APPLICATION AND SERVICE DEVELOPMENT USING UML AND SDL

Christian Schwingenschlögl, Stefan Schönauer
Technische Universität München (TUM), Institute of Communication Networks
Arcisstr. 21, D-80333 Munich, Germany, e-mail: schwinge@lkn.ei.tum.de
Arcisstr. 21, D-80333 Munich, Germany, e-mail: stefansc@lkn.ei.tum.de

Abstract

SDL is a formal description language and is especially useful for software engineering in the area of communication networks. As communication systems can be planned and implemented very elegant and fast when using SDL, it is widely used by universities and industry working in this field. However, there is still room for improvement in the typical software engineering process. In early phases the specification of the system is usually done with a simple word processor or something similar. This approach has major disadvantages: The format of the specification is not standardized, difficult to read and often highly ambiguous. As it is difficult to develop the application based on this specification, the programmer will probably regard the writing of the system specification a tiresome task. Given the fact that software is frequently changed resp. updated and that everything has to be completed until yesterday we come to the next problem: Changes made in the system have to be manually inserted in the specification documents. Regarding the various different formats of the specification, it is often more work to make the correkt changes in the specification then the program changes itself.
In our paper we describe the integrated utilization of UML and SDL in the software engineering process to overcome these problems. It is based on our experience in a software development project with one of our industry partners, the tool we used for specification and development with UML and SDL was TAU4.0 from Telelogic with its integrated UML-Suite.

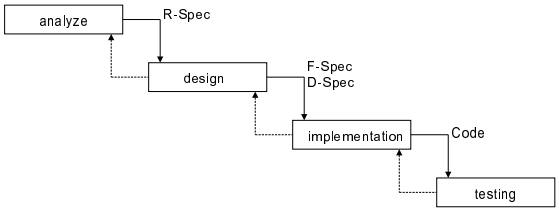

Figure 1: model of the process

1 PROBLEMS IN 'CONVENTIONAL' SOFTWARE DEVELOPMENT

The software engineering process usually consists of the phases requirement analysis, specification, design, implementation and testing. The beginning of each stage is based on the results from previous stages. In this paper we will not discuss the advantages and disadvantages of well known models, e.g. the waterfall- or the spiral model [Boe81]. We will rather discuss problems and shortcomings that arise in software engineering projects and which are independent of the used model. However, to discuss these shortcomings, we have to exemplarily explain one software development model first. We have decided to show a model (see Fig. 1) which corresponds to a iterative phase model, i.e. compared to the basic waterfall model it has iterations between its phases.

In the analysis phase the technical and qualitative demands on the system are usually specified in close work with the costumer. The formal result of the analysis phase is the requirement specification (R-Spec) which contains all demands on the software. Up to now, the R-Spec often consists of prose text and non standardized graphics. Similar to other documents and also the source code, this document is

reviewed. In the design phase, which comes directly after the analysis phase, two documents are produced as the main result: the so called functional specification (F-Spec) and the design specification (D-Spec). Roughly speaking, the F-Spec describes what is developed, the D-Spec describes how it is developed. After the design phase the implementation phase is started - here the F-Spec and D-Spec are transformed into code. The last phase in our model is the testing of the system. Not completely unexpected, the pracical experience shows remarkable weak points in this and also in other development processes (see [Bur99]).

- Results from one phase usually influence the results from previous phases. Therefore it is rather difficult to keep e.g. documents from early design phases consistent with documents written in later phases or the source code. Usually, changes made in the software during implementation or during a redesign have to be inserted in the various documents manually. As a change in the demands (which is not unusual in everyday work) after completing a phase has an impact on all the previous results, the change of documentation means a lot of work.

- In the individual phases a lot of different representations are used - caused by e.g. the utilization of different tools on different plattforms. The definition of common 'document formats' (widely used in things like word, etc.) does not help to improve this situation. One developer describes his software in text only, the next one is a friend of graphical representation which is not necessarily standardized.

