Document Title	Interaction with Behavioral Models
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	205
Document Classification	Auxiliary

Document Status	Final
Part of AUTOSAR Release	4.2.2

Document Change History

Release	Changed by	Change Description
4.2.2	AUTOSAR Release Management	• Long name of document changed
4.2.1	AUTOSAR Release Management	• Editorial changes
4.1.1	AUTOSAR Administration	• Finalized for Release 4.1
3.1.4	AUTOSAR Administration	• Legal disclaimer revised
3.1.1	AUTOSAR Administration	• Legal disclaimer revised
3.0.2	AUTOSAR Administration	• Add figures
3.0.1	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
2.1.15	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
2.1.1	AUTOSAR Administration	• Legal disclaimer revised
2.0	AUTOSAR Administration	• Initial release

Disclaimer

Advice for users

AUTOSAR specifications may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction 5
1.1 Aspects of AUTOSAR Authoring Tools (non-normative) 5
1.2 Origins and Goals (non-normative) 8
1.3 Terminology 8
2 Requirements Tracing 10
3 Use-Cases for Behavior Modeling within AUTOSAR 11
3.1 Behavior Modeling of Software Components 11
3.2 Software Component Description to Behavior Model 11
3.2.1 Atomic Software Component to Behavior Model 12
3.2.2 Several atomic Software Components to Behavior Model 12
3.3 Behavior model to Software Component Description 13
3.3.1 AUTOSAR conform Behavior Model to Software Component 13
3.3.2 Legacy Behavior Model to Software Component 13
3.4 Combined Use-cases 14
3.4.1 Behavior Authoring within BMTs 14
4 Parts of the AUTOSAR Meta Model Relevant for Behavior Modeling 15
4.1 Introduction 15
4.1.1 Separation of Feature Categories of the Software Component Template 15
- C 16
4.1.2 Modeling Distributed Systems 17
4.2 Software Components 17
4.2.1 Atomic Software Component and Ports 17
4.2.2 Interface 18
4.2.3 Sensors and Actuators 20
4.2.4 Composition 21
4.2.5 Datatypes 22
4.2.6 Constants 23
4.2.7 Sender Receiver Annotation 24
4.2.8 Services 26
4.2.9 Runnable Entities 27
4.2.10 Inter Runnable Communication 28
4.3 Software Component Environment 29
4.3.1 ComSpec 29
4.3.2 RTEEvents 33
4.3.3 Execution Constraints 34
4.3.4 Services 35
5 Requirements for the Interaction of SW-C descriptions with Behavior Models 36
5.1 Conversion SW-C and Behavior Models 36
5.1.1 Tasks of the Converter 36
5.1.2 Directions of the Conversion 37
5.1.3 Round-trip Issue –Type concept 37
5.1.4 Generation of the Software Component Environment by the Converter 39
5.2 Creation of functional behavior models 39
5.2.1 [TR_ATBM_00001] Creation of functional behavior model frames 40

5.2.2 [TR_ATBM_00005] Update of functional behavior model frames 41
5.3 Conversion of Software Components 41
5.3.1 [TR_ATBM_00003] Creation of behavior model frames of an atomic software component 42
5.3.2 [TR_ATBM_00030] Creation of behavior model frames of a selection of atomic software components 43
5.3.3 [TR_ATBM_00004] Creation of a hierarchical model frame structure ..44
5.4 Creation of software component descriptions 45
5.4.1 [TR_ATBM_00002] Creation of interface description of software components 45
5.4.2 [TR_ATBM_00032] Update of interface description of software components 45
5.4.3 Legacy Models 46
5.4.4 AUTOSAR conform behavior models 48
5.4.5 [TR_ATBM_00031] Creation of runnable description from an AUTOSAR conform behavior model 49
5.5 Creation of the Software Component Environment 50
5.5.1 [TR_ATBM_00025] Creation of the Software Component Environment Model 50
6 Appendix 51
6.1 References 51
6.1.1 Normative References 51
6.1.2 Normative References to External Documents 52

1 Introduction

In the domain of automotive electronics, the development of control functions is increasingly determined by design methods based on mathematical models – usually called "model based design". Mathematical models are formal descriptions and are thus associated with various advantages. They improve the expressiveness of function designs, the degree of automation of single development steps, the quality of development results and much more. Tools supporting this approach have become commonplace in automotive industry and should therefore be considered in the scope of AUTOSAR.
This document provides general use cases and requirements for mapping AUTOSAR modeling elements to functional behavior models and vice versa. The specifications are independent of particular functional behavior modeling tools. Tool specific instantiations are addressed in further AUTOSAR deliverables.

1.1 Aspects of AUTOSAR Authoring Tools (non-normative)

The AUTOSAR methodology document [2] describes the major steps of a development of system with AUTOSAR: from the system level to the generation of an ECU executable. It describes the dependencies of work-products and activities. Each authoring tool can support one or more activities.
The term *AUTOSAR authoring tool* refers to all tools that support the activities of interpretation, modification and creation of AUTOSAR models which describe a system as defined in e.g. the

- Software-Component Template [3],
- ECU Resource Template [4],
- and System Template [5].

In particular, AUTOSAR authoring tools are required to be able to interpret, create or modify AUTOSAR XML descriptions (i.e. the XML representation of AUTOSAR models, see [14]).
Figure 1 sketches the descriptions that can be maintained by AUTOSAR authoring tools within the AUTOSAR methodology (for a detailed description of the notation see [2]). According to the AUTOSAR methodology, the System Configuration Input consists of models describing software-components, ECU hardware and some system constraints.
The formal description of AUTOSAR software-components does not include a complete formal description of the behavior of the software-component. The latter is intentionally left to dedicate Behavior Modeling Tools (BMT). It is therefore necessary to bridge the gap between a software-component model and the corresponding behavior model created by a particular BMT. This task is carried out by the "Coupling Tool" mentioned in Figure 1.

