Andreas Vogelsang
Sebastian Eder
Dr. Martin Feilkas
Daniel Ratiu

5

Functional Viewpoint

The major concern of the functional viewpoint is to provide a formal and model-based behavior specification for the system under development. Therefore, the viewpoint provides two model types that structure the behavioral requirements according to user functions and provide an abstract realization of these. A user function captures a set of solution-oriented requirements, as specified in the models of the requirements viewpoint, and integrates them into a functional black box model — a behavioral description of the entire system under development. By using formally founded models, the functional black box model provides the basis for detecting undesired interactions between user functions at an early stage of the development process. User functions are later refined by a functional white box model that decomposes a user function into functions that represent smaller units of functionality and provide an abstract realization of the user function. Due to the high level of abstraction, this viewpoint is a step towards closing the gap between semiformal requirements and a formal system design.

K. Pohl et al. (eds.), *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*,
DOI 10.1007/978-3-642-34614-9_5,

5.1 Introduction

The functional viewpoint has two model types: functional black box model and functional white box model

The starting point for the functional viewpoint is a set of requirements for the behavior of the SUD provided by the models of the requirements viewpoint, especially the context model, the scenario models, and the behavioral requirements models (see Chapter 4). These models provide a complete set of requirements in a semiformal, model-based representation. The functional viewpoint provides two model types: the functional black box model, which formalizes the requirement models as user functions and integrates them into a comprehensive system specification [Broy 2010], and the functional white box model, which provides a decomposition of the user functions from the functional black box model into smaller functional units in order to give an abstract description of the realization of the user functions.

The models that are provided by the requirements viewpoint describe requirements for the SUD from the view of a specific usage context. The functional viewpoint translates these partial usage models into the notion of user functions that define the intended system behavior, including all interactions and dependencies between them. Thus, the result of the functional viewpoint is a comprehensive system specification.

A user function hierarchy consists of user functions and dependencies between them.

Within the functional black box model, the requirements models are translated into user function hierarchies consisting of user functions and dependencies between them (see Fig. 5-1 for an informal representation). Each user function realizes a piece of black box functionality and is defined by its syntactic interface and its behavioral specification. The syntactic interface comprises the ports via which the user function is connected to its context, and the behavioral specification defines the messages exchanged on these ports.

User functions may have quite complex functional requirements that may not even give any information about a possible solution for this requirement. In order to reduce this complexity and also to facilitate reuse of existing partial solutions, the user functions are refined in a functional white box model. This model consists of a set of functions that give an abstract solution of the functionality that is required by the user function.

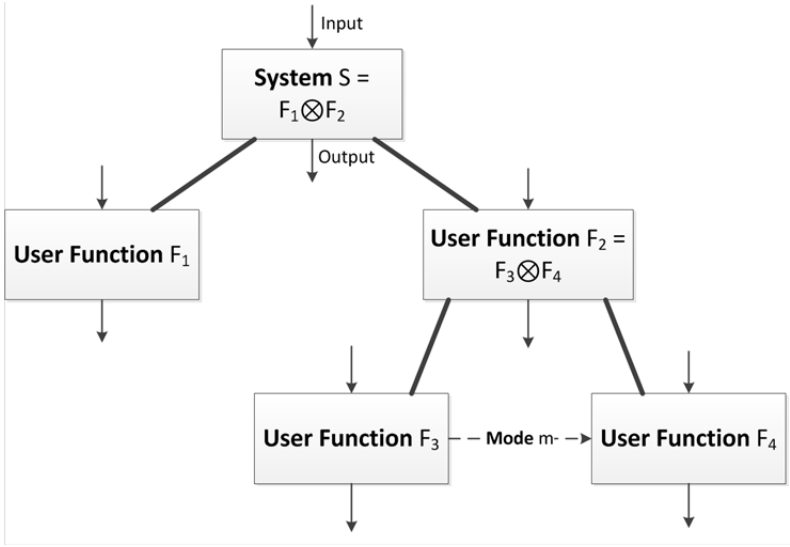


Fig. 5-1 *Informal representation of a user function hierarchy: user functions are composed into more complex user functions taking account of their dependencies (dashed horizontal arrows) and finally into the specification of the entire SUD.*

