Rediseño del meta modelo de OCL aplicando patrones GoF

Juan Martín Chiaradía , Claudia Pons

LIFIA – Facultad de Informática, Universidad Nacional de La Plata
La Plata, Buenos Aires, Argentina

jmchiara@sol.info.unlp.edu.ar

Resumen. OCL es un lenguaje de especificación estándar, el cual probablemente será soportado por la mayoría de las herramientas de modelado de software. Por esto es importante que OCL posea una base formal sólida tanto en la definición de su sintaxis como en la de su semántica. Actualmente OCL esta formalizado mediante meta modelos expresados en MOF, complementados con reglas de buena formación escritas en el propio OCL. Por otra parte el meta modelo de OCL presentan falencias en su calidad debido a que no respeta ciertos patrones del diseño orientado a objetos. En este articulo presentamos una nueva definición para este meta modelo, aplicando patrones GoF y la técnica de Dynamic Meta Modeling, evitando la circularidad en la definición de OCL e incrementando su legibilidad y precisión. Esta nueva definición está siendo utilizada en el desarrollo de herramientas de soporte para las actividades de modelado de software.

Keywords: Ocl; Semántica formal de Ocl; Dynamic Metamodeling; E-Platero.

1 Introducción

El OCL (Object Constraint Language) [8] es un lenguaje de especificación formal, fácil de leer y escribir, aceptado como estándar por el OMG (Object Management Group). OCL permite definir restricciones sintácticas y semánticas sobre modelos expresados en notaciones graficas tales como UML[13], ampliando de esta forma el poder expresivo de dichas notaciones. De esta forma, los diagramas complementados con expresiones OCL son más precisos y completos.
Tanto UML como OCL están definidos a través de MOF (Meta Object Facility) [6], el cual es un meta-lenguaje estandarizado por OMG cuya finalidad es permitir la creación de meta-modelos.
El lenguaje OCL ha sido definido formalmente a través de los siguientes documentos:

- un (meta) modelo MOF que define precisamente su sintaxis abstracta.
- un (meta) modelo MOF que describe sus dominios semánticos.
- un conjunto de clases MOF que especifican la semántica (significado) de OCL, es decir la conexión entre las construcciones sintácticas y el dominio semántico.

Es sabido que los modelos orientados a objetos, debido a su proximidad con la realidad, transmiten un significado intuitivo, fácil de percibir por sus lectores; sin embargo cuando el diseño de tales modelos no es adecuado, la intuición se pierde y los modelos se tornan difíciles de comprender. Esta situación desfavorable se observa en la especificación estándar de OCL 2.0 [8]. La causa puede deberse a la no aplicación (o aplicación inadecuada) de algunos bien conocidos patrones de diseño. Si bien en la sintaxis abstracta el resultado obtenido es claro y preciso, en la definición de la semántica se generan varios interrogantes que entorpecen el entendimiento del

---

Esta investigacion es parcialmente financiado por la Universidad Abierta Interamericana UAI, en el contexto del Proyecto " Modelado de Software: un enfoque formal" .

lenguaje. Creemos que esta falta de "autoexplicación" que se observa en la especificación estándar de OCL 2.0 [8] se debe simplemente a una elección errónea de patrones de diseño empleados en el diseño del meta modelo semántico.
Para solucionar este problema proponemos crear una definición alternativa de la semántica de OCL, que resulte más clara y simple. Para ello se aplicarán patrones GoF [4] al diseño presentado en el documento de especificación estándar [8]. Nuestra hipótesis consiste en que la aplicación de patrones contribuirá a mejorar la legibilidad y la precisión de la definición de OCL.
Para brindar un adecuado contexto a la lectura de esta propuesta, en la sección 2 mostramos, de forma resumida, la semántica actual de OCL[8]. Luego, en la sección 3 proponemos una nueva definición para la semántica de OCL, basada en la aplicación del patrón *Visitor* [4] sobre los meta modelos semánticos; utilizamos además la técnica conocida como *Dynamic Meta Modeling* (DMM) para lograr una especificación precisa de la semántica pero sin perder claridad y logrando comunicar los conceptos de manera más intuitiva. Finalmente, en la sección 4 presentamos conclusiones y trabajos futuros.

2 Sinopsis de la Especificación Estándar

Una expresión OCL es definida en la especificación estándar [8] como una expresión que puede ser evaluada en un entorno dado; dicha evaluación produce un valor. De esta manera, dar semántica a una expresión sintáctica no es otra cosa que asociarla, en un entorno o ambiente dado, con un valor correspondiente en el dominio semántico del lenguaje. Con este propósito, en el documento estándar [8] se propone la estructura ilustrada en la figura 1.

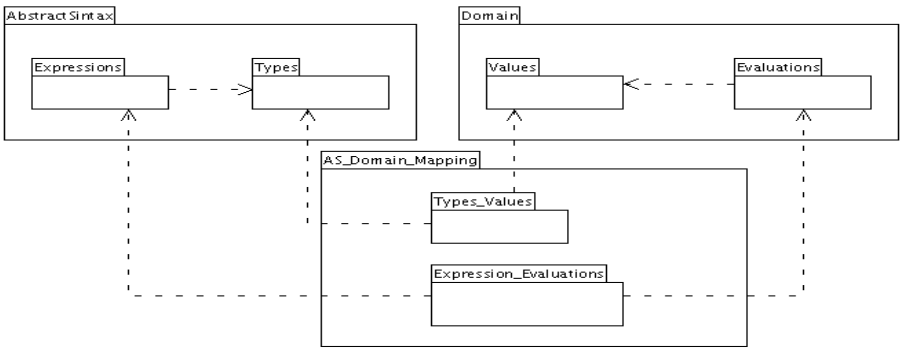

Figura 1: Meta modelos de los dominios sintáctico y semántico.