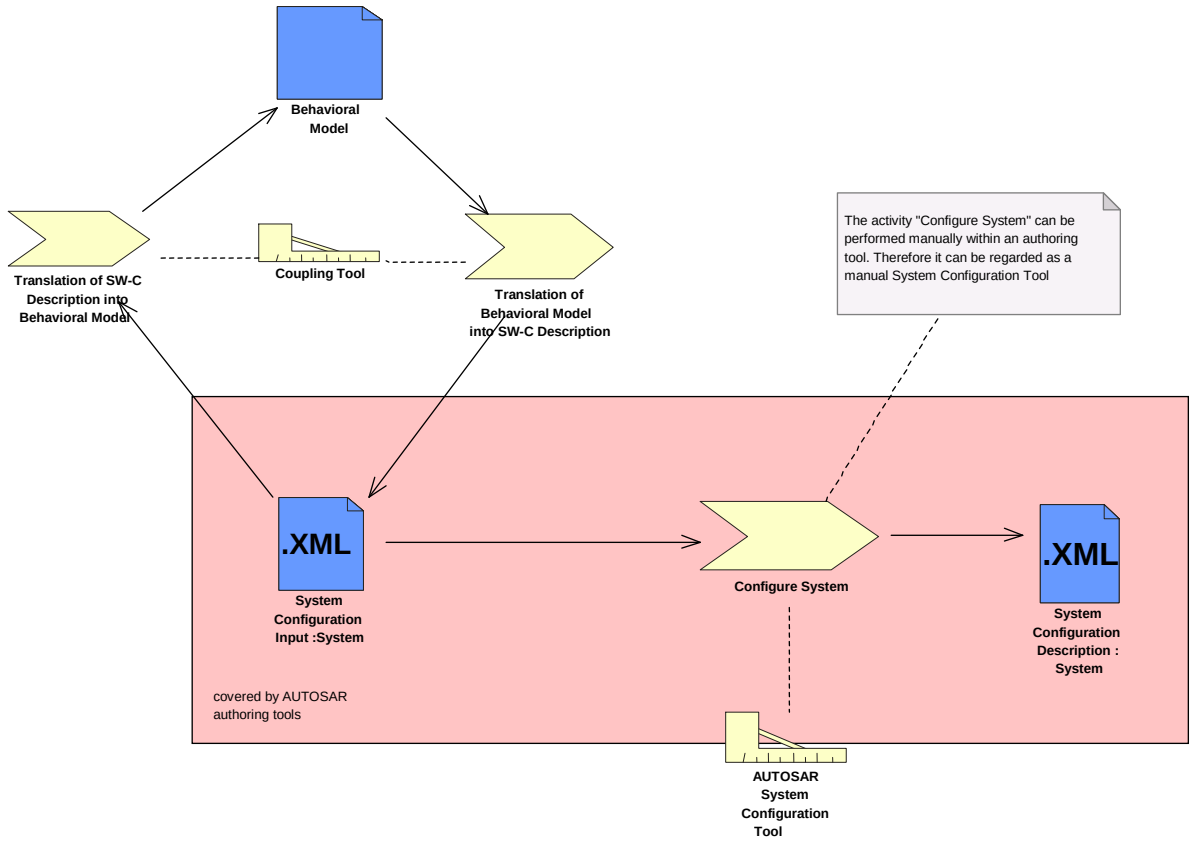

Figure 1: Descriptions which can be created and modified by AUTOSAR authoring tools

A further aspect of AUTOSAR authoring tools is the configuration of the System that (as sketched in Figure 1) produces the System Configuration Description as an output. Please note that this task can be carried out manually or (to some extent) automatically.

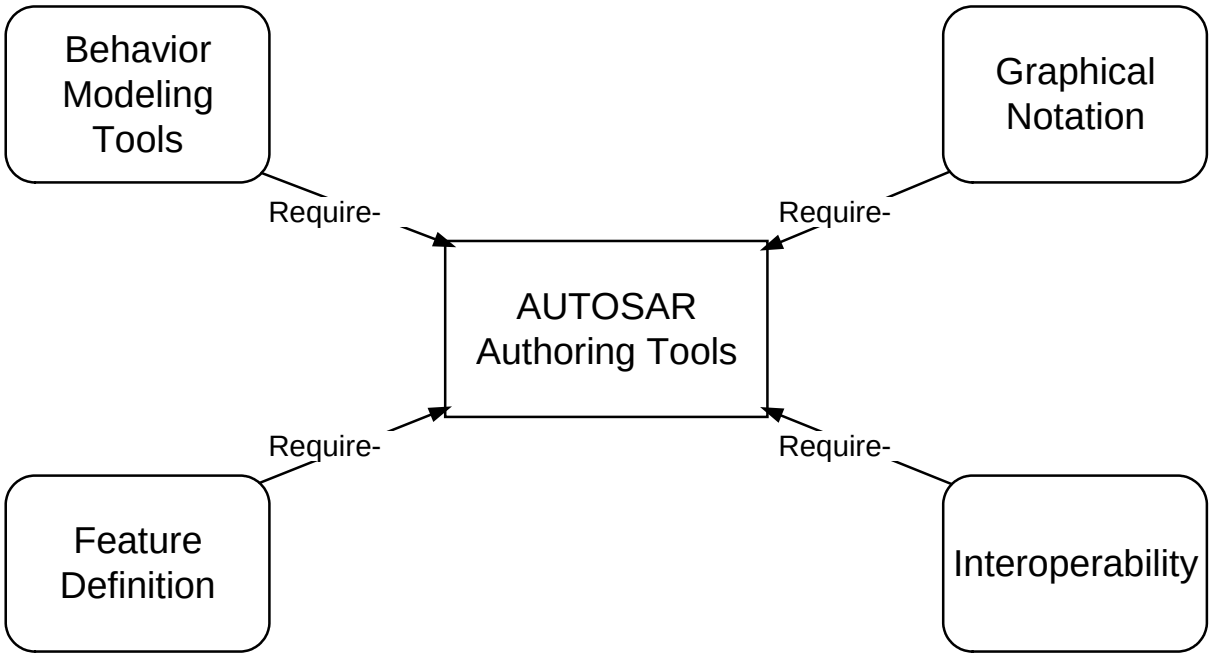

Figure 2: Aspects of AUTOSAR authoring tools

The description of AUTOSAR authoring tools covers several important aspects as depicted in Figure 2. Please note that the description of all these aspects results in the formulation of requirements on AUTOSAR authoring tools.
Each aspect as depicted in Figure 2 is described in a separate AUTOSAR document. In other words: beyond the scope of this document at hand, a separate discussion of specific aspects of AUTOSAR authoring tools (as depicted in Figure 2) is available (in a separate document for each aspect):

- **Specification of Feature Definition of Authoring Tools [11]**
  The document gives a recommendation for a stepwise implementation of the overall AUTOSAR concept with respect to the interchange descriptions, namely the Software-Component Template, the ECU Resource Template and the System Template. As the basis for a first implementation, a subset (corresponding to the definition of features) of the AUTOSAR templates mentioned above for a first implementation of AUTOSAR authoring tools is defined.

- **Interoperability of Authoring Tools [12]**
  This document emphasizes on issues that might come up when exchanging AUTOSAR models between different tools. After describing some basic concepts of data exchange this document sketches strategies on how these issues can be resolved. Requirements on AUTOSAR authoring tools for ensuring interoperability are defined.

- **Interaction with Behavioral Models (this document)**
  The document "AUTOSAR Interaction with Behavioral Models" lists use-cases for behavior modeling within AUTOSAR. Parts of the AUTOSAR meta model which are relevant for behavior modeling are identified. Requirements for the interaction of software component descriptions with behavior models are derived.

- **Specification of Graphical Notation [13]**
  The "Graphical Notation" document defines the graphical AUTOSAR notation for AUTOSAR authoring tools. For example, the document provides a comprehensive schema for graphically modeling `CompositionTypes`. The graphical notation should be used as a guideline for implementing AUTOSAR authoring tools.

It is advised to read all of these documents in order to understand the overall concept of AUTOSAR authoring tools.
Please note that other tasks within the AUTOSAR concept, for example the creation of an ECU Configuration, are not in the scope of AUTOSAR authoring tools.

