Semantic Links and Co-Evolution in Object-Oriented Software Development

Ragnhild Van Der Straeten
Vrije Universiteit Brussel
System and Software Engineering Lab
Pleinlaan 2, 1050 Brussel, Belgium
rvdstrae@vub.ac.be

Abstract

This research focuses on the problem of the semantic linking and co-evolution of the different design diagrams and models of an object-oriented software application. The blueprint of an object-oriented software application consists mainly of models drawn in a modeling language. The state-of-the-art modeling language in object-oriented software development is the Unified Modeling Language (UML). Current UML case-tools do not support the linking and co-evolution of the different models in an adequate way. This results in less maintainable, reusable and understandable models which are nowadays only used as a kind of documentation. We propose to develop a formal framework to support the linking of the different diagrams and models within the software development life cycle (SDLC). For this purpose, we investigate different fragments of first order logic such as Description Logic and Modal Logic. To support co-evolution of the design models in a semi-automatic way we investigate the query capabilities of these logic families.

1. Problem

The major activities in software development are the determination and specification of requirements the system should meet, the architectural design which is a description of the system in terms of its modules, the design of the system and at last the implementation of the system. Nowadays, the de-facto modeling language used in object-oriented development is the Unified Modeling Language (UML) [2]. The visual representation of this language consists of several diagram types. All major UML case-tools support different UML diagram types. Several levels of abstraction are used in the SDLC. Within one abstraction level several kinds of UML diagrams are used, e.g. class diagrams, state diagrams, interaction diagrams. We will call one such layer a model of the application. Even in state-of-the-art case-tools the different types of diagrams are not linked or only linked on a syntactic level. For example, in the Rational Rose [1] case-tool, a state diagram is coupled to the corresponded class by clicking on the class with the right mouse button in the browser. However, no checks are done if an event or action or guard is introduced.

What is not supported in the case-tools is the semantic linking between the different diagrams of a model and between the different models. Designing software applications is also a group activity. Different people are designing different parts of the software application. The resulting designs should be semantically consistent. This implies that reasoning about the different diagrams and links between those diagrams is necessary.

A second problem is that without any linking of the diagrams, keeping track of the changes and checking consistency of the different diagram types and models becomes very difficult, if not impossible. This is because of the additive character and the refinement process inherent to the design and maintenance of software. As the development of a software application progresses, the models get refined and information is added or deleted resulting in new more detailed models.

As a result of these problems, the different models are less maintainable, reusable and understandable. Nowadays models are only used as a kind of documentation. In the extreme case, changes to the software application are not reflected in the different models and these models become obsolete.

2. Scope, Goals and Research Question

To tackle the problem as presented in section 1, we propose to develop a formal framework to support the object-oriented software development life cycle. In this research project we will focus on the design phase of the software development life cycle. Nowadays, the most common software development technique is object orientation.

The state-of-the-art object-oriented modelling language is UML. As such, we will also limit this research to UML diagrams. However, we are confident that our approach is also applicable to other object-oriented modelling languages.

Our research hypothesis is: *Using decidable fragments of first order logic to express the different UML diagrams, enables the semantic linking of the different diagrams and models and enables the support for co-evolution which can be semi-automated, enhancing the reusability, maintainability and understandability of the design of the software application and of the software application in general.*

Our goal is to develop a formal framework to support the semantic linking and the co-evolution of the different diagram types within a model and of the different models. The advantages of a formal framework are:

- reasoning capabilities are provided about diagrams and models in the SDLC,
- through the semantic links the co-evolution of the models is more guaranteed,
- reuse and adaptability of software is improved,
- the understandability of the software design increases.

This formal framework will be the basis of an intelligent modelling tool supporting the software developer in his development and maintenance task of object-oriented software applications. We believe that adequate tools for the development of object-oriented software have to be simple and quite powerful and as such must have a formal framework as their foundation.

3. Our Approach

Our requirements are: it must be possible to reason about diagrams, to keep track of the several links between the diagram types and between the models. We discern two different views on the software design: the *horizontal* and the *vertical* view. The horizontal view concerns the linking of the different diagram types of one model. The vertical view concerns the linking of the different diagrams of successive models. The following sections will deepen these views.

