Checking semantics in UML models

Susana Kahnert – Pablo Fillottrani
Depto. Ciencias e Ingenieríade la Computación
Universidad Nacional del Sur
Av. Alem 1253 – Bahía Blanca, Argentina
e-mail: {sak,prf }@cs.uns.edu.ar

Introduction

Modeling is one of the most crucial activities during all the phases in any software development life cycle. Typically, system development is a very complex task, and working with models helps to handle this complexity in an organized way, allowing us to reason about the properties that entities possess. In software engineering, notation, techniques and methodology for object-oriented model building has been lately the focus of active research work [8]. In the case of object-oriented software development, models are composed by a number of communicating and well delimited elements. Although such models are sometimes harder to develop, they are easier to understand, and simpler to maintain and modify [5].Thus, reusability of elements among models is enhanced. In order to maximize these properties, we need tools that support the process of object- oriented development by serving as repository of previously asserted knowledge, checking the integrity of the model and maintaining the different views that form each model. Consequently, it is expected this automatic control will reduce the manual, error-prone work of maintaining model consistency across all life cycle phases.
UML, Unified Modeling Language is a graphical language for visualizing, specifying, constructing and documenting the elements of a software intensive system [6].UML provides notation for expressing the model, in the form of graphic and text elements. Attached to these elements, there is a semantic interpretation that attempts to capture the meaning of the model, and it is represented by the constraint mechanism. Constraints are one of the three extensibility mechanisms that UML introduces, although currently the language for expressing them is not standard (natural language or OCL[10] can be used for this purpose). Supporting tools for object oriented development in UML should not only provide a graphic editor for the notation, but also help to ensure the coherence of its semantics aspects. This research line attempts to formalize the consistency check process in UML models.

Constraints and Rules Checkings - General concepts

A *well-formed model* is a correctly constructed model. UML specification does not formally define this concept. Constructs are defined by means of UML notation, natural language and well-formedness rules. In[4] we have classified semantic information, according to its genericity, as Rules and Constraints. We can define now a ***well-formed model*** as one that satisfies all the predefined and model-specified rules and constraints. Such a model has meaningful semantics. A model that is not well formed is called ill-formed [9]. A metamodel describes the contents of a well-formed model; it does not make sense to ask for the semantics of an ill-formed model.
Much of the time, however, models under development are not well formed. They are incomplete and possibly inconsistent. The UML metamodel describes correct, well-formed models, and leaves it to the tool makers to decide what kind of semantic support to give to ill-formed (or “work in progress”) models.

In order to define the checks, we need to establish precisely what do "predefined and model-specified rules and constraints" mean within the UML metamodel.

- ***Constraints*** are semantic information attached to an element and represented as an expression. A constraint is an assertion, not an executable mechanism. It indicates a restriction that must be enforced by correct design of a system. Certain constraints are predefined in the UML, others may be defined by modelers. *Predefined (or standard) constraints* are those common constraints that have names to avoid writing a full statement each time they are needed. For example, the constraint xor between two associations.[9]

In the metamodel a Constraint is a boolean expression on an associated Model Element(s) which must be true for the model to be well-formed. [9]. Constraints are classified as:

invariants : constraints that must be attached to a set of classifiers or relationships. Its conditions must be true at all times when no operation is incomplete.

preconditions : constraints that must be attached to an operation. Its conditions must be true after the invocation of the operation.

postconditions : constraints that must be attached to an operation. Its conditions must be true for the invocation of the operation.

- ***Rules*** are general semantic conditions the model as a whole should satisfy, in order to be well-formed.[2]. *Predefined rules* are those required by the static and dynamic semantics of UML constructs. The static semantics of a language define how an instance of a construct should be connected to other instances to be meaningful; the dynamic semantics define the meaning of a well-formed construct. In UML metamodel, the main language constructs are specified as metaclasses in the metamodel. Their static semantics (or well-formedness rules), except for multiplicity and ordering constraints, are defined as sets of invariants of an instance of the corresponding metaclass. The rules thus specify constraints over attributes and associations defined in the metamodel. The dynamic semantics (or meanings) of the constructs are defined using natural language.

