Revista Avances en Sistemas e Informática
ISSN: 1657-7663
avances@unalmed.edu.co
Universidad Nacional de Colombia
Colombia

Tabares, Marta Silvia; Zapata, Carlos Mario; Arango, Fernando
Un Método para el Refinamiento de los Atributos Derivados del Diagrama de Clases
Revista Avances en Sistemas e Informática, vol. 2, núm. 1, junio, 2005, pp. 1-8
Universidad Nacional de Colombia
Medellín, Colombia

Un Método para el Refinamiento de los Atributos Derivados del Diagrama de Clases

Marta Silvia Tabares
ESCUELA DE INGENIERÍA DE ANTIOQUIA.
mstabare@unalmed.edu.co

Carlos Mario Zapata y Fernando Arango

UNIVERSIDAD NACIONAL DE COLOMBIA.
Facultad de Minas. Escuela de Sistemas. Grupo UN-INFO
{cmzapata ; farango}@unalmed.edu.co

Recibido para revisión Mar 2005, aceptado May 2005, versión final recibida May 2005

Resumen: Las técnicas de Abstracción y Refinamiento aplicadas a los modelos conceptuales orientados a objetos, permiten a los ingenieros de software depurar los requisitos del interesado, de forma tal que puedan obtenerse productos válidos y óptimos, que garanticen la consistencia de la información en la etapa de desarrollo. En este artículo se propone un método que provee reglas de refinamiento expresadas en un lenguaje formal para ser empleadas entre las etapas de análisis y diseño del ciclo de vida del desarrollo de un sistema de información. Específicamente, estas reglas parten de casos de uso cuya funcionalidad se expresa mediante métodos y atributos derivados pertenecientes a clases relacionadas con el caso de uso.

Palabras Clave: Diagrama de Clases UML, Refinamiento, Métodos Formales, Atributos Derivados.

Abstract: Abstraction and Refinement techniques applied to Object - Oriented Conceptual Models, allows Software Engineers debugging stakeholders' requirements, getting optimal and valid products that guarantee the information consistency at the development stage. In this paper, we propose a method that provides refinement rules, expressed in a formal language to be used in the transition between analysis and design stages of the Information Systems Life Cycle. Specifically, these rules are taken from use a case whose functionality is expressed by methods and derived attributes belonging to related classes to the use case.

Keywords: UML Class Diagram, Refinement, Formal Methods, Derived Attributes.

1 INTRODUCCIÓN

En el contexto de UML [véase UML (2005)], a las Clases se les proveen características que les permiten agrupar objetos del dominio del negocio a través de la definición de atributos, operaciones y relaciones entre ellas, para así describir los términos que el dominio requiera. A partir de esto, es posible la aplicación de técnicas de Abstracción y Refinamiento que le permitan a los analistas descender en el nivel de abstracción y acercarse paulatinamente a la solución informática concreta a los problemas de los interesados. Este proceso generalmente se realiza de manera heurística, tomando como punto de partida los modelos del dominio Larman (2003), en los cuales se incluye sólo una descripción de las clases y las relaciones básicas entre ellas; sin embargo, en la definición de los diagramas de clases, UML considera elementos que permiten realizar un proceso de refinamiento en la etapa de análisis, refiriéndose éste a la adición de más elementos a medida que se avanza en las etapas del desarrollo [véase Shen, Lu y Low (2003)], tales como: estereotipos, interfaces y encapsulamiento, además de otras operaciones no consideradas en la construcción inicial de las clases [Booch, Jacobson y Rumbaugh (1997)]. Otros elementos como los atributos derivados, por ejemplo, requieren una definición precisa que se suele dejar a etapas muy cercanas a la programación, lo cual suele dejar en manos del programador decisiones de diseño que deberían estar más ligadas con los analistas y diseñadores.

A partir de los casos de uso, y más particularmente de las interacciones entre el usuario final y el sistema, que se suelen bosquejar inicialmente al elaborar los casos de uso, es posible identificar tales atributos, y darles un tratamiento formal que posibilite a los programadores

una transición más suave hacia el código.

