

Model based Systems Engineering - A Unified Approach using UML

Peter Braun¹, Martin Rapp²

¹Munich University of Technology, Department of Computer Science,
Arcisstr.21, D-80333 Munich

braunpe@in.tum.de

²Munich University of Technology, Department of Computer Science,
Arcisstr.21, D-80333 Munich

rappl@in.tum.de

Abstract. Dynamically controlled safety applications, passenger comfort, information and entertainment for operational convenience are the field of permanent innovation. In BMW's high performance luxury segment's vehicles, about 80 electronic control units are networked together to provide the driver with numerous features. To guarantee an appropriate level of quality within the allowed integration time of features, new and progressive techniques as well as notations are needed to support the early phases of development. The early phases of development include a maximum amount of potential to solve issues on the future development methodology. In addition to new notations reliably controlled integration processes for components with significantly different life-cycles including information management and change control are sketched. For these purposes BMW chose to utilize the Unified Modeling Language (UML) and to define its automotive-specific subset of UML. The structural part of the analysis model is specified by a state based specification notation to generate "executable requirements". To support our methodology, domain specific and commercial tools have been tightly integrated to support a seamless development process. Effective information management and change control are based on a sound and seamless management of the projects' requirements. Thus, the information model of the developed products is embedded in the requirements model.

INTRODUCTION

Though innovative features are key potentials to competitive advantage, their merit will be limited, if quality, cost and time-to-market constraints are not met. Customers' wishes and requirements and the systems' constraints set up the entirety of functional and non-functional requirements, that have to be met as a prerequisite for successful products. To be able to keep these requirements it has to be ensured, that they are completely elicited and documented to know them in their entirety, without ambiguities, incorrectness or contradictions, in an up-to date status. The requirements model evolves during product development, i.e. requirements continuously change, are updated, deleted or new ones are created.

To be able to control the "magic triangle" of quality, cost and time constraints reliably, these changes have to be integrated into the current requirements model, their impact has to be analyzed and evaluated, trade-offs have to be carried out, decisions have to be made and to be documented (EIA et al. 1999).

This paper presents results of our research towards an integrated development methodology and management principles of the requirements model (functional and non-functional) with notations of the automotive modeling language AML¹ in terms of processes, methods and tools. The selected tools for the targeted process are Telelogic's UML Suite (Telelogic 1999) for a model based requirements documentation, ETAS's ASCET-SD (ETAS 1998) for the specification of continuous as well as discrete and hybrid systems. All of these tools are underpinned by QSS's DOORS (Quality Systems & Software 1998) for managing the complete set of requirements.

For a better illustration of our methodology a prototypical run through our new development process is sketched in application to a running example. The example focuses on the sequencing development activities in the requirements engineering phase to elaborate a requirements model for the electronic control unit of an interior light control within a car.

The paper is organized as follows. The section *Requirements Engineering* describes the requirements engineering process for electronic control units including all notations and generated products. Essential steps of this process are illustrated by an example, the development of the interior lighting. Section *Requirements Management* introduces the basic concepts of requirements management and their deployment within the requirements engineering process. Finally, section *Conclusion* gives a survey of our work on an integrated tool suite, which is underpinning the described processes.

REQUIREMENTS ENGINEERING

Common development processes of electronic control units in the field of automotive systems are

¹ The AML comprises a subset of UML and ASCET-SD notations

logically structured into a number of sequencing phases. Each phase results in a set of artefacts, which represent a view on the system. Validation and verification activities cause feedback flows in this process to elaborate successively software for control units, which is working correct and which is fulfilling customer oriented functions.

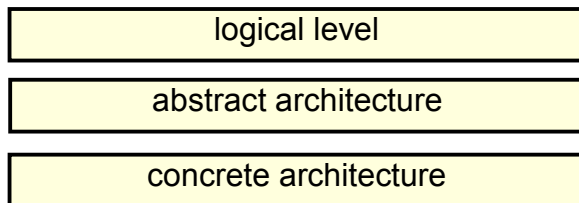


Fig. 1. Abstraction layers

Fig. 1 shows a common used classification scheme for informations (Keck et al. 1998). During the development cycle information in a different level of abstraction is collected by the extension of the systems information model. In our methodology we concentrate on the formation of user requirements models, which can be associated to the first abstraction layer. Models for system requirements and architecture design are associated with the abstract architecture layer resp. the concrete architecture layer (figure 5). We depict notations to represent user requirements and sketch processes as they are intended to get a standard for the BMW company. Aim of the initial development phase is the complete and unambiguous elicitation and the model based representation of requirements.

