The usage of MSC with uBET-toolsupport in the software development process

B. Knaack*, S. Mauw†

March 8, 1999

Abstract

The usage of formal methods and design tools in the software development process is becoming a key issue in the industrial software development world. In this light we studied the Lucent Technologies software design process. We describe the MSC-requirements description method and investigate how the usage of MSC can aid in the software design process. Furthermore we look at the uBET-tool for the support of MSC deployment and compare the uBET syntax and semantics with the formal MSC definition.

The usage of MSC.....(extend,extend)

1 Introduction

In the telecommunication market the time to market window is decreasing rapidly due to new competitors and new customers in a dynamic market. Therefore the overall throughput time of projects such as customer reported problem solution, new features and software updates is becoming a competitive asset on the telecommunication market. To speed up the total design cycle a new software design process is currently needed.

In this light we strive to investigate in this paper the current software design process at Lucent Technologies, a major player on the telecommunication market.

Lucent Technologies was part of the AT&T concern until it split up in the new AT&T, NCR and Lucent Technologies. The main business for Lucent Technologies is the development of telecommunication equipment like optical networks, SDH networks, wireless communication and telephone switches. For these products software need to be developed, extended and constantly updated. This results in a major software development effort positioned at several sites world wide.

At Lucent Technologies a department in Holmdel USA is at the moment looking at a radical change for this software development process as a whole. At the moment pilot projects are run in Malmsbury and Indian Hill to gain experience in the usage of new methods and tools.

The new overall strategy as proposed by the Holmdel department for the adaptation of the software design process foresees in a structural change. In this new strategy attention is paid to an integrated tool support for all phases. In such an approach it is possible to exchange -next to traditional documentation- other forms of information as well, such as for instance computer models or program models.

In this paper we want to study part of the approach for changing the software development process to get to a more efficient structure. To keep a clear scope on the subject we decided to zoom in on part of the total project and try to describe how that part might be adapted, such that it is compatible with future changes needed in other phases of the design process. Therefore we decided to concentrate on the first phases of the software design process, where the requirements are written and transferred to the groups responsible for the other phases.

In this paper we concentrate on the MSC standard for the following of reasons:

*Lucent Technologies

†Eindhoven University of Technology

- It is an ITU standard ([2]) which is used and known by other telecommunication companies like providers and PTT's as well.
- There is tool support for MSC connecting it to other well known method such as SDL ([1])
- Lucent is already working on their own toolset for MSC support in the form of the uBET tool([**?**]), which also implies that there is expertise available within Lucent.
- It is graphical based method, which is very appealing to the intuition, which makes it easy to use and to understand, At the same time is has a very strong formal background.

In this paper we give a short description of MSC and explore how MSC and uBET can support the software design process at Lucent in the different phases of the design process. In order to do this we first give a description of the current software development process. This description is followed by a description of MSC and uBET. After this we give an outline of where and how MSC/uBET can be used to enhance the current design process.

In the last two sections we present some conclusions and present the first steps on how to introduce uBET in Lucent Hilversum and integrate all this in a bigger plan for the total software design cycle.

2 The Current Software Development Process.

The process description in this section is an abstraction of the real process in the sense that financial, resource and organizational compounds of the process are not described. For a full description of the process several pages would be needed. Therefore we present an outline of that part of the process that is subject of our research.

The current software development process at Lucent consists -amongst others- of the following phases; Requirements analysis, System Engineering, Software Development, Deliverable test, Feature Test, Customer Delivery and Ongoing support. Each of these phases is conducted by a separate group although there is some overlap; the groups responsible for these phases are: Project Marketing and Management, Customer team, System Engineering, the Development groups (both for development as for deliverable testing), Feature Test, Customer Technical Support and Local Field Support.

In each of these phases documentation is written which is then used in the other phases. These documents are mainly written in natural language, which can be ambiguous and sometimes hard to use to correctly transfer ideas and concepts. On turn over from one phase to the next, the documents are the main (if not only) means of sharing information.