5.2 Concerns

Aims of the functional viewpoint

The central aims of the functional viewpoint are:

- Consolidating the functional requirements by formally specifying the requirements of the system behavior from the black box perspective
- Mastering feature interaction: detection and resolution of inconsistencies within the functional requirements
- Reducing complexity by structuring the functionality hierarchically from the user's point of view
- Understanding the functional interrelationships by collecting and analyzing the interactions between different (sub-) functionalities.

The functional viewpoint provides a hierarchically structured specification of the SUD behavior as it is perceived by the user at the system boundary (also known as usage behavior). In this context, a user may be a person but also another system. The functional viewpoint comprises the formal definition of the SUD interface with surrounding systems and users. The behavior of the entire SUD is then specified from the black box perspective by describing the exchange of messages between the SUD and its context. Here, the abstract data flow is specified, namely the intentional meaning of the exchanged data (as

opposed to the concrete message types). By formally describing the requirements, we create the basis for measuring the completeness of and detecting inconsistencies in the requirements, especially for interacting requirements of different functions (cf. feature interaction [Zave 1993]).

The functional viewpoint is independent from realization

The overall system functionality can be obtained from the composition of user functions (with respect to the dependencies between them). Here, the decomposition/structuring is not guided by architectural or technical aspects but is executed merely along the functional aspects required by the users.

Thus, an informal requirement can be realized by one or several user functions and a user function can realize one or more informal requirements. The structuring and refinement of the requirements models in the functional viewpoint makes it possible to analyze existing requirements and thus to detect and resolve inconsistencies (e.g., feature interaction) and missing requirements.

5.3 Functional Black Box Model

User functions

The central construct of the functional black box model is a user function that defines a part of the behavior of the SUD that can be observed at the system boundary. Fig. 5-2 shows how user functions are related to the black box interface of the SUD. Consequently, a user function does not contain any information about how it is implemented in the SUD.

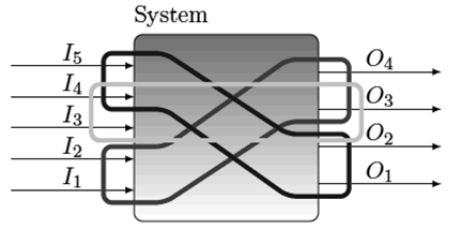


Fig. 5-2 *The system interface is structured into three user functions that all comprise a subset of input and output channels and their related behavior.*

Each user function has a syntactic interface that consists of a number of typed ports that are either input ports or output ports. In general, input and output ports define messages that an actor sends to the SUD or receives from the SUD. We also assign an interface behavior to each user function. A possible formalization of this notion can be found in [Broy 2010].

Example 5-1: Transportation user function

Fig. 5-3 and Fig. 5-4 illustrate a simple transportation belt user function. The user function has two input channels transmitting two values that indicate whether a work piece is present at the beginning or the end of the transportation belt. The output channel controls the transportation belt motor. The user function formalizes the requirement that the transportation belt shall be switched on if a work piece is present at the beginning of the belt but no other work piece is waiting at the end of the belt. The behavior specification is given by a simple table specification that maps input values to output values. The "?" character is an abbreviation for all possible values on this channel.

