Precise Modeling with UML: Why OCL?

R.Duarte, J.Júnior, A.Mota

Federal University of Pernambuco
Centre of Informatics
P.O.Box 7851 - Cidade Universitária
50740-540 - Recife - PE - Brazil

Abstract

Nowadays UML is the most common graphical notation for object-oriented software development. But UML itself is not enough to model some software aspects precisely. Hence, IBM has proposed the Object Constraint Language (OCL) as the UML standard specification language. To be attractive, OCL is announced to be easy as well as avoid mathematical and logical descriptions. In this paper we argue that OCL is not special, being easily comparable to Object-Z, an object-oriented extension of the model-based language Z. In particular, aspects concerning formal semantics and refinement reveals that OCL is indeed an immature language. The comparison also yields a translation from OCL to Object-Z.

1 Introduction

The Unified Modeling Language (UML), a standard graphical notation provided by the Object Management Group (OMG), is the most popular and widespread graphical modeling notation for object-oriented software development [BRJ99]. Indeed, there are software processes entirely based on the use of UML, such as the RUP [Kru00]. Unfortunately, neither UML has a formal semantics nor capabilities to characterise software properties formally.

Formal methods, on the other hand, are based on a formal semantics, where elements of logic and mathematics are used. However, since in general they lack graphical notation and still have limited integrated tool support, these have contributed to formal methods still suffer some rejection by the software industry [CW96].

Therefore, bringing together UML and Formal Methods seems a very promising effort for two main reasons. First, one can disseminate the use of formal methods through UML graphical diagrams. And second, one can turn the current software development practices more rigorous. In view of this, IBM has proposed the Object Constraint Language (OCL) as the UML standard specification language. To be attractive, OCL is announced to be easy as well as avoid mathematical and logical descriptions.

However, although OCL seems more easy than traditional specification languages to the industry, it still lacks a complete formal semantics. Furthermore, the literature does not mention the OCL characteristics related to formal development to the best of our knowledge.

In this paper we investigate OCL in various aspects, ranging from syntax and readability aspects through formal semantics, tool and refinement support. To accomplish this we use Object-Z, an object-oriented version of the model-based language Z, as a reference language.

A further contribution of this paper concerns determining the practical usage of OCL in a formal development life-cycle. In order to do that we establish a correspondence between OCL and Object-Z. We aim at still applying OCL due to the wide acceptance UML and OCL already have by the software industry. Indeed, this is an overall goal provided by the project ForMULa (**For**mal **M**ethods and **U**m**L** integr**a**tion) by which the present effort contributes.

This paper is organised as follows. Section 2 presents a UML overview using an example. In Section 3 the main elements of the OCL language are introduced and explained. The reference language, Object-Z, is the focus of the Section 4. The comparisonbetween Object-Z and OCL is considered in Section 5. Finally, concluding and future research remarks are described in Section 6.

2 UML Overview

The Unified Modeling Language (UML), a standard graphical notation provided by the Object Management Group (OMG), is the most popular and widespread graphical modeling notation for object-oriented software development [BRJ99]. Indeed, there are software processes entirely based on the use of UML, such as the RUP [Kru00].

UML is composed of various diagrams, each one with its own specific purpose. Some diagrams are provided for analysis and design, whereas others are related to implementation and distribution. The purpose of an UML diagram is to visualize some aspect of a software system easily. For instance,

UML class diagrams are used to present the static view of a system, whereas state diagrams for the dynamic aspects.

Unfortunately, almost all UML diagrams do not have a formal semantics[1]. What it is common for the UML community is that UML diagrams have rigid rules on how they can be depicted. Such rules are stated using a meta-model description with a core of UML itself. However, these rules are not sufficient to provide more specific characteristics of software systems and this is one reason the UML community has adopted the OCL [WK99] (Object Constraint Language) language for such a purpose.

In this paper we are concerned specifically with UML class diagrams. The reason is simple: they are the most common diagrams found in software development projects.

2.1 UML Class Diagrams

The static aspect of a system is normally described using UML class diagrams. In such diagrams, the specifier is concerned with classes, interfaces, collaborations, and relationships.

UML diagrams can accommodate constraints (predicates) as notes attached to any UML graphical element. In particular, UML class diagrams use such a facilite to annotate pre- and post-conditions of operations as well as class invariants.

