Concretization and Formalization of Requirements for Automotive Embedded Software Systems Development

A. Fleischmann, J. Hartmann, C. Pfaller, M. Rappl, S. Rittmann, D. Wild
Technische Universität München, Fakultät für Informatik
Boltzmannstraße 3, D-85748 Garching bei München, Germany
{fleischa, hartmanj, pfaller, rappl, rittmann, wildd}@in.tum.de

Abstract

Within Requirements Engineering it is a difficult task to systematically increase the quality of individual requirements and the whole specification.
In this position paper we present our current research effort on a requirement engineering process for automotive software, which is an intermediate result of the mobilSoft project[1]*. In order to improve the requirements specification we propose the integration of conceptual dimensions of quality – namely concretization and formalization – into the Requirements Engineering Process (REP) and we illustrate how a process using these dimensions will guide and support achieving completeness of the specification.*

1. Introduction

It is well known that the majority of faults found in software programs can be traced back to mistakes in the analysis phase. Thus, Requirements Engineering plays a decisive part in the software development process. However, the tasks of the analysis phase are far away from being simple.

During the development of a system, the requirements of various stakeholders have to be integrated. A typical scenario could look as follows: In the beginning, the requirements engineer is confronted with a set of informal, incomplete, inconsistent, vaguely formulated requirements. These requirements are usually submitted in natural language (e.g. text documents, memos, notes) by various stakeholders having different views on the system. Moreover, when it comes to embedded systems, the requirements do not only refer to the system's functionality, but also to technical issues like interfaces and communication protocols. The analyst's task is to derive additional requirements, and to validate and verify the overall set. The result of this requirements specification process should be a description of the future system that is unambiguous, complete and consistent.

We are currently working on a framework supporting the stepwise transformation from informal and incomplete requirements to precise specifications. In this paper we focus on the aspects *concretization* and *formalization* of requirements, present some intermediate results and show, how they can be used in order to improve the quality of requirements descriptions.

The rest of this paper is structured as follows: In section 2 we introduce the domain of automotive embedded systems by describing specific challenges of this domain and by giving an example. Section 3 gives an overview of the different dimensions of quality for a specification, and how our work on concretization and formalization can be seen as means to address those quality issues. The next two sections (sections 4 and 5) describe the concretization and formalization in detail and constitute the main part of this paper. In section 6 we sketch, how the concepts of concretization and formalization can be synchronized and integrated into the REP. In the concluding sections 7, 8, and 9, we compare our results with related research, give a summary and outlook, and list the literature we refer to.

2. Automotive Embedded Systems

2.1. Specific Challenges

The domain of embedded automotive software systems implies specific challenges for Requirements Engineering, such as the need for real-time reactions, strict safety constraints, the need for robustness, a broad range of variants (product lines), and limited

[1] This work is partially funded by the Bavarian Government under grant license number IuK 100/188.

hardware resources [1]. In this section we describe two important specifics of embedded systems[2].

Embedded automotive systems usually do not have their own human user interface. Instead, they are operated through the user interface of the overall system. Hence, when describing the requirements of such an embedded system in terms of user interaction, the requirements engineer must not restrict the requirements to the system to be developed, but has to consider the entire system. For example, the main functionality of a *cruise control system* has to be described with respect to the overall system (as behavior of the car). As a consequence, the specification of an embedded system on the level of user requirements is tightly coupled with the specification of the whole system. On the lower levels of system requirements, this coupling has to be resolved by introducing a precise differentiation between the system and its interfaces.

The influence of existing hardware solutions is an important factor in the development of embedded automotive systems and has to be considered in the REP. For example, such a requirement might be: "*For determining the speed, the software has to run on a control unit XS-B.*" Such requirements have to be classified into the right concretization level and have to be properly connected within the tracing and justification structure of the specification. For example, the aforementioned requirement must be connected with a business requirement "*save money by reusing well proven hardware*" and must be connected with a system requirement on a higher level "*the system must be able to react within 2ms*", and it has to be checked, whether the suggested control unit is compliant (can fulfill) with this requirement.

2.2. A Running Example

In this section we introduce a small example, which will be used in the remainder of this paper to demonstrate problems when doing Requirements Engineering for embedded automotive systems and our approaches for their solutions.

