Prof. Dr. Manfred Broy
Prof. Dr. Werner Damm
Dr. Stefan Henkler
Prof. Dr. Klaus Pohl
Andreas Vogelsang
Dr. Thorsten Weyer

3 Introduction to the SPES Modeling Framework

Today's and, even more so, the future development of embedded systems faces a variety of challenges. Key success factors to meeting these challenges are suitable concepts for abstraction and structure at different levels of granularity. The result of these concepts is a seamless development approach that heavily facilitates reuse and automation. A basic requirement for such a seamless approach is a clear notion of a system that is formalized by a comprehensive modeling theory. According to this modeling theory, a modeling framework has to provide appropriate models and description techniques for modeling the different aspects and artifacts of system development. This section explains these conclusions and introduces the idea of system and the modeling framework. It also references the modeling theories used in SPES.

3.1 Motivation for the SPES Modeling Framework

The aim of model-based development is to use models as main development artifacts in all phases of the development process. It promises to increase the productivity and the quality of the software development process by raising the level of abstraction at which the development is done, as well as the degree of automation, with the help of models that are tailored and appropriate for specific development tasks.

Current model-based approaches

Even though adopted in practical development of embedded systems today, model-based development approaches often fail due to the lack of sufficiently powerful modeling theories and missing integration of theories, methods, and tools. The models applied in the development process are based on separate and unrelated modeling theories (if foundations are given at all), which makes the transition from one model to another unclear and error-prone.

3.2 Characteristics of Software-Intensive Embedded Systems

Embedded systems and the SPES modeling framework

An embedded system can be characterized as a technical system that operates in a physical and technical environment and is built by means of technical resources that collaborate in order to achieve an overall purpose (see [Braun et al. 2010]). Embedded systems monitor and control their environment using variables that refer to specific properties of the environment (e.g., physical or technical properties; see [Parnas and Madey 1995]). IEEE Standard 1362 states that a system can be characterized as "software-intensive" if the software of the system is the major technical challenge and perhaps the major factor that affects its schedule, cost, and risk (see [IEEE 1362]). Typically, software-intensive embedded systems consist of software and hardware.

Software-intensive embedded systems are widespread in our daily life. They can be found in many application domains such as automation, healthcare, consumer electronics, avionics, transportation, and automotive.

Addressed characteristics of embedded systems

Software-intensive embedded systems exhibit some characteristics that have a far-reaching impact on the corresponding engineering and modeling approach with which they are developed:

- *Multifunctional:* Software-intensive systems provide a wide range of functionalities, i.e., they offer a variety of functions that interact with the environment and additionally with each other (see [Broy 2010]).
- *Complex:* A significant increase in the complexity of software-intensive embedded systems can be observed over the last years. This corresponds with the effect that perceivable functions are increasingly being realized by integrating fine-grained software-intensive embedded subsystems. Therefore, the complexity of such systems increases in two ways: the complexity of a single subsystem (intrasystem complexity) and the complexity of the relationship to other subsystems (intersystem complexity).
- *Reactive and interactive:* Software-intensive embedded systems are generally interactive or reactive systems. They are characterized by the constant interaction and synchronization between the system and its environment. While for interactive systems the interaction is determined by the system, the interaction of reactive systems is determined by the physical-technical environment.
- *Distributed:* In many cases, software-intensive embedded systems are no longer realized within a single electronic control unit (ECU) but distributed over a network of logical or physical components that interact with each other heavily in order to realize the desired functionality. These components execute their computations simultaneously on multiple cores in the same chip, with different threads on the same processor, or on physically distributed systems.
- *Control of continuous physical and technical processes:* Software-intensive embedded systems are frequently used to control physical processes and devices that exhibit a time-continuous behavior. The controller within the software-intensive embedded system is implemented in software and is consequently asynchronous and time-discrete. An engineering methodology for developing embedded systems must accommodate for that fact.
- *Exhibiting real-time properties:* Many software-intensive embedded systems must meet real-time requirements during system operation, for instance, to be able to guarantee the safety of the occupants within a vehicle. In such cases, the system must fulfill certain constraints that restrict the time behavior of the system's response if a specific crucial event in the environment of the system occurs.
- *Safety-critical:* Safety is a major quality of embedded systems that must be considered in any activity during engineering. Safety can be characterized as the extent to which the system under development will not have effects on its environment that result in harm to people,

significant monetary losses, or any other negative impacts to its environment.