Figure 1 illustrates a UML class diagram that models a simple company. From this diagram we can identify three packages, *company*, *person*, and *util*, which group other packages and classes. The remainder elements of this diagram are explained in corresponding sections.

2.2 Classes

A class is a description of a set of objects that share the same attributes, operations, relationships and semantics. A class has a name, attributes and operations. An attribute has a name, a visibility, a type, and a multiplicity, and describes an interval of values that the instances of the property can assume. An operation has a name, a visibility and parameters, and is an implementation of a service that can be requested of any object of a class to affect its behavior.

The class diagram of Figure 1 has several classes, where the class *Person* belongs to the package *person* and the class *Department* belongs to the package *company*, for example. The class *Person*, in particular, has three private

[1] In particular, state chart diagrams have a formal semantics since they are based on state charts [HPSS87].

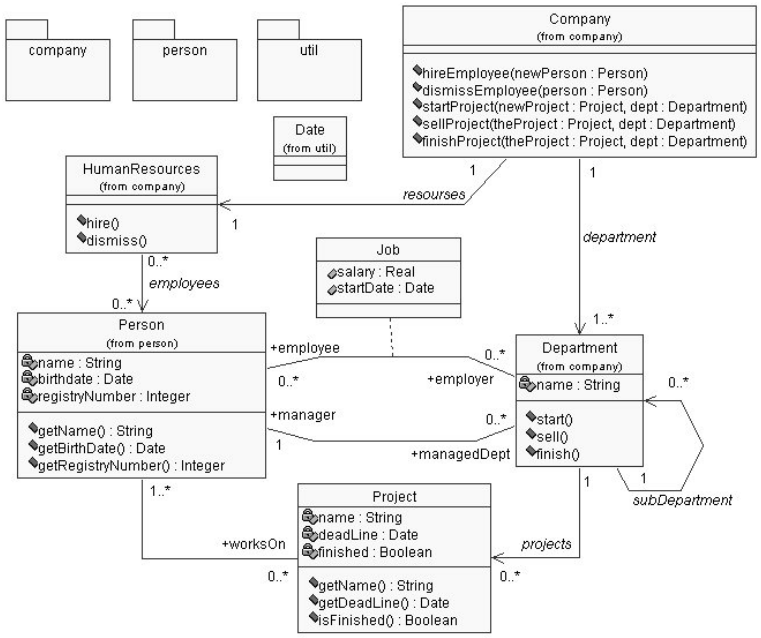


Figure 1: Example of a UML class diagram

(invisible) attributes, *name*, *birthdate*, and *registryNumber*, as well as three public methods, *getName*, *getBirthDate*, and *getRegistryNumber*.

2.3 Associations

An association is a structural relationship that specifies what objects of a type (class) are connected to objects of, possibly, another type. For instance, Figure 1 has a directed association from class *Company* to class *Department*, called *department*. The association direction enforces the navigability (or reference) from one class to the other and not the reverse. Furthermore, each end of an association can have a multiplicity, that is, how many objects can exist at each end. For the *department* association, we can see that one *Company* can be associated to one or more *Department*s.

Figure 1 also illustrates an important kind of association, named a class association. A class association is an association between two classes, where the association itself has an attached class. The attached class serves to accomodate attributes and methods that do not belong to the main related classes.

3 UML Precise Modeling with OCL

UML provides itself a formal language to express constraints, the Object Constraint Language (OCL) [OMG03]. It is a formal language that can be used to specify invariants, pre- and post- conditions as well as describe guards and constraints on operations. OCL expressions do not have side-effects. That is, when they are evaluated the system state does not change. But expressions can be used to specify state changes like other formal languages [Spi92, Smi00]. Since OCL is a typed language, expressions have a well-defined type, although some of them can result in an undefined type.

3.1 Invariants, Pre, Post, and Definitions

In this section we consider the main constructs of OCL using simple examples based on the UML class diagram of Figure 1.

