Lecture Notes in Computer Science 1618

Jean Bézivin Pierre-Alain Muller (Eds.)

The Unified Modeling Language

«UML»'98: Beyond the Notation

First International Workshop
Mulhouse, France, June 1998
Selected Papers

Springer

Jean Bézivin Pierre-Alain Muller (Eds.)

The Unified Modeling Language

«UML»'98: Beyond the Notation

First International Workshop
Mulhouse, France, June 3-4, 1998
Selected Papers

Springer

Series Editors

Gerhard Goos, Karlsruhe University, Germany
Juris Hartmanis, Cornell University, NY, USA
Jan van Leeuwen, Utrecht University, The Netherlands

Volume Editors

Jean Bézivin
Université de Nantes, Faculté des Sciences et Techniques
2, Rue de la Houssinière, B.P. 92208, F-44322 Nantes Cedex 3, France
E-mail: jean.bezivin@sciences.univ-nantes.fr

Pierre-Alain Muller
ObjeXion Software
5, Rue Gutenberg, F-68800 Vieux-Thann, France
E-mail: pa.muller@essaim.univ-mulhouse.fr

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

The unified modeling language : first international workshop ; selected papers / UML '98: Beyond the Notation, Mulhouse, France, June 3 - 4, 1998. Jean Bézivin ; Pierre-Alain Muller (ed.). - Berlin ; Heidelberg ; New York ; Barcelona ; Hong Kong ; London ; Milan ; Paris ; Singapore ; Tokyo : Springer, 1999
(Lecture notes in computer science ; Vol. 1618)
ISBN 3-540-66252-9

CR Subject Classification (1998): D.2, D.3

ISSN 0302-9743
ISBN 3-540-66252-9 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN: 10705238 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

Table of Contents

UML: The Birth and Rise of a Standard Modeling Notation

Jean Bézivin[1], Pierre-Alain Muller[2]

[1] Laboratoire de Recherche en Sciences de Gestion
Université de Nantes
Faculté des Sciences et Techniques
2, rue de la Houssinière
BP92208
44322 Nantes cedex 3
France
Jean.Bezivin@sciences.univ-nantes.fr

[2] ESSAIM
Université de Haute-Alsace
12, rue des frères Lumière
68093 Mulhouse
France
pa.muller@essaim.univ-mulhouse.fr

Abstract. Officially the Unified Modeling Language UML is a graphical language for visualizing, specifying, constructing and documenting the artifacts of a software-intensive system. For many, UML is much more than that and symbolizes the transition from code-oriented to model-oriented software production techniques. It is very likely that, in a historical perspective, UML will be given credit for the perspectives opened as well as for the direct achievements realized. This introductory paper presents some of the characteristics of the notation and discusses some of the perspectives that have been and that are being opened by the UML proposal.

Introduction

The first few years of the 90s saw the blossoming of around fifty different object-oriented methods. This proliferation is a sign of the great vitality of object-oriented technology, but it is also the fruit of a multitude of interpretation of exactly what an object is. The drawback of this abundance of methodologies is that it encourages confusion, leading users to adopt a 'wait and see' attitude that limits the progress made by methods.

In 1996, the Object Management Group (OMG) put together a task force chartered with defining and approving a notational and meta-model standard for object-oriented analysis and design. The task force was made up of vendors of related tools that initially clustered themselves into four major camps. One of these camps aggregated around the submission originated by Rational Software and promoted the Unified Modeling Language (UML) that Rational built from the OMT, Booch and OOSE methodologies created by the three methodologists (Rumbaugh, Booch, and Jacobson) in its employ. The four proposals were submitted to OMG in January 1997. Other camps noted the absence of support for software development process, business process modeling, and real-time extensions within de UML definition. In March 97, the factions agreed to work closely together to add the capabilities needed for the UML to satisfy their various needs, and in December 1997, the standard was formally adopted.

From the Unified Method to the Unified Modeling Language

The unification of object-oriented modeling methods became possible as experience allowed the evaluation of the various concepts proposed by existing methods. Based on the fact that differences between the various methods were becoming smaller, and that the method war did not move object-oriented technology forward any longer, Jim Rumbaugh and Grady Booch decided at the end of 1994 to unify their work within a single method: the Unified Method. About one year later, they were joined by Ivar Jacobson, the father of use cases, a very efficient technique for the determination of the requirements.