Model-specified rules allow characterizing the semantics of a particular model, for example "all attributes must be private", or "multiple inheritance is not allowed". The problem is that the current version of UML specification does not take into account explicitly this kind of rules. It is up to the tool maker to define a suitable way of modeling these rules.

Constraints and Rules in Class Diagrams

The checking process comprises two levels:

- ***Definition check****:* performed automatically on edition. It is responsible for finding out whether the elements involved are well defined according to UML predefined rules, as well as if their type and, in case of binary associations, their navigability are of the proper kind.. The tool should warn when the elements are not yet been defined.
- ***Semantic check***: performed on user demand. According to the point where it is called from, it may control if a constraint is satisfied, or if it satisfaction is feasible, or just if it is consistent with other constraints in the model.

Class Diagrams play an important role in two major views that represent different aspects of a system:

- Model management view: models the organization of the model itself. A model comprises a set of packages that hold model elements such as classes, state machines and use cases. Model specified rules may be considered as constraints attached to the model element Package. They will be checked for satisfaction.
- Static view: models concepts in the application domain, as well as internal concepts invented as part of the implementation of an application. It does not describe the time dependant behavior of a system. The main constituents of the static view are classes and their relationships.

At this stage of the research, we are considering semantic checks over Class Diagrams, Static View. We have implemented controls in order to check consistency among class invariants, pre and post conditions attached to operations, and constraints attached to associations, analyzing how consistency may be compromised when model elements are introduced or modified during system development. For allowing automatic verification these controls had to be expressed in a formal language.

In order to achieve this goal, we have defined a reduced constraint specification language which prevents our attention from focusing on complex constraint checks, but instead on describing how model semantics is maintained through the expected evolution of the model.

Our work is intended to provide controls over user-defined semantics. At this stage, however, it relies on the assumption of a shared ontology supporting constraint definition; it is not possible to find inconsistences unless constraints with similar or contradictory meanings can be compared.

Further research should be devoted to consider the consequences of more expressive constraint specification languages, as well as to provide support for a coherent user-defined ontology. Semantic verification of other UML diagrams and constraint satisfaction are also expected to be addressed.

References

[1] EVERETT, J., BORROW, D., STOLLE, R., CROUCH, R., DE PAIVA, V., CONDORAVDI, C., VAN DEN BERG, M., AND POLANYI, L. Making Ontologies Work for Resolving Redundancies Across Documents. *Communications of the ACM 45 2 (February,2000),55 –60.*

[2] Fillottrani, P., Estevez, E., and Kahnert, S. Applying Logic Programming Techniques to Object-Oriented Modeling in UML.Proceedings of the 14th. International Conference on Software Engineering and Knowledge Engineering (SEKE'02) (2001),228 –235.

[3] Georget, Y., Codognet, P., and Rossi, F. Constraint Retraction in CLP(FD):Formal Framework and Performance Results.CONSTRAINTS: An international journal, 4 1 (1999). Kluwer.

[4] Gruninger, M., and Lee, J. ONTOLOGY:Applications and design.Communications of the ACM 45 2 (February,2000).

[5] I. Jacobson and M. Christerson and P. Jonsson and G. ¨ Overgaard Object-Oriented Software Engineering: A Use Case Driven Approach Addison-Wesley,1997.

[6] Ivar Jacobson, G. B., and Rumbaugh, J. The Unified Modeling Language Reference Guide ACM Press,Addison-Wesley,1999.

[7] Ivar Jacobson, G. B., and Rumbaugh, J. The UnifiedModeling Language UserGuide ACM Press,Addison-Wesley,1999.

[8] Ivar Jacobson, G. B., and Rumbaugh, J. The Unified Software Development Process ACM Press,Addison-Wesley,1999.

[9] OMG Unified Modeling Language Specification Version 1.4 Object Management Group,Inc., September,2001.

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