Fig. 5-3 *Syntactic interface of the Transportation user function*

IndicatorPos1	IndicatorPos2	BeltMotor
WPPresent	NoWP	Activate
?	WPPresent	Deactivate

Fig. 5-4 *Interface behavior of the Transportation user function described by a simple I/O table that maps values of the input channels to values of the output channel*

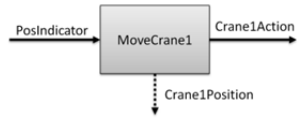
Mode channels

Fig. 5-2 also reveals that user functions can have common input or output channels. This indicates a certain kind of dependency between user functions. This dependency expresses that a user function also needs information about another function's state or input values to determine the correct output values. In order to model dependencies between user functions, we use special ports that connect user functions via a *Mode Channel*. These channels are called mode channels as they usually transmit values that represent an abstract state of the SUD — a mode.

In principle, we could avoid the appearance of functional dependencies completely by explicitly all inputs ever relevant for a user function and its output behavior to syntactic interfaces. However, due to the high number of dependencies in a system, this leads to a completely confusing and unmanageable behavior. Therefore, it is better to capture the dependencies using modes.

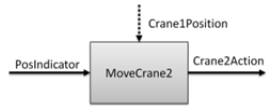
Example 5-2: Dependent crane user functions

An automation system has two cranes that transport work pieces through an assembly line with six stations: **S**upply **B**elt, **Mi**lling, **Gr**inding, **Me**asuring, **As**sembling, and **D**elivery **B**elt. Two user functions steer the cranes. However, to avoid collisions, Crane 2 is not allowed to approach station "Measuring" when Crane 1 is approaching that station. Thus, the user function "MoveCrane2" depends on the state of the user function "MoveCrane1." We model this dependency by introducing a mode channel "Crane1Position" and extend the behavior specification of the user function "MoveCrane2" to prevent the user function approaching the "Measuring" station if Crane 1 is approaching it. The resulting user functions and their behavior specifications are given by Fig. 5-5 and Fig. 5-6.



PosIndicator	Crane1Action	Crane1Position
WP@SB ∧ ¬WP@Mi ∧ ¬WP@Gr	SB2Mi	Mi
WP@Mi ∧ ¬WP@Gr	Mi2Gr	Gr
WP@Gr ∧ ¬WP@Me	Gr2Me	Me

Fig. 5-5 *Syntactic interface and behavior specification of the user function "MoveCrane1." The current position of the crane is propagated by the mode channel "Crane1Position."*



Crane1Position	PosIndicator	Crane2Action
¬Me	WP@Me ∧ ¬WP@As	Me2As
?	WP@As ∧ ¬WP@DB	As2DB

Fig. 5-6 *Syntactic interface and behavior specification of the user function "MoveCrane2." Whether or not the crane can approach the "Measuring" station depends on the "Crane1Position" mode.*

User function hierarchy

We distinguish two kinds of user functions: *atomic* user functions and *composite* user functions. Composite user functions are composed of at least two subuser functions. User functions that are not decomposed are called atomic user functions. Atomic user functions must provide a behavior specification that defines their behavior observed at the system boundary. In a nutshell, atomic and composite user functions allow the developers to decompose the functional requirements, formalized as user functions, into a hierarchy of user functions.

Example 5-3: User function hierarchy of the Automation system

We can integrate the given user functions from Example 5-1 and Example 5-2 into a comprehensive system that provides all the user functions. The two user functions "MoveCrane1" and "MoveCrane2" are composed to a composite user function "MoveCranes." Fig. 5-7 visualizes this as a user function hierarchy. The syntactic interface and the interface behavior of the composite system are derived from the composition of the user functions. Fig. 5-8 illustrates the resulting syntactic interface of the automation system.

