

Endowing UML with a MOF Based Semantic Core

Jamal Abd-Ali, Karim El Guemhioui

Department of Computer Science and Engineering

Université du Québec en Outaouais (UQO)

P.O. Box 1250 Station Hull, Gatineau QC, J8X 3X7 Canada

{abdj01, karim}@uqo.ca

Abstract

In a model driven engineering (MDE) context, using UML to precisely specify a system with no technology dependency is still an outstanding challenge. Based on the principle that different technologies implement their concepts and mechanisms relying on a finite set of low level concepts, we propose a UML extension that integrates dedicated elements supporting these concepts with well defined semantics based on specifying the behaviour semantics via a code mapping approach. This endeavour is mainly intended to serve Model Driven Engineering (MDE) needs and to address the specification of UML elements' semantics.

1. Introduction

UML is a universal modelling language that can, in principle, cater for a wide variety of information system domains, and its general purpose mandate is largely the reason of its wide use and success. Subsequently, UML abounds of polymorph constructs whose semantics depend on the domain and context of use. Thereby, a UML class can either represent a class of objects or a structure of a table of records, depending on the designer's choice. So a UML model represents an abstraction of some real system; abstraction that needs to be interpreted in light of some mapping (originating in the author's mind) between representing modelling elements and the represented reality. For example, the use of a UML Component element, per se, falls short in providing necessary information to handle the Component's hosting.

Obviously, UML elements' semantics are not enough precise and flexible to describe all the specific semantics a modeller needs. UML profiles, as

extending mechanism, are intended to cover such specific semantics expression needs, but there is no UML based way to specify the semantics of new introduced elements in a UML profile. We often resort to an informal textual explanation of the purpose of use of these elements. A standardized and non redundant solution to the above frustration would be of great help.

One possible approach could be to seek a unique UML extension, or rather "base", that addresses this issue. Since UML is a general purpose modelling language, such a base must encompass the basic raw elements needed to express any object oriented information system semantics. This does not seem that far fetched if we make an analogy with a programming language, say the C language, which can serve to compile or even translate domain specific languages (DSLs) without loss of information. This implies that C has expressed the DSL semantics, and if we merge the DSL language and C, we can say that C is a base of the DSL language. A similar analogy can be made between, on the one hand, a class library with its implementation programming language, and, on the other hand, existing UML elements with the sought after base (or extension) elements. In a class library of a programming language, each class is fully specified in terms of the underlying low level language, enriched with other library classes, with no cyclic semantic dependency. In a similar way, we propose to fully specify each UML element in terms of a proposed base of new elements, enriched with other existing UML elements.

Thus we can consider UML as a high level modelling language that is built on top of a partially missing low level language or constructs. Our approach is to try to rewrite the UML language in terms of some lower level language that can express all the UML elements' semantics. This low level language

will provide a non ambiguous semantic base; we will call it the UML semantic core or UMLScore. It will represent a UML extension that precisely determines what a UML model element is supposed to represent in an information system.

In the next section we will give a quick overview of some UML related work. Our perspective on UMLScore properties will be presented in section 3. An illustration of the proposed UMLScore for component-based applications is shown in section 4. The direct advantages of our approach and a discussion of our expectations are provided in section 5. Section 6 concludes this paper.

2. Related Work

A huge project aiming to endow UML with a formal semantics is lead by IBM and other international companies [10]. It started three years ago and, to our knowledge, very few results have been made public. This project is concerned with the precise specification of UML elements' semantics but it doesn't aim to enhance the expressive power which is our main concern in parallel to the rigorousness issue. The IBM project is oriented toward the use of formal languages in order to specify UML elements, while our approach addresses the integration of new components in UML in order to solve ambiguity and reusability issues. We can also mention an interesting research aimed at providing a semantic base for the UML elements based on a philosophical and theoretical approach [2]. This work, however, is not focused on extending the UML expressive capability and, therefore, significantly differs from ours. On the other hand, many attempts to provide technology independent meta-models serving the MDE approach [3] have been conducted. We mention here:

- The UML profile for Enterprise Distributed Object Computing [5] witch is an OMG standardized profile aiming to simplify the development of component based applications in conformance with the UML 1.4 and fitting in the Model Driven Architecture (MDA) [3].
- A UML 2.0 profile for security requirement [7]
- A UML 2.0 profile for CORBA and CORBA Component Model. This specification provides a UML 2 profile that facilitates representation of concepts needed to represent a pure CORBA or CORBA Components

PSM. In conjunction with existing OMG specifications, namely UML 2, CORBA, CORBA Components and the MOF 2, this will result in significant benefits to the CORBA and CORBA Components user [9].

