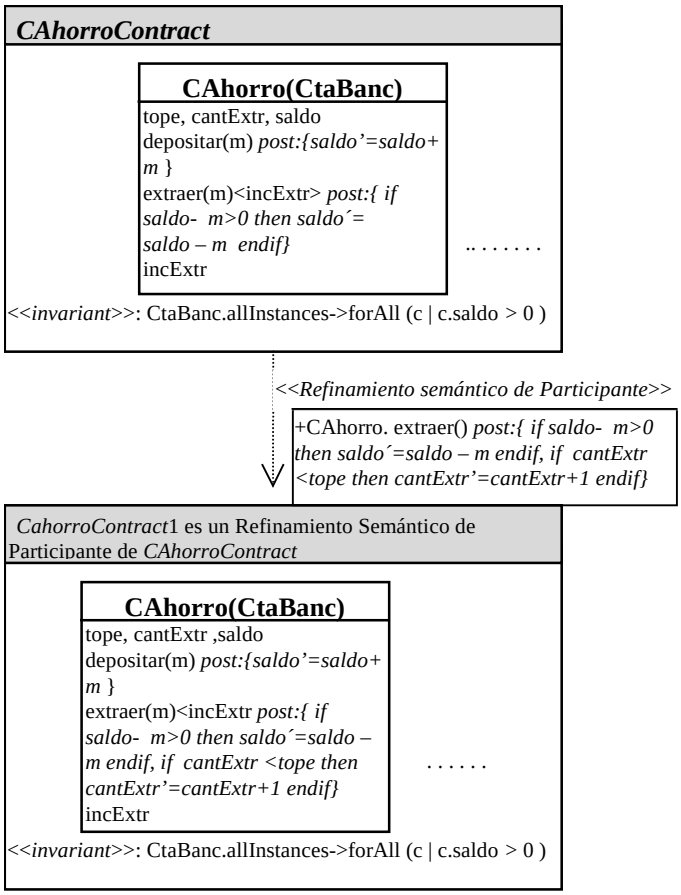


*Fig. 4.1. Un ejemplo de Refinamiento Semántico de Participante*

*Definición 1 Modificador*

Un Modificador de Refinamiento semántico de Participante M es un conjunto de pares (*p,int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.* La interfaz

*int* consistirá de un conjunto de nombres de operaciones y una post-condición asociada cada una.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es refinable semánticamente de participante por un modificador de refinamiento semántico de participante M si para cada par (*p*,*int*) en M:
1. p es un nombre de participante en R
2. para cada nombre de operación *m* en *int:*
a) *m* aparece en la interfaz del participante *p* en R , la post-condición de *m* en *int* es un conjunto de expresiones OCL bien formadas que involucra operaciones y participantes de R
b) en la post-condición en int no se repite el mismo término primado y se incluye a la post-condición en R
c) la post-condición en int hace que *m* preserve el invariante de R.

*Definición 3. Resultado del Refinamiento semántico de Participante*

Si un contrato de reuso R es refinable semánticamente de participante por un modificador M, entonces el contrato de reuso R' es el refinamiento de participante de R por M y escribimos R'= Refsp(R, M), donde:

1. R' contiene todos los participantes de R que no estan mencionados en M;
2. para cada (*p, int*) en M: R' contiene un participante
(a) con nombre p y la misma cláusula de conocimiento que p en R
(b) que contiene todas las operaciones de p en R no mencionadas en *int*.
(c) que contiene todas las operaciones de *int* con la cláusula de especialización idéntica en R y como post-condición, la post-condición dada en int.
3. El invariante de R' es el invariante de R.

### *4.2 Refinamiento semántico de contexto: Refsc*

El refinamiento semántico del contexto del contrato, será el refinamiento de la invariante del contrato de reuso, donde la invariante resultado debe implicar la del contrato base.

Supongamos que en el invariante del contrato base que aparece en la figura 4.2 se quiere indicar que todas las instancias de CAhorro deben mantener su saldo mayor que 0 y la cantidad de extracciones menor a un tope. Esto puede hacerse mediante un Refinamiento Semántico de Contexto sobre este contrato, que refina al invariante. Este refinamiento es válido dado que el nuevo invariante implica al anterior y se mantiene consistente con las post-condiciones de operaciones del contrato.

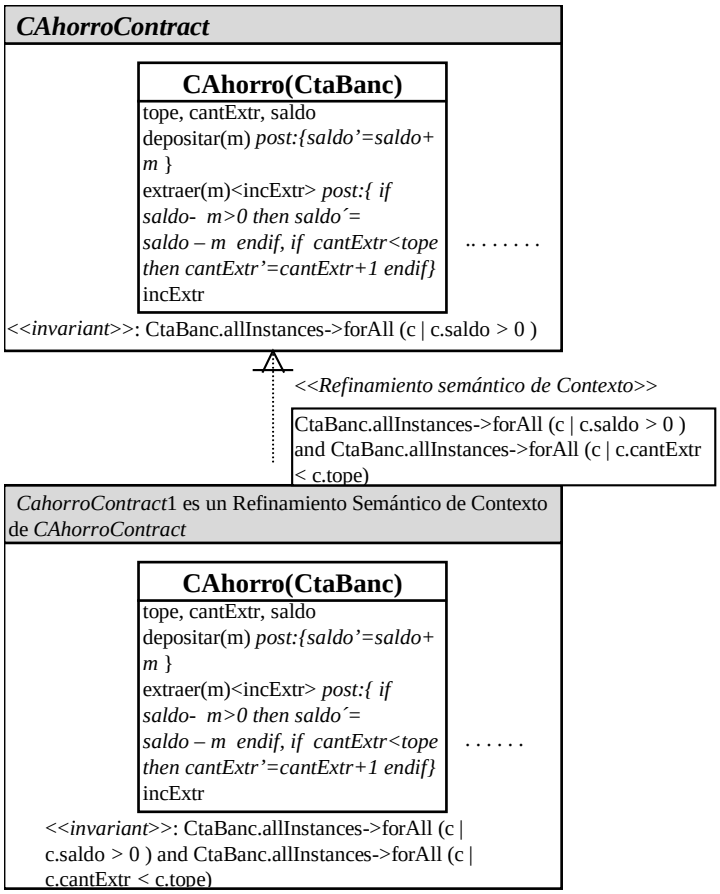


*Fig 4.2. Un ejemplo de Refinamiento Semántico de Contexto*

*Definición 1 Modificador.*

Un Modificador de Refinamiento semántico de Contexto M es una fórmula OCL.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es refinable semánticamente de contexto por un modificador de refinamiento de contexto M si

1. el invariante es una expresión OCL bien formada e involucra operaciones y participantes de R.
2. si el invariante en M es válido se puede deducir que el invariante en R es válido.
3. cada operación en R preserva el invariante en M

*Definición 3. Resultado del Refinamiento semántico de Contexto*

Si un contrato de reuso R es refinable semánticamente de contexto por un modificador M, entonces el contrato de reuso R' es el refinamiento semántico de contexto de R por M y escribimos R' = Refsc(R,M), donde:

1. R' contiene todos los participantes de R, con el mismo nombre y la misma cláusula de conocimiento que en R, que contienen todas las operaciones en R con la cláusula de especialización y post-condición idénticas en R .

2. El invariante de R’ es el invariante en M.

### *4.3 Coarsening semántico de participante: Coasp*

Cuando se necesita debilitar el comportamiento de una operación puede modificarse su post-condición, eliminando alguna de sus expresiones, de manera tal que la operación continúe preservando el invariante. Al operador semántico que permite documentar este cambio lo llamamos Coarsening semántico de participante. El coarsening semántico de una operación no altera, sintácticamente, la definición del contrato de reuso base ya que no se quitan invocaciones originales. Si esto fuera necesario, se hará mediante el coarsening sintáctico.

En la Figura 4.3 se muestra un ejemplo de Coarsening semántico de Participante donde se debilita la post-condición de la operación *extraer*() del participante CAhorro del contrato base, con el fin de que en las cuentas ya no se controle la cantidad de extracciones cada vez que se ejecute esta operación. Este *Coarsening* es aplicable pues la operación preserva el invariante del contrato.

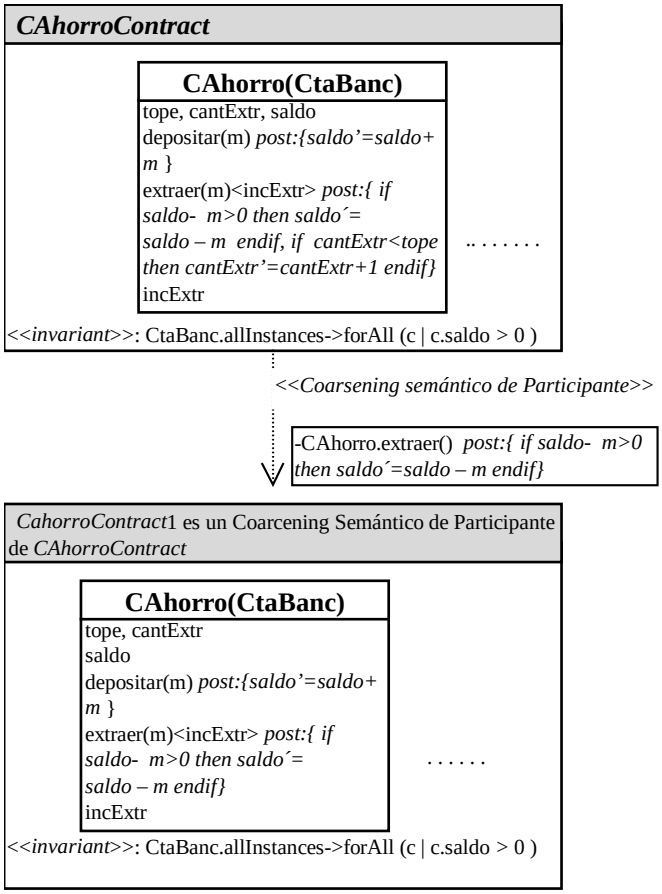


*Fig. 4.3. Un ejemplo de Coarsening Semántico de Participante*

*Definición 1.Modificador*

Un Modificador de coarsening semántico de Participante M es un conjunto de pares (*p,int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.* La interfaz *int* consistirá de un conjunto de nombres de operaciones y una post-condición asociada cada una.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es *coarsenable* semánticamente de participante por un modificador de coarsening semántico de participante M si para cada par (*p,int*) en M:
1. *p* es un nombre de participante en R
2. para cada nombre de operación *m* en *int*:
a) m aparece en la interfaz del participante *p* en R y la post-condición de *m* en *int* está bien formada, involucra operaciones de participantes de R y está incluída en la post-condición en R.
b) la post-condición en int hace que *m* preserve el invariante de R.

*Definición 3. Resultado del Coarsening semántico de Participante*

Si un contrato de reuso R es *coarsenable* semánticamente de participante por un modificador M, entonces el contrato de reuso R' es el coarsening semántico de participante de R por M y escribimos R' = Coasp(R, M), donde:

1. R' contiene todos los participantes de R que no estan mencionados en M;
2. para cada (*p, int*) en M: R' contiene un participante
(a) con nombre *p* y la misma cláusula de conocimiento que *p* en R
(b) que contiene todas las operaciones de *p* en R no mencionadas en *int*.
(c) que contiene todas las operaciones de *int* con la cláusula de especialización idéntica en R y como post-condición, la post-condición dada en int.
3. El invariante de R' es el invariante de R

### *4.4 Coarsening semántico de contexto: Coasc*

El coarsening semántico del contexto del contrato, consistirá en debilitar la invariante del contrato de reuso, de manera que la invariante base implique la del contrato resultante.

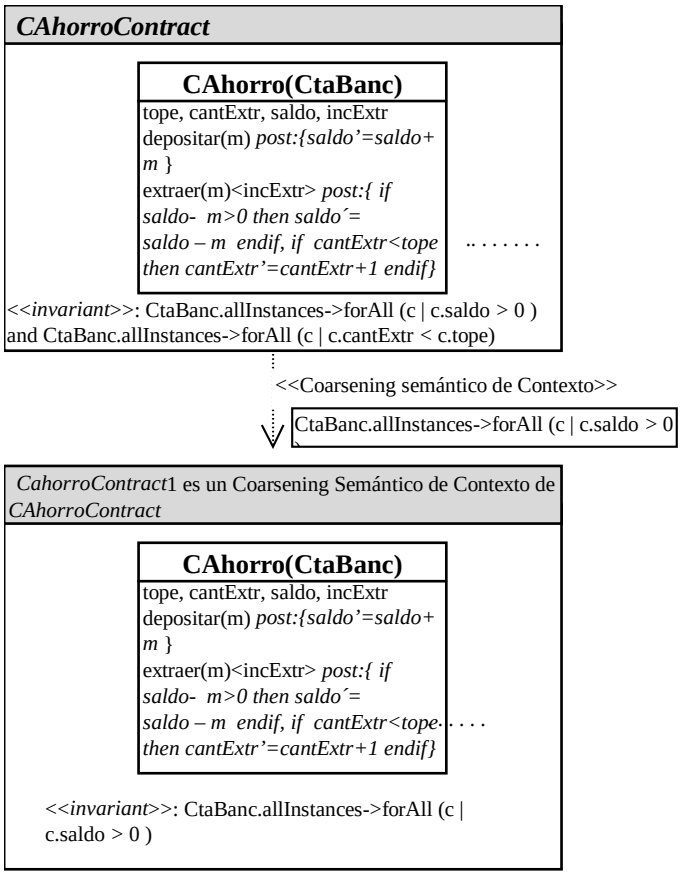


*Fig 4.4 Un ejemplo de Coarsening Semántico de Contexto*

Un ejemplo de Coarsening semántico de Contexto se muestra en la Figura.4.4 donde se debilita el invariante del contrato base, con el fin de que en las cuentas ya no se controle la cantidad de extracciones. Esta operación es aplicable pues el invariante del contrato base implica al nuevo.

*Definición 1.Modificador*

Un Modificador de Coarsening semántico de Contexto M es una fórmula OCL

*Definición 2. Aplicabilidad*

Un contrato de reuso R es *coarsenable* semánticamente de contexto por un modificador de coarsening de contexto M si

1. el invariante de M es una expresión OCL bien formada e involucra operaciones y participantes de R.
2. si el invariante en R es válido se puede deducir que el invariante en M es válido.
3. cada operación en R preserva el invariante en M

*Definición 3. Resultado del Coarsening semántico de Contexto*

Si un contrato de reuso R es *coarsenable* semánticamente de contexto por un modificador M, entonces el contrato de reuso R' es el coarsening semántico de contexto de R por M y escribimos R' = Coasc(R, M), donde:

1. R’ contiene todos los participantes de R, con el mismo nombre y la misma cláusula de conocimiento que en R, que contienen todas las operaciones en R con la cláusula de especialización y post-condición idénticas en R .
2. El invariante de R’ es el invariante dado en M.

### *4.5 Redefinición semántica de participante: Red*

Cuando se necesita sobreescribir la post-condición de una operación resulta necesario un operador que permita documentar este cambio: Redefinición semántica de participante. La redefinición de una operación no altera, sintácticamente, la definición del contrato de reuso base ya que no se agregan ni se quitan invocaciones originales, sino que indica un cambio de su post-condición, que debe seguir preservando el invariante del contrato.

La Figura 4.5 muestra una Redefinición Semántica de Participante donde la post-condición de la operación *extraer*() del participante CAhorro es redefinida con el fin de indicar que la cantidad de extracciones, que debía ser menor que 5 al finalizar la ejecución de esa operación, se mantenga ahora menor que 10. Este operador es aplicable pues la nueva post-condición es consistente con el invariante.

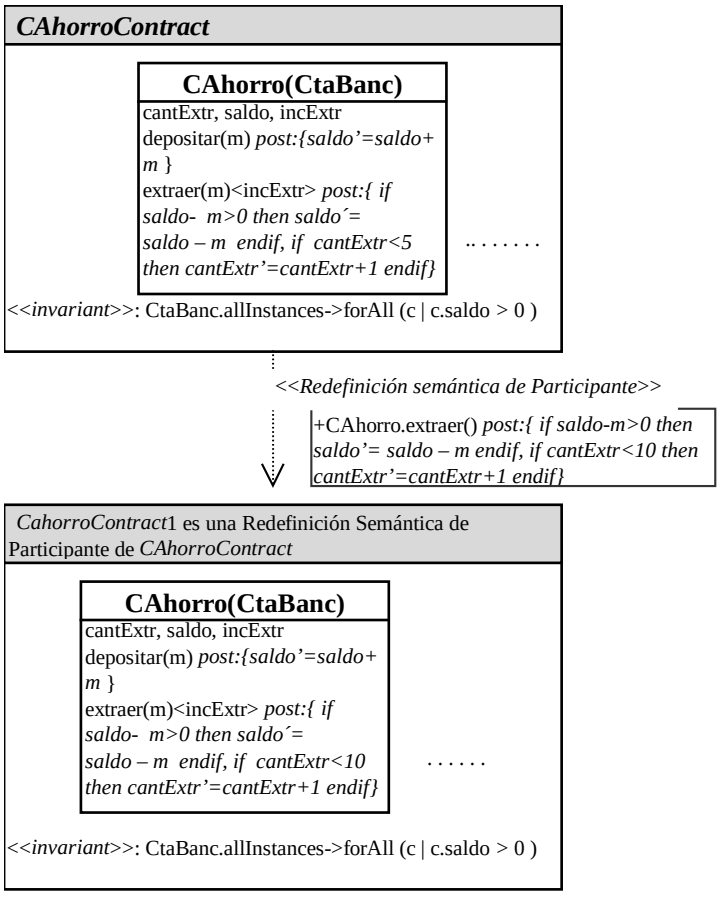


*Fig. 4.5 Un ejemplo de Redefinición Semántica de Participante*

*Definición 1.Modificador*

Un modificador de Redefinición semántica de Participante M es un conjunto de pares (*p*,*int*) cada uno consistente de un nombre de participante *p* y una interfaz *int.* La interfaz *int* consistirá de un conjunto de nombres de operaciones y una post-condición asociada cada uno.

*Definición 2. Aplicabilidad*

Un contrato de reuso R es redefinible semánticamente de participante por un modificador de redefinición semántica de participante M si para cada par (*p*,*int*) en M:
1. p es un nombre de participante en R
2. para cada nombre de operación *m* en *int*: *m* aparece en la interfaz del participante *p* en R y la post-condición de *m* en *int* está bien formada e involucra operaciones que existen en R.
3. la post-condición de *m* en *int* hace que se preserve el invariante de R.

*Definición 3. Resultado de la Redefinición Semántica de Participante*

Si un contrato de reuso R es redefinible semánticamente de participante por un Modificador M, entonces el contrato de reuso R' es la redefinición de participante de R por M y escribimos R' = Red (R, M), donde:
1. R' contiene todos los participantes de R que no están mencionados en M;
2. para cada (*p, int*) en M: R' contiene un participante
(a) con nombre p y la misma cláusula de conocimiento que p en R
(b) que contiene todas las operaciones de p en R no mencionadas en int.
(c) que contiene todas las operaciones de *int* con la cláusula de especialización idéntica en R y como post-condición, la condición dada en int.
3. El invariante R' es el invariante de R

### *4.6 Redefinición semántica de contexto: Redc*

La redefinición también puede darse a nivel contexto, pues el desarrollador puede necesitar sobreescribir la invariante del contrato de reuso.

La Figura 4.6 muestra una Redefinición Semántica de Contexto. El invariante del contrato base es redefinido para indicar ahora que todas las instancias de CAhorro puedan girar en descubierto hasta $100.Esta redefinición es válida pues el nuevo invariante es consistente con las post-condiciones del contrato.

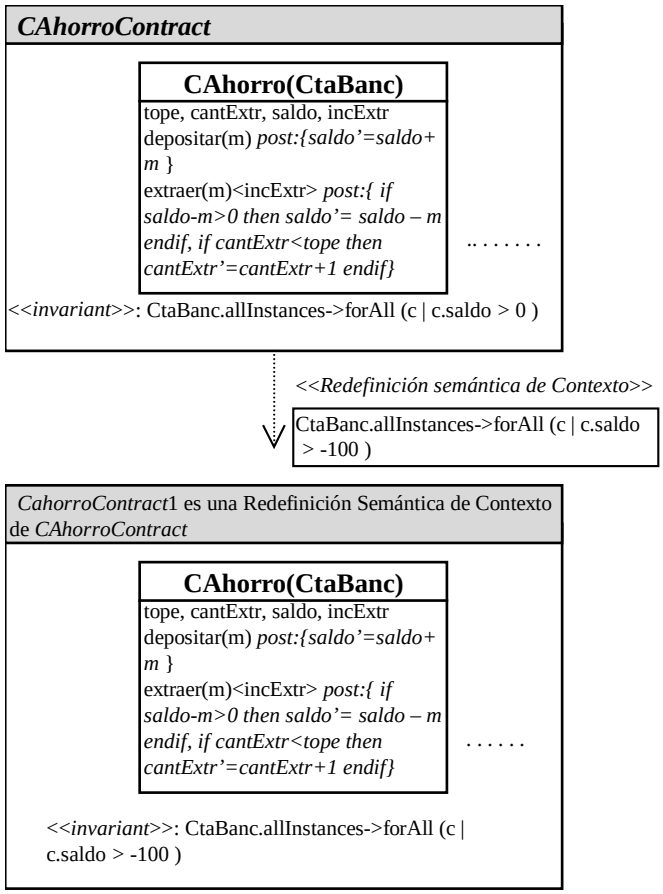


*Fig 4.6 Un ejemplo de Redefinición Semántica de Contexto*

*Definición 1. Modificador*

Un Modificador de Redefinición semántica de Contexto M es una fórmula OCL

*Definición 2. Aplicabilidad*

Un contrato de reuso R es redefinible semánticamente de contexto por un modificador de redefinición de contexto M si
1. el Invariante en M es una expresión OCL bien formada e involucra participantes y operaciones que existen en R.
2. cada operación en R preserva el invariante en M.

*Definición 3. Resultado de la Redefinición semántica de Contexto*

Si un contrato de reuso R es redefinible semánticamente de contexto por un modificador M, entonces el contrato de reuso R' es la redefinición semántica de contexto de R por M y escribimos R' = Redc(R, M), donde:
1. R' contiene todos los participantes de R, con el mismo nombre y la misma cláusula de conocimiento que en R, que contienen todas las operaciones en R con la cláusula de especialización y post-condición idénticas en R.
2. El invariante de R' es el invariante en M.

# Capítulo 5: Manejando Evolución de un Contrato de Reuso con Semántica de Comportamiento

En este capítulo presentamos la aplicación de operadores de reuso sobre el contrato de reuso *CtaBancaria* que fue introducido como ejemplo de contrato de reuso bien formado en el capítulo 2. Como podrá observarse en la sección 5.2, al combinar operadores sintácticos que se aplican sobre el mismo contrato base, se pueden producir ciertos conflictos. Consecuentemente, al combinar operadores semánticos con los sintácticos y entre ellos pasará algo similar. Este tema es tratado en el siguiente capítulo.

## *5.1 Evolución de un contrato base*

Supongamos que, con el objetivo de obtener cuentas específicas, por ejemplo cajas de ahorro, se aplica un operador de reuso sobre el contrato *CtaBancaria*, descripto en el cap.2. Este operador es un <<Refinamiento Combinado de Participante>> que agrega y refina sintáctica y semánticamente operaciones del participante CtaBanc. Dado que éste es el único participante que evoluciona y que los cambios preservan el invariante del contrato, se lo puede mostrar ligado por la relación de herencia con CAhorro, como muestra la Fig. 5.1a, sin necesidad de graficar todo el contrato. El motivo de la evolución es agregar comportamiento a la cuenta bancaria (generándose así el participante CAhorro), llevando un contador en cada cuenta con la cantidad de extracciones que se realizan. En este nuevo participante, la cláusula de especialización de la operación *extraer* es *<incExtr>*.

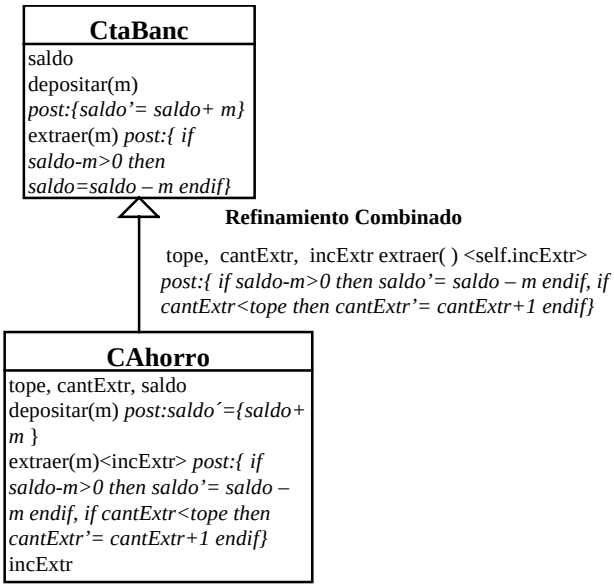


*Fig.5.1a Evolución de CtaBanc*

Para reflejar el comportamiento específico de las instancias de CAhorro, es válido refinar el invariante del contrato con el fin de que las cajas de ahorro nunca superen una cantidad máxima de extracciones; este refinamiento se realiza a través de la aplicación del operador

<<Refinamiento Semántico de Contexto>>. La fig. 5.1b muestra el contrato completo *CAhorro*, resultante de aplicar ambos operadores, utilizando notación UML. El significado de la aplicación del primer operador es el mismo que el que se muestra en la Fig. 5.1a.