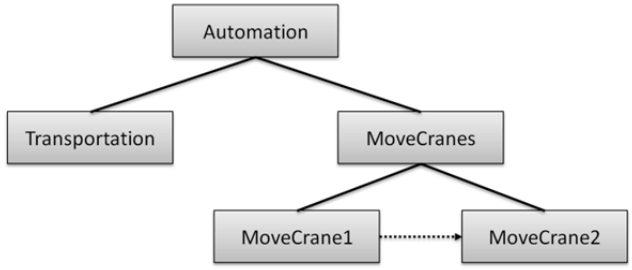


Fig. 5-7 *User function hierarchy of the composite Automation system. The dashed arrow represents the mode channel between the user functions "MoveCrane1" and "MoveCrane2." The solid arrows indicate a subfunction relationship.*

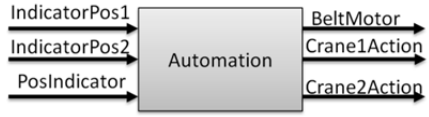


Fig. 5-8 *Syntactic interface of the composite Automation system, derived by the composition of the interfaces of the subfunctions. The mode channel between the user functions "MoveCrane1" and "MoveCrane2" becomes invisible when composing the user functions.*

All of the presented are taken from a comprehensive case study that has been modeled in SPES 2020 [Eder et al. 2011].

5.4 Functional White Box Model

The second model type within the functional viewpoint is the functional white box model. The purpose of this model is to provide a decomposition of the user functions from the functional black box model into smaller functional units in order to give an abstract description of the realization of the user functions. These smaller functional units are called functions.

Function

A function itself can be described by a syntactic interface with an associated interface behavior. This is an example of how we take advantage of a common underlying modeling theory in SPES that provides modeling concepts that are reused in each of the viewpoint models. The purpose of this decomposition is to master the complexity that arises from the black box specification, or, more specifically, the question of how the specified outputs will be computed from the given inputs.

Example 5-4: Functional white box model for a user function

In order to realize the behavior of the user function "MoveCrane2" as specified in Fig. 5-6, we have to decompose the user function into smaller functional units called functions. Fig. 5-9 shows the functional white box model for this decomposition. The user function is broken down into five functions. In contrast to the black box specification of the user function, the white box model already provides some information about the realization, e.g., the sensor values are read in separately and the control logic is encapsulated in a single function.

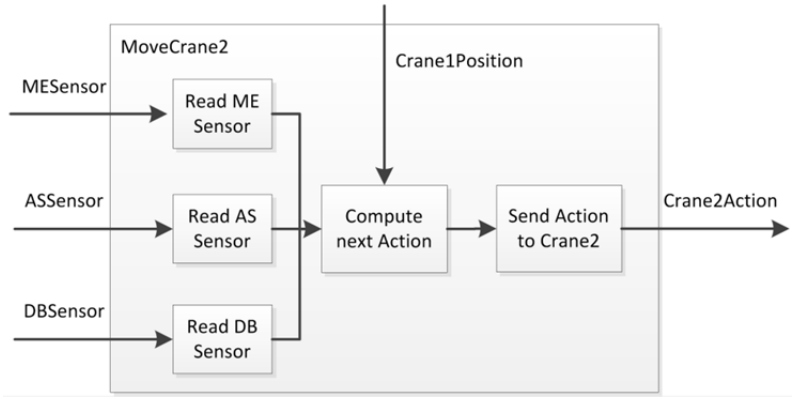


Fig. 5-9 *Functional white box model for the user function "MoveCrane2"*

The relation between the functional black box model and the functional white box model is as follows: the functions of the functional white box model together must show the same behavior as specified by the user function of the functional black box model. Therefore, it is necessary to provide a mapping between the inputs and outputs of the user function and the inputs and outputs of the functional white box model. This mapping can then be used to check whether the functional white box model conforms to the functional black box model.

Example 5-5: Mapping between black box and white box models

The user function "MoveCrane2" from Fig. 5-6 has only one input for indicating the position of work pieces and this is called "PosIndicator." The white box model as depicted in Fig. 5-9, however, has three inputs for indicating the position of work pieces: MESensor, ASSensor, and DBSensor. Therefore, in order to check whether the functional white box model specifies the same behavior as the functional black box model, mapping that maps the values of the inputs to each other is required. Fig. 5-10 gives this mapping in a tabular representation.