We start by considering an invariant (`inv`) over the class *Job* (`context Job`). The following constraint states that the attribute *salary* of the current object of the class *Job* must be greater than 1000.

```
context Job inv:
    self.salary > 1000
```

To see how OCL can work as a navigation language, accessing objects via associations, we use the next invariant. We navigate from class *Department* to class *Project* (`self.projects`), count how many projects are associated to (`->size()`), and state that this amount must be less than or equal to 10.

```
context Department
    inv: self.projects->size() <= 10
```

To illustrate an OCL first-order based constraint we present the last invariant of this section. In this invariant, we enforce that all projects a person is working on must not be finished yet.

```
context Person
    inv projectFinished:
    self.worksOn->forAll(p: Project | p.isFinished() = false)
```

Finally, we consider the specification of an operation. In OCL, like VDM or B, pre- and post-conditions are stated explicitly. For OCL, the keywords `pre` and `post` are used. The method `hire` requires that a person (`p`) must not be (`->exclude(p)`) an employee (`self.employees`) provided by the class `HumanResources`. If the method works normally then the person must now be included in the employees collection of class `HumanResources`.

```
context HumanResources::hire(p: Person)
    pre: self.employees->exclude(p)
    post: self.employees->including(p)
```

4 UML Precise Modeling with Object-Z

In the previous section we have seen that OCL is used to complement UML class diagrams; this is the standard combination provided by the OMG. However, other approaches can be used as well [MBM03].

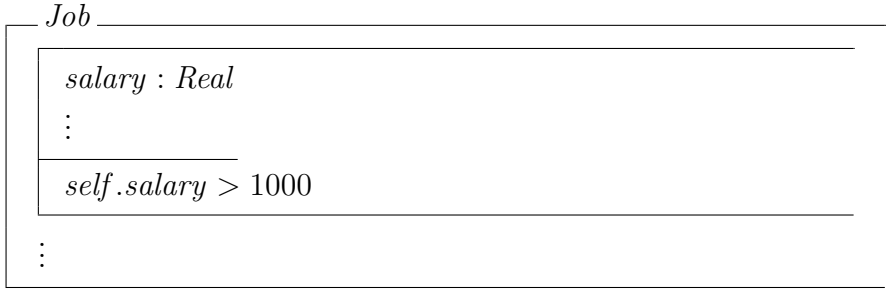
In this section we show examples corresponding to the ones presented in the previous section using fragments of the language Object-Z.

Object-Z [Smi00] is an object-oriented version of the model-based language Z [Spi92], whose purpose is to improve the clarity of large specifications (and to facilitate its design) through enhanced structuring in an object-oriented style [Boo91].

4.1 Invariants, Pre, Post, and Definitions

Object-Z alone is able to elaborate models equivalents to the ones originated using UML annotated with OCL. That is, Object-Z has constructs to specify classes, attributes, and methods, as well as invariants, pre- and post-conditions, definitions, and theorems.

Our first example defines a class *Job* (corresponding to the UML description) which has the attribute salary of type real (corresponding to the UML description) and states that every salary must be greater than 1000 (correspoding to the OCL description, although using a simplified presentation provided by the self contained class definition).



The next example corresponds to the OCL constraint () of section 3. In this fragment, it is usith noting that the cardinality operator `#` replaces the combination `->size()` of OCL.

Department

$projects : \mathbb{P}\, Project$

$\vdots$

$\#self.projects \leq 10$

$\vdots$

The following example states that for all person `p` working on some project (`self.worksOn`), a call to the method isFinished of p generates a result and this result must be False.

Person

$worksOn : \mathbb{P}\, Project$

$\vdots$

$\forall\, p : self.worksOn \bullet p.isFinished \land p.result! = False$

$\vdots$

The last example of this section presents the specification of a method. Although Object-Z makes no explicit destinction between pre- and post-conditions, the predicate $p? \notin self.employees$ corresponds to the pre-condition of OCL and the predicate $p? \in self.employees'$ corresponds to the post-condition.

HumanResources

$employees : \mathbb{P}\, Person$

$\vdots$

$\vdots$

hire

$\Delta(employees)$

$p? : Person$

$p? \notin self.employees$

$p? \in self.employees'$

$\vdots$

5 OCL versus Object-Z

Sections 3 and 4 have presented an overview of OCL and Object-Z respectively. Clearly, OCL and Object-Z have some kind of correspondence. But what is it? Are they completely equivalent?

In this section are ivestigate these languages more deeply and systematicly in order to highlight the advantages and drawbacks of both. This analysis will also originate a way of translating OCL into Object-Z and vice-versa where it is possible.

5.1 Syntax and Readability

From section 3, we can see that OCL is a language combining object-orinted programming constructs (`->, ., ::, and context` from, for example, C++) and a verbose (textual) version of graphical languages of Logic (`forall` instead of $\forall$ and `size()` instead of `#`) and Mathemathics. On the other hand, section 4 shows that Object-Z was basically the traditional graphical language of Logic and Mathemathics with addtional graphical notation from the Object-Z/Z scheme calculus.