The problem with the aforementioned meta-models is the fact that each one of them is intended for a specific domain with no clear path for integrating them, or provision of a rigorous interpretation mechanism for newly introduced elements. Each one of these approaches provides a separate UML profile, but no one can be merged with the other without redundancy issues and other conflicts. The end result is a bunch of scattered extensions of UML for a large inventory of domains. This is in contrast with our approach which seeks a solid and unified semantics and a technology neutral base for UML that can address several specific domains.

3. UMLScore Properties

As stated in the introduction, we propose rewriting the UML language in terms of some lower level language that can express the semantics of UML elements. This low level language will provide a non ambiguous semantic base which we will call the UML semantic core (UMLScore). Below, we detail a set of properties that must characterize UMLScore.

3.1. MOF Based

In order for UMLScore to be a kernel part of UML, UMLScore must be MOF based like UML, and its elements will be defined at the same M2 OMG meta-modelling level; i.e. UMLScore elements' instances can show in any UML model. Thus, from now on, we force the UMLScore model to enable the merging of UML and UMLScore in a unique model that we call the semantic UML or SUML.

3.2. No Redundancy with UML Elements

Any UML element will either represents a simple concept that can be mapped to a non ambiguous implementation, and thus it will join the set of UMLScore elements, or it will represent a complex concept that can be expressed in terms of already defined elements of the UMLScore, and eventually other SUML elements, to express its carried semantics. That way, SUML elements avoid any non beneficial redundancy. The above requirement implies that

UMLScore and UML are made up of disjoint set of elements. This is possible because of the existence of some UML elements that represent atomic concepts with no ambiguity. An example can be given by considering the element Class and its attributes that will be an essential building block of any UMLScore candidate.

3.3. Technology Neutral

Although the goal is not to depend on any technology while defining or choosing UMLScore elements, these latter must constitute a vocabulary sufficient enough to express any implementation technology concept. Thus, the vocabulary must cover all the basic, non specific concepts that implementation technologies use to build or express these specific concepts and constructs. For example, buffering is a common mechanism used in many implementation technologies to offer a flow control construct. Other examples of such basic concepts that are commonly represented in implementation technologies are data structures (such as stacks, queue, and trees), queuing policies (e.g., LIFO, FIFO), etc.

3.4. Modularity

Like in a high level programming language, complexity is managed through modularity, allowing the designer to transparently use high level constructs without being bothered by the underlying low level infrastructure. Nevertheless, this infrastructure is always there to provide full semantics and implementation support for complex constructs. It also supports the definition of new customized constructs when the existing ones do not meet the designer needs.

3.5. Compatibility

Since SUML is the result of merging UMLScore and UML, any newly defined UML model will be a SUML model, which means that the features allowed by the elaboration of SUML will be applicable to any UML model. Conversely, existing UML models whose semantics are deemed insufficient can be refined by adding appropriate UMLScore elements that can express the missing semantics in terms of fine grained specifications of their modelled systems.