1.2 Origins and Goals (non-normative)

Automotive developers are used to the so-called model based design of control functions and plant models. There are sophisticated tools that support the development of functional behavior models using simulation, automatic code generation and validation methods. Thus, model based design complements the structural design approach of the AUTOSAR methodology.
The distinction between model based and structural design, however, is not always clear. These design approaches may overlap regarding the capabilities of the used models and their supporting tools. In order to have the deliverables as introduced in section 1.1 consistent and free of redundant information, this deliverable mainly addresses use cases and requirements for mapping of AUTOSAR elements to functional behavior models and vice versa. Requirements that are related to authoring tools have been collected in [11], including e.g. requirements that would be fulfilled by a functional behavior modeling tool used as an authoring tool to edit a functional behavior model according to the AUTOSAR meta model. This distinction has been considered for editorial issues. The descriptions of real uses cases, however, might be orthogonal to this document structure.
Reasoning about "interaction with behavior models" shows that this interaction is neither obvious nor self-explanatory. The terms behavior and model are used both within AUTOSAR and by automotive function developers without being directly interchangeable. In this document, the terminology as defined in section 1.3 will be used.

1.3 Terminology

In this section the terminology as used throughout this document is defined. The definitions are to some extend specific for the scope of AUTOSAR and especially for this deliverable. Common use of these terms, however, is taken into account as far as possible.

- An **Authoring Tool** is an AUTOSAR tool operating on any form of AUTOSAR models describing systems (software component, ECU hardware, network topology and system constraint descriptions). It is regarded to be a design entry tool for AUTOSAR descriptions according to the respective templates. Typical functions may include creating, retrieving, modifying, validating and storing such descriptions. An authoring tool may provide a tool specific language or notations for the design entry, typically used as the expressive language at the tool's user in-

terface. These languages might differ from those used for AUTOSAR standard description formats. E.g. a graphical behavior modeling tool could be used to edit a software component description using a behavior modeling language, stored as an XML file according to the software component template. Being an authoring tool is thus more a dedicated role of a tool than a classification of a tool itself.

- **AUTOSAR Model** is a generic expression for any kind of representation of instances of the AUTOSAR meta model. It might be a set of files in a file system, an XML stream, a database or memory used by some running software, etc.

- **AUTOSAR Tools** are software tools that may occur within the AUTOSAR methodology and support interpreting, processing and/or creating of AUTOSAR models

- **AUTOSAR Authoring Tools** are AUTOSAR tools operating on any form of AUTOSAR models describing systems (software component, ECU hardware, network topology and system constraint descriptions).

- **Behavior** is used in two major variants. On the one hand, behavior is used as an abbreviation for `InternalBehavior` – as part of a description of a software component template. On the other hand, behavior is a common control engineering term used to identify the functional input/output relation of a control design over time. Throughout this document the term behavior and combinations of this term like behavior models should be understood in this control engineering interpretation. To avoid any misunderstandings the term functional behavior will be used – opposed to `InternalBehavior`.

- A **Behavior Model (BM)** is a specification or design model expressed in a functional behavior modeling language.

- A **Behavior Modeling Language (BML)** is a (often graphical) notation primarily used to capture a functional behavior specification or design of a function or system. Usually, a functional behavior modeling language is regarded to be executable, i.e. its semantics is sufficiently precise to execute functional behavior models by means of a simulation engine. Furthermore, the precision in its semantics then allows the transformation of a functional behavior model into a source code of in a programming language like C. Many functional behavior modeling languages are based on finite state machine or data flow semantics.

- A **Behavior Modeling Tool (BMT)** is used to edit a functional behavior model in a functional behavior modeling language.

- A **Model Frame** is a container for a functional behavior model consisting of structural building blocks, usually called subsystems or modules. A model frame is a contract of the functional behavior model with its environment, thus it can be regarded as the counterpart of a software component template as introduced by AUTOSAR.

2 Requirements Tracing

The requirements on this document are described in the document "Requirements on Interaction with Behavioral Models" [6]. This document contains a requirements trace matrix that indicates where these requirements are covered in this document.

ID	Requirement	Section
RS_ATBM_015	Define interaction	5

Table 1: Requirements trace matrix

3 Use-Cases for Behavior Modeling within AUTOSAR

The AUTOSAR Software Component Templates [2] covers interface descriptions of software components. According to the AUTOSAR methodology, the behavior of software components is implemented in one of the supported programming languages C, C++ or Java. Optionally, it can be modeled with BMTs from which these implementations can be generated.
To support model driven approaches, use-cases are developed in this section that explains the interaction between AUTOSAR interface descriptions of software components and functional behavior models. For better readability AUTOSAR meta model [5] terminology is avoided in this section. Each use case receives a unique identifier starting with the prefix "UC_ATBM_" (meaning "Authoring Tool Behavioral Modeling Use Case").

3.1 Behavior Modeling of Software Components

[UC_ATBM_00013]: The user aims to have modeling elements provided by specific BMTs, which represent the structure and interfaces of AUTOSAR software components.

[UC_ATBM_00014]: The user aims to model the functional behavior of atomic software components in BMT, using model building blocks of the BMT like e.g. state charts, mathematical blocks etc.

Please note: A BMT can be enabled to become an authoring tool for AUTOSAR descriptions. Although this deliverable is addressing BMTs the main documents for requirements regarding authoring tools are [11] and [12]. The same holds for use case descriptions. In this respect, UC_ATBM_0013 states that it is desirable to have modeling elements in a BMT to represent AUTOSAR elements, e.g. as a result from a conversion from an authoring tool. Implicitly, this does imply the means to do some kind of structure modeling within such a tool, but this is not explicitly addressed in these use cases.

3.2 Software Component Description to Behavior Model

[UC_ATBM_00001]: The user aims to have a software component description being converted into a functional behavior model. The structural elements of a software component description need to be translated into corresponding modeling elements which reflect the same structural information in a BMT.

[UC_ATBM_00018]: The user wants to update functional behavior models according to changes in the software component definition.

Based on **[UC_ATBM_00001]** different use-cases are possible:

3.2.1 Atomic Software Component to Behavior Model

[UC_ATBM_00002]: The user aims to model and simulate the functional behavior of one selected atomic software component in combination with an internal behavior by means of a BMT.

3.2.2 Several atomic Software Components to Behavior Model

[UC_ATBM_00003]: The user aims to model and simulate the functional behavior of several atomic software components together in a BMT, in order to do simulation experiments, where the interaction of the functional behavior of different atomic-software components can be evaluated. Different sub-use-cases are possible:

- **[UC_ATBM_00004]:** The user wants to model and simulate the functional behavior of the atomic software components of a selected composition by means of BMTs, where the hierarchical structure of the composition is converted into a structured behavior model.

- **[UC_ATBM_00005]:** The user wants to model and simulate the functional behavior of selected atomic software components, which need not to belong to one composition.

- **[UC_ATBM_00006]:** The user wants to model and simulate the functional behavior of all or a set of atomic software components, which are allocated on the same ECU but do not necessarily belong to the same composition, in order to evaluate the interoperability of "functions" that will be mapped to the same ECU.

