Una Formalización Para Documentación Y Evolución De Componentes Reusables

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

LIFIA-Universidad Nacional de La Plata
Calle 50 y 115 - 1er Piso - La Plata (1900), Argentina
[giandini, cpons]@sol.info.unlp.edu.ar

Resumen

Los principales problemas que surgen en la construcción 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 rehusar componentes y cómo controlar cambios en la evolución. En [Giandini 98a] presentamos un mecanismo de especificación: *contratos de reuso con semántica de comportamiento,* 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 este trabajo, presentamos un *modelo matemático* para estudiar estos contratos y sus operadores de reuso sintáctico y semántico. Basándonos en este modelo expresamos *propiedades de aplicabilidad* para la combinación de operadores en la etapa de evolución. Desarrollamos también, un *caso de estudio* sobre el que instanciamos el modelo y que muestra cómo los contratos de reuso con semántica de comportamiento pueden ser útiles en el proceso de desarrollo de software.

1 Introducción

Actualmente, los desarrolladores de software se ven confrontados a cambios constantes en el mercado. Por esto, surge la necesidad de obtener componentes flexibles y adaptables que acompañen la naturaleza cambiante de las aplicaciones. Utilizar este tipo de componentes para la construcción de aplicaciones puede resultar costoso aún contando con documentación apropiada. La calidad de la documentación es básicamente importante en las etapas de mantenimiento y evolución. En la documentación de componentes *reusables* se presentan problemas de mayor complejidad por la naturaleza abstracta del diseño.
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 [Goldberg 95], [Pancake 95].
En cuanto al problema de la consistencia en la evolución de librerías de clases y *frameworks,* diversos autores, [Pree 96, Johnson 88, Johnson 92, Kiczales 92, Stroustrup 86] enfatizan la importancia de contar con descripciones de interfaces claras pero no ofrecen soluciones particulares. En tal sentido, el trabajo de [Klarlund 96] ayuda en el chequeo automático de validación de un número de restricciones de diseño, mientras el trabajo de [Mezini 97] ayuda además a resolver automáticamente algunos problemas que fueron especificados por diseñadores. 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.
Como una solución a estos problemas surgen los contratos de reuso [Lucas 97a]. 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.
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 [Codenie 97]. Los contratos de reuso simple-clase [Steyaert 96] están basados en la noción de Lamping de interfaz de especialización [Lamping 93]. Otros autores, como Ivar Jacobson sugieren controlar mediante *facades* (interfaces especiales) el acceso a sistemas de componentes [Jacobson 97]. Los contratos de reuso multi-clase [Lucas 97 y Lucas 97a] están basados en los contratos de interacción de Helm y Holland [Helm 90].
Helm define 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. Otra propuesta similar descripta mediante mecanismos del lenguaje de programación orientado a objetos Eiffel, es presentada por [Meyer 92]. Sin embargo, en ambos casos, los autores no brindan elementos de documentación que permitan expresar distintas formas de evolución del contrato ni la de los participantes.
En [Lucas 97b] 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. La noción de contrato de reuso es utilizada también en [Mens 98a], donde se propone extender UML [Rational 97] 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 [Mens 98b] los autores logran la extensión del metamodelo UML e introducen una notación para expresar reuso y evolución de cualquier elemento dentro de UML. En [Pons 99] se presenta una semántica formal basada en UML y lógica dinámica para conflictos en la evolución.
En [Giandini 98a] hemos presentado una 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. En [Giandini 98b] realizamos 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.
El presente trabajo continúa con el estudio de los contratos de reuso con semántica de comportamiento. Específicamente:

- Definimos un *modelo matemático* que nos permita estudiar contratos de reuso con semántica de comportamiento y sus operadores de reuso sintáctico y semántico.
- Basándonos en el modelo matemático expresamos *propiedades de aplicabilidad* que surgen del estudio de casos específicos en los que no ocurren conflictos al combinar operadores.
- Finalmente presentamos un *caso de estudio* sobre el que instanciamos el modelo y que muestra cómo los contratos de reuso con semántica de comportamiento pueden ser útiles en el proceso de desarrollo de software.

