Improving the OCL Semantics Definition by Applying Dynamic Meta Modeling and Design Patterns

Juan Martín Chiaradía [1] Claudia Pons [1,2]

[1] LIFIA – Facultad de Informática, Universidad Nacional de La Plata

[2] CONICET (Consejo Nacional de Investigaciones Científicas y Técnicas)

La Plata, Buenos Aires, Argentina

{jmchiara,cpons}@sol.info.unlp.edu.ar

Abstract. OCL is a standard specification language, which will probably be supported by most software modeling tools in the near future. Hence, it is important to OCL to have a solid formal foundation, for its syntax and its semantic definition. Currently, OCL is being formalized by metamodels expressed in MOF, complemented by well formedness rules written in the own OCL. This recursive definition not only brings about formal problems, but also puts obstacles in language understanding. On the other hand, the OCL semantics metamodel presents quality weaknesses due to the fact that certain object-oriented design rules (patterns) were not obeyed in their construction. The aim of the proposal presented in this article is to improve the definition for the OCL semantics metamodel by applying GoF patterns and the dynamic metamodeling technique. Such proposal avoids circularity in OCL definition, and increases its extensibility, legibility and accuracy.

Keywords: OCL; formal semantics; dynamic meta modeling; design patterns.

1 Introduction

OCL (Object Constraint Language) [8] is a formal specification language, easy to read and write, accepted as a standard by the OMG (Object Management Group). OCL permits to define syntactic and semantic restrictions upon models expressed in graphic notations such as the UML [13], thus extending the expressive capacity of such notations. In this way, diagrams complemented by OCL expressions are more accurate and complete.

Both UML and OCL are defined by MOF (Meta Object Facility) [6], which is a meta-language maintained by OMG whose aim is to allow for metamodel creation.

The OCL language has been formally defined through the following documents:

- a MOF (meta) model that defines its abstract syntax.
- a MOF (meta) model that describes its semantic domains.
- a set of MOF classes that specify the OCL semantic (meaning), i.e. the connection between the syntactic constructions and the semantic domain.

It is known that the object-oriented models, due to their proximity to reality, transmit an intuitive meaning, easy to be perceived by their readers; however, when the design of such models is not adequate, intuition disappears, and models become difficult to understand. This unfavorable situation is observed in some parts of the OCL 2.0 standard specification [8]. The reason why this occurs may be the non-application (or inadequate application) of some well-known design patterns. Although in the abstract syntax definition the result obtained is clear and accurate, in the semantic definition several questions arise that hamper language understanding. We believe that this lack of “self- explaining” observed in [8] is due to an erroneous selection of the design patterns used in the design of the semantics metamodel.

Working towards the solution to this problem, we propose to create a clearer and simpler alternative definition for the OCL semantics. For that purpose, GoF patterns [4] will be applied to the design presented in the standard specification document [8]. Our hypothesis is that pattern application will contribute to improve legibility, extensibility and accuracy of OCL definition.

To provide an adequate context to the reading of this proposal, in Section 2 we present a summary of the current OCL semantics [8]. Then, in Section 3, we propose a new definition for the OCL semantics, based on the *Visitor* pattern application [4] upon the semantics metamodels; we also use the technique known as *Dynamic Meta Modelling* (DMM) [2] [5] to achieve an accurate specification of the semantics, but keeping clarity, and communicating concepts in a more intuitive manner. Finally, in Section 4, we present conclusions and future works.

2 OCL Specification Overview

OCL expression is defined in [8] as "an expression that can be evaluated in a given environment" and it states that “evaluation of the expression yields a value". Taking it into account, the ‘meaning’ (semantics) of an OCL expression can be defined as the value yielded by its evaluation in a given environment. In order to specify this semantics, [8] proposes the structure illustrated in figure 1.