3.6. Mapping to a Programming Language

Considering the granularity of the aforementioned examples and noticing that a UML element can be

rewritten in terms of the UMLScore elements which are technology neutral and semantically non ambiguous, the mapping of UMLScore elements to their implementation promises to be manageable and less error prone than ever. This can be achieved even more smoothly if the designer of UMLScore elaborates this mapping while creating the UMLScore language elements. Indeed, what needs to be mapped to code is not only UMLScore elements, but also associations between these elements because such associations contain the sought after semantics. For example, we can consider the enrichment of UML elements with new standardized UMLScore elements such as Stack, Queue, Process, Task, Processor and a set of standardized attributes and associations between them and with other UML elements like Node. We can then move forward and map these UMLScore elements to their implementation with no ambiguity since they belong to basic information system concepts. This will allow deeper expressiveness and an implementation mapping semantic expression, without falling in a technology specific dependency. That way, a Node can be described by a set of Processors that are associated with a Stack of Tasks, a Queue of Tasks, or a priority access Task, etc. This is made possible since the queue, stack and queuing policies were introduced and implemented with no ambiguity. Thus, the Node is a higher level element that was specified with a parameterized set of lower UMLScore constructs. Node can also be considered as a reusable module that is parameterized and that exposes well defined association possibilities to other elements.

One can notice here that the UMLScore elements encompass a set of standardized associations, and may be a new set of association types, in order to allow reusable mapping to code for each type of association connecting two elements. For example, we can define an association named *processes* to connect a task to another task or to a set of tasks.

The mapped code is essential to specify behavioural semantics that go beyond structural aspects and generally concern one or more related SUML elements. The code will be more readable than activity or state diagrams, and it definitely serves better the code generation phase of the software development process.

We are aware that the mapping of modelling elements to code is a challenging task, and we envision to start with an operating system programming language like C which is extensible to cater for object oriented concepts (using C++). Another promising option would be the elaboration of a new programming language which will be tailored to simplify the

definition of a transformation converting any SUML model to executable code in that appropriate programming language. Thus, the code mapping is considered, in first view point, as a formal semantics for the UMLScore elements, expressed in the implementation programming language, and in the other view point as an advanced step toward a solid code generation in an MDE framework.

4. An Illustrative Subset of UMLScore

We propose here some potential UMLScore elements. Our perspective focuses on UMLScore components representation. We chose these components because of their centric role in enterprise applications relying on distributed processing.

We recall that component technologies are in continuous evolution and do not comply with a well known standardized metamodel, and that's why new metamodels and profiles emerge to fill that gap for each technology [6], [9], [1]. Nevertheless, they rely on an enumerable number of concepts and mechanisms that can be captured and injected into the UMLScore. This will, in turn, enable SUML to describe the specifications of components that can be offered within any technology. Thus, SUML must express with no ambiguity concepts like: pooling, instances, listening, calls, etc. It can also use simple and generic concepts like queuing, channel, port, protocol, etc.

The following figure shows proposed enrichment elements to UML for component modelling.

