UML Semantics Representation of Enterprise Modelling Constructs

Hervé Panetto
CRAN CNRS UMR 7039, France, herve.panetto@cran.u-nancy.fr

Abstract: Enterprise modelling contributes to understand enterprise structure by providing an explicit description of enterprise processes. Among many key issues in an engineering project, formalisation appears to be a suitable technique to check the global consistency between all the various specifications a system is intended to cover. This paper deals with the use of UML semantics representation by means of stereotypes and OCL invariant formalisation to cope with a global consistency of the UML definition.

1 INTRODUCTION

Enterprise modelling contributes to the understanding of enterprise structure by providing an explicit description of enterprise processes, which could help in performance measurement and improvement to make the best possible decision by the enterprise managers. Among many key issues in an engineering project, formalisation appears to be a suitable technique to check the global consistency between all the various specifications a system is intended to cover. Applying that within the enterprise modelling framework, leads to the formalisation of some existing enterprise standards such as CIMOSA (AMICE, 1993, Vernadat, 1998) in order to provide them with refutable foundations.

Our approach is based on the UML (2001) meta-modelling of CIMOSA constructs (Panetto, et al, 2000) and, more generally, of the European Pre-Standard ENV 12204 (CEN, 1995) constructs, in order to establish enterprise constructs described with a common language, UEML (Unified Enterprise Modelling Language) (Kosanke, 1999, UEML IST TN, 2002), which

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35621-1_43

K. Kosanke et al. (eds.), *Enterprise Inter- and Intra-Organizational Integration*

formalises, not only their definitions and their relationships, but also the constraints they have to meet in order to gain semantics. The first section outlines the formalisation requirements in enterprise modelling and illustrates the UML modelling of enterprise constructs. The second section illustrates the semantics approach defined in the UML standard. The third section shows the UML semantics representation of some UEML constructs. Conclusion and prospects are discussed in the last section.

2 UEML CONSTRUCTS

The European Pre-Standard ENV 12204 contains definitions, descriptions and detailed attributes of the common constructs (IFAC/IFIP, 2001) extracted from enterprise models such as CIMOSA (Fig. 1), GERAM, GRAI, ... and the relationships between these (Fig. 2).

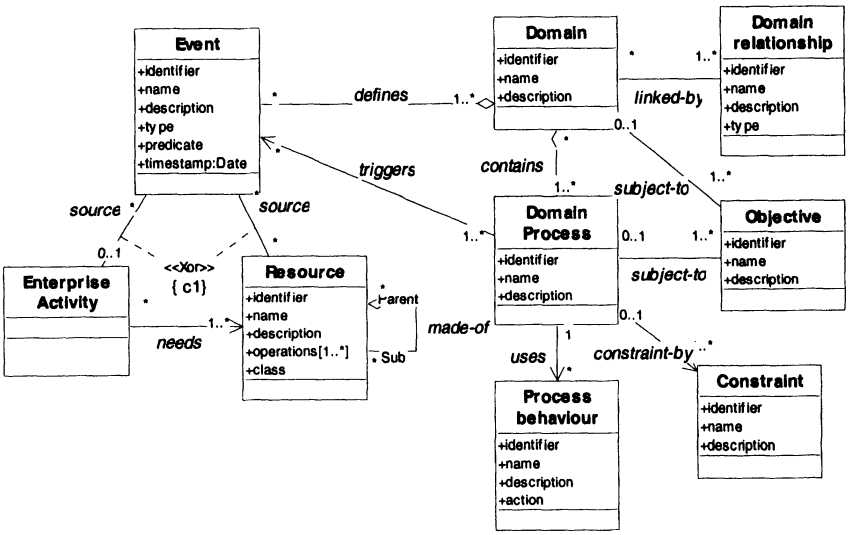

Figure 1: Part of the CIMOSA constructs (Panetto, et al, 2000)

These constructs to be standardised suffer from the lack of a semantic foundation for formally verifying their use in the scope a particular modelling process. Indeed, these constructs are promoted without any ability to check their conformance with the user's requirements. Moreover, their instantiation within a particular enterprise model is not guaranteed to respect some enterprise constraints and properties. Formalisation of these constructs (including enterprise properties) is expected to cope with these two key issues in enterprise models verification.

