IDEA GROUP PUBLISHING

701 E. Chocolate Avenue, Suite 200, Hershey PA 17033-1240, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP5098

Modelling Spaces and the UML

B. Unhelkar
Ph.D., FACS, University of Western Sydney, Locked Bag 1797, Penrith South DC, 1719, NSW, Australia, , Tel: +61-2-9685-9232, Mo:+61-413-821-454; Fax: +61-2-9685-9245, bhuvan@cit.uws.edu.au

B. Henderson-Sellers
Ph.D., D.Sc., FACS, FIMA, FIEAust, Faculty of IT, University of Technology, Sydney, brian@it.uts.edu.au

ABSTRACT

As the industrial adoption of the UML grows, so also do the issues faced by practitioners of the art of modelling. While the UML does provide a standard mechanism for modelling, it spans a wide gamut of work products produced by means of varied activities within the entire Software Development Life Cycle. However, not all features of the UML apply to all lifecycle "phases". This paper considers in detail the importance and relevance of various UML-diagrams in corresponding modelling spaces relevant to these different phases. By considering these differences in applicability of the UML diagrams, we propose that these diagrams and the UML overall will become even more relevant in practice.

INTRODUCTION

The Unified Modeling Language or UML (OMG, 2001) has been advocated as an object-oriented software development modelling language with wide applicability, although the standard contains no directives as to which part of the UML should be applied at what stage in the software development process.

As the UML becomes adopted increasingly in industry, its applicability to particular "phases" os the lifecycle is brought into question. As the UML evolves, it also grows in size, complexity and difficulty of understanding. Consequently, its validity and usefulness as a standard could be challenged. How the UML should be used, not discussed in the standard but only in derivative texts (e.g. Booch et al.¸1999, Stevens, 2000), becomes a key question to industry adopters. While processes play their role in making modelling relevant (see Henderson-Sellers and Unhelkar, 2000), it is still important to clearly delineate the relevance of the modelling constructs provided by the UML to the corresponding modelling roles and, therefore, modelling spaces in practice. It is only with such categorisation and provision of domain of applicability to the modelling constructs that practitioners can made practical use of the wide range of UML offerings at the correct stage within the software development lifecycle (SDLC).

This paper starts with a brief discussion of the UML from a practitioner's viewpoint. The practical issues are then considered, together with a range of UML diagrams. This is followed by a discussion on the understanding of three relevant modelling spaces: Problem, Solution and Background modelling spaces; together with the corresponding roles played by project team members in these spaces. The importance of the UML diagrams in each of these three modelling spaces is then considered in detail. Finally, we conclude with the relevance of such an approach in making good use of UML in practice.

THE UNIFIED MODELING LANGUAGE

About the UML

The Unified Modeling Language (UML) of the Object Management Group (OMG, 2001) is the current de facto modeling language for (primarily) object-oriented software applications development. The UML consists, as do all other modelling languages, of (i) a metamodel and (ii) a notation. There is also a language to add constraints - the OCL or Object Constraint Language (Warmer and Kleppe, 1998). Formally, it aims to provide one or more of the following in software development:

Visualizing – UML notations and diagrams provide an excellent industry standard mechanism to represent pictorially the requirements, solution and architecture

Specifying – together with visual representations, UML also facilitates the specification of some of the artefacts. This includes specifications for actors, use cases, classes, attributes, operations and so on.

Constructing – UML can also be used for software construction as it is possible to generate code from UML visual representations depending on the CASE tool being used. Furthermore, the use of OCL provides the ability to write software specifications with pseudo-code.

Documenting – With the help of the UML, additional and detailed documentation can be provided to enhance specifications and visual representations.

Unhelkar (2003) has argued that the UML can be effectively used in the following six different types of project categories:

a. New Development projects, wherein systems are designed from scratch; new business applications can be modelled using, for example, the UML's use cases and activity diagrams.
b. Integration projects, wherein newer systems are integrated with existing (typically legacy) systems
c. Package implementation; for example, implementation of CRMS or ERP systems
d. Outsourcing projects, wherein UML provides the basis for scoping, delivery and testing
e. Data Warehousing and Conversion projects, wherein not only are the data and related information modelled using the UML, but also the conversion and testing processes use UML to document the flow
f. Educational projects, wherein UML can be used for testing out concepts, e.g. for teaching-learning object orientation

Despite the possibility of its application in various types and sizes of projects, it provides a challenge, more often than not, in practice. This is because the UML is made up of a suite of diagrams that need to be understood before being applied in practice.