El propósito de este artículo es mostrar un método aplicable al refinamiento del diagrama de clases en la transición del análisis al diseño, que permita identificar los atributos derivados a partir de las interacciones con el sistema futuro, que se describen de manera general en los casos de uso del sistema; una vez identificados, se expresan las reglas de refinamiento en el diagrama de clases a través de un lenguaje formal que combina la lógica de predicados y los métodos formales orientados a objetos.

Este artículo está organizado así: en la sección 2 se muestran diferentes trabajos alrededor del refinamiento del diagrama de clases; en la sección 3 se compendian algunos de los problemas asociados con el refinamiento del diagrama de clases; en la sección 4 se propone un método de refinamiento para el diagrama de clases que identifique las operaciones a partir de los atributos derivados; en la sección 5 se incluye un caso de aplicación del método y en la sección 6 se presentan las conclusiones y los trabajos futuros que se pueden derivar de este artículo.

2 ANTECEDENTES

El refinamiento del Diagrama de Clases ha sido tratado en la literatura desde diferentes ópticas. En D'Souza y Wills (1998) el refinamiento y la abstracción se definen como dos puntos de vista que permiten dinamizar los modelos dependiendo del detalle mismo que se desee manejar de ellos, formando a la vez la consistencia necesaria para que puedan mantenerse y evolucionar en el tiempo. El detalle de rigurosidad en el refinamiento que se requiere para hacer un seguimiento no tiene un formato determinado: se puede desde documentar hasta chequear el detalle matemático. En la propuesta de Catalysis [D'Souza y Wills (1998)] se realiza una combinación de notación gráfica y descripciones en C^{++} y OCL para la descripción de los diferentes tipos de refinamiento que se pueden realizar sobre un conjunto específico de modelos que hacen parte de un proyecto de software. En este caso se habla de refinamiento de objetos, acciones, operaciones y modelos, con el fin de determinar los diferentes niveles de abstracción que se pueden generar cuando se ocultan ciertos detalles del modelo. En este caso específico se utiliza constantemente el concepto de colaboración para la realización del refinamiento del modelo y se puede llegar hasta detalles de implementación del mismo. El refinamiento que aquí se utiliza se enfoca en la determinación de las relaciones internas entre los elementos de los modelos para ir develando su nivel de detalle.

Una óptica diferente se encuentra en France y Bieman (2001), donde se entregan algunos lineamientos para realizar el manejo del refinamiento del diagrama de clases, considerando múltiples vistas de una determinada pieza de software, que no son otra cosa que diferentes tipos de modelos UML que apoyan la comprensión del problema (diagramas de clases, interacción o casos de uso referidos en la misma). Específicamente lo definen como "el resultado en la elaboración de las propiedades de las clases que puede involucrar: refinamiento de atributos y operaciones (refinamiento del tipo de dato abstracto), la adición de nuevas propiedades (atributos, operaciones, relaciones) o la descomposición de una clase en una subestructura". Para hacer el refinamiento toma parámetros básicos de D'Souza y Wills (1998), pero no se establece un método preciso para realizarlo.

En otros estudios como el de Shen et al. (2003), se va más allá de adicionar elementos básicos a las clases y se plantea como tal la dificultad de modelar diferentes vistas del sistema en UML en el proceso de refinamiento. Para esto se toma uno de los mecanismos de extensión proporcionado por esta herramienta al nivel del metamodelo, el estereotipo, que es un metaelemento definido por el interesado, el cual es usado para extender un metaelemento UML llamado clase base.

Además, dividen el refinamiento del diagrama de clases en dos categorías: refinamiento de clases y refinamiento de relaciones, dando mayor relevancia a este último porque a partir de su profundización se agregan más relaciones y clases. Las reglas de refinamiento se basan en la definición de estereotipos para las relaciones de generalización, asociación bidireccional y asociación unidireccional, entre las clases presentes en un modelo cercano al modelo del dominio.