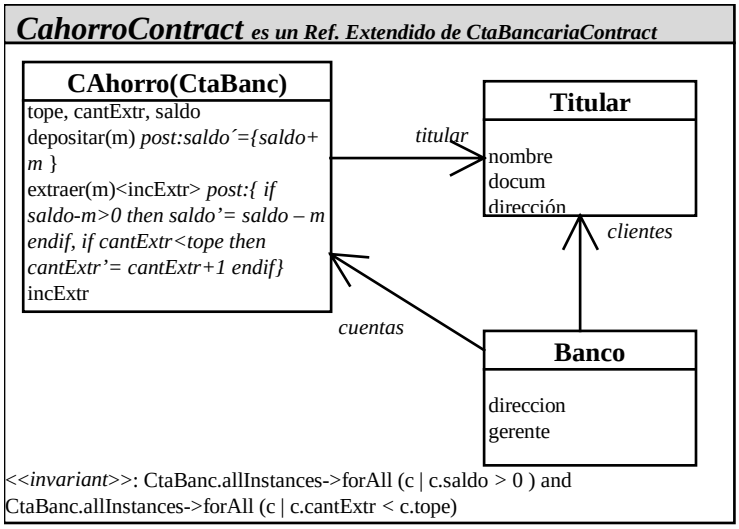


*Fig 5.1b. Evolución del contrato CtaBancaria*

### *5.2 Cambios sobre un contrato que ya tenía un derivado*

Supongamos ahora que queremos tener cuentas bancarias que llevan registro de todos sus movimientos, a fin de obtener resúmenes mensuales. Esto conduce a la evolución del participante *CtaBanc* del contrato *CtaBancaria* (Fig. 2.1), quién anteriormente había sido modificado. Como es el único participante que evoluciona se lo puede mostrar ligado por la relación de herencia con *CtaBancResumen*, como muestra la Fig. 5.2. Además se debe agregar al contrato el participante *Movimiento* con operaciones tipo y monto, que devuelven el tipo y monto de la transacción y la relación de conocimiento *movs* entre *CtaBanc* y *Movimiento*, a través de la cual la cuenta registra y conoce sus movimientos o transacciones.

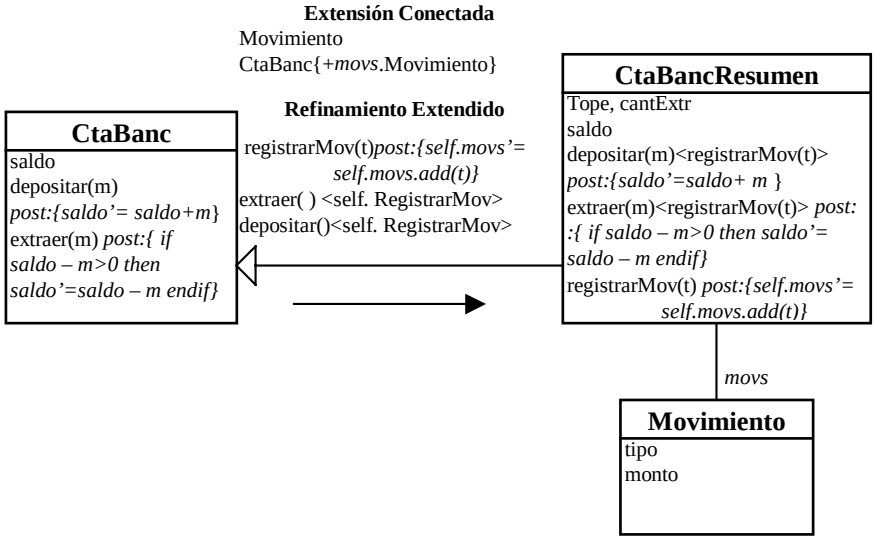


*Fig. 5.2 Otra evolución de CtaBanc*

En este punto, planteada la evolución de un participante (CtaBanc) que ya había evolucionado y tenía otro participante (*CAhorro*) que deriva de él, se debe analizar el efecto causado por ambas modificaciones sobre el mismo contrato base. La fig. 5.3 muestra al participante *CtaBanc*, con su derivado *CAhorro*, modificándose hacia una nueva versión.

Debemos observar:
(1) ¿qué efecto tiene este cambio sobre CAhorro, que ya existía?
(2) ¿qué efecto tiene la existencia de otra derivación (CAhorro), sobre CtaBancResumen?
(3) ¿existe conmutatividad?, es decir ¿podemos obtener CAhorroResumen tanto desde CAhorro como de CtaBancResumen utilizando operadores similares a los ya aplicados sobre el contrato base para obtener ambas modificaciones?

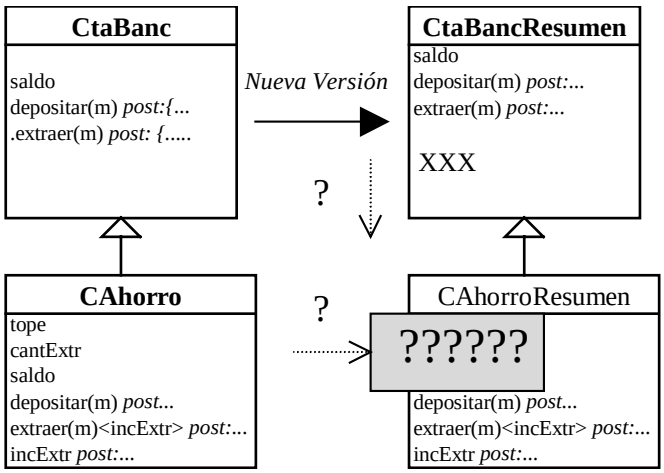

*Fig. 5.3 Evolución de la superclase CtaBanc*

En respuesta al punto (1), observando los modificadores de reuso ( Fig 5.1a y Fig 5.2) vemos que al combinarlos aparece un conflicto de invocación de operación, del grupo de interfaz [Lucas97c], ya que ambos refinamientos se aplican sobre la operación *extraer* del participante *CtaBanc*.
Una solución a este conflicto puede darse analizando las post-condiciones de las operaciones que refinan la operación en cuestión y el orden en que se dan estos refinamientos, de lo que surge la siguiente regla:

*Si* las operaciones que refinan la misma operación tienen post-condiciones independientes *entonces*:
*si* ambos refinamientos se realizan en la misma posición dentro de la cláusula de especialización original entonces se puede aplicar el segundo en la misma posición y en paralelo,
*sino* mantener el refinamiento con posición menor e incrementar en uno la posición mayor.

Podemos decir que estos refinamientos son compatibles pues cada uno enriquece la operación *extraer* en diferente sentido, con post-condiciones que no tienen elementos en común.
Con respecto al punto (2) el análisis es similar; para poder combinar ambos modificadores, CtaBancResumen deberá obtenerse haciendo un refinamiento en modo paralelo sobre CtaBanc, en la misma posición que CAhorro, pues ambos refinamientos aplicados a *extraer* se realizan en la misma posición.

La Figura 5.4 muestra la derivación de *CAhorroResumen* desde *CAhorro* (punto(3)). El operador aplicado es similar al que se aplicó sobre *CtaBanc* para obtener *CtaBancResumen*, pero ahora el refinamiento es en paralelo. La misma figura muestra que sucede si queremos evolucionar desde CtaBancResumen a CAhorroResumen, es decir ver si el gráfico de la Fig. 5.3 conmuta (llegar a CAhorroResumen desde las dos subclases de CtaBanc). Aquí vuelve a plantearse la misma cuestión: el operador aplicado es similar al que se aplicó sobre CtaBanc para obtener CAhorro, pero ahora es en paralelo. Con esto comprobamos que el gráfico conmuta.

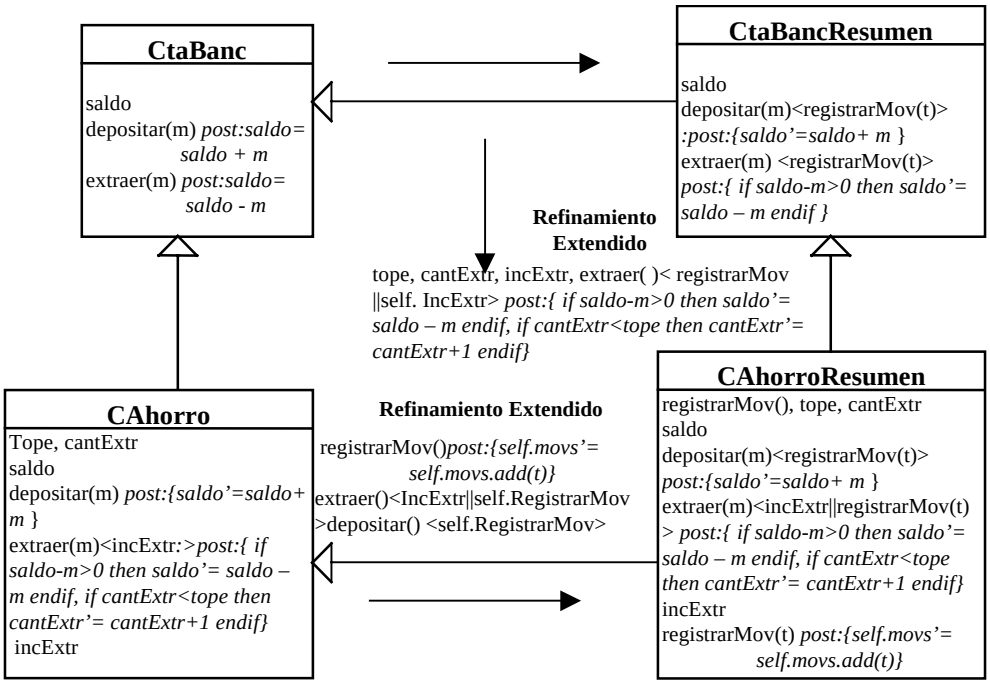


*Fig. 5.4 Evolución de CAhorro y CtaBancResumen*

### *5.3 Involución de un contrato*

En el caso de una involución, por ejemplo querer obtener CAhorro desde CAhorroResumen, se debe realizar:
(1) un *coarsening* de participante de *registrarMov* en *extraer* y *depositar*.
(2) un *coarsening* sintáctico de contexto de la relación de conocimiento *movs* (que da los movimientos asociados a una cuenta). Como *movs* ocurre en la post-condición de *registrarMov*, ésta operación también debe ser cancelada por la definición de aplicabilidad de *coarsening* sintáctico de contexto.
(4) una cancelación de contexto de Movimiento.

La fig. 5.5 muestra los operadores que deben aplicarse sobre CAhorroResumen. En general, estos operadores coinciden con los operadores inversos a los aplicados en la evolución.

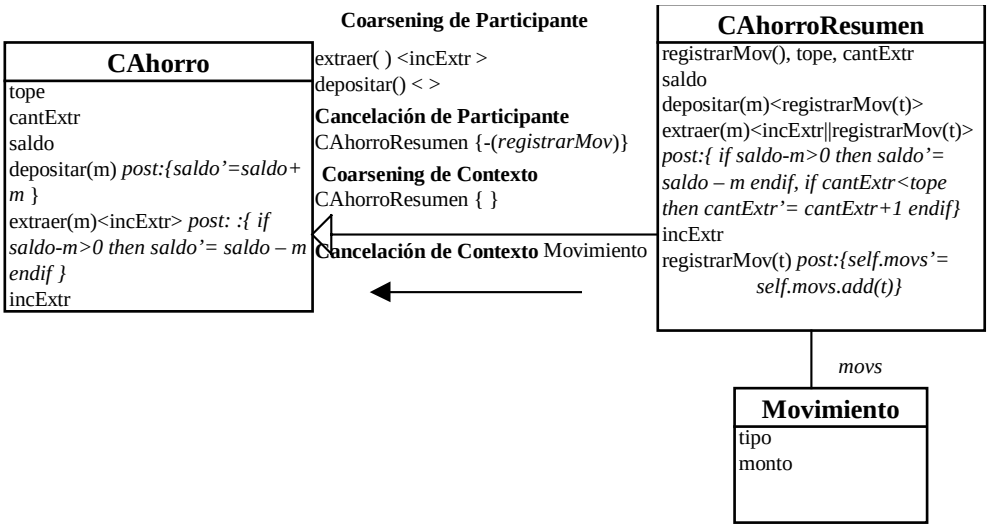


*Fig.* 5.5 *Involución de CtaAhorroResumen*

# Capítulo 6: Conflictos en la evolución de un contrato de reuso

Los contratos de reuso proveen una documentación estructurada de sistemas de software. Utilizar contratos y operadores de reuso hace posible detectar problemas al producirse cambios sobre diferentes partes del sistema que se relacionan y al propagarse los cambios, lo cual es de fundamental importancia para los desarrolladores de componentes reusables.
La detección de conflictos se lleva a cabo investigando cómo interactúan dos modificaciones hechas sobre un mismo contrato de reuso base. Es decir, si tenemos dos modificadores M1 y M2, que son aplicables al contrato base, como se muestra en la figura 6.1, queremos ver si la combinación de ambos es posible y si el resultado será el esperado.

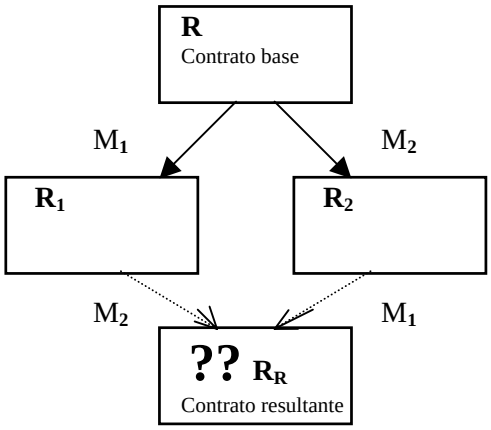


*Fig. 6.1 Combinación de Operadores*

En general, todos los conflictos considerados, son detectados comparando los modificadores de reuso sin necesidad de consultar el contrato base. En [Lucas97c] se analizan los posibles conflictos producidos al aplicar dos modificadores de reuso sintáctico. Ese análisis sigue siendo válido para la adaptación hecha en esta propuesta, ya que los operadores de reuso sintáctico conservan los elementos semánticos.

En este capítulo planteamos y analizamos conflictos que pueden ocurrir al incorporar los operadores de reuso semántico definidos en capítulos anteriores. Estos conflictos pueden generarse al aplicar un operador sintáctico y otro semántico o al aplicar dos operadores semánticos.
Podemos clasificar los conflictos en:

- Conflictos de referencia semántica colgada
- Conflictos semánticos de participante
- Conflictos semánticos de contexto
- Conflictos de inconsistencia semántica

Cada uno de estos grupos es analizado en las secciones siguientes.

## *6.1 Conflictos de referencia semántica colgada*

Los conflictos de *referencia semántica colgada* ocurren cuando un operador sintáctico elimina un elemento de la interfaz, mientras un operador semántico hace referencia al mismo elemento.

La tabla siguiente muestra en que casos específicos, al combinar operadores de reuso sintáctico y semántico, se producen conflictos de referencia semántica colgada:

| M1<br>M2 | Cancelación Sintáctica Part. *p {m}* | Cancelación Sintáctica Contexto de *p* | Coarsening Sintáctico Contexto de *p* |
|---|---|---|---|
| Redefinición o Refinamiento semántico de Contexto | Referencia Colgada en Invariante | Referencia Colgada en Invariante | Referencia Colgada en Invariante |
| Cualquier operador semántico de Participante *p {m} cond* | Referencia Sem.de Operación Colgada | Referencia Sem.de Operación Colgada | - |

Un *conflicto de Referencia Colgada en Invariante* ocurre cuando existen una operación *m* y un nombre de participante *p* tales que:

M1 representa una cancelación sintáctica de participante de *m* sobre *p,* o una Cancelación Sintáctica de Contexto de *p* o un Coarsening Sintáctico de Contexto de alguna relación de conocimiento de *p*
M2 describe una Redefinición o Refinamiento semántico de Contexto que involucra a *m* o a *p*.
Cuando M2 es un coarsening semántico de Contexto no se produce conflicto pues debilita la invariante original, que no contenía al elemento.

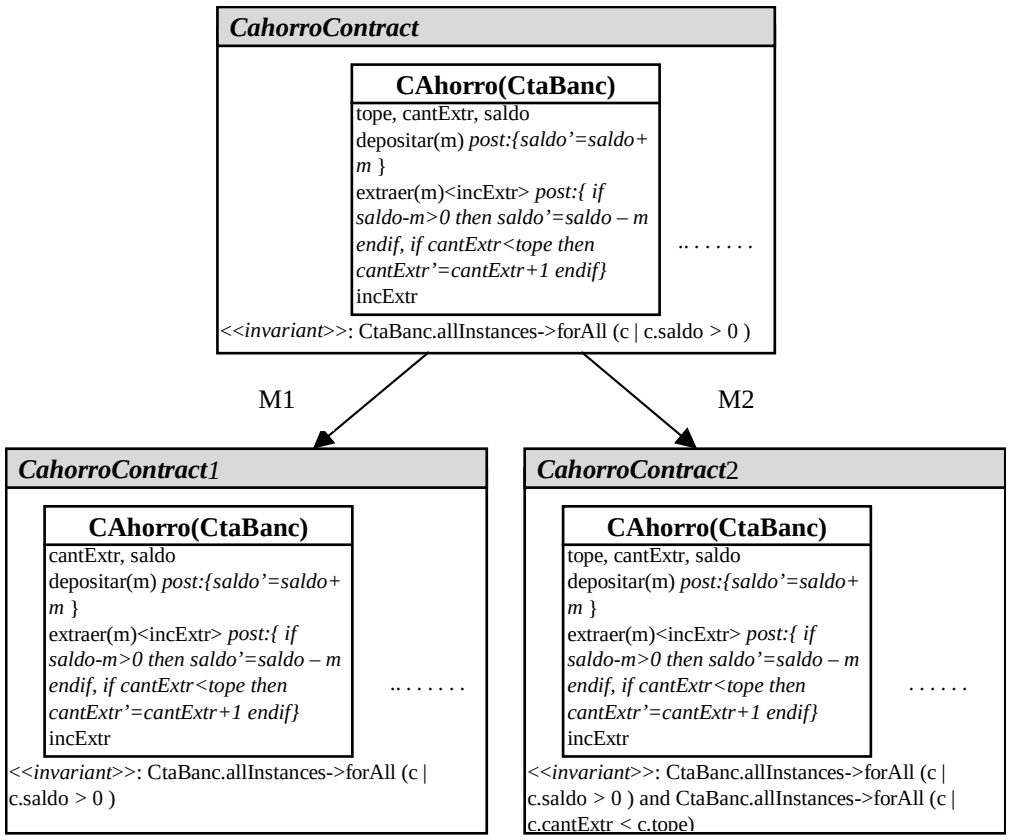


*Fig 6.2. Referencia Colgada en Invariante*

La figura 6.2 muestra un ejemplo de conflicto de referencia colgada en invariante. El modificador M1 cancela sintácticamente a *tope* en el participante CAhorro, mientras M2 representa un Refinamiento semántico de contexto, que modifica al invariante del contrato.

Dado que *tope* ocurre en la expresión invariante resultante, se genera el conflicto. En éste y todos los ejemplos del capítulo, cada modificador en particular es *aplicable* al contrato base.

Un *conflicto de Referencia Semántica de Operación Colgada* ocurre cuando existe un nombre de participante *p* tal que:

M1 representa una cancelación sintáctica de participante de *m* sobre *p* o una Cancelación Sintáctica de Contexto de *p*
M2 describe cualquier operador semántico de participante de *m* sobre *p.*

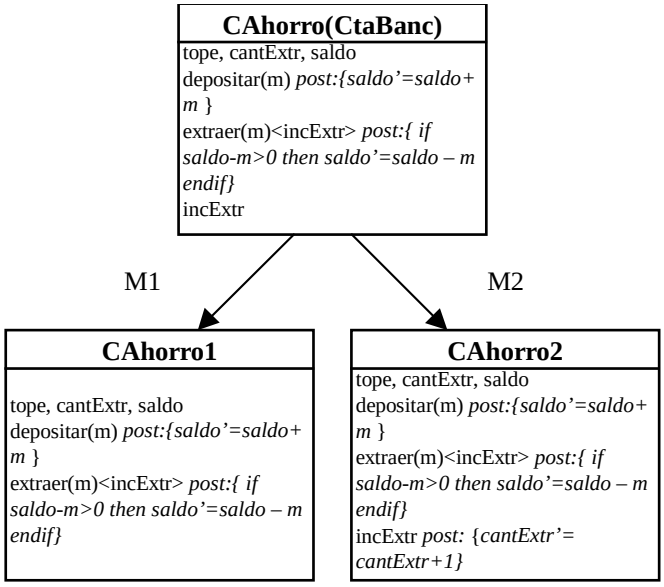

*Fig 6.3. Referencia Semántica de Operación Colgada*

En la fig 6.3 el modificador M1 cancela sintácticamente a *incExtr* en el participante CAhorro; previamente esta operación se debe eliminar de la cláusula de especialización de *extraer* (mediante un *coarsening* sintáctico). Por otro lado M2 representa una Redefinición semántica de participante, que modifica la post-condición de *incExtr*, por lo que la combinación de ambas modificaciones genera conflicto. Suponemos para el ejemplo que se preserva el invariante que indica que el saldo de todas las cuentas debe ser mayor que 0.

## *6.2 Conflictos semánticos de participante*

Los conflictos *semánticos de participante* ocurren cuando dos operadores semánticos de participante hacen referencia al mismo elemento.
La tabla siguiente muestra en que casos específicos, al combinar operadores de reuso, se producen conflictos semánticos de participante:

| M1<br>M2 | Redefinición Sem.<br>*p { m, cond1}* | Refinamiento Sem.<br>*p {m cond1}* | Coarsening Sem.<br>*p {m cond1}* |
|---|---|---|---|
| Redefinición Sem.<br>*p { m, cond2}* | Incoherencia Sem.<br>de Operación | Incoherencia Sem.<br>de Operación | Incoherencia Sem. de<br>Operación |
| Refinamiento Sem.<br>*p {m cond2}* | Incoherencia Sem.<br>de Operación | Inconsistencia Sem.<br>.de Operación | *No hay Conflicto* |
| Coarsening Sem.<br>*p {m cond2}* | Incoherencia Sem. de<br>Operación | *No hay Conflicto* | Inconsistencia Sem. de<br>Operación |

Un *conflicto de Incoherencia Semántica de Operación* ocurre cuando existen una operación *m* y un nombre de participante *p* tales que:

M1 representa una Redefinición semántica de participante de *m* sobre *p* y M2 describe cualquier operador semántico de participante que involucre a *m* de *p*,ó
M1 representa un Refinamiento o Coarsening semántico de participante de *m* sobre *p* y M2 representa una Redefinición semántica de participante de *m* sobre *p.*

La figura 6.4, muestra dos redefiniciones semánticas de participante sobre la operación *extraer* del participante CAhorro. La aplicación de dos redefiniciones semánticas sobre la misma operación produce conflicto.

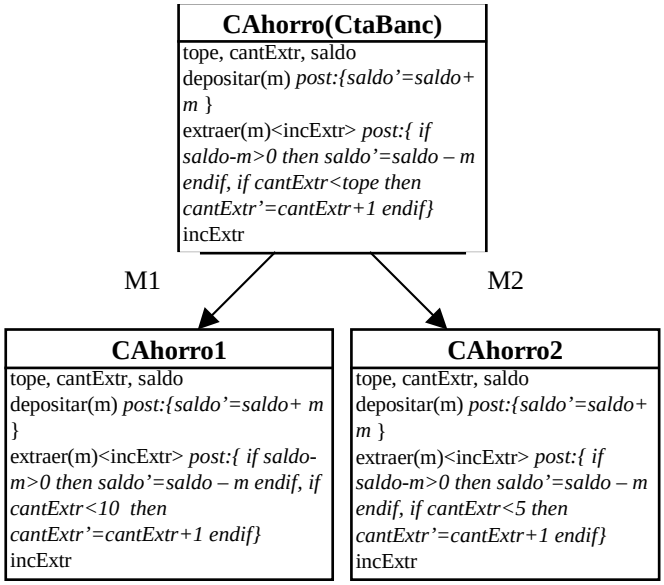


*Fig 6.4 Incoherencia Semántica de Operación*

Un *conflicto de Inconsistencia Semántica de Operación* ocurre cuando existen una operación *m* y un nombre de participante *p* tal que:

M1 y M2 representan un Refinamiento semántico de participante de *m* sobre *p* ó
M1 y M2 representan un Coarsening semántico de participante de *m* sobre *p*.

Refinamiento y Coarsening semánticos combinados (y viceversa) no producen conflicto.

En la figura 6.5, el modificador M1 representa un Refinamiento semántico de participante de *extraer* sobre el participante CAhorro, donde la post-condición ahora expresa que la

cantidad de extracciones no debe superar un cierto tope; supongamos que este tope se inicializa con el valor 6. Por otro lado M2 representa otro Refinamiento semántico de participante sobre extraer en CAhorro. En este caso la post-condición indica que deben superarse las 5 extracciones. Suponemos para el ejemplo que ambas modificaciones preservan el invariante que indica que el saldo de todas las cuentas debe ser mayor que 0, pero su combinación conduce a una situación conflictiva.

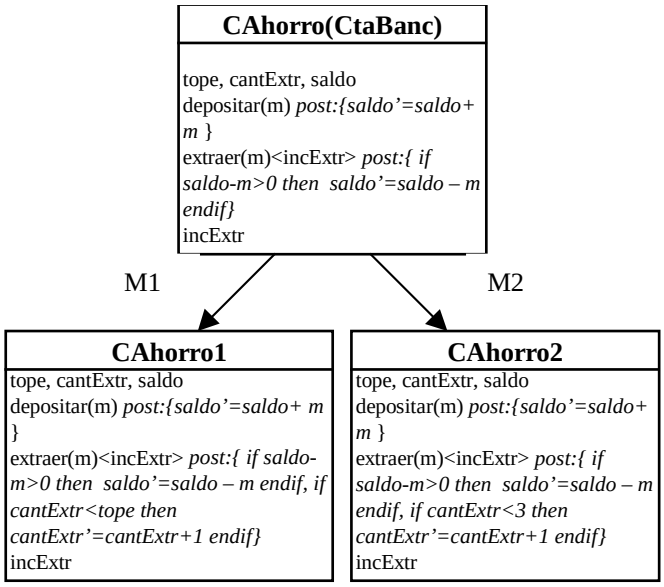


*Fig 6.5. Inconsistencia Semántica de Operación*

### *6.3 Conflictos semánticos de contexto*

Los conflictos *semánticos de contexto* ocurren cuando dos operadores semánticos de contexto hacen referencia al mismo elemento.
La tabla siguiente muestra en que casos específicos, al combinar operadores de reuso, se producen conflictos semánticos de contexto:

| M1<br>M2 | Redefinición Sem.<br>*{ Inv1}* | Refinamiento Sem.<br>*{ Inv1}* | Coarsening Sem.<br>*{ Inv1}* |
|---|---|---|---|
| Redefinición Sem.<br>*{ Inv2}* | Incoherencia Sem.<br>de Invariante | Incoherencia Sem.<br>de Invariante | Incoherencia Sem. de Invariante |
| Refinamiento Sem.<br>*{ Inv2}* | Incoherencia Sem.<br>de Invariante | Inconsistencia Sem. de Invariante | *No hay Conflicto* |
| Coarsening Sem.<br>*{ Inv2}* | Incoherencia Sem. de Invariante | *No hay Conflicto* | Inconsistencia Sem. de Invariante |

Un *conflicto de Incoherencia Semántica de Invariante* ocurre cuando:

M1 representa una Redefinición semántica de contexto y M2 describe cualquier operador semántico de contexto, ó
M1 representa un Refinamiento o Coarsening semántico de contexto y M2 representa una Redefinición semántica de contexto.