Booch, Rumbaugh and Jacobson adopted four goals:

- To represent complete systems (instead of only the software portion) using object-oriented concepts
- To establish an explicit coupling between concepts and executable code
- To take into account the scaling factors that are inherent to complex and critical systems
- To create a modeling language usable by both humans and machines

The authors of the Unified Method rapidly reached a consensus with respect to fundamental object-oriented concepts. However, convergence on the notation elements was more difficult to obtain, and the graphical representation used for the various model elements went through several modifications.

The first version of the description of the Unified Method was presented in October 1995 in a document titled Unified Method V0.8. This document was widely distributed, and the authors received more than a thousand detailed comments from the user community. These comments were taken into account in version 0.9, released in June 1996. However, it was version 0.91, released in October 1996, which represented a substantial evolution of the Unified Method. The main effort was a change in the di-

rection of the unification effort, so that the first objective was the definition of a universal language for object-oriented modeling, and the standardization of the object-oriented development process would follow later. The Unified Method was transformed into UML (the Unified Modeling Language for object-oriented development). As we are approaching today the version 1.4 of UML [7], the OMG Revision Task Force is already thinking to a future version 2.0. At the same time the notation is now well documented, with a rapidly increasing number of textbooks (e.g. [1], [3], [4], [5], etc.)

Model and Meta-model

The initial effort focused on the identification and definition of the semantics of fundamental concepts - the building blocks of object-oriented modeling. These concepts are the artifacts of the development process, and must be exchanged between the different parties involved in a project. To implement these exchanges, it was first necessary to agree on the relative importance of each concept, to study the consequences of these choices, and to select a graphical representation, of which the syntax must be simple, intuitive, and expressive.

To facilitate this definition work, and to help formalize UML, all the different concepts have themselves been modeled using a subset of UML. This recursive definition, called meta-modeling, has the double advantage of allowing the classification of concepts by abstraction level, by complexity and by application domain, while also guaranteeing a notation with an expressive power such that it can be used to represent itself.

A meta-model describes formally the model elements, and the syntax and semantics of the notation that allow their manipulation. The raise in abstraction introduced by the construction of a meta-model facilitates the discovery of potential inconsistencies, and promotes generalization. The UML meta-model is used as a reference guide for building tools, and for sharing models between different tools.

A model is an abstract description of a system or a process - a simplified representation that promotes understanding and enables simulation. The term 'modeling' is often used as a synonym of analysis, that is, the decomposition into simple elements that are easier to understand. In computer science, modeling usually starts with the description of a problem, and then describes the solution to the problem. These activities are called respectively 'analysis' and 'design'.

The form of the model depends on the meta-model. Functional modeling decomposes tasks into functions that are simpler to implement. Object-oriented modeling decomposes systems into collaborating objects. Each meta-model defines model elements, and rules for the composition of these model elements.

The content of the model depends on the problem. A modeling language like UML is sufficiently general to be used in all software-engineering domains and beyond - it could be applied to business engineering, for example.

A model is the basic unit of development; it is highly self-consistent and loosely coupled with other models by navigation links. Dependent on the development process in use, a model may relate to a specific phase or activity of the software lifecycle. A model by itself is usually not visible by users. It capture the underlying semantics of a problem, and contain data accessed by the tools to facilitate information exchange, code generation, navigation, etc. Models are browsed and manipulated by users by means of graphical representations, which are projections of the elements contained in one or more models. Many different perspectives can be constructed for a base model - each can show all or part of the model, and each has one or more corresponding diagrams.

The UML diagrams

UML defines nine different types of diagram:

- Class diagrams
- Sequence diagrams
- Collaboration diagrams
- Object diagrams
- Statechart diagrams
- Activity diagrams
- Use case diagrams
- Components diagrams
- Deployment diagrams

Different notations can be used to represent the same model. The Booch, OMT, and OOSE notations use different graphical syntax, but they all represent the same object-oriented concepts. These different graphical notations are just views of the same model elements, so that it is quite possible to use different notations without loosing the semantic content.

At heart, then, UML is simply another graphical representation of a common semantic model. However, by combining the most useful elements of the object-oriented methods, and extending the notation to cover new aspects of system development, UML provides a comprehensive notation for the full lifecycle of object-oriented development.

The UML notation is a fusion of Booch, OMT, OOSE and others. UML is designed to be readable on a large variety of media, such as whiteboards, paper, restaurant tablecloths, computer displays, black and white printouts, etc. The designers of

the notation have sought simplicity above all – UML is straightforward, homogeneous, and consistent. Awkward, redundant and superfluous symbols have been eliminated, in order to favor a better visual rendering.