3.3 The Principles of the SPES Modeling Framework

The overall requirements for the SPES engineering methodology as described in Chapter 2 led to a set of fundamental principles for the SPES modeling framework. These principles aimed at establishing specific ways of thinking to be applied when performing the modeling activities suggested by the engineering process for software-intensive embedded systems in order to meet the requirements from the application domains and the characteristics of such systems. These principles are:

- ❑ *Distinguishing between problem and solution:* This principle aims at distinguishing between the analysis of the underlying problem that has to be solved and the development of an appropriate solution in the form of a software-intensive embedded system.
- ❑ *Explicitly considering system decomposition:* Decomposition plays an important role as a lever to master complexity in nearly all engineering activities. It encompasses, for example, the decomposition of systems into subsystems, of functions into subfunctions, or the decomposition of hardware topologies. Following the decomposition of the system, its engineering process can be divided into a number of individual fine-grained engineering processes, complemented by certain activities to support the integration of the various engineering artifacts.
- ❑ *Seamless model-based engineering:* This principle aims at establishing a continuous model-based documentation or specification of all the information that is created during the different engineering activities. The notion “model-based” is used in two related ways. Firstly, the conceptual structures of the artifacts are defined by metamodels that specify the information structure of the artifact as well as the structural dependencies between artifacts. Secondly, the relevant information is documented or specified using conceptual modeling languages. In addition, an engineering approach can be characterized as “seamless” if relations between different types of artifacts exist and have clearly defined (formal) semantics. This makes it possible to use the models not only as documentation but also to perform automatic analysis and to transform one model into another.

- *Distinguishing between logical and technical solutions:* This principle aims at separating the logical solution, which focuses on general solution concepts and corresponding conceptual properties of the system under development, from technological constraints and technological design decisions. By following this principle, the engineers can clearly separate the logical solution, which is largely independent of technological constraints and technology-related design decisions, from the actual technical solution for the system under development. Due to the fact that the logical solution is largely independent of technological design decisions, the logical solution is more stable than the technical solution and can be reused for different technical realizations.
- *Continuous engineering of crosscutting system properties:* This principle aims at establishing the ability to consider crosscutting properties of the system under development. Typical crosscutting properties are safety or real-time properties of the system: they must be considered in any engineering activity and the corresponding artifacts, such as requirements, design, and implementation artifacts.

3.4 Core Concepts of the SPES Modeling Framework

To establish the fundamental principles mentioned above, the SPES modeling framework uses the following core concepts.

3.4.1 Abstraction Layers

Advantages of using abstraction layers

A system under development or a design element (e.g., a logical component) can be modeled on different abstraction layers. Less abstract models belonging to a lower abstraction layer may increase the level of detail of the description of the design element at hand. For example, the interface description may be refined as well as the behavior demanded. Therefore, increasing the level of detail, and at the same time decreasing the layer of abstraction, adds knowledge about the design element. Often, a reduction in the level of abstraction is accompanied by a refined granularity — that is, structurally significant design elements are decomposed into multiple finer grained items in order to keep them at a manageable size. Refining the granularity is, however, not a necessary condition when introducing a lower layer of abstraction. For example, it is possible to describe the same design item on two different layers of abstraction with the same granularity, but specify a certain aspect more precisely. Despite an increasing level of detail, another motivation for introducing a dedicated abstraction layer can be the handover of an initial system specification to another organizational unit. In this way, the initial specification remains unchanged and the design element is refined on a new abstraction layer. Mappings between engineering artifacts allow these refinements to be traced. While models on lower abstraction layers provide more detail, the design elements still have to respect the aspects specified for their higher level counterparts.

3.4.2 Views and Viewpoints

Views and viewpoints according to [IEEE 1471]

Multiple stakeholders with different concerns are involved in the engineering process for a software-intensive embedded system. The aim of the concept of viewpoints is to separate the various concerns of different stakeholders during the engineering process. It serves as a construct for managing the different artifacts during the engineering process. IEEE Standard 1471 [IEEE 1471] characterizes viewpoints as a specification of the conventions for constructing and using a view. In other words, a viewpoint is a pattern or template that can be used to develop individual views on a system (and its environment). Typically, the specification of a viewpoint defines that viewpoint in terms of its syntax, semantics, and pragmatics by providing, among other things, the name of the viewpoint, the corresponding stakeholder concerns, the viewpoint language (probably given by a metamodel), and techniques that can be used during the construction and analysis of the corresponding view (see [IEEE 1471]). Given a viewpoint specification, a view can be characterized as a concrete model of the system that represents the information that is relevant for the corresponding viewpoint concerns by using the conceptual structure of the underlying viewpoint language.

3.5 The SPES Modeling Framework

While software-intensive embedded systems are becoming more and more complex, market pressure requires companies to develop high-quality products in a short time. To deal with such complexity, software and systems engineering approaches propagate a structured, well-defined design process consisting of several steps based on abstraction and refinement techniques (e.g., [Sage and Rouse 2009, Sommerville 2010]).