Donde:

- *AbstractSyntax* describe la sintaxis abstracta del lenguaje OCL.

- *Domain* describe los valores semánticos y las evaluaciones:
a) *Values* describe el dominio semántico de OCL. Muestras los posibles valores que pueden encapsular las expresiones como resultado.
b) *Evaluations* describe las diferentes evaluaciones de las expresiones OCL. Este paquete contiene las reglas para determinar el valor de una expresión dada.

- *AS-Domain-Mapping* describe las asociaciones de los valores y evaluaciones con los elementos de la sintaxis abstracta (AbstractSyntax):
c) *Type-Value* contiene las asociaciones entre las instancias en el dominio semántico y los tipos en la sintaxis abstracta.

d)*Expression-Evaluation* contiene las asociaciones entre las clases de evaluaciones y las expresiones en la sintaxis abstracta.

La figura 2 muestra la definición de la sintaxis abstracta de las expresiones OCL a través de una jerarquía de meta clases. El paquete *Evaluations* define la semántica de dichas expresiones utilizando también una jerarquía de meta clases representantes de cada evaluación particular (ver figura 3). De esta manera, cada evaluación tiene un valor resultado para un determinado ambiente, y asociando cada instancia de evaluación a un modelo de expresión, se establece la evaluación semántica de una expresión en un determinado ambiente (ver figura 4).

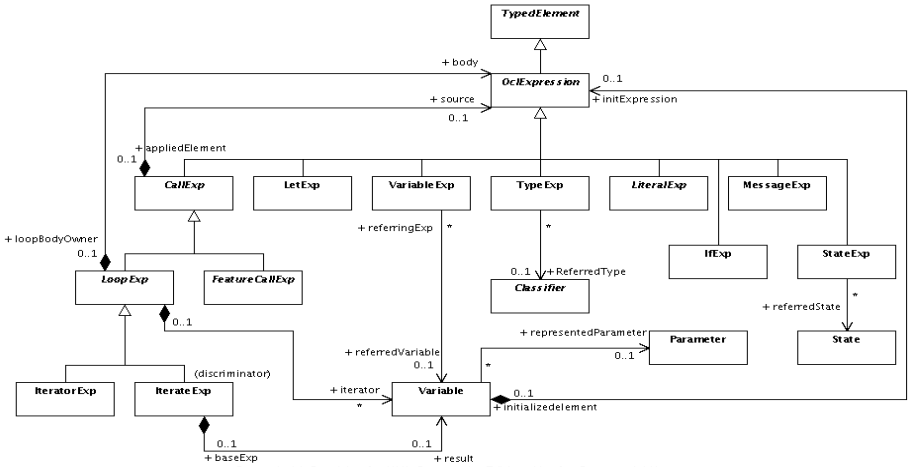

Figura 2: Overview del paquete AbstractSyntax

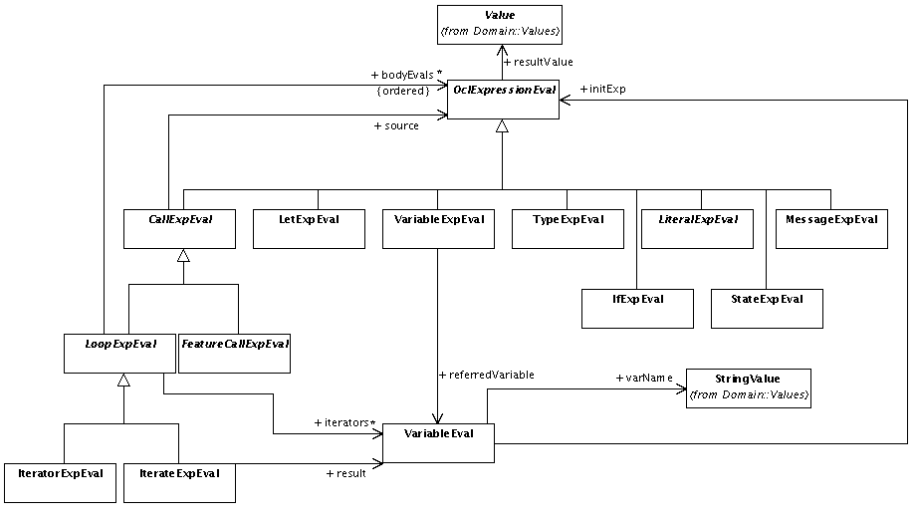

Figura 3: Overview del paquete Evaluations

Puede observarse que las evaluaciones replican la estructura jerárquica de la sintaxis abstracta. Creemos que esta duplicación de jerarquías es innecesaria y origina inconvenientes tales como reducción en la legibilidad del meta modelo e ineficiencia en las herramientas automáticas que utilizan esta semántica como fundamento. Entraremos en detalle sobre estos temas en la siguiente sección.