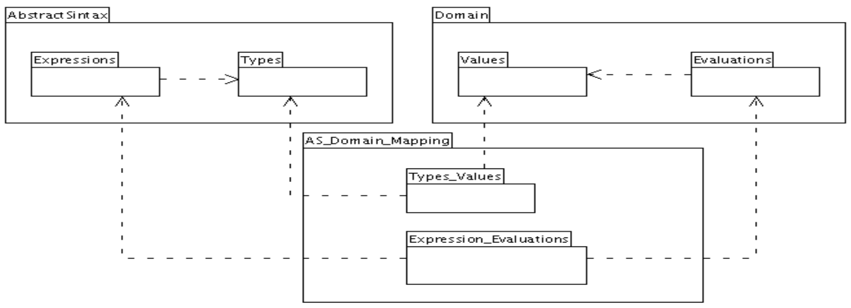

Figure 1: Overview of packages in the UML-based semantics

Figure 2 shows the overview of the *AbstractSintax* package, which defines the abstract syntax of OCL as a hierarchy of meta classes. In the other hand, *Evaluations* package defines the semantics of these expressions using also a hierarchy of meta classes where each one represents an evaluation of a particular kind of expression (see figure 3). The idea behind this representation is that each evaluation yields a result in a given environment, therefore, the semantics evaluation of an expression in a specific environment is given by associating each evaluation instance to an expression model (see figure 4).

It is easy to see how *Evaluations* package replicates the hierarchy of the abstract syntax. We believe that this duplication is unnecessary and yields to disadvantages such as low legibility of the meta model and inefficiency in the development of automatics tools based on this semantics. We will expand this in the next section.

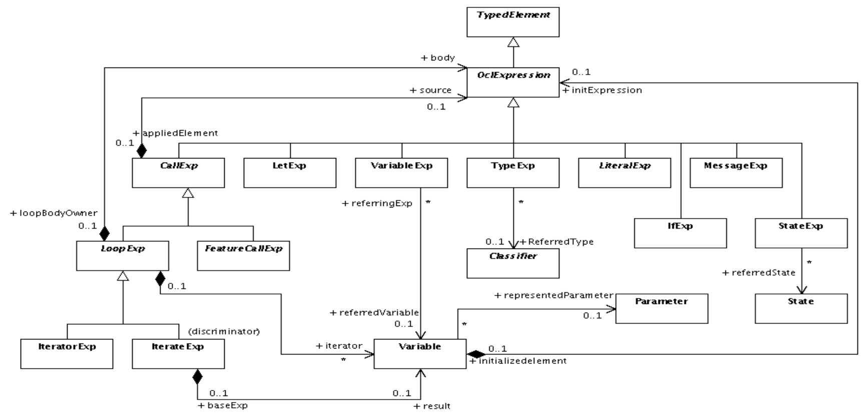
Figure 2: *AbstractSyntax* package overview

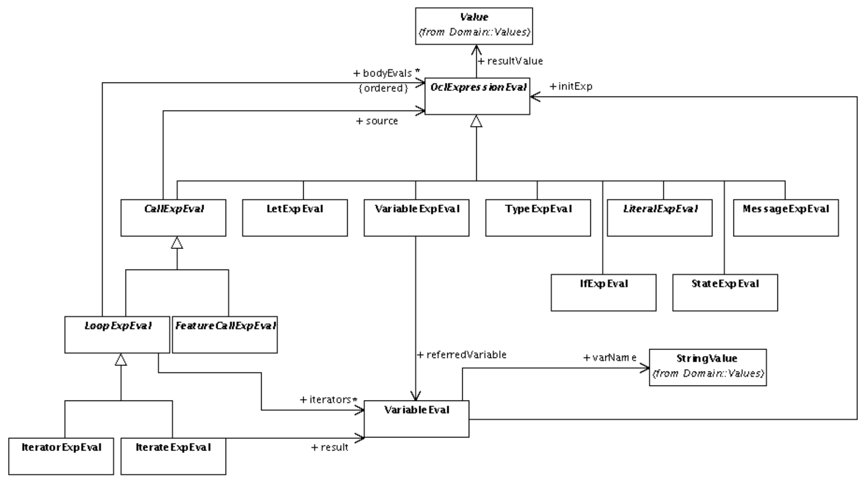
Figure 3: *Evaluations* package overview

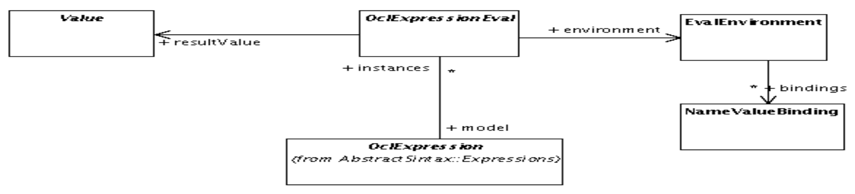
Figure 4: Semantics Evaluation of an expression.