Este artículo está organizado de la siguiente forma: en la sección 2 presentamos una definición de la noción de contrato de reuso con semántica de comportamiento y los operadores de reuso sintáctico y semántico. La tercera sección presenta análisis de conflictos en la interacción de operadores sobre un contrato. La cuarta sección introduce el *modelo matemático* para estudiar estos contratos. En la sección 5 se definen propiedades de aplicabilidad de operadores sobre un contrato, expresadas mediante el modelo matemático. La sexta sección desarrolla un *caso de estudio* que, utilizando del modelo matemático, muestra el beneficio de estos contratos en el desarrollo de software. Por último se presentan conclusiones y se discute la dirección que seguirá el trabajo futuro.

2 Conceptos Básicos

En los siguientes apartados presentamos las definiciones básicas de esta metodología de especificación.

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.

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. La post-condición debe asegurar que *m* preserve 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) [Rational 97a] es un lenguaje de especificación formal integrado al UML [Rational 97]. Las invariantes en un contrato, son expresiones OCL (properties) libres de efectos laterales.
En las post-condiciones pueden aparecen variables "*primadas"* que representan el valor de la variable en el estado siguiente a la ejecución de la operación; esto puede verse en la Figura 1. Las variables "*primadas"* son una extensión de la seudovariable "*result*" de OCL. El contrato de reuso puede expresarse en UML para poder aplicarlo al campo de librerías de clases y frameworks orientados a objetos (ver [Giandini 98a]).

La Fig.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.*

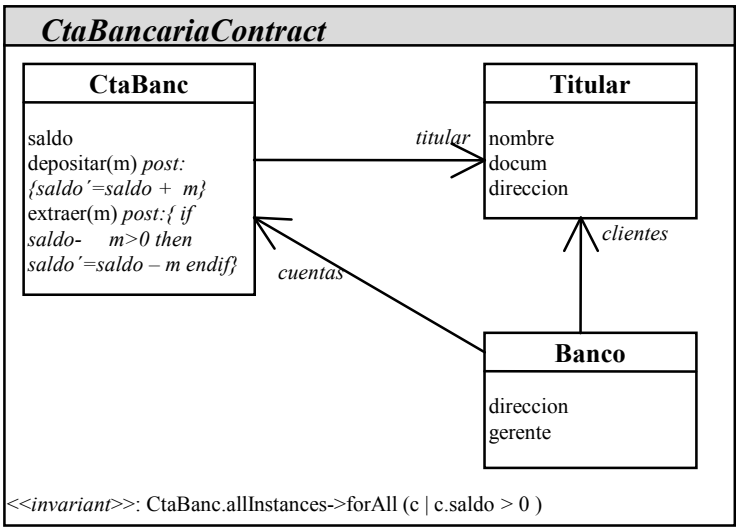

Figura 1. El contrato de reuso CtaBancaria

2.2 Operadores sobre Contratos de Reuso

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.
La Tabla 1presenta los *operadores de reuso sintáctico.*

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

Tabla 1. Operadores de reuso sintáctico

Por ejemplo, el objetivo de la **Extensión sintáctica de participante** es agregar nuevas operaciones a uno o más participantes en un 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.
La extensión sintáctica de participante no admite que la operación se agregue con post-condición. Si es necesario explicitarla, se hará aplicando un operador de reuso semántico.

Los *operadores de reuso semántico*, presentados en la Tabla 2, son operadores introducidos en [Giandini 98a] 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 su post-condición), como del contrato (es decir que refina o redefine el invariante).

Operador	Descripción	Significado
Red	Redefinición semántica de Participante	redefinir la post-condición de una operación del participante
Refsp	Refinamiento semántico de Participante	refinar la post-condición de una operación del participante
Coasp	*Coarsening* semántico de Participante	debilitar la post-condición de una operación del participante
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

Tabla 2. Operadores de reuso Semántico

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 el Anexo I se define y ejemplifica el **Refinamiento semántico de participante** que enriquece el comportamiento de una operación modificando su post-condición, adicionándole expresiones, de manera tal que la operación siga preservando el invariante del contrato.
En los capítulos 3 y 4 de [Giandini 99], se puede ver la definición completa de todos los operadores de reuso.