- **[UC_ATBM_00016]:** The user aims to model and simulate the functional behavior of "distributed functions" realized by compositions, where the atomic software components will be mapped to different ECUs.

- **[UC_ATBM_00017]:** The user wants to model and simulate the functional behavior of selected atomic software components, where the underlying topology aspects like network communication latencies are taken into account.

  *Please note: This use-case is not covered within this document (see also section 4.1.2 "*Modeling Distributed Systems*").*

3.3 Behavior model to Software Component Description

- **[UC_ATBM_00007]:** The user aims to convert a functional behavior model to a software component description. The structural elements of a software model need to be translated into corresponding modeling elements of a software component description.

- **[UC_ATBM_00020]:** The user aims to update a software component description which has already been converted according to **[UC_ATBM_00007]**.

Based on **[UC_ATBM_00007]** different sub-use-cases are possible:

3.3.1 AUTOSAR conform Behavior Model to Software Component

[UC_ATBM_00008]: The user aims to convert a functional behavior model with BMT specific model elements that correspond to AUTOSAR modeling concepts to a structured software component composition, i.e. the original structure of the functional behavior model is conserved by converting all structural elements of the functional behavior model to AUTOSAR software components.

3.3.2 Legacy Behavior Model to Software Component

[UC_ATBM_00009]: The user aims to take an existing legacy functional behavior model, which does not necessarily correspond to the AUTOSAR modeling concepts, and attempts to create the description of components.

Different sub-use-cases can be derived from this use case:

- **[UC_ATBM_00010]:** The user aims to convert the legacy functional behavior model to only one atomic software component in combination with one component functional behavior. The structure of the functional behavior model is flattened.

- **[UC_ATBM_00011]:** The user aims to convert a hierarchical legacy functional behavior model to a composition, i.e. the original structure of the functional behavior model is conserved by converting all structural elements of the functional behavior model to AUTOSAR software components. In general, this conversion requires a correspondence between the modeling concepts of the underlying BMT with AUTOSAR templates.

- **[UC_ATBM_00012]:** The user aims to convert the legacy functional behavior model to an AUTOSAR software component description by first introducing AUTOSAR conform functional behavior model elements and then converting the functional behavior model according to use-case **[UC_ATBM_00008]**.

3.4 Combined Use-cases

3.4.1 Behavior Authoring within BMTs

[UC_ATBM_00021]: The user wants to create an AUTOSAR conform functional behavioral model from the scratch for simulation purposes without basing it on a given SW-C description.

[UC_ATBM_00019]: The user wants to have an interface description of software components converted to a functional behavior model frame (see **[UC_ATBM_00001]**). Then he wants to e.g. define the internal behavior (runnables) in the BMT based on simulation experiments. From the resulting functional behavior model a description of the functional behavior shall be converted **[UC_ATBM_00007]** and merged with the interface description of software components.

Please Note: This use-case also covers the merging aspects of the interoperability of authoring tools [12].

4 Parts of the AUTOSAR Meta Model Relevant for Behavior Modeling

4.1 Introduction

In this chapter parts of the AUTOSAR meta model which are relevant for functional behavior modeling are identified. Modeling features are described as combinations of meta-classes contained in the selected parts. The selection is driven by common experience in using tools for functional behavior modeling of automotive control functions and by the use cases as stated in chapter 3. The result is a restricted but first reasonable step. Any changes in the AUTOSAR meta model and new requirements derived from e.g. advanced processes and methodologies might lead to a revision of this first step.

4.1.1 Separation of Feature Categories of the Software Component Template

Within the software component template, two main categories of classes in the meta model can be distinguished:

- Meta model classes which are relevant for functional behavior modeling of a software component itself. The resulting functional behavior model might also be the basis for automatic code generation for the software component implementation.

- Meta model classes which are relevant for modeling a software component environment. The resulting model is not used for automatic code generation for the software component implementation.

An example may clarify this topic. The reaction to an RTEEvent is part of the software component and thus belongs to the model of the software component. The generation of the RTEEvent in the real ECU software is realized by the RTE together with the operating system. The generation of RTEEvents is not part of the software component's functional behavior model, which might be the basis for code generation, thus for simulation purposes it is done in the software component environment.

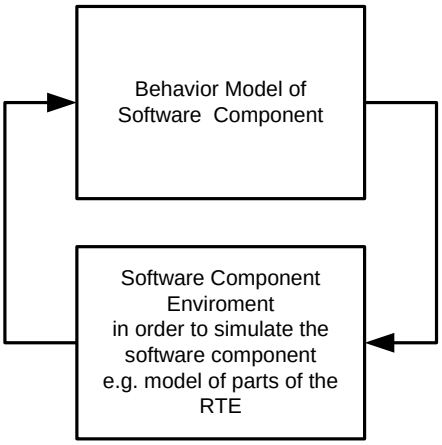

Figure 3: Distinction between functional behavior modeling of the software component and software component environment

The following features are identified, based on the software component template. The identified features are listed in section 4.2 and 4.3, by annotating the AUTOSAR meta model following the principles in [7].

- Software Component
  - Atomic Software Component and Ports
  - Interface
  - Sensors and Actuators
  - Composition
  - Datatypes
  - Constants
  - Sender Receiver Annotation
  - Services
  - Runnable Entities
  - Inter Runnable Communication
- Software Component Environment
  - **C**omSpec
  - RTEEvents
  - Execution Constraints
  - Services

4.1.2 Modeling Distributed Systems

Section 4.1.1 introduces the approach concerning the functional behavior modeling of software components which are not mapped to ECUs, or which are in the scope of a single ECU.
As listed in the use-case [UC_ATBM_00017] the functional behavior of software components, which are mapped to a networked system, communicating via different communication channels and protocols, is of significant interest. The simulation of networked systems in combination with well defined timing information and constraints can be used to capture high-level requirements and to do a formal analysis of whether or not the designed system can meet the defined requirements. Therefore we consider that in a later stage there may be the need for guidelines, where the parts of the AUTOSAR metamodel relevant for modeling distributed systems (mainly meta classes of the system template [5]) will be identified. Also several levels of abstraction regarding modeling and simulation of distributed systems and their timing behavior will be defined.

4.2 Software Components

4.2.1 Atomic Software Component and Ports

Software Components are the essential part modeled within BMTs. Relevant meta-classes are mainly `AtomicSoftwareComponentType`, `Characteristic` and `PortPrototype` as well as its subclasses `RPortPrototype` and `PPortPrototype` (see Figure 4).
The meta class `Characteristic` is used to describe constant parameters or lookup-tables of behavior models. Thus by including this meta class, the user of a BMT is able to provide the functional behavior model internal parameters and lookup tables in the AUTOSAR description. And the other way round, the defined `Characteristic` of software components can be accessed by modeling with BMTs.
If the attribute `isInVariantTable` is true it is possible to change this Characteristics by, for example, end of line programming.