El UML-RT (UML para tiempo real), presentado en Sampaio, Mota y Ramos (2003), incluye tres constructores nuevos utilizados mediante la figura de estereotipos: «capsule» y «protocol» como estereotipos de clases y «connector» como prototipo de asociación. Las cápsulas describen clases activas posiblemente complejas que pueden interactuar con su entorno a través de mensajes y que pueden ser compuestas; los protocolos definen la posible manera única en que una cápsula puede interactuar con su entorno, mediante servicios de una interfaz y los conectores se usan para interconectar cápsulas. Para definir el refinamiento se emplean algunas reglas que realizan la transformación de diagramas de clases y cápsulas para obtener de ellos extracción de clases, descomposición de cápsulas, refinamiento de datos y herencias de comportamientos; este refinamiento se realiza con un lenguaje muy cercano al nivel de implementación, y no se contempla para este proceso la adición de nuevos métodos en las clases que no puedan ser derivados de las relaciones de generalización y estereotipos ya presentes en el modelo.

Otros formalismos definen, a través de la teoría de conjuntos y la lógica de predicados, la forma en que los diagramas de clases pueden ser refinados bajo el con-

cepto de poder adicionar más detalles como atributos y asociaciones; en este trabajo se realiza el refinamiento del diagrama de clases combinándolo con el de los casos de uso y empleando para ello el denominado cálculo de refinamiento [véase Back, Petre y Paltor (1999)]. El concepto de refinamiento manejado en el contexto de este artículo se apoya más en la transformación de esquemas conceptuales en especificaciones de diseño, que buscan, por ejemplo, un refinamiento del diagrama de clases para atender las necesidades del diseñador y permitir que se pueda realizar posteriormente una transición fácil hacia el código. En este sentido, se trata de un tipo de refinamiento más enfocado a ir develando la estructura de los métodos y su forma de programarlos, que a la completitud de los mismos.

De manera similar, en Arango y Jiménez (1997) se propone la descomposición de un sistema en un conjunto de operaciones que realizan modificaciones a las clases existentes adicionando operaciones que apoyan la funcionalidad de los atributos. Para esto, se definió una lógica formal donde se expresan transacciones en función de las clases, sus relaciones y sus atributos, combinadas con una notación objetual que expresa la pertenencia de los atributos a las clases y sus restricciones (de la forma "clase.atributo").

El cálculo orientado a objetos denominado "join calculus" en Fournet, Laneve, Maranget y Remy (2000) se usa también de manera muy cercana a los lenguajes de programación, específicamente para realizar una forma de refinamiento que busca la sincronización de la herencia entre clases, es decir, tomar en consideración las relaciones de herencia y verificar que los métodos de las clases hijas invoquen de manera correcta los métodos correspondientes en las clases padres. Esto lo realiza mediante un operador de refinamiento selectivo que se aplica a la clase padre y reescribe las reglas de reacción del padre de acuerdo con sus patrones de sincronización. Este, se ocupa de lenguajes muy cercanos a la implementación, que parten de diagramas de clases próximos al diseño.

En la siguiente sección se compendian los principales problemas de estas propuestas en relación con el refinamiento de los atributos derivados del diagrama de clases, que se pueden identificar en las interacciones interesado-sistema.

3 PROBLEMAS ASOCIADOS CON EL REFINAMIENTO DEL DIAGRAMA DE CLASES

Los artículos tratados en la sección anterior concentran el proceso de refinamiento en la generación de nuevos elementos en el diagrama de clases, pero sólo alrededor de las clases y las relaciones. Se parte de un modelo incompleto para el diseño, que se confronta mediante las reglas de refinamiento definidas en cada caso, como son los diferentes tipos de asociación y la creación de nuevos elementos, entre otros. En el modelo faltan elementos, que pueden originarse a partir del análisis de los casos de uso, en especial en las interacciones interesado-sistema, que involucran las clases del diagrama; algunos de estos elementos son los atributos derivados.

Además, los artículos que emplean lenguajes matemáticos o de programación, como Back et al. (1999), D'Souza y Wills (1998), Fournet et al. (2000) y Sampaio et al. (2003), aplican las reglas de refinamiento en las etapas finales del desarrollo, pero no permiten realizar el análisis de las posibles variaciones que incluyan modificaciones en el modelo. En Sampaio et al. (2003) y D'Souza y Wills (1998), se confía más en la expresión de las reglas de refinamiento en lenguajes como OhCircus y C^{++} con OCL, próximos al lenguaje de implementación con el fin de acercarse a la codificación de las clases y su refinamiento. Sin embargo, UML permite la descripción de este tipo de restricciones usando cualquier tipo de lenguaje, siempre y cuando se coloque entre llaves ({}); es posible, entonces, usar lenguaje natural, lógica de predicados u OCL como forma de expresión de las restricciones, pero el uso de OCL puede introducir en el modelo el riesgo de malas interpretaciones debido a que no todos los analistas y/o diseñadores manejan adecuadamente este lenguaje. En este sentido, es mucho más conveniente usar lenguajes más cercanos al lenguaje natural o, como en el caso del join calculus.