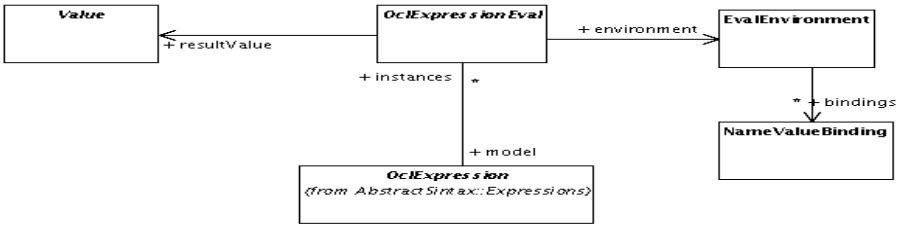

Figura 4: Evaluación semántica de una expresión.

3 Aplicando el Patrón "*Visitor*" y *DMM* para la Evaluación Semántica

En esta sección definiremos la meta clase *OclEvaluator* que dará significado semántico a nuestras expresiones sintácticas asociándolas a su valor correspondiente, funcionando así como un puente entre la Sintaxis Abstracta y el Dominio Semántico. De esta manera, la figura 5 muestra la nueva propuesta para el meta modelo de OCL.
El patrón Visitor [2] representa una operación ejecutada sobre los elementos de una estructura bien definida. Este patrón nos permite definir nuevas operaciones sin interferir con la definición de la estructura sobre la que opera. La aplicación del patrón Visitor es apropiada en situaciones donde la estructura de objetos definida no variará, pero , pero pueden aparecer nuevas operaciones sobre esta
Para evaluar cada expresión, el *OclEvaluator* utiliza un ambiente de evaluación llamado *EvalEnvironment* siguiendo la estrategia clásica que se utiliza para la definición de la semántica de lenguajes de programación (por ejemplo las definiciones dadas en [3] y [10]). Aparte de dicho ambiente, la evaluación de una expresión depende sólo de su estructura sintáctica. Esta estructura sintáctica raramente será modificada y varias operaciones pueden definirse en su entorno (Ej.: operaciones de refactorización, evaluación semántica, generación de código, etc.). Por esto consideramos que es más apropiado mantener limpia, clara y simple la estructura y aplicar el patrón Visitor para definir operaciones como la evaluación semántica. Esta nueva estructura de evaluación se refleja en la figura 6.

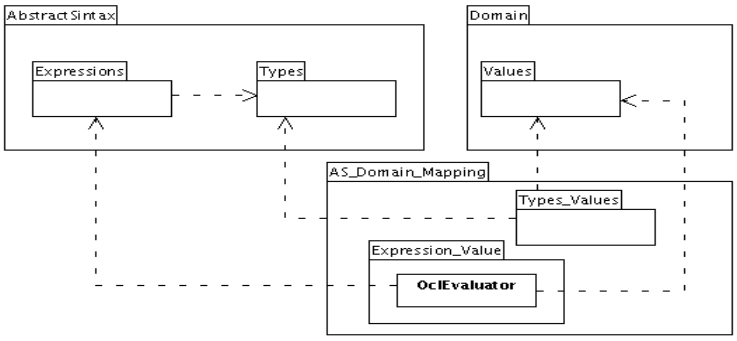

Figura 5: meta modelo de OCL aplicando el patrón Visitor

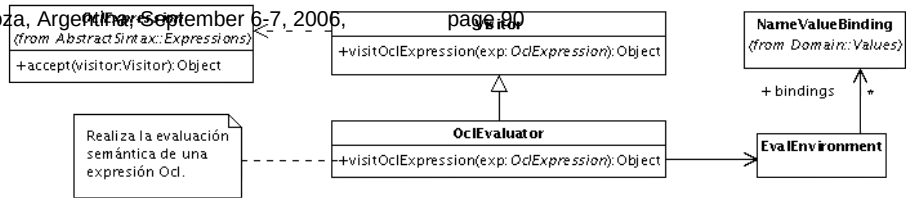

Figura 6: Modelo de evaluación semántica utilizando el patrón Visitor

Por otro lado, creemos que la mejor forma de entender la semántica de una evaluación es mostrando el proceso de evaluación utilizado. Al usar solamente diagramas de clases para reflejar la semántica, no se ve claramente el proceso antes mencionado debido a la naturaleza estática inherente de estos diagramas. Así, para entender todo el proceso es necesario observar las restricciones planteadas sobre el diagrama de clases. Estas restricciones se encuentran escritas en OCL. Esto trae dos consecuencias negativas:

- se pierde el sentido de la utilización de UML en el modelado: mostrar la semántica de forma simple y auto explicativa, sin necesidad de ver la base matemática; ya que es necesario entender las restricciones para comprender la semántica.
- se esta definiendo la semántica de OCL en términos de OCL!. Alguien que no comprende OCL, tampoco lo comprenderá viendo estas restricciones.

Por lo anterior, a lo largo de esta sección utilizaremos distintos diagramas de secuencia para visualizar los distintos pasos seguidos en la evaluación semántica de las expresiones, esperando así lograr una explicación clara y precisa. Esta metodología para dar semántica a lenguajes de modelado se conoce como *Dynamic Meta Modeling* (DMM) [2] [5], ha sido utilizada para dar semántica a otros elementos de UML (tales como State Machines y Collaborations), pero hasta el momento no había sido aplicada a OCL.