- The integration of actual results in results of later resp. previous phases is a work intensive task and also a well known source of various errors. If results from the design-phase can not or only to a small part be used during development and testing, this is highly demotivating for the developer. Therefore, design specifications are, despite their importance, widely regarded as a useless task.

The utilization of UML in connection with the appropriate tool can provide a lot of improvements in this situation. A goal of this paper is to show possibilities and advantages of UML when used in the right situations. Particularly, the integration of UML and SDL in software engineering projects for network protocols and real time problems is regarded.

2 THE UNIFIED MODELING LANGUAGE (UML)

In this paragraph we will give a brief overview of the UML and describe which concepts are included in this modeling language. We will also describe some scenarios in which we think utilization of the UML will be especially advantageous. Further information about UML can be found in the corresponding literature, e.g. [FK97], [Oes98] and [BRJ98].

The unified modeling language is a standardized modeling language for describing software systems. It can also be used to specify other complex systems, however, we will focus on software systems in this paper. Basically, graphical descriptions are used in modeling languages for describing the system structure and the different methods [FK97]. According to [Kah98], a 'modeling language' is conceptually similar to programming- and machine languages. We don't agree with this completely and prefer the characterization in [BRJ98]. Here, the UML is described as a language for visualizing, for specifying, for constructing and for documenting.

So, the UML does basically describe a uniform notation and a semantic - i.e. it is defining a meta model and can not be described as a 'development technique' [Oes98].

The attribute 'unified' is expressing the basic idea behind the UML - the usage of a unified and standardized notation in various areas. The UML consists of different types of diagrams, the diagrams themselfes consist of various different graphical elements. With the diagrams, one can describe static and dynamic aspects of the system - dependent on which diagrams are used with different points of emphasis. The following overview shows the different types of diagrams available, the corresponding operational area and the phases in the software engineering model where they are used preferrably.

The attribute 'unified' is expressing the basic idea be-

hind the UML - the usage of a unified and standardized notation in various areas. The UML consists of different types of diagrams, the diagrams themselfes consist of various different graphical elements. With the diagrams, one can describe static and dynamic aspects of the system - dependent on which diagrams are used with different points of emphasis. The following overview shows the different types of diagrams available, the corresponding operational area and the phases in the software engineering model where they are used preferrably (according to [Wah98]).

STATIC DIAGRAMS

- class diagram
  - analyze, design, implementation
  - The class diagram is the most important and most common diagram of the UML. You can use it almost everywhere.
- package diagram
  - design
  - General overview showing which class you find in which modul.
- implementation diagram
  - analyze, design
  - Description of the systems structure of hardware and software. The UML knows two kinds.
    1. component diagram:
       Software components, their interfaces and their interrelationships. Connections between the different software moduls.
    2. deployment diagram:
       Configuration of run-time processing units, including the hardware and software that runs on them. Shows the physical structure of the system.

BEHAVIORAL DIAGRAMS

- use case diagram
  - analyze, design, testing
  - Shows the interactions between acteurs and the system. General operational areas (e.g. also business processes).
- interaction diagram
  - analyze, design, implementation
  - Shows the massage flow between the objects and so the temporal behavior of the system.
    1. sequence diagram:
       Temporal Calling structure with a little number of classes.
    2. collaboration diagram:
       Temporal Calling structure with a little number of messages.

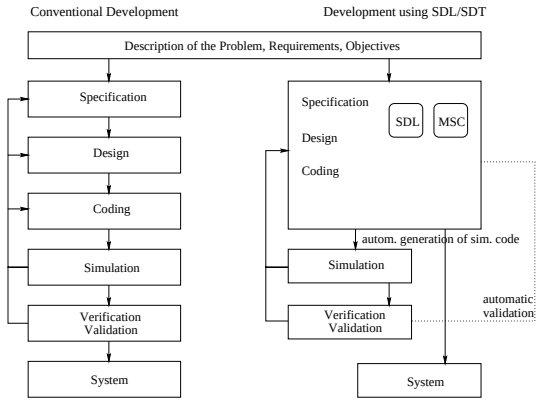

Figure 2: Comparison between conventional development and development using SDL (Source: [Kel95])

- statechart diagram
  - analyze, design, implementation
  - Shows a state machine. Representation of the dynamic bahavior.
- activity diagram
  - analyze, design
  - A special form of statechart diagrams. Useful for showing parallel processes.