Con base en estas carencias se propone un método para obtener un diagrama de clases refinado que permita adicionar atributos derivados mediante la inspección de las interacciones interesado-sistema que se identifican en los casos de uso durante la etapa de análisis, y posteriormente transformarlos en operaciones en la etapa de diseño. Tal método se propone en la sección siguiente.

4 MÉTODO DE REFINAMIENTO DE LOS ATRIBUTOS DERIVADOS DEL DIAGRAMA DE CLASES

En el desarrollo de una pieza de software, durante la fase de análisis se realiza un examen minucioso de los problemas de la organización, su modo de proceder y los objetivos ligados con cada proceso, con el fin de definir una nueva forma de proceder apoyada en una solución informática; la descripción de esa solución se realiza a través de Diagramas de Casos de Uso, que se materializan en interacciones interesado-sistema. La construcción de los Diagramas de Clases, en esta etapa, tiene una dependencia directa con los Casos de Uso, puesto que los datos que se reflejan en las interacciones deben ser atendidos totalmente por los Diagramas de Clases. Ahora, en las interacciones, expresadas por formas de diálogo que simulan lo que será la interacción futura entre el interesado y el sistema (si bien en esta etapa el software como

tal aún no existe), se pueden encontrar ciertos campos que no se pueden obtener por digitación o selección directa del interesado en el momento de usar las formas de diálogo. Estos elementos están ligados con los atributos derivados que se encuentran en el Diagrama de Clases, y de ellos se suministra, por lo general en un lenguaje natural o lógico, una fórmula que permite su cálculo. Ciertos datos como los promedios de ventas, la nota definitiva de un estudiante o el costo total de un elemento producido hacen parte de esta categoría.

En Booch et al. (1997) se reconoce que los atributos derivados son importantes en la conceptualización de los modelos, pero que paulatinamente se vuelven inconvenientes al avanzar el ciclo de desarrollo del software, debido a que no agregan información semántica; en efecto, no se trata de datos "nuevos" sino de ciertas formas de visualización de los datos existentes. Para que el comportamiento de este tipo de atributos no se delegue a los programadores en la etapa de construcción de la pieza de software, es importante realizar su refinamiento hacia elementos estructurales que sí hagan parte de la conceptualización del modelo, como son las operaciones del diagrama de clases.

El método propuesto establece reglas de refinamiento específicamente sobre atributos derivados de diferentes clases involucradas en una interacción expresada en los casos de uso, para posteriormente, en la etapa de diseño ser convertidos en operaciones de las clases, garantizando que los cambios que se realicen sobre las operaciones por alteración de las reglas del negocio no afecten las otras clases. Esta forma de proceder puede facilitar la construcción de otros diagramas más cercanos a la implementación, tales como los diagramas de secuencias, donde se detallan de manera secuencial los casos de uso.

Para especificar las reglas de refinamiento se utiliza un lenguaje formal que resulta interpretable, eficaz y fácil de traducir a cualquier lenguaje Orientado a Objetos, como una transición que se podría usar posteriormente hacia OCL o incluso hacia el lenguaje de programación seleccionado. Para elaborar las reglas de refinamiento, se deben identificar: Los casos de uso y sus interacciones, las clases y sus atributos derivados y el grado de composición y anidamiento de las operaciones asociadas al cada caso de uso. UML identifica los atributos derivados como se muestra en la Figura 1.