Structuring the problem space

Abstraction allows the designer to concentrate on the essentials of a problem. Refinement adds detail to abstract models while preserving properties established on the more abstract level. In other words, a more concrete description of a design entity also has to fulfill all requirements of the abstract design entity it has been derived from (see Section 3.4.1).

Following the principle of separation of concerns [Dijkstra 1976, Tarr et al. 1999], the concept of viewpoints that provide artifacts and rules for describing different views of the system under development should be supported. A view is created by a set of models that describe the system under development and/or its parts, and is related to a viewpoint (see Section 3.4.2).

Structuring the solution space

In a complex design process, it is important to have a clear idea of decomposition in order to be able to ensure that the final implementation meets the requirements. A decomposition relationship within the SPES modeling framework introduces a level of interconnected subparts whose collaboration shall provide any functionality the decomposed unit shall provide. Engineering activities for the single parts should be largely independent of each other: firstly, to simplify work by limiting the required focus of attention, and secondly to enable development in a distributed fashion. Another point to note is that parts can be exchanged consistently provided that the part's original specification is also fulfilled by the new part (see Section 3.3).

While many modeling approaches partially support such concepts, they often do not cover the whole engineering space from initial requirements down to a final implementation. This means, for example, that these approaches cannot continuously consider crosscutting system properties (see Section 3.3) as they do not support traceability in the design process. Furthermore, none of these approaches consider a well-defined combination of abstraction layers and viewpoints at all. It is the aim of this work to provide a modeling framework that does not suffer from these shortcomings.

Using abstraction layers and viewpoints in the SPES modeling framework

In our approach, *abstraction layers* and *viewpoints* form a two-dimensional engineering space (see Fig. 3-1). Based on well-understood software engineering approaches, the SPES modeling framework focuses on the following viewpoints to support views, starting with solution-neutral requirements through to concrete technical solutions: *requirements* (see Section 3.5.1), *functional* (see Section 3.5.2), *logical* (see Section 3.5.3), and *technical* (see Section 3.5.4). Note that our approach is not limited to these viewpoints in principle. For example, in other application scenarios a geometrical viewpoint is required [Baumgart et al. 2011]. The requirements, functional, and logical viewpoints are especially pertinent in software engineering, also for systems with no technical background. The need to also support a technical viewpoint is driven by the fact that the SPES modeling framework considers software-intensive embedded systems in which the software is affected by the physical environment of the system and has to interact with (or react to) the surrounding technical system (see Section 3.2).

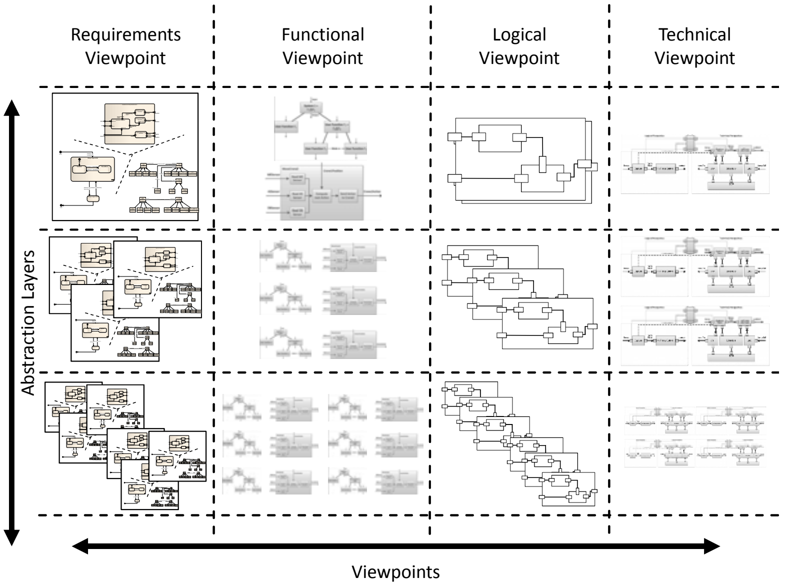

Fig. 3-1 *SPES Modeling Framework*

While the four viewpoints mentioned in the SPES modeling framework are the same for the different domains, the abstraction layers differ [Baumgart et al. 2010, Baumgart et al. 2011, Sikora et al. 2012]. For example, in the avionics domain, the abstraction layers *aircraft*, *system*, *subsystem*, *sub-subsystem*, *component,* and *unit* can be found, and in the automotive domain, the layers *supersystem*, *system*, *subsystem*, and *hardware/software component*. Hence, our approach abstracts from these different layers by supporting user-defined layers.