Doing an impartial analysis, we can say that OCL has a more familiar syntax to software practitioners than other formalisms. Thus OCL description can obviously be expected to be more readable. However, if OCL does not need a strong mathematical background then the other formalisms do not require as well. Thus, we agree that it can be interesting to use OCL descriptions to annotate UML diagrams instead of, for example, Object-Z.

These observations that lead the creators of OCL to claim OCL is easier than other languages, or that there is no need of a strong mathematical backgroung to use OCL.

5.2 Semantics

Since OCL was created it has been analyzed and several formal semantics have already been presented, in particular by Bickford and Guaspari [BG98], Hamie, Howse, and Kent [HHK98], and Richters and Gogolla [RG02, RG98] for OCL 1.1 and by Clark [Cla99], and Cengarle and Knapp [CK01] for OCL 1.4. However, there is not a definitive formalization because OCL still does not have a final version - as we said before, the OCL 2.0 was already submitted, but not approved yet.

Because these semantics show deficiencies in handling the OCL types OclAny and OclType [RG98, HHK98], the OCL flattening rules [BG98, HHK98], empty collections [RG98, Cla99], undefined values [HHK98, Cla99],

non-determinism [BG98, HHK98, RG98], and overridden properties [BG98, HHK98, RG98].

While OCL is on formalization process, Object-Z already has a formal semantics that is fundamental for its usage in Formal Methods. Then if OCL has a similar syntax in relation to Object-Z, why not translate the OCL constraints to (Object-)Z?

While syntax and readability aspects favor OCL usage, semantics is a critical problem for OCL. Currently, in its proposal 2.0 (version), the OCL team developers and collaborators try to fix innumerors design flaws. From vague informal semantical descriptions to features like , OCL is far form being a stable language. Therefore, in this respect, Object-Z is the best choice because it is now a stable language having a complete formal semantics.

5.3 Refinement

OCL, being completely new, cannot reuse any existing tools. Therefore, Object-Z is the best choice again.

This is a delicate point because neither Object-Z nor OCL have a vast repository of tools. Basically, both have partial syntax and type checkers. That is, they work for a subset of the languages.

However, Onject-Z has an experimental theorem prover support via its encoding in HOL [], and a proposal for modelchecking []. Besides these instable tools, all Z tools can be used for Object-Z after its translation into pure Z according to its semantics [].

One of the most interesting and important aspects of a formal specification language is its hability to support refinament-based software development. Or achieving a concrete model (implementation) from an abstract one, guaranteeing the properties specified initially.

Object-Z is a language with refinament support, in particular, having static and dinamic notions of refinament. However, OCL lacks this important feature; Only [] reports an initial notion of refinament for OCL which is insufficient to be applied actually.

Another drawback we point out about refinament for OCL is the fact that an OCL specification only makes sence if accompained with a UML diagram. Therefore, refinament relations cannot be defined for OCL isolated.

5.4 Tool Support

Object-Z has many benefits to the specification development, but still suffer some rejection, due to the lack of tool support - if compared to Z. But there are two projects that can bring some benefits to its use in formal

methods: a approach to integrate UML and Object-Z/Z [MBM03]; and a design environment (see http://www10.org/cdrom/papers/182/) for Object-Z. These two projects use the Rational Rose UML, but with different goals.

The first one tries to improve the use of Formal Methods through the mapping of UML diagrams to Object-Z/Z specifications. Then we can refine and make property and type checking. It is done integrating three applications: Rose, Z-EVES and Wizard. While the second is a web based environment to visualize an Object-Z specification, supporting inheritance expansion and that can export the specification, via XSLT technology [2], to Rose.

While Object-Z has a growing use in formal methods, OCL still is starting its life in the community. But, like Object-Z, OCL also has a type-checker that can be found in the klasse objecten web site:

http://www.klasse.nl/ocl/

Moreover, the OCL tools that have been created are, in most cases, idealized to be used in CASE tools and they provide several different OCL implementations, most noteworthy the Bremen USE tool, the Dresden OCL tool, and the Argo UML. These differ e.g. in their handling of collections and "oclAsType", some do not flatten a Collection of Collections; the USE tool does not include "let expressions". Although these OCL tools have many benefits to the language development, the differences make inconveniences to the modeler and can be dangerous.