Se formalizan estos elementos de la siguiente manera:

$$\textbf{atributos(C)} = \text{el conjunto de } [< a1 : \mathbf{T}1 >; ...; < am : \mathbf{T}m >] \text{ atributos} \quad (1)$$

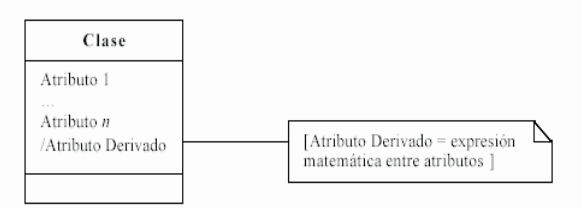

Figura 1: Definición del atributo derivado en UML

La extensión del conjunto **atributos** para la clase **C** incluyendo atributos derivados se expresa como:

$$\textbf{atributos(C)'} = \text{el conjunto de } [< a1 : \mathbf{T}1 >; ...; < am : \mathbf{T}m, \textbf{atributo\_derivado}1 : \mathbf{T}1 >; ...; < \textbf{atributo\_derivado}m : \mathbf{T}n >] \text{ atributos} \quad (2)$$

En (1) y (2): **C** es una clase genérica, $a1, \cdots, am$ son atributos y $T1, \cdots, Tm$ son tipos de datos.

Se declara un elemento llamado **atributo_derivado**, el cual se define como el conjunto de todas las posibles operaciones matemáticas que se pueden conjugar con atributos simples de la misma clase o de otras clases. Así, se conforma la fórmula genérica requerida para un atributo derivado. Dichas operaciones se definen a partir del análisis realizado sobre las interacciones interesado-sistema expresadas en el caso de uso, que a su vez involucra diferentes clases que hacen parte del diagrama de clases que representa el dominio. Se propone la siguiente expresión formal para el atributo derivado, tomando algunos principios de Arango y Jiménez (1997):

$$\begin{array}{lcl} \textit{Atributo\_derivado} & \leftarrow & \sum \textit{ factor atributo\_derivado} \\ & | & \prod \textit{ factor atributo\_derivado} \\ & | & \textit{factor} \\ & | & \phi \\ \textit{factor} & \leftarrow & \textit{término + factor} \\ & | & \textit{término - factor} \\ & | & \textit{término} \\ \textit{término} & \leftarrow & \textit{elemento * término} \\ & | & \textit{elemento / término} \\ & | & \textit{elemento} \\ \textit{elemento} & \leftarrow & \textit{clase.atributo} \end{array} \quad (3)$$

Para (3) se define:

- Los operadores $\sum$ y $\prod$ utilizan una variable que pertenece a una clase invocada por la interacción.
- Elemento: atributo perteneciente a una clase.
- Término y Factor: conjuntos de operaciones de carácter recursivo entre elementos.

Una vez formalizados los atributos derivados involucrados en el Caso de Uso, se procede a identificar cuáles

son los elementos participantes de una clase en la fórmula expresada para el atributo derivado y que pueden transformarse en operaciones propias de esa clase; tales elementos están asociados con las variables de pertenencia de los operadores $\sum$ y $\prod$. Las "nuevas" operaciones se expresan de la siguiente manera:

$$\begin{aligned}&\textbf{Operación\_atributo\_derivado} = \\ &\quad f(atributo\_1; :::; atributo\_n), \\ &\quad \text{donde } atributo\_1, \cdots, atributo\_n \\ &\quad \in \text{conjunto de clases relacionadas } [C1; :::; Cm]\end{aligned} \tag{4}$$

En la etapa de diseño, las clases que involucraban atributos derivados se transforman como se muestra en la Figura 2:

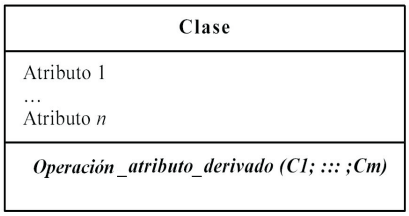

Figura 2: Diagrama de Clases para el manejo de solicitudes de mantenimiento

En la sección siguiente se muestra un caso que aplica el método propuesto.

5 CASO ESTUDIO

El caso de estudio se fundamenta en un sistema de información para manejar solicitudes de Mantenimiento de Edificios. Para este sistema un usuario solicita un servicio de mantenimiento vía telefónica; una vez se le atiende la llamada, se crea una solicitud que se alimentará posteriormente con los diferentes servicios que ésta conlleve. Dichos servicios serán realizados mediante visitas de los empleados a los usuarios, las cuales son reportadas con su duración real. Para cada servicio se toman en consideración los diferentes reportes y los materiales utilizados.