La figura 6.6 muestra en el modificador M1 una Redefinición semántica de contexto, que modifica al invariante del contrato, mientras M2 lo redefine cambiando el valor mínimo permitido para el saldo. La combinación de ambos lleva a un conflicto.

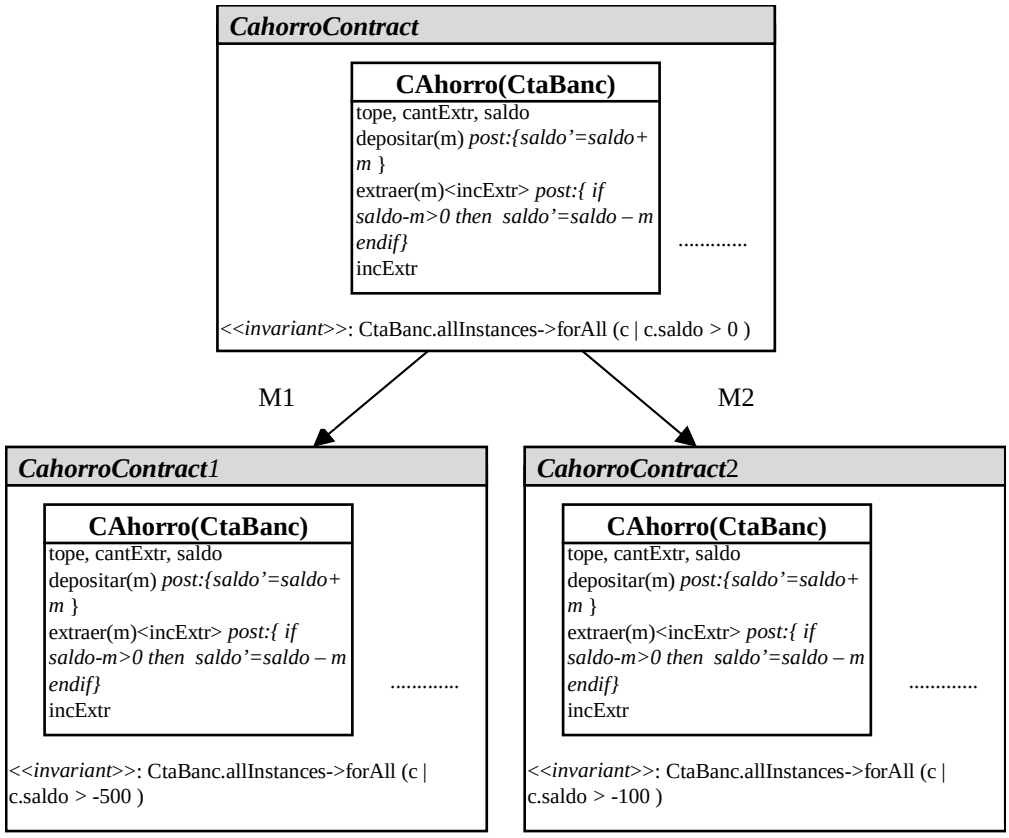


*Fig 6.6 Incoherencia Semántica de Invariante*

Un *conflicto de Inconsistencia Semántica de Invariante* ocurre cuando:
M1 y M2 son modificadores de Refinamiento semántico de contexto, ó
M1 y M2 son modificadores de Coarsening semántico de contexto.

En la figura 6.7, M1 representa un Refinamiento semántico de contexto, donde el invariante ahora expresa que la cantidad de extracciones en las cuentas debe ser menor que 3. Por otro lado M2 representa otro Coarsening semántico de contexto. En este caso el invariante indica que la cantidad de extracciones debe estar entre 3 y 5, por alguna decisión política del banco. La combinación de ambos refinamientos no puede aplicarse pues ambas condiciones no instancian un pattern de refinamiento.

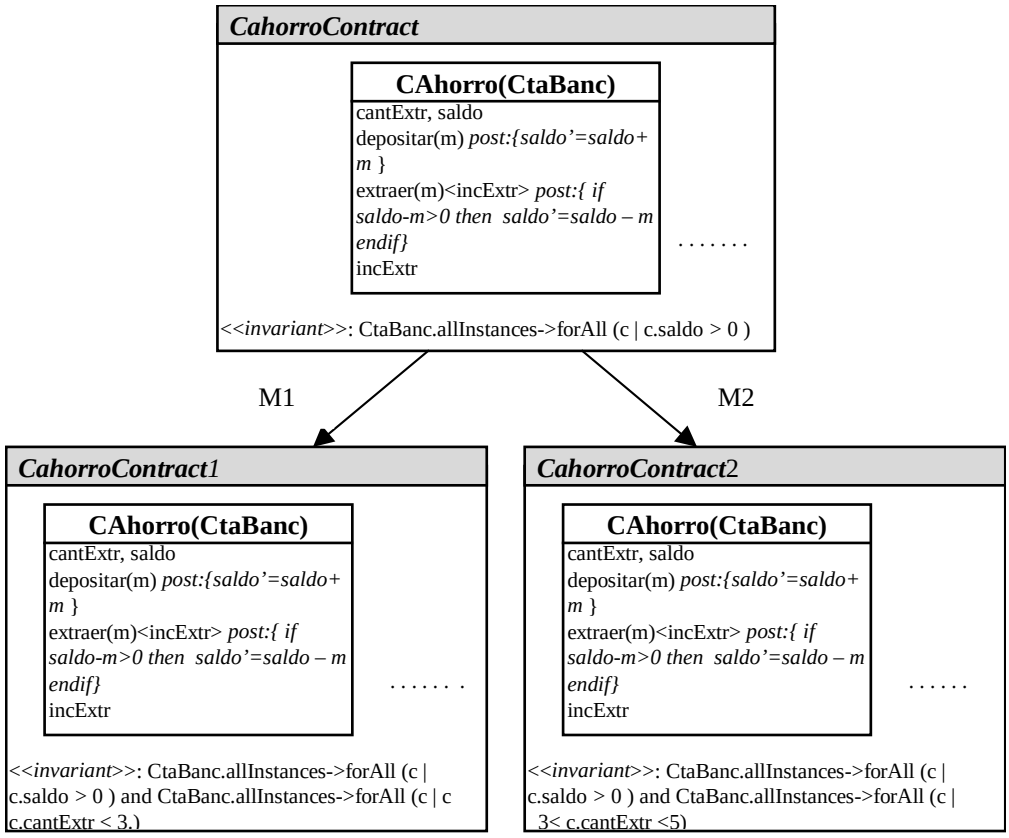


*Fig 6.7 Inconsistencia Semántica de Invariante*

### *6.4 Conflictos de inconsistencia semántica*

Los conflictos de *inconsistencia semántica* ocurren cuando se aplica un operador semántico de participante y un operador semántico de contexto sobre el mismo contrato. Es decir, un operador semántico de participante modifica la post-condición de una operación, mientras que un operador semántico de contexto modifica el invariante.
El análisis de este conflicto se basa en la relación de consistencia entre elementos semánticos de un contrato, que fuera incluída en el Capítulo 4.
La siguiente tabla muestra en que casos específicos, al combinar operadores de reuso, se producen conflictos de inconsistencia semántica:

| M1<br>M2 | Redefinición Sem. de Participante | Refinamiento Sem. de Participante | Coarsening Sem. de Participante |
|---|---|---|---|
| Refinamiento, Coarsening ó Redefinición Semántica de Contexto | Inconsistencia Semántica | Inconsistencia Semántica | Inconsistencia Semántica |

El cambio en el orden de aplicación de los operadores (primero de contexto, luego de participante) produce los mismos conflictos.

Un *conflicto de Inconsistencia Semántica* ocurre cuando:
M1 representa una Redefinición, Refinamiento ó Coarsening semántico de participante y M2 describe cualquier operador semántico de contexto.

La figura 6.8 muestra en el modificador M1 una Redefinición semántica de participante, que modifica la post-condición de la operación extraer limitando el saldo de las cuentas, mientras M2 modifica al invariante indicando que los saldos deben mantenerse sobre 50. Ambos refinamientos son aplicables pero su combinación lleva a un conflicto de inconsistencia semántica.

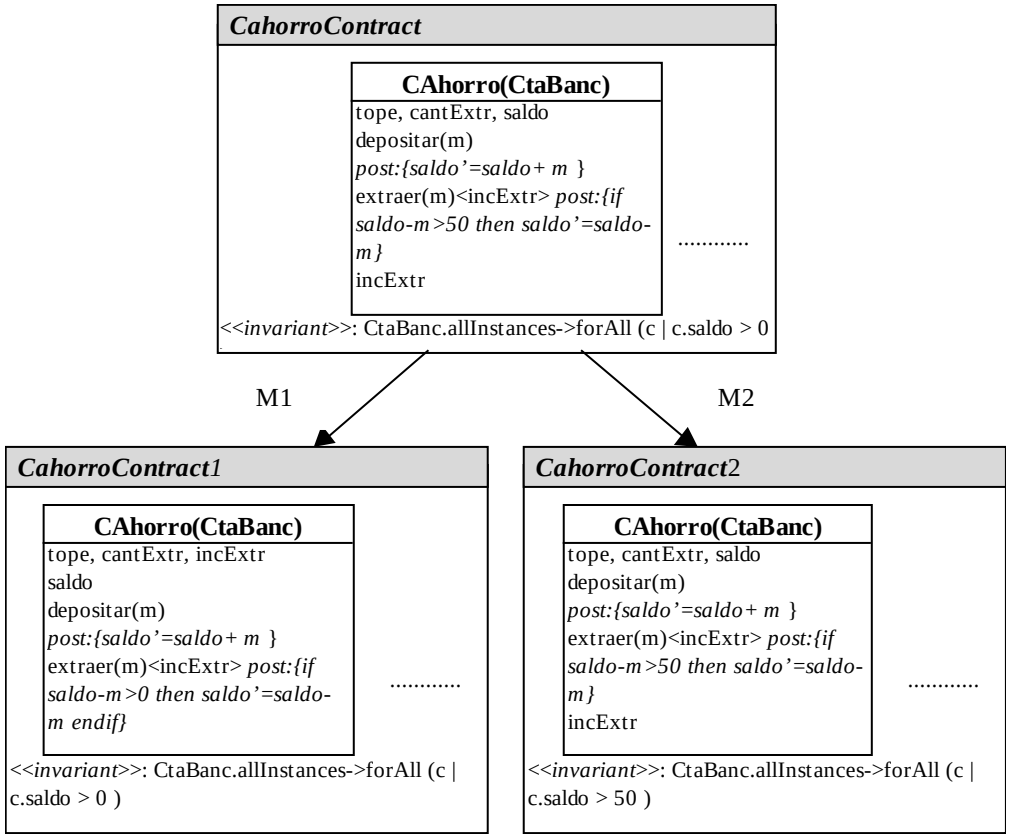


*Fig 6.8 Inconsistencia semántica*

# Capítulo 7: Un Modelo Matemático para Contratos de Reuso con Semántica de Comportamiento

Basándonos en Mens [MLS96], que formaliza contratos de reuso simple-clase definidos en [SLMH96 y CHSV97], presentamos un *modelo matemático* para estudiar los contratos de reuso con semántica de comportamiento y operadores de reuso.
En este modelo se pueden definir y probar propiedades concernientes a la aplicación combinada de operadores de reuso sobre un mismo contrato base; estas propiedades y su demostración se presentan en el Capítulo 8.
Esta notación permitirá, como trabajo posterior, la implementación de un chequeador de conflictos en la evolución de componentes reusables.

## *7.1 Contrato de reuso con semántica de comportamiento*

Definimos los dominios semánticos para contratos de reuso de la siguiente forma:

CONTRATO = $\mathcal{P}$(PART) X $\mathcal{P}$(CON) X $\mathcal{P}$(INTERF) X INV dominio para contratos de reuso, donde $\mathcal{P}$(S) denota el conjunto de partes del conjunto S.

CON = PART1 X PART1 X ASOC dominio para relaciones de conocimiento entre participantes del contrato

INTERF = PART1 X OP X CESPEC X COND dominio para interfaces de operación (cláusula de especialización y post-condición) de participantes del contrato

PART = $\mathcal{P}$(STRING) dominio para nombres de participantes del contrato

ASOC, PART1, OP $\subset$ STRING dominios para nombres de relaciones, participantes que se relacionan y operaciones del contrato, respectivamente. Estos conjuntos son disjuntos dos a dos.

INV = EXPOCL dominio para invariante del contrato, es el dominio para las expresiones OCL.

COND = $\mathcal{P}$(EXPOCL) dominio para post-condición de operación, donde los términos *primados* son una extensión de la pseudovariable "*result*" de OCL y representan el valor del término luego de ejecutarse la operación. Cada expresión e perteneciente a COND se define de la siguiente forma:

e:: = x' = f(x, $y_1$..$y_n$) | x' = *if* expB *then* e1 *else* e2 *endif* [2]

donde f(x, $y_1$..$y_n$) es una expresión OCL primaria que involucra al término x , expB es una expresión OCL booleana.

[2] Generalmente usaremos la notación *if* expB *then* x'=e1 *else* x'=e2 *endif* en lugar de x' = *if* expB *then* e1 *else* e2 *endif*

Sea $I(x, y_1..y_p)$ el invariante de un contrato de reuso; el reemplazo de un término *primado* x' en el invariante se define de la siguiente forma:
$I(x, y_1..y_p)[ x' = f(x, y_1..y_n)] = I( f(x, y_1..y_n), y_1..y_p)$

$I(x, y_1..y_p)[ x' =$ *if* expB *then* e1 *else* e2 *endif* $] = I(x, y_1..y_p)[$ e1$]$ si expB
$= I(x, y_1..y_p)[$ e2$]$ si $\neg$expB

CESPEC es el dominio para cláusulas de especialización, donde cada elemento elem perteneciente a CESPEC se define recursivamente de la siguiente forma:

elem:: = OP | elem1; elem2 | elem1 || elem2 | $\sigma$

y donde $\sigma$ denota el valor nulo para el dominio de las cláusulas de especialización CESPEC; STRING es el dominio primitivo de las cadenas de caracteres.

**Funciones observadoras sobre contratos de reuso**

Sea R = <part, con, interf, inv> un elemento del dominio CONTRATO. Definimos las siguientes funciones observadoras sobre R:

- Part :: CONTRATO $\rightarrow \mathcal{P}$(PART)

Part(R)= $\Pi_1$(R) conjunto de todos los nombres de participantes de R

Donde $\Pi_i$ es la proyección i de la n-upla para algún i de 1 a n.

- Inv :: CONTRATO $\rightarrow$ INV

Inv(R) = $\Pi_4$(R) invariante del contrato de reuso R.

- Con::CONTRATO $\rightarrow$ PART$\rightarrow \mathcal{P}$(ASOC)

$Con_R$ (p) = *map* $\Pi_3$ ( *filter* $\Pi_1(x)$=p Rel (R)) conjunto de todas las relaciones de conocimiento del participante *p* en R. Donde Rel (R) = $\Pi_2$(R) y las funciones *map* y *filter* que se aplican a una lista o conjunto de elementos, se definen:

*map* f $X_S$ = [f x | x $\leftarrow$ $X_S$] retorna una lista con el resultado de aplicar la función f a cada elemento de la lista $X_S$

*filter* p $X_S$ = [ x | x $\leftarrow$ $X_S$ , p x ] toma un predicado p y retorna la sublista de $X_S$ cuyos elementos satisfacen a p.

- Int:: CONTRATO $\rightarrow$ PART$\rightarrow \mathcal{P}$(OP)

$Int_R$ (p) = *map* $\Pi_2$ ( *filter* $\Pi_1(x)$=p Interf(R)) conjunto de nombres de operaciones del participante p en R
Donde Interf (R) = $\Pi_3$(R)

- Espec:: CONTRATO $\rightarrow$ PART$\rightarrow \mathcal{P}$(OP) $\rightarrow$ CESPEC

$Espec_{R\,p}$ (m) = $\Pi_3$ (*detect* ($\Pi_1(x)$=p and $\Pi_2(x)$=m) Interf(R)) secuencia de invocaciones de operación, de la operación m de *p* en R (cláusula de especialización de m en *p*).
Donde *detect* p $X_S$ = hd (*filter* p $X_S$ ) toma un predicado p y retorna el primer elemento de la lista $X_S$ que satisface a p.

- Pcond :: CONTRATO $\rightarrow$ PART$\rightarrow \mathcal{P}$(OP)$\rightarrow$ COND

$Pcond_{R\,p}$ (m) = $\Pi_4$ (*detect* ($\Pi_1(x)$=p and $\Pi_2(x)$=m) Interf(R)) post-condición de la operación m del participante p en R, dada por un conjunto de expresiones OCL

**Contrato de Reuso Bien Formado**

*Definición.* Un contrato de reuso R está Bien Formado si:
(1) $\forall$ p $\in$ Part(R):
(a) $\forall$ a.q $\in$ $Con_R$ (p): q $\in$ Part(R)
(b) $\forall$ m $\in$ $Int_R$ (p): $\forall$ a.n $\in$ $Espec_{R\,p}$ (m):
$\exists$ q $\in$ Part(R) :
(b1) a.q $\in$ $Con_R$ (p)
(b2) n $\in$ $Int_R$ (q)
(c) $\forall$ m $\in$ $Int_R$ (p): $\forall$ e $\in$ $Pcond_{R\ p}$(m): *bienFormada*(e, R) $\wedge$ Inv(R)$\rightarrow$Inv(R)[$Pcond_{R\,p}$(m)]
(2) *bienFormada*(Inv(R), R)

donde el predicado *bienFormada* se define:

*bienFormada*:: EXPOCL X CONTRATO $\rightarrow$ BOOL

el significado intuitivo del predicado es:
*bienFormada* (exp, R) = true , si exp es una expresión OCL bien formada respecto al contrato R, es decir involucra elementos de R.
= false , en caso contrario.

y donde Inv(R)[$Pcond_{R\,p}$(*m*)] representa el invariante del contrato R donde se ha efectuado el reemplazo de términos *primados* especificados en la post-condición de la operación *m*. El resto de los términos, que no aparecen *primados* en la post-condición, permanecen sin cambios.
Por ejemplo,
sean Inv : cont>0, Pcond : cont' = cont +1, entonces
Inv[Pcond]= (cont>0)[cont' = cont +1], lo que es igual a (cont+1)>0

**Modificadores de reuso**

Definimos a los modificadores de reuso como elementos del dominio CONTRATO, a fin de unificar la estructura del modificador de todos los operadores de reuso.

Las funciones observadoras que se le pueden aplicar a un modificador son las aplicables a contratos de reuso. De acuerdo al operador que estemos definiendo, los elementos que formen el modificador serán todos o parte de los que forman un contrato de reuso, por lo tanto los resultados de estas funciones pueden ser secuencias o conjuntos vacíos o elementos nulos.
En los Capítulos 3 y 4, al introducir cada operador de reuso, se describen las particularidades de cada modificador.
En algunos casos, el modificador puede no ser un contrato de reuso bien formado. Sin embargo, si el operador es aplicable sobre un contrato bien formado, el resultado de la aplicación *siempre* será un contrato de reuso bien formado.

**Operadores de reuso**

Dada esta definición de Modificador, definimos el dominio semántico para Operadores de Reuso de la siguiente forma:

OPERADOR = [(CONTRATO x CONTRATO) $\rightarrow$ CONTRATO]

Donde la primer componente del par ordenado denota el dominio de los contratos de reuso base, la segunda componente denota el dominio de los modificadores. Un operador de reuso aplica sobre un contrato de reuso base un modificador de reuso, dando como resultado otro contrato de reuso.
En las siguientes secciones damos la notación matemática para cada operador de reuso sintáctico y semántico, utilizando los dominios semánticos definidos.

## *7.2 Operadores de Reuso Sintáctico*

En esta sección presentamos la notación matemática de las definiciones de aplicabilidad y resultado para cada operador sintáctico definido informalmente en el Capítulo 3. Esta notación sustentará los enunciados y pruebas de propiedades de aplicabilidad de operadores definidas en el Capítulo 8.
Como ya fue aclarado, no se permite especificar post-condiciones al agregar una operación. En ausencia de post-condiciones se asume el conjunto vacío, el cual representa el reemplazo neutro, es decir, para un contrato de reuso R, Inv(R) [{}] = Inv(R).

**Extensión sintáctica de participante:** *Extp*

**Definición de Aplicabilidad**

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador ***Extp*** (extensión sintáctica de participante) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$
$\forall$ *p* $\in$ Part (M):
(1) *p* $\in$ Part(R)
(2) $\forall$ n $\in$ $Int_M$ (p): n $\notin$ $Int_R$ (p)

(3) $\forall$ n $\in$ $Int_M(p)$:
$\forall$ a.m $\in$ $Espec_{M\ p}$ (n):
(a) $\exists$ q $\in$ Part(R): a.q $\in$ $Con_R$ (p)
(b) si a.q $\in$ $Con_R$ (p) $\Rightarrow$ m $\in$ $Int_M$ (q)

*Definición del Resultado de la Extensión de Participante*

Sea R $\in$ CONTRATO, R´ = *Extp*(R, M) es una Extensión sintáctica de Participante de R por M $\Leftrightarrow$

(1) *Extp* es aplicable a R por M.

(2) $\forall$ *p* $\in$ Part(R) - Part(M) : *p* $\in$ Part(R')

(3) $\forall$ *p* $\in$ Part(M) : *p* $\in$ Part(R') $\wedge$
(a) $Int_{R'}$ (p) = $Int_R$ (p) $\cup$ $Int_M$ (p)
(b) $Con_{R'}$ (p) = $Con_R$ (p)
( c) $\forall$ m $\in$ $Int_M$ (p) : $Espec_{R',p}$ (m) = $Espec_{M\,p}$(m) $\wedge$ $Pcond_{R',p}$ (m)={}

(4) Inv(R') = Inv(R)

**Extensión sintáctica de contexto:** *Extc*

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador ***Extc*** ( extensión sintáctica de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

$\forall$ p $\in$ Part (M): p $\notin$ Part(R)

*Definición del Resultado de la Extensión sintáctica de Contexto*

Sea R $\in$ CONTRATO, R´ = *Extc* (R, M) es una Extensión sintáctica de Contexto de R por M $\Leftrightarrow$

(1) *Extc* es aplicable a R por M

(2) Part(R') = Part(R) $\cup$ Part(M)

(3) $\forall$ *p* $\in$ Part(R) : $Con_{R'}$ (p) = $Con_R$ (p) $\wedge$ $Int_{R'}$ (p) = $Int_R$ (p) $\wedge$
($\forall$ m $\in$ $Int_R$ (p): $Pcond_{R',p}$ (m) = $Pcond_{R\ p}$ (m) $\wedge$ $Espec_{R',p}$ (m) = $Espec_{R\ p}$ (m))

(4) $\forall$ *p* $\in$ Part(M) : $Con_{R'}$ (p) = $Con_M$ (p) $\wedge$ $Int_{R'}$ (p) = $Int_M$ (p) $\wedge$
($\forall$ m $\in$ $Int_M$ (p): $Pcond_{R',p}$ (m) = {} $\wedge$

$Espec_{R',\ p}$ (m) = $Espec_{Mp}$ (m))

(5) Inv(R') = Inv(R)

**Cancelación sintáctica de participante:** *Canp*

*Definición de Aplicabilidad*

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.

El operador ***Canp*** ( cancelación sintáctica de participante) ∈ OPERADOR es aplicable a R por M ⇔

∀ *p* ∈ Part (M):
(1) *p* ∈ Part(R)
(2) ∀ n ∈ $Int_M$(p): n ∈ $Int_R$ (p)

(3) ∀ q ∈ Part(R) : ∀m ∈ $Int_R$ (q): ∀n ∈ $Int_R$ (p): n. *p* ∈ $Espec_{R\ q}$ (m) ⇒ ( n ∈ $Int_M$(p) ⇒ m ∈ $Int_M$(q))

(4) ∀ q ∈Part(R): ∀ m ∈ $Int_R$ (q):∀n ∈ $Int_R$ (p): n ocurre en $Pcond_{R\ q}$(m) ⇒ (n ∈ $Int_M$(p) ⇒ m ∈ $Int_M$ (q))

(5) ∀ m ∈ $Int_M$ (p): m no ocurre en Inv(R)

*Definición del Resultado de la Cancelación sintáctica de Participante*

Sea R ∈ CONTRATO, R´ = *Canp* (R, M) es una Cancelación sintáctica de Participante de R por M ⇔

(1) *Canp* es aplicable a R por M

(2) ∀ *p* ∈ Part(R) - Part(M) : *p* ∈ Part(R'):
(a) $Con_{R'}$ (p) = $Con_R$ (p)
(b) ∀ m ∈ $Int_R$ (p) : m ∈ $Int_{R'}$ (p) ∧ $Espec_{R',\ p}$ (m) = $Espec_{R\ p}$(m) ∧ $Pcond_{R',\ p}$(m) = $Pcond_{R\ p}$ (m)

(3) ∀ *p* ∈ Part(M) : *p* ∈ Part(R') **:**
(a) $Con_{R'}$ (p) = $Con_R$ (p)
(b) ∀ m ∈( $Int_R$ (p) - $Int_M$ (p)): m ∈ $Int_{R'}$ (p) ∧ $Espec_{R',\ p}$ (m) = $Espec_{R\ p}$(m) ∧ $Pcond_{R',\ p}$(m) = $Pcond_{R\ p}$ (m)

(4) Inv(R') = Inv(R)

**Cancelación sintáctica de contexto: *Canc***

*Definición de Aplicabilidad*

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.