3 Conflictos en la Evolución de un Contrato de Reuso

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 2, queremos ver si la combinación de ambos es posible y si el resultado será el esperado.
En general, todos los conflictos considerados, son detectados comparando los modificadores de reuso sin necesidad de consultar el contrato base. En [Giandini 98b] planteamos y analizamos algunos conflictos que pueden ocurrir al incorporar los operadores de reuso semántico. 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

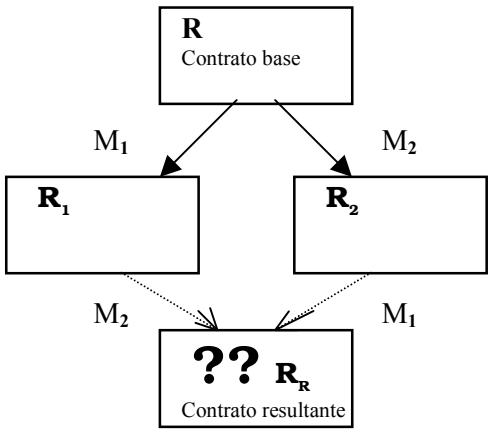

Figura 2. Combinación de Operadores

Los conflictos de *inconsistencia semántica*, por ejemplo, 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, o viceversa. El análisis de este conflicto se basa en la relación de consistencia entre elementos semánticos de un contrato.

4 Un Modelo Matemático para Contratos de Reuso con Semántica de Comportamiento

Basándonos en Mens [Mens 96], que formaliza contratos de reuso simple-clase definidos en [Steyaert 96 y Codenie 97], 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, como se verá en la próxima sección. Esta notación permitirá, como trabajo posterior, la implementación de un chequeador de conflictos en la evolución de componentes reusables.

4.1 Contrato de reuso con semántica de comportamiento

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

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

CON = PART1 X PART2 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 = P(STRING) dominio para nombres de participantes del contrato

ASOC, PART1, PART2, OP $\subset$ STRING dominios para nombres de relaciones, participantes que se relacionan y operaciones del contrato, respectivamente.

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

COND = 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.

CESPEC es el dominio para cláusulas de especialización, donde cada elemento e perteneciente a CESPEC se define recursivamente de la siguiente forma:
e:: = OP | e1; e2 | e1 || e2 | σ
Y donde σ denota el valor nulo para el dominio de las cláusulas de especialización CESPEC;

STRING es el dominio primitivo de las cadenas de caracteres.

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

- Part :: CONTRATO $\rightarrow$ P(PART)
  Part(R) = Π_1(R) conjunto de todos los nombres de participantes de R
  Donde Π_i es la proyección i de la n-upla para algún i de 1 a n.

- Inv :: CONTRATO $\rightarrow$ INV
  Inv(R) = Π_4(R) invariante del contrato de reuso R.

- Con::CONTRATO $\rightarrow$ PART$\rightarrow$ P(ASOC)
  Con_R (p) = *map* Π_3 (*filter* $\Pi_1(x)$=p Rel (R)) conjunto de todas las relaciones de conocimiento del participante *p* en R. Donde Rel (R) = Π_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$\rightarrow$ P(OP)
  Int_R (p) = *map* Π_2 (*filter* $\Pi_1(x)$=p Interf(R)) conjunto de nombres de operaciones del participante p en R
  Donde Interf (R) = Π_3(R)

- Espec:: CONTRATO $\rightarrow$ PART$\rightarrow$ P(OP) $\rightarrow$ CESPEC
  $Espec_{R\,p}$ (m) = Π_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$\rightarrow$ P(OP)$\rightarrow$ COND
  $Pcond_{R\,p}$ (m) = Π_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 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

4.2 Operadores y 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 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.
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 [Giandini 99] capítulo 7, damos la notación matemática para cada operador de reuso sintáctico y semántico, que no podemos incluír aquí por límites de espacio. En el Anexo I, incluímos la formalización del refinamiento semántico de participante.

5 Propiedades de Aplicabilidad de Operadores de Reuso

Basándonos en el modelo matemático para contratos y operadores de reuso presentado en la sección anterior y en el análisis de conflictos 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. A continuación enunciamos algunas de estas propiedades, su demostración puede encontrarse en el capítulo 8 de [Giandini 99]. Su aplicación puede verse en el caso de estudio de la próxima sección, en la etapa de análisis de conflictos al combinar operadores.