UML focuses on the description of software development artifacts, rather than on the formalization of the development process itself, and it can therefore be used to describe software entities obtained through the application of various development processes. UML is not a rigid notation: it is generic, extensible, and can be tailored to the needs of the user. UML does not look for over-specification – there is not a graphical representation for all possible concepts. In the case of particular requirements, details may be added using extension mechanisms and textual comments. Great freedom remains for tools to filter the information displayed. The use of colors, drawings, and particular visual attributes is left up to the user.

Achievements and perspectives

It is now clear that UML is being adopted, with benefits, by a variety of users. We have mainly presented above, the short term achievements of UML, in a rather conventional way. Before concluding this introductory presentation, let us take a more high level view of the potential long term contribution of UML.

The OMG has grown to be an adaptable organization with an ability to detect very rapidly the evolution of industrial trends in technology deployment. At a time when many were still discovering the virtues of object orientation, OMG was already working on one of the first detected bottleneck of this technology: lack of interoperability. The answer to this has been the CORBA software bus. It is not by pure chance that the work on UML started there, it was because a real and urgent need to define modeling standards in the domain of object-oriented analysis and design emerged. However the consequences of this move are generally underestimated. What really happened then, was not only the definition of another specific new standard OMG recommendation, but also the starting point for a whole set of new activities. Previous activities were centered around the software transfer bus CORBA with its associated IDL language, IIOP protocol and OMA architecture. In the post-UML period, a new modeling culture is emerging, with a new knowledge bus incorporating UML, MOF, the OCL language [8] and the XMI transfer format [6]. The two buses and the two OMG activities are obviously linked, but the modeling camp is rapidly becoming important. It is now recognized that there are two ways to consider object interoperability, one is executable code interoperability and the second one model interoperability.

UML is now a conceptual tool, but it has also served as an experimentation field. As previously mentioned, the self definition of UML was an interesting exercise and was successful per se. However, it also demonstrated that the applicability of this technique could be made broader than just the handling of software artifacts. As a

consequence a new architecture was defined around the MOF (Meta-Object Facility). This architecture is complex and still evolving, but it could be compared to the OMA in importance. At the heart there is this self-defined MOF, which is more or less synchronized with the core definitions of UML. The MOF uses UML in various ways, for example for graphical presentations. But the main differences is that the MOF and UML are not at the same level in the OMG four-level model architecture. The MOF is a meta-meta-model and is at the M3 level while UML is a meta-model and stands at the M2 level. The MOF is a language for defining meta-models and UML is just one of these meta-models. Other meta-models that are being defined at the M2 level are for example related to common warehouse, workflow, software process, etc.

So, UML has been instrumental in triggering the development of a new modeling architecture based on the MOF. Many ideas have been successfully tested on UML and then transferred to the MOF because they were found to be of broader applicability. The first one is the OCL (Object Constraint Language [8]). OCL is an expression language that enables one to describe constraints on object-oriented models and other artifacts. The word constraint is used here with the meaning of a precisely identified restriction on one or more values of a model. We see here a pleasant property of the global OMG modeling architecture. Since a meta-meta-model is structurally similar to a meta-model, features applied to one, may also be applied to the other one. So OCL, that could be applied to meta-models to give more precise semantics to models, could also be applied to meta-meta-models to give more precise semantics to meta-models. And this is exactly what happens when OCL is applied at the MOF level.

Another example is the recent answer to the SMIF RFP of the OMG [6]. Initially the purpose of the Stream-based Model Interchange Format was mainly to exchange UML models. As it has finally been issued, answered and approved, the proposal is being known as XMI, a new standard for Metadata Interchange based on XML and on the MOF. Once again, there is nothing to loose, if by providing a technical solution to an UML problem, it is possible to provide a more general solution that could be applied to the UML meta-model, as well as to other meta-models already defined or yet to be proposed.

Many more examples could be given of this trend. There is for example several demands to provide structured extension mechanisms for UML, going beyond single stereotypes, tagged values and constraints. Requests are being submitted for specialized UML-based meta-models on subjects like real-time or business objects. A possible answer to this would be some notion of profiles. In the case where this improvement is allowed to the UML meta-model, there is no reason why other MOF-compliant meta-models should not also benefit from these added modular modeling mechanisms. A UML profile may be defined as a subset or a superset of the basic meta-model. There is however no agreement yet on the way this notion of a profile could be defined.