El operador ***Canc*** ( cancelación sintáctica de contexto) ∈ OPERADOR es aplicable a R por M ⇔

∀ *p* ∈ Part (M):
(1) *p* ∈ Part(R)
(2) ∀ q ∈ ( Part ( R ) - Part (M)) : ∀ a.r ∈ $Con_R$ (q) ⇒ r ≠ p

(3) ∀ q Part(R): ∀ n ∈ $Int_R$ (q) ⇒ ( *p* ocurre en $Pcond_{R\ q}$ (n) ⇒ q ∈ Part(M))

(4) *p* no ocurre en Inv(R).

*Definición del Resultado de la Cancelación sintáctica de Contexto*

Sea R ∈ CONTRATO, R´ = *Canc* (R, M) es una Cancelación sintáctica de Contexto de R por M ⇔

(1) *Canc* es aplicable a R por M

(2) Part(R') = Part(R) - Part(M)

(3) ∀ *p* ∈ ( Part(R) - Part(M)): $Con_{R'}$ (p) = $Con_R$ (p) ∧ $Int_{R'}$ (p) = $Int_R$ (p) ∧
(∀ m ∈ $Int_R$ (p): $Pcond_{R',\ p}$ (m) = $Pcond_{R\ p}$ (m) ∧ $Espec_{R',\ p}$ (m) = $Espec_{R\ p}$ (m))

(4) Inv(R') = Inv(R)

**Refinamiento sintáctico de participante: *Refp***

*Definición de Aplicabilidad*

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.

El operador ***Refp*** (refinamiento sintáctico de participante) ∈ OPERADOR es aplicable a R por M ⇔

∀ *p* ∈ Part (M):

(1) *p* ∈ Part(R)

*(2)* ∀ m ∈ $Int_M$(p): m ∈ $Int_R$ (p) ∧

$(\forall$ a.op $\in$ ($Espec_{M\ p}$ (m) - $Espec_{R\ p}$ (m)) : ($\exists$ q $\in$ Part(R): a.q $\in$ $Con_R$ (p) $\wedge$ op $\in$ $Int_R$ (q)) $\wedge$ *refina* ($Espec_{M\ p}$ (m), $Espec_{R\ p}$ (m), a.op) )

Donde a.op indica relación de conocimiento a, por la que se conoce a la operación op y el predicado *refina* se define:

*refina*:: CESPEC X CESPEC X {OP} $\rightarrow$ BOOL

el significado intuitivo del predicado es:

*refina* (S, S', op) = true , si S es una cláusula de especialización válida obtenida al agregar (en secuencia o paralelo) la operación op en la cláusula de especialización S'.
= false , en caso contrario.

La definición es:

| | | |
|---|---|---|
| *refina* (op, $\sigma$, op) | = true | |
| *refina* ($\sigma$, resto, op) | = false | |
| *refina* (op1, $\sigma$, op) | = false | si op1$\neq$op |
| *refina* (op1; resto, op2; resto', op) | = *refina* (resto, resto', op) | si op1=op2 |
| | = *iguales* (resto, op2; resto') | si op1=op |
| | = false | en caso contrario |
| *refina* (op1\|\| resto, op2\|\| resto', op) | = *refina* (resto, resto', op) | si op1=op2 |
| | = *iguales* (resto, op2\|\| resto') | si op1=op |
| | = false | en caso contrario |
| *refina* (op1\|\| resto, op2; resto', op) | = *iguales* (resto, op2; resto') | si op1=op |
| | = false | en caso contrario |
| *refina* (op1; resto, op2\|\| resto', op) | = *iguales* (resto, op2 \|\| resto') | si op1=op |
| | = false | en caso contrario |

donde el predicado *iguales* (S, S') = true , si la cláusula de especialización S tiene las mismas operaciones y en el mismo orden que la cláusula de especialización S'.
= false , en caso contrario.

| | | |
|---|---|---|
| *iguales* (resto, $\sigma$) | = false | |
| *iguales* ($\sigma$, resto) | = false | |
| *iguales* ($\sigma$, $\sigma$) | = true | |
| *iguales* (op1; resto, op2; resto') | = *iguales* (resto, resto') | si op1=op2 |
| | = false | en caso contrario |

| | | |
|---|---|---|
| *iguales* (op1\|\| resto, op2\|\| resto') | = *iguales* (resto, resto') | si op1=op2 |
| | = false | en caso contrario |
| *iguales* (op1\|\| resto, op2; resto') | = false | |
| *iguales* (op1; resto, op2\|\| resto') | = false | |

*Definición del Resultado del Refinamiento sintáctico de Participante*

Sea R ∈ CONTRATO, R' = *Refp*(R, M) es un Refinamiento de Participante de R por M ⇔

(1) *Refp* es aplicable a R por M

(2) $\forall\ p\ \in (Part(R) - Part(M)) : p \in Part(R') \wedge Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_R(p)$

(3) $\forall\ p\ \in Part(M) : p \in Part(R')$ :

(a) $Con_{R'}(p) = Con_R(p)$

(b) $\forall\ m \in Int_R(p) - Int_M(p) : m \in Int_{R'}(p) \wedge Espec_{R',p}(m) = Espec_{R\ p}(m)$

( c) $\forall\ m \in Int_M(p) : m \in Int_{R'}(p) \wedge Espec_{R',p}(m) = Espec_{M\ p}(m)$
$\wedge\ Pcond_{R',p}(m) = Pcond_{R\ p}(m)$

(4) Inv(R') = Inv(R)

**Coarsening sintáctico de participante:** *Coap*

*Definición de Aplicabilidad*

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.

El operador ***Coap*** (coarsening sintáctico de participante) ∈ OPERADOR es aplicable a R por M ⇔

$\forall\ p\ \in Part(M)$:
(1) $p\ \in Part(R)$
(2) $\forall\ m \in Int_M(p)$: $m \in Int_R(p) \wedge$
($\forall\ op \in Espec_{R\ p}(m)$: *refina* ($Espec_{R\ p}(m)$, $Espec_{M\ p}(m)$, op) )

*Definición del Resultado del Coarsening sintáctico de Participante*

Sea R ∈ CONTRATO. R' = *Coap*(R, M) es un Coarsening de Participante de R por M ⇔

(1) *Coap* es aplicable a R por M

(2) $\forall\ p\ \in$ Part(R) - Part (M) : $p\ \in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_R(p)$

(3) $\forall\ p\ \in$ Part (M) : $p\ \in$ Part(R') :

(a) $Con_{R'}(p) = Con_R(p)$

(b) $\forall\ m\ \in Int_R(p) - Int_M(p) : m\ \in Int_{R'}(p) \wedge Espec_{R',\ p}(m) = Espec_{R\ p}(m)$

( c) $\forall\ m\ \in Int_M(p) : m\ \in Int_{R'}(p) \wedge Espec_{R',\ p}(m) = Espec_{M\ p}(m)\ \wedge$
$Pcond_{R',\ p}(m) = Pcond_{R\ p}(m)$

(4) Inv(R') = Inv(R)

**Refinamiento sintáctico de contexto:** *Refc*

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador ***Refc*** (refinamiento sintáctico de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

$\forall\ \ p\ \in$ Part (M):

(1) $p\ \in$ Part(R)

(2) $Con_R(p) \subseteq Con_M(p)$
(3) $\forall$ a.q: a.q $\in Con_M(p) \Rightarrow$ q $\in$ Part(R)

*Definición del Resultado del Refinamiento sintáctico de Contexto*

Sea R $\in$ CONTRATO, R' = *Refc*(R, M) es un Refinamiento de Contexto de R por M $\Leftrightarrow$

(1) *Refc* es aplicable a R por M

(2) $\forall\ p\ \in$ (Part(R) - Part (M)) : $p\ \in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_R(p)$

(3) $\forall\ p\ \in$ Part (M) : $p\ \in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge$
$Con_{R'}(p) = Con_M(p)$

(4) Inv (R') = Inv ( R)

**Coarsening sintáctico de contexto:** *Coac*

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador ***Coac*** (coarsening sintáctico de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

$\forall$ $p$ $\in$ Part (M):
(1) $p$ $\in$ Part(R)

(2) $Con_M(p) \subseteq Con_R(p)$

(3) $\forall$ a.q $\in (Con_R(p) - Con_M(p)) : \forall$ $m$ $\in Int_R$ (p): ($\forall$ b.n $\in Espec_{R\ p}$ (m): b $\neq$ q)

(4) $\forall$ a.q $\in (Con_R(p) - Con_M(p))$: $\forall$ $m$ $\in Int_R$ (p): a.q no ocurre en $Pcond_{R\ p}$ (m)

(5) $\forall$ a.q $\in (Con_R(p) - Con_M(p))$: a.q no ocurre en Inv(R)

**Definición del Resultado del Coarsening sintáctico de Contexto**

Sea R $\in$ CONTRATO, R'= *Coac*(R, M) es un Coarsening sintáctico de Contexto de R por M $\Leftrightarrow$

(1) *Coac* R es aplicable a R por M

(2) $\forall$ $p$ $\in$ Part(R) - Part (M) : $p$ $\in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_R(p)$

(3) $\forall$ $p$ $\in$ Part (M) : $p$ $\in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_M(p)$

(4) Inv (R') = Inv ( R)

## *7.3 Operadores de Reuso Semántico*

De la misma forma en que presentamos los operadores de reuso sintáctico en la sección anterior, en esta sección presentamos la notación matemática para definir operadores de reuso semántico que fueran introducidos informalmente en el Capítulo 4. Estos operadores surgen en este trabajo como consecuencia de haber introducido elementos semánticos a un contrato de reuso.
En el siguiente apartado presentamos los *patterns* de refinamiento que nos serán de utilidad al definir matemáticamente los operadores de refinamiento y coarsening semántico de contexto.

**Patterns de Refinamiento**

Para poder expresar en una forma general la aplicabilidad del refinamiento y *coarsening* de un invariante, utilizamos el concepto de *patterns de refinamiento*.

En [DL96] se introducen *patterns* de refinamiento independientes del dominio. Un *pattern* de refinamiento es un árbol AND de un nivel, cuyos elementos son fórmulas, en nuestro caso expresiones booleanas OCL bien formadas, tales que cumplen la siguiente condición:

Sean $G_1$, $G_2$,..,$G_n$ las fórmulas en las hojas del árbol. Sea G la fórmula en la raíz del árbol, entonces

1. $G_1 \wedge G_2 \wedge ... \wedge G_n \models G$ (*entailment)*
2. $G_1 \wedge G_2 \wedge ... \wedge G_n \not\models$ false (consistencia)

Bajo estas condiciones definimos el predicado *instancian:*
*instancian*:: EXPOCL X EXPOCL $\rightarrow$ BOOL

*instancian* (exp, exp´) = true, si exp y exp´ son expresiones OCL que instancian un pattern de refinamiento ( árbol AND) donde exp es una instancia de la raíz del árbol y exp´ es una instancia de la conjunción de las hojas.
= false, en caso contrario

Estos *patterns* son genéricos, es decir, pueden ser instanciados para situaciones específicas. En [DL96] se define una biblioteca de *patterns*, cuya correctitud y completitud ha sido demostrada.
En la siguiente figura se muestra un ejemplo de *pattern* de refinamiento.

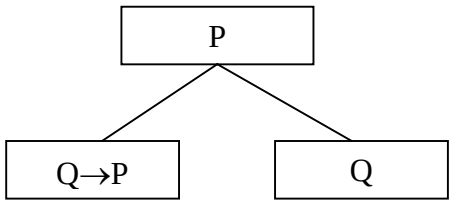


**Fig.7.1 Un ejemplo de pattern de refinamiento**

**Refinamiento semántico de participante: *Refsp***

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.
El operador *Refsp* ( refinamiento semántico de participante) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

$\forall$ p $\in$ Part (M):
(1) p $\in$ Part(R)
(2) $\forall$ m $\in$ $Int_M(p)$: m $\in$ $Int_R$ (p)
(3)$\forall$ m $\in$ $Int_M(p)$: $\forall$ e $\in$ ($Pcond_{M\ p}(m)$ - $Pcond_{R\ p}(m)$): *bienFormada*(e, R) $\wedge$ *noExiste*(e, $Pcond_{R\ p}(m)$)
(4)$\forall$ *m* $\in$ $Int_M(p)$: Inv(R)$\rightarrow$Inv(R)[$Pcond_{M\ p}(m)$]

donde el predicado *noExiste* se define:

*noExiste*:: EXPOCL X $\mathcal{P}$(EXPOCL) $\rightarrow$ BOOL

el significado es:
*noExiste* (e, pCond) = true , si los términos *primados* en la expresión e no existen en la post-condición pCond.
= false , en caso contrario.

*Definición del Resultado del Refinamiento semántico de Participante*

Sea R $\in$ CONTRATO, R'= *Refsp*(R, M) es un Refinamiento semántico de Participante de R por M $\Leftrightarrow$

(1) *Refsp* es aplicable a R por M

(2) $\forall$ p $\in$ Part(R) - Part (M) : p $\in$ Part(R') $\wedge$ $Int_{R'}(p) = Int_R(p) \wedge Con_{R'}(p) = Con_R(p)$

(3) $\forall$ p $\in$ Part (M) : p $\in$ Part(R') :

(a) $Con_{R'}(p) = Con_R(p)$
(b) $\forall$ m $\in$ $Int_R$ (p) - $Int_M$ (p) : m $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R', p}$ (m) = $Espec_{R\,p}$ (m) $\wedge$ $Pcond_{R'}$ (m) = $Pcond_R$ (m)
( c) $\forall$ m $\in$ $Int_M$ (p) : m $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R', p}$ (m) = $Espec_{R\,p}$(m) $\wedge$ $Pcond_{R', p}$ (m) = $Pcond_{M\,p}$ (m)

(4) Inv(R') = Inv(R)

**Refinamiento semántico de contexto: *Refsc***

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador *Refsc* ( refinamiento semántico de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

(1) *bienFormada* (Inv(M), R)
(2) *instancian* (Inv(R) , Inv (M))
(3) $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$ (p) : Inv( M)$\rightarrow$ Inv( M) [ $Pcond_{R\,p}$ (m)]

*Definición del Resultado del Refinamiento semántico de Contexto*

Sea R $\in$ CONTRATO, R'= *Refsc* (R, M) es un Refinamiento semántico de Contexto de R por M $\Leftrightarrow$
(1) *Refsc* es aplicable a R por M

(2) $\forall$ p $\in$ Part(R) : p $\in$ Part(R') $\wedge$ $Int_{R'}$(p) = $Int_R$ (p) $\wedge$ $Con_{R'}$ (p) = $Con_R$(p)

(3) Inv(R') = Inv (M)

**Coarsening semántico de participante: *Coasp***

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador *Coasp* ( coarsening semántico de participante) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$
$\forall$ *p* $\in$ Part (M):
(1) *p* $\in$ Part(R)
(2) $\forall$ *m* $\in$ $Int_M$(p): *m* $\in$ $Int_R$ (p)
(3) $\forall$ m $\in$ $Int_M$(p): $\forall$ e $\in$ $Pcond_{M\,p}$(m): e $\in$ $Pcond_{R\,p}$(m) $\wedge$ Inv(R)$\rightarrow$Inv(R)[$Pcond_{M\,p}$(m)]

*Definición del Resultado del Coarsening semántico de Participante*

Sea R $\in$ CONTRATO, R'= *Coasp*(R, M) es un Coarsening semántico de Participante de R por M $\Leftrightarrow$

(1) *Coasp* es aplicable a R por M

(2) $\forall$ *p* $\in$ Part(R) - Part (M) : *p* $\in$ Part(R') $\wedge$ $Int_{R'}$(p) = $Int_R$ (p) $\wedge$ $Con_{R'}$ (p) = $Con_R$(p)

(3) $\forall$ *p* $\in$ Part (M) : *p* $\in$ Part(R') :

(a) $Con_{R'}$ (p) = $Con_R$(p)
(b) $\forall$ *m* $\in$ $Int_R$ (p) - $Int_M$ (p) : *m* $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R',\,p}$ (m) = $Espec_{R\,p}$ (m) $\wedge$ $Pcond_{R'}$ (m) = $Pcond_R$ (m)
( c) $\forall$ *m* $\in$ $Int_M$ (p) : *m* $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R',\,p}$ (m) = $Espec_{R\,p}$(m) $\wedge$ $Pcond_{R',\,p}$ (m) = $Pcond_{M\,p}$ (m)

(4) Inv(R') = Inv(R)

**Coarsening semántico de contexto: *Coasc***

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador *Coasc* ( coarsening semántico de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

(1) *bienFormada* (Inv (M) , R)
(2) *instancian* (Inv (M), Inv(R))
(3)$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$ (p) : Inv( M)$\rightarrow$ Inv( M) [ $Pcond_{R\,p}$ (m)]

*Definición del Resultado del Coarsening semántico de Contexto*

Sea R $\in$ CONTRATO, R'=*Coasc* (R, M) es un Coarsening semántico de Contexto de R por M $\Leftrightarrow$

(1) *Coasc* es aplicable a R por M

(2) $\forall$ *p* $\in$ Part ( R) : *p* $\in$ Part(R') $\wedge$ $Int_{R'}$(p) = $Int_R$ (p) $\wedge$ $Con_{R'}$ (p) = $Con_R$(p)

(3) Inv(R') = Inv (M)

**Redefinición semántica de participante:** *Red*

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador *Red* (redefinición semántica de participante) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

$\forall$ p $\in$ Part (M):
(1) p $\in$ Part(R)
(2)$\forall$ *m* $\in$ $Int_M$(p): $\forall$ e $\in$ $Pcond_{M\,p}$(m): *bienFormada*(e, R)
(3)$\forall$ *m* $\in$ $Int_M$(p): Inv(R)$\rightarrow$Inv(R)[$Pcond_{M\,p}$(m)]

*Definición del Resultado de la Redefinición Semántica de Participante*

Sea R $\in$ CONTRATO, R'= *Red*(R, M) es una Redefinición semántica de Participante de R por M $\Leftrightarrow$

(1) *Red* es aplicable a R por M

(2) $\forall$ p $\in$ Part(R) - Part (M) : p $\in$ Part(R') $\wedge$ $Int_{R'}$(p) = $Int_R$ (p) $\wedge$ $Con_{R'}$ (p) = $Con_R$(p)

(3) $\forall$ p $\in$ Part (M) : p $\in$ Part(R') :

(a) $Con_{R'}$ (p) = $Con_R$(p)
(b) $\forall$ m $\in$ $Int_R$ (p) - $Int_M$ (p) : m $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R'\,p}$ (m) = $Espec_R$ (m) $\wedge$ $Pcond_{R'}$ (m) = $Pcond_R$ (m)
( c) $\forall$ m $\in$ $Int_M$ (p) : m $\in$ $Int_{R'}$ (p) $\wedge$ $Espec_{R'\,p}$ (m) = $Espec_{R\,p}$(m) $\wedge$ $Pcond_{R'\,p}$(m) = $Pcond_{M\,p}$ (m)

(4) Inv(R') = Inv(R)

**Redefinición semántica de contexto:** *Redc*

*Definición de Aplicabilidad*

Sea R $\in$ CONTRATO un contrato de reuso bien formado y M $\in$ CONTRATO un modificador de reuso.

El operador *Redc* (redefinición semántica de contexto) $\in$ OPERADOR es aplicable a R por M $\Leftrightarrow$

(1) *bienFormada* (Inv(M), R)
(2)$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$ (p) : Inv(M)$\rightarrow$Inv(M)[$Pcond_{R\ p}$(m)]

*Definición del Resultado de la Redefinición semántica de Contexto*

Sea R $\in$ CONTRATO, R'=*Redc*(R, M) es una Redefinición semántica de Contexto de R por M $\Leftrightarrow$

(1) *Redc* es aplicable a R por M
(2) $\forall$ p $\in$ Part(R) : p $\in$ Part(R') $\wedge$ $Int_{R'}$(p) = $Int_R$ (p) $\wedge$ $Con_{R'}$ (p) = $Con_R$(p)
(3) Inv(R') = Inv (M)

# Capítulo 8: Propiedades de aplicabilidad de operadores de reuso

Basándonos en el modelo matemático para contratos y operadores de reuso presentado en el Capítulo 7 y en el análisis de conflictos del capítulo 6, es posible expresar y probar propiedades concernientes a la aplicabilidad de operadores de reuso. El número total de los casos de análisis posibles está dado por la combinación de todos los operadores de reuso semántico entre sí y de ellos con los operadores de reuso sintáctico. En cada combinación puede presentarse o no la posibilidad de conflicto. En el caso de posible conflicto pueden encontrarse situaciones particulares donde éste nunca se produzca. Este hecho se define como una propiedad. En las secciones siguientes definimos, probamos y ejemplificamos propiedades de aplicabilidad que surgen del análisis particular de cada tipo de conflicto planteado en el capítulo 6 y sus casos específicos de no ocurrencia.

## *8.1 Propiedad de Aplicabilidad de Operadores semánticos de participante*

En esta sección presentamos una propiedad de aplicabilidad general para la composición de operadores de reuso semántico de participante.
Esta propiedad describe en que casos pueden aplicarse *dos operadores de reuso semántico de participante* sobre un contrato de reuso base sin que se produzca ningún tipo de conflicto.

**Propiedad 1: Composición de Operadores semánticos de participante**

*Si* $M_1$ y $M_2$ son modificadores semánticos de participante aplicables a R, $R_1$ es el contrato resultante de aplicar M1 a R *y*
si el mismo participante está en ambos modificadores, pero las operaciones que se modifican son distintas *entonces* $M_2$ es aplicable a $R_1$.

La demostración presentada a continuación muestra la composición de refinamientos. Demostraciones similares, bajo las mismas hipótesis, pueden ser hechas componiendo el resto de los operadores semánticos de participante.

*Demostración*:

Sean $M_1$ y $M_2$ modificadores de refinamiento semántico de participante
Sea R refinable semánticamente de participante por $M_2$
*y* $R_1 = Refsp(R, M_1)$
*y* $\forall\, p \in (Part(M_1) \cap Part(M_2)): (\forall\, m \in Int_{M1}(p): m \notin Int_{M2}(p))$

Para probar la propiedad necesitamos mostrar que $R_1$ es refinable semánticamente de participante por $M_2$ o sea:
$\forall\, p \in Part(M_2)$ :
(1) $p \in Part(R_1)$
(2) $\forall\, m \in Int_{M2}(p): m \in Int_{R1}(p)$
(3) $\forall\, m \in Int_{M2}(p): \forall\, e \in (Pcond_{M2\,p}(m) - Pcond_{R1\,p}(m))$: *bienFormada*$(e, R_1) \wedge$ *noExiste*$(e, Pcond_{R1\,p}(m))$
(4) $\forall\ m \in Int_M(p): Inv(R_1) \rightarrow Inv(R_1)[Pcond_{M2\,p}(m)]$
Contamos con las siguientes hipótesis:

a) $R_1$ = Refsp (R,$M_1$ )
b) R es refinable semánticamente de participante por $M_2$ ,por lo tanto:

$\forall$ p $\in$ Part ($M_2$):

($\forall$ m $\in$ $Int_{M2}$(p): )$\forall$ m $\in$ $Int_{M2}$(p): $\forall$ e $\in$ ($Pcond_{M2\ p}$(m) - $Pcond_{R\ p}$(m)): *bienFormada*(e, R) $\wedge$ *noExiste*(e, $Pcond_{R\ p}$(m))

$\forall$ m $\in$ $Int_M$(p): Inv(R)$\rightarrow$Inv(R)[$Pcond_{M2\ p}$(m)]

Por a) tenemos que R tiene idénticos participantes y operaciones que $R_1$ y por b) $\forall$ m $\in$ $Int_{M2}$(p): $\forall$ e $\in$ ($Pcond_{M2\ p}$(m) - $Pcond_{R\ p}$(m)): *bienFormada*(e, R) por lo tanto:
$\forall$ m $\in$ $Int_{M2}$(p): $\forall$ e $\in$ ($Pcond_{M2\ p}$(m) - $Pcond_{R1\ p}$(m)): *bienFormada*(e, R1)
Ahora como: $\forall$ *p* $\in$ ( Part ($M_1$) $\cap$ Part ($M_2$)): ($\forall$ m $\in$ $Int_{M1}$(p): m $\notin$ $Int_{M2}$(p) )
y por a) se tiene que $\forall$ m $\in$ $Int_{M2}$(p): $Pcond_{R\ p}$ (m) = $Pcond_{R1\ p}$ (m), entonces se cumple (3);
por a) Inv( $R_1$) = Inv( R) y por b) se cumple (4);
por a) se cumple (1);
por a) y b) se cumple (2).
Queda demostrado que $R_1$ es refinable semánticamente de participante por $M_2$.

*Un Ejemplo de la Propiedad 1*

El ejemplo de la figura 8.1 muestra al modificador M1 que representa un Refinamiento semántico de *extraer* sobre el participante CAhorro, cuya post-condición no había sido expresada. Por otro lado M2 representa otro Refinamiento semántico de participante sobre la operación *depositar* del mismo participante. Dado que las operaciones involucradas son distintas no se genera conflicto: se cumple la propiedad 1.