3 THE SYSTEM DEFINITION LANGUAGE (SDL)

The application of SDL in engineering offers a series of advantages. Fig. 2 shows the comparison between conventional development and development using SDL.

As several phases of development are grouped together, the use of SDL accelerates and simplifies the engineering process. Startimg from a rough system description all work can be done within one document. In the conventional development, many documents are needed. These documents are usually written in different (programming) languages, resulting in an increased effort. The SDL Design Tool SDT which we currently use provides the possibility of doing simulations, system verification and validation automatically. In conventional development, an additional 'simulation code' has to be added to the system code.

3.1 Development of SDL

This section gives a brief overview about the development of SDL [wf]:

- 1968: ITU study of stored program control systems
- 1972: Specification, programming and HMI studies are started
- 1976: 'Orange Book SDL': Basic graphical language
- 1980: 'Yellow Book SDL': Process semantics defined
- 1984: 'Red Book SDL': Structure and data added, a more rigorous definition. Start of tools and user guide.
- 1988: 'Blue Book SDL' (aka SDL-88): Formal syntax definition, effective tools. Language is similar to Red Book SDL.
- 1992: 'White Book SDL' (aka SDL-92): Types for blocks, processes, services with inheritance and parameterisation. Methodology guidelines.
- 1995: SDL with ASN.1 (Z.105)
- 1996: Addendum 1 to SDL-92. SDL+ Methodology. Tools offer SDL-92 features.
- 1999: 'SDL-2000': Object modelling support, improved implementation support. Revised data model. (ITU-T Z.100)

4 SOFTWARE DEVELOPMENT USING UML AND SDL

Using UML today usually means specification in UML and implementation in C++ or java. Also the automatic code generation function of UML tools is often used. This function can generate the 'headers' of the software, or, in other words, translates the UML specification into the corresponding class definitions. Without doubt, the UML is a great help to develop the software system in such cases, however, there are still problems using it for real time systems.
We have had exactly these problems in one of our projects - it involved the development of a relatively

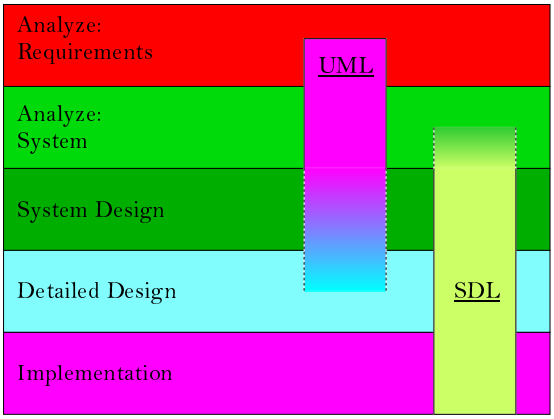

Figure 3: advantages of UML and SDL

large and complex real-time system for telecommunication equipment. As we have some experience with SDL our first thought was to use this FDT. Due to the complexity of the system we wanted to start our software engineering process with simple use-cases. Here we experienced exactly the problem described above - we would be specifying our system using use cases, etc. and later we would have to transfer the results manually in our SDL diagrams. So we decided to try the integrated approach - starting with UML and switching to SDL somewhere during the software engineering process. These two mainly graphic oriented description techniques can very well be used in combination: as the UML has its strengths in the early phases of software engineering because of its informality, the SDL has its advantages mainly in the design phase and later. (see Fig. 3 (source: [And99])).