Composición de cancelación sintáctica y operador semántico de contexto (Conflicto de Referencia Colgada en Invariante)

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 $R_1 \Leftrightarrow \forall\, p \in Part\ (M_1) : \forall\, m \in Int_{M1}(p)$: m no ocurre en Inv (M_2)

Las propiedades 2.b y 2.c combinan el operador semántico de contexto con cancelación y coarsening de contexto respectivamente.

Consistencia en la Composición semántica de participante (Conflicto de Inconsistencia Semántica de Operación)

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 $\mathbb{P}$(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.

La Propiedad 3.b combina dos operadores de coarsening semántico de participante.

Consistencia en la Composición semántica de contexto (Conflicto de Inconsistencia Semántica de Invariante)

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))

donde el predicado *instancian* se define:
instancian:: EXPOCL X EXPOCL $\rightarrow$ BOOL
instancian (exp, exp') = true, si exp y exp' son expresiones OCL consistentes y tales que exp' $\rightarrow$ exp
= false, en caso contrario

La Propiedad 4.b combina dos operadores de coarsening semántico de contexto.

Composición de operador semántico de participante y operador semántico de contexto
(Conflicto de Inconsistencia Semántica)

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).

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 $R_1 \Leftrightarrow$
$\forall p \in Part(M_1): \forall m \in Int_{M1}(p): Inv(M_2) \rightarrow Inv(M_2)[Pcond_{M1\ p}(m)]$

6 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 esta sección presentamos, utilizando e instanciando al modelo matemático ya definido, 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. El control de situaciones puede ser implementado en términos del modelo matemático.
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 3 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.

6.1 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. El operador a aplicaro es un Refinamiento sintáctico del Participante Profesor.

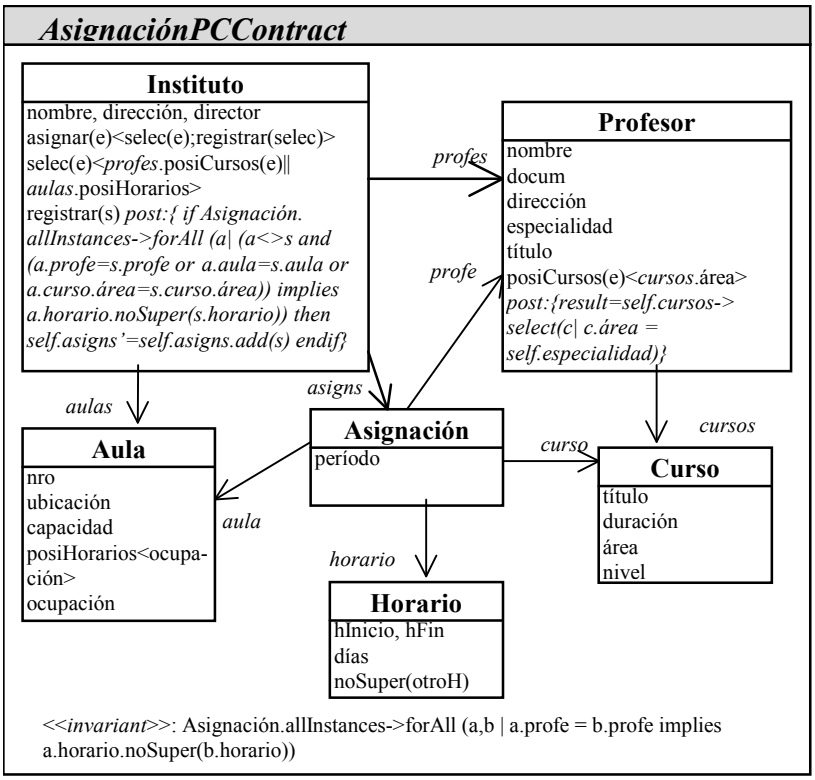

Figura 3. El contrato de Reuso AsignaciónPC

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.
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.
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.
Llamamos AsignaciónPC2 al contrato 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 uno de los operadores utilizados, instanciando el modelo matemático. En forma similar pueden verificarse los demás.