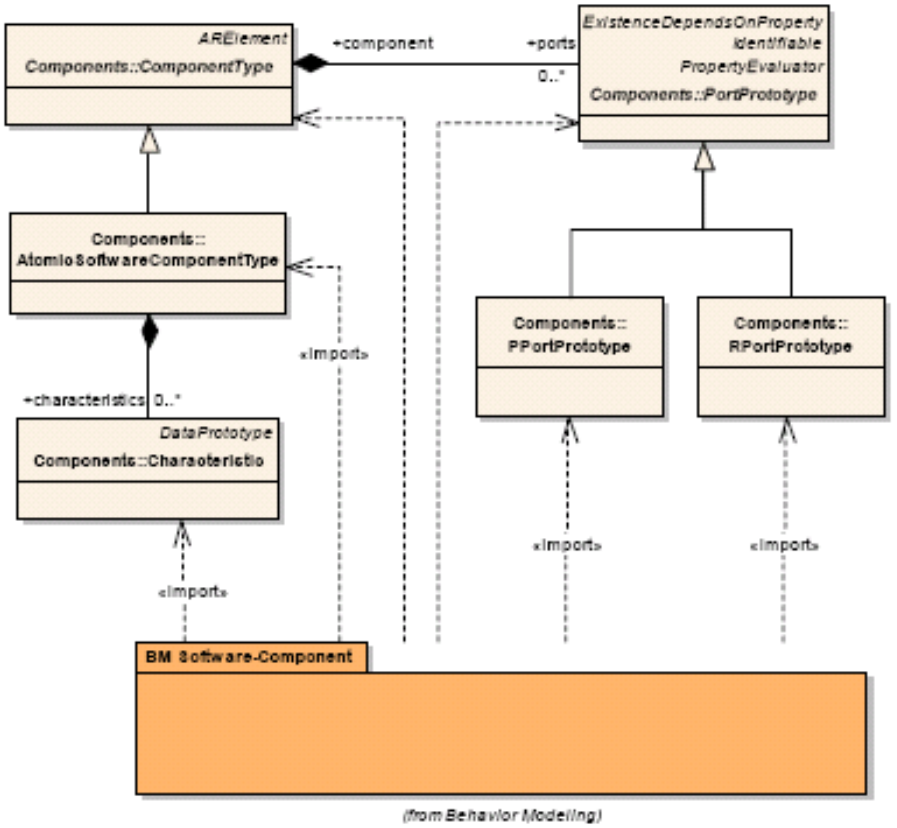

Figure 4: Atomic Software Component and Ports

4.2.2 Interface

For modeling interfaces within BMTs the meta model object `DataElementPrototype` is regarded to be sufficient (see Figure 5). Application `ClientServerInterface` will be considered in a next version. Furthermore, the meta class `PortInterface` is not imported into the package because the Port-PortInterface-pattern implemented in the AUTOSAR meta-model will most likely only be available in some of eligible BMTs. However, it can safely be assumed that a port concept is available to some extent. As the consequence, for BMTs not supporting the Port-PortInterface concept, the information gathered in a `PortInterface` and a `Port` typed by the `PortInterface` should be merged in the process of creating or updating a functional behavior model frame.
In other words, the representation of an AUTOSAR `Port` in these BMTs contains both the information of `Port` and `PortInterface`. Consequently, it is not easily possible to implement a way back from the functional behavior model to the AUTOSAR description in these cases. Of course, it would technically be feasible to separate the information appropriately during the process of converting the model frame back to a `ComponentType` description.

But since the type contract of `Port` and `PortInterface` has been cancelled in the process of merging Port and `PortInterface` for behavior modeling, it is difficult (in terms of implementation effort and risk) to observe whether information regarding the `PortInterface` have been changed in one of the `Ports` typed by a specific `PortInterface`.
In addition, the compatibility rules for connecting `Ports` with each others must be observed especially if the connection of Ports is created or maintained within a functional behavior model consisting of several connected `ComponentPrototypes`. A change of the connection of `ComponentPrototypes` without a strict observation of compatibility rules (which might or might not be difficult to implement in an arbitrary BMT) especially in combination with an attempt to feed back the change to the AUTOSAR description is potentially hazardous.
Therefore, the existing concept for the interaction with functional behavior models (as described in this document) does not necessarily require an explicit support for `PortInterfaces`. As reasoned before, the information attached to `Ports` and `PortInterfaces` can also be merged in the process of creating or updating the functional behavior model frame under certain conditions.

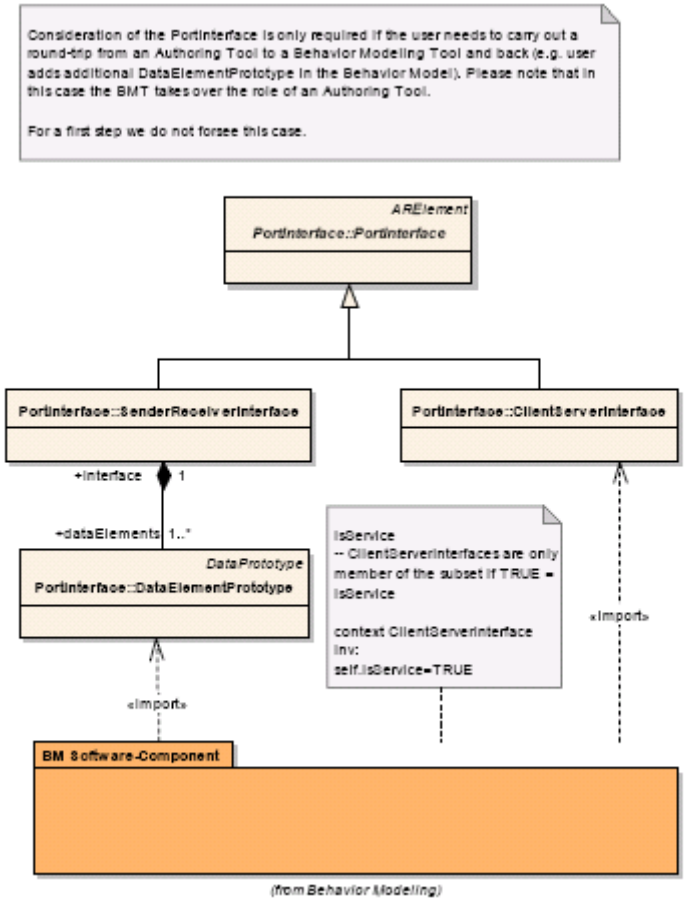

Figure 5: Interface

4.2.3 Sensors and Actuators

`SensorActuatorSoftwareComponentType` defines the mapping (link or interface) to physical world. Modeling of `SensorActuatorSoftwareComponentType` within BMTs is a key feature. The reference to `SensorActuatorHW` is not considered necessary within the BM.