If for instance Local Field Support has to solve a customer reported problem they have to browse through this documentation and through the software code to find out what the problem is and how it should be solved. Since the documentation is mostly in natural language and is quite extensive, this process is time consuming and error prone.

In the first phases of the software development process the customer requirements need to be acquired. Once the customer requirements are known they have to be translated to a specification document which can then be used for software development. In this specification document, scenarios for the possible system behavior are given to clarify the descriptions, which are mainly given in natural language. These specifications are then used by the development phases.

In the current software engineering process if examples of scenarios are given, they are given in *seqflow*, which is a very simple scenario description language. In *seqflow* the processes in a scenario are denoted by vertical line and messages between processes are (mostly horizontal) arrows. The vertical 'axis' denotes the time, the further down a message the later it appears in the scenario (see Figure 1).

In this section we look at four groups in more detail and outline the internal structure of the work done by each group. We have chosen these four groups since we expect them to benefit the most from the methods we describe in this paper. An oversight of the relation between the four different groups is given in Figure 2.

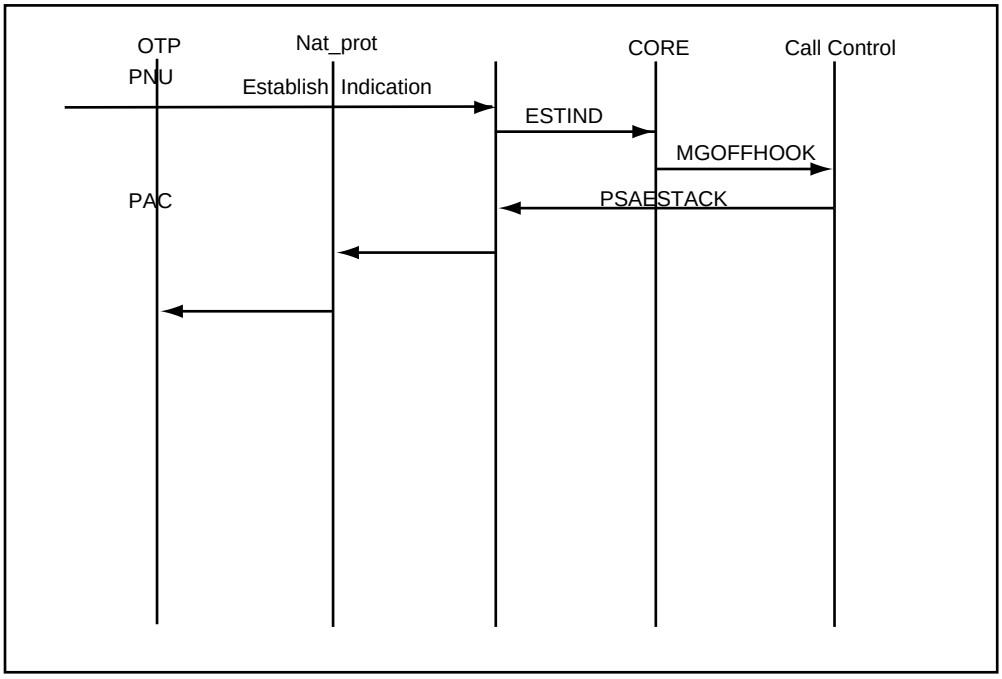

Figure 1: An example of a *seqflow* scenario

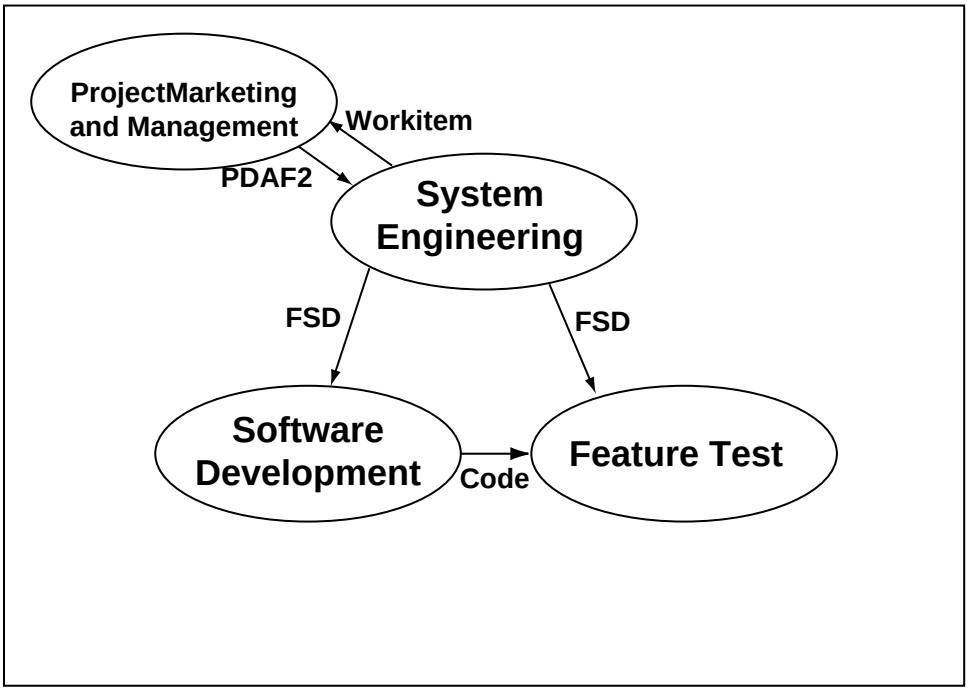

Figure 2: The process overview

2.1 Product Marketing & Management

The Product Marketing & Management group (PM&M) defines in discussion with a customer team or as a result of market analysis a product that should be developed. This definition is refined in a Product Definition (AF) document at a conceptual level in natural language. This product definition can be at a very high conceptual level.

2.2 System Engineering

The System Engineering team (SE) is involved in the software development process on more then one occasion. First they are required by PM&M to work out the PDAF2 document into several work-items. Each work-item is a brief technical description of part of the features as demanded by PM&M in the PDAF2. The work items divide the work in logically 'separate' tasks that have to be performed by the different development groups to come to the full functionality.