The practical issues in formalising UEML constructs aim to meta-model them using class diagrams from UML and to formalise constructs constraints and relationships using the OCL (Object Constraint Language) as defined in UML standard.

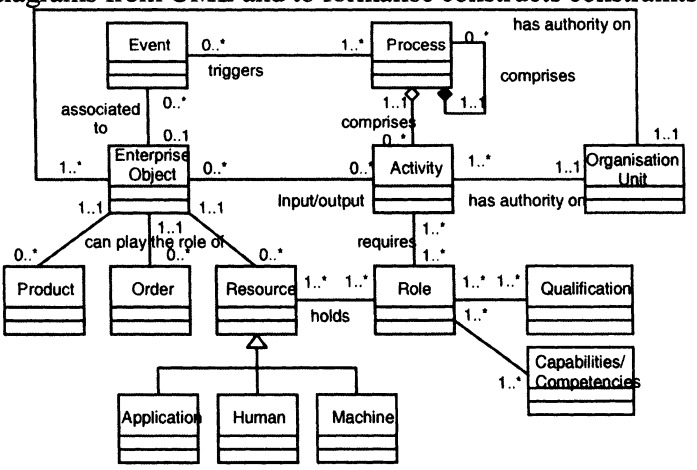

Figure 2: Selected constructs and concepts from ENV 12204:1995 (IFAC/IFIP, 2001)

The formal quality of the system model can be reached by the quality of a formal modelling language. A model is a representation used to formalise a system with semantics while a meta-model is a model used to formalise another model with semantics. Indeed, Gödel's second incompleteness theorem states that any formal system that is interesting enough to formulate its own consistency can prove its own consistency if and only if it is inconsistent. This means that a model cannot be formalised by itself, but only by a higher-level meta-language.

Such meta-languages manipulate basic concepts of the formalised model to help its understanding. For example, the Fig.3 and Fig.4 represent meta-models of relational model and UML with respectively sNets (sNets Formalism, 1998) and MOF, (1997).

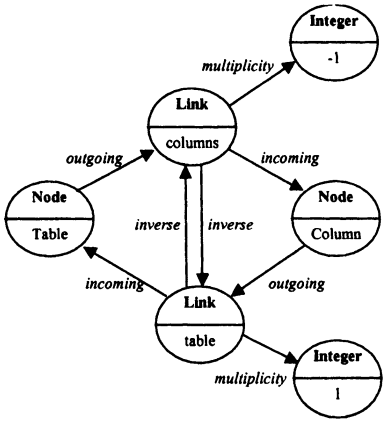

Figure 3: Meta-model of the relational model with sNets

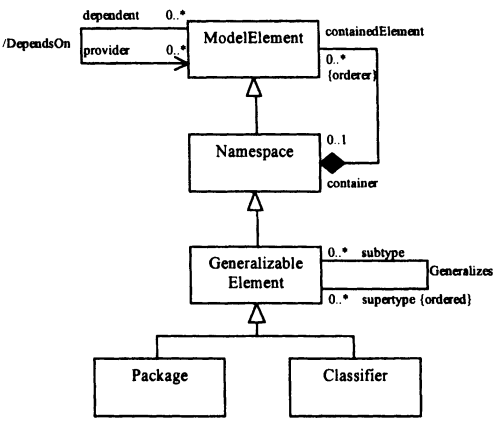

Figure 4: Part of the UML meta-model with MOF (Meta Object Facilities)

As has already been done for products data definitions in ISO STEP 10303 application protocols (ISO, 1994), constructs can be defined as object classes or template structures, which can be assembled to model a system. Due to the complexity and the variety of models, their coherent integration needs the definition of a limited set of constructs that can be applied for their formal representation. A construct is a generic object class or template structure, which models a basic concept independently of its use (ISO, 1994). As an example, the *"if-then-else"* control structure is a particular construct of programming languages.