3.1 Un Ejemplo: Semántica de la Expresión *LetExp*

La evaluación de una *LetExp* propuesta en [8] se muestra en la figura 7. Puede observarse como la evaluación es la que encapsula tanto el valor resultado como el ambiente de evaluación; asimismo no termina de apreciarse bien cuál es el método de evaluación, como tampoco cuestiones estructurales sobre este diagrama.

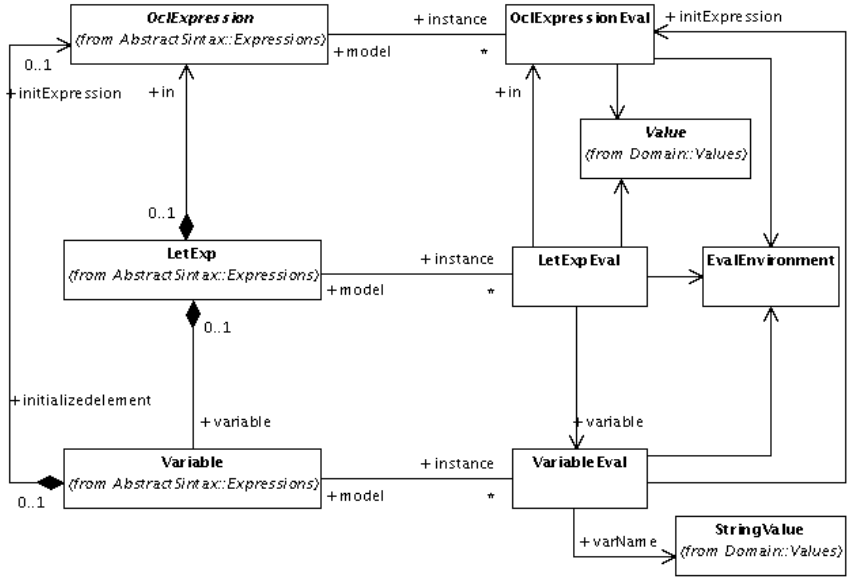

Figura 7: Evaluación de LetExp planteada en la especificación estándar

En si, con un estudio simple del diagrama mostrado en la figura 7 no obtenemos demasiada información sobre la evaluación semántica. Para comprender de forma total el diagrama anterior, tenemos que estudiar las reglas de buena formación que se incluyen en [8]:

- Todas las partes de una *LetExpEval* se corresponden con las partes de la *LetExp* a la que está asociada:

```
context LetExpEval inv:
in.model = model.in and
initExpression.model = model.initExpression and
variable = model.variable
```

- El resultado de una *LetExpEval* es el resultado de la *inExpEval* asociada:

```
context LetExpEval inv: resultValue = in.resultValue
```

- Una *LetExpEval* agrega una nueva variable al ambiente de la *inExpEval* asignada al resultado de la *InitExpEval*:

```
context LetExpEval inv:
in.environment = self.environment->add(
NameValueBinding(variable.varName,variable.initExpression.re
sultValue))
```

- El ambiente de la *initExpEval* es el mismo que el de la *LetExpEval*:

```
context LetExpEval inv: initExpression.environment =
self.environment
```

El problema de las restricciones anteriores es que se encuentran escritas en OCL, si bien son simples y relativamente fáciles de comprender, son un obstáculo para aquellos que están dando sus primeros pasos en OCL.
Teniendo en cuenta la semántica matemática expresada en [7] apéndice A, mostrada en la figura 8, estamos en condiciones de reescribir este algoritmo matemático utilizando diagramas de secuencia, siguiendo el orden aplicativo para realizar la evaluación anterior (ver figura 9).

El contexto para una evaluación esta dado por un ambiente $\tau = (\sigma, \beta)$ donde: σ es un estatdo del sistema que provee acceso al conjunto de los existentes, los valores de sus atributos y sus links con otros objetos; β es una asignación de variables $\beta: Var_t \rightarrow I(t)$ que mapea nombre de variables con valores del dominio semántico.
Luego, sea *Env* el conjunto de ambientes $\tau = (\sigma, \beta)$. La semántica de una *LetExp* viene dada por una función: $I[e]: Env \rightarrow I(t)$ y esta definida de la siguiente manera:

$$I[\![\text{let } v = e_1 \text{ in } e_2]\!](\tau) = I[\![e_2]\!](\sigma, \beta\{v/I[\![e_1]\!](\tau)\}).$$

Figura 8: Semántica matemática de una *LetExp*

Lo que primero haremos es evaluar la expresión inicial ($I[\![e_1]\!](\tau)$, señales 4 y 5 de la figura 9) para extender el ambiente con esta evaluación ($\beta\{v/I[\![e_1]\!](\tau)\}$, señales 6, 7 y 8 de la figura 9). Por último evaluaremos la expresión *in* con el ambiente extendido, siendo el resultado de esta expresión el resultado de la evaluación de la *LetExp* ($I[\![e_2]\!](\sigma, \beta\{v/I[\![e_1]\!](\tau)\})$, señales 9, 10 y 11 de la figura 9). Debemos resaltar que para evitar que las modificaciones en el ambiente se propaguen fuera de

la *LetExp* lo guardamos al inicio y para recuperarlo al final del proceso de evaluación (señales 2 y 12 de la figura 9).