3 Semantics evaluation through "Visitor" pattern and DMM

In this section we define a meta class named *OclEvaluator* to give semantic meaning to syntax expressions by associating them with its corresponding value. In this way, *OclEvaluator* works as a bridge between AbstractSyntax and Values packages (see figure 5).

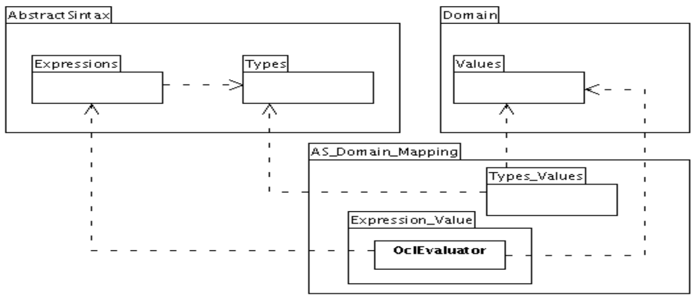

Figure 5: OCL meta model using the Visitor

In order to evaluate an expression, the *OclEvaluator* uses an evaluation environment called *EvalEnvironment* following the classic strategy used in semantic definition of programming languages (examples of this approach can be found in [3] and [10]). An expressions evaluation depends on its evaluation environment as well as its syntax structure.

The *OclExpression* structure is not likely to change, and several operations might be defined (e.g. refactoring operations, semantics evaluation, code generation operations, etc.). Consequently, we consider that it is more appropriate to avoid polluting the static structure with these operations and then to apply the Visitor pattern [4], in order to keep it simple and clear (See figure 6).

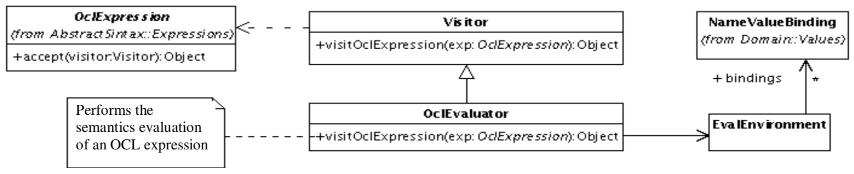
OclExpression
(from AbstractSintax::Expressions)
+accept(visitor:Visitor):Object
Visitor
+visitOclExpression(exp: OclExpression):Object
NameValueBinding
(from Domain::Values)
+ bindings
*
Performs the semantics evaluation of an OCL expression
OclEvaluator
+visitOclExpression(exp: OclExpression):Object
EvalEnvironment

Figure 6: Evaluation metamodel using the Visitor pattern.

In addition, we believe that the best way to understand the semantics evaluation is by showing the evaluation process itself. By using only class diagrams to reflect the semantics evaluation, it is hard (or almost impossible) to reveal the latter process, because of the static nature inherent to these diagrams. Furthermore, to completely understand all the process it is necessary to pay attention to the constraints established on these diagrams. In [8], these constrains are written in OCL with two negatives outcomes:

- The expressibility and simplicity obtained from the use of UML in the semantics metamodel over the math one is lost because of the necessity of be aware of the constraints to fully understand the semantic.
- The constraints are written in OCL, so that the semantics of OCL is defined in terms of OCL itself! If someone didn't understand OCL, they would neither understand these constraints.

Consequently, with the aim of a simple, precise and clear explanation, in this section we use sequence diagrams to visualize the distinct steps throughout the semantics evaluation of expressions. This approach is known as *Dynamic Meta Modelling* (DMM) [2] [5], and has been used in the semantics specification of UML elements (such as State Machines and Collaborations), but its use in OCL specification has not been explored before.

3.1 *Semantics of a LetExp*

The evaluation of a *LetExp* proposed in [8] is shown in figure 7. The diagram shows how the evaluation encapsulates the result value and the evaluation environment, although neither the evaluation method nor structural constraints are specified on this diagram.