The SPES modeling framework is designed to be independent of any application domain (e.g., automation, automotive, avionics, energy, and medical). It defines fundamental concepts and how these concepts are related to one another. Domain-specific metamodels are interpreted in these concepts, thereby making general analysis techniques available for the specific application domain. In the following, we provide an overview of the viewpoints in Sections 3.5.1 to 3.5.4. In Section 3.5.5, we consider the relationship between the viewpoints that allows the seamless integration of the viewpoints into a comprehensive modeling approach.

3.5.1 Requirements Viewpoint

From system context to solution-oriented requirements

The goal of the requirements viewpoint is to support the requirements engineering process in a development project in eliciting, documenting, negotiating, validating, and managing requirements for the system under development. These requirements are derived from the system's context. The context of the system comprises entities from within the environment of the system, such as users, stakeholders, and external systems, but also legal documents. It also comprises physical properties of the environment that affect the system or are affected by the system in some way. Since these context entities are hence related in specific ways to the system under development, they must also be considered when eliciting requirements. The requirements viewpoint provides a requirements artifact model that allows for a systematic consideration of the context, the entities therein, and the resulting requirements. The basic aim of the artifact model of the requirements viewpoint is to capture the requirements from the system's context completely and correctly and to provide a means for structuring these artifacts on different levels of abstraction (cf. Section 3.4.1).

Requirements artifacts

The artifact model comprises a number of artifact types that are briefly explained in the following:

- *The context model* regards the system as a black box and documents the context of the system under development. It provides the basis for systematically eliciting the requirements that the system under development must satisfy during operation in order to meet its overall purpose. Context models are well suited for use as a foundation when performing activities for validating the correctness of requirements (see [Weyer 2011]).
- *The goal model* documents the stakeholders' goals with regard to the system under development (see [Levenson 2000, Lamsweerde 2009]). Goals can be elicited by analyzing the system's interaction with entities in the context and they serve as a rationale for more concrete requirements.
- *The scenario model* documents examples of concrete interactions between the system under development and its context. Each scenario describes an example in which at least one goal is satisfied. Scenarios can also be used to elicit new system goals [Potts 1995, Yu 1997].
- *The solution-oriented requirements model* documents the concrete and complete technical requirements that the system under development has to realize in order to satisfy its purpose during operation. These requirements must be as precise as possible to serve

as a foundation for the realization of the system under development. For solution-oriented requirements models, the SPES modeling framework distinguishes between three complementary model types (see [Davis 1993]): the structural requirements model documents requirements that describe the structure of the information that is exchanged between the system and its environment; the behavioral requirements model describes the externally visible behavior of the system by documenting the externally recognizable state space and corresponding state transitions; the operational requirements model documents required system functions by considering the functional relation between incoming and outgoing flows of information as well as the necessary control flow.

The requirements viewpoint is explained in detail in Chapter 4.

3.5.2 Functional Viewpoint

The purpose of a system is to offer a set of user functions [Broy et al. 2007]. Typical systems offer a number of different user functions and each user function serves a specific purpose. A system function is characterized by a particular observable system behavior in terms of specific interactions between inputs to the system (e.g., via sensors or user actions) and outputs of the system (specific effects on actuators, general reactions). The observations are captured in terms of the primary events of the user function (see [Broy 2010]).

User functions vs. realization functions

An example of a user function in a car might be an adaptive cruise control (ACC) or a cooling control for the engine. In both cases, the behavior of the user functions can be defined via the inputs of the system from the environment and the outputs of the system to the environment. Thus, both user functions are user functions of the system *Car*. All user functions of the system *Car* are structured in the functional black box model within the functional viewpoint. In contrast to this notion, a functionality that is needed to fulfill the user function is not considered as a user function of the system *Car*. Let us assume that the ACC user function is realized such that at one stage, the input of different speed sensors has to be aggregated. We could consider *SensorAggregation* as a user function of the system *Car* as well. However, this is not our understanding of a user function in the SPES modeling framework. Instead, *SensorAggregation* is considered part of a description of an abstract realization of the system *Car* (cf. Section 3.5.3). These parts are called functions (in contrast to user functions) and are specified in the functional white box model of the functional viewpoint. As the name suggests, in the white box model, user functions of the black box model are described by an abstract description of their realization. This differentiation will become clearer within the following sections about the functional viewpoint and its relation to the logical viewpoint.

The functional viewpoint integrates the set of user functions into a comprehensive model of the system functionality. This functional model describes the system behavior as it is observed at the system boundary. In contrast to the requirements viewpoint, where requirements are captured with respect to a certain usage context and at a certain level of granularity, the models of the functional viewpoint integrate these requirements into a comprehensive system specification. This especially includes behavior that arises from the complex interplay of different user functions.