3.1. Horizontal View

We believe that decidable fragments of first order logic can be used as a formal framework to ease the design tasks of the developer. Our approach is to translate the different UML diagrams in logics of the Description and Modal logic family [14]. These are both decidable fragments of first order logic with automatic reasoning procedures. We will especially focus on the dynamic part of UML. Next to the different UML diagrams, we will also translate the constraints written down in the Object Constraint Language [4] into one of these logics. The Object Constraint Language (OCL) comes with the UML specification. UML modelers can use OCL to specify application-specific constraints in their models. These constraints typically specify invariant conditions that must hold for the application being modeled.

Because the different diagram types are translated in formalisms which can be compared with each other [5], this allows to link the different diagram types semantically. The translation of these diagram types to a specific logic will be automated.

3.2. Vertical View

Once all the different diagram types are translated in a formalism, we still need to keep track of the co-evolution of the different models. This boils down in supporting the co-evolution of the different diagrams in the different models. This is also important in the context of the Model Driven Architecture (MDA) drive of the OMG [3] - to provide mechanisms for allowing developers to come up with platform independent UML models that can be translated into specific models later.

In the context of the vertical view, we make the following observation. When designing applications, a lot of problems reoccur. For example the mapping of classes onto a three-tier architecture. As a solution, a kind of standard solution template can be used in almost all cases. Thus, a problem and solution that is applicable in a particular context, can in most cases also be applied in other contexts. Such recurring ideas are called *patterns* in software engineering. Different kinds of patterns exist in software engineering. For example, analysis patterns [12] were developed capturing solutions to problems by introducing several alternatives from a conceptual point of view. On the design level, patterns [9] are an attempt to collect and catalog the smallest recurring architectures in object-oriented software systems. All those patterns consist of a problem statement, some constraints on the problem, the presentation of a widely accepted solution and discussion of the consequences of that solution. In our opinion a modeling tool should be able to register some of the recurring design actions undertaken by the developer. After the developer has executed these actions some times in different contexts, the tool could provide some suggestions in yet another context to the developer concerning this recurring problem. As a consequence even co-evolution steps can be seen as patterns, we call this *co-evolution patterns*. These co-evolution patterns are in fact specific model transformations.

To support co-evolution and co-evolution patterns, we need the query capabilities of the Description Logic and Modal Logic. In the next sections, we will discuss the choice of Description and Modal Logics.

3.3. Description Logic

There are several reasons for choosing the Description Logic family as formalism. Many concepts in object-oriented programming languages originate from frame-based systems. Description logics are based on the same ideas as semantic networks and frames but provide these with exact semantics. DLs have already proven their use in knowledge representation and reasoning. A DL consists of a description language, a knowledge specification language and automatic reasoning procedures. The most important aspect is these automatic reasoning procedures. They allow to reason about the consistency of knowledge bases. The question remains if state-of-the-art DLs are expressive and powerful enough to express and reason about different diagram types, constraints and the coupling of the different diagram types. Due to their semantics, DL's are more suited to express the static structure of the software application. In [7] UML class diagrams are translated to the description logic $\mathcal{DLR}$.

3.4. Modal Logic

Modal logic [13] deals with description languages for talking about various kinds of relational structures. Modal languages have a propositional flavour, a simple and uniform semantics and effective reasoning systems. Modal languages are very well suited to describe for example graphs. While DL's emphasize concepts, modal logic put more emphasis on the relations between different concepts or worlds. Together with their decision methods, this results in a powerful combination. In contrast to DL's, due to their semantics, modal logics are more suited to express the behaviour of the software application.

3.5. Query Capabilities

First of all, to support co-evolution, we have to keep track of the changes between two diagrams (i.e. the Δ's). A library of change operations is needed. Change operations on a UML class diagram for example, include: add/delete class, association, aggregation, inheritance, attribute and method. These operations correspond to change operations in the underlying logic. To be able to find out if some links are present in the design models or that specific changes were executed in a certain design model resulting in a successive design model, we should be able to query the logical statements. Query languages are defined for the different logics and should be investigated in more detail.

4. Plan and Contribution

First of all the formalization of the different UML diagrams in terms of fragments of first order logic needs to be done. This translation will be automated. In a second phase, different kinds of links need to be defined between the diagrams of different models. The necessary queries have to be constructed to link the diagrams and as such the models. Also different model transformations and as a subset of this, co-evolution patterns need to be defined to support co-evolution.

This research will be validated first of all by a case-study. As a case, we will take the specifications of a real world project about broadcasting TV programs. This project has been carried out by one of our industrial partners [8]. This project is about the planning of programs and spots. At the end, a schedule is released that is ready for transmission. Starting from the specifications, high-level models can be constructed. Taking into account several remarks and refinements of the software users, the models change and get refined. This project is already one year old, in the mean time, the software and also the models are still evolving. Some co-evolution tasks have been executed on the models and the code. As such this case-study can act as an excellent proof of concept.