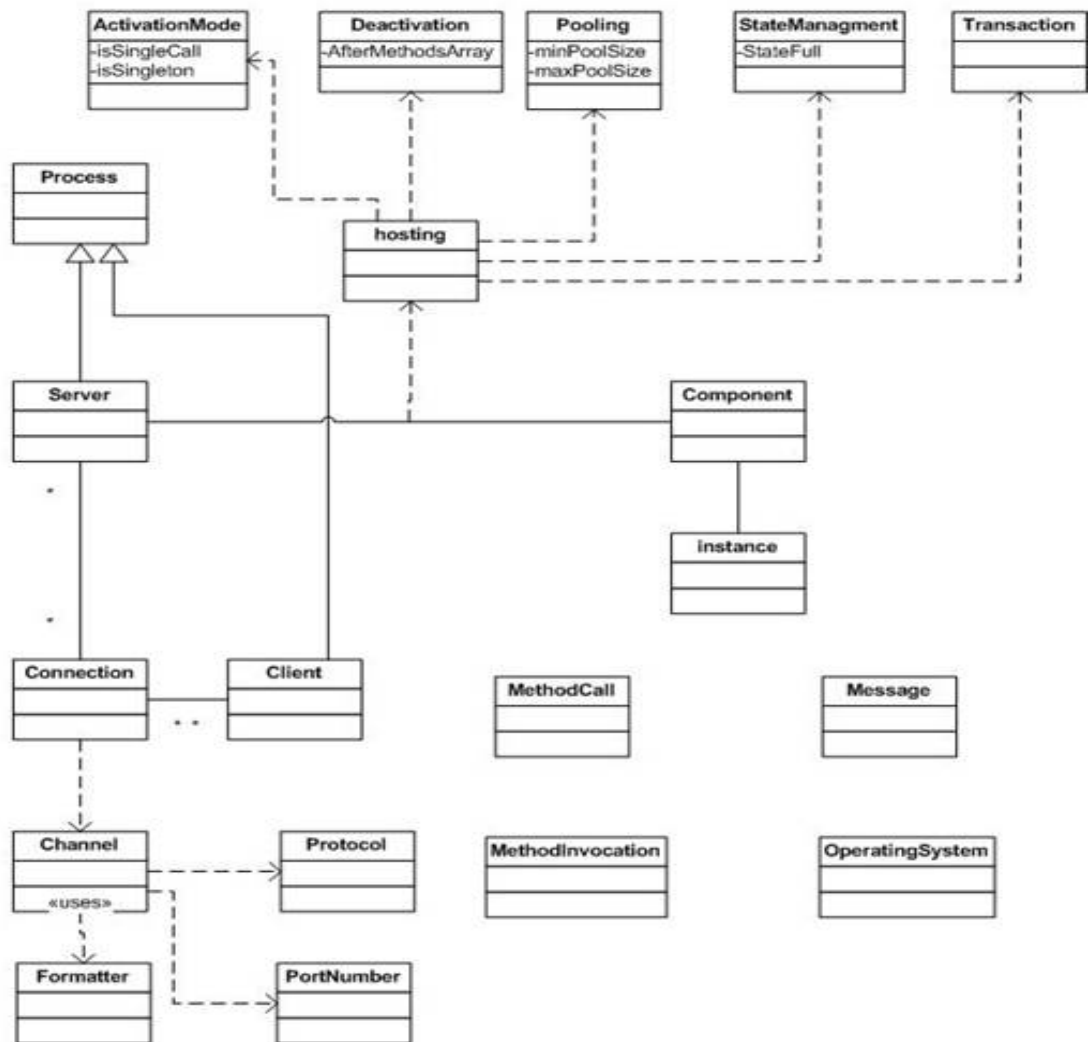


Figure 1 - Set of elements suiting a semantic development using other SUML elements

5. Expressive Power and Complementary Code Mapping

The elements in Figure 1 with the remaining part of UMLScore representing basic low level information systems (mentioned in section 3.6), will be sufficient to describe any component hosting technology with no ambiguity (the elements attributes are not fully shown for legibility purposes). Indeed, we notice that we can specify:

- The pooling policy by specifying the pooling object attributes that indicate the maximum and minimum size of the pool, and which are generally declared in the configuration file of .Net Component hosting applications. Idem for the pooling strategy which can be specified through the deployment configuration of an Enterprise JavaBeans' component.
- The stateless or state-full nature explicitly specified for an EJB component in the deployment descriptor.
- The transaction element that allow specifying the transaction management service for every method of a component. This will be sufficient to express any transaction service regardless of the implementation technology of the components.
- The concurrency control with many attribute values related to the re-entrance of the EJB, or related to one or more methods that need concurrent access prevention service.
- The deactivation and activation elements can be enriched with a large inventory of atomic concepts that can describe the way an instance is allocated to a caller, recycled to another, pooled, deactivated or made candidate for garbage collection service.

The above points lead to the deduction that elements with a good number of attributes with no conflicting constraints, will allow specifying any hosting strategy parameter that we can imagine in any particular technology. This constitutes the

sought-after, full, and technology independent expressiveness power of the proposed SUML. This is made possible since all hosting mechanisms depend on a unique set of low level concepts of communication channels, instances, queuing, and so forth.

In order to better evaluate the expression effectiveness of Figure 1, we recommend the consultation of component dedicated and technology dependant metamodels (e.g., [6], [1]).