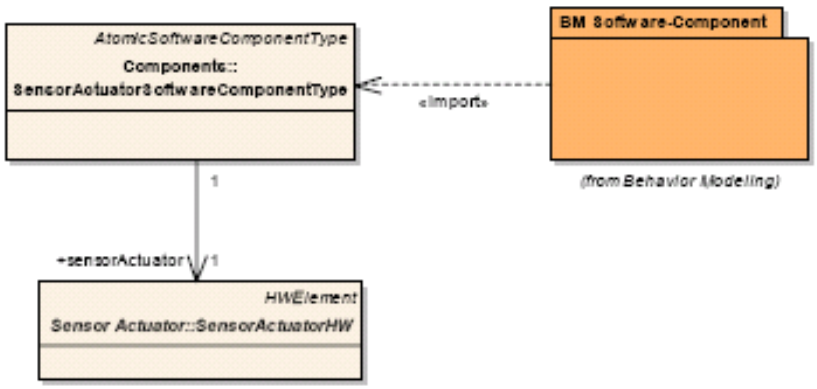

Figure 6: Sensors and Actuators

4.2.4 Composition

For modeling `compositions` within a BMT, the `AssemblyConnectorPrototype` is also required (see Figure 7) to connect a p-port to an r-port. When modeling a `Composition` within a BMT it has to preserve typing constraints. The `Assembly-ConnectorPrototype` does not contain any information about the execution behavior of the underlying communication, e.g. blocking/non-blocking, synchronous/asynchronous etc., nor about the underlying communication characteristics, e.g. the bus protocol, bus bandwidth, etc. In some BMTs the port connectors may contain information and refinement regarding different layers of communication. In this case the relevant elements of the AUTOSAR description should be used to retrieve the required information for the BMT or to import the information from the BMT to the AUTOSAR description. On the other hand `AssemblyConnectorPrototype` may contain some data to exchange attributes concerning the communication between sender and receiver ports, e.g. encryption of the transmitted information, maximum transfer time for `DataElementPrototypes`, etc., these attributes should be considered with care for proper exchange between the BMT and the AUTOSAR description.

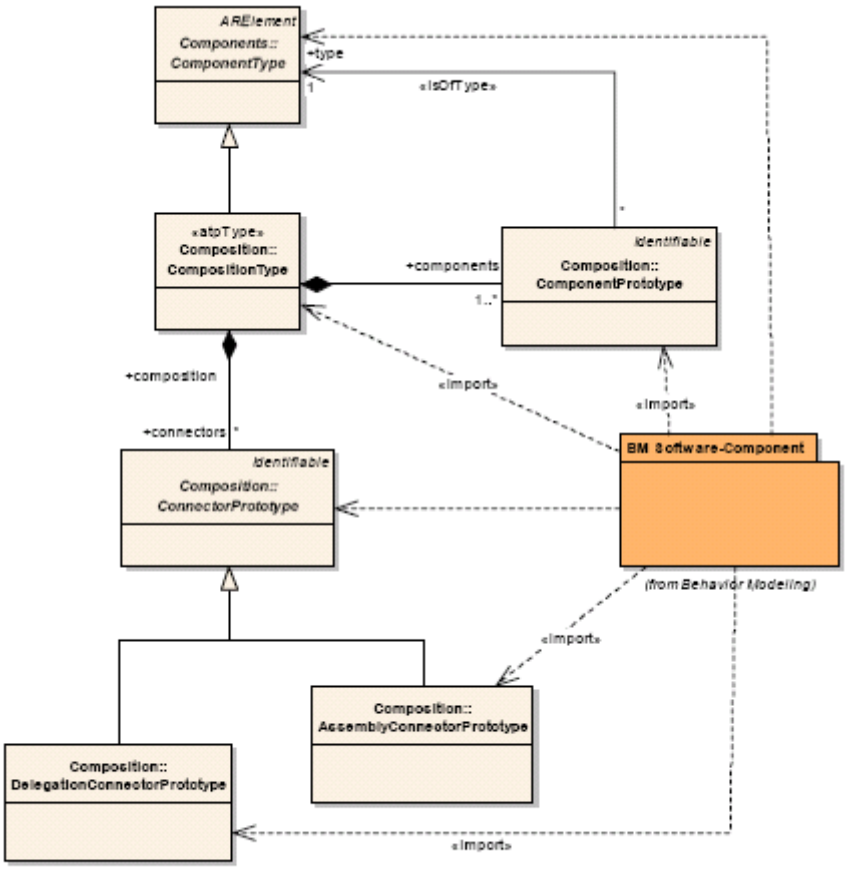

Figure 7: Composition

4.2.5 Datatypes

Primitive `DataTypes` like `BooleanType`, `IntegerType`, `RealType` and primitive type with associated semantics (`PrimitiveTypeWithSemantics`) should be realized within the BMTs (see Figure 8).

- The meta class `PrimitiveTypeWithSemantics` provides the link to physical world. This meta class includes MSR definitions. These are not considered in the first step, except for `SW-COMPU-METHOD`, `LOWER-LIMIT` and `UPPER-LIMIT` [15].

- The datatype `OpaqueType` represents an array of exactly `numberOfBits` bits. This datatype is not considered in the first step.

- Whether the datatypes `StringType` and `CharType` are required for functional behavior modeling depends on use-cases. In this document version they are not considered.

- CompositeTypes are not considered. It can be noted that there is a distinction between CompositeTypes which do not exceed 8 bytes and CompositeTypes which exceed 8 bytes, since there is no transport protocol defined for COM.

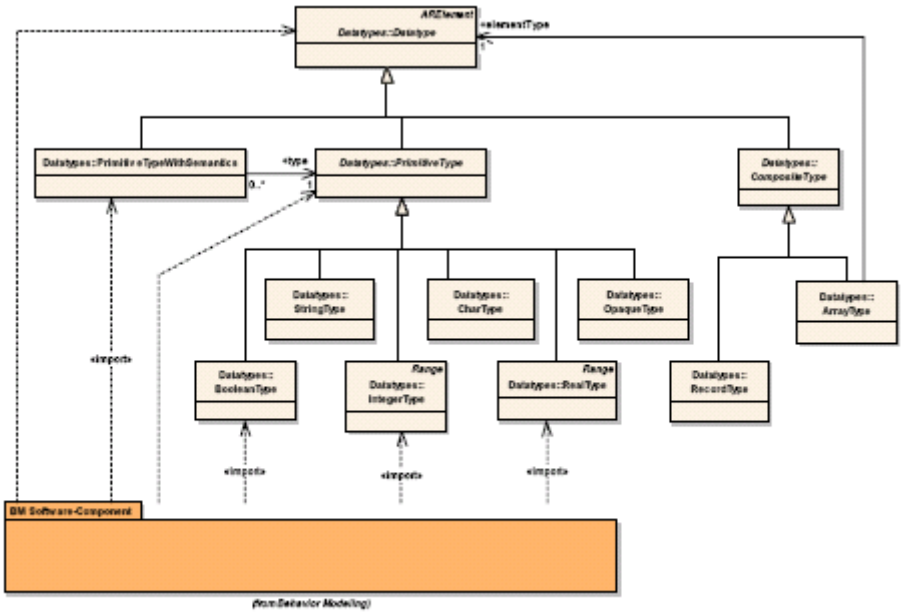

Figure 8: Datatypes

4.2.6 Constants

According to the `Datatypes`, the corresponding `Constants` should be modeled within BMTs (see Figure 9). Please note that the data type float is regarded to be more capable than e.g. the data type integer, thus numeric constants of type integer are implicitly covered as well.