A second validation of this research will be carried out by the international research community. We will submit papers about this research to international conferences.

5. Related Work

In [10] and [11], formalization of UML is discussed. In these papers, the focus is on making precise the areas of ambiguity and/or missing semantic details within the current UML *meta-model* using a denotational approach. The emphasis is put on the development of a meta-modelling facility, consisting of a meta-modelling language, a meta-modelling tool and a meta-modelling method. The aim is to support the precise definition of families of modelling languages.

Other related work is the formalization of UML within rewriting logic, implemented in the Maude system [17]. The emphasis of this work is on the evolution of the language metamodel. Apparently they are also searching for semantic properties of these models in order to be rigorously tested.

In [7] the formalization of class diagrams in terms of the DL $\mathcal{DLR}$ has been investigated as mentioned in section 3. This approach also provides automated reasoning techniques based on sound and complete algorithms. But only class diagrams are considered in this work.

Within the automated software community, the automatization of evolving object-oriented designs is investigated in [15] and [16]. However the focus here is on refactorings - behavior-preserving program transformations that automate many design level changes. The focus in this work is on the impact of these changes in the implementation of the software application.

Also in the object-oriented database world some related work can be found. In [6] object-oriented database schema transformations are defined. Object-oriented schema transformations can be drawn from these database schema transformations, especially for the edits performed on a class diagram.

6. Conclusion

This research project deals with the semantic linking of different UML diagram types and models. A formal framework will be build based on decidable fragments of first order logic, underlying an intelligent tool supporting the software developer in designing and maintaining object-oriented software applications. Due to this, the understandability, reusability and maintainability of the software application will increase. The validation of this research will be done by the application of these ideas to a real world case-study and by the international research community. This research is still very preliminary. Now that we have stated our research hypothesis and done a literature study, it is time to focus and to start to make our ideas concrete.

References

[1] *Rational website*. http://www.rational.com.

[2] The OMG Unified Modeling Language Specification. The Object Management Group http://www.omg.org.

[3] *The Model Driven Architecture*. http://www.omg.org/mda/.

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

[5] C. Areces. *Logic Engineering. The Case of Description and Hybrid Logics*. PhD thesis, ILLC University of Amsterdam, 2000.

[6] Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. Semantics and implementation of schema evolution in object-oriented databases. *ACM SIGMOD Record*, 16(3):311–322, 1987.

[7] Andrea Calí, Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Reasoning on UML Class Diagrams in Description Logics. In *Proc. of IJCAR Workshop on Precise Modelling and Deduction for Object-oriented Software Development (PMD 2001)*, 2001.

[8] Wim Codenie, Koen De Hondt, Patrick Steyaert, and Arlette Vercammen. From Custom Applications to Domain-Specific Frameworks. *Communications of the ACM*, 40(10):70–77, 1997.

[9] Gamma E., Helm R., Johnson R., and Vlissides J. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[10] A. Evans and S. Kent. Core Meta-Modelling Semantics of UML: The pUML Approach. In *Proceedings of UML'99*. IEEE Computer Society Press, October 1999.

[11] Andy S. Evans. Reasoning with UML class diagrams. In *Second IEEE Workshop on Industrial Strength Formal Specification Techniques, WIFT'98, Boca Raton/FL, USA*. IEEE CS Press, 1998.

[12] Martin Fowler. *Analysis Patterns, Reusable Object Models*. Addison Wesley, 1997.

[13] Blackburn P., de Rijke M., and Venema Y. *Modal Logic*. Cambridge University Press, 2001.

[14] R. Van Der Straeten. Using Description Logic in Object-Oriented Software Development. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.

[15] L. Tokuda and D. Batory. Automated Software Evolution via Design Pattern Transformations. In *Proceedings of the 3rd International Symposium on Applied Corporate Computing, Monterrey, Mexico*, 1995.

[16] Lance Tokuda and Don S. Batory. Evolving object-oriented designs with refactorings. In *Automated Software Engineering*, pages 174–, 1999.

[17] A. Toval and J.L.F. Alemán. Formally modeling UML and its evolution: a holistic approach. In S.F. Smith and C.L. Talcott, editors, *Formal Methods for Open Object-based Distributed Systems IV*, pages 183–206, 2000.