For a stable set of requirements, which serves as a basis for further development activities (see Requirements Management section), the requirements model has to be systematically worked out within three activities. The analysis process starts with the *elicitation* of requirements, continuous with the *structuring* of the initial set of requirements and concludes with the *specification* of requirements. For a model based representation of requirements we have to distinguish different classes of requirements. In this paper functional and nonfunctional requirements are differentiated (Stevens et al. 1998) (Partsch 1998) as it is usually done in the Requirements Engineering community. Both of them are originated by user, developer or system demands.

Functional Requirements

Functional requirements precisely define the behavior of the intended system and their representation can be done by a precise, mathematical, functional description. In the field of automotive systems, the functional description has to be done with respect, that electronic control units are reactive, mostly hybrid systems. Depending on the viewpoint of a system, functional requirements can be treated as functions the system has to fulfill (from the viewpoint of the system) or they can be treated as features, the system is offering (viewpoint of the user). All features (resp. functions) are working in

parallel and are interacting. The set of features can be understood as a functional network with communication relations defined by pure logical data dependencies of the functions.

In traditional methodical approaches, there exist a number of sequencing phases in the development process where different models with different views on the intended system are elaborated. All of them have the property that the representation is done in different languages with different levels of abstraction. Generally structural parts of the models are eliminated or modified by the transition from one model to another. These drawbacks have negative effects on the traceability of requirements. If there is a missing mapping between requirements and specifications of the system, there is a great additional expense, which has to be invested to relocate the exact system functionality in models of later development phases. Changes, as they occur every day, are very expensive to integrate.

A higher level of modularity is achieved by distinguishing between two different types of structure. The first kind is the structure of functional requirements (features) and the second kind is the structure of the architecture of the hardware system (network of control units). An architectural model of the control unit network is used to show the mapping of feature instances to each single control unit. In contrast to other methods the architecture is not derived from requirements by exploiting the architectural information which is contained in requirements. The architectural model is regarded as an abstraction of an existing network of control units, which is evolving as the consequence of the integration of new features. In our approach we are making use of the great amount of hardware independency which is common to most features. Within the activities elicitation, structuring and specification the set of features are consistently and completely worked out.

Features and their structure represent a logical encapsulation of functional requirements. Features occur in the design model as instances of their declaration in the requirements model. Traceability of requirements and the control of changes is facilitated.

For a comprehensive notation of features, different views on features have to be considered. These are structure (including interfaces), behavior and interaction communication. Due to the requirements of the initial development phase, features have to be represented in notations, which can be pragmatically applied in the first development phase. The elaborated documents serve as a communication basis between different stakeholders. Therefore the notations have to be intuitively understood. Graphical or textual formalisms are well suited.

For all of these views, the Unified Modeling Language (UML) (Object Management Group 1999) offers appropriate means of notation. All diagrams of the UML have been standardized by the OMG. Standardization is an important aspect for the

integration of notations in the development process of the BMW company and it is also very important for tools to support the notations. All diagrams of the UML are abstract, containing a wealth of constructs to fit into nearly every application domain. For an optimal adaptation of the UML to the field of automotive systems, the set of UML notations has been tailored to a subset, called automotive modeling language AML. Notations in AML are more or less orthogonal relating the kind of information they represent.

Activity	View	Notation
Elicitation	Structure/Interface Interaction Communication	Use Cases MSC's
Structuring	Structure	Class Diagrams
Specification	Behavior	Statecharts

Table 1. Using UML diagrams for a model based representation of requirements

Table 1 shows the three basic requirements engineering activities and correlates them with the different views on features. In the third column the used notations in the analysis phase of the automotive modeling language are listed.