Semantic Elements Complemented with Code Mapping

In order to produce a complete specification of a system, we need to use high level elements whose semantics specification depends on their association to other UMLScore elements. The semantics of these UMLScore elements are specified through their mapping to code in a chosen programming language. To illustrate this point, we consider the case of specifying a simplified hosting configuration for a component, by a hosting process called Server.

We insert in Figure 1 a UMLScore association, named Listen, between a Process and a Channel element. This association semantics is given by mapping any instance of such association (Listen) to a block of code of the chosen programming language that achieves a basic listening task. This code will depend on the attributes values of the element Channel specifying the port number and the communication protocol. The listening is considered in its simplest form, but it can be mapped to more sophisticated parameters that we can attach to the association. We don't do it here to preserve the simplicity and clarity of the example.

We add a UMLScore Message element to Figure 1 and associate it to the Channel element. We could also omit this element in the figure and simply include its semantics in the mapped code for the Listen association.

On the other hand, a Component element is associated to the Server element with a Hosting element specifying their association. Such hosting association semantics is provided through its mapping to a portion of code that makes the associated server process the messages delivered from the channel, and look for calls targeting the Component interface.

Thereby, the behavioural aspect of the hosting semantics is specified in a code fragment and it will depend on the delivered messages represented by the Message element. The listening process (Server) will

invoke the requested method of the hosted component.

This was a very simplified hosting configuration. For a real world hosting, the mapped code to the Hosting element must take into consideration all the attributes of the elements associated to Hosting in Figure 1. Such a code would be complicated, and it must be done once for all as a part of the formal semantics of the involved UMLScore elements.

We notice here that the mapped code to an element processes non static values handled as parameters that govern its execution. This entails that the mapped code will take into consideration the state of the system and the value of different parameters provided as attribute values of related elements.

This huge effort of coding UMLScore elements is tedious, but it's worth it because it needs to be done only once, and it will be of great help in a standardization endeavour to define a new modelling language that joins the visual aspect of design to rigorous semantics, flattening the road for full blown executable models.

6. Conclusion

Although, UML is already suffering from a large vocabulary, we expect that augmenting it with a set of basic elements will improve the situation with respect to semantics precision and expressiveness. Furthermore, we keep in mind the fact that the value of such a project will depend on its potential to become a standard, and it should rather be lead in the context or with the help of a well known consortium that promotes standards. The huge effort required to take to completion this ambitious project should be considered in light of the important expectations behind it. We believe that the accomplishment of this project can give a good boost to the MDE approach and provide answers to recurring questions about UML rigorousness and its semantics dimension.

As future work, a direct mapping from a consistent and autonomous (self-sufficient or self-dependent) UMLScore subset to a chosen programming language is planned to provide a concrete proof of concept. This work will also flatten the road towards a new approach for executable UML.

7. References

[1] Abd-Ali J., El Guemhioui, K. "An MDA-Oriented .NET Metamodel", 9th IEEE International Enterprise

Distributed Object Computing Conference (EDOC 2005), Enschede, The Netherlands, 2005

[2] Kurtev, I. "Metamodels: Definitions of Structures or Ontological Commitments", Workshop on TOWERS of models, collocated with TOOLS Europe 2007

[3] Object Management Group. MDA Guide Version 1.0.1. OMG, 2003. <http://www.omg.org/docs/omg/03-06-01.pdf>

[4] OMG, Meta Object Facility (MOF) Specification. <http://www.omg.org/spec/MOF/2.0>

[5] OMG, UML Profile for EDOC. Available at http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML_for_EDOC

[6] OMG. Metamodel and UML Profile for Java and EJB Specification. February 2004. Version 1.0, formal/04-02-02.

[7] Security requirement with UML 2.0 profile. Available at <http://portal.acm.org/citation.cfm?id=1130988>

[8] Selic, B. "The Pragmatics of Model-Driven Development", *IEEE Software*, Vol. 20, No. 5, 2003

[9] UML profile for CORBA and CORBA Component Model. An OMG specification available at: http://www.omg.org/technology/documents/formal/profile_ccm.htm

[10] UML 2 Semantics Project, Queens University. <http://research.cs.queensu.ca/~stl/internal/uml2/>