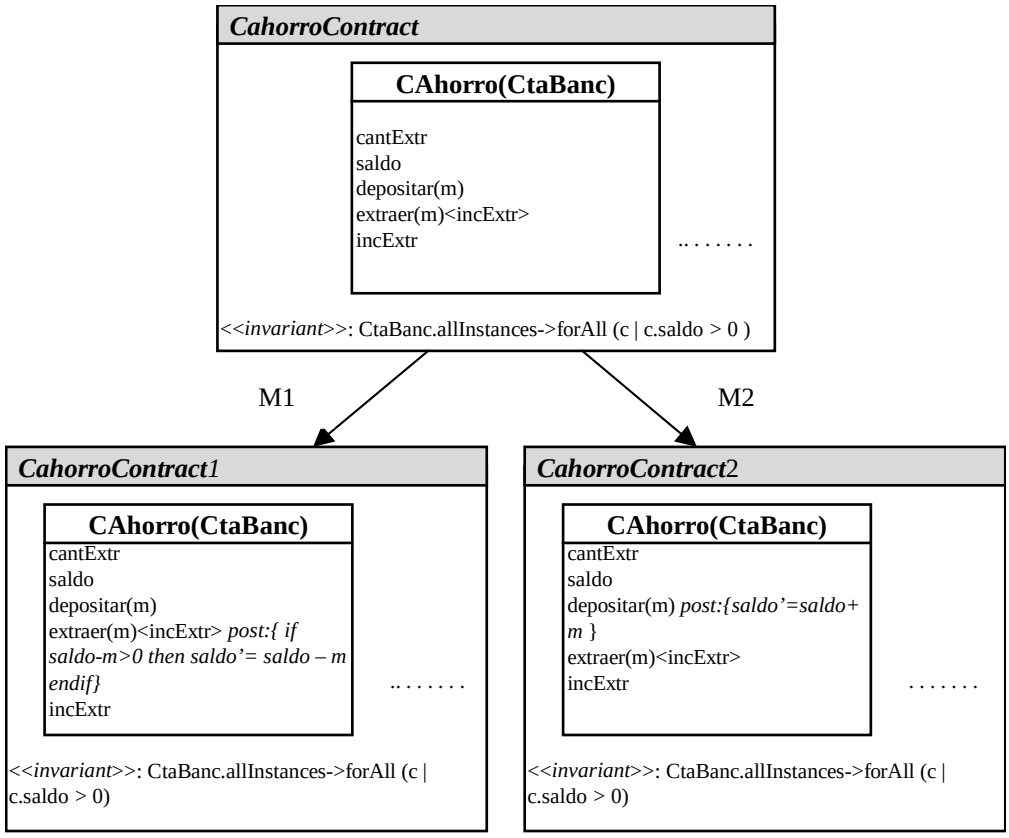


*Fig. 8.1 Propiedad 1*

## *8.2 Propiedades relacionadas con la no ocurrencia de conflictos*

En esta sección definimos y demostramos propiedades de aplicabilidad que surgen del análisis particular de cada tipo de conflicto que puede producirse al combinar o integrar operadores durante la etapa de evolución. Estas propiedades se relacionan con los casos específicos de no ocurrencia de conflictos. Presentamos ejemplos de cumplimiento de las propiedades, acompañando algunas de las definiciones.

### 8.2.1 Conflicto de Referencia Colgada en Invariante

Analizando el Conflicto de Referencia Colgada en Invariante y los operadores que participan en su ocurrencia, surgen casos en los que este conflicto no ocurre, por lo que pueden enunciarse las siguientes propiedades de aplicabilidad:

**Composición de cancelación sintáctica y operador semántico de contexto**

*Si* $M_1$ es un modificador de cancelación de participante o contexto y $M_2$ es un modificador de refinamiento o redefinición semántica de contexto y son aplicables a R *y* $R_1$ es el contrato resultante de aplicar M1 a R *entonces*
$M_2$ es aplicable a $R_1$ sii el elemento cancelado no ocurre en $M_2$.

**Propiedad 2.a:**
Sea $M_2$ un modificador de refinamiento o redefinición semántica de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1$ = Canp (R, $M_1$ ) cancelación sintáctica de participante de R por $M_1$

*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow \forall\ p \in \text{Part}(M_1) : \forall\ m \in \text{Int}_{M1}(p)$: m no ocurre en $\text{Inv}(M_2)$

*Demostración*:

Sea $M_2$ modificador de refinamiento semántico de contexto

Sea (a) R refinable semánticamente de contexto por $M_2$
(b) $R_1 = \text{Canp}(R, M_1)$

Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es refinable semánticamente de contexto por $M_2 \Rightarrow$

$$\forall\ p \in \text{Part}(M_1) : \forall\ m \in \text{Int}_{M1}(p)\text{: m no ocurre en Inv}(M_2)$$

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (c), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

(1) *bienFormada* ($\text{Inv}(M_2)$, $R_1$)
(2) *instancian* ($\text{Inv}(R_1)$ , $\text{Inv}(M_2)$)
(3)$\forall$ p $\in \text{Part}(R_1) : \forall\ m \in \text{Int}_{R1}(p) : \text{Inv}(M_2) \rightarrow \text{Inv}(M_2)[\text{Pcond}_{R1\ p}(m)]$

Por (b), definición de Resultado de cancelación sintáctica de participante, tenemos que:

$\forall\ p \in \text{Part}(M_1) : p \in \text{Part}(R_1) : \forall\ m \in (\text{Int}_R(p) - \text{Int}_{M1}(p))$, o sea

$\forall\ p \in \text{Part}(M_1)$: ( $m \in \text{Int}_{M1}(p) \rightarrow m \notin \text{Int}_{R1}(p)$)
y como por (c ) *bienFormada* ($\text{Inv}(M_2)$, $R_1$)
queda demostrado que

$\forall\ p \in \text{Part}(M_1) : \forall\ m \in \text{Int}_{M1}(p)$: m no ocurre en Inv ( $M_2$ )

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) $\forall\ p \in \text{Part}(M_1) : \forall\ m \in \text{Int}_{M1}(p)$: m no ocurre en Inv ( $M_2$ ) $\Rightarrow$ $R_1$ es refinable semánticamente de contexto por $M_2$

O sea, debemos probar que:

(1) *bienFormada* ($\text{Inv}(M_2)$, $R_1$)
(2) *instancian* ($\text{Inv}(R_1)$, $\text{Inv}(M_2)$)
(3)$\forall$ p $\in \text{Part}(R_1)$: $\forall\ m \in \text{Int}_{R1}(p) : \text{Inv}(M_2) \rightarrow \text{Inv}(M_2)[\text{Pcond}_{R1\ p}(m)]$

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R), Inv ($M_2$))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de cancelación sintáctica de participante, tenemos que:

Part($R_1$) = Part ( R)
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :$\forall$ m $\in$ ($Int_R$(p) - $Int_{M1}$(p)), o sea

$\forall$ *p* $\in$ Part ($M_1$): ( m $\in$ $Int_{M1}$(p) $\rightarrow$ m $\notin$ $Int_{R1}$(p))

Ahora, por (a) y como por (d) vale que

$\forall$ *p* $\in$ Part ($M_1$) :$\forall$ m $\in$ $Int_{M1}$(p): m no ocurre en Inv ( $M_2$ )

los participantes y operaciones de R y $R_1$ coinciden por (b), salvo las operaciones en $M_1$, el punto (1) que demostrado.

Por (a ) vale: *instancian* (Inv(R) , Inv ($M_2$))

Y como por (b), Inv($R_1$) = Inv (R) queda demostrado el punto (2).

Por (a ), $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]

los participantes y operaciones de R y $R_1$ coinciden por (b), salvo las operaciones en $M_1$, que no están en $R_1$,
entonces $\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : $Pcond_{R\ p}$ (m) = $Pcond_{R1\ p}$ (m) , reemplazando en la tercer condición de (a), el punto (3) queda demostrado.

Por lo tanto queda demostrado que $R_1$ es refinable semánticamente de contexto por $M_2$.

*Un ejemplo de la Propiedad 2.a*

En la figura 8.2, el modificador M1 cancela sintácticamente a *tope* en el participante CAhorro, mientras M2 representa un Refinamiento semántico de contexto, que modifica al invariante del contrato. Dado que *tope* no ocurre en la expresión invariante resultante, no se genera conflicto en la combinación: se cumple la propiedad 2.a.

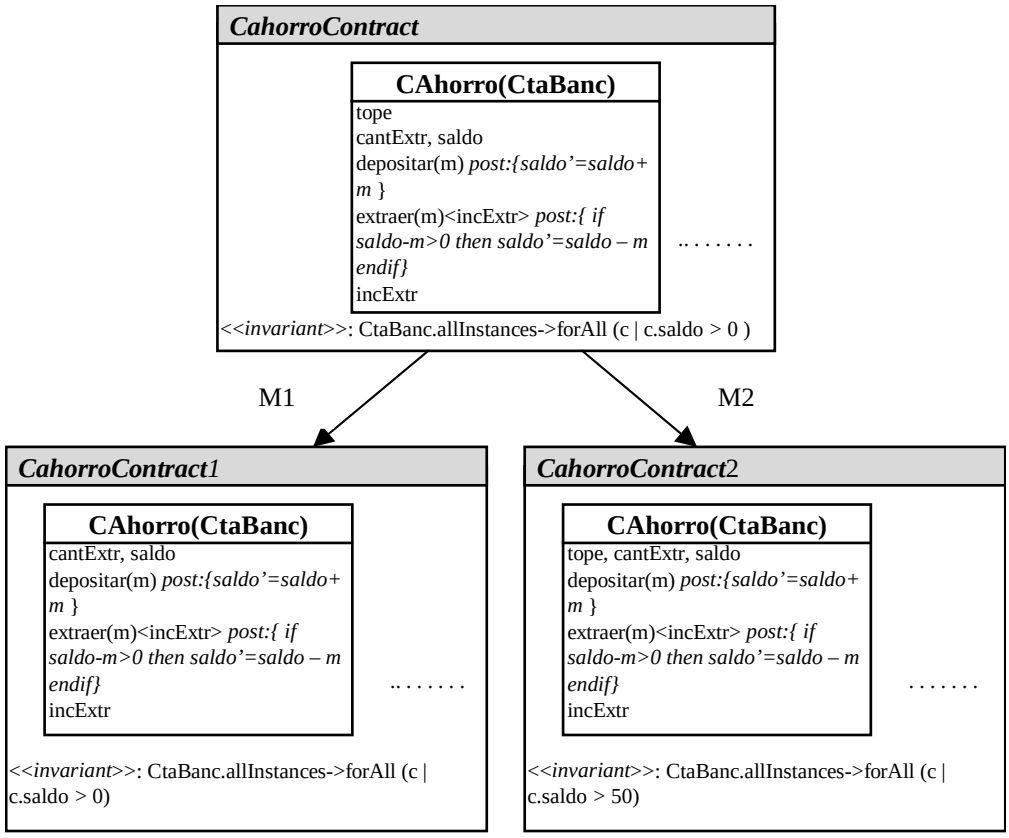


*Fig 8.2 Propiedad 2.a*

**Propiedad 2.b:**
Sea $M_2$ un modificador de refinamiento o redefinición semántica de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1$ = Canc (R, $M_1$ ) cancelación sintáctica de contexto de R por $M_1$
*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow \forall\ p \in$ Part ($M_1$) : p no ocurre en Inv ( $M_2$ )

*Demostración*:

Sea $M_2$ modificador de refinamiento semántico de contexto

Sea (a) R refinable semánticamente de contexto por $M_2$
(b) $R_1$ = Canc (R, $M_1$ )
Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es refinable semánticamente de contexto por $M_2 \Rightarrow$

$\forall\ p \in$ Part ($M_1$) : p no ocurre en Inv ( $M_2$ )

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (c), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

(1) *bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($R_1$) , Inv ($M_2$))

(3)$\forall$ p $\in$ Part($R_1$): $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R1\ p}$ (m)]

Por (b), definición de Resultado de cancelación sintáctica de contexto, tenemos que:

Part ($R_1$) = (Part(R) - Part ($M_1$)), o sea $\forall$ *p* $\in$ Part ($R_1$): p $\notin$ Part($M_1$)

y como por (c ) vale *bienFormada* (Inv($M_2$), $R_1$), queda demostrado que

$\forall$ *p* $\in$ Part ($M_1$) : p no ocurre en Inv ( $M_2$ )

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) $\forall$ *p* $\in$ Part ($M_1$) : p no ocurre en Inv ( $M_2$ ) $\Rightarrow$ $R_1$ es refinable semánticamente de contexto por $M_2$

O sea, debemos probar que:

(1) *bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($R_1$) , Inv ($M_2$))
(3)$\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R1\ p}$ (m)]

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R) , Inv ($M_2$))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de cancelación sintáctica de contexto, tenemos que:

Part ($R_1$) = (Part(R) - Part ($M_1$)), o sea $\forall$ *p* $\in$ Part ($R_1$): p $\notin$ Part($M_1$)

$\forall$ *p* $\in$ Part ($M_1$): ( m $\in$ $Int_{M1}$(p) $\rightarrow$ m $\notin$ $Int_{R1}$(p))

Ahora, por (a) y como por (d ) vale que

$\forall$ *p* $\in$ Part ($M_1$) : p no ocurre en Inv ( $M_2$ )

los participantes y operaciones de R y $R_1$ coinciden por (b), salvo los participantes en $M_1$ , el punto (1) que demostrado.

Por (a ), *instancian* (Inv(R) , Inv ($M_2$))
Y como por (b), Inv($R_1$) = Inv (R) queda demostrado el punto (2).

Por (a ), $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]
los participantes de R y $R_1$ coinciden por (b), salvo los de $M_1$, que no están en $R_1$,
entonces, $\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : $Pcond_{R\ p}$ (m) = $Pcond_{R1\ p}$ (m) , reemplazando en la tercera condición de (a), el punto (3) queda demostrado.

Por lo tanto queda demostrado que $R_1$ es refinable semánticamente de contexto por $M_2$ .

**Propiedad 2.c:**
Sea $M_2$ un modificador de refinamiento o redefinición semántica de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1$ = Coarc (R, $M_1$ ) coarsening sintáctico de contexto de R por $M_1$
*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow \forall\ p \in$ Part ($M_1$): $\forall$ a.r $\in$ ($Con_R$ (p) - $Con_{M1}$ (p)) : a.r no ocurre en Inv ( $M_2$ )

*Demostración*:

Sea $M_2$ modificador de refinamiento semántico de contexto

Sea (a) R refinable semánticamente de contexto por $M_2$
(b) $R_1$ = Coarc (R, $M_1$ )

Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es refinable semánticamente de contexto por $M_2 \Rightarrow$

$\forall\ p \in$ Part ($M_1$) : $\forall$ a.r $\in$ ($Con_R$ (p) - $Con_{M1}$ (p)) : a.r no ocurre en Inv ( $M_2$ )

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (c), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

(1) *bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($R_1$) , Inv ($M_2$))
(3)$\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R1\ p}$ (m)]

Por (b), definición de Resultado de coarsening sintáctico de contexto, tenemos que:

$\forall\ p \in$ Part ($M_1$) : $p \in$ Part ($R_1$) :$\forall$ a.r $\in$ $Con_{R1}$ (p): a.r $\in$ $Con_{M1}$(p), o sea

$\forall\ p \in$ Part ($M_1$): a.r $\in$ $Con_{M1}$(p) $\rightarrow$ a.r $\notin$ ($Con_R$ (p) - $Con_{M1}$ (p))
y como por (c ) vale *bienFormada* (Inv($M_2$), $R_1$), queda demostrado que

$\forall\ p \in$ Part ($M_1$) :$\forall$ a.r $\in$ ($Con_R$ (p) - $Con_{M1}$ (p)): a.r no ocurre en Inv ( $M_2$ )

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) $\forall\ p \in$ Part ($M_1$) :$\forall$ a.r $\in$ ($Con_R$ (p) - $Con_{M1}$ (p)): a.r no ocurre en Inv ( $M_2$ ) $\Rightarrow$ $R_1$ es refinable semánticamente de contexto por $M_2$

O sea, debemos probar que:

(1) *bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($R_1$) , Inv ($M_2$))

(3)$\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R1\ p}$ (m)]

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R) , Inv ($M_2$))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de coarsening sintáctico de contexto, tenemos que:

Part($R_1$) = Part ( R)
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :$\forall$ a.r $\in$ $Con_{M1}$(p), o sea

$\forall$ *p* $\in$ Part ($M_1$): ( a.r $\in$ $Con_{M1}$(p) $\rightarrow$ a.r $\notin$ ($Con_R$ (p) - $Con_{M1}$ (p)))

Ahora, por (a) y como por (d) vale que

$\forall$ *p* $\in$ Part ($M_1$) :$\forall$ a.r $\in$ ($Con_R$ (p) - $Con_{M1}$ (p)): a.r no ocurre en Inv ( $M_2$ )

los participantes y operaciones de R y $R_1$ coinciden por (b), salvo en las relaciones que no aparecen en $M_1$, pero éstas no ocurren en Inv ( $M_2$ ), por lo tanto el punto (1) que demostrado.

Por (a ) vale *instancian* (Inv(R) , Inv ($M_2$)), como por (b), Inv($R_1$) = Inv (R) queda demostrado el punto (2).

Por (a ), $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv( $M_2$)[ $Pcond_{R\ p}$ (m)]
los participantes de R y $R_1$ coinciden por (b), salvo las relaciones que no aperecen en $M_1$ y por lo tanto no están en $R_1$,
entonces, $\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : $Pcond_{R\ p}$ (m) = $Pcond_{R1\ p}$ (m) , reemplazando en la tercer condición de (a), el punto (3) queda demostrado.

Por lo tanto queda demostrado que $R_1$ es refinable semánticamente de contexto por $M_2$ .

*Un Ejemplo de la Propiedad 2.c*

En la figura 8.3, el modificador M1 representa un *coarsening* sintáctico de contexto de la relación de conocimiento *clientes* entre el participante Banco y el participante Titular, mientras que M2 es un modificador de refinamiento semántico de contexto; dado que la relación *clientes* no aparece en la expresión que refina al invariante, no se genera conflicto y se cumple la propiedad 2.c.

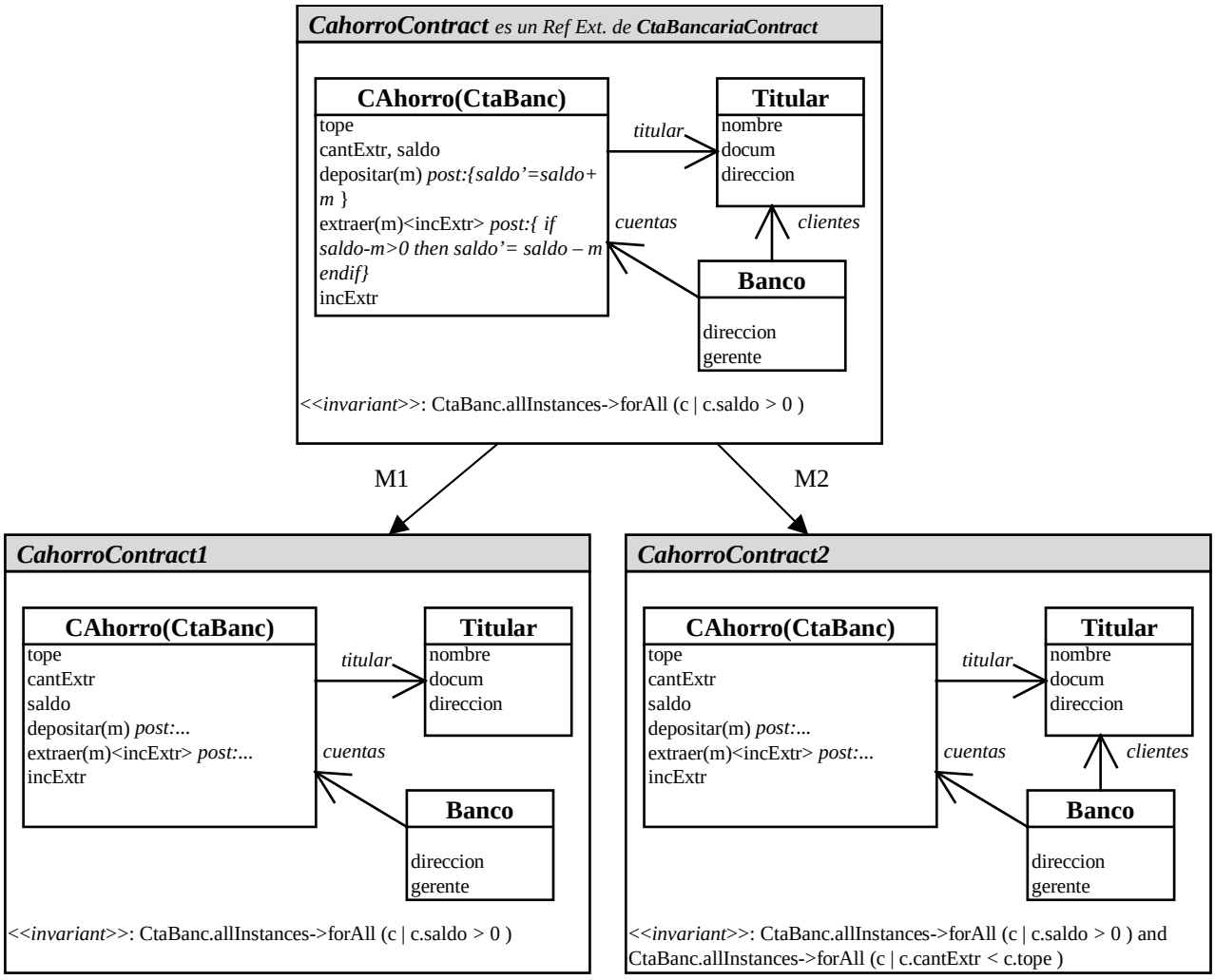


*Fig.8.3 Propiedad 2.c*

### 8.2.2 Conflicto de Inconsistencia Semántica de Operación

Analizando el Conflicto de Inconsistencia Semántica de Operación y los operadores que participan en su ocurrencia, surgen casos en los que este conflicto no ocurre, por lo que pueden enunciarse las siguientes propiedades de aplicabilidad:

*Consistencia en la Composición semántica de participante*

*Si* $M_1$ y $M_2$ son modificadores de refinamiento o coarsening semántico de participante aplicables a R *y* $R_1$ es el contrato resultante de aplicar M1 a R *y* ambos modifican la misma operación, pero las expresiones en M1 y M2 que modifican la post-condición no comparten ningún término *entonces* $M_2$ es aplicable a $R_1$

**Propiedad 3.a:**
*Si* R es refinable semánticamente de participante por $M_2$
*y* $R_1 = Refsp (R, M_1)$

*y* ($\forall$ *p* $\in$ (Part ($M_1$) $\cap$Part ($M_2$)): ($\forall$ m $\in$ ( $Int_{M1}(p) \cap Int_{M2}(p)$ ):
($\forall$ exp $\in$ ($Pcond_{M2\ p}(m) - Pcond_{R\ p}(m)$): *noComparte*(exp, $Pcond_{M1\ p}(m) - Pcond_{R\ p}(m)$)
$\Rightarrow$ $R_1$ es refinable semánticamente de participante por $M_2$

donde el predicado *noComparte* se define:

*noComparte*:: EXPOCL X $\mathcal{P}$(EXPOCL) $\rightarrow$ BOOL

el significado es:
*noComparte* (e, pCond) = true , si ningún término (*primado* o no) en la expresión e existe en la post-condición pCond.
= false , en caso contrario.

*Demostración*:

Sean $M_1$ , $M_2$ modificadores de refinamiento semántico de participante

Contamos con las hipótesis:

(a) R refinable semánticamente de participante por $M_2$
(b) $R_1$ = Refsp (R, $M_1$ )
( c ) ($\forall$ *p* $\in$ (Part ($M_1$) $\cap$Part ($M_2$)): ($\forall$ m $\in$ ( $Int_{M1}(p) \cap Int_{M2}(p)$ ):
($\forall$ exp $\in$ ($Pcond_{M2\ p}(m) - Pcond_{R\ p}(m)$): *noComparte*(exp, $Pcond_{M1\ p}(m) - Pcond_{R\ p}(m)$)

Debemos probar que:
$\forall$ p $\in$ Part($M_2$) :
(1) p $\in$ Part($R_1$)
(2) $\forall$ m $\in$ $Int_{M2}(p)$: m $\in$ $Int_{R1}$ (p)
(3)$\forall$ m $\in$ $Int_{M2}(p)$: $\forall$ e $\in$ ($Pcond_{M2\ p}(m)$ - $Pcond_{R1\ p}(m)$): *bienFormada*(e, $R_1$) $\wedge$ *noExiste*(e, $Pcond_{R1\ p}(m)$)
(4)$\forall$ *m* $\in$ $Int_{M2}(p)$: Inv($R_1$)$\rightarrow$Inv($R_1$)[$Pcond_{M2\ p}(m)$]

Por (a) tenemos que R es refinable semánticamente de participante por $M_2$ ,por lo tanto:
$\forall$ p $\in$ Part ($M_2$): p $\in$ Part(R):
$\forall$ m $\in$ $Int_{M2}(p)$: $\forall$ e $\in$ ($Pcond_{M2\ p}(m)$ - $Pcond_{R\ p}(m)$): *bienFormada*(e, R) $\wedge$ *noExiste*(e, $Pcond_{R\ p}(m)$)
$\forall$ *m* $\in$ $Int_{M2}(p)$: Inv(R)$\rightarrow$Inv(R)[$Pcond_{M2\ p}(m)$]

Por (b), definición de Resultado de refinamiento semántico de participante, tenemos que:

Part($R_1$) =Part(R)
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :
$\forall$ m $\in$ $Int_{M1}(p)$ : m $\in$ $Int_{R1}(p)$ : $Pcond_{R1\ p}(m) = Pcond_{M1\ p}$ (m)

Por (a) y (b) podemos decir que se cumple (1) y (2), o sea $\forall$ p $\in$ Part($M_2$): p $\in$ Part($R_1$) :
$\forall$ m $\in$ $Int_{M2}(p)$: m $\in$ $Int_{R1}$ (p)

Para probar (3) y (4), por (b) tenemos que
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :

$\forall$ m $\in$ $Int_{M1}(p)$ : m $\in$ $Int_{R1}(p)$ : $Pcond_{R1\ p}(m) = Pcond_{M1\ p}(m)$
$Inv(R_1) = Inv(R)$

podemos decir que, reemplazando $Pcond_{M1\ p}(m)$ con $Pcond_{R1\ p}(m)$ en (c) tenemos
($\forall$ *p* $\in$ (Part ($M_1$) $\cap$Part ($M_2$)): ($\forall$ m $\in$ ( $Int_{M1}(p)$ $\cap$ $Int_{M2}(p)$ ):
($\forall$ exp $\in$ ($Pcond_{M2\ p}(m)$ – $Pcond_{R\ p}(m)$)): *noComparte*(exp, $Pcond_{R1\ p}(m)$ – $Pcond_{R\ p}(m)$)
en particular vale ($\forall$ exp $\in$ ($Pcond_{M2\ p}(m)$ – $Pcond_{R\ p}(m)$)): *noExiste*(exp, $Pcond_{R1\ p}(m)$ – $Pcond_{R\ p}(m)$), como en $Pcond_{R\ p}(m)$ no existen las nuevas expresiones de $Pcond_{M1\ p}(m)$ y $Pcond_{M2\ p}(m)$ y por (a)
$\forall$ m $\in$ $Int_{M2}(p)$: $\forall$ e $\in$ ($Pcond_{M2\ p}(m)$ - $Pcond_{R\ p}(m)$): *bienFormada*(e, R), pasa lo mismo respecto a $R_1$, el punto (3) vale en la intersección.
Además, como $Inv(R_1) = Inv(R)$, reemplazando en (a) tenemos
($\forall$ *p* $\in$ (Part ($M_1$) $\cap$Part ($M_2$)): ($\forall$ m $\in$ ( $Int_{M1}(p)$ $\cap$ $Int_{M2}(p)$ ): $Inv(R_1) \rightarrow Inv(R_1)$ [$Pcond_{M2\ p}(m)$] , el punto (4) vale en la intersección.