Note: All diagrams, with respect to standardization, are restricted to time discrete, state based systems. For modeling hybrid systems, the discrete means of notation have to be replaced by their hybrid variants, e.g. hyMSC (Grosu et al. 1999) or hyCharts (Alur et al. 1995).

Nonfunctional Requirements

Nonfunctional requirements are the second category of requirements which have to be elicited in the initial development phase. In opposite to functional requirements there exists a great variety in classification approaches of nonfunctional requirements. A typical classification is the following:

1. Quality attributes of the elicited features
2. Requirements for the implemented system as a whole
3. General conditions of the systems development
4. Requirements for test, maintenance and operation

Nonfunctional requirements are usually documented in natural language. Certain kinds of nonfunctional requirements, namely the requirements which can be directly mapped to features typically evolve during its lifecycle. They can be described later on as features with automotive modeling language. A first classification has assigned them to the category of nonfunctional requirements, because they failed to be formalized by a functional description. In later steps, initiated by the existence of already elicited and specified features, they evolve to functional requirements. Quality attributes are typical

for these evolution. *Quality attributes* define the timing behavior, the dependability, the resilience and the robustness of features during execution. *Requirements for the implemented system as a whole* comprise all defaults and properties concerning the intended system and its components (f.e. realization in software or hardware, system topology, ...). *General conditions of the systems development* (process requirements) contain specific details and restrictions on the circumstances of the system development. Above the aforementioned classes of nonfunctional requirements there are requirements referring to the installation and the use of the system by the user (requirements for test, maintenance and operation).

In the sequel, the features for the interior light control are elaborated. Within the three requirements engineering activities elicitation, structuring and specification a model of functional requirements (features, functional network) is successively developed by using the notations of automotive UML. Because of the easy, well understood functionality, a first, rather simple model can be drawn as a statemachine with three states. All three states are directly dependent on the actual position of the interior light switch. In two cases, the interior light is switched on and off immediately, in the third case, the light is switched on and off in order to the position of the front doors. This basic functionality has been completed with features like get in assistance, get out assistance or emergency on. In addition to this, the features have been extended with timers and repetition protectors to ensure a proper work of the control unit. A behavioral model of this extended system includes about 30 states.

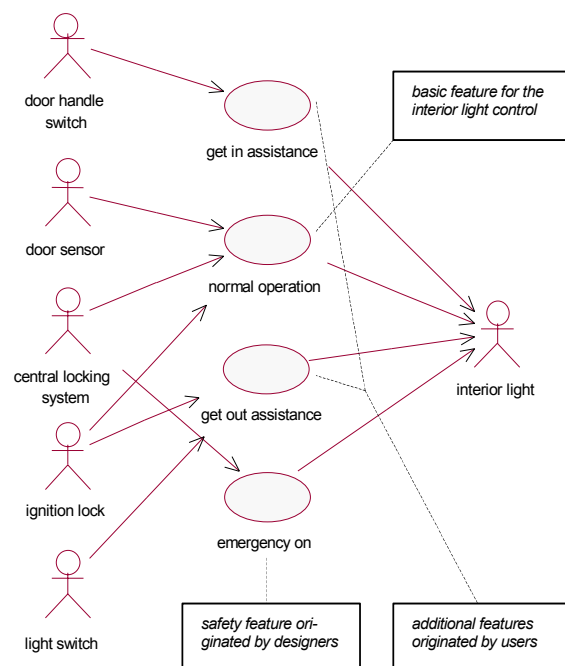


Fig. 2. Features - interior light controller

Activity 1: Elicitation of the features

In practice the development of electronic control

units is a process which is based on the modification and extension of development products, which emerge during the development of the preceding system. The new system is generally a system which is offering at least a certain set of features which is common with the scope of features of the old system. The use case diagram supports the acquisition of the set of features and the integration of new services. Every feature, resp. use case in the terms of the UML, is added to existing, already elicited features. The integration of new features to existing ones is done without regard to the execution of features. It doesn't matter if features have to be executed in mutual exclusion with other features or if there have to be other mechanisms to avoid any kind of interaction conflicts among features. To facilitate the enhancement of an existing use case diagram with new features, the analyst is supported by the assumption, that a daemon is controlling the parallel execution of features and is resolving any occurring conflicts.