Although the definition of the workitems is done by SE it is under the responsibility of Project Marketing and Management. After the workitems have been committed –after (possibly) a number of feedback loops with PM&M– each workitem gets appointed to a software development group that 'owns' the functional area expertise that is needed for that workitem.

After having defined these workitems, each workitem is written out in a FSD (Feature Description Document) which is a more detailed description of the functionality splitting up the workitem in a list of requirements, that are to be met by the system. In these FSD's scenarios are sometimes used to illustrate the design, but they are not used to describe requirements.

2.3 System Engineering to Software Development

To make sure that the FSD contains all necessary information needed by the software development groups, a document review is held at the end of the system engineering phase. In this document review mostly one or two developers from the team that has to implement these changes are participating. After this document has been reviewed and the rework has been done the FSD is handed over to a software development group.

2.4 Software Development

In the software development group the FSD is used as a basis for a High Level Design that zooms in on the specific functional area of the software group. It contains a conceptual description of the solution that will be used to implement the changes, a survey on what code will have to be changed and what data structures need to be added. All of these are part of the standard High Level Design Template.

For clarification a number of scenarios might be included. New requirements will be given in the High Level Design that cover (and mostly refine) the requirements in the FSD. After the review of the High Level Design, a Low Level Design is made that clearly states how the solutions described in the high level design are to be implemented. The Low Level Design is then used for the eventual coding.

The next step in the development phase is the deliverable test phase, where whitebox testing is performed. To execute the whitebox tests the scenarios are identified that need to be executed on the code simulator or on an actual telephone switch. The result of these whitebox test is stored in a test database together with a scenario that was used to perform the test.

2.5 System Engineering to Feature Test

The same FSD that was given to the Software Development group is also given to the Feature Test Group. To keep the test procedure unbiased the feature test group does not get the High Level Design nor the Low Level Design that were made by the Software Development group.