Resta probarlo para ($\forall$ *p* $\in$ (Part ($M_2$) - Part ($M_1$)):
como por (a) tenemos que
$\forall$ p $\in$ Part ($M_2$): p $\in$ Part(R):
$\forall$ m $\in$ $Int_{M2}(p)$: $\forall$ e $\in$ ($Pcond_{M2\ p}(m)$ - $Pcond_{R\ p}(m)$): *bienFormada*(e, R) $\wedge$ *noExiste*(e, $Pcond_{R\ p}(m)$)
$\forall$ *m* $\in$ $Int_{M2}(p)$: $Inv(R) \rightarrow Inv(R)[Pcond_{M2\ p}(m)]$
y como ($\forall$ *p* $\in$ (Part ($M_2$) - Part ($M_1$)) estos participantes no fueron modificados por $M_1$
vale que ($\forall$ *p* $\in$ (Part ($M_2$) -Part ($M_1$)): ($\forall$ m $\in$ ( $Int_{M2}(p)$ - $Int_{M1}(p)$ ):
$Pcond_{R\ p}(m) = Pcond_{R1\ p}(m)$ , reemplazando en (a), se cumple (3) y
como $Inv(R_1) = Inv(R)$, reemplazando en (a) se cumple (4)

Queda demostrado que $R_1$ es refinable semánticamente de participante por $M_2$.

*Un Ejemplo de la Propiedad 3.a*

El modificador M1, en la figura 8.4, representa un Refinamiento semántico de *extraer* sobre el participante CAhorro, donde la post-condición de la operación ahora expresa que el saldo debe ser mayor que 0. Por otro lado, el modificador M2 representa otro Refinamiento semántico de participante sobre *extraer* en CAhorro. En este caso la post-condición indica que no deben superarse las 5 extracciones. Como ambas modificaciones no tienen términos en común y preservan el invariante del contrato (saldo>0 en todas las cuentas), su combinación es aplicable y se cumple la propiedad 3.a.

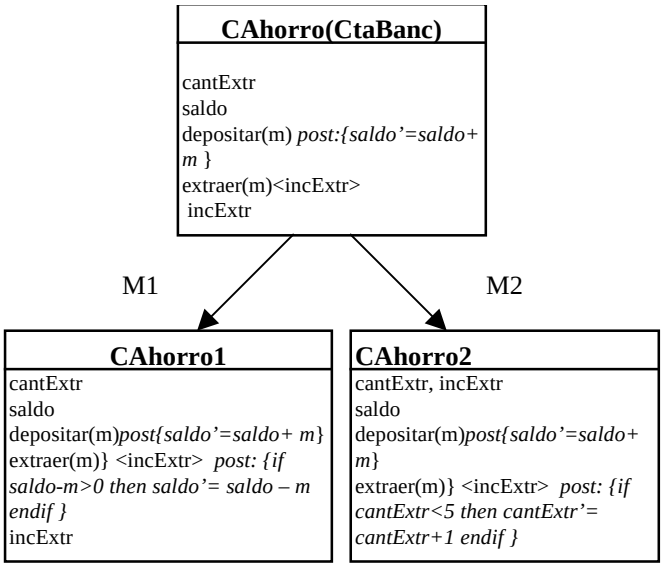


*Fig.8.4 Propiedad 3.a*

**Propiedad 3.b:**

*Si* R es *coarsenable* semánticamente de participante por $M_2$

*y* $R_1 = Coasp(R, M_1)$

*y* $(\forall p \in (Part(M_1) \cap Part(M_2)): (\forall m \in (Int_{M1}(p) \cap Int_{M2}(p)):$

$(\forall exp \in (Pcond_{R\ p}(m) - Pcond_{M2\ p}(m)): noComparte(exp, Pcond_{R\ p}(m) - Pcond_{M1\ p}(m))$

$\Rightarrow R_1$ es *coarsenable* semánticamente de participante por $M_2$

*Demostración*:

Sean $M_1$ , $M_2$ modificadores de coarsening semántico de participante

Contamos con las hipótesis:

Sea (a) R coarsenable semánticamente de participante por $M_2$

(b ) $R_1 = Coasp(R, M_1)$

( c ) $(\forall p \in (Part(M_1) \cap Part(M_2)): (\forall m \in (Int_{M1}(p) \cap Int_{M2}(p)):$

$(\forall exp \in (Pcond_{R\ p}(m) - Pcond_{M2\ p}(m)): noComparte(exp, Pcond_{R\ p}(m) - Pcond_{M1\ p}(m))$

Debemos probar que:

$\forall p \in Part(M_2)$ :

(1) $p \in Part(R_1)$

(2) $\forall m \in Int_{M2}(p): m \in Int_{R1}(p)$

(3) $\forall m \in Int_{M2}(p): \forall e \in Pcond_{M2\ p}(m): e \in Pcond_{R1\ p}(m) \wedge$

$Inv(R_1) \rightarrow Inv(R_1)[Pcond_{M2\ p}(m)]$

Por (a) tenemos que R es coarsenable semánticamente de participante por $M_2$ ,por lo tanto:

$\forall p \in Part(M_2): p \in Part(R):$

$\forall m \in Int_{M2}(p): m \in Int_R(p) \wedge$

$\forall m \in Int_{M2}(p): \forall e \in Pcond_{M2\ p}(m): e \in Pcond_{R\ p}(m) \wedge Inv(R) \rightarrow Inv(R)[Pcond_{M2\ p}(m)]$

Por (b), definición de Resultado de coarsening semántico de participante, tenemos que:

$Part(R_1) = Part(R)$
$\forall\ p \in Part(M_1) : p \in Part(R_1) :$
$\forall\ m \in Int_{M1}(p) : m \in Int_{R1}(p) : Pcond_{R1\ p}(m) = Pcond_{M1\ p}(m)$
$Inv(R_1) = Inv(R)$

Por (a) y (b) podemos decir que se cumple (1) y (2), o sea $\forall\ p \in Part(M_2): p \in Part(R_1)$:
$\forall\ m \in Int_{M2}(p): m \in Int_{R1}(p)$

Para probar (3) por (b) tenemos que
$\forall\ p \in Part(M_1) : p \in Part(R_1) :$
$\forall\ m \in Int_{M1}(p) : m \in Int_{R1}(p) : Pcond_{R1\ p}(m) = Pcond_{M1\ p}(m)$
$Inv(R_1) = Inv(R)$

podemos decir que, reemplazando $Pcond_{M1\ p}(m)$ con $Pcond_{R1\ p}(m)$ en (c) tenemos
$(\forall\ p \in (Part(M_1) \cap Part(M_2)): (\forall\ m \in (Int_{M1}(p) \cap Int_{M2}(p)):$
$(\forall\ exp \in (Pcond_{R\ p}(m) - Pcond_{M2\ p}(m)):$ *noComparte*$(exp, Pcond_{R\ p}(m) - Pcond_{R1\ p}(m))$
en particular
$(\forall\ exp \in (Pcond_{R\ p}(m) - Pcond_{M2\ p}(m)):$ *noExiste*$(exp, Pcond_{R\ p}(m) - Pcond_{R1\ p}(m))$, como las expresiones que permanecen en R por $M_2$ no fueron eliminadas en $R_1$
y $Inv(R_1) = Inv(R)$, el punto (3) queda demostrado en la intersección.

Resta probarlo para $(\forall\ p \in (Part(M_2) - Part(M_1))$:
como por (a) tenemos que
$\forall\ p \in Part(M_2): p \in Part(R):$
$\forall\ m \in Int_{M2}(p): m \in Int_R(p) \wedge$
$\forall\ m \in Int_{M2}(p): \forall\ e \in Pcond_{M2\ p}(m): e \in Pcond_{R\ p}(m) \wedge Inv(R) \rightarrow Inv(R)[Pcond_{M2\ p}(m)]$
y como $(\forall\ p \in (Part(M_2) - Part(M_1))$ estos participantes no fueron modificados por $M_1$ vale que $(\forall\ p \in (Part(M_2) - Part(M_1))$: $(\forall\ m \in (Int_{M2}(p) - Int_{M1}(p)):$
$Pcond_{R\ p}(m) = Pcond_{R1\ p}(m)$ , reemplazando en (a), se cumple (3)

Queda demostrado que $R_1$ es coarsenable semánticamente de participante por $M_2$.

### 8.2.3 Conflicto de Inconsistencia Semántica de Invariante

Analizando el Conflicto de Inconsistencia Semántica de Invariante y los operadores que participan en su ocurrencia, surgen casos en los que este conflicto no ocurre, por lo que pueden enunciarse las siguientes propiedades de aplicabilidad:

**Consistencia en la Composición semántica de contexto**

**Propiedad 4.a:**
*Si* R es refinable semánticamente de contexto por $M_2$
*y* $R_1 = Refsc(R, M_1)$
*entonces* $R_1$ es refinable semánticamente de contexto por $M_2 \Leftrightarrow$ *instancian* $(Inv(M_1), Inv(M_2))$

*Demostración*:

Sean $M_1$ y $M_2$ modificadores de refinamiento semántico de contexto

Sea (a) R refinable semánticamente de contexto por $M_2$
(b ) $R_1$ = Refsc (R,$M_1$)

Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es refinable semánticamente de contexto por $M_2$ $\Rightarrow$ *instancian* (Inv($M_1$), Inv ($M_2$))

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (a), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R) , Inv ($M_2$))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de refinamiento semántico de contexto, tenemos que:

$\forall$ p $\in$ Part(R) : p $\in$ Part($R_1$) $\wedge$ $Int_{R1}$ (p) = $Int_R$ (p)
$\forall$ p $\in$ Part($R_1$): $\forall$ m $\in$ $Int_{R1}$ (p): $Pcond_{R1\ p}$ (m) = $Pcond_{R\ p}$ (m)
Inv($R_1$) = Inv ($M_1$)

Por (c ), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), $R_1$)
*instancian* (Inv($R_1$) , Inv ($M_2$))
$\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R1\ p}$ (m)]

Ahora, como Inv($R_1$) = Inv ($M_1$) por (b), queda demostrado que vale
*instancian* (Inv($M_1$), Inv ($M_2$))

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) *instancian* (Inv($M_1$) , Inv ($M_2$)) $\Rightarrow$ $R_1$ es refinable semánticamente de contexto por $M_2$

O sea, debemos probar que:

(1*) bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($R_1$) , Inv ($M_2$))
(3) $\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R1\ p}$ (m)]

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:
*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R) , Inv ($M_2$))

$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de refinamiento semántico de contexto, tenemos que:

Part($R_1$) = Part ( R)
Inv($R_1$) = Inv ($M_1$)

Ahora, por (a) y como los participantes y operaciones de R y $R_1$ coinciden por (b), el punto (1) que demostrado.

Por (d ) vale *instancian* (Inv($M_1$) , Inv ($M_2$)), y como por (b) Inv($R_1$) = Inv ($M_1$) queda demostrado el punto (2).

Por (a) $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R\ p}$ (m)]
y como por (b)
$\forall$ p $\in$ Part($R_1$): $\forall$ m $\in$ $Int_{R1}$ (p): $Pcond_{R1\ p}$ (m) = $Pcond_{R\ p}$ (m), reemplazando en (a) queda demostrado el punto (3).
Por lo tanto queda demostrado que $R_1$ es refinable semánticamente de contexto por $M_2$ .

*Un Ejemplo de la Propiedad 4.a*

En la figura 8.5, M1 representa un Refinamiento semántico de contexto, donde el invariante ahora expresa que el saldo mínimo para todas las cuentas debe mantenerse mayor que 50. Por otro lado M2 representa otro Refinamiento semántico de contexto. En este caso el invariante indica que el saldo de las cuentas debe superar los 100 pesos; como ambas condiciones instancian un pattern de refinamiento, se cumple la propiedad 4.a.

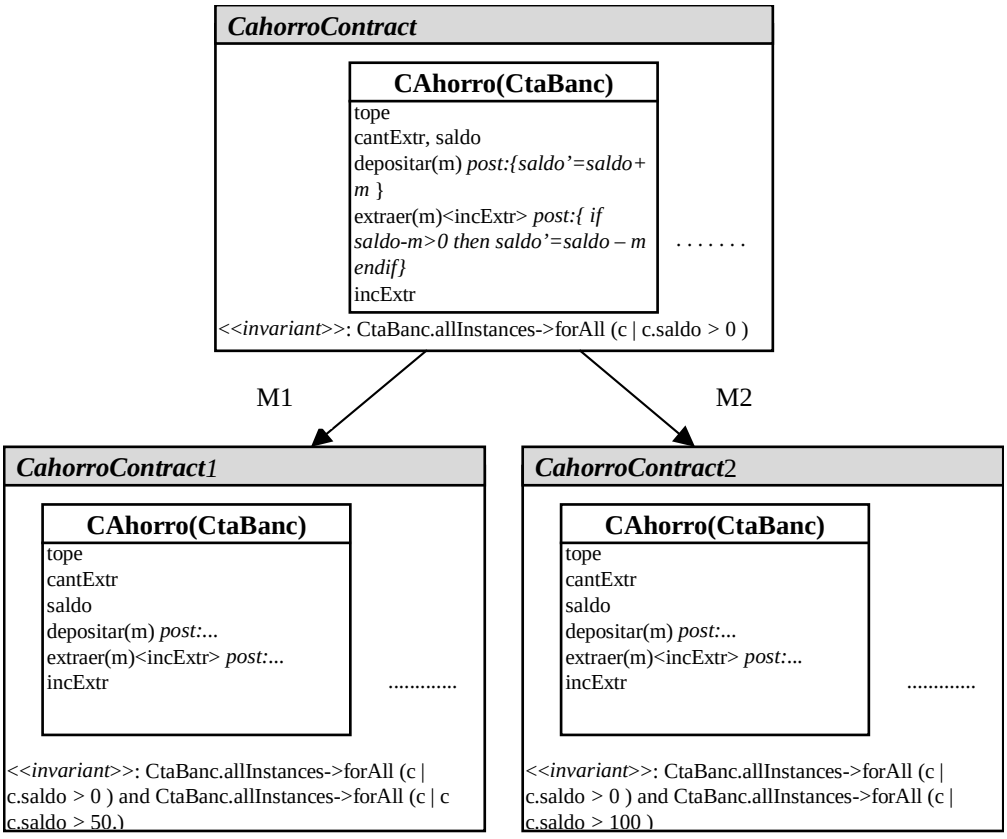


*Fig.8.5 Propiedad 4.a*

**Propiedad 4.b:**

*Si* R es *coarsenable* semánticamente de contexto por $M_2$
*y* $R_1 = Coasc (R, M_1)$
*entonces* $R_1$ es *coarsenable* semánticamente de contexto por $M_2 \Leftrightarrow$
*instancian* (Inv $(M_2)$, Inv$(M_1)$)

*Demostración*:

Sean $M_1$ y $M_2$ modificadores de *coarsening* semántico de contexto

Sea (a) R *coarsenable* semánticamente de contexto por $M_2$
(b) $R_1 = Coasc (R, M_1)$

Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es *coarsenable* semánticamente de contexto por $M_2 \Rightarrow$
*instancian* (Inv $(M_2)$, Inv$(M_1)$)

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (a), definición de aplicabilidad de *coarsening* semántico de contexto, tenemos que:

*bienFormada* (Inv $(M_2)$, R)
*instancian* (Inv $(M_2)$, Inv(R))
$\forall p \in Part(R) : \forall m \in Int_R(p) : Inv(M_2) \rightarrow Inv(M_2)[Pcond_{R\,p}(m)]$

Por (b), definición de Resultado de *coarsening* semántico de contexto, tenemos que:

$\forall p \in Part(R) : p \in Part(R_1) \wedge Int_{R1}(p) = Int_R(p)$
$\forall p \in Part(R_1): \forall m \in Int_{R1}(p): Pcond_{R1\,p}(m) = Pcond_{R\,p}(m)$
$Inv(R_1) = Inv(M_1)$

Por (c ), definición de aplicabilidad de *coarsening* semántico de contexto, tenemos que:

*bienFormada* (Inv$(M_2)$, $R_1$)
*instancian* (Inv$(M_2)$ , Inv $(R_1)$)
$\forall p \in Part(R_1) : \forall m \in Int_{R1}(p) : Inv(M_2) \rightarrow Inv(M_2)[Pcond_{R1\,p}(m)]$

Ahora, como $Inv(R_1) = Inv(M_1)$ por (b), queda demostrado que vale
*instancian* (Inv$(M_1)$, Inv $(M_2)$)

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) *instancian* (Inv$(M_2)$ , Inv $(M_1)$) $\Rightarrow$ $R_1$ es coarsenable semánticamente de contexto por $M_2$

O sea, debemos probar que:

(1*) bienFormada* (Inv($M_2$), $R_1$)
(2) *instancian* (Inv($M_2$) , Inv ($R_1$))
(3) $\forall$ p $\in$ Part($R_1$) : $\forall$ m $\in$ $Int_{R1}$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R1\ p}$ (m)]

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a), definición de aplicabilidad de *coarsening* semántico de contexto, tenemos que:

*bienFormada* (Inv($M_2$), R)
*instancian* (Inv ($M_2$) , Inv (R))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de *coarsening* semántico de contexto, tenemos que:

Part($R_1$) = Part ( R)

Inv($R_1$) = Inv ($M_1$)

Ahora, por (a) y como los participantes y operaciones de R y $R_1$ coinciden por (b), el punto (1) que demostrado.

Por (d ) vale *instancian* (Inv($M_2$) , Inv ($M_1$)), y como por (b) Inv($R_1$) = Inv ($M_1$) queda demostrado el punto (2).

Por (a) $\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_R$(p) : Inv( $M_2$)$\rightarrow$ Inv($M_2$)[$Pcond_{R\ p}$ (m)]
y como por (b)
$\forall$ p $\in$ Part($R_1$): $\forall$ m $\in$ $Int_{R1}$ (p): $Pcond_{R1\ p}$ (m) = $Pcond_{R\ p}$ (m), reemplazando en (a) queda demostrado el punto (3).
Por lo tanto queda demostrado que $R_1$ es coarsenable semánticamente de contexto por $M_2$ .

### 8.2.4 Conflicto de Inconsistencia Semántica

Analizando el Conflicto de Inconsistencia Semántica y los operadores que participan en su ocurrencia, surgen casos en los que este conflicto no ocurre, por lo que puede enunciarse la siguiente propiedad de aplicabilidad:

**Composición de operador semántico de participante y operador semántico de contexto**

*Si* $M_1$ es un modificador de operador semántico de participante y $M_2$ es un modificador de operador semántico de contexto y ambos son aplicables a R *y* $R_1$ es el contrato resultante de aplicar $M_1$ a R *entonces*
$M_2$ es aplicable a $R_1$ sii cada post-condición modificada por $M_1$ es consistente con el invariante en $M_2$ (es decir, la operación modificada por $M_1$ preserva el invariante en $M_2$).
La propiedad probada a continuación es también demostrable en forma similar para redefinición y coarsening semántico de participante combinados con operadores semánticos de contexto.

**Propiedad 5:**
Sea $M_2$ un modificador de refinamiento, redefinición o coarsening semántico de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1 = Refsp\ (R, M_1)$ refinamiento semántico de participante de R por $M_1$
*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow$
$\forall\ p \in Part\ (M_1): \forall\ m\ \in Int_{M1}(p): Inv(M_2) \rightarrow Inv(M_2)[Pcond_{M1\ p}\ (m)]$

*Demostración*:

Sea $M_2$ modificador de refinamiento semántico de contexto

Sea (a) R refinable semánticamente de contexto por $M_2$
(b) $R_1 = Refsp\ (R, M_1)$

Para demostrar la implicación hacia la derecha ($\Rightarrow$) debemos probar que:

( c ) $R_1$ es refinable semánticamente de contexto por $M_2 \Rightarrow$

$\forall\ p \in Part\ (M_1): \forall\ m\ \in Int_{M1}(p): Inv(M_2) \rightarrow Inv(M_2)[Pcond_{M1\ p}\ (m)]$

Contamos con las hipótesis: (a), (b) y el antecedente de ( c )

Por (c), definición de aplicabilidad de refinamiento semántico de contexto, tenemos que:

$p \in Part\ (R_1) : \forall\ m\ \in Int_{R1}(p) : Inv(M_2) \rightarrow Inv(M_2)[Pcond_{R1\ p}\ (m)]$

Por (b), definición de Resultado de refinamiento semántico de participante, tenemos que:

$\forall\ p \in Part\ (M_1) : p \in Part\ (R_1) :$
$\forall\ m\ \in Int_{M1}(p) : m\ \in Int_{R1}(p) : Pcond_{R1\ p}(m) = Pcond_{M1\ p}\ (m)$

entonces en (c ) , podemos reemplazar $Pcond_{R1\ p}(m)$ por $Pcond_{M1\ p}\ (m)$ en los participantes que están en Part $(M_1)$ por lo tanto queda demostrado que

$\forall\ p \in Part\ (M_1): \forall\ m\ \in Int_{M1}(p): Inv(M_2) \rightarrow Inv(M_2)[Pcond_{M1\ p}\ (m)]$

Para demostrar la implicación hacia la izquierda ($\Leftarrow$) debemos probar que:

( d ) $\forall\ p \in Part\ (M_1): \forall\ m\ \in Int_{M1}(p): Inv(M_2) \rightarrow Inv(M_2)[Pcond_{M1\ p}\ (m)] \Rightarrow R_1$ es refinable semánticamente de contexto por $M_2$

O sea, debemos probar que:
(1*) bienFormada* $(Inv(M_2), R_1)$
(2*) instancian* $(Inv(R_1), Inv\ (M_2))$
(3)$\forall\ p\ \in Part(R_1) : \forall\ m\ \in Int_{R1}(p) : Inv(M_2) \rightarrow Inv(M_2)[Pcond_{R1\ p}\ (m)]$

Contamos con las hipótesis: (a), (b) y el antecedente de (d)

Por (a) tenemos que R es refinable semánticamente de contexto por $M_2$ ,por lo tanto:
*bienFormada* (Inv($M_2$), R)
*instancian* (Inv(R) , Inv ($M_2$))
$\forall$ p $\in$ Part(R) : $\forall$ m $\in$ $Int_{R'}$(p) : Inv($M_2$)$\rightarrow$Inv($M_2$)[$Pcond_{R\ p}$ (m)]

Por (b), definición de Resultado de refinamiento semántico de participante, tenemos que:

Part($R_1$) =Part(R)
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :
$\forall$ m $\in$ $Int_{M1}$(p) : m $\in$ $Int_{R1}$(p) : $Pcond_{R1\ p}$(m) = $Pcond_{M1\ p}$ (m)
Inv($R_1$) = Inv (R)
Por (a) y (b) podemos decir que se cumple (1) y (2)

Para probar (3) por (b) tenemos que
$\forall$ *p* $\in$ Part ($M_1$) : *p* $\in$ Part ($R_1$) :
$\forall$ m $\in$ $Int_{M1}$(p) : m $\in$ $Int_{R1}$(p) : $Pcond_{R1\ p}$(m) = $Pcond_{M1\ p}$ (m)

podemos decir que, reemplazando en (d) tenemos
$\forall$ *p* $\in$ Part ($R_1$): $\forall$ m $\in$ $Int_{R1}$(p): Inv($M_2$)$\rightarrow$Inv($M_2$)[$Pcond_{R1\ p}$ (m)]

Queda demostrado que $R_1$ es refinable semánticamente de contexto por $M_2$.

*Un Ejemplo de la Propiedad 5*

En la figura 8.6, M1 representa un Refinamiento semántico de participante donde la post-condición de la operación extraer se refina limitando el saldo de la cuenta. Por otro lado M2 representa un Refinamiento semántico de contexto donde el invariante ahora expresa que el saldo mínimo para todas las cuentas debe mantenerse mayor que -10. Como las modificaciones mantienen consistencia entre los elementos semánticos, se cumple la propiedad 5.