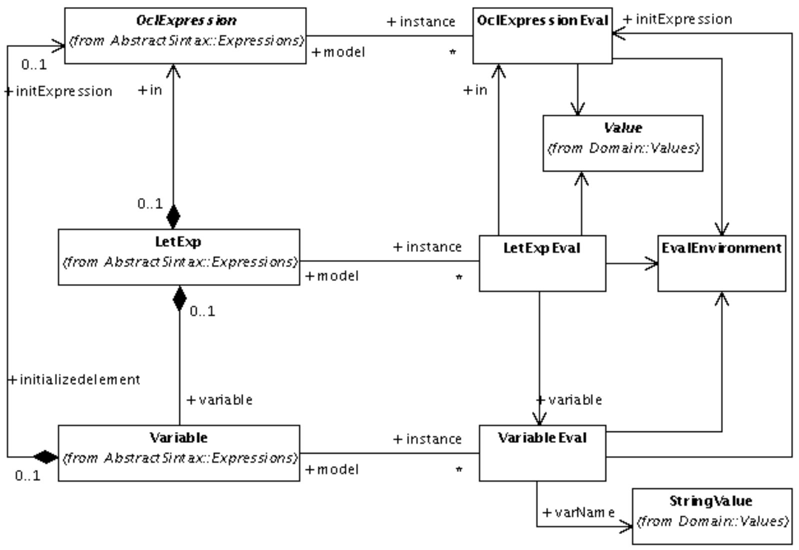

Figure 7: Standard UML based semantic evaluation of a *LetExp*.

Therefore a simple analysis of the last diagram doesn't give us too much information about the semantics of a *LetExp*; it only gives us information about the static structure of the elements implied in this evaluation. In order to fully understand the previous diagram, we must study its well formed rules [8].

As we previously established, these constraints have the disadvantage that they are written in OCL, which clearly becomes an obstacle for those who give their first steps in OCL.

The appendix A of [7] presents the maths model of the OCL semantics (see figure 8). Taking it into account, we can translate this algorithm under the applicative order reduction into a sequence diagram (see figure 9).

A context for evaluation is given by an environment $\tau = (\sigma, \beta)$ consisting of a system state σ and a variable assignment $\beta\colon Var_t \rightarrow I\ (t)$. A system state σ provides access to the set of currently existing objects, their attribute values, and association links between objects. A variable assignment β maps variable names to values.
Let *Env* be the set of environments $\tau = (\sigma, \beta)$. The semantics of a *LetExp* is a function $I[e]\colon Env \rightarrow I(t)$ that is defined as follows.

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

Figure 8: Maths semantics of *LetExp*

As a first step of evaluation, we evaluate the *init* expression ($I[\![e_1]\!](\tau)$, signals 4 and 5) to extend the evaluation environment with the latter evaluation ($\beta\{v/I[\![e_1]\!](\tau)\}$, signals 6, 7 and 8). Then, we evaluate the *in* expression in the new environment, and the value returned by this is the result of the whole *LetExp* evaluation

($I[\![e_2]\!](\sigma, \beta\{v/I[\![e_1]\!](\tau)\})$, signals 9, 10 and 11). Note that the internal environment modification were propagated outside the *LetExp* evaluation, we saved the environment at the beginning of the evaluation process to recover it when the evaluation is finished (signals 2 and 12).