List of UML Diagrams

The source document by which a practitioner (or student) can learn about the UML is the OMG standards document (OMG, 2001) or from one of the many (derivative) texts on the subject. From these documents, a number of UML diagram types can be identified.(Table 1.1). There are basically eight diagram types although some texts introduce a further three. Also, sometimes, sequence and collaboration diagrams are counted as a single Interaction Diagram type. Robustness diagrams are rarely discussed. Rosenberg and Scott (1999) reintroduce them, based on original work by Jacobson et al. (1992). Although not official UML, Package diagram are often listed as separate from Class diagrams, because of their increasing importance in organizational and architectural areas of the system. They are also listed separately by at least one

Table 1.1: Table of UML diagrams in Practice (indicates additional diagrams to the original OMG list)*

UML diagrams	*Model Representing the …*
1. Use case diagrams	functionality from a user's viewpoint
2. Activity diagrams	the flow - within a method or the system
3. Class diagrams	Classes, entities, business domain, database
4. Sequence diagrams	the interactions between objects
5. Collaboration diagrams	the interactions between objects
6. Object diagrams *	objects and their links
7. State chart diagrams	the run-time lifecycle of an object
8. Component diagrams	the executables, linkable libraries etc.
9. Deployment diagrams	the hardware nodes, processors and, optionally, corresponding components
10. Package diagrams *	Subsystems, organizational units
11. Robustness diagrams *	architecture by ensuring separation of interface from business models

UML CASE tool. The component and deployment diagrams are also referred to as implementation diagrams in UML literature. Object diagrams are sometimes treated as independent diagrams in their own right, but are generally not supported by CASE tools as independent diagrams. At other times, object diagrams get treated as collaboration diagrams. This discussion indicates the astute reader that the list of the diagrams itself is not as important as knowing their precise strengths and the purpose for which the diagrams can be used.

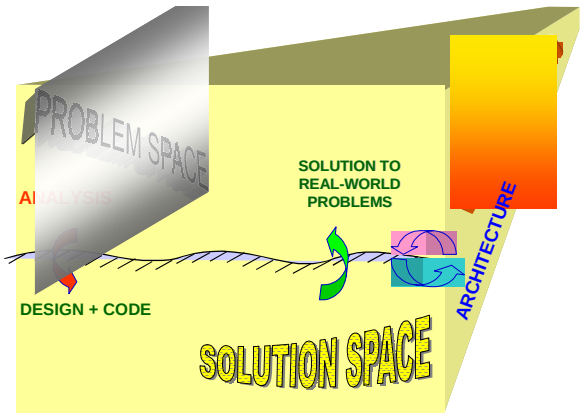
Furthermore, note that these UML diagrams are not orthogonal. They give different viewpoints on the one system (or rather system model). Many of these diagrams are interconnected, having dependencies on each other that are important from both syntactic and semantic angles (Henderson-Sellers, 1998). These diagrams, and the artefacts within the diagrams, are also augmented by their corresponding specifications and documentation resulting in greater information than just the visual means would provide.

MODELLING SPACES

Understanding Modeling Spaces in Practice

The model of the system should be unique: for a single system there is, at least cognitively, a single model. However, models reside in the modeller's brain. To communicate that model some diagrammatic and/or textual notation is used to create modelling work products. Typically the model itself is too large to fit into a single diagram so that several diagrams are necessary. Each of these highlights a different aspect of the system (e.g. static architecture, behaviour, functionality) and at a different granularity (Unhelkar and Henderson-Sellers, 1995). Furthermore, the suite of diagrams necessary depends upon the stage in the SDLC. Thus, understanding and appreciation of the value and the limitations of the UML diagrams and their congruent modelling spaces is important to provide us with the necessary background to start applying the modelling approach to practical software development.

Figure 1: The three modeling spaces (based on Unhelkar, 2003)



Practical difficulties in application of the UML arises because, although it is common to find a focus on creating various views of the same model, it is all too easy, in applying the UML, to ignore the fact that there are different roles within the software modelling exercise that requires different expertise. For example, the business analysts play a significant role in understanding, documenting and modelling the business issues or problems that the software is trying to solve. This requires significant understanding of the business but far less understanding of the technologies of implementation. The system designers, however, need significant understanding of the languages and databases of implementation, and therefore need solid technical knowledge. They need modelling constructs that will help them understand the implementation issues, and also explain their designs to the programmers to help them undertake the coding. Finally, in most software applications, the project, organizational and even industrial infrastructure provides numerous constraints. These constraints, and the operational requirements, also need to be expressed in a model, although staying in the background. This leads to an understanding that there are three major areas or spaces in which modelling work happens: understanding the problem, creating a solution, and doing both of these within the constraints of the architecture, in the background. These three modelling spaces are shown in Figure 1. These three distinct, yet related, spaces of modeling (Unhelkar, 2003) are:

Model Of Problem Space (MOPS)

Model Of Problem Space (MOPS) results from work in the Problem space that primarily deals with understanding and documenting the business problem for which the software is being created. This is the work of Business Analyst using the techniques of Object-oriented Analysis. Less often, it can also be a technical problem. In any case the problem space deals with all the work that goes on in understanding the software or system problem that is yet to be developed. In the case of projects dealing with existing legacy applications, this problem space handles the creation of a formal model of what exists, followed by a model of what is needed.

As the problem space focuses on what is happening with the business, the major activities that take place in the problem space are documenting and understanding the requirements, analyzing the requirements, investigating the problem in detail, optionally creating a conceptual prototype, and understanding the flow of the process within the business. Since it is a non-technical description of what is happening with the user or the business, the problem space needs those UML diagrams that explain the problem without going into the specifics of the technology used to provide the solution. These UML diagrams are the Use Case diagram and the Activity diagrams, followed by high-level use of Class and Sequence diagrams, and optionally State Chart diagrams.

Model Of Solution Space (MOSS)

Model Of Solution Space (MOSS) results from the design for the system that will enable handling of the problem described in the problem space. This is the technical work and the role that creates this model in

the solution space is that of a System Designer. The designer takes the information made available in the problem space in order to create and provide a solution that satisfies the needs of the user understood in the problem space. Since this is a technical modeling space, it needs detailed understanding of the programming languages, development of programming environments, understanding of databases, middleware, web application solutions, and a number of other related technical details. As a result, the primary diagram used in the solution space is the class diagram including the lower-most details of a class such as attributes, types of attributes, their initial values, operations and their signatures. The class diagrams are followed by sequence diagrams, together with their messages and protocols, in the solution space. Furthermore, State chart diagrams and object diagrams may also be used sparingly here. Finally, in order to complete the solution we will also need component diagrams, which may be initiated in the background space discussed next. The component diagrams represent the executable chunks of code or libraries (in a Windows environment these will be the .exe and the .dll), which are finally incorporated in the software solution.

Model Of Background Space (MOBS)

Model Of Background Space (MOBS) – deals with the background architecture of the system. The role is played by an Architect, and is supported by the Manager. This modelling space deals with two major aspects of software development that are not covered by either the problem space or solution space: architecture and management.

Management deals primarily with the planning of the entire project and does not necessarily form part of the problem or the solution space. To rephrase, management includes both problem and solution space. There are a number of activities within management that are handled in the background by the person playing the role of the project manager. They include planning the project, resourcing the project hardware, software, and people, budgeting and performing cost benefit analysis, tracking the project as it progresses so that the various iterations are performed per the requirements of the process, and providing the checkpoints that yield quality results for the roles in the problem and solution space. Planning activities fall under the general category of management, or to be precise, project management. It is worth mentioning here that while we have the job of managing such projects, the UML itself does not provide any direct help in project management as such. For example, there are no diagrams of the UML that can be used in managing projects.

Architectural work, on the other hand, deals with a large amount of technical background work that must consider major issues of architecture of the solution, existing architecture of the environment in the organization, and the operational requirements of the system (requirements of the system in operation—for example the stress and the volume and the bandwidth that the system needs). Further background issues include the important strategic aspects of reusability of programs, designs, and even architecture. These activities will most certainly require the local knowledge of the way in which the environment works, and the industrial knowledge of the availability of reusable architectures and designs. In all modern day software systems, making software in-house is almost sacrilegious, unless the avenues of buying it, or at least some sizeable component of it, haven't been exhausted. This all-important "make versus buy" decision is heavily influenced by the fact that work in this space is abstract yet precise. Explicit models of software or components greatly influence the decision to buy (or not to buy) them. The background space will need help and support from UML in modelling the deployment environment as well as in reusing both architecture and design. The UML diagrams that the background space uses as provided by the UML are primarily the deployment diagrams and component diagrams. More importantly though, the background space will end up using large material in the UML domain that deals with analysis patterns such as the work by Martin Fowler [1997, Design Patterns by the Gang of Four [Gamma et al. 1995], Cognitive patterns [Gardner et al. 1998], Anti patterns [Brown et al. 1998], and so on.