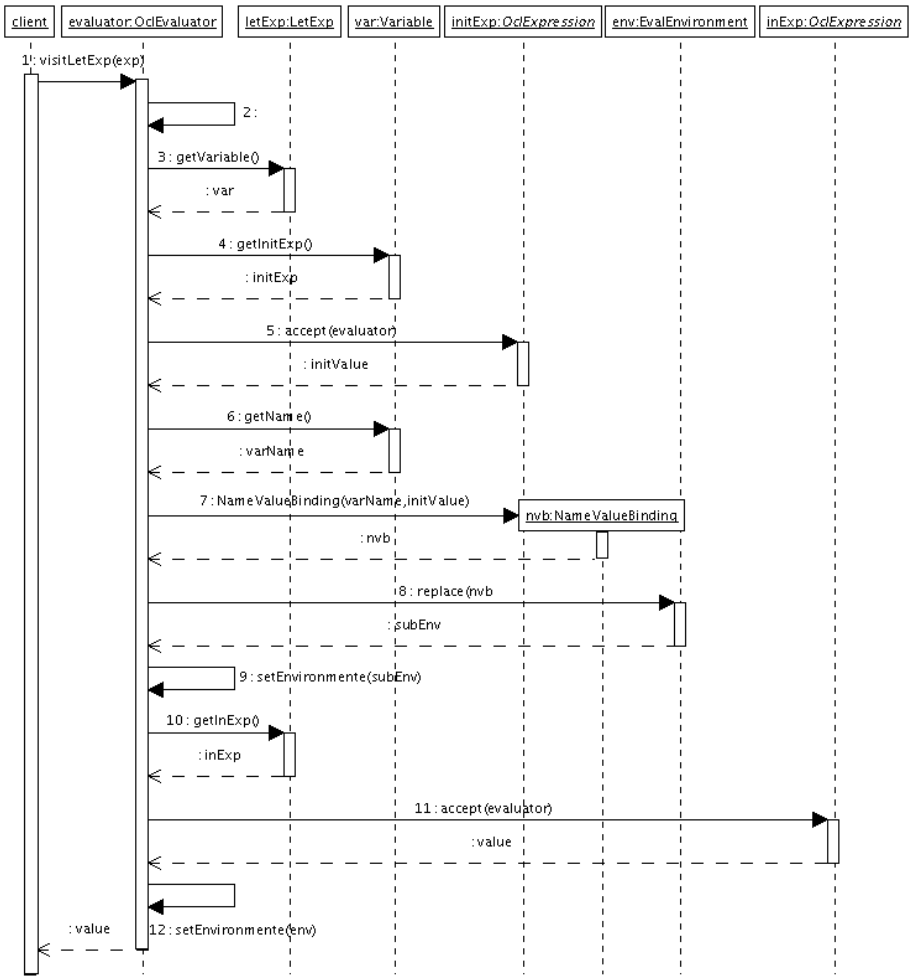

Figura 9: Evaluación de LetExp usando el evaluador planteado

3.2 Otro Ejemplo: Semántica de la Expresión *IterateExp*

La evaluación de una *IterateExp* propuesta en [8] se muestra en la figura 10.

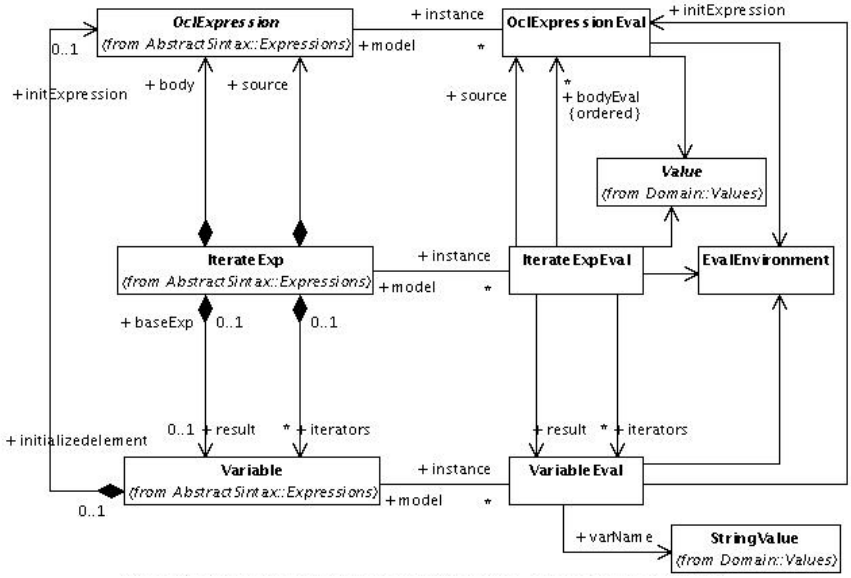

Figura 10: Evaluación de IterateExp planteada en la especificación estándar

De forma análoga con la *LetExp*, el diagrama de clases no nos ofrece mucha información sobre la semántica de una *IterateExp*. Una vez más debemos recurrir a las reglas OCL planteadas sobre el diagrama para lograr comprender en profundidad como se asocia una *IterateExp* con su valor en el dominio semántico:

- El ambiente de evaluación de una *IterateExp* es igual al del source:

```
context IterateExpEval inv: source.environment =
self.environment
```

- El source de la evaluación de una *IterateExp* se corresponde con el source de su expresión asociada:

```
context IterateExpEval inv: source.model = model.source
```