User functions

We define the notion "user function" as a concept that has a specific purpose and corresponds to a determined behavior in the form of an interaction across the system boundaries. In addition, user functions typically have identifying names. The behavior of a user function can be captured by a behavior specification.

If there are dependencies between user functions such that the output of a user function depends implicitly on the behavior of another user function, this is referred to as a functional dependency or feature interaction and we model this using *modes.* This will be discussed in Chapter 6.

Hierarchies of user functions

According to this idea, we describe the functionality of a system under development using functional hierarchies in which we combine user functions into functional groups. The leaves of the resulting hierarchical structure correspond to individual atomic user functions. A functional hierarchy specifies the functionality of the system under development at a specific level of abstraction. The granularity that is chosen for a function hierarchy depends on the choice and the skill of the developer. The more intelligently the functional hierarchy is chosen, the more independently the user functions can be described, and the clearer the functional dependencies between the user functions captured by the modes.

3.5.3 Logical Viewpoint

In order to realize the desired functionality that is specified in the models of the functional viewpoint, the developer has to think about a decomposition of the system under development into an architecture of logical components. The logical viewpoint describes this glass box structural decomposition of the system, whereas the functional black box model of the functional viewpoint in particular focuses purely on describing the black box behavior (see [Schätz 2005]). The result is a description of the logical solution independent from any technological constraints. This description can be reused for multiple platforms. The reasons for decomposing the system under development into subsystems are manifold. In addition to mastering the complexity, further aims of the logical viewpoint are the division of labor and in particular, improving the capability of reuse. Grouping functionality that contributes to the realization of multiple user functions into one subsystem can save development costs and increase quality (see [Lim 2002]).

Logical component architecture

A logical component architecture as described in the logical viewpoint consists of a number of logical components that are connected via logical channels. Logical components exchange data via their logical channels, in the sense of a data flow architecture. Logical architectures can be structured hierarchically such that coarse-grained logical components are themselves again broken down into fine-grained logical components. At the level at which subsystems should not be further broken down, they can in turn be described by behavior description techniques such as state machines with input and output. The decomposition of a system into logical components is a starting point for the next iteration of eliciting requirements and defining user functions for each logical component.

The behavior of individual logical components can—similar to user functions—be represented by behavior descriptions (e.g., state machines). These behavior descriptions implement the individual logical components and their logical behavior. If we manage to capture all of these logical components in their logical behavior using state machines, a purely logical system simulation can be performed.

Relationship between user functions and logical components

An important question is how the relationship between the user functions and the logical components of the logical component architecture can be used methodically. Typically, only a portion of the logical component architecture is needed to provide a specific user function. We can thus define micro-architectures that show the portion of the logical component architecture that is relevant for a user function. This is interesting insofar as it may ultimately determine which of the logical components are involved in the provision of a user function. Particularly appealing is the representation of the modes from the hierarchy to the level of logical structure and the function of their technical components.

3.5.4 Technical Viewpoint

The technical viewpoint combines the software of the system under development with its hardware. It thus contains a description of the physical architecture. A deployment mapping specifies where software tasks of the logical viewpoint are executed and where and how logical subsystems are realized. The viewpoint considers aspects such as timing, resource consumption, and redundancy insofar as these have not been addressed before.

Goals

The main goals of the technical viewpoint are:

- Providing a description of the target hardware, with ECUs, memory, communication infrastructure, and peripheral devices
- Fixing the deployment of software modules
- Realizing logical subsystems
- Studying the interaction of software and hardware
- Ensuring that the behavior conforms to the specifications of the logical viewpoint, and that constraints concerning timing, independency, etc. are observed

The technical architecture comprises components for information processing (including communication) and their connection to the environment via sensors and actuators. The nature of the controlled system may have a considerable impact on the structure of the architecture and the characteristics of the information-processing components. These components will usually be described in the form of models abstracting from the details of the actual components that are used. The models also cover relevant services of operating systems, middleware, and so on. There are elaborated domain-specific approaches such as AUTOSAR (automotive) or IMA (avionics) that will often be employed.

The realization of the logical viewpoint via the deployment mapping results in a description of the system close to its final implementation. Therefore, properties that have been specified abstractly in previous engineering steps must be shown to have been realized in the technical architecture. Most prominently, these concern resource consumption, timeliness, and issues such as reliability and availability. For instance, tasks may be regarded as independently executable in the models of the logical viewpoint. However, if they are allocated on the same computing resource, they now have to be scheduled in a way that is consistent with all requirements. Communication, which was modeled as point-to-point and without delay in the logical viewpoint, has to be shown as appropriately realized by shared media such as busses.