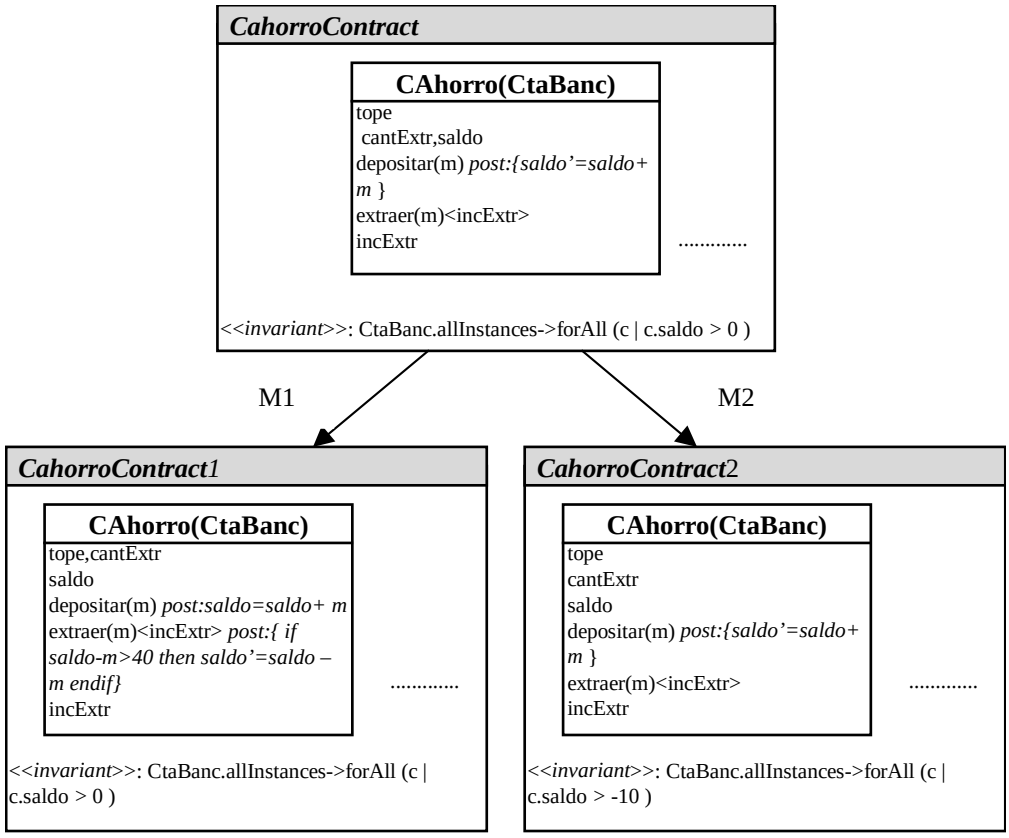


*Fig.8.6 Propiedad 5*

# Capítulo 9: Un Caso de Estudio

Con el fin de mostrar cómo los contratos de reuso son útiles en el proceso de desarrollo de un diseño reusable, en este capítulo presentamos, utilizando e instanciando al modelo matemático definido en el Capítulo 7, un Caso de Estudio donde se plantean algunas situaciones conflictivas en la evolución del diseño y otras donde se cumplen las propiedades de aplicabilidad asociadas a conflictos, que fueron presentadas en el capítulo anterior. El control de situaciones puede ser implementado en términos del modelo matemático.

## *9.1 El caso: Asignación de Profesores a Cursos. Un Primer Diseño*

El caso que presentamos corresponde sólo al proceso de asignación de profesores a cursos dentro de un Instituto de Enseñanza. El instituto tiene la responsabilidad de realizar la asignación y para ello, dada una especialidad, por cada profesor de esa especialidad, selecciona posibles cursos y posibles horarios de ocupación de aulas. Los posibles cursos son solicitados al profesor que se relaciona con los cursos. La Figura 9.1 presenta un contrato de reuso para este problema. El área de los cursos posibles para un profesor debe coincidir con su especialidad. Esto es expresado como post-condición de la operación posiCursos del Participante Profesor. Para poder validar esta condición, posiCursos invoca a la operación área de cada curso con el que se relaciona por medio de la relación de conocimiento *cursos*. Una vez obtenida esta información, el Instituto realiza una asignación conveniente y la registra con los datos del profesor, el curso que dictará, el aula y el horario. Para todas las asignaciones que se realicen, si coincide el profesor que las dicta, no debe superponerse el horario; esta restricción aparece como invariante del contrato.

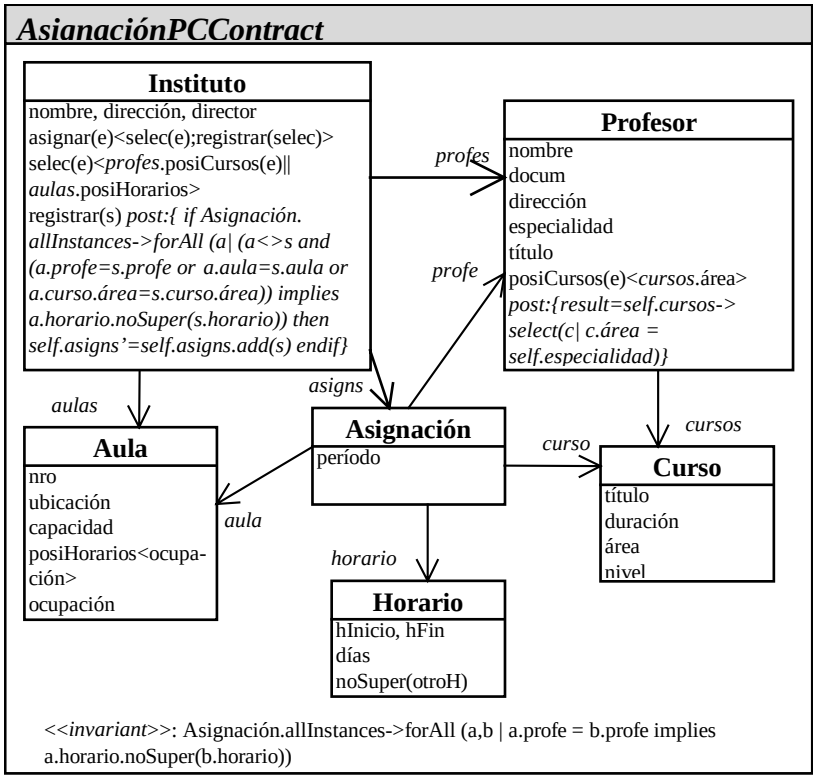


*Fig 9.1. El contrato de Reuso AsignaciónPC*

### *9.2 Evolución del diseño: Un nuevo Contrato de Reuso derivado*

Supongamos que, a fin de modelar profesores que tengan un comportamiento especial en el contrato descripto en la sección anterior, por ejemplo poder dictar solamente cursos de grado, se debe agregar una invocación a la operacón *posiCursos* del participante Profesor para conocer a que área pertenece cada curso. Se da así origen al participante ProfeGrado.
Dado que evoluciona un solo participante, se puede mostrar sólo este elemento del contrato de reuso AsignaciónPC, ligado por la relación de herencia con su derivado, como muestra la figura 9.2. El operador aplicado es un Refinamiento sintáctico del Participante Profesor. La aplicabilidad de este operador sobre el contrato base es verificada en el apartado 9.2.1 de esta sección.

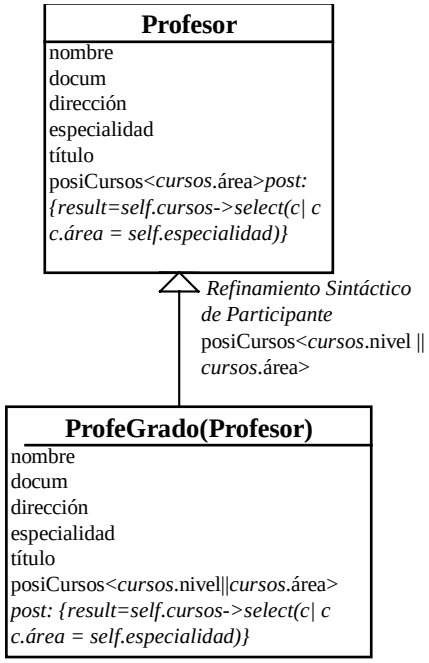


*Fig 9.2. Evolución Sintáctica de Profesor*

Como consecuencia de este cambio resulta necesario expresar que los cursos posibles para este profesor derivado (ProfeGrado), tengan nivel de grado. Por lo tanto, se debe aplicar una Redefinición Semántica del Participante Profesor, sobre la operación posiCursos. Se obtiene así el participante derivado ProfesorGrado, como se puede ver en la Figura 9.3. La aplicabilidad de este cambio es también verificable.

Otra evolución deseable es poder expresar que al menos a un profesor se le asignará un curso. Una forma de modelar esto es mediante la post-condición de la operación *asignar* en el Participante Instituto, que aparece nula. Se aplica aquí un Refinamiento Semántico de Participante, como muestra la Figura 9.4.

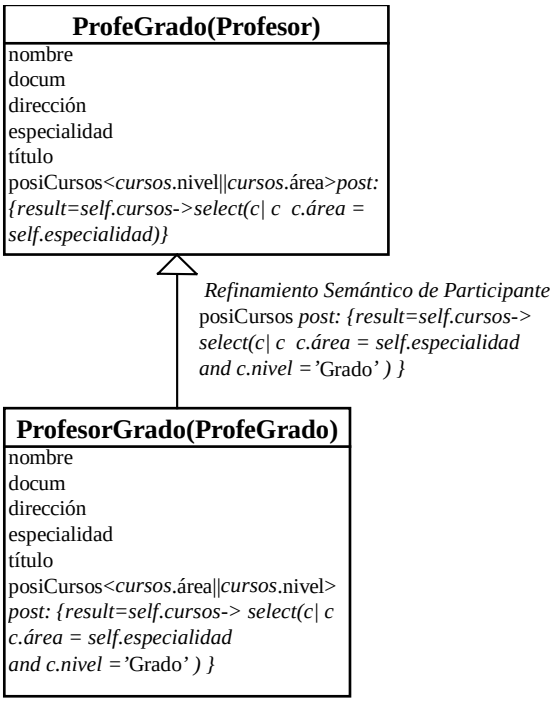


*Fig 9.3. Evolución Semántica de Profesor*

Por último, supongamos que además de mantener la restricción de que para todas las asignaciones que se realicen, si coincide el profesor que las dicta, no debe superponerse el horario, queramos también que los horarios de las asignaciones en la misma aula no se superpongan. Este cambio se representa mediante un Refinamiento Semántico de Contexto, que es aplicable pues la operación *registrar* de Instituto, que realiza cada asignación, preserva al nuevo invariante.
La figura 9.5 muestra el contrato de reuso AsignaciónPC2, resultante de aplicar todos los cambios mencionados en esta sección, cada uno de los cuales es aplicable sobre el contrato original.
El siguiente apartado verifica la aplicabilidad de alguno de los operadores utilizados, instanciando el modelo matemático desarrollado en el Capítulo 7. En forma similar pueden verificarse los demás.

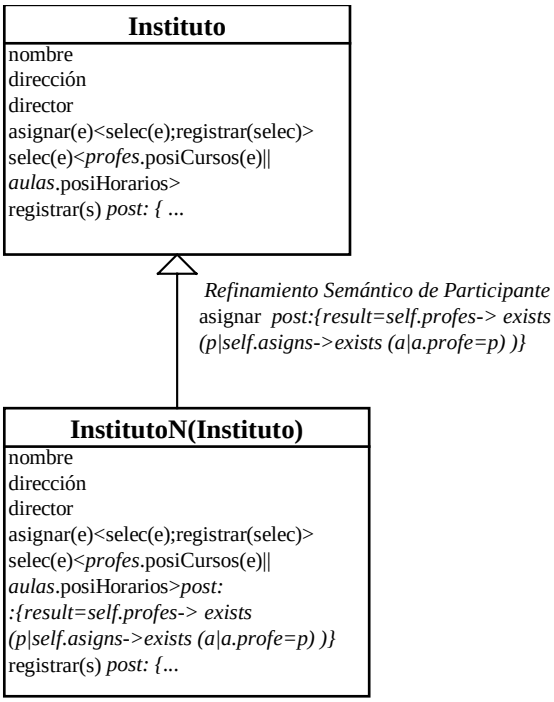


*Fig 9.4. Evolución Semántica de Instituto*

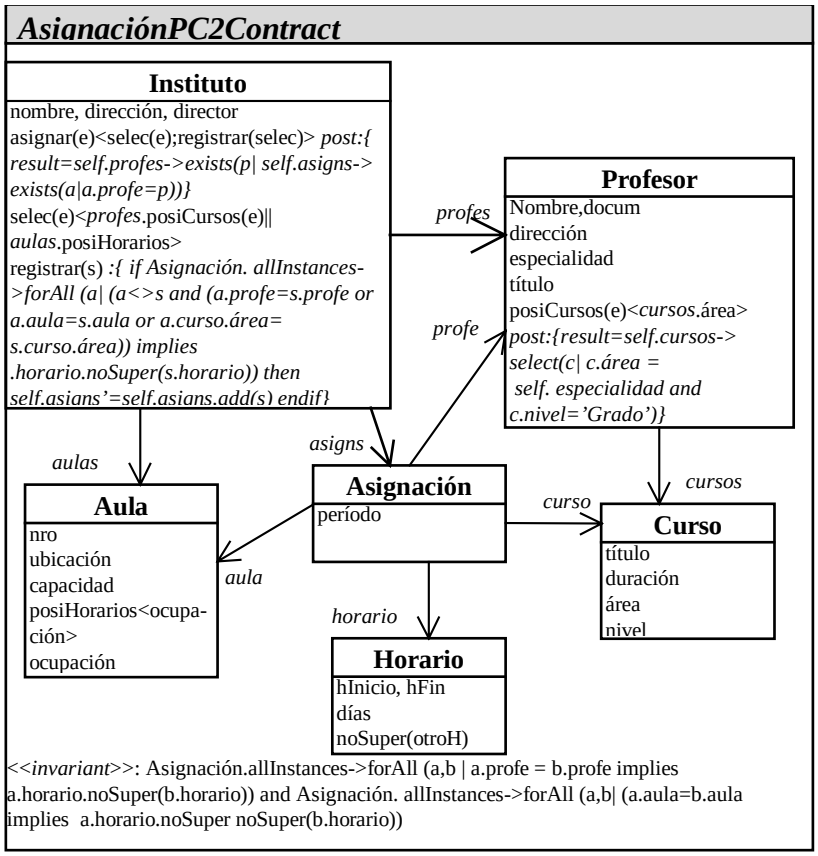


*Fig 9.5. El contrato derivado AsignaciónPC2Contract*

### 9.2.1 Instanciación del Modelo Matemático

Instanciando el Modelo Matemático definido en el Capítulo 7 con este caso de estudio, podemos mostrar que los operadores de reuso arriba descriptos son aplicables al contrato de reuso original.

Definimos, en primer lugar al contrato de reuso base AsignaciónPC y el valor de las funciones observadoras que se pueden aplicar sobre él.

El contrato AsignaciónPC perteneciente al dominio CONTRATO se define en el modelo de la siguiente forma:

*AsignaciónPC* = <**{**Instituto, Profesor, Asignación, Aula, Curso, Horario**}**, **{**<Instituto, Profesor, *profes*>, <Instituto, Aula, *aulas*>, <Instituto, Asignación, *asigns*>,<Profesor, Curso, *cursos*>, <Asignación, Horario, *horario*>, <Asignación, Profesor, *profe*>, <Asignación, Aula, *aula*>, <Asignación, Curso, *curso*>**}**, **{**<Instituto, asignar(), <selec(); registrar()>, {}>, <Instituto, nombre, σ, {}>, <Instituto, dirección, σ, {} >, <Instituto, director, σ, {} >, <Instituto, selec(), <*profes*.posiCursos || *aulas*.posiHorarios >, {} >, <Instituto, registrar(), σ, *{ if Asignación.allInstances->forAll (a| (a<>s and (a.profe=s.profe or a.aula=s.aula or a.curso.área=s.curso.área)) implies a.horario.noSuper(s.horario)) then self.asigns'=self.asigns.add(s) endif}* >, <Profesor, posiCursos, <*cursos*.área>, {*result=self.cursos-> select(c| c.área=self.especialidad* }>, <Profesor, nombre, σ, {} >, <Profesor, dirección, σ, {} >, <Profesor, docum, σ, {} >, <Profesor, especialidad, σ, {} >, <Profesor, título, σ, {} >, <Aula, nro, σ, {} >, <Aula, ubicación, σ, {} >, <Aula, capacidad, σ, {}>, <Aula, posiHorarios, σ, {} >, <Curso, título, σ, {}>, <Curso, duración, σ, {}>, <Curso, área, σ, {}>, <Curso, nivel, σ, {}>,<Asignación, período, σ, {}>, <Horario, hInicio, σ, {}>, <Horario, hFin, σ, {}>, <Horario, días, σ, {}>, <Horario, noSuper(), σ, {}>**}**, *Asignación.allInstances -> forAll (a,b | a.profe = b.profe implies a.horario.noSuper(b.horario))*>

Por lo tanto, algunas de las funciones observadoras tienen los siguientes valores:

Part(*AsignaciónPC*) = {Instituto, Profesor, Asignación, Aula, Curso, Horario}
conjunto de todos los nombres de participantes del contrato de reuso *AsignaciónPC*

Inv(*AsignaciónPC*) = *Asignación.allInstances -> forAll (a,b | a.profe = b.profe implies a.horario.noSuper(b.horario))*
invariante del contrato de reuso *AsignaciónPC* .

$Con_{AsignaciónPC}$ (Instituto) = { *profes*.Profesor, *aulas*.Aula, *asigns*.Asignación }
conjunto de todas las relaciones de conocimiento del participante *Instituto* en el contrato *AsignaciónPC*.

$Int_{AsignaciónPC}$ (Instituto) ={ asignar(), nombre, dirección, director, selec(), registrar() }
conjunto de nombres de operaciones del participante *Instituto* en *AsignaciónPC*

$Espec_{AsignaciónPC\ Instituto}$ (asignar()) = <selec(); registrar()>
secuencia de invocaciones de operación, de la operación *asignar* de *Instituto* en *AsignaciónPC*. (cláusula de especialización de *asignar* ).

$Pcond_{AsignaciónPC\ Instituto}$ (registrar()) = *{ if Asignación.allInstances->forAll (a| (a<>s and (a.profe=s.profe or a.aula=s.aula or a.curso.área=s.curso.área)) implies a.horario.noSuper(s.horario)) then self.asigns'=self.asigns.add(s) endif}*
post-condición de la operación registrar() del participante *Instituto* en *AsignaciónPC*.

**Aplicabilidad de los operadores de reuso**

Utilizando el valor de las funciones observadoras aplicadas al contrato, y las definiciones de los operadores de reuso expresadas en el modelo matemático (Capítulo 7), podemos mostrar la aplicabilidad de los operadores empleados en la evolución del contrato.

***Refinamiento Sintáctico del participante Profesor*** (fig. 9.2):

Por *Definición de Aplicabilidad* del Refinamiento sintáctico de participante: *Refp*

Debemos mostrar que:

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.

El operador *Refp* (refinamiento sintáctico de participante) ∈ OPERADOR es aplicable a R por M ⇔

∀ *p* ∈ Part (M):
(1) *p* ∈ Part(R)
(2) ∀ m ∈ $Int_M$(p): m ∈ $Int_R$ (p) ∧
(∀ a.op ∈ ($Espec_{M\ p}$ (m) - $Espec_{R\ p}$ (m)) : (∃ q ∈ Part(R): a.q ∈ $Con_R$ (p) ∧ op ∈ $Int_R$(q)) ∧ *refina (*$Espec_{M\ p}$ (m), $Espec_{R\ p}$ (m), a.op) )

Instanciando el modelo, el modificador es:

M = <{ Profesor }, {}, {<Profesor, posiCursos, <*cursos*.nivel || *cursos*.área>, {} >},*true* >

Para mostrar (1)
∀ *p* ∈ Part (M): *p* ∈ Part(*AsignaciónPC*) , reemplazando
∀ *p* ∈ {Profesor}: *p* ∈ Part(*AsignaciónPC*) , es decir
∀ *p* ∈ {Profesor}: *p* ∈ {Instituto, Profesor, Asignación, Aula, Curso, Horario} , es decir
Profesor ∈ {Instituto, Profesor, Asignación, Aula, Curso, Horario}, lo cual es verdadero y (1) queda demostrado.
Para mostrar (2)
∀ *p* ∈ {Profesor}: ∀ m ∈ $Int_M$ (Profesor) ∧

(∀a.op ∈ ($Espec_{M\ Profesor}$ (m) – $Espec_{AsignacionPC\ Profesor}$ (m)): (∃ q ∈ Part(AsignaciónPC): a.q ∈ $Con_{AsignaciónPC}$ (Profesor) ∧ op ∈ $Int_{AsignaciónPC}$ (q))) ∧ *refina (*$Espec_{M\ Profesor}$ (m), $Espec_{AsignaciónPC\ Profesor}$ (m), a.op) )

reemplazando
∀ *p* ∈ {Profesor}: ∀ m ∈ {posiCursos} ∧
(∀ a.op ∈ {*cursos*.nivel} : (∃ q ∈ {Instituto, Profesor, Asignación, Aula, Curso, Horario}: *cursos*.Curso ∈ {*cursos*.Curso}∧ op ∈ {título, duración, área, nivel})) ∧ *refina (*<*cursos*.nivel || *cursos*.área>, < *cursos*.área>, *cursos*.nivel) )

reemplazando
∀ *p* ∈ {Profesor}: ∀ m ∈ {posiCursos} ∧
(∀ a.op ∈ {*cursos*.nivel} : (∃ Curso: *cursos*.Curso ∈ {*cursos*.Curso}∧ op ∈ {título, duración, área, nivel})) ∧ *refina (*<*cursos*.nivel || *cursos*.área>, < *cursos*.área>, *cursos*.nivel) )

como *refina (*<*cursos*.nivel || *cursos*.área>, < *cursos*.área>, *cursos*.nivel) es verdadero por el quinto caso de definición del predicado, dado en el modelo matemático, el punto (2) queda demostrado. Por lo tanto, el operador es aplicable.

***Redefinición semántica del participante Profesor*** (fig. 9.3):

Por *Definición de Aplicabilidad* de Redefinición semántica de participante: *Red*

Debemos mostrar que:

Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.
El operador *Red* ( redefinición semántica de participante) ∈ OPERADOR es aplicable a R por M ⇔
∀ p ∈ Part (M):
(1) p ∈ Part(R)
(2) *m* ∈ $Int_M$(p): *m* ∈ $Int_M$(p)
(3)∀ *m* ∈ $Int_M$(p): ∀ e ∈ $Pcond_{M\ p}$(m): *bienFormada*(e, R)
(4)∀ *m* ∈ $Int_M$(p): Inv(R)→Inv(R)[$Pcond_{M\ p}$(m)]

Instanciando el modelo, el modificador es:

M = <{Profesor}, {}, {<Profesor, posiCursos, {*result=self.cursos-> select (c| c.área=self.especialidad and c.nivel =* ‘ Grado’)} >}, *true*>

Para mostrar (1)

∀ *p* ∈ Part (M): *p* ∈ Part(*AsignaciónPC*) , reemplazando
∀ *p* ∈ {Profesor}: *p* ∈ Part(*AsignaciónPC*) , es decir
∀ *p* ∈ {Profesor}: *p* ∈ {Instituto, Profesor, Asignación, Aula, Curso, Horario} , es decir
Profesor ∈ {Instituto, Profesor, Asignación, Aula, Curso, Horario}, lo cual es verdadero y (1) queda demostrado.

Para mostrar (2)

$\forall\ p \in$ {Profesor}: $\forall$ m $\in \text{Int}_M$ (Profesor): m $\in \text{Int}_{\text{AsignacionPC}}$ (Profesor) reemplazando
$\forall\ p \in$ {Profesor}: $\forall$ m $\in$ {posiCursos}: m $\in$ { posiCursos, nombre, dirección, docum, especialidad, título} es decir,
para el participante Profesor, la operación posiCursos $\in$ { posiCursos, nombre, dirección, docum, especialidad, título} lo cual es verdadero

Para mostrar (3)

$\forall\ p \in$ {Profesor}: $\forall$ m $\in$ {posiCursos}: $\forall$ e $\in \text{Pcond}_{M\,p}$(m):
*bienFormada* (e, AsignaciónPC)

Es decir, para el participante Profesor, para la operación posiCursos:
*bienFormada* ( *result=self.cursos-> select (c| c.área=self.especialidad and c.nivel* = ‘ Grado’), AsignaciónPC)

la expresión OCL está bien formada ya que puede derivarse de la gramática del Lenguaje (ver [RMH97]) e involucra elementos de AsignaciónPC. Por lo tanto el punto (3) se cumple.
Para mostrar (4),
$\forall\ p \in$ {Profesor}: $\forall$ m $\in$ {posiCursos}:
Inv(AsignaciónPC)$\rightarrow$Inv(AsignaciónPC )[$\text{Pcond}_{M\,p}$(m)], o sea
Inv(AsignaciónPC)$\rightarrow$Inv(AsignaciónPC )[ *result=self.cursos-> select (c| c.área=self.especialidad and c.nivel* = ‘ Grado’]
como
Inv(AsignaciónPC)=*Asignación.allInstances->forAll(a,b| a.profe = b.profe implies a.horario.noSuper(b.horario))*
y dado que la post-condición no cambia el estado de términos que aparezcan en el invariante, se cumple la implicación.
Por lo tanto, el operador es aplicable.

Para el resto de los operadores y para la composición de ellos, se puede mostrar la aplicabilidad sobre el contrato de manera similar, tomando las definiciones del modelo matemático.

### *9.3 Nuevos Cambios en el Contrato Base*

El contrato de reuso base de la Sección 9.1, AsignaciónPC, puede ser objeto de nuevos cambios. Todos los cambios que se presentan a continuación se pueden expresar por operadores de reuso *aplicables* al contrato original, basta instanciar el modelo matemático para probarlo. También puede mostrarse que la composición de estas modificaciones es aplicable a AsignaciónPC; con lo que puede obtenerse un nuevo contrato derivado. Dado que AsignaciónPC ya tenía un contrato derivado: AsignaciónPC2, para verificar si es posible la integración de este derivado con la nueva evolución, debemos ver cómo actúan cada uno de los nuevos cambios combinados con cada uno de los cambios que llevaron a la formación de AsignaciónPC2. Esto significa aplicar el análisis de conflictos expresados en

el Capítulo 6 y controlar el cumplimiento de las propiedades de aplicabilidad presentadas en el Capítulo 8.