Aplicando el método propuesto para este diagrama de clases, se define el caso de uso: "***Costear una solicitud***", el cual implica conocer el costo de todos y cada uno de los ***Servicios*** que se realizaron para ésta, que se descomponen en los ***Reportes*** de cada uno de los ***Empleados*** y el costo de los ***Materiales utilizados***. En la Figura 3 se puede apreciar la forma de diálogo asociada con el caso de uso explicitado. Allí se puede apreciar el atributo derivado "Valor Total", que se reconoce por ser un valor no editable que debería calcular el sistema de manera automática. Parte del Diagrama de Clases se muestra en la Figura 4.

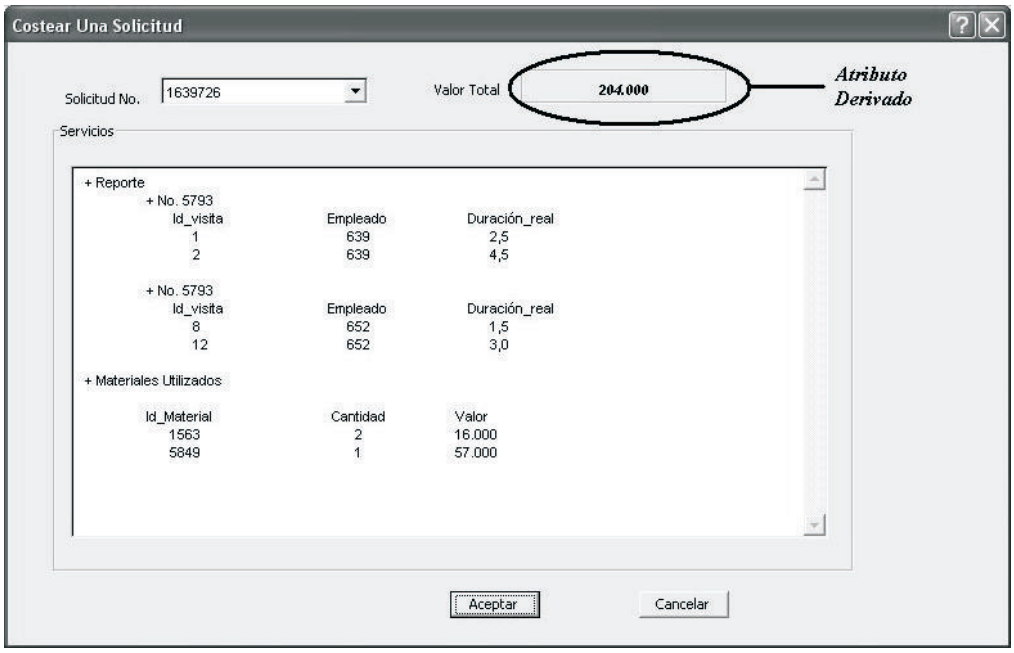

Figura 3: Interacción del Caso de Uso "Costear una Solicitud"