2.6 Feature Test

The feature test is responsible for testing the functionality of the new code using a black box testing approach. Furthermore they have to test possible feature interactions with other features. To specify the test they have to perform they write a Requirements Test Plan (RTP). This RTP is based on the description in the FSD.

Since the feature test team is not involved in the development nor in the review of the FSD, the feature testers have to '(re)interpret' the requirements as given in the FSD. In order to get a clear understanding of the system requirements, they are sometimes translated into scenarios. Once the system requirements are known the feature tester has to conceive which other features might interact with the new functionality and set up tests for these interactions.

After this survey the tests are developed that have to be performed on the actual telephone switch to test the interactions of the new feature with already existing features and the functionality of the new code on itself.

The RTP is then reviewed together with the development team, to make sure that the tests and their expected results are indeed feasible. During these reviews the mismatches in interpretation of the requirements from Feature Test and software development rise to the surface.

After this the tests are performed and their results -including the test scenario- are stored in a test tracking system.

3 Message Sequence Charts

Many languages have been designed to describe the behaviour of information systems. Using such a language, one can describe the high-level behaviour of a system without having to worry (yet) about the exact implementation details. One such language is Message Sequence Chart (MSC) [1]. It differs from other languages in two important aspects. In the first place it puts emphasize on communication between processes, not paying much attention to the internal behaviour of these processes. This way, it specializes on systems in which communication is important. One area where it is much used, and the one for which it was originally created, is telecommunication systems. In the second place, MSC provides a fully graphical picture, rather than a textual description. Because of this, it can be more easily and intuitively understood by human users. Still, behind this graphical syntax lies an exact meaning and a well-defined semantics. Because of this, it can also be well understood by commercial and academic tools.

MSC-like diagrams have a long history in formal descriptions of information systems, but the official Message Sequence Chart language has been developed starting in the early nineties within the ITU (International Telecommunication Union) and its predecessor, the CCITT (Comité Consultatif International Télégraphique et Téléphonique)

MSCs are used in different contexts. The original purpose of MSC when it was first formalized, was to describe requirements in the early phases of the development process. It was intended to be an addition to SDL (Specification and Description Language), where the two languages would be used in different phases of the development process, MSC early on, when requirements and global specifications are made, SDL later on, when specifications are closer to the final implementation.

However, the language is now used in many more applications. To name a few: the description of the actual behaviour of an existing system, especially in the context of testing, the generation of test cases, the specification of protocols and the formalization of use cases.

As we saw in the introduction there are important advantages in preferring the MSC method for system design and analysis:

- Due to its standardization by the ITU, there is world wide interest in the further development of the language and its supporting tools. It is well documented and known throughout the telecommunication society. MSC is a stable language, which is maintained by the ITU study group in a four year cycle.
- It is graphical based method, which is very appealing to the intuition, which makes it easy to use and to understand.

- The MSC language supports many features. Basic features include asynchronous communication, timers and local actions. The ordering of events can be weakened by so-called *coregions* and strengthened by means of general orderings. Inline expressions can be used to define small variations of a scenario. Top-down design is supported by instance decomposition. High-Level MSCs and MSC references can be used for a modular design and allow for the reuse of parts of an MSC specification. Finally, the upcoming revision of the language, MSC2000, will support the inclusion of a data language, timing requirements, and a control flow mechanism (as in UML).
- There is tool support for MSC connecting it to other well known methods such as SDL ([1]). This tool support is still improving and new tools are still developed to increase the usability of MSC. The Lucent Technologies uBET toolset is an example of a tool that provides support in MSC-design and is still evolving to become more powerful and usable.
- Despite its graphical appearance, MSC has a strong formal background. This makes it possible to:
  - make unambiguous system-descriptions;
  - interface with other tools that might be used to support the other design phases;
  - use formal validation methods on the designs.

Because MSCs are based on a simple and intuitive paradigm, many similar languages have been developed independently. The seqflow diagrams can actually be seen as an MSC-like language. However seqflow is limited in its application and expressive power with respect to the full MSC language.