This easy method of integrating new services implies, that changes of functional requirements have to be applied top down in the initial development phase. The method of treating functional requirements as features eases the way of pushing this changes into models of the later phases. Figure 2 shows the elaborated features for the interior light control unit.

To obtain a very first requirements specification which is accessible to different kinds of stakeholders, the elicited features have to be textually described. For the informal description it is highly recommended to use a structured text (description form) as suggested in (Östereich 1998) (Partsch 1998) (figure 3). By filling out this form during textual specification, all necessary information is captured to provide a basis for a specification with a more formal technique. At this stage it is the aim to elicit all potential actors, which are communicating with the system. Depending on the feature, actors may be elements of the user interface, sensors or actors. Each identified actor is part of a nondeterministic environment and has the capability to send out a set of signals with a unique signature. The interface of the deterministic system, which has to react on all possible incoming signals, is the result of the union of signatures, which are given by all incoming signals and all outgoing signals, which are fed back to the specific actor. Therefore the use case diagram is equivalent with the context diagram, known in structured methods, because of the precise separation between system and environment and its complete interface declaration.

After the elicitation of features and the identification of actors, exemplary scenarios of the use of a feature are specified. A scenario (Lamsweerde van. et al. 1998) is defined as a temporal sequence of interactions among different kind of actors and the feature to achieve a certain amount of system behavior. MSC's are used as a formal notation for representing scenarios by showing the communication interaction of the actors and the

feature along a timeline. Figure 4 shows an exemplary run of the feature get out assistance. MSC specifications are no comprehensive specifications of features. A MSC captures just one particular, fragmentary instance of the behavior of the feature.

Use Case	<i>Name of use case</i>
Actors	<i>List of actors</i>
Precondition	<i>Precondition</i>
Description	<i>Textual specification</i>
Sub Use Cases	<i>List of references to sub use cases</i>
Exceptions	<i>Reaction during exceptions</i>
Postcondition	<i>Postcondition</i>

Fig. 3. Textual use case (feature) specification

Activity 2: Structuring

The input for the second requirements engineering activity is a set of low level structured requirements, split up into functional and nonfunctional requirements. During requirements structuring, the different fragments of functional requirements models (the requirements structure) are put together into one model represented in class diagrams. Furthermore the elicited features are systematically decomposed into small, manageable and reusable objects with low complexity. Each leave object denotes an elementary feature and the logical communication relations between these objects form the functional network of features. Further relations among this objects have to be worked out properly. By the transition from the use case diagram to the class diagram structuring mechanisms like aggregation and inheritance relations are used to refine the requirements structure. One of the goal is to

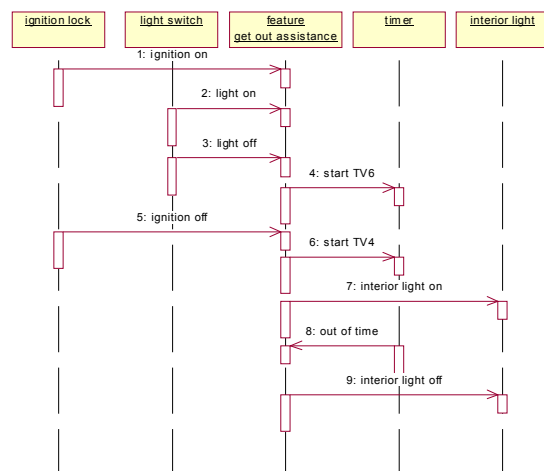


Fig. 4. Exemplary use of the feature get out assistance

achieve a hierarchical decomposition of objects, resp. classes and to exploit the reuse of objects within the

same diagram. Whenever decomposition is done by using the aggregation relation, it is done under the premise, that functional requirements are decomposed. System boundaries or other structural design information is not regarded.