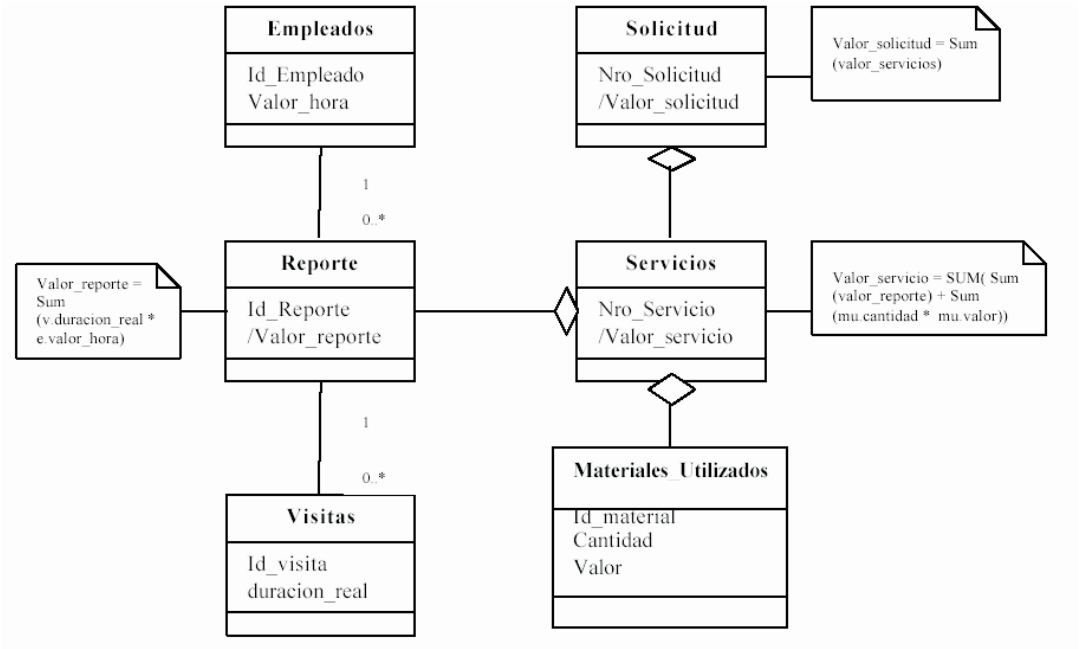

Figura 4: Diagrama de Clases para el manejo de Solicitudes de mantenimiento

En la Figura 4 se definen las fórmulas asociadas a cada atributo derivado, las cuales se formalizan empleando la formulación (3). Como ejemplo se presenta la formulación para /Valor_solicitud:

$$\textbf{Valor\_solicitud.Solicitud} = \sum S \in Servicios(\sum R \in S.Reporte * (\sum V \in R.visitas * (V.duracion\_real * Empleados.valor\_hora)) + \sum M \in S.Materiales\_utilizados * (M.cantidad * M.valor)) \quad (5)$$

Con base en esto se crean tres nuevas operaciones en las clases que desde el análisis se identificaron como participantes en el caso de uso: Reporte, Servicios y Solicitud. El diagrama resultante en la etapa de diseño se muestra en la Figura 5.

En la etapa de diseño se ha complementado el refinamiento de las clases tomando en consideración los atributos derivados de la etapa de análisis, de tal forma que se entregan nuevos elementos para la etapa de implementación, incluyendo las formulaciones correspondientes a las operaciones, y suministrándole una mayor calidad al modelo sin necesidad de modificar las clases y las relaciones entre ellas.

6 CONCLUSIONES

En este artículo se propuso un método para el refinamiento de los atributos derivados de un diagrama de clases, transformándolos en expresiones formales de operaciones de las clases participantes. Se definió para ello una sintaxis basada en lógica formal que puede ser más entendible que el OCL, pero que puede permitir una migración hacia este lenguaje o incluso hacia lenguajes de programación de las diferentes soluciones de software.

Los atributos derivados se constituyen en problemas de diseño, si bien se debe realizar su formulación lo más pronto posible en el ciclo de desarrollo del software. La expresión de los atributos derivados en un lenguaje formal posibilita la identificación de las fórmulas que deberán ser codificadas cuando se desarrollen los métodos que los reemplazan. Esto permite la identificación de piezas de código desde el principio de la fase de diseño, lo cual podría contribuir a la detección de problemas antes de pasar a la codificación. Además, el método planteado permite hacer una transición casi automática desde la formulación en lógica de los atributos derivados hasta los lenguajes de programación, si bien éstos también se basan en formas lógicas.