The identification of constructs consists of meta-modelling the models using a formal meta-language to define the basic concepts that they use. Integration of different models is done by analysing their respective constructs by mean of their meta-model and their definitions, in order to build new constructs that merge their respective capabilities. The objective is *not here to build a new modelling language* but only to formalise constructs that *help to understand the common concepts* of different modelling languages, their relationships and constraints.

3 UML

The Unified Modeling Language (UML), an OMG standard, is a widely adopted and used modelling language. The UML emerged from the unification that occurred in the 1990s following the "method wars" of the 1970s and 1980s. Even though the UML evolved primarily from various second-generation object-oriented methods (at the notation level), the UML is not simply a third-generation object-oriented modelling language.

The UML is defined by nine languages. In this work, we use the Class Diagram, which defines objects with their attributes, their operations and the relationships between them. Research work in progress aims to use state-transition diagrams to describe the dynamic behaviour of operations. Moreover UML standard specifies the Object Constraint Language (OCL), an expression language that enables one to describe constraints on object-oriented models. OCL is a formal constraint language based on first order predicate logic. It formalises constraints, which are restrictions on a model or a system. Thus, a constraint states, «this should be so». Constraints are attached on every modelled item. This is called the *context* of the constraint.

There are three types of constraints:

- An *invariant* formalises a condition that must always be met by all instances of the class
- A *precondition* to an operation is a restriction that must be true at the moment that the operation is going to be executed.

- A *postcondition* to an operation is a restriction that must be true at the moment that the operation has just ended its execution.

As a modelling language, UML can be used to meta-model enterprise modelling standards to ensure their integration through unique and coherent definitions.

In order to extend its meta-model, UML provides an expendability mechanism through the definition of so called "Profiles". A profile contains one or more related extensions of standard UML semantics. These are normally intended to customise UML for a particular domain or purpose. They can also contain data types that are used by tag definitions for informally declaring the types of the values that can be associated with tag definitions. In effect, these extension mechanisms are a means for *refining* the standard semantics of UML and do not support arbitrary semantic extension. They allow the modeller to add new modelling elements to UML for use in creating UML models for process-specific domains such as enterprise models. *Constraints* can also be attached to any model element to refine its semantics.

4 CONSTRUCTS SEMANTICS

The construct semantics representation deals with defining an UML Profile using the extensibility mechanisms of UML, which allow modellers to customise UML for specific domains. Profiles are used for:

- Defining new meta-classes (stereotypes),
- Defining new meta-attributes (tagged values),
- Defining new meta-associations (tagged values, referencing to other model elements),
- Defining new constraints.

The UML standard already defines 8 profiles: Scheduling, performance and time, Enterprise Distributed Object Computing, CORBA, EJB, Software Process Engineering Management, EAI and QoS and fault tolerance. A profile defines a projection of a reference meta model and provides a mechanism to define facets that can be applied to model elements and combined.

Moreover, as the UML specification relies on the use of well-formedness rules to express constraints on model elements, this profile uses the same approach. The constraints applicable to the profile are added to the ones of the stereotyped base model elements, which cannot be changed. Constraints attached to a stereotype must be observed by all model elements branded by that stereotype. If the rules are specified formally in a profile (for example, by using OCL for the expression of constraints), then a modelling tool may

be able to interpret the rules and aids the modeller in enforcing them when applying the profile.

As an example, the "Enterprise Object" construct is defined as an "Enterprise Object" stereotype, based on the UML "Class" meta class (Fig. 5).

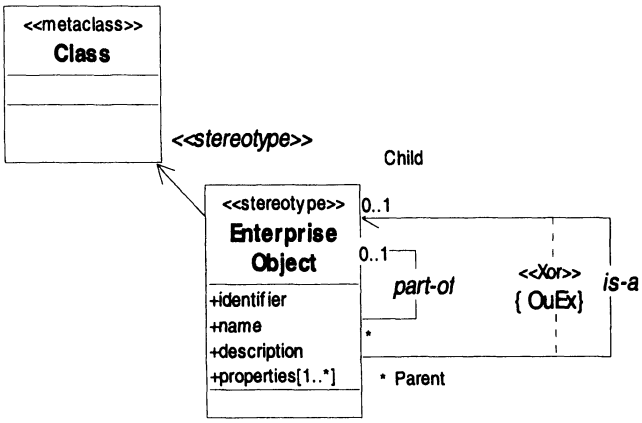

Figure 5 : Stereotype definition

That stereotype defines that an "Enterprise Object" could be "part-of" another "Enterprise Object" and that an "Enterprise Object" could be a subclass ("is-a" relationship) of another "Enterprise Object". That stereotype defined, also, tagged values such as identifier, name, description and a set of properties.