The complete expressive power of the MSC language can not be explained in just a few pages. Therefore, we will only highlight some of the most relevant features by means of a simple example. For a complete description of the language we refer to [2]. An extensive tutorial is in [3].

A scenario in MSC exists -just like in seqflow- of a number of processes (represented by vertical lines) and messages between these processes (denoted by 'horizontal' arrows) (see Figure 3).

The (meaningless) example in this figure also shows some other features, such as the *condition*, denoted by a hexagon, which indicates a state of the system. The hour glass symbol denotes the setting of a timer.

Small variations on scenarios can be expressed by inline expressions. This is denoted by the rectangular box labeled with *alt* in the upper left corner. The two alternative parts of the scenario are separated by a dashed line.

A scenario specification of a system will in general consist of a large number of such MSCs. In order to structure these scenarios, High-Level MSCs (HMSCs) can be used. Such an HMSC shows in which way the respective scenarios must be combined. An example of an HMSC is given in Figure 4). The HMSC in this figure expresses that first MSC *Setup* is executed. Then there are two alternative scenarios to continue with, namely *Metering* and *Emergency*. After having executed one of these, execution continues at MSC *Forced end* and finishes with *Disconnect*.

The high level description given by HMSCs can be used to give an abstract overview of the system. but it can also be used to design the system requirements top down by starting to describe the abstract behavior of the system and refining these HMSC until the basic MSC level is reached.

Other extra primitives on Basic MSC level; such as gates, timers and the fact the each basic MSC has its own unique ID enhance the expressive power of the MSC method and make modularisation and reuse possible. As the MSC method describes the interaction between processes it is specifically useful for modelling behavioral aspects of multiprocess systems. For data intensive systems MSC is not the ideal modelling tool.