5.5 OCL as Object-Z

From the previous sections, it is obvious that currently OCL alone can only be used to give an apparent formal description. Since our present effort is in the direction of the project ForMULa, a clear usage of OCL is simply as a front-end of a real formal language. Thus, in this section are investigate a possible mapping between OCL and Object-Z in such a way that OCL can be used to write specifications but formal development be done using Object-Z.

From the literature we have found the work of [LS02] gives a mapping from OCL to B [], for non-object-oriented constructs. Hence, as long as B constructs are very closer to Object-Z ones, we only need to extend the mapping to catch the object-oriented features of OCL. Table summerises our correspondence.

A tool base on the material of this section, has already started to be developed. It reuses part of The Dresden OCL tool [3], which is able to generate Java code from OCL constraints.

[2] http://nt-appn.comp.nus.edu.sg/fm/zml/

[3] This tool is intended to be integrated to the tool described in [MBM03], allowing

6 Conclusion

The creation of OCL was a very interesting effort to bring some formality to UML. Although, OCL is still quite far from being a well accepted language, due to aspects concerning its semantics, consistency and expressivity. The translation between OCL and Object-Z is then shown as viable alternative to effectively use OCL within UML, .

Acknowledgement 1 *We would like to thank Augusto Sampaio for many valuable comments about the present research effort. In particular, for his criticisms and suggestions concerning the refinement aspects of OCL and Object-Z.*

References

[BG98] M. Brickford and D. Guaspari. Lightweight analysis of uml. Technical report, 1998.

[Boo91] G. Booch. *Object-Oriented Analysis and Design with Applications*. Benjamin–Cummings, Redwood City, Calif., 1st edition, 1991.

[BRJ99] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley, 1999.

[CK01] M. Cengarle and A. Knapp. A formal semantics for OCL 1.4. In Martin Gogolla and Cris Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 118–133. Springer, 2001.

[Cla99] T. Clark. Type checking UML static diagrams. In Robert France and Bernhard Rumpe, editors, *UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings*, volume 1723 of *LNCS*, pages 503–517. Springer, 1999.

[CW96] E. M. Clarke and J. M. Wing. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys*, 1996.

Software developers annotate UML class diagrams with OCL but working implicity with Object-Z, achieving proof of properties, data refinament and model checking.

[HHK98] A. Hamie, J. Howse, and S. Kent. Interpreting the Object Constraint Language. In *Proceedings 5th Asia Pacific Software Engineering Conference (APSEC '98), December 2-4, 1998, Taipei, Taiwan*. IEEE Computer Society, 1998.

[HPSS87] D. Harel, A. Pnueli, J. Schmidt, and R. Sherman. On the Formal Semantics of Statecharts. *2nd IEEE Symp. on Logic in Computer Science - Ithaca*, 1987.

[Kru00] P. Kruchten. *An Introduction to the Rational Unified Process*. Addison-Wesley, 2000.

[LS02] H. Ledang and J. Souquiéres. Derivation schemes from ocl expressions to b. Technical report, 2002.

[MBM03] P. Moura, R. Borges, and A. Mota. Experimenting Formal Methods through UML. *submitted to the Workshop of Formal Methods*, 2003.

[OMG03] OMG. Unified modeling language. Specification v1.5, Object Management Group, March 2003. http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[RG98] M. Richters and M. Gogolla. On formalizing the UML Object Constraint Language OCL. In Tok Wang Ling, Sudha Ram, and Mong Li Lee, editors, *Proc. 17th Int. Conf. Conceptual Modeling (ER'98)*, volume 1507 of *LNCS*, pages 449–464. Springer, 1998.

[RG02] M. Richters and M. Gogolla. OCL: Syntax, semantics, and tools. In Tony Clark and Jos Warmer, editors, *Object Modeling with the OCL: The Rationale behind the Object Constraint Language*, pages 42–68. Springer, 2002.

[Smi00] G. Smith. *The Object-Z Specification Language*. Kluwer Academic Publisher, 2000.

[Spi92] M. Spivey. *The Z Notation*. Prentice-Hall, 1992.

[WK99] J. Warmer and A. Kleppe. *The Object Constraint Language: Precise Modeling with UML*. Addison-Wesley, 1999.