- Habrá una *OclExpEval* (una sub evaluación) para cada combinación de valores de los iteradores. Cada iterador tomará los valores de cada elemento del source:

```
context IterateExpEval inv:
bodyEvals->size() =
if iterators->size()=1
then source.value->size()
else source.value->size()*source.value->size()
endif
```

- Todas las sub evaluaciones tienen el mismo modelo, que es el cuerpo de la *IterateExp* asociada:

```
context IterateExpEval  inv:
bodyEvals->forAll(model = self.model.body)
```

- Todas las sub evaluaciones tienen diferentes ambientes. La primer sub evaluación comienza con un ambiente en el cual todos los iteradores se encuentran ligados al primer elemento del source, y donde la variable resultado se encuentra ligada al valor de la initExp correspondiente:

```
context IterateExpEval inv:
```

```
let bindings: Sequence(NameValueBindings)= iterators-
>collect(i |NameValueBinding(i.varName,source->asSequence()-
>first())

in     bodyEvals->at(1).environment = self.environment-
>addAll(bindings)->add(
NameValueBinding(result.name,result.initExp.resultValue))
```

- El ambiente de las demás sub evaluaciones es el mismo al de la sub evaluación anterior, reemplazando los iteradores por la próxima combinación de valores del source, y la variable resultado ligada al resultado de la sub evaluación anterior:

```
context IterateExpEval inv:

let SS: Integer = source.value->size()

in if iterators->size() = 1

then Sequence{2..SS}->forAll(i:Integer | bodyEvals-
>at(i).environment = bodyEvals->at(i-1).environment-
>replace(NameValueBinding(iterators->at(1).varName,
source.value->asSequence()->at(i)))-
>replace(NameValueBinding(result.varName,bodyEvals->at(i-
1).resultValue )))

else -- iterators->size() = 2

Sequence{2..SS*SS}->forAll(i: Integer | bodyEvals-
>at(i).environment = bodyEvals->at(i-1).environment->replace(
NameValueBinding( iterators->at(1).varName,source-
>asSequence()->at(i.div(SS) + 1)))->replace(
NameValueBinding( iterators->at(2).varName,source.value-
>asSequence()->at(i.mod(SS))))->replace(
NameValueBinding(result.varName,bodyEvals->at(i-
1).resultValue )))

endif
```

- El resultado de una *IterateExpEval* es el resultado de la última sub evaluación:

```
context IterateExpEval inv:

resultValue = bodyEvals->last().resultValue
```

- El modelo de la variable resultado de una IterateExpEval es la variable resultado de la IterateExp asociada:

```
context IterateExpEval inv:

result.model = model.result
```

Si bien una *IterateExp* es por naturaleza más complicada que una *LetExp*, resulta muy difícil poder comprender todas las restricciones anteriores sin antes tener conocimientos mínimos de Ocl; aún teniéndolos, la lectura y comprensión de la semántica anterior no es natural.
Al igual que hicimos con la *LetExp*, centrándonos en la semántica matemática de una *IterateExp* ([7] Apendice A) mostrada en la figura 11(sólo mostramos la semántica sobre *Sequence* ya que las semánticas sobre *Bag* y *Set* son análogas) podemos traducir esta semántica a nuestro evaluador de forma casi directa (figura 12 y figura 13).

$I[\![e_1 \rightarrow \text{iterate}(v_1; v_2 = e_2 \mid e_3)]\!](\tau) = I[\![e_1 \rightarrow \text{iterate}'(v_1 \mid e_3)]\!](\tau')$ where $\tau' = (\sigma, \beta')$ and $\tau'' = (\sigma, \beta'')$ are environments with modified variable assignments

$$\beta' := \beta\{v_2 / I[\![e_2]\!](\tau)\}$$
$$\beta'' := \beta'\{v_2 / I[\![e_3]\!](\sigma, \beta'\{v_1 / x_1\})\}$$

and it

(a) If $e_1 \in \text{Expr}_{Sequence(t_1)}$ then

$$I[\![e_1 \rightarrow \text{iterate}'(v_1 \mid e_3)]\!](\tau') = \begin{cases} I[\![v_2]\!](\tau') \\ \quad \text{if } I[\![e_1]\!](\tau') = \langle\rangle, \\ I[\![\text{mkSequence}_{t_1}(x_2, \ldots, x_n) \rightarrow \text{iterate}'(v_1 \mid e_3)]\!](\tau'') \\ \quad \text{if } I[\![e_1]\!](\tau') = \langle x_1, \ldots, x_n \rangle. \end{cases}$$