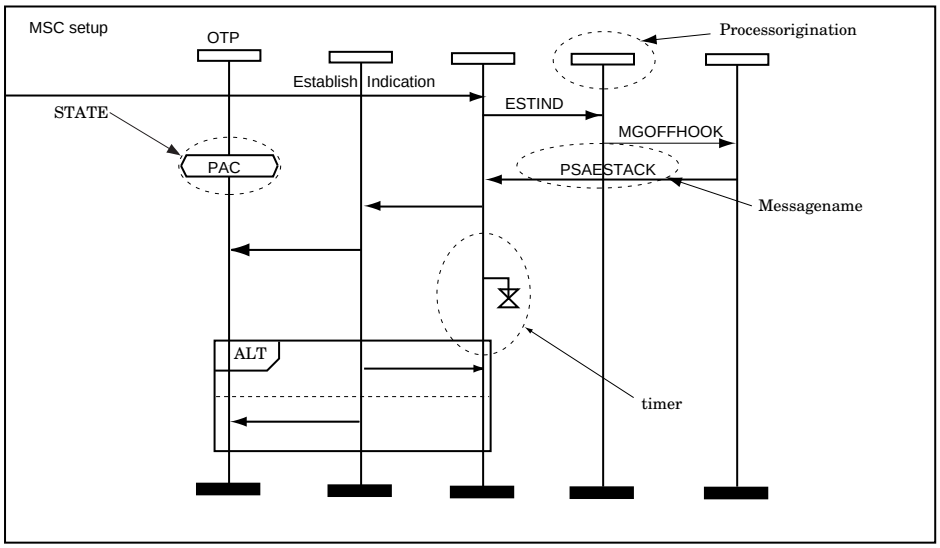

Figure 3: A basic MSC with explanatory comments

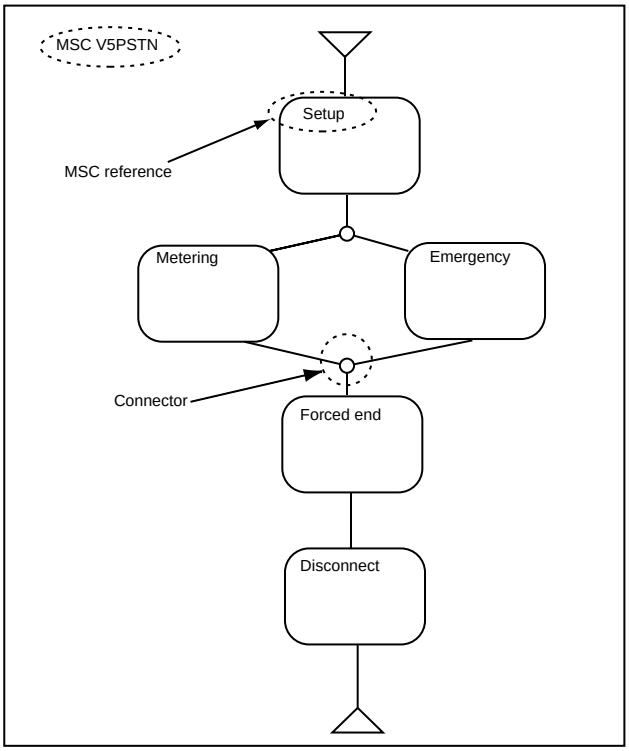

Figure 4: A High-level MSC

4 uBET

The uBET tool is developed by Lucent, Murray Hill. The toolset started of as an MSC editor for Basic MSC. It provided a means for quickly drawing processes, messages between processes, timers and states. These basic building blocks provided a good basis for drawing MSCs as described in MSC92.

Later in a graph editor was introduced that can be used to combine the Basic MSCs into larger structure. These graphs can recursively be used as part of a graph at a higher level making it a true hierarchical structure. The result of this combination is an MSC editor which incorporates important feature of the MSC96 description.

The structure of uBET allows for a top down approach of design as well as bottom up by not enforcing a order of design. Furthermore uBET offers a number of analysis possibilities. The tool is available on both PC and SUN platform and is freely available for all engineering groups within Lucent.

[Screen shot]

If we compare the uBET tool to full MSC we have to conclude that there are differences in both syntax and expressivity.

The main differences in syntax can be summarized as:

- the timer-set, -reset and -expiration events are denoted differently
- the edges in graphs of uBET are .... labelled, where as in HMSC the connections between MSC references are unlabelled.
- reference between a node in the graph and the scenario that 'refines' it is not denoted in the graphical representation but electronically in the model.

These differences in syntax.....

The main difference in expressiveness can be summarized as:

- no gates
- absence of refinement

Finally the syntactic sugar that is introduced in MSC such as inline expressions and the possibility to reference to multiple MSC with one MSC reference is not all implemented in the uBET-tool. The addition of this syntactic sugar to uBET would not be of influence on the expressiveness of the uBET method, but it substantially aid in the readability and clarity of the scenarios.

5 The uBET/MSC support possibilities

It is clear from the description of the current software development process (see Section 2 that scenarios play an important role in many respects. They are applied in many ways, such as requirements specification, documentation, design, and testcase specification. Currently, scenario specifications are being used in an informal and non-standardized way, with only elementary tool support.

Clearly, adoption of the recommended MSC standard will strengthen the use of scenarios in the development process. The advantages of using the MSC language as mentioned in Section 3 evidently apply to the situation at Lucent. There is, e.g. a clear need for composing larger and more complete scenarios from simple ones.

Of course, there will only be maximal benefit from the MSC method if appropriate tool support is provided for. Although the uBET tool is not fully compatible with the most recent ITU standard, we think that it can be effectively applied in the current development process.

Below, we will indicate at various places in the software process what the impact would be of using the MSC/uBET methodology. Rather than giving a complete and detailed description

of the use of MSC/uBET in these situations, we aim at making a global inventory of its use. A follow up in the form of a number of case studies is needed for understanding the full applicability of the MSC/uBET method.

System Engineering Starting at the system engineering phase the uBET-tool and MSC can be used to capture part of the requirements. These descriptions can be produced with aid of Software Development and Feature Test. The uBET description of these scenarios unambiguously defines the desired system behavior. The uBET tool can then be used to extract example scenarios for documentation.

