UML SUPPORTED SOFTWARE DESIGN

Darko Gvozdanović, Saša Dešić, Darko Huljenić
Ericsson Nikola Tesla d.d., Krapinska 45, HR-10000 Zagreb, Croatia,
tel.: +385 1 365 3889 , faks: +385 1 365 3548, e-mail: darko.gvozdanovic@etk.ericsson.se

Abstract: *UML stands for Unified Modelling Language, a general-purpose notational language for specifying and visualising complex systems, especially for large, object-oriented projects. This article considers usability of UML in software projects. Several projects' startups, supported by UML, are analysed. UML's advantages and disadvantages are commented according to user's level of knowledge (novice against experienced developer), application type (e.g. protocol oriented or GUI oriented applications) and requirements detail level. During analysis, some differences between UML and SDL (Specification and Description Language) are emphasised.*

KEYWORDS: UML, software development, code generation, development process

INTRODUCTION

The good news for software developers is that world development and modern style of living depends increasingly on software. Software-intensive systems that technology makes possible and society demands are expanding in size, complexity, distribution and importance. But, expansion of these systems pushes the limits of what software industry know how to develop. The final result is that building and maintaining software is hard and getting harder. Companies use different methods trying to cope with the challenge. Proposed solution is to implement rigorous software development process (own process can be built or existing process can be deployed/customised) [3]. Also, deployment of notification language like UML will help communication between all stakeholders in development process. This is just one of the reasons for UML usage in process of software development.
This article analyses aspects of UML utilisation in development process. Ericsson Nikola Tesla Company has started with implementation of UML in projects. Small group of developers was created in order to investigate usability of UML. While some of them were beginners in UML, others had several projects behind them. All of them at least attended UML course. After course, two test projects were created and group was divided in two teams. Experiences from analysis will be described in this article, as follows: Section 1 gives short overview of UML (root, causes and purposes). UML diagrams are explained briefly. Next section introduces test projects that were made, reasons for their selection and development analysis. The third section discusses code generation and testing. Final section summarises results.

1 INTRODUCTION TO UML

The unified modelling language (UML) is graphical language for visualising, specifying, constructing, and documenting software-intensive systems [1]. The UML provides a standard way to write system's blueprints, covering conceptual things, classes written in a specific programming language, database schemas, and reusable software components. UML is standard notation, which is used by anyone involved in production, deployment, and maintenance of software.
UML includes nine diagrams for describing system:

- Class diagram describes set of classes, interfaces, and their relationships. It shows static design view of a system. This diagram is very useful in modelling object-oriented systems.
- Object diagram shows set of objects and snapshots of instances of the things found in class diagrams.
- Use case diagram shows a set of use cases, actors and their relationships. This diagram is especially important in organising and modelling the behaviour of a system.
- Sequence and collaboration diagrams are interaction diagrams, which describe interaction between objects. They show their relationships including messages between objects. Interaction diagrams explain dynamic view of the system. A sequence diagram emphasises the time ordering of messages. A collaboration diagram emphasises the structural organisation of objects in interaction. Both diagrams are isomorphic, which means that sequence diagram can be transformed into collaboration one and vice versa.
- Statechart diagram shows a state machine consisting of states, transitions events, and activities. It addresses dynamic view of system. This diagram is very important in modelling behaviour.
- Activity diagram is special kind of statechart diagram. It emphasises a flow from activity to activity within system. It is very useful in finding concurrent activities in system.
- Component diagram describes organisation and dependencies among a set of components. It is related to class diagrams in a way of mapping component to one or more classes, interfaces, or collaborations.
- Deployment diagram explains configuration of run-time processing nodes and components on them. It shows static deployment view of architecture.

2 UML DESIGN EXPERIMENTS

2.1 Experimental projects

In order to obtain useful data, two particular pilot projects were launched. First project, called JAMES, was design of application for communication based on specific protocol. When using this application, user should be able to make basic communication actions:

- Initiate connection establishment,
- Communicate through file transfer, sending and receiving messages,
- Release the connection.

Specification for the protocol this application should use, indicated that it should be decomposed into three main components (Figure 1)

- USI (User Side Interface) – control procedures for the user side,
- NSI (Network Side Interface) - control procedures for the network side, and
- GUI (Graphical User Interface) – graphical interface.