Figura 11: Semántica matemática de una IterateExp

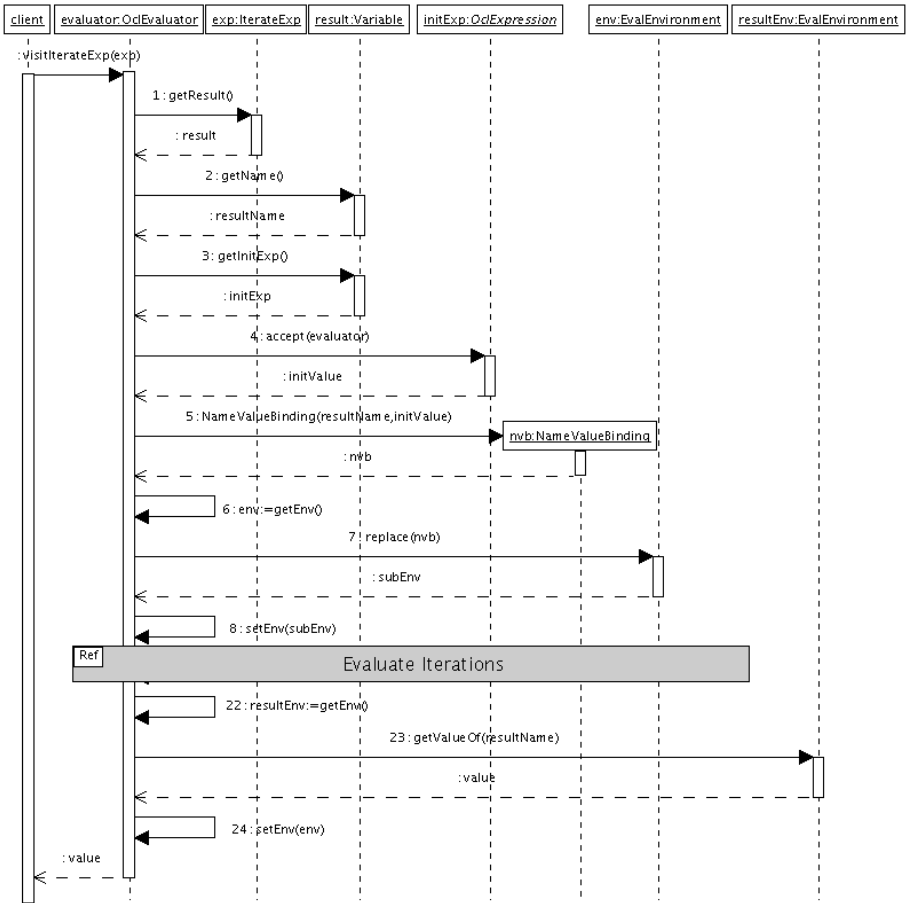

Figura 12: Evaluación de IterateExp utilizando el OclEvaluator

Una *IterateExp* se evaluará de la siguiente manera:
Para la primera iteración se agregará al ambiente la variable result con el valor inicial indicado ($\beta' := \beta\{v_2 / I[\![e_2]\!](\tau)\}$, señales 1 a 8 en la figura 12); luego procederá a evaluarse el cuerpo para las diferentes combinaciones del source (figura 11). El reemplazo de los iteradores por las diferentes combinaciones de valores de la colección fuente ($\beta'\{v_1/x_1\}$ en $\beta'' := \beta'\{v_2 / I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$) es realizada por un *CombinationGenerator* (señales 13, 14 y 15 la figura 13), siguiendo la estrategia 'depth first search'. Esta estrategia determinará la cantidad de evaluaciones realizadas sobre el cuerpo de la *IterateExp* ($I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})$ en

$\beta'' := \beta'\{v_2/I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$; señales 17 y 18 de la figura 13) , por último, las diferentes evaluaciones actualizarán la variable result ($\beta'\{v_2/I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$, señales 19, 20 y 21 de la figura 13).

Notar que una vez finalizadas las evaluaciones intermedias, y recuperado el resultado, para retornar el valor adecuado (señales 22 y 23 de la figura 12), se restaura el ambiente inicial, de modo tal que las expresiones ajenas al *IterateExp* no se vean modificadas por las alteraciones internas del ambiente.