6.2 Aplicabilidad de los operadores de reuso instanciando el Modelo Matemático

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))*>

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

Redefinición semántica del participante Profesor :
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) $m \in Int_M(p)$: $m \in Int_M(p)$
(3)∀ $m \in Int_M(p)$: ∀ e ∈ $Pcond_{M\,p}(m)$: *bienFormada*(e, R)
(4)∀ $m \in Int_M(p)$: $Inv(R) \rightarrow Inv(R)[Pcond_{M\,p}(m)]$
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)
∀ $p \in$ Part (M): $p \in$ Part(*AsignaciónPC*) , reemplazando
∀ $p \in$ {Profesor}: $p \in$ Part(*AsignaciónPC*) , es decir
∀ $p \in$ {Profesor}: $p \in$ {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)
∀ $p \in$ {Profesor}: ∀ m ∈ Int_M (Profesor): m ∈ $Int_{AsignacionPC}$ (Profesor) reemplazando
∀ $p \in$ {Profesor}: ∀ m ∈ {posiCursos}: m ∈ { posiCursos, nombre, dirección, docum, especialidad, título} es decir,
para el participante Profesor, la operación posiCursos ∈ { posiCursos, nombre, dirección, docum, especialidad, título} lo cual es verdadero
Para mostrar (3)
∀ $p \in$ {Profesor}: ∀ m ∈ {posiCursos}: ∀ e ∈ $Pcond_{M\,p}(m)$:
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 [Rational 97a]) e involucra elementos de AsignaciónPC. Por lo tanto el punto (3) se cumple.
Para mostrar (4),
∀ $p \in$ {Profesor}: ∀ m ∈ {posiCursos}:
$Inv(AsignaciónPC) \rightarrow Inv(AsignaciónPC)[Pcond_{M\,p}(m)]$, 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.

6.3 Nuevos Cambios en el Contrato Base

El contrato de reuso base 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 y controlar el cumplimiento de las propiedades de aplicabilidad asociadas.

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 3. 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.

AsignaciónPC (Original)	**Nueva Versión**: 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* (verificar Propiedad 2.a)

Tabla 3. Cancelación Sintáctica de Profesor (operación posiCursos)

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 (ver Sección 5), para lo cual la instanciamos.
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 $\in$ 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**: 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 4. Un nuevo Refinamiento Semántico de Contexto

Esto puede representarse refinando el Invariante del contrato. La Tabla 4 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, que los dos operadores semánticos de participante preservan el nuevo invariante. Otro conflicto que se plantea es de Inconsistencia Semántica de Invariante, que tiene asociadas propiedades de aplicabilidad; como se trata de refinamientos se debe instanciar la Propiedad 4.a en forma similar a lo realizado para la Propiedad 2.a.

Un nuevo contrato combinado
Los cambios presentados en este apartado son aplicables al contrato original así cómo, por otro lado,

lo son las modificaciones presentadas en el apartado 6.1 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 4 muestra que no es posible la integración de estas dos evoluciones. Se considera entonces un posible contrato derivado, AsignaciónPC3´, donde la Cancelación de la operación posiCursos no se aplique 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.