The structure that can be used in describing the requirements will greatly aid in the readability of the requirements and will also aid in the communication of high level concepts and behavioral structures. Combining the descriptions of the different workitems, can clarify the cohesion between the different work items that need to be developed.

Software development As soon as part of the FSD is designed in a uBET model, this uBET model can be used by the software development group to understand the meaning of the requirements. The refinement of the requirements that are given in MSC can now be done by refining these MSCs. From these MSCs whitebox tests can be derived by using uBET to produce scenarios that are compliant with the MSC.

Again the refined MSCs can be included in the documentation to give an unambiguous description of the expected system behavior.

On the longer term as soon as part of the existing code is already available in MSCs, these existing MSCs can be reused and adapted to desired system description. This reusability does require that the MSC descriptions are kept up to date and that they have been checked to represent the current code.

Another possibility for support would be that the system specification in MSC can be used to analyse and verify the behavior of the system that has been built. This would however require a considerable amount of extra effort in tool development and an abstract description of the code in a formalism like VFSM or OBJECT-time.

Feature test The uBET description of the requirements in the FSD can help feature test to understand the meaning of the requirements since they structurize the information and give the possibility to interactively view the scenarios that are described by the uBET model.

The feature tester can choose to refine the uBET model that is given in FSD to come to a better system description as a basis for the test to be developed. The refinement made by feature test can then be compared to the one made by software development to come to clear understanding of possible mismatches in interpretation. Since the MSC language offers an unambiguous description the number of mismatches in interpretation can go down anyway.

On the long term as soon as the description of other features is given as uBET models as well t can be possible to automatically detect possible feature interactions and create the proper test scenarios.

6 Conclusions

Since scenarios already play an important role in a number of phases in the development process, the usage of MSC to describe these scenarios can greatly help. On the short term the usage of MSC (uBET) enables the designer to structurize the scenarios, aid in top down and bottom up design and help to communicate the scenarios with other groups. The automatic processing of scenarios from the Ubet models makes it possible to quickly use the editor even in the current Software design process.

A prerequisite for this is however that all involved engineers will have to learn how the MSC method works and -to a certain extent- what the underlying theory is like.

On a longer term, much profit can be gained from reusing scenarios. The MSC method supports this reuse by means of its constructs for modularization. The formal semantics guarantee an unambiguous description so interaction and compatibility with other support tools can be achieved.

The uBET tool is a sophisticated and user friendly tool that is compatible with the MSC method. Even though the syntax might differ a bit and the full expressiveness of MSC is not captured in the tool it can be a very useful tool to use for capturing part of the system requirements.

To fully use the power of MSC, the uBET tool will however have to be extended with extra functionality.

In addition to the usage of the uBET tool and MSC to denote scenarios and specify (part of) the requirements the software development process as a whole can be supported by formal methods and tools. If these tools are combined with some slight changes in the process structure, Lucent Technologies can greatly benefit from the power of formal methods.

7 Future work

The information in this paper has been based on literature studies, interviews with experts and some data from projects at Lucent in Malmsbury and Indian Hill. As a follow up to this paper we advise to do some experiments and projects using uBET in Hilversum, such that we can gain first hand experience

In order to get such an experiment going we would suggest to start up a small project where the engineers involved use uBET to specify their requirements and then look to what extend they can use these descriptions in other parts of the design process.

To fully use the MSC method, research is needed to find out whether and how MSC descriptions can be abstracted from the current code.

Acknowledgments

We would like to acknowledge Willem van Willigenburg and Jeroen Collard for their help and support. Furthermore we would like to thank Kanwilander Singh and Gerard Holzmann for the information they have provided. We thank André Engels for his support in describing the MSC language.

References

[1] ITU-TS. *ITU-TS Recommendation Z.100: Specification and Description Language (SDL).* ITU-TS, Geneva, 1988.

[2] ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Chart (MSC).* ITU-TS, Geneva, 1997.

[3] E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on message sequence charts (msc'96). In *FORTE*, 1996.