Thus, the technical viewpoint studies the properties of the final implementation and has to establish that the physical realization meets all logical requirements.

3.5.5 Relation between Viewpoints

Relating the requirements and functional viewpoints

The requirements viewpoint introduces context factors, such as external stimuli or usage factors imposed by users or other systems, into the development process and establishes them as part of the requirements of the system under development. It therefore serves as a starting point for other viewpoints in the SPES modeling approach, as it specifies requirements that must be adhered to by other viewpoints. For example, solution-oriented requirements models of the requirements viewpoint must be fulfilled by user functions that are integrated in the functional hierarchy developed in the functional viewpoint. Furthermore, the external interfaces specified in the context models must correspond to logical (external) interfaces in the functional viewpoint as well as in the logical viewpoint. However, this is not a strict top-down process. Subsequent development activities in other viewpoints may require artifacts from the requirements viewpoint to be altered, modified, removed, or extended. Furthermore, the system context may also directly affect viewpoints other than the requirements viewpoint. For example, if the system to be developed has to be integrated into an existing environment consisting of legacy systems, these systems will inevitably affect the technical viewpoint.

Relating the functional and logical viewpoints

The logical viewpoint describes the internal logical structure of the system by means of communicating logical components. Thus, the viewpoint considers the system in a glass box view in contrast to the functional viewpoint where the system is considered as a black box. The relation between a user function (from the functional viewpoint) and a logical component (from the logical viewpoint) is often mixed up. A user function formalizes a part of the requirements that a system must fulfill at its boundary, whereas a logical component captures a part of functionality that lies within a system and that contributes (among other things) to the realization of a user function. In general, there is an n:m mapping between user functions and logical components. This means that one user function can be realized by a number of logical components, and one logical component can contribute to the realization of a number of user functions.

Fig. 3-2 illustrates the relation between these two notions. As shown, the system under development S is structured into a hierarchy of user functions (functional black box model). For each user function, there is a functional white box model. The figure only shows the functional white box model for user function UF_1. The functions of the white box model are allocated to logical components in the logical viewpoint. The logical component LC_1 itself can in turn also be considered a system under development in the next lower abstraction layer. We can again provide a functional black box model (with function F_1 as one of the user functions), functional white box models, and a logical component architecture for the SUD LC_1.

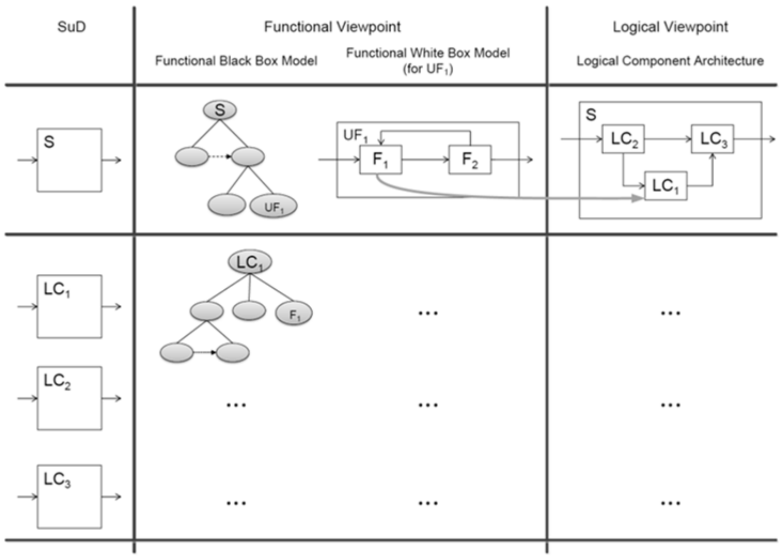
SuD
Functional Viewpoint
Functional Black Box Model
Functional White Box Model (for UF₁)
Logical Viewpoint
Logical Component Architecture
S
UF₁
F₁
F₂
LC₁
LC₂
LC₃
...

Fig. 3-2 *Relation between the functional and logical viewpoints across abstraction layers*

Relating the logical and technical viewpoints

At the core of the relation between the logical and the technical viewpoints is the deployment mapping. It specifies where and how logical components are realized on the technical architecture: which technical parts (ECUs, busses etc.) are involved in implementing such logical components, which communication resources are used in their interaction, and so on. Once the deployment is specified, a check is required to determine whether properties established for the logical viewpoint remain valid in the technical viewpoint. A typical issue that arises is schedulability, for instance, when several software tasks have been allocated to one ECU. The availability of features of the target has a considerable impact on the form of the deployment mapping and the type of analyses to be performed. Platforms such as AUTOSAR provide an abstraction layer that alleviates several of these tasks. Another issue in this design step concerns requirements inherited from previous steps that now have to be implemented. For example, redundancy, reliability, and independence usually have to be taken care of when moving to the technical viewpoint.