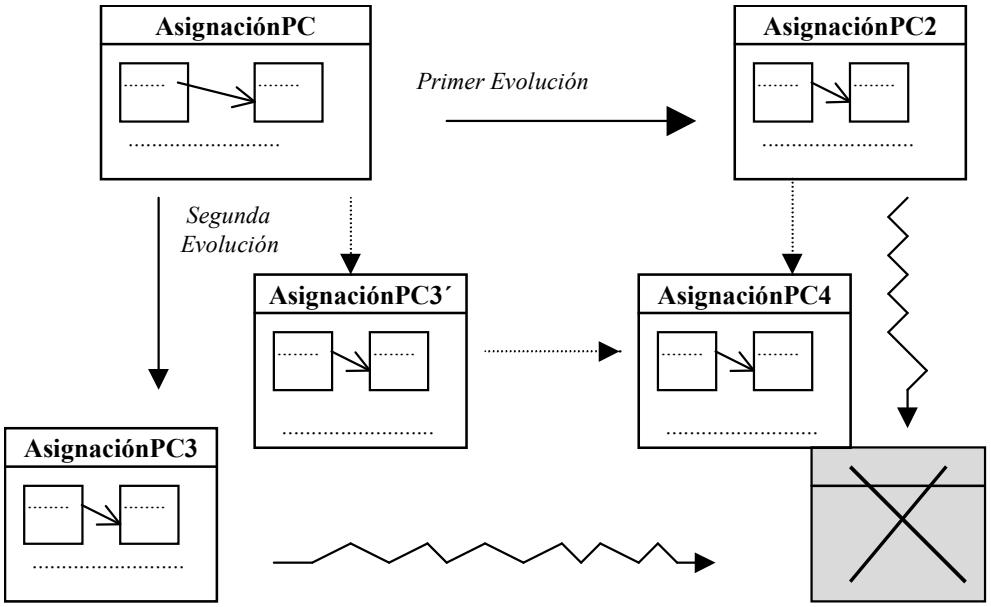

Figura 4 Combinación de evoluciones del contrato AsignaciónPC

7 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 [Giandini 98a] presentamos un mecanismo de especificación: *contratos de reuso con semántica de comportamiento,* 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. Una extensión de esta idea fue aportada en [Giandini 98b], donde se incluye el análisis de conflictos en la evolución y se presentan algunas propiedades de aplicabilidad.

En este trabajo, específicamente:

- Presentamos un *modelo matemático* que nos permita estudiar contratos de reuso con semántica de comportamiento y sus operadores de reuso sintáctico y semántico.
- Basándonos en el modelo matemático expresamos *propiedades de aplicabilidad* que surgen del estudio de casos específicos en los que no ocurren conflictos al combinar operadores.

- Finalmente presentamos un *caso de estudio* sobre el que instanciamos el modelo y que muestra cómo los contratos de reuso con semántica de comportamiento pueden ser útiles en el proceso de desarrollo de software.

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

[Rational 97] G. Booch, J. Rumbaugh, I. Jacobson. *Unified Method Language 1.0*, Technical Report Rational, 1997.

[Codenie 97] W. Codenie, K. De Hondt, P. Steyaert, A. Vercammen. *Evolving Custom-made applications into domain-specific frameworks.* Communications of the ACM, October 1997.

[Giandini 99] R. Giandini.*Documentación y evolución de componentes reusables: Contratos de reuso con semántica de comportamiento*. Tesis del Magister en IS (no publicada)http://www-lifia.info.unlp.edu.ar/~giandini, 1999.

[Giandini 98a] 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 realizadas en la Facultad de Ingeniería, UBA, Argentina, pág.119-132, Agosto 1998.

[Giandini 98b] R. Giandini, C. Pons, G. Baum. *Manejando Formalmente Evolución de Contratos de Reuso con Semántica de Comportamiento*. En actas de las III Jornadas en Ingeniería del Software realizadas en Murcia, España, del 11 al 13 de Noviembre de 1998.

[Goldberg 95] A. Goldberg, K. Rubin. *Succeeding with objects: Decision Frameworks for Proyect Management.* Addison-Wesley, 1995.

[Helm 90] 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.

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

[Johnson 88] R.E. Jhonson, B. Foote. *Designing reusable classes*. Journal of Object-Oriented Programming, 1(2), pag. 122-132. February 1988.

[Jacobson 97] I. Jacobson, M. Griss, P. Jonsson. *Software Reuse: Arquitecture, Process and Organization for Business Success.* Addison-Wesley, 1997.

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

[Kiczales 92] 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.

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

[Lucas 97] 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.

[Lucas 97a] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts*, C*hap.1.* PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Lucas 97b] Carine Lucas. *Documenting Reuse and Evolution with Reuse Contracts, Chap.2.* PhD Thesis, Department of Computer Science Vrije Universiteit Brussel, Belgium, Set. 1997

[Meyer 92] B.Meyer. *Advances in Object-Oriented Software Engineering. Chapter 1 "Design by Contract"*.Prentice Hall, 1992

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

[Mens 96] 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.

[Mens 98a] 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.

[Mens 98b] 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.

[Pancake 95] C. Pancake. *Object rountable, the promise and the cost of object technology: a five-year forecast.* Communications of the ACM, October 1995.

[Pons 99] C. Pons, R. Giandini. *Precise Semantics of Model Evolution.*Conferencia IDEAS'99 realizada en el Instituto Tecnológico de Costa Rica, Costa Rica, del 24 al 26 de Marzo de 1999.