*Table 2: Importance of UML diagrams to respective models (a maximum of 5 * for maximal importance to that particular space)*

UML diagrams	*MOPS (Business Analyst)*	*MOSS (Designer)*	*MOBS (Architect)*
Use case diagrams	*****	**	*
Activity diagrams	*****	**	*
Class diagrams	***	*****	**
Sequence diagrams	***	*****	*
Collaboration diagrams		**	*
Object diagrams	*	*****	***
State chart diagrams	***	****	**
Component diagrams	*	***	*****
Deployment diagrams	**	**	*****
Package diagrams	***	**	****
Robustness diagrams	*	***	*****

SUMMARY OF THE MAPPING THE UML TO THE MODELLING SPACES

Further to the above description of the work that happens in the modelling spaces, Table 2 summarizes the relative importance of each of the UML diagrams in each of the modelling spaces and to each of the major modelling roles within the project. While project team members can work in any of these modelling spaces using any of the UML diagrams, good quality models will result by understanding the importance of the diagrams with respect to each of the modelling spaces.

CONCLUSIONS

In this paper we have presented a practical approach to using the UML. This is achieved by understanding the various modelling spaces, namely Problem, Solution and Background modelling space. We then created a simple star rating (1 to 5) of the various UML diagrams and corresponding modelling spaces. By creating such an understanding we anticipate that the UML will become more relevant in practice and, even more importantly, will constrain the modeller's choice of diagram and notation influence strongly by the modelling space in which they are working. They only need to address the appropriate subset of the UML before making their choice. Division of the work in the three modelling spaces in shown in Figure 1, and summarised in the table. It should be noted that there is no attempt made to create three watertight compartments but, instead, understanding and giving weighting to the roles working in corresponding modelling space. Thus the oft-heard claim that the UML is all things to all people, it is "too large" for practical usage can be obviated. However, the best way to use the UML still requires process knowledge. In this paper, we have purposefully not addressed this important issue but, instead, focussed on the modelling issues in order to clarify the realm of applicability of the various types of UML diagram.

ACKNOWLEDGEMENTS

The primary author wishes to acknowledge the support of the University of Western Sydney, School of Computing and IT, notably MIRAG: Mobile Internet Research and Applications Group.

REFERENCES

Booch, G., Rumbaugh, J. and Jacobson, I., 1999, *The Unified Modeling Language User Guide*, Addison-Wesley, Reading, MA, USA, 482pp

Brown, W, Malveau, R. "Skip," McCormick III, H., and Mowbray, T., *Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Inc., 1998.

Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*, Reading, Mass.: Addison-Wesley, 1999.

Gang of Four (GOF)—Gamma, Erich, et al., *Design Patterns: Elements of Reusable Object-Oriented Software*, Reading, Mass.: Addison-Wesley, 1995.

Gardner, K., Rush A., Crist, M., Konitzer, R., and Teegarden, B., *Cognitive Patterns: Problem-solving Frameworks for Object Technology*, Cambridge University Press: 1998.

Henderson-Sellers, B., 1998, OO diagram connectivity, *JOOP/ ROAD, 11(7),* 60-68

Henderson-Sellers, B., and Unhelkar, B., 2000, *OPEN Modeling with UML*, Addison-Wesley, UK

Jacobson, I., Christerson, M., Jonsson, P. and Övergaard, G., 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley, Reading, MA, USA 524pp,

OMG, 2001, OMG: OMG Unified Modeling Language Specification, Version 1.4, September 2001, OMG document formal/01-09-68 through 80 (13 documents) [Online]. Available http://www.omg.org (2001)

Rosenberg, D. and Scott, K., 1999, *Use Case Driven Object Modeling with UML. A Practical Approach*, Addison-Wesley, Reading, MA, USA, 165pp

Stevens, P. with Pooley, R., 2000, *Using UML Software Engineering with Objects and Components*, updated edition, Addison-Wesley, Harlow, England, 256pp

Unhelkar, B., *Process QA for UML-based Projects*, Addison-Wesley, Boston, 2003

Unhelkar, B. and Henderson-Sellers, B., 1995, ODBMS considerations in the granularity of a reusable OO design, **In** *TOOLS15*, (ed. C. Mingins and B. Meyer), Prentice Hall, 229-234

Warmer, J. and Kleppe, A., 1998, *The Object Constraint Language. Precise Modeling with UML*, Addison-Wesley, Reading, MA, USA, 144pp