Consider a *cruise control system* for a car [2] with the following functionality: The cruise control is activated by pressing a button. When activated, it comfortably (for example, not fitfully) accelerates or slows down until the vehicle's speed is at a value which has been specified by the driver (target speed). The vehicle then autonomously maintains this speed until the cruise control is deactivated; this is done by either braking or manually accelerating.

Three typical requirements for this speed control might be: (1) "*The system must not accelerate with more than 0.2g.*"(2) "*After pressing the brake, the cruise control must be deactivated within 2ms.*" (3) "*The driver must be able to easily change the target speed while driving.*"

3. Requirements Engineering Dimensions

Requirements Engineering aims at systematically increasing the quality of a specification (by using as few resources as possible). The quality of a specification can be defined and measured along many different criteria (dimensions of quality), such as structure, completeness, concretization, consistency, agreement, formalization and correctness (validation by stakeholder).

A REP that targets the highest quality of a specification has to take all these dimensions into account and integrates them as the foundation into its activity model and product model. In this paper, we focus on two of these dimensions, namely concretization and formalization; we describe each of them in detail (section 4 and 5), and we sketch, how they can be integrated into the REP (section 6).

The goal of **concretization** is to partition the set of requirements according to the abstraction levels they refer to and to build up a concretization hierarchy between the requirements. In such a concretization hierarchy, abstract requirements are described in more detail (and thus more clearly) with help of the concrete requirements they are linked with. The benefits of such a concretization are the ability to examine the logical refinement of requirements, to detect gaps in the justification of requirements and to complete the requirements set.

The goal of **formalization** is to formulate the description of a requirement (and maybe other attributes of a requirement, too) in mathematically defined semantics (e.g. sets, relations, state machines), so that the description is precise and can be automatically reasoned about. The benefits of formalization are: automatic checks for consistency, bridging the gap between requirements and design, better structuring of requirements and the specification, automatic test case generation and much more.

In the next two sections, we will examine these two dimensions in more detail.

[2] Please note that in the following we will make use of a *relative* notion of the term "system". I.e., depending to the scope, system may refer to the whole system or to subsystems.

4. Concretization Layers

In this section, we first give an overview of the five concretization layers and then we present each layers in detail.

It is important to note that our concretization layers are not meant as process steps (as others do, see section 7, and as we do with *formalization*, see section 5), but are meant as a sorting structure only.

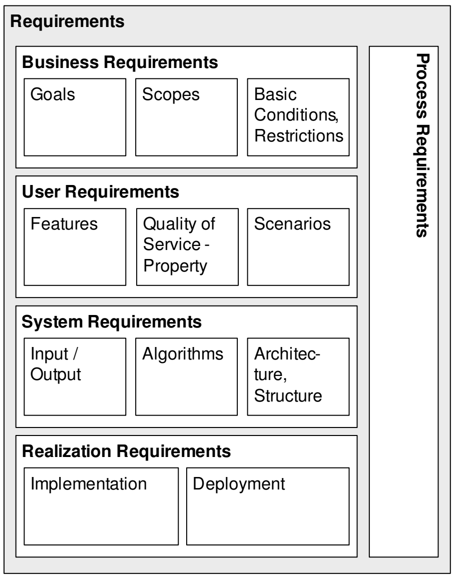

Figure 1. Concretization Layers

Figure 1 shows the five concretization levels (business requirements, user requirements, system requirements, realization requirements, and process requirements) and the most important information types on each layer. Each concretization layer has its individual characteristics. These are:

Business Requirements: In the business requirements layer, everything is about how the company might profit from the product to be developed; this profit might not be restricted to money, but can also refer to establishing a market, better the image or visibility of the company, and more.

On this layer, business goals are formulated, that the product shall achieve (for example, "*The product must be the door-opener for the Asian market*"). Also, basic conditions and restrictions are formulated, that shall ensure that the product will achieve the goals (for example, "*The projects budget is 600.000$*", or "*The product must not cost more than 150$*"). Also, on this level a first scoping takes place that coarsely defines the functionality of the product (for example, "*The product must better the driving comfort while driving long distances on a highway*"). Moreover, a target group, which is a set of users (such as driver, co-driver, and service personal at a garage), has to be defined.