Conclusion

It is very tempting to draw a parallel between the historical development of programming languages since the early fifties and the more recent development of modeling languages. The important usage of graphical symbols in analysis and design notations may be made in correspondence with the old time art of flowcharting. Some of the OA&D notations were more business-oriented and some other were more scientific or real-time oriented, like Cobol and Fortran were also two different answers to these programming communities. We may also remember that these programming languages were usually the result of normative, industrial-oriented processes. So, should UML be considered as the PL/1 of modeling languages? The question is in fact troubling because the similarities in the definition process are numerous, specially in the way ingredients have been put together in order to satisfy the maximum of needs. If we take this resemblance for granted, what will then be the Algol 60, Algol 68, Pascal, C, C++, Occam or Java of modeling languages? As we know, the history of programming languages has not always been a linear progression according to scientific or technical criteria. At the beginning of this new period of development of modeling languages, we may hope that some lessons of the past have been learnt, but we shall not bet on this. Anyway, as we have sometimes heard in the last decade that "programming is thinking" we will surely hear in the coming years that "modeling is thinking" (or why not that "thinking is modeling"), and a good notation to write down its thinking will always be most valuable.

One of the recognized contributions of UML is that it has stopped many sterile wars of notations on aspects that were not highly significant. No more long discussions on the fifteen ways or so to note cardinalities or to draw classes and instances. This does not mean that the choices have always been the best possible ones [2], only that they have been grown from a general consensus and that they will allow a higher and more productive level of debate.

Another important decision that has been reported above is the separation of the debate on the notation from the debate on the process. This was a decision that was not easy to take and that will probably be considered as one of the main contribution of the authors. Now the work on the notation can progress and the work on the process can start integrating known research results and experience knowledge.

UML is not the first achievement in the modeling world. If we had to quote some of them we could choose SADT/IDEF0 for the simplicity and JSD for the principle of coupling the modeling of the system to the modeling of its environment. The next big challenge that UML will have to face is how to deal with the emerging and multifaceted notion of software component. This will be a major test in the coming years and if successfully passed, it may well become the main qualification title of this modeling notation.

References

1. Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language: User guide Addison Wesley, (November 1998)
2. Bergner, K. et al. A Critical Look at UML1.0. The Unified Modeling Language - Technical Aspects and Applications, M. Schader and A. Korthaus (eds.), Physica-Verlag (1998)
3. Fowler, M. UML Distilled: Applying the Standard Object Modeling Notation. Addison Wesley (1997)
4. Harmon, P., Watson, M. Understanding UML - The Developer's Guide with a Web-based Application in Java. Morgan-Kaufmann (1998)
5. Muller, P.A Instant UML Wrox Press, Chicago, (December 1997)
6. OMG XML MetaData Interchange (XMI) Proposal to the OMG OA&D TF RFP3 : Stream Based Model Interchange Format (SMIF) Document ad/98-10-05, (October 20, 1998), Adopted at the Washington Meeting, (January 1999)
7. UML Specification. Version 1.3R9, Rational Software (January 1999)
8. Warmer, J., & Kleppe, A. The Object Constraint Language Precise Modeling with UML Addison Wesley, (October 1998)

Developing with UML - Some Pitfalls and Workarounds

Martin Hitz[1], Gerti Kappel[2]

[1] Department of Data Engineering
Institute of Applied Computer Science and Information Systems
University of Vienna
A-1010 Vienna, Austria
hitz@ifs.univie.ac.at
[2] Department of Information Systems
Institute of Applied Computer Science
Johannes Kepler University of Linz
A-4040 Linz, Austria
gerti@ifs.uni-linz.ac.at

Abstract. The object-oriented modeling language UML offers various notations for all phases of application development. The user is left alone, however, when applying UML in up-to-date application development involving distribution, data management, and component-oriented mechanisms. Moreover, various shortcomings have been encountered, most notably w.r.t. refinement of model elements throughout the development life cycle and employment of interaction diagrams to formalize use cases. The paper will shed some light on how these issues may be handled with UML.

1 Introduction

"When it comes down to it, the real point of software development is cutting code. Diagrams are, after all, just pretty pictures." [4, p.7]

This opinion is still alive among researchers working in the area of software development as well as practitioners involved in software projects. Nonetheless, it has been more and more commonly accepted that the early phases of software development such as requirements specification, analysis, and design are key to the successful development and deployment of software systems. Not least due to the usage of some intuitive but rigor diagrammatic notations representing the artifacts of these development phases the software development process has been improved considerably. Object-oriented software development follows the same lines of thought. From the very beginning of requirements specification, object-oriented modeling notations provide intuitive mechanisms for representing the objects and

their interactions for reaching a common goal, namely the required system functionality.