3.6 Underlying Modeling Theories

The SPES modeling framework is founded on several formal modeling theories and uses these theories' basic concepts in a continuous modeling approach. However, the SPES modeling framework was not based on one single specific theory and may be formalized using any of the underlying modeling theories. The aim of this was to allow practitioners to tailor the SPES modeling approach for different development projects and to allow for a maximum of flexibility in formalization. The underlying modeling theories are briefly summarized below.

The SPES modeling framework supports the use of formalized notions but does not force it. In the requirements viewpoint, for example, to document scenarios, informal sequence charts can be used in addition to formal message sequence charts [ITU 2004]. While informal sequence charts are mainly used for describing and discussing the system's functions and behavior with stakeholders, formal message sequence charts may be used to specify strict and formal solution-oriented requirements consistent with different components.

Similarly, in order to integrate different architecture viewpoints (functional, logical and technical), formal semantics can be used. Therefore, at least two different formal modeling theories have been developed that fit the SPES modeling approach. One modeling theory describes modeling entities such as functions, subsystems, or technical components using stream processing functions. In this theory, a stream processing function describes the behavior of an entity at its interface in terms of input and output data streams (see [Broy 2010]). Another modeling theory describes modeling entities as heterogeneous rich components [Damm et al. 2005], where the term "rich" alludes to the key ingredient of heterogeneous rich components to provide (rigorous) interface specifications for multiple aspects, encompassing both functional and extrafunctional (e.g., safety and real-time) characteristics of components. Specifically, the proposal is to use contract-based specifications that allow, for each aspect, characterization of the allowed design context of a component.

Besides the different formal modeling theories that can be chosen electively, there is a common understanding that forms the foundation of the complete SPES modeling framework with all of its views. The SPES modeling framework distinguishes between the system under development and the system's context. Whereas modeling the context is a key element of the requirements viewpoint, the different architectural views focus on modeling the interfaces connecting the system with its context. The system may be decomposed into subsystems or components with their own contexts. The context of a subsystem, for example, will contain the context of the overall system as well as the other subsystems of the system.

3.7 Overview of the Following Chapters

All four viewpoints of the SPES modeling framework are explained in more detail in the following chapters. Chapter 4 outlines the requirements viewpoint and illustrates how requirements from the context can be documented and refined by means of requirements models. The functional viewpoint is explained in Chapter 5 and shows how the requirements models from the requirements viewpoint can be refined into a functional architecture of the system that fulfills the solution-oriented requirements. Chapter 6 describes the logical viewpoint that allows specification of a logical architecture that consists of internal and external interfaces of the system in compliance with the system requirements as well as the functional architecture. Finally, Chapter 7 illustrates the technical viewpoint. This viewpoint maps a logical architecture onto concrete physical and technical components such that the system requirements are fulfilled.

Safety and real-time are two very important concerns of embedded system development. These concerns are crosscutting and have to be considered in every viewpoint on every abstraction layer that is used during development. Chapters 8 and 9 show how safety and real-time respectively are accounted for during the development within all viewpoints.

All four viewpoint chapters and both crosscutting concerns illustrate their respective concepts by means of a common example system in the form of a case study. The case study is a cylinder head production system taken from the automation domain. The cylinder head production system is an assembly line that produces cylinder heads for gasoline engines. The system takes raw material that is fed into the production cell. It consists of an input and an output conveyor belt, a milling station, a grinding station, a measuring station, as well as an assembly station that finally delivers the finished product. Production of cylinder heads must obey strict real-time constraints, as material may only be processed under the right physical conditions. Furthermore, system safety is an important concern that must be observed.

3.8 References

[Baumgart et al. 2010] A. Baumgart, P. Reinkemeier, A. Rettberg, I. Stierand, E. Thaden, R. Weber: A model-based design methodology with contracts to enhance the development process of safety-critical systems. In: Proceedings of the 8th IFIP WG 10.2 international conference on Software technologies for embedded and ubiquitous systems, SEUS'10, 2010, pp. 59-70.

[Baumgart et al. 2011] A. Baumgart, E. Böde, M. Büker, W. Damm, G. Ehmen, T. Gezgin, S. Henkler, H. Hungar, B. Josko, M. Oertel, T. Peikenkamp, P. Reinkemeier, I. Stierand, R. Weber: Architecture modeling. In: OFFIS Technical Report, OFFIS Oldenburg, March 2011.