User Requirements: In the user requirements layer, everything is about how a user should perceive and benefit from the product. In this layer, the externally visible functionality and properties of the product are formulated. Since the user mostly does not perceive an embedded system itself, the description of the product's functionality is embedded in the description of the overall system's behavior.

On this level, scenarios are formulated, that describe the functionality of the desired product in the context of the entire system and on the perspective of a user (for example, "*The user enters a high-way, he then defines a speed of 60mph, he then activates the cruise control*"). Those scenarios can later be used to generate test cases. The scenarios lead to basic features (such as "*it must be possible for the driver to set the target speed*") and quality-of-service-properties (for example "*The usage of the systems must be possible with both hands on the steering wheel*").

System Requirements: In the system requirements layer, the focus switches from the user's perspective (a set of perceived features and properties) to the system's perspective; it is dealt with the question, how the embedded system can perform those features in interaction with its environment. Hence, in this layer, the system and its interfaces to the environment are described. The requirements within this layer may be further structured in various sublevels corresponding to subsystems.

On this layer, the information flow and interfaces between the system and its environment are described (for example, "*The system must know the current speed of the car*", "*The system gets the current speed of the car by the sensor SSB via the CAN bus*", "*The speed is coded as 8bit integer*"), the behavior of the system is described (such as "*The system must calculate the difference of the actual speed an the target speed with a maximum tolerance of 1mph*"), and the architectural structure of the system is described (for example, "*The system shall consist of a scenario manager and a speed calculator component*").

Realization Requirements: Here, requirements are sorted in, which restrict the software implementation or the hardware deployment of a system.

On this layer, implementation constraints such as "*the component X must be programmed in C*" or "*in component Y, the certified code of program Z has to be reused*" and deployment constraints (for example "*the scenario manager and the speed calculator have to share one ECU*") are formulated.

There is also a **process requirements layer**, which is orthogonal to the other layers. It contains requirements that restrict only the way how the system is to be developed, but not directly the product (for example, "*The development process shall be conforming to the rational unified process*").

5. Formalization Steps

In this section, we describe the steps that have to be performed in order to transform informal requirements into formal ones. These four steps are: Identification (section 5.1), Normalization (section 5.2), Structuring (section 5.3), and Formalization (section 5.4).

As starting point, we expect informal textual information, such as structured or unstructured text, catchword lists, drawings, tables, and mind maps. Hence, we do not deal with the actual acquisition of requirements (e.g. in interviews, workshops) in this paper, and focus on the formalization of information instead.

5.1. Identification (Step 1)

Input: Informal textual information, such as structured or unstructured text, catchword lists, drawings, or tables.

Tasks: In a first step, requirements have to be extracted from the input documents. They have to be identified (and attached with an unique identifier) and separated into atomic requirements; they have to be put into a defined form and attributed (by using a template that might contain fields such as "id", "name", "description", "source", "date", and much more); often, they have to be written out in full, because the input information might just be a catchword or a short fragment of a sentence.

Output: A set of atomic requirements, written out and attached with attributes (as defined in a template).

Example: For example, a catchword list entry "*accelerate and slowing down*" might become two elaborated requirements.

5.2. Normalization (Step 2)

Input: In the first step, we separated and formatted individual requirements; each requirement is not yet aligned with the set of the other requirements.

Tasks: The requirements have to be walked through, and by analyzing the descriptions, a glossary has to be built. In this glossary, synonyms have to be resolved and a unified usage of terms has to be established.

Output: A set of requirements that uses the same terms for the same meanings, respectively; a glossary that defines the used terms and connects them with their synonyms and antonyms.

Example: In one requirement's description, there could be talked about "*the car's speed*" and in another requirement "*the speed of the vehicle*". These might be two terms for the same meaning ("*car*", "*vehicle*"), so it has to be resolved, if they indeed mean the same, and it has to be decided, which term will be used in the specification. In the glossary, the term will be defined, and its synonyms will be listed.

5.3. Structuring (Step 3)

Input: In the second step, we normalized the requirements, so that they use a set of uniform terms, which are defined in a glossary. Those requirements are written in a generic template and are not yet semantically captured.