Activity 3: Specification

At least the behavior of the atomic classes has to be specified with Statecharts. In the third activity no new knowledge is added to the requirements model. Behavioral diagrams are drawn by using all the information, which was elaborated in the preceding phases. This information has to be stepwise transformed from an informal, textual description into the formal statechart notation. By the restriction of the degree of freedom in the textual behavioral description (scenarios) a semiautomatic transition can be derived. In contrast to MSC specifications, statechart specifications are complete in the sense of mathematical computability. That means, that statechart specifications which are derived from MSC specifications have to be manually completed to describe all possible output on each input.

REQUIREMENTS MANAGEMENT

The information model of any project is represented by information of different maturity. Starting with vague wishes (market pull) or ideas (technology push) products have to be defined and developed. To efficiently handle changes, these different qualities of information have to be defined in their appropriate "levels". Any information within the information model has to be allocated to the concerning level. The level reflect the increasing maturity of data and thus the development process.

Requirements Flow-down

The only and most valuable information in projects' earliest phases are the customers' and users' needs and wishes or the developers' ideas. They state the user requirements for a "product". User requirements describe the WHATs of the desired product and the purpose of its existence. They are formulated independently of the later technical solution usually in the language of the customer or user.

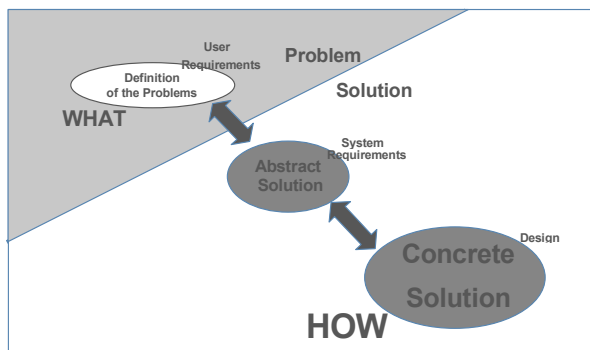


Fig. 5. Requirements flow-down

User requirements have to be translated into the language of the developers. By this translation

additional information is acquired, first decisions have to be made and the information matures. The requirements will be enriched with "technical" information and thus represent a certain content of the subsequent solution. The system requirements can be interpreted as a first abstract solution of the stated problem.

The most concrete information within the requirements model will be the architecture design, the description of the concrete solution. The architecture design already knows about the system's components and its partitioning in hardware and software. It can be seen as the HOW of the stated problem's solution. Figure 5 shows the requirements flow-down (Quality Systems & Software 1999).

Traceability

As information matures, the data model gets enriched and decision have been made, decisions and data will be changed. It is a key to successful development to effectively control these changes. Therefore any information has to be allocated to its concerning level or layer. Information of different quality cannot be handled within the same level of abstraction. The elements within the layers (user requirements, systems requirements, architecture design) have to be linked, i.e. their mutual implications have to be allocated and the layers have to be linked, i.e. the requirements model has to be enriched with context information like "Which part in the architecture design will cover which requirement?", "Which user requirement is the reason for the statement of a system requirement?", "Which test verifies the completion of a system requirement", etc.. Thus the implications of changes of each element of the requirements model can be traced to all other elements. Figure 6 shows the principle of information tracing.

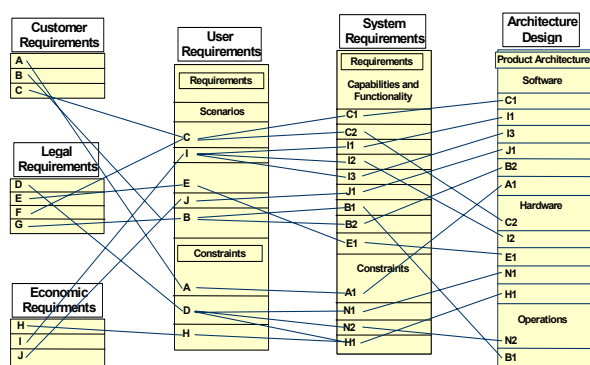


Fig. 6. Traceability

Development Process

For the early phases of a complete development process the results of requirements engineering, i.e. the analysis model and specification model, and the requirements management have to be integrated into one information model methodologically. This means, the elements of both have to be set in a sensible context.