Several object-oriented modeling notations and methods had been developed in the late eighties and early nineties (for an overview we refer to [5]). After different merging efforts and a request for proposals by the Object Management Group, UML (Unified Modeling Language) was adopted in November 1997 as the official industry standard for object-oriented software modeling notations [3, 4].

UML covers several advantages, among which only three shall be mentioned here. First and most importantly, the standardization of UML helps to bypass notational discussions and to concentrate on the real problems, such as modeling guidelines and design heuristics, proper development process, and proper tool support. Second, UML represents the fusion of the Booch method, Jacobson's Objectory, and Rumbaugh's OMT. As such and thanks to Objectory, the very first step of object-oriented modeling does not encompass finding objects in the problem domain - as has been the case in most other object-oriented modeling techniques - but the identification of the system functionality as required by the users. These so called *use cases* correspond to what has been depicted in level zero data flow diagrams known from traditional structured analysis. With use cases it has been possible both to overcome the "everything is an object and everything taken from structured development is bad"-mentality and to concentrate at the very beginning of software development on the user's requirements, which is just functionality and not objects. And third, different model views supported by UML allow to comprehend a complex system in terms of its essential characteristics. These are its system functionality (use case view), its internal static and dynamic structure (logical view), its synchronization behavior (concurrency view), and its implementation and physical layout (deployment view, component view) [3]. In this contribution, however, we will not dig into a further discussion of UML's goodies, but rather concentrate on pitfalls (which are more interesting anyway).

The main problems encountered during the development of a web-based calendar manager [8] are due to UML´s partially sloppy definition of notations, which lack a precise semantic specification. The main contribution of this paper is to shed some light on some of these deficiencies and discuss possible workarounds, some of which may be considered as suggestions of future enhancements of the notation. In the next section some refinements of UML constructs are discussed. Section 3 concentrates on the employment of interaction diagrams to formalize use cases. Finally, Section 4 points to the development of data-intensive, distributed applications based on component technology. Section 5 concludes the paper.

2 Refinement of Models

Development of complex systems based on various model views requires that the modeled diagrams can be related to each other for the purpose of *traceability*, i.e., connecting two model elements that represent the same concept at different levels of granularity. In addition, *consistency checking* between various model views

representing different though overlapping characteristics of the system at hand is a prerequisite for correct system development. Last but not least, most applications have to cope dynamically with changing requirements. Thus, various kinds of *evolution* mechanisms should be provided by the modeling notation. To adequately support traceability, consistency checking, and evolution, UML should provide for the refinement of model elements. In this context, refinement refers to "... a historical or derivation connection between two model elements with a mapping (not necessarily complete) between them." [16, p.71]. Note, that in contrast to the official UML document which refers to traceability as being mainly a tools and process problem we advocate the necessity to offer some kind of "meta notation" to graphically relate model elements which are derived from each other. In the following we will question some of UML's refinement mechanisms. We will investigate use case diagrams, class diagrams, and statechart diagrams. Sequence diagrams are discussed in the context of use case diagrams, too.

2.1 Refinement of Use Case Diagrams

A use case represents some system functionality. Several use cases together depicted in a use case diagram (not necessarily limited to a single physical page) make up the whole system to be implemented. To support both reuse and the stepwise specification of the required functionality, two use case relationships are provided by UML, the extends relationship, and the uses[1] relationship. Their precise meaning, however, is only poorly specified.

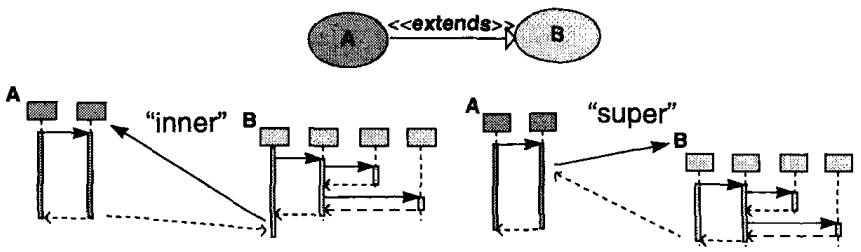

Fig. 1. Extends relationship between two use cases

Concerning the *extends* relationship, in [16, p.78] it is stated that if use case A extends use case B then an instance of use case B may include the behavior specified by A. Figure 1 depicts such a use case relationship. In the object-oriented literature there are two well-known interpretations for this relationship, which are captured by the inner concept of Beta and the super concept of Smalltalk, respectively.

[1] At the time of publication of this paper – Oct. 98 – the OMG UML revision task force is discussing UML 1.3, where "uses" will be renamed to "includes". Since these documents have not been officially released, we stick to the notions of the official UML 1.1 documentation.