PosIndicator	MESensor	ASSensor	DBSensor
WP@Me ∧ ¬WP@As	WP	¬WP	?
WP@As ∧ ¬WP@DB	?	WP	¬WP

Fig. 5-10 *Tabular representation of the mapping between the inputs of user function "MoveCrane2" and its corresponding white box model*

Another important aspect of this white box model is reuse. The functionality that is captured by a function can be reused in several functional white box models. In our example, the "Read AS Sensor" function could be reused to decompose the user function "MoveCrane1." In such scenarios it is particularly important to check the conformity of the composed white box model to the original specification of the user function because reuse bears the risk of incorrectly reusing certain functions.

5.5 Analyses

The functional viewpoint offers a model of the functionality of an SUD at a very early stage of development. Several analyses and evaluations can be performed on this model to verify and validate the functional requirements of the SUD. Validation of requirements in this sense means that the requirements are reasonable with respect to each other, i.e., no contradictions between requirements, no unintended behavior due to unintended interactions between requirements, or no missing requirements. In contrast, verification ensures that the behavior as described in the model of the logical viewpoint fulfills the behavior that is specified in the model of the functional viewpoint.

In the following we outline some of the analyses that can be used for validation or verification:

Simulative validation

If the behavior is specified by executable models (e.g., state machines), the models of the functional viewpoint provide a functional prototype of the SUD. This prototype can be used to run through certain actions and scenarios to validate the behavior of the SUD in cooperation with the user. Thus, unintended behavior can be revealed and evaluated.

Detection of inconsistencies

The functional black box model of the functional viewpoint also allows the detection of inconsistencies. Inconsistencies are input patterns that cause different functions to output conflicting values. This can be checked prior to execution and thus reveal contradicting requirements [Harhurin 2010].

Completeness analyses

At the beginning of development, requirements only define part of the entire functionality. During development, more and more requirements complete the system specification. The models of the functional viewpoint facilitate this process by showing input patterns for which no output is defined. These situations represent insufficient specification that should be discussed and possibly refined.

Test cases

The functional viewpoint models and formalizes the functional requirements and thus serves as a specification. In this role, the model of the functional viewpoint can be used as a testing oracle. Test cases can be generated from it to verify that a model of the logical viewpoint fulfills the specified behavior of the functional viewpoint.

Tracing between user functions and logical components

The model of the logical viewpoint describes the inner structure of the system that implements the user functions. Input and output ports of user functions can be linked to logical components that implement these user functions. This yields a tracing relation between user functions and logical components, allowing design faults to be detected and functions to be tested individually, and ensuring the implementation of all user functions [Vogelsang et al. 2012].

5.6 Integration in the SPES Modeling Framework

Within the SPES modeling framework, the functional viewpoint is located between the requirements and the logical viewpoints. This section gives an overview of the relation and the differences between the functional viewpoint and its neighbors in the SPES modeling framework.

Relation to the requirements viewpoint

The goal of the requirements viewpoint is to capture the requirements for the SUD and to document their relation to the context of the SUD. The result is a set of models that represent a variety of requirements for the SUD. These models are derived from abstract goals and information about the system's context. A crucial task is now to integrate all of these

requirements models into a system specification that is consistent and sound. This task is done in the functional viewpoint (at least for the requirements that refer to system behavior). The requirements models of the requirements viewpoint are captured and refined as user functions that the SUD offers to its context. The complex interactions that arise from the interplay of the requirements (feature interaction) are explicitly modeled in the functional black box model of the functional viewpoint. The result is an integration of the isolated requirements models from the requirements viewpoint into a comprehensive system model that describes the required system behavior at its boundary to its context and serves as a specification. Technically, this relation is realized by reusing and refining the models of the requirements viewpoint. The context model that is defined in the requirements viewpoint defines the syntactic interface of the SUD, i.e., the inputs from the environment as well the outputs of the SUD to the environment. This interface is authoritative for the definition of user functions. The syntactic interface of a user function must always be a subset of the syntactic interface of the SUD as defined in the context model. Moreover, a behavioral requirements model of the requirements viewpoint can initially be reused as behavior of a user function. When integrating all user functions, the behavioral models must be enriched with additional behavior that arises from the concurrent integration in the SUD. The scenario models of the requirements viewpoint serve as test cases and validation conditions for the functional black box model of the functional viewpoint.