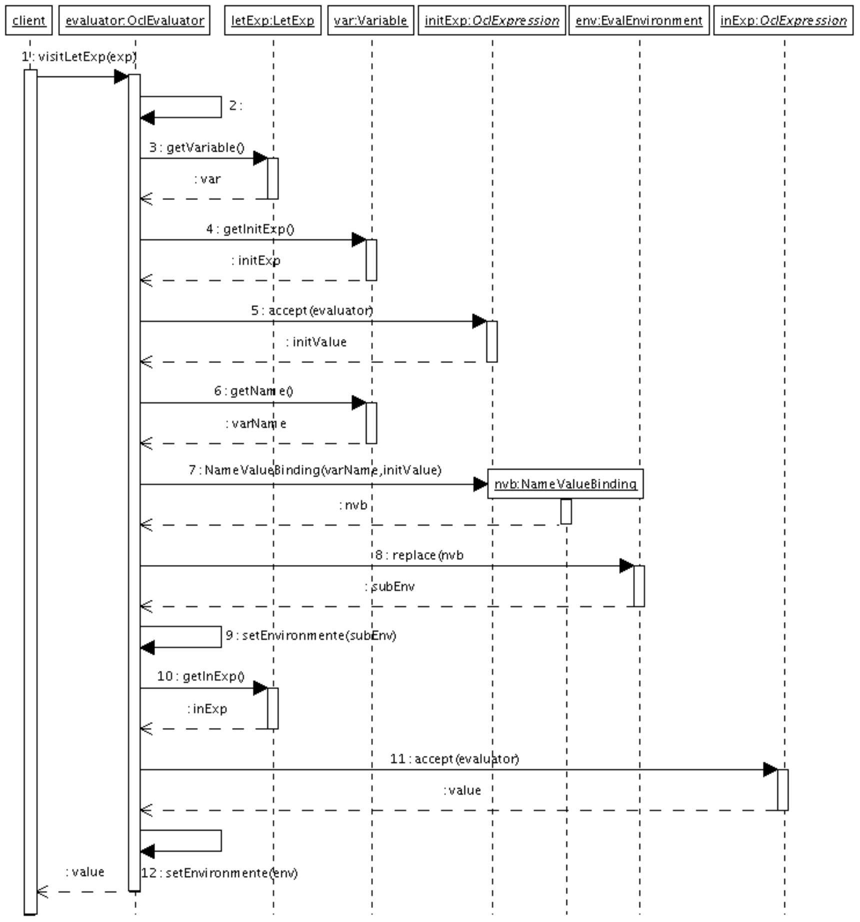

Figure 9: Sequence diagrama of a *LetExp* evaluation.

3.2 *Semantics of IterateExp*

The semantics evaluation of an *IterateExp* as is expressed in [8] is shown in figure 10. Once again we have the problem that the chart doesn't express too much about the semantics evaluation process and we have to appeal to the well formedness rules established on this diagram [8]; without these constraints we would be unable to completely understand the semantic process of an *IterateExp.*

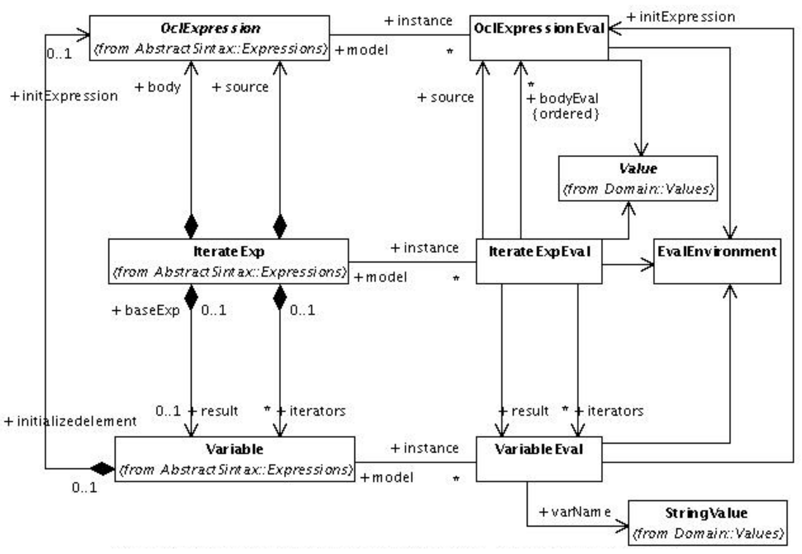

Figure 10: Standard UML based semantic evaluation of an *IterateExp.*

Even worse, such constraints try to explain how *IterateExp* works but lacks of correctness due to the fact that the *IterateExp* is defined in terms of a *ForAllExp* wich is itself defined in terms of *IterateExp*, as follows:

The environment of any sub evaluation is the same environment as the one from its previous sub evaluation, taking into account the bindings of the iterator variables, plus the result variable which is bound to the result value of the last sub evaluation.

```
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
```

Although an *IterateExp* is more complicated than a *LetExp,* without a previous OCL knowledge, it is almost impossible to understand these constraints, and with the proper knowledge of the language, the reading and comprehensiveness of these constraints is a hard task to do.

As we have done whit the *LetExp*, we use the math semantics of the *IterateExp* as guidance for showing this process through a sequence diagram. A summary of the math semantics is shown in figure 11 (see Appendix A of [7] for the full version), while figure 12 and figure 13 display the semantics expressed via sequence diagrams.