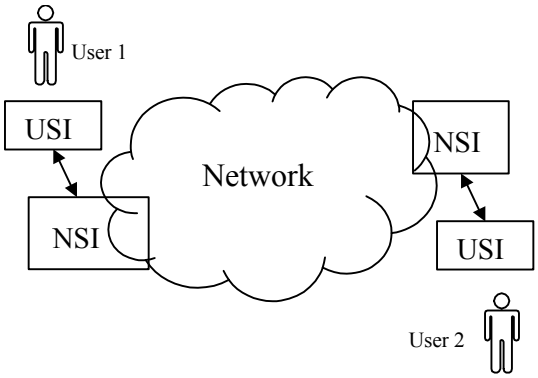

Figure 1Project "JAMES"

It was obvious that architectural design of this type of application was not very complex, opposite to behaviour that had to be designed in details. This is exactly situation that was presumed being not suitable for UML supported design, thus intentionally chosen as part of our experiment.

The outcome of the second project, called BOND, was functional prototype of the simulator for distributed processing system (Figure 2). Distributed processing system could be, for example, a web site with several servers acting as a group. In short, all nodes in the system receive requests for jobs that could be rescheduled from highly loaded nodes to less loaded ones.

The prototype was extended with mobile agents which task is system maintenance and monitoring. Agent should collect data about remote sites and save it on the management site.

User should be able to start or stop simulation. Previously, he or she must define network topology and load distribution (defines job arrival intensity per node). That could be done through graphical editors. During simulation, user is able to change simulation parameters (network topology and load distribution). Also, user can view statistics for the system and analyse overall system performance during or after simulation.

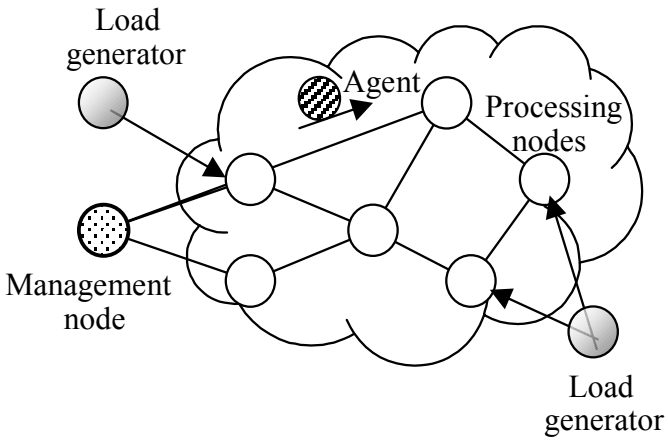

Figure 2Project "BOND"

Opposite to the JAMES project, this one is highly architecturally demanding. Of course, behaviour has to be specified here too (by programmer himself because it is not specified in requirements), but designing behaviour isn't so predominant activity like in "JAMES" project.

Main reason for choosing exactly these two kinds of projects, was investigating impact of application type on UML utilisation in design. But in parallel, we aimed toward investigating potential use of whole chain of formal methods (primarily use of UML to SDL converter

followed by SDL design and TTCN testing). That work is still in progress and results will be available in our other articles.

2.2 Features and benefits of UML utilisation

Upon end of design phase (i.e. both static and dynamic system architecture are captured and tested; implementation remains) some predictions regarding UML utilisation were met, and many useful additional conclusions emerged from it.

2.2.1 Who can use UML?

Everyone involved in these two project liked UML. However, reasons for that are somewhat different for non-experienced and experienced users. The former liked the fact that UML helps them to acquire object-oriented way of thinking and that it has no strict formal structure or terminology. However, mentors should ensure that misuse of customizing UML (beginners tend to do that) does not jeopardize uniformity of the system representation.
Experienced users appreciate many UML's features among which are: different points of view on the system, less formality in analysis, more in system design and implementation. In short UML gives adequate support for every part of a system design. It is logical consequence of the fact that UML incorporated the best practices of the various modelling techniques that existed before.

2.2.2 What types of application should be developed with UML?

UML proved to be much more appropriate for designing systems with complex static architecture. If that is the case, visual modelling gives its best. It allows multiple views on the system, from plain system overview to the detail design of each class. UML do provide means for defining system behaviour but it is neither possible nor intended to do that up to the last detail (simulation or validation of designed behaviour isn't possible nor you can use this type of information for automatic code generation). So, even if behaviour of the system is in the main focus, UML can be used, but that decision should be reconsidered. Utilization of some other formal languages may be better decision (SDL processes as communicating extended finite state machines are very suitable for designing behavior).