Tasks: The normalized requirements now are structured by its contents in a taxonomy. That means that requirements can be grouped according to different aspects,. After that, those groups of requirements can be specifically dealt with, for example, by developing or reusing specific templates.

Output: A structure of requirements, which groups requirements that belong together within specific templates.

Example: Requirements that deal with the measurement of the vehicle's speed might be grouped together. There might be no specific template for measuring speed yet, but there might be a template for general measurements (containing slots for *minimum value, maximum value, increment size, reaction times, tolerance*, and so on), which can be reused and adapted to the measurement of speed. The slots in this specific template can be seen as both a checklist and a formal sorting of requirements to semantic issues.

5.4. Formalization (Step 4)

Input: In the third step, we produced a set of requirements, which are semantically captured by traces and by specific templates. However, the descriptions of those requirements are still informal.

Tasks: In this step, the informal description of the requirements will be transformed into a formal specification. Therefore, there exist several possibilities for formalization:

- Formalizing structure: We use AUTOFOCUS *system structure diagrams* [4] to formalize structure, so this target language consists of components, channels, ports, types, names. Other techniques for formalizing structure could be, for example, UML or Z [7].

- Formalizing behavior: We use AUTOFOCUS *state transition diagrams* [4] to capture behavior in a formal way. Hence the target language consists of states, transitions, events, pre- and post conditions. Any other state machine [e.g. 5, 6] could also be used.
- Formalizing interaction: We use AUTOFOCUS *extended event traces* [4] to formalize behavior, so this target language consists of components and interaction patterns. Other techniques for formalizing interactions could be UML sequence diagrams or message sequence charts (MSC).
- Formalizing data: We make use of AUTOFOCUS *data definition types* [4] to formalize data, so the target language consists of data types and operators.

Output: Fragments of models that contain the formalized information of the requirements descriptions; since some requirements might not be able to be formalized, those requirements stay informal.

Example: The requirement "*When the break is used, the cruise control is deactivated*" could be transformed into a fragment of a state machine with a state "*active*", another state "*not active*", and a transition from "*active*" to "*not active*" with the triggering condition "*break is used*".

6. Integrating Formalization and Concretization into the REP

In section 3 we listed a couple of important goals of Requirements Engineering, such as agreement (the requirements must reflect the opinions of all stakeholders), completeness (there must not be any vagueness abut the system to be build), preciseness (the requirements must not be misunderstood), and economy (the REP must use as few resources – such as money, time, people, skills – as possible).

In this section we show with the example goal "achieving completeness", how formalization and concretization layers together can be a powerful concept to guide and support the REP.

The goal "completeness" can be divided into two sub goals: the structural completeness of one requirement (for example, every requirement must be prioritized, every requirement must have an author) and the completeness of the whole requirements set (that is, that all information must be captured that is needed to build the right product). We show how concretization and formalization can help achieving these goals.

Structural completeness of individual requirements: The first formalization step, identification (see section 5.1), enforces a uniform representation of requirements by using a template. Hereby the structural completeness of requirements can be easily achieved because the template structure serves as a checklist for the requirements attributes. Additional verbalization templates (for example, that a requirement must be formulated in the form "*if … then … must …* ") can support formulating sound requirements.

Completeness of the whole specification: Sorting requirements into concretization layers is an important means to get at (and to attest) the completeness of a specification: The built hierarchy immediately indicates, whether high level requirements (such as business goals) have been broken down to concrete requirements or not; it also shows, when requirements on the lower levels (for example, system requirements) have no connection (and thus, no explicit justification) to a high level requirement.

The third formalization step, structuring (see section 5.3), groups requirements that belong together. Such groups can be, for example, "*requirements that deal with controlling the acceleration*", "*requirements that describe the modes of the system*", or "*requirements that describe the system's behavior in case of errors*". By grouping requirements under certain specific aspects, these groups, which should be kept small, can be better analyzed with regard to completeness.

The fourth formalization step, formalization (see section 5.4), is a further step towards the automatic detection of incompleteness. Here, requirements are translated into model fragments. When using state machines, for examples, it can be detected, whether there are missing transitions: this is an indicator that there are requirements missing that would specify the behavior of the system in case of an event.