[Pree 96] W. Pree. *Frameworks Patterns*. SIGS Publications, 1996.

[Rational 97a] Rational Software, Microsoft, Hewlett-Packard et all. *Object Constraint Language Specification version 1.1,* September 1997.

[Steyaert 96] 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.

[Stroustrup 86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.

ANEXO I

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* (Fig.1, sección 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 A.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 la figura se muestra la parte del contrato que interesa a las modificaciones.

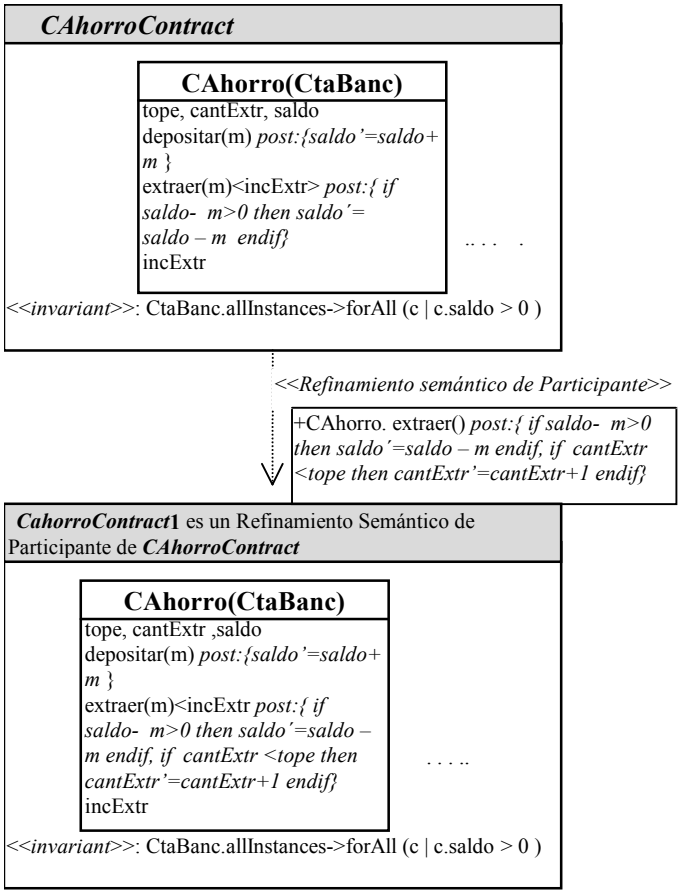

Figura A.1. Un ejemplo de Refinamiento Semántico de Participante

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.

Es decir, el operador **Refsp**, en el *modelo matemático* se define:

Definición de Aplicabilidad
Sea R ∈ CONTRATO un contrato de reuso bien formado y M ∈ CONTRATO un modificador de reuso.
El operador *Refsp* (refinamiento semántico 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)
(3)∀ m ∈ Int_M(p): ∀ e ∈ ($Pcond_{M\ p}$(m) - $Pcond_{R\ p}$(m)): *bienFormada*(e, R) ∧ *noExiste*(e, $Pcond_{R\ p}$(m))
(4)∀ *m* ∈ Int_M(p): Inv(R)→Inv(R)[$Pcond_{M\ p}$(m)]
donde el predicado *noExiste* se define:
noExiste:: EXPOCL X P(EXPOCL) → 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 ∈ CONTRATO, R'= *Refsp*(R, M) es un Refinamiento semántico de Participante de R por M ⇔
(1) *Refsp* es aplicable a R por M
(2) ∀ p ∈ Part(R) - Part (M) : p ∈ Part(R') ∧ $Int_{R'}$(p) = Int_R (p) ∧ $Con_{R'}$ (p) = Con_R(p)
(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) ∧ $Pcond_{R'}$ (m) = $Pcond_R$ (m)
(c) ∀ m ∈ Int_M (p) : m ∈ $Int_{R'}$ (p) ∧ $Espec_{R'\ p}$ (m) = $Espec_{R\ p}$(m) ∧ $Pcond_{R'\ p}$ (m) = $Pcond_{M\ p}$ (m)
(4) Inv(R') = Inv(R)