2.2.3 Where in project UML can be used?

UML can be used anywhere from requirements capture to implementation phase but it is most useful in early project phases (requirements capture and early system design). Two main reasons for that are elaborated in sequel.
After customer has supplied designers with requirements (frequently vague and insufficient), system analysis has to be done. Utilizing UML for that purpose makes analysis both simpler and faster because of better mutual understanding and clear visual representation. Use-case and sequence diagrams provide everything needed for capturing all the systems features, yet retaining simplicity needed for customer to understand designed behaviour. That way misunderstanding between designers and customers (the fastest way to project failure) is excluded. Furthermore, it is easy, even for customer, to transform use cases into scenarios for verifying and validating partly or fully designed system. Customer, of course, does not have to be someone outside company that develops software. It can be some other department or even developers themselves can be customers (like it is in our case). Important thing is that customer supply requirements and judges final result.

Second reason for using UML early in the project is adequate support for system division. That division is important for many reasons:

- Parallel design - UML with its packages and interfaces enables independent design of the subsystems. Interface descriptions are everything that various design groups responsible for those subsystems have to conform to. Consequently, future redesign and maintenance are facilitated, too.
- Design activities scheduling - use case diagrams could be very useful for this purpose. Indeed, it is easier to estimate implementation time for small parts of the project than for the whole project.

So if you have requirements that are not so clear and documented, utilizing UML is clever choice. Refining requirements is necessary either you will proceed designing manually, switching to other formal languages (e.g. SDL) or continue design in UML. Possibility of combining UML with another design approaches/tools is another of its great features. One possible design process that utilises this combined approach is shown in Figure 3. More detailed analysis of that approach will be available in our other papers.

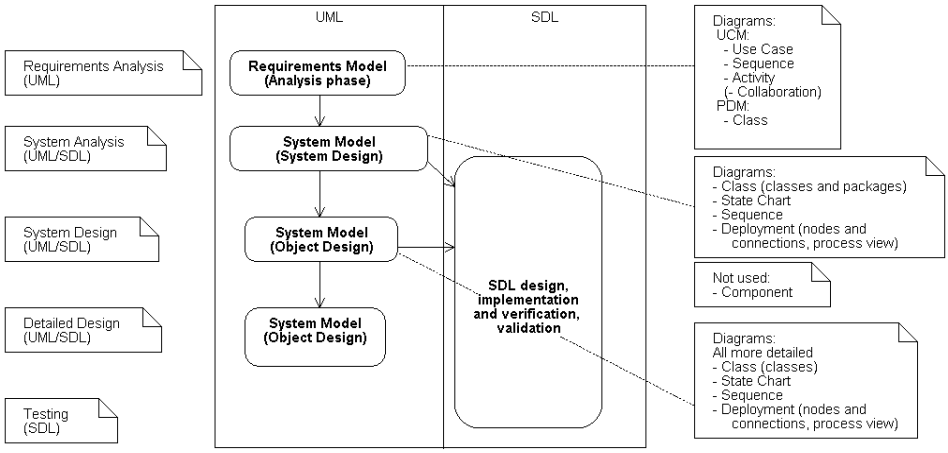

Figure 3 Software design process supported by UML/SDL combination

2.2.4 What about reusability?

Software development is not only about writing new code. Nor one can expect that one approach/tool is suitable for complete project. Therefore support for reuse of old legacy code is very important UML feature. It is not only that you can use old code as “black box” package, but you can even incorporate it into UML diagrams by using reverse engineering (Figure 4). That way, complete design stays in UML facilitating understanding of whole system.