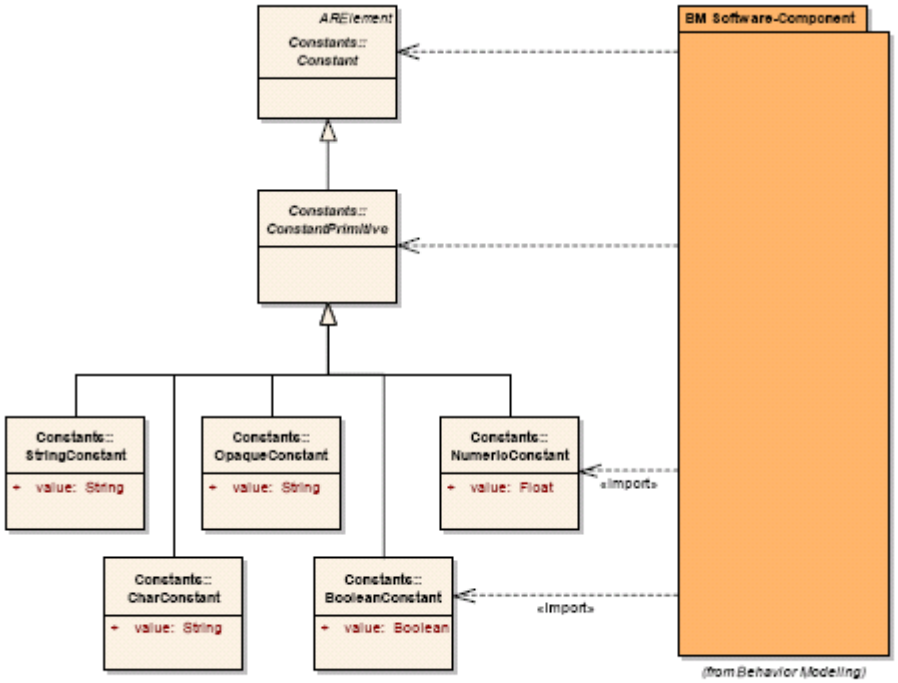

Figure 9: Constants

4.2.7 Sender Receiver Annotation

The `SenderReceiverAnnotation` is driven by the application. The `SenderReceiverAnnotation` annotates data elements in a port that realizes a sender receiver interface. The attributes `ProcessingType`, `Computed` and `LimitType` may provide the user of the BMT with additional information about data elements (Figure 10). Attributes are Figure 10:

- `ProcessingType` indicates the kind of processing applied to the data element.
- "raw" specifies that a signal is taken directly from the basic software modules, i.e. from the ECU abstraction layer. It indicates to a developer that the control algorithm in the software has to provide filters.
- "filtered" indicates that a raw signal has been manipulated by some application software components by using filters.
- "none" is specified if none of the previous two options apply.
- The Flag `Computed` indicates that this data element was not measured directly but instead was calculated from possibly several other measured or calculated values.
- `LimitType` indicates whether the data element carries a minimum or maximum value, thereby limiting the current range of another value.
- The relevance for `ReceiverAnnotation` and the usage of the attribute `SignalAge` is not finally clear. The attribute `SignalAge` is a requirement specified on the receiver side. It might be used to inform the user of the BMT about the maximum allowed age of the signal since it was originally detected by a sensor.

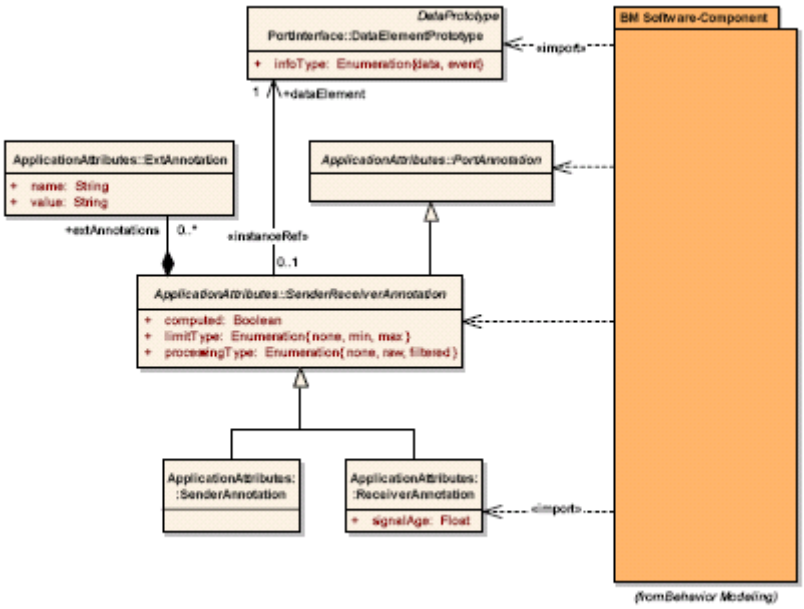

Figure 10: Sender Receiver Annotation

4.2.8 Services

Software components can use services provided by the basic software via RTE. It should be especially possible within the BMT to model synchronous and asynchronous `ServerCallPoint` (see Figure 11), as this latter provides an interface mechanism for the interaction of the software components with the basic software.
The interaction of the software component with the service may be represented within the BMT with different levels of granularity:

- As a generic service, where one modeling block is used to represent all required services (e.g. NVRAM-Manager and ECU-State-Manager, …)
- A unique modeling block for each service (e.g. NVRAM-Manager)
- A unique modeling block for each operation of a service (e.g. NvM_ReadBlock)

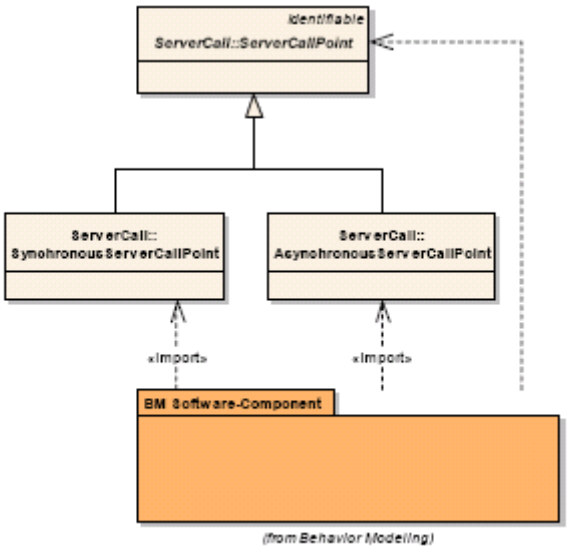

Figure 11: Services

4.2.9 Runnable Entities

The runnable entities represent the smallest code-fragments that are provided by the component and are executed in the RTE. `RunnableEntity` is the key feature to be modeled within BMTs.
For communication the following classes need to be modeled within BMTs (see Figure 12): `DataWriteAccess`, `DataReadAccess`, `DataSendPoint`, `DataReceivePoint`.