An invariant constraint represented by well-formedness rules ensures the consistency in the relationships between modelled elements. Such a formal rule could be:

```
context EnterpriseObject
  inv : self.partOf->forall(p | p <> self)  (1)
  inv : self.properties->forall(p | p.stereotype.name = "Enterprise
Object"
 implies not self.partOf->exists(p | p = self) (2)
```

That invariants state that (1) a particular "Enterprise Object" could not be part of itself, and (2) a particular "Enterprise Object" could not be itself included in the set if its own properties.

Instantiation of that stereotype in a particular model aims at defining a stereotyped class that should meet the previous invariant formalisation. For example, Fig. 6 shows the "Client

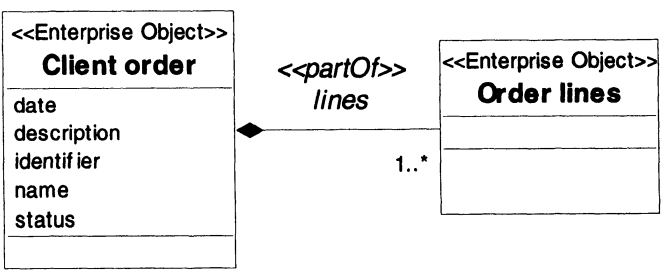

Figure 6: A Stereotype instantiation

order" object and the "Order lines" object as instances of the "Enterprise Object" stereotype. The "partOf" relationship between these two "Enterprise Objects" comes from the "part of" relationship defined in Fig. 5.

The same invariant as defined previously is applied to that model ensuring its consistency. In particular, that rules avoid the definition of a relationship between two "Client orders". The only authorised relationship is the "partOf" composition between a "Client order" and one or more "Order lines".

5 CONCLUSION

There is a need to provide a semantic foundation for formally verifying its use in the scope a particular modelling process. UML provides extensibility mechanisms able to formalise enterprise modelling constructs. Constraints are also expressed and could be used, by engineering tools, to aid the modeller in ensuring the global consistency of its model. These rules are expressed in the generic view of the model. There are tools that can interpret these rules using class instances values for particular models. In order to be able to verify them in partial model (for domain-based models), work is in progress to translate them into the B language (Abrial, 1996), which allows properties proofs, based on non refutable mathematical theories.

6 REFERENCES

Abrial J.R. (1996), *The B Book: Assigning Programs to Meanings*. Cambridge Univ. Press.

AMICE, (1993), *CIMOSA: Open System Architecture for CIM*, Springer-Verlag.

CEN, (1995), European Pre-Standard ENV 12204, *Advanced Manufacturing Technology, Systems Architecture, Constructs for Enterprise Modelling*, TC 310/WG1 (currently under revision).

IFAC/IFIP Task Force (2001), "*Architectures for Enterprise Integration*", UEML Interest Group.

ISO 10303, (1994), *STEP, Standard for the Exchange of Product Model data*, TC 184 SC4.

Kosanke, K. Vernadat F.B. Zelm, M. (1999), *CIMOSA enterprise engineering and integration*, Computers in Industry, Volume 40, Issues 2-3, Pages 83-97.

MOF Specifications, (1997), Joint Revised Submission, OMG Document ad

Panetto H. Mayer F. Lhoste P. (2000), *Unified Modeling Language for meta-modelling: towards constructs definitions*, Proceedings of ASI'2000, ISBN 960-530-050-8.

sNets Formalism, (1998), technical report, LRGS, Université de Nantes.

UML 1.4, (2001), *Unified Modeling Language*, Object Management Group standard.

Vernadat, F.B. (1998), *The CIMOSA languages*, Handbook of Information Systems. P. Bernus, K Mertins and G. Schmidt (Eds.), Springer-Verlag