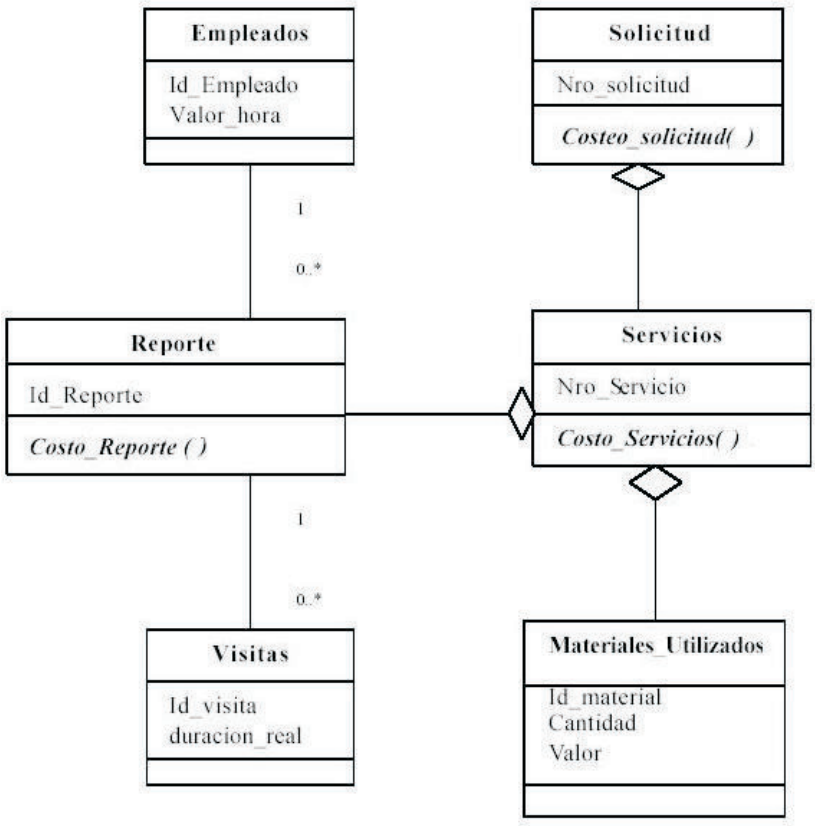

Figura 5: Nuevo diagrama de clases que sustituye los atributos derivados por métodos de las clases

El análisis realizado sobre los atributos derivados puede aportar directamente al refinamiento de otros diagramas de UML. En particular, los cambios y consultas que se realizan sobre cada una de las clases posibilitan la identificación de los elementos del diagrama de comunicación, de lo cual podría desprenderse un método para su trazado a partir del método propuesto.

Otros trabajos futuros alrededor del método propuesto incluyen:

- La expresión formal de las pre y postcondiciones de los eventos formulados a través de los casos de uso, los cuales generan cambios en los datos persistentes de la base de datos.
- Se puede explorar en el futuro la manera como este método podría impactar en la elaboración de los diagramas de secuencias, en especial en el pseudocódigo que tienen asociado.
- Otros elementos como reglas del negocio se podrían formular desde el principio en una forma lógica, facilitando la migración de estos elementos y garantizando la trazabilidad de tales elementos hacia el código.

REFERENCIAS

Arango, F. y Jiménez, C. (1997), Metodología para la evolución de software de 3^a generación, Technical report, Universidad Nacional de Colombia, Sede Medellín, Postgrado en Ingeniería de Sistemas. Informe final proyecto COLCIENCIAS-UN, código 1118-14-006-93. 261 p.

Back, R., Petre, L. y Paltor, I. (1999), Formalizing UML use cases in the refinement calculus, Technical Report 279, Turku Centre for Computer Science, Turku, Finland. En: <http://www.tucs.fi/publications/techreports/· · · · · ·/TR279.php>.

Booch, G., Jacobson, I. y Rumbaugh, J. (1997), *The* OMG *Unified Modeling Language Specification*, Rational Software Corp. 566 p.

D'Souza, D. y Wills, A. (1998), *Catalysis: Objects, Frameworks and Components with* UML, Addis on-Wesley.

Fournet, C., Laneve, C., Maranget, L. y Remy, D. (2000), Inheritance in the join calculus, *in* 'Foundations of Software Technology and Theoretical Computer Science (FSTTCS2000)', New Delhi, pp. 397–408.

France, R. y Bieman, J. (2001), Multi-view software evolution: a UML-based framework for evolving object-oriented software, *in* 'Proceedings of the International Conference on Software Maintenance (ICSM 2001)', p. 10.

Larman, C. (2003), UML *y patrones*, Prentice Hall, Segunda Edición, Madrid.

Sampaio, A., Mota, A. y Ramos, R. (2003), Class and capsule refinement in uml for real time, *in* 'Workshop de Métodos Formais'.

Shen, W., Lu, Y. y Low, W. (2003), Extending the UML metamodel to support software refinement, *in* 'Proceedings of the Workshop on Consistency Problems in UML - based Software Development II', San Francisco, pp. 35–42.

UML (2005), 'Unified modeling language uml. resource page'. En Línea: <http://www.uml.org> C-03/05.