In the beginning of this section we listed several goals of Requirements Engineering. Some of them are not addressed by the formalization and concretization layers at all - such as agreement or economy; here, other dimensions (for example, an agreement dimension) must be integrated into the process model. Others, such as preciseness, can be easily mapped to the formalization and are thus almost already covered.

7. Related Studies

The distinction of requirements into business, user, and system requirements is not new [9]. But in contrast to these approaches, our work is more detailed and more specific, since we do not provide a generic model, but focus on embedded automotive domain.

Pohl [3] introduced three dimensions of requirements engineering: “Agreement”, “Specification”, and “Representation”. These three dimensions are process oriented. The dimensions, we focused on in this paper, differ from Pohl’s dimensions. They can somewhat be seen as a detailed view on Specification (our concretization as an important aspect and technique to accomplish high specification values) and Representation (our formalization can be seen as a more elaborated and more specific approach in representing a specification).

The goal-oriented requirements approach KAOS [6] shows several similarities to our work (each requirement must be judged by a goal, the requirements are formalized, etc.). Due to different types of goals and the formalization of requirements, the KAOS approach wants to reduce the chaos arising of many informal, contradictory requirements. In our work, this is achieved by abstraction layers justifying the specifics characteristics of requirements for embedded systems. Additionally, KAOS uses a formal specification via (real time) temporal logic. Instead, we make use of more intuitive models like sequence diagrams and automata.

Over the last years various specification techniques like state charts [5], Z [7], and SDL [8] have been developed. All have in common that they describe a language which can be used to specify the system, but they do not focus on the process, i.e. how to get a formal model out of a collection of heterogeneous, inconsistent and incomplete requirements as described in this paper.

8. Summary and Outlook

In this paper, we introduced and explained two dimensions of quality for requirements engineering: concretization and formalization. We showed four formalization steps, which transform informal text into a partial formal specification; we explained concretization layers which are specific for embedded automotive systems. Additionally we showed how we plan to integrate these two dimensions into a REP.

In future work, we will elaborate both the dimensions and their integration into the REP. We also will introduce, explain and integrate more dimensions, such as the agreement dimension, and we will show, how all these dimensions together, as underlying concepts for a process, guide achieving all requirements engineering goals as mentioned in section 6. It will be the basis for developing an appropriate activity model and product model for a REP for the domain of embedded automotive systems.

Furthermore, we will investigate how high-quality requirements (like safety, reliability, maintainability) can be treated in our framework. This includes how they can be broken down to functional requirements, how functional requirements can be derived from them, and to what extend they can be formalized. Traceability between requirements on and within the abstraction layers is also in the scope of future work.

9. References

[1] A. Fleischmann, E. Geisberger, and M. Pister, *Herausforderungen für das Requirements Engineering eingebetteter Systeme*, Technischer Bericht der TU München, TUM-I0414. Munich, 2004.

[2] W. B. Ribbens (ed.), *Understanding Automotive Electronics*, 6th edition, Newnes Press, 2003.

[3] K. Pohl, The Three Dimensions of Requirements Engineering, In: *Fifth International Conference on Advanced Information Systems Engineering (CAiSE'93)*, Springer-Verlag, Paris, 1993, pages 275-292.

[4] F. Huber, B. Schätz, A. Schmidt, and K. Spies, AutoFOCUS: A Tool for Distributed Systems Specification. In: *Proceedings FTRTFT'96 - Formal Techniques in Real-Time and Fault-Tolerant Systems*, LNCS 1135, Springer-Verlag, 1996, pages 467- 470.

[5] D. Harel and M. Politi, *Modeling Reactive Systems with Statecharts: The STATEMATE Approach*, McGraw-Hill, 1998.

[6] A. Lamsweerde, Goal-Oriented Requirements Engineering: A Guided Tour**,** 5th IEEE International Symposium on Requirements Engineering, Toronto, August, 2001

[7] ISO/IEC 13568, *Information technology – Z formal specification notation – Syntax, type system and semantics*, International Standard ISO/IEC, 2002.

[8] CCITT, *Specification and description language (SDL)*, CCITT Recommendation Z.100 (03/93), 1993.

[9] Richard Stevens, Peter Brook, Ken Jackson, and Stuart Arnold, *Systems Engineering*, Prentice Hall Europe, ISBN 0-13-095085-8, 1998.