Due to the quite similar concept of the two formal description techniques, the automated generation from UML to SDL and also the possibility of reverse-engineering is becoming more and more attractive. The standardization efforts of the ITU (Z.100 [Z.199a] and Z.109 [Z.199b]) show that this is a point of general interest. In a diploma thesis at the LKN ([Sch00]) we have designed a software engineering process that shows the integration of UML and SDL with focus on design and implementation phases.

4.1 The Tool Telelogic Tau 4.0

In our project we use the tool Tau4.0 from Telelogic as our development environment. This tool does already integrate the two methods UML and SDL. After the standards are approved by the ITU, we expect further improvement of this tool regarding the integration of these two concepts. Besides the automatic generation of SDL from UML, this tool does also provide the possibility to generate (at least parts of) the development documentation automatically. Up to now, the translation from UML to SDL is done as follows:

- SDL structures (SDL-system structures or SDL package structures) are generated from class diagrams. Based on the stereotype of a class in UML it can be decided if SDL processes or SDL blocks should be generated. The methods and attributes of a class with stereotype process are translated into procedures and variable declarations of the according SDL process. Also associations (they are translated into channels) and inheritance relations are regarded when generating SDL.
- UML statecharts are translated into SDL state machines. Here we have to regard the first discrepancies between the two techniques: Up to now we can not describe all possible SDL class concepts as UML statecharts. As the UML is a technique for modeling and not necessarily for implementation it is not the main goal of UML to support all SDL class concepts. However, this 'feature' is needed for reverse- and roundtrip-engineering.
- The UML sequence diagrams can be translated into SDL MSCs (message sequence charts) - as sequence charts are only a slightly different representation of MSCs this is the easy part of the whole thing.

4.2 THE DEVELOPMENT PROCESS WITH UML AND SDL

In Fig. 4, the process for the phases analysis design and implementation is given.

For the analysis phase, the UML is usually a good choice. Here the UML use case and sequence diagrams can be used. It is not the goal to specify all thinkable use cases in this phase - only the most important use cases (i.e. relevant to the system design) have to be specified. Furthermore informal class diagrams can be very usefull for a graphical

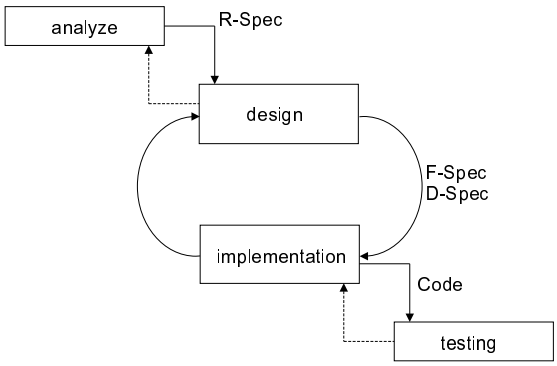

Figure 4: modified process

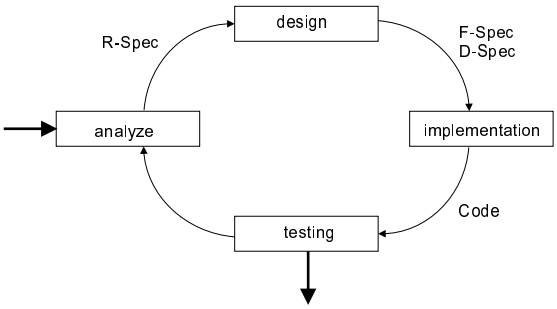

Figure 5: future process

representation of special problems. After completing these steps, a R-Spec can be generated automatically by the tool. The format of the document can be specified in the generation preferences - i.e. the user can choose which diagrams and which graphics should be used in the document. It is not yet possible to generate a R-Spec which has not to be edited manually, but the result is very near to this goal. The results from the R-Spec can nwo be used directly by the developer in the integrated development environment. If a developer wants to read the requirements for a system which have often been specified by other developers, he does not have to deal with several R-Specs in different representations any more - he can work in a uniform development environment and with a uniform notation.