Due to the quantity of data, this can only be

controlled by supporting tools. The goal to integrate the two information models into one requires the integration of the concerning engineering (analysis, specification) and project management (requirements management) tools. Within a project of the Bayerische Forschungsstiftung BMW, ETAS, QSS, Telelogic and the Technische Universität München are defining the methods and developing the tool-chain for a seamless design process.

CONCLUSION

In our work we sketched principles, how to put the development of electronic control units on a solid fundament of processes and notations in the early development phase. But if we want to use Requirements Engineering and Requirements Management techniques as a standard at the BMW company, we have to underpin our theory with a seamless tool chain. In respect to this, we are working with the tool vendors Telelogic, ETAS and QSS on an integrated tool suite, to support a seamless workflow management of the different models. With the integration of three different development tools for requirements engineering, requirements management and system design, we hope to be well prepared for future issues on the development methodology of embedded systems.

ACKNOWLEDGMENTS

We thank Bernd Gebhard (BMW) and Bernd Deifel for carefully reading a draft version of this paper. This work has partially been funded by the Bayerische Forschungsstiftung (BayFor) within the FORSOFT II project AUTOMOTIVE.

REFERENCES

- Alur, R., Courcoubetis, C., Halbwachs, N., Henzinger, T.A. and Sifakis J. *The algorithmic analysis of hybrid systems*, Theoretical Computer Science, 138:3-34, 1995.
- EIA, INCOSE and EPIC. *Systems Engineering Capability Model*, EIA/IS-731, 1999.
- ETAS. *Ascet-SD 4.0 - Entwicklungsumgebung für elektronische Steuergeräte*, 1998.
- Grosu, Radu, Krüger, Ingolf and Stauner, Thomas. *Hybrid Sequence Charts*, Technische Universität München, TUM-19914, 1999.
- Keck, Dirk and Kuehn, Paul. *The Feature and Service Interaction Problem in Telecommunications Systems: A Survey*, IEEE Transactions on Software Engineering, Vol. 24, No. 10, October 1998.
- Lamsweerde van, Axel and Willemot, Laurent. *Inferring Declarative Requirements Specifications from Operational Scenarios*, IEEE Transactions on Software Engineering, VOL. 24, NO. 12, December 1998.
- Leffingwell, Dean and Widrig, Don. *Managing Software Requirements - A unified Approach*, Addison Wesley, 2000.
- Object Management Group. *OMG Unified Modeling Language Specification (draft)*, Version 1.3

alphaR5, March 1999.

Österreich, Bernd, *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*. Oldenbourg, München, 1998.

Partsch, Helmut. *Requirements Engineering systematisch - Modellbildung für softwaregestützte Systeme*, Oldenbourg Verlag München, 1998.

Quality Systems & Software. *Keys to successful product development*, 1998.

Quality Systems & Software. *DOORS - Bringing you the vision*, 1999.

Stevens, Richard, Brook, Peter, Jackson, Ken and Arnold, Stuart, *Systems Engineering - Coping with complexity*, Prentice Hall Europe, 1998.

Telelogic: Telelogic Tau 3.4. *A great leap forward for openness, integrability and versatility*, 1999.

BIOGRAPHY



Peter Braun received the Dipl. Inf. Degree in computer science from the Technical University of Munich, Germany, in 1997. From 1998 he worked as a research assistant at the chair of Software and Systems Engineering – Prof. Broy. He is a member of the

project AUTOMOTIVE – Requirements Engineering for embedded systems. His research interests include formal and pragmatic methods for system specification and design, and the development of appropriate tools to support this methods.



Martin Rapp received the Dipl. Inf. degree in computer science from the Technical University of Munich, Germany, in 1998. From 1998 he worked as a research assistant at the chair of Software- and Systems Engineering – Prof. Broy. Since then he is a member of the Forschungsverbund für

Software Engineering. He heads the project AUTOMOTIVE – Requirements Engineering for embedded systems. His research interests include formal and pragmatic methods for system specification, requirements engineering and semantics.