Relation to the logical viewpoint

The model of the logical viewpoint describes the internal logical structure of the SUD by means of communicating logical components. Thus, the viewpoint considers the SUD in a glass box view in contrast to the functional viewpoint where the SUD is considered as a black box. The relation between a user function (from the functional black box model) and a logical component (from the logical viewpoint) is often confused. A user function formalizes a part of the requirements that the SUD has to fulfill at its boundary, whereas a logical component captures a part of functionality that lies within the SUD and that contributes (amongst 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. As this gap is sometimes confusing and hard to manage, the functional viewpoint provides the functional white box model. In this model the user functions from the functional black box model are decomposed into functions that give an abstract solution for the realization of the user function. The functions are then mapped to

a logical component of the logical viewpoint. In this way, a logical component contains a set of functions that contribute to the realization of one or several user functions.

Functional viewpoint and the logical Viewpoint across abstraction layers

Depending on what is considered the system under development on a certain abstraction layer, the functional black box model of the functional viewpoint provides a functional black box specification of the respective SUD. This model structures the interface specification of the system according to user functions of the system. As a consequence, we get new functional black box models for the functional viewpoint if we change the abstraction layer. In the uppermost abstraction layer, for example, the system is considered as a whole with one functional black box model that specifies the black box behavior of the system at its boundary. If we step into the next lower abstraction layer by decomposing the system into a number of logical components, each logical component can again be considered as a system (with a different context) and thus has an own functional black box model in the functional viewpoint. These models again specify the intended functional black box behavior for each logical component. An important perception here is to recognize that the user functions of the system in one abstraction layer do not have to be the same as the user functions of a logical component in the next lower abstraction layer. An example for this was already given in Section 3.5.2. In fact, the user functions of a logical component are heavily influenced by the functions of the functional white box model. When mapping the functions of the functional white box model to logical components, we determine user functions that the logical component must fulfill and that are part of the functional black box model of the logical component.

5.7 The Functional Viewpoint Process

In the following, we will give an idealized process through the functional viewpoint. Note that in reality, this process is highly iterative and also interweaved with the processes of the other viewpoints. It is also important to note that the requirements viewpoint process does not have to be completed before the functional viewpoint process is started, nor does the functional viewpoint process have to be completed before the process of the logical or technical viewpoint is started. Furthermore, we can divide the functional viewpoint process into building the functional black box model and building the functional white box model. We will start with the black box model that is subsequently refined in the white box model.

Step 1: Extract user functions

In the first step, we use the solution-oriented requirement models from the requirements viewpoint to extract user functions for the SUD. Initially we can translate each behavioral model of the requirements viewpoint into one user function. Later we might find it useful to merge a set of behavioral models into just one user function. We make sure that the inputs and the outputs of the user functions conform to the syntactic interface as defined by the context model of the requirements viewpoint, i.e., the inputs and outputs of a user function are a subset of the inputs and outputs of the context model. Defining user functions is not a canonical step. There are a variety of possibilities for structuring a system according to user functions. However, a guiding principle is to define the different functions as they are perceived by the user (similar to the notion of a use case).

Step 2: Structure user functions in a user function hierarchy

Once we have a set of user functions we try to structure them in a user function hierarchy. We may also introduce new user functions that group a set of user functions as their subfunctions. There are multiple ways to arrange the user functions into a user function hierarchy. The user function hierarchy should again reflect a structure of the user functions as they are perceived by the user. Another goal of choosing user functions and arranging them in a user function hierarchy is to gain a manageable set of user functions that, on the one hand are not too complex to give their behavior as a behavior specification, but on the other hand do not have too many dependencies to other user functions.