Deployment diagrams provide a very good overview of the whole system in the design phase and the system structure can be shown in a more uniform way as before. Another advantage is the possibility to generate SDL code directly from statecharts and sequence diagrams. This means the results from the design phase are not only available for the documentation in the F- and D-Specs - as with R-Specs, the results can directly be used for coding.

As shown in Fig. 4 it is now possible to keep code and documentation consistent with only moderate effort. The next goal here is to eliminate still necessary manual tasks for documentation. We think the direct utlilization of sequence diagams from the analysis phase for system testing will be possible in the near future. Also 'big' iterations (they have to be made if the requirements change) can be managed easier as without the UML-SDL integration. The documents do not have to be changed manually, most of this work can be done automatically.

4.3 A Future Software Development Process

The process described above is based on our current work. Possible improvements include better support for iterations and the orientation on processes like the object engineering process (see [OHJ+99]). In figure 5 a proposal for a future software development process is given.

5 UML OR SDL?

We expect exciting developments in the area of UML and SDL for the next years. We have clearly seen the trend to reuse results from early specification and analysis phases in later phases of software engineering. Work in this direction is done by the ITU-T in Z.100 (SDL2000) and Z.109 (UML and SDL). Especially Z.109 provides help for SDL developers who want to integrate UML in their software engineering process.
Up to now, mainly SDL is used in software engineering for real time systems (sometimes integrated with UML in the early phases). As of today, UML-only approaches in software engineering for real time systems are very rare because the UML is not yet precise and formal enough to be used for the whole process for such systems. But as we see in the OMG RFP ad/98-11-01 "Action Semantics for the UML", the OMG is starting to work in exactly this direction.

References

[And99] J. Andersson. Die UML echtzeitfähig machen mit der formalen Sprache SDL. *OBJEKTspektrum*, 6(3):80–85, 1999.

[Boe81] B. W. Boehm. *Software Engineering Process.* Englewood Cliffs, 1981.

[BRJ98] G. Booch, J. Rumbaugh, and I. Jacobsen. *The Unified Modeling Language User Guide.* Addison-Wesley, Bonn, New York, Sydney, 1998.

[Bur99] R. Burkhardt. *UML Unified Modeling Language. Objektorientierte Modellierung für die Praxis.* Addison-Wesley, Bonn, New York, Sydney, 1999.

[FK97] M. Fowler and S. Kendall. *UML Distilled.* Addison-Wesley, Bonn, New York, Sydney, 1997.

[Kah98] B. Kahlbrandt. *Software-Engineering. Objektorientierte Software-Entwicklung mit der Unified Modeling Language.* Springer, Berlin, Heidelberg, New York, 1998.

[Kel95] W. Kellerer. Spezifikation, Simulation und Implementierung eines Sicherungsprotokolls mit SDL. Master's thesis, Lehrstuhl für Kommunikationsnetze, TU München, 1995.

[Oes98] B. Oesterreich. *Objektorientierte Softwareentwicklung.* R. Oldenburg, München, Wien, 1998.

[OHJ+99] B. Oestereich, P. Hruschka, N. Josuttis, H. Kocher, H. Krasemann, and M. Reinhold. *Erfolgreich mit Objektorientierung. Vorgehensmodell und Managementpraktiken für die objektorientierte Softwareentwicklung.* Oldenbourg Wissenschaftsverlag, München, 1999.

[Sch00] Stefan Schönauer. Development of a Maintainance Component using UML and SDL. Master's thesis, Munich University of Technology, 2000.

[Wah98] G. Wahl. UML kompakt. *OBJEKTspektrum*, 6(2):22–33, 1998.

[wf] www.sdl forum.org. SDL Forum Society.

[Z.199a] ITU-T Recommendation Z.100. Specification and Description Language (SDL). ITU, 1999.

[Z.199b] ITU-T Recommendation Z.109. SDL in combination with UML. ITU, 1999.