***Una nueva Redefinición semántica de Instituto***

Supongamos que queremos que el sistema original asigne a *todos* los profesores un curso. Esto se expresa mediante la post-condición de la operación *asignar* del participante Instituto. El operador a aplicar es una Redefinición Semántica del Participante Instituto. Las consecuencias de este cambio respecto a las modificaciones ya sufridas por el contrato original, son investigadas en la Tabla 9.1. En este caso, la tabla muestra un conflicto dado que la operación en cuestión ya había sido refinada semánticamente. Este conflicto no tiene asociadas propiedades de aplicabilidad.

Otro posible conflicto que se plantea es de Inconsistencia Semántica, sin embargo puede verificarse el cumplimiento de la propiedad 5, asociada a dicho conflicto. La nueva post-condición, se mantiene consistente con el nuevo invariante, lo mostramos instanciando la propiedad:

| **AsignaciónPC (Original)** | **Nueva Versión**:<br>Redefinición semántica de Participante Instituto , operación *asignar* con *result=self.profes-> forAll (p\| self.asigns->exists(a\| a.profe=p))* |
|---|---|
| Refinamiento sintáctico de Participante Profesor , operación *posiCursos* con <*cursos*.nivel \|\| *cursos*.área> | *No hay conflicto* |
| Redefinición semántica de Participante Profesor , op. *posiCursos* con *result= self.cursos-> select(c\| c.área=self.especialidad and c.nivel =* ‘ Grado’) | *No hay conflicto* |
| Refinamiento semántico de Participante Instituto , operación *asignar* con *result=self.profes-> exists (p\| self.asigns->exists(a\| a.profe=p))* | Conflicto de Incoherencia Semántica de Operación |
| Refinamiento semántico de Contexto con *Asignación.allInstances->forAll (a,b \| a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))* | Posible conflicto de Inconsistencia Semántica (verificar Propiedad 5) |

*Tabla 9.1 Un nuevo Refinamiento semántico de Instituto*

*Propiedad 5:*
Sea $M_2$ un modificador de refinamiento, redefinición o coarsening semántico de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1$ = Refsp (R, $M_1$) refinamiento semántico de participante de R por $M_1$
*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow$
$\forall\ p \in \text{Part}(M_1)$: $\forall\ m \in \text{Int}_{M1}(p)$: $\text{Inv}(M_2) \rightarrow \text{Inv}(M_2)[\text{Pcond}_{M1\ p}(m)]$

$M_1$= <{Instituto}, {}, {<Instituto, asignar, σ, {*result=self.profes-> forAll (p| self.asigns->exists(a| a.profe=p))*} >},*true* >

$M_2$= <{}, {}, {}, *Asignación.allInstances->forAll (a,b | a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))* >

Debemos mostrar:
$\forall\ p\ \in$ {Instituto}: $\forall$ m $\in$ {asignar}:
Inv($M_2$)$\rightarrow$Inv ($M_2$)[$Pcond_{M1\ p}$(m)], o sea
Inv($M_2$)$\rightarrow$Inv($M_2$)[ *result=self.profes-> forAll (p| self.asigns->exists(a| a.profe=p))*]
como
Inv($M_2$)= *Asignación.allInstances->forAll (a,b | a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))* y dado que la post-condición no cambia el estado de términos que aparezcan en el invariante, se cumple la implicación y la propiedad.

| **AsignaciónPC (Original)** | **Nueva Versión**:<br>Cancelación Sintáctica de Participante Profesor , operación *posiCursos* |
|---|---|
| Refinamiento sintáctico de Participante Profesor , operación *posiCursos* con <*cursos*.nivel \|\| *cursos*.área> | *Conflicto de Referencia de Operación Colgada* |
| Redefinición semántica de Participante Profesor , op. *posiCursos* con *result= self.cursos-> select(c\| c.área=self.especialidad and c.nivel =* ‘ Grado’) | *Conflicto de Referencia Semántica de Operación Colgada* |
| Refinamiento semántico de Participante Instituto , operación *asignar* con *result=self.profes-> exists (p\| self.asigns->exists(a\| a.profe=p))* | *No hay conflicto* |
| Refinamiento semántico de Contexto con *Asignación.allInstances->forAll (a,b \| a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))* | *Posible conflicto de Referencia Colgada en Invariante*<br>(verificar Propiedad 2.a) |

*Tabla 9.2 Cancelación Sintáctica de Profesor (operación posiCursos)*

Cancelación Sintáctica de Profesor

Pensemos ahora que para seleccionar los posibles cursos que pueden asignarse a un profesor, la operación *selec*() de Instituto en el contrato AsignaciónPC ya no necesita invocar a *posiCursos* de Profesor, sino que realiza directamente el proceso. Un coarsening de Participante eliminaría a *posiCursos* de la cláusula de especialización de *selec*() y posteriormente debería realizarse una Cancelación Sintáctica del Participante Profesor para eliminar la operación en cuestión. Las consecuencias que trae la aplicación de este operador pueden verse en la Tabla 9.2. Vemos aquí que se producen dos conflictos: uno al combinarse dos operadores sintácticos y otro al combinarse un sintáctico (la cancelación) con un semántico. Estos dos conflictos son errores que no ofrecen solución, por lo que este cambio no podría formar parte en la combinación de las evoluciones.
Además, se plantea un conflicto de *Referencia Colgada en Invariante*, el cual tiene asociadas propiedades de aplicabilidad; tratándose de una cancelación de participante, debemos ver si se verifica la Propiedad 2.a, para lo cual la instanciamos.

Propiedad 2.a:

Sea $M_2$ un modificador de refinamiento o redefinición semántica de contexto;
*Si* $M_2$ es aplicable a R
*y* $R_1$ = Canp (R, $M_1$ ) cancelación sintáctica de participante de R por $M_1$
*entonces* $M_2$ es aplicable a $R_1 \Leftrightarrow \forall\ p \in$ Part ($M_1$) : $\forall$ m $\in \text{Int}_{M1}$(p): m no ocurre en Inv ( $M_2$ )

En nuestro caso,
$M_1$ = <{Profesor}, {}, {<Profesor, posiCursos, σ, {}>},*true* >

$M_2$ = <{}, {}, {}, *Asignación.allInstances->forAll (a,b | a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario)) >*

Debemos mostrar que:
$\forall\ p \in$ Part ($M_1$) : $\forall$ m $\in \text{Int}_{M1}$(p): m no ocurre en Inv ( $M_2$ )
Reemplazando por los valores de las funciones observadoras:
($\forall\ p \in$ {Profesor}: $\forall$ m $\in$ {posiCursos): m no ocurre en *Asignación.allInstances->forAll (a,b | a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))*
es decir,
para el participante Profesor, la operación posiCursos no ocurre en *Asignación.allInstance -> forAll (a,b | a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))*
lo cual es verdadero. Por lo tanto, la propiedad se cumple.

Un nuevo Refinamiento Semántico de Contexto

Por último, consideremos sobre el contrato original la posibilidad de expresar que en todas las asignaciones no sólo no deben superponerse los horarios del mismo profesor sino que además debe controlarse la superposición horaria de aulas y del dictado de cursos dentro de la misma área.

| **AsignaciónPC (Original)** | **Nueva Versión**:<br>Refinamiento semántico de Contexto con *Asignación.allInstances->forAll (a,b \| (a.profe = b.profe or a.aula = b.aula or a.curso.área=b.curso.área) implies a.horario.noSuper(b.horario))* |
|---|---|
| Refinamiento sintáctico de Participante Profesor , operación *posiCursos* con <*cursos*.nivel \|\| *cursos*.área> | *No hay conflicto* |
| Redefinición semántica de Participante Profesor , op. *posiCursos* con *result= self.cursos-> select(c\| c.área=self.especialidad and c.nivel =* ‘ Grado’) | *Posible conflicto de Inconsistencia Semántica* (verificar Propiedad 5) |
| Refinamiento semántico de Participante Instituto , operación *asignar* con *result=self.profes-> exists (p\| self.asigns->exists(a\| a.profe=p))* | *Posible conflicto de Inconsistencia Semántica* (verificar Propiedad 5) |
| Refinamiento semántico de Contexto con *Asignación.allInstances->forAll (a,b \| a.profe = b.profe or a.aula = b.aula implies a.horario.noSuper(b.horario))* | *Posible conflicto de Inconsistencia Semántica de Invariante* (verificar Propiedad 4.a) |

*Tabla 9.3 Un nuevo Refinamiento Semántico de Contexto*

Esto puede representarse refinando el Invariante del contrato. La Tabla 9.3 muestra el resultado de combinar este cambio con los que dieron origen al contrato derivado AsignaciónPC2.
Se generan aquí posibles conflictos de Inconsistencia Semántica pero puede probarse en ambos casos, instanciando la propiedad 5 asociada, la solución de la situación. Otro conflicto que se plantea es de Inconsistencia Semántica de Invariante, que tiene asociadas propiedades de aplicabilidad; como se trata de refinamientos debemos verificar la Propiedad 4.a.

Propiedad 4.a:
*Si* R es refinable semánticamente de contexto por $M_2$
*y* $R_1$ = Refsc (R, $M_1$ )
*entonces* $R_1$ es refinable semánticamente de contexto por $M_2 \Leftrightarrow$ *instancian* (Inv($M_1$), Inv ($M_2$))

En nuestro caso,

$M_1$ = <{}, {},{}, *Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula ) implies a.horario.noSuper(b.horario)) >*

$M_2$ = <{}, {},{}, *Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula or a.curso.área=b.curso.área) implies a.horario.noSuper(b.horario)) >*

Debemos mostrar que:

*instancian* (Inv($M_1$), Inv ($M_2$))

por definición de *pattern de refinamiento* dada en el modelo matemático:

Sean $G_1$,..,$G_n$ las fórmulas en las hojas del árbol. Sea G la fórmula en la raíz del árbol, entonces
1. $G_1 \wedge G_2 \wedge ... \wedge G_n \models G$ (*entailment)*
2. $G_1 \wedge G_2 \wedge ... \wedge G_n \not\models$ false (consistencia)

En este caso:

$G_1$ = *Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula) implies a.horario.noSuper(b.horario))*
$G_2$=*Asignación.allInstances -> forAll (a,b| a.curso.área=b.curso.área implies a.horario.noSuper(b.horario))*
G = *Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula) implies a.horario.noSuper(b.horario))*

Para mostrar 1.: $G_1 \wedge G_2 \models G$, o sea
*Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula) implies a.horario.noSuper(b.horario))* $\wedge$
*Asignación.allInstances->forAll(a,b| a.curso.área =b.curso.área implies a.horario. noSuper(b.horario))* $\models$
*Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula) implies a.horario.noSuper(b.horario))*

Para mostrar 2.: $G_1 \wedge G_2 \mid \neq$ false, o sea
*Asignación.allInstances->forAll (a,b | (a.profe = b.profe or a.aula = b.aula) implies a.horario.noSuper(b.horario))* $\wedge$
*Asignación.allInstances->forAll(a,b| a.curso.área =b.curso.área implies a.horario. noSuper(b.horario)* $\mid \neq$ false

basta considerar tener sólo dos instancias de Asignación con la siguiente valuación:
Sea a, b: Asignación donde :

a.profe.nombre = Juan Pérez, a.curso.títilo = Historia, a.aula.nro = 101,
a.horario.hInicio = 10, a.horario.hFin = 12
b.profe.nombre = Juan Pérez, b.curso.títilo = Historia, b.aula.nro = 102,
b.horario.hInicio = 12, b.horario.hFin = 14

Con esta valuación para a y b, se cumple con la definición de *pattern de refinamiento* dada en el modelo matemático. Por lo tanto, la propiedad se cumple.

***Un nuevo contrato combinado***

Los cambios presentados en esta sección son aplicables al contrato original así cómo, por otro lado, lo son las modificaciones presentadas en la sección 9.2 que dieran origen al contrato AsignaciónPC2. Llamamos AsignaciónPC3 al contrato derivado por los cambios de esta sección aplicados sobre AsignaciónPC.
El análisis de conflictos se realiza a fin de mantener consistencia en la evolución del diseño reusable y poder integrar modificaciones realizadas en forma independiente, en este caso, para poder integrar AsignaciónPC2 y AsignaciónPC3. De otra forma, el diseño evolucionaría por dos o más caminos separados.
La figura 9.6 muestra que no es posible la integración de estas dos evoluciones. Se considera entonces un posible contrato derivado, AsignaciónPC3´, donde la Cancelación de operación posiCursos y la Redefinición semántica del Participante Instituto no se apliquen dado que producen situaciones conflictivas, como lo mostró el análisis realizado en esta sección.
La integración entre AsignaciónPC2 y AsignaciónPC3´ da como resultado el contrato AsignaciónPC4, que se muestra en detalle en la Figura 9.7.

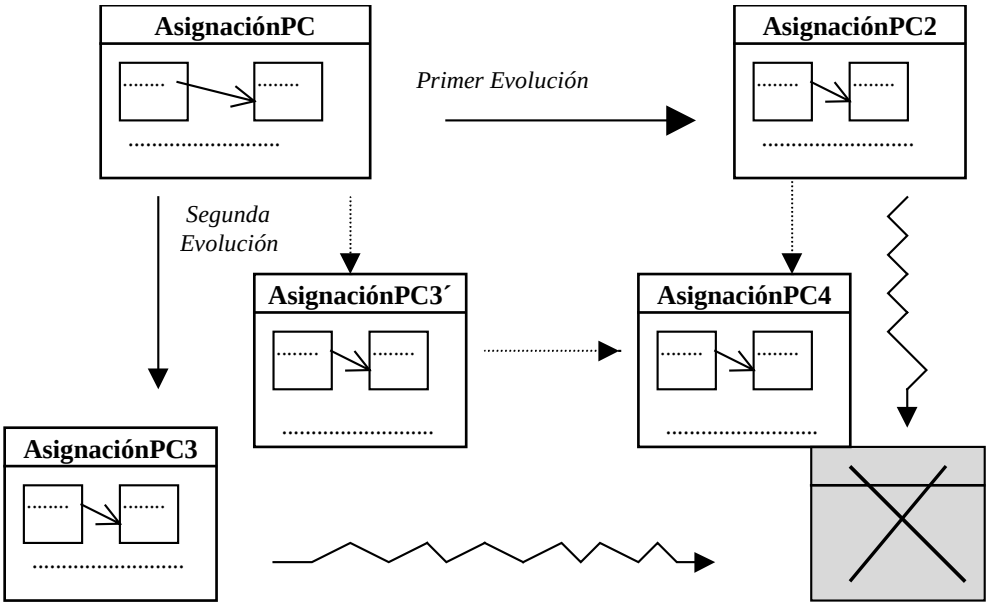


*Fig 9.6 Combinación de evoluciones del contrato AsignaciónPC*

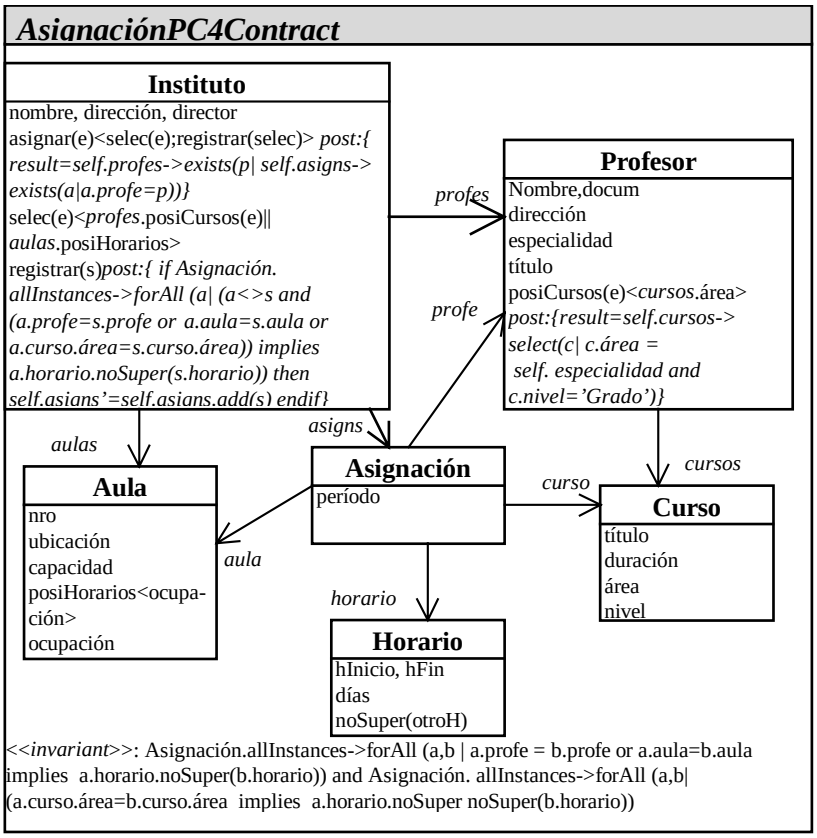


*Fig 9.7 El contrato combinado AsignaciónPC4Contract*

# Capítulo 10: Conclusiones

Los principales problemas que surgen en la construcción de componentes de software son, por un lado, la naturaleza cambiante de las aplicaciones actuales que obliga a desarrollar componentes flexibles y adaptables y por otro, la falta de documentación adecuada para lograr esa adaptación. Por lo tanto es necesario un mejor soporte para documentar cómo reusar componentes y cómo controlar, en la etapa de evolución, la propagación de cambios.

En este trabajo presentamos un mecanismo de especificación que permite documentación estructurada de componentes reusables, tanto a nivel sintáctico como semántico y ayuda al desarrollador de software a entender cómo una componente puede ser reusada y cómo manejar su evolución.

En [Lucas97a] se define un mecanismo para especificación de componentes denominado *contratos de reuso* y los operadores que documentan y restringen su evolución. Estos contratos brindan documentación estructurada de componentes reusables y asisten al ingeniero de software en la adaptación de componentes para necesidades particulares pero dejan de lado la verificación semántica de condiciones que los participantes de un contrato deben cumplir al desplegar ciertas acciones.

El objetivo de nuestro trabajo se centró en la integración de los contratos de interacción [HHG90], que permiten ciertas verificaciones, con los contratos de reuso a fin de poder expresar y documentar tanto la evolución estructural de componentes como la evolución semántica del comportamiento entre sus participantes.

Para lograr este objetivo, en este trabajo hemos presentado una extensión para contratos de reuso: *los contratos de reuso con semántica de comportamiento*, enriquecidos con post-condiciones para operaciones e invariantes para el contrato. Además, hemos introducido seis nuevos operadores de reuso que permiten modelar posibles cambios semánticos en el contrato: *los operadores de reuso semántico* con el fin de documentar la evolución tanto de participantes individuales, como del contexto del contrato.

Estos contratos fueron integrados al lenguaje de especificaciones gráficas UML, para permitir su uso práctico. Las post-condiciones para operaciones e invariantes para el contrato, fueron expresadas en el lenguaje de especificación de restricciones OCL, integrado a UML.

Específicamente:

- Presentamos un *modelo matemático* que nos permita estudiar contratos de reuso con semántica de comportamiento y sus operadores de reuso sintáctico y semántico.

- Realizamos un análisis de posibles conflictos que pueden surgir en la etapa de evolución al combinar de operadores de reuso (ya sea operadores sintácticos como semánticos). Los conflictos de evolución pueden generarse al realizar cambios sobre diferentes partes del sistema que se relacionan y al propagarse los cambios.

- Basándonos en el modelo matemático expresamos, demostramos y ejemplificamos *propiedades de aplicabilidad* que surgen del estudio de casos específicos en los que no ocurren conflictos al combinar operadores.

- Finalmente, hemos presentado ejemplos que muestran cómo los contratos de reuso con semántica de comportamiento pueden ser útiles en el proceso de desarrollo de software.

## *10.1 Contribuciones principales*

Este trabajo:

- Presenta una propuesta general para reuso disciplinado, basada en una comunicación bien documentada entre componentes reusables y quienes las reusen.

- Contribuye a que el desarrollador de componentes reusables pueda expresar y documentar tanto la evolución estructural de estas componentes como la evolución semántica del comportamiento entre sus participantes.

- Ayuda a que los desarrolladores de aplicaciones puedan conseguir un mayor entendimiento tanto de la estructura y comportamiento operacional del contrato como de la semántica del comportamiento de sus participantes.

- Finalmente permite la adaptación de componentes a necesidades particulares, estimando y manejando el impacto de cambios en la etapa de evolución.

## *10.2 Trabajo Futuro*

Basándonos en la noción de contrato de reuso con semántica de comportamiento, pueden ser construidas herramientas que brinden soporte a desarrolladores en diversas tareas como adaptación, evolución de componentes, evaluación de calidad, entre otras.

Nuestro trabajo futuro consiste en la implementación de un ambiente que soporte la definición y evolución de estos contratos de reuso. Este ambiente utilizará una notación gráfica basada en UML y soportará detección automática de conflictos y propagación de cambios.

## Bibliografía


[BRJ97] G. Booch, J. Rumbaugh, I. Jacobson. *Unified Method Language 1.0*, Technical Report Rational, 1997.

[CHSV97] W. Codenie, K. De Hondt, P. Steyaert, A. Vercammen. *Evolving Custom-made applications into domain-specific frameworks.* Communications of the ACM, October 1997.

[DL96] R.Darimont and A.Lamsweerde, Formal refinement patterns for goal-driven requirements elaboration. Software Engineering Notes, vol.21 no.6, November 1996.

[GHJV94] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns.*Addison-Wesley, 1994.

[GPB98a] R. Giandini, C. Pons, G. Baum. *Evolución de Contratos de Reuso multi-Clase con semántica de Comportamiento.*En Anales del Simposio en Orientación a Objetos "ASOO´98" en el marco de las XXVII JAIIO, Facultad de Ingeniería, UBA, Argentina, pág.119-132. Agosto 1998.

[GPB98b] R. Giandini, C. Pons, G. Baum. *Manejando Formalmente Evolución de Contratos de Reuso con Semántica de Comportamiento*. Actas III Jornadas en Ingeniería del Software, Murcia, España, pág. 167-179. Noviembre de 1998.

[GR95] A. Godlberg, K. Rubin. *Succeeding with objects: Decision Frameworks for Proyect Management.* Addison-Wesley, 1995.

[HHG90] R. Helm, I.M. Holland, D. Gangopadhyay. *Contracts: Specifying behavioral compositions in object-oriented systems*. In ECOOP/OOPSLA´90. ACM Press, pag. 169-180, October 1990.

[Joh92] R.E. Johnson. *Documenting Frameworks using patterns.* In Proceedings OOPSLA´92, ACM SIGPLAN Notices,pag. 63-76,October 1992.

[JF88] R.E. Jhonson, B. Foote. *Designing reusable classes*. Journal of Object-Oriented Programming, 1(2), pag. 122-132. February 1988.

[JGJ97] I. Jacobson, M. Griss, P. Jonsson. *Software Reuse: Arquitecture, Process and Organization for Business Success.* Addison-Wesley, 1997.

[KKS96] N. Klarlund, J. Koinstinen, M. Schwartzbach. *Formal design constraint*. In Proceedings OOPSLA´96, ACM SIGPLAN Notices,pag. 370-383, October 1996.

[KL92] G. Kiczales, J. Lamping. *Issues in the Design and specification of Class Libraries*. In Proceedings OOPSLA´92, ACM SIGPLAN Notices,pag. 435-451, October 1992.

[Lam93] J. Lamping. *Typing the specialization interface*. In Proceedings OOPSLA´93, ACM SIGPLAN Notices, pag. 201-214,October 1993.

[LSM97] C. Lucas, P. Steyaert, K. Mens. *Managing Software Evolution through Reuse Contracts*. In Proceedings of the First Eorumicro Conference on Software Maintenance and Rengineering, IEEE Press, March 1997.

[Lucas97a] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts, Chap.1*. PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Lucas97b] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts, Chap.2*. PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Lucas97c] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts, Chap.1 pag,11-16, Chap.3*. PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Lucas97d] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts, Chap.5 pag.127-133*. PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Mey92] B.Meyer. *Advances in Object-Oriented Software Engineering. Chapter 1 "Design by Contract"*.Prentice Hall, 1992

[Mez97] M. Mezini. *Maintaining the consistency of class libraries during their evolution*. In Proceedings OOPSLA´97, ACM SIGPLAN Notices, pag. 1-21. October 1997.

[MLS96] K. Mens, C. Lucas, P. Steyaert. *Formalising Operations on ACIDs and Their Interactions*.Technical Report vub-prog-tr-96-03, Vrije Universiteit Brussel, Belgium, 1996.

[MLS98a] K. Mens, C. Lucas, P. Steyaert. *Giving Precise semantics to Reuse in UML*. In ICSE'98 Workshop on Precise semantics of Modeling Techniques, Japan, April 1998.

[MLS98b] T. Mens, C. Lucas, P. Steyaert. *Supporting reuse and evolution of UML models*. In P.-A. Muller and J. Bézivin, editors, Proceeding of <<UML>>'98 International Workshop, Mulhouse, France, pages 341-350, 1998.

[Pan95] C. Pancake. *Object rountable, the promise and the cost of object technology: a five-year forecast.* Communications of the ACM, October 1995.

[Pre94] W. Pree. Meta-Patterns: *Abstracting the essentials of Object-Oriented Frameworks*. In Proceedings ECOOP 1994, Berlin: Springer-Verlag, pag. 150-164, 1994.

[Pre95] W. Pree. *Design Patterns for Object-Oriented software Development*. Addison-Wesley, 1995.

[Pre96] W. Pree. *Frameworks Patterns*. SIGS Publications, 1996.

[PG99] C. Pons, R. Giandini. *Precise Semantics of Model Evolution.* Conferencia IDEAS'99 realizada en el Instituto Tecnológico de Costa Rica, Costa Rica, Marzo de 1999.

[RMH97] Rational Software, Microsoft, Hewlett-Packard et all. *Object Constraint Language Specification version 1.1,* September 1997.

[SLMH96] P. Steyaert, C. Lucas, K. Mens, T. D´Hondt. *Reuse Contracts:Managing the evolution of Reusable Assets.* In Proceedings OOPSLA´96, ACM SIGPLAN Notices, pag. 268-285. Octobre 1996.

[Str86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

[Sun97] Sun Microsystems, JavaSoft. *Javabeans 1.0 API Specification*, 1997.