Step 3: Specify interfaces for all atomic user functions

The next step is to specify an interface for each atomic user function in the user function hierarchy. This is done by providing both a syntactic interface by means of input and output channels of the user function and a behavior specification. Behavior specifications can be given by any specification technique that an interface behavior abstraction can be assigned to. Examples for such specifications are state machines [Broy 2010], I/O tables [Thyssen and Hummel 2011], or data flow diagrams [Leuxner et al. 2010].

Step 4: Model dependencies using mode channels

The next step is to model dependencies and resolve inconsistencies between the user functions. Inconsistencies between user functions can have many facets. For example, two user functions that share a common output channel are in a conflicting situation as they both write values to that output channel at the same time [Harhurin 2010]. Dependencies on the other hand can be intended or unintended. Intended dependencies represent desired interaction between user functions such as one user function interrupts another or works differently depending on results of another user function. Unintended dependencies arise from unconscious interplay between user functions. Inconsistencies and dependencies are modeled in the functional viewpoint by means of mode channels. Thus,

we extend the user functions' interface with additional mode channels that transmit information necessary for resolving inconsistencies and modeling dependencies.

Step 5: Extend the behavior specifications

The last step of the functional black box model is to extend the behavior specifications of the user functions with respect to the mode channels introduced. This means that we have to integrate the information that is provided by the mode channels into the behavior of the user function. For two conflicting user functions, for example, a mode channel between them resolves this conflict by transmitting the information that one user function is currently not allowed to send a value over the common output channel.

Step 6: Building the functional white box model

From here, we start building the functional white box model. We do not have to wait for the black box model to be fully specified before commencing with the white box model. We could also start building parts of the white box model immediately after Step 1. We build a functional white box model for each atomic user function, and this model provides a high-level description of tasks that need to be performed in order to realize the user function and the data flow between these tasks. We capture such tasks as functions with a syntactic interface that defines the data that is processed and produced by the function and we additionally associate an interface behavior that specifies the behavior of the function.

Step7: Check conformance between white box and black box models

Finally, we need to ensure that the functional white box model conforms to the functional black box model and that this method is a valid realization for the user function. Therefore, we have to check that the composition of the functions in the functional white box model yields the same behavior that the user function from the functional black box model demands. Several methods, all with advantages and disadvantages, can be used for this purpose, for example, testing, model checking, or formal verification.

5.8 References

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

[Eder et al. 2011] S. Eder, A. Vogelsang, M. Feilkas: Seamless modeling of an automation example using the SPES methodology. In: Technical Report TUM-I1110. Technische Universität München, May 2011.

[Harhurin 2010] A. Harhurin: From Interaction Patterns to Consistent Specifications of Reactive Systems. PhD Thesis, Technische Universität München, 2010.

[Leuxner et al. 2010] C. Leuxner, W. Sitou, B. Spanfelner: A formal model for work flows. In: SEFM 2010: Proceedings of the 8th International Conference on Software Engineering and Formal Methods, 13-18 Sept. 2010, pp. 135-144.

[Thyssen and Hummel 2011] J. Thyssen, B. Hummel: Behavioral specification of reactive systems using stream-based i/o tables. Software and Systems Modeling, DOI: 10.1007/s10270-011-0204-1.

[Vogelsang et al. 2012] A.Vogelsang, S.Teuchert, J.-F. Girard. Extend and characteristics of dependencies between vehicle functions in automotive software systems. Proceedings of the 2012 International Workshop on Models in Software Engineering, 2012

[Zave 1993] P. Zave: Feature interactions and formal specifications in telecommunications. In: IEEE Computer, Vol. 26, No. 8, 1993, pp. 20-28.