$I[\![e_1 \rightarrow \text{iterate}(v_1; v_2 = e_2 \mid e_3)]\!](\tau) = I[\![e_1 \rightarrow \text{iterate}'(v_1 \mid e_3)]\!](\tau')$ 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 $e_1 \in \text{Expr}_{\mathit{Sequence}(t_1)}$ then

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

Figure 11: Maths semantics of IterateExp

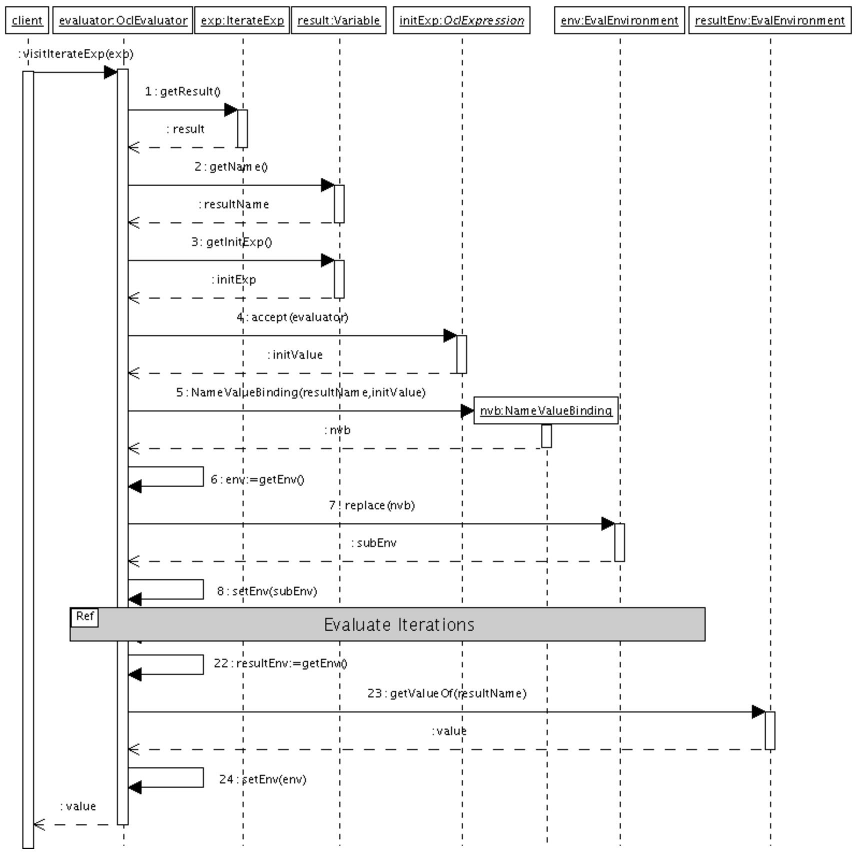

Figura 12: *IterateExp* semantics as sequence diagram.

IterateExp evaluation can be seen as follows:

The first sub evaluation will start with an environment in which the result variable is bound to the init expression of the variable declaration in which it is defined ($\beta' := \beta\{v_2/I[\![e_2]\!](\tau)\}$, signals 1 to 8 in figure 12); then we proceed to evaluate the body with all iterator variables bound to the different combinations of the source (figure 11). The iterators binding ($\beta'\{v_1/x_1\}$ in $\beta'' := \beta'\{v_2/I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$) is done by *CombinationGenerator* (signals

13, 14 and 15 in figure 13), under a 'depth first search' strategy. This strategy determines the number of sub evaluations over the body ($I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})$ in $\beta'' := \beta'\{v_2/I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$; signals 17 and 18 in figure 13); as last step, these sub evaluations will update the result variable ($\beta'\{v_2/I[\![e_3]\!](\sigma, \beta'\{v_1/x_1\})\}$,signals 19, 20 and 21 in figure 13).

Once again we save the environment at the beginning of the evaluation process and, after recovered the value bound to the result variable (signals 22 and 23 in figure 12), we restore the initial environment.