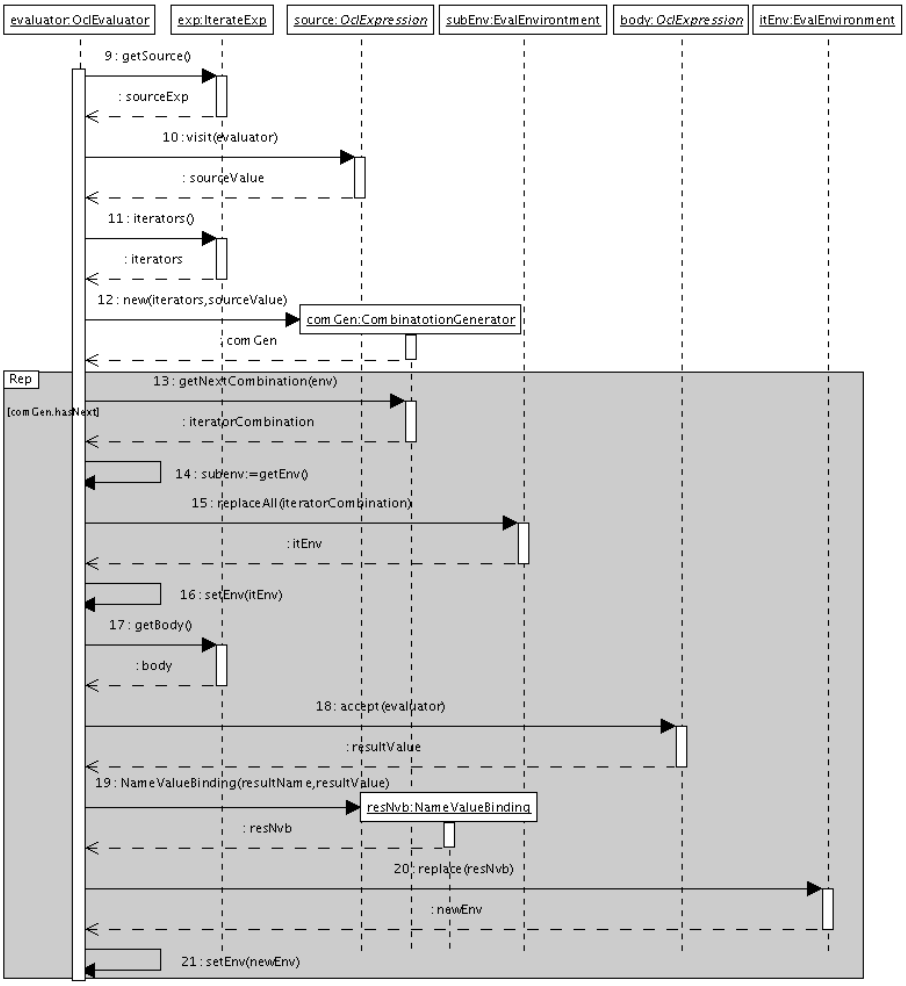

Figura 13: Evaluación del cuerpo de una IterateExp utilizando el OclEvaluator

4 Conclusión y Trabajo Futuro

OCL es un lenguaje de especificación de propiedades sobre objetos, simple y fácil de utilizar, lo que lo convierte en una opción más que interesante para el desarrollo de herramientas de verificación o derivación de código. Para explotar todo su potencial resulta indispensable que OCL posea una base formal sólida tanto en la definición de su sintaxis como en la de su semántica. Actualmente OCL esta formalizado mediante meta modelos expresados en MOF, complementados con reglas de buena formación escritas en el propio OCL. Esta recursividad en la definición no solamente acarrea

problemas formales [11] sino que obstaculiza el entendimiento del lenguaje. Por otra parte los meta modelos de OCL presentan falencias en su calidad debido a que en su construcción no se respetaron ciertas reglas (o patrones) del diseño orientado a objetos. En este trabajo hemos presentado una nueva definición para el meta modelo de OCL. La propuesta reutiliza el meta modelo de la sintaxis de OCL, rediseña el meta modelo de la semántica de OCL aplicando el patrón de diseño 'Visitor' y finalmente define la relación entre la sintaxis y la semántica mediante diagramas de secuencia de UML. De esta forma se evita la circularidad en la definición de OCL y se logra incrementar su capacidad de comunicación intuitiva.

Por otra parte, el funcionamiento adecuado de las herramientas que soportan OCL [12] [1] depende fuertemente de la calidad de la definición del lenguaje. Contar con una sintaxis y semántica bien definidas reportará en beneficios para dichas herramientas. En este sentido estamos trabajando sobre la redefinición del evaluador ePlatero[9] de acuerdo a la propuesta presentada en este articulo, con el objetivo de analizar las ventajas potenciales respecto a distintos indicadores, tales como confiabilidad, eficiencia, modificabilidad, etc.

Referencias

[1] Akehurst David: "OCL 2.0 – Implementing the Standard for Multiple Metamodels" - URL: http://www.cs.kent.ac.uk/projects/ocl/Documents/OCL%202.0%20-%20Implementing%20the%20Standard.pdf

[2] Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: "Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in uml". In Evans, A., Kent, S., Selic, B., eds.: UML 2000, York, UK, October 2-6, 2000, Proceedings. Volume 1939 of LNCS, Springer (2000) 323–337

[3] Hennessy, M.: "The Semantics of Programming Languages: An elementary introduction using structural operational semantics". J Wiley&Sons. England. 1990.

[4] Gamma, E. Helm, R. Johnson, R. and Vlissides, J.: "Design Patterns, Elements of Reusable Object-Oriented Software". Addison-Wesley Publishing Company, 1995.

[5] Hausmann, J.H.: "Dynamic Meta Modeling. A Semantics Description Technique for Visual Modeling Languages" PhD thesis, Universit¨at Paderborn, Germany (2005)

[6] Meta Object Facility MOF 2.0. OMG Adopted Specification ptc/2003-10-04. URL:www.omg.org

[7] OCL 2.0.– OMG draft Specification /ptc/03-10-14. URL: www.omg.org/docs/ptc/03-10-14.pdf

[8] OCL 2.0 Specification ptc/2005-06-06. URL:www.omg.org

[9] Pons, Claudia, R.Giandini, G. Pérez, P. Pesce, V.Becker, J. Longinotti, J.Cengia. "Precise Assistant for the Modeling Process in an Environment with Refinement Orientation" In "UML Modeling Languages and Applications: UML 2004 Satellite Activities, Revised Selected Papers". Lecture Notes in Computer Sciefnce number 3297. Springer-Verlag. Lisbon, Portugal, October 11-15, 2004. ISBN: 3-540-25081-6

[10] Reynolds, John C.: "Theories of Programming Languages" Cambridge University Press.

[11] Tchertchago Alexei: "Analysis of the Metamodel Semantics for OCL". URL: http://www.hwswworld.com/downloads/9_28_05_e/Cherchago-thesis.pdf

[12] Richters Mark and Gogolla Martin. "OCL-Syntax, Semantics and Tools" in Advances in Object Modelling with the OCL Lecture Notes in Computer Science 2263. Springer. (2001).

[13] UML 2.0. The Unified Modeling Language Superstructure version 2.0 – OMG Final Adopted Specification.. http://www.omg.org. August 2003. URL:www.omg.org