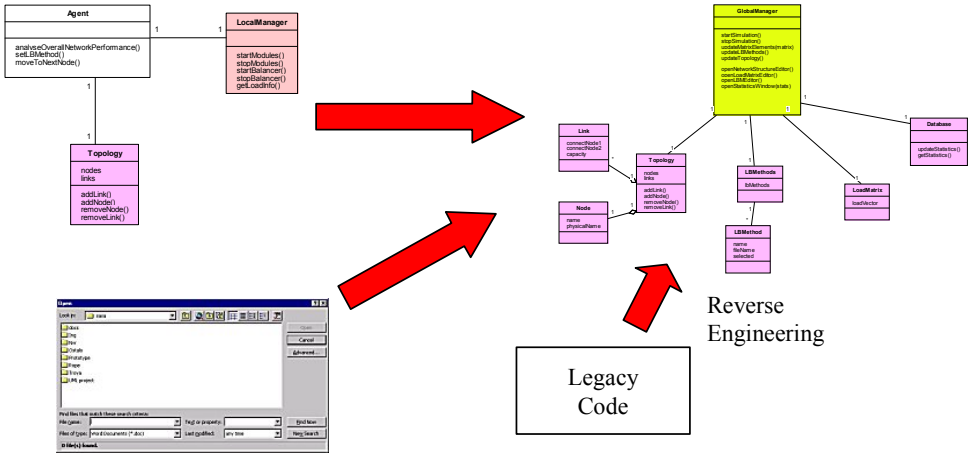

Figure 4 Importing GUI's and legacy code into UML project

Just like using legacy code, you can choose (and probably will) to design GUI in some of specialized tools and incorporate it as package into UML design (Figure 4). The difference is, comparing to other types of code, that it doesn't make sense to reverse-engineer GUI to UML because visual representation of GUI design (dozens of classes) doesn't help designing other part of the systems at all, and that is what UML is all about.
The bottom line is that you should use UML until implementation only for parts of design that you can perform better than using some other approach/tool. Everything else can be either included or even reverse-engineered into UML in the later phases of design.

3 CODE GENERATION AND TESTING

UML tool that was used in our experiment has had code generators for several most interesting programming languages. Principle of uniformity is somewhat broken in this last phase, because UML design has to be adjusted according to the intended code language. Also reverse engineering gives slightly different results if different languages are used. In any case, UML will not give 100% code generation. What it will give is code skeleton that has to be filled with some architectural things, and complete behaviour. That is consequence of the fact that automatic code generators use only class diagrams for their task. Implementation of all the behaviour defined in other UML diagrams is left to the implementers.
That fact arises interesting question about using UML for designing systems whose focus is on behaviour (e.g. implementations of some protocols). Most of the work for project JAMES (it is "behaviourally oriented" project) is designing sequence and state chart diagrams. Although that information is not lost (system design is described and documented in that way), it is not incorporated into generated code.

In any case, generated code is very clean, readable, well structured and documented. Notes added to UML diagrams are included into generated code in form of comments. Code is clearly divided in sections for attributes, constructors, and user defined methods. Another good feature is that generated code clearly reflects system design (relationships between classes, their associations, role names, multiplicities etc). Also, prior code generation, consistency or omission of all mentioned things (not so important in system design, but important for code generation) is automatically checked. That way tool assures correctness of generated code. That also means that any kind of simulation is impossible with UML tools. If you want more code generation, support for simulation and validation, you should choose SDL. Of course, types of systems

suitable only for SDL utilisation are complementary to ones suitable only for UML utilisation. But picture is rarely so black & white. In reality, utilizing right combination of UML and SDL can be solution for a number of different problems (Figure 3).

4 CONCLUSION

Everyone in process of software design likes UML. It offers various things to various stakeholders. It can be formal if you need so, but also can be as informal as plain English. You can use it for all type of applications. Expectancies should be different though. Architecturally demanding systems can be designed in details, while some other will have only packages indicating place of code (functionality, GUI etc.) in overall system picture. That code can be obtained in number of ways (some SDL Tool, Visual Studio, JBuilder etc.).
One of UML's main goals is to simplify communication between various project stakeholders. In our opinion that goal is definitely reached.
UML features regarding "real" design are almost equally good as those in "communicating" area. However, UML performs the best in early phases of design. Diagrams intended for that part of design play their role the best. Furthermore, dividing design through package diagrams enables parallel work thus shortening overall time needed for complete job. Reverse engineering of old code and code written in any other way is another one great UML features.
Implementation phase does not result in 100% code generation. But developer has clear, readable and commented code skeleton along with all kinds of descriptions either in shape of UML diagrams or informal text associated with them.
The bottom line is that you should use UML from analysis to implementation only for parts of design that you can perform better than using some other approach/tool. Everything else should be part of UML but only in order to have complete picture in one place. Those things should be developed some other way and either included or even reverse-engineered into UML in the later phases of design.

REFERENCES

[1] Booch, G., Rumbaugh, J., Jacobson, I., The Unified Modeling Language Use Guide, Addison-Wesley, 1999.
[2] Fowler, M., Scott, K., UML Distilled, 2nd edition Addison-Wesley, 2000.
[3] Kruchten, P., The Rational Unified Process, 2nd edition Addison-Wesley, 2000.
[4] Ellsberger, J., Hogrefe, D., Sarma, A., SDL Formal Object-oriented Language for Communicating Systems, Prentice Hall 2000.