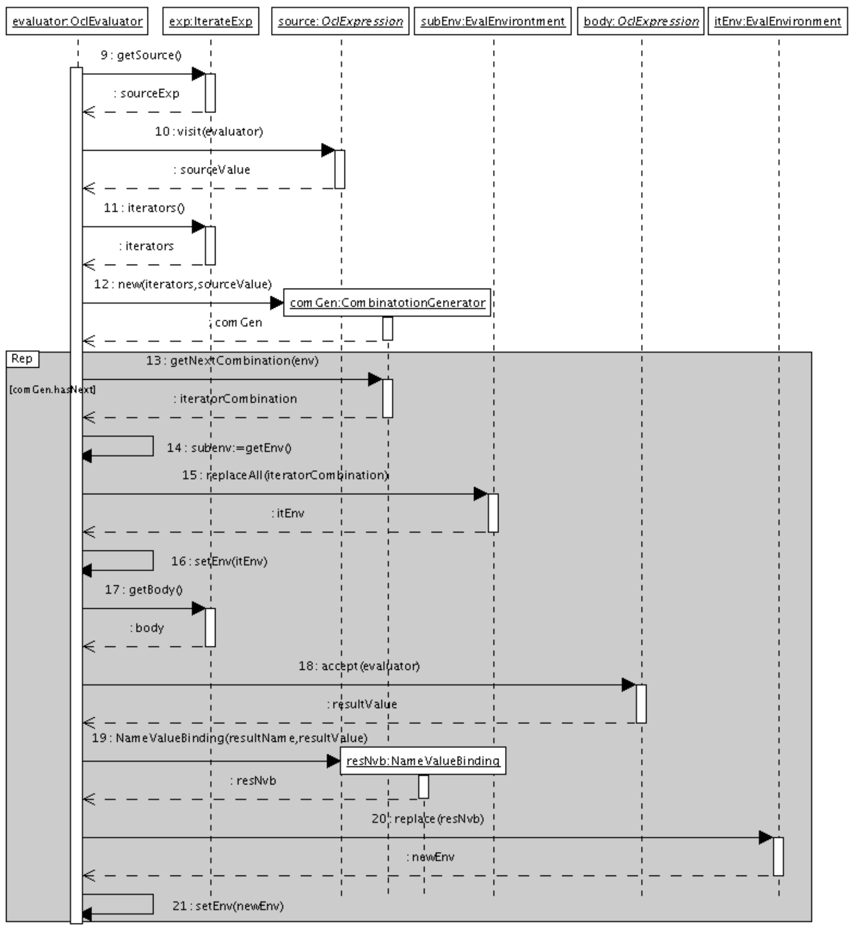

Figure 13: Body Evaluation of an *IterateExp.*

With this approach, each meta-class belonging to the Domain package will be replaced with a sequence diagram which states the concrete semantics and evaluation process of the corresponding syntactic construction.

4 Conclusion and Future Works

OCL is an object property specification language, which is rigorous but simple and easy to use. Therefore, it becomes a very interesting option for the development of code verification and derivation tools. To exploit all its potential, it is fundamental that OCL has a solid formal foundation for both its syntax and its semantic definitions. The OCL standard is formalized by metamodels expressed in MOF, complemented by well formedness rules written in the own OCL. This circular definition not only gives rise to formal problems [11], but also puts obstacles in language understandings. Additionally, we think that the use of (static) meta-classes to express the OCL semantics was a wrong choice, because of the dynamic nature of semantics evaluation which requires a dynamic (meta) modeling tool.

In this article, we elaborate an alternative definition for the OCL semantics. This proposal re-uses the OCL syntax metamodel, re-designs the OCL semantics metamodel by applying the 'Visitor' design pattern, and finally defines the relation between syntax and semantics through UML collaboration diagrams adhering to the DMM approach. In this way, circularity on the OCL definition is avoided, and intuitive communication is increased. Besides, the OCL math semantics was used as a foundation and guidance for the semantics definition. Although math semantics could be tedious and hard to understand, and demands users with more academic background, we showed that it could be translated into sequence diagrams offering a more readable and simple semantics metamodel.

On the other hand, the adequate performance of the tools supporting OCL [12] [1] strongly depends on the quality of language definition. To count on a well-defined syntax and semantics will result in benefits for such tools. Also, it is almost straightforward to translate this semantics into a programming language such as Java, because of the proximity between sequence diagrams and programming languages.

Finally, the application of the visitor pattern makes it easier the creation of new functionality over the OCL syntax structure and its integration into the CASE tool to get a powerful one. For example, concerning model transformations, it is possible to define OCL constraints transformations by adding a new "visitor" for the OCL syntax hierarchy. In this sense, we are working on the redefinition of the ePlatero evaluator [9] following the proposal presented in this article in order to analyze the potential advantages regarding the different indicators, such as reliability, efficiency, modifiability, etc.

References

[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". Lecture Notes in Computer Sciefnce number 3297. Springer-Verlag. 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