[Braun et al. 2010] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, T. Weyer: Guiding requirements engineering for software-intensive embedded systems in the automotive industry. Computer Science - Research and Development. DOI: 10.1007/s00450-010-0136-y, 2010.

[Broy 2010] M. Broy: Multifunctional Software Systems: Structured modelling and specification of functional requirements. In: Science of Computer Programming, Vol. 75, No. 12, 2010, pp. 1193-1214.

[Broy et al. 2007] M. Broy, I. Krüger, M. Meisinger: A formal model of services. In: ACM Transactions on Software Engineering and Methodology (TOSEM), Vol. 16, No. 1, ACM, New York, 2007.

[Damm et al. 2005] W. Damm, A. Votintseva, A. Metzner, B. Josko, T. Peikenkamp, E. Böde: Boosting re-use of embedded automotive applications through rich components. In: Foundations of Interface Technologies, FIT’05, 2005.

[Davis 1993] A. M. Davis: Software Requirements – Objects, Functions, States. 2nd Edition, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[Dijkstra 1976] E. Dijkstra: A discipline of programming. Prentice-Hall Series in Automatic Computation, 1976.

[IEEE 1362] Institute of Electric and Electronic Engineers: Guide for information technology – system definition – concepts of operations (IEEE 1362-1998), IEEE Press, 1998.

[IEEE 1471] Institute of Electric and Electronic Engineers: Architectural Description of Software-Intensive Systems (IEEE 1471-2000), IEEE Press, 2000.

[ITU 2004] International Telecommunication Union: ITU-T Z.120: Message Sequence Chart (MSC), 2004.

[Lamsweerde 2009] A. van Lamsweerde: Requirements Engineering – From System Goals to UML Models to Software Specifications. Wiley, West Sussex, 2009.

[Levenson 2000] N. Levenson: Intent Specifications – An approach to building human-centered specifications. IEEE Transactions on Software Engineering, Vol. 26, No. 1, 2000, pp. 15-35.

[Lim 2002] W. Lim: Effects of reuse on quality, productivity, and economics. IEEE Software 11(5), 2002.

[Parnas and Madey 1995] D. L. Parnas, J. Madey: Functional documents for computer systems. In: Science of Computer Programming, Vol. 25, No. 1, pp. 41-61.

[Pohl 2010] K. Pohl: Requirements Engineering – Fundamentals, Principles, Techniques. Springer, Berlin/Heidelberg, 2010.

[Potts 1995] C. Potts: Using schematic scenarios to understand user needs. In: Proceedings of the ACM Symposium on Designing Interactive Systems – Processes, Practices, Methods and Techniques (DIS'95). ACM, New York, 1995, pp. 247-266.

[Sage and Rouse 2009] A. P. Sage, W. B. Rouse: Handbook of Systems Engineering and Management. John Wiley and Sons, 2nd Edition, 2009.

[Schätz 2005] B. Schätz. Building components from functions. In: Electronic Notes in Theoretical Computer Science, Vol. 160. Proceedings of the International Workshop on Formal Aspects of Component Software FACS 2005.

[Sikora et al. 2010] E. Sikora, M. Daun, K. Pohl: Supporting the consistent specification of scenarios across multiple sbstraction levels. In: R. Wieringa, A. Persson (Hrsg.): Proceesdings of the 16th Intl. Working Conf. on Requirements Engineering: Foundation for Software Quality. LNCS 6182, Springer, Berlin/Heidelberg, 2010, pp. 45-59.

[Sikora et al. 2012] E. Sikora, B. Tenbergen, K. Pohl. Industry needs and research directions in requirements engineering for embedded systems. In: Requirements Engineering Journal, Vol. 17, No.1, 2012, pp. 57-78.

[Sommerville 2010] I. Sommerville: Software Engineering. Pearson, 9th Edition, 2010.

[Tarr et al. 1999] P. Tarr, H. Ossher, W. Harrison, Jr. S. M. Sutton: N degrees of separation: multi-dimensional separation of concerns. In: ICSE '99: Proceedings of the 21st international conference on Software engineering, ACM, New York, 1999, pp. 107-119.

[Weyer 2011] T. Weyer: Kohärenzprüfung von Anforderungsspezifikationen: Ein Ansatz zur Prüfung der Kohärenz von Verhaltensspezifikationen gegen Eigenschaften des operationellen Kontexts. Südwestdeutscher Verlag für Hochschulschriften, 2011.

[Yu 1997] E. Yu: Towards modelling and reasoning support for early-phase requirements engineering. In: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97), IEEE Computer Society Press, Los Alamitos, 1997, pp. 226-235.