- `ModeDisablingDependency` is not regarded as long as this concept is not fully developed. As it is a key concept for runnables it has to be regarded, as soon as its definition is complete.

- The support for `Waitpoints` within a BMT strongly depends on its modelling capabilities. This document version therefore does not not strictly consider `WaitPoints` and thus no category 2 runnables. These concepts are related to the class of application level client server communication which is not assumed for the first version.

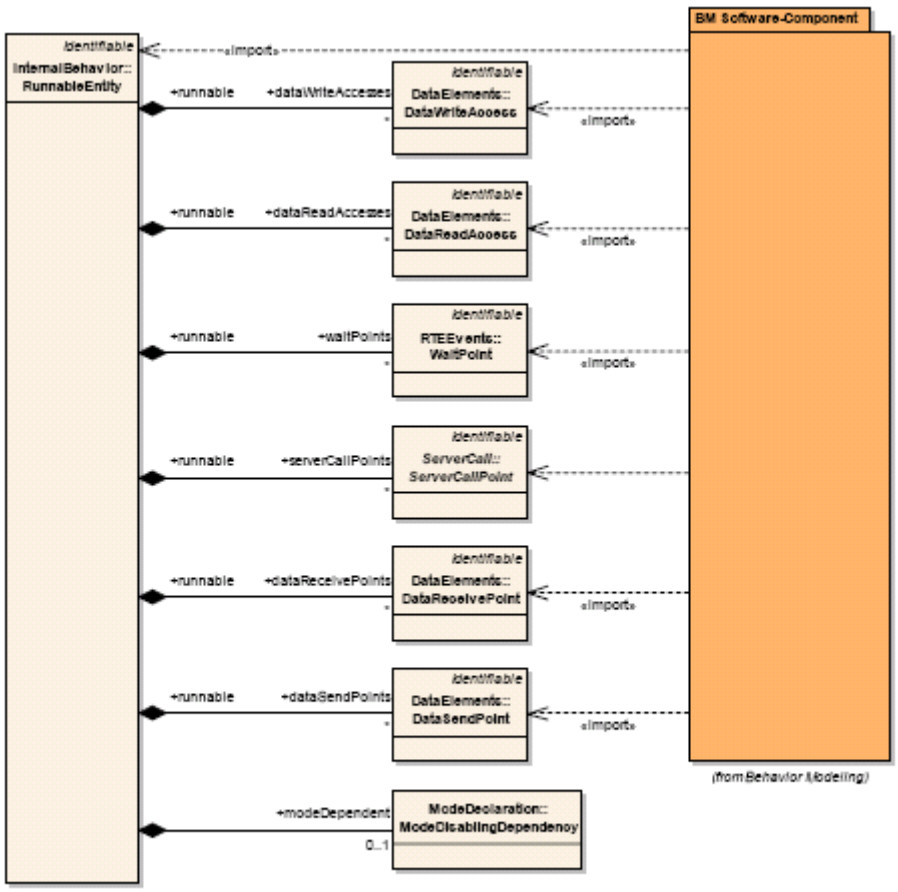

Figure 12: Runnable Entities

4.2.10 Inter Runnable Communication

Inter runnable communication is an important communication concept on its own. A specific use of `InterRunnableVariables` as well as `ExclusiveAreas` is to ensure data consistency. For inter runnable communication both concepts are included. The decision to use one or the other concept, highly depends on the capabilities of the BMT.

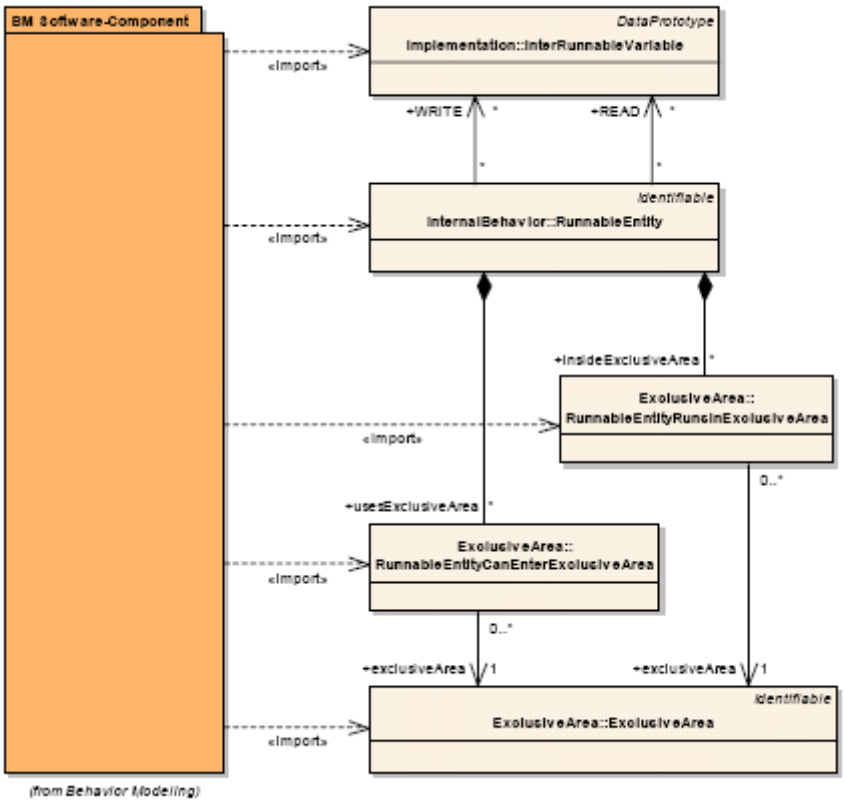

Figure 13: Inter Runnable Communication

4.3 Software Component Environment

4.3.1 ComSpec

In the `ComSpec` all communication relevant attributes are defined. These are application independent attributes opposite to the application dependent attributes which are defined in `ApplicationAttributes` in the feature Sender Receiver Attributes. These attributes are not considered to be part of the application and can be changed without effecting the application.

RPortComSpec defines communication attributes of an R-Port (see Figure 14 and Figure 16). This class contains attributes that are valid for all kinds of required ports, independent of client-server or sender-receiver communication patterns.

- The `InitValue` attribute of `DataReceiverComSpec` can be used within an environment model for communication modeling (see Figure 14).

PPortComSpec defines communication attributes of a P-Port (see Figure 15 and Figure 17). This class contains attributes that are valid for all kinds of provided ports, independent of client-server or sender-receiver communication patterns.

- The ServerComSpec provides the attribute queueLength, and since the queue is realized by the RTE, ServerComSpec is part of the environment (see Figure 17).

- The `InitValue` attribute of `DataSenderComSpec` can be used within an environment model for communication modeling (see Figure 15).

- The class AcknowledgementRequest acknowledges that data has been sent successfully. The success or failure of data transfer is reported via the SendPoints of the Runnables. The AcknowledgementRequest could be part of the environment for simulation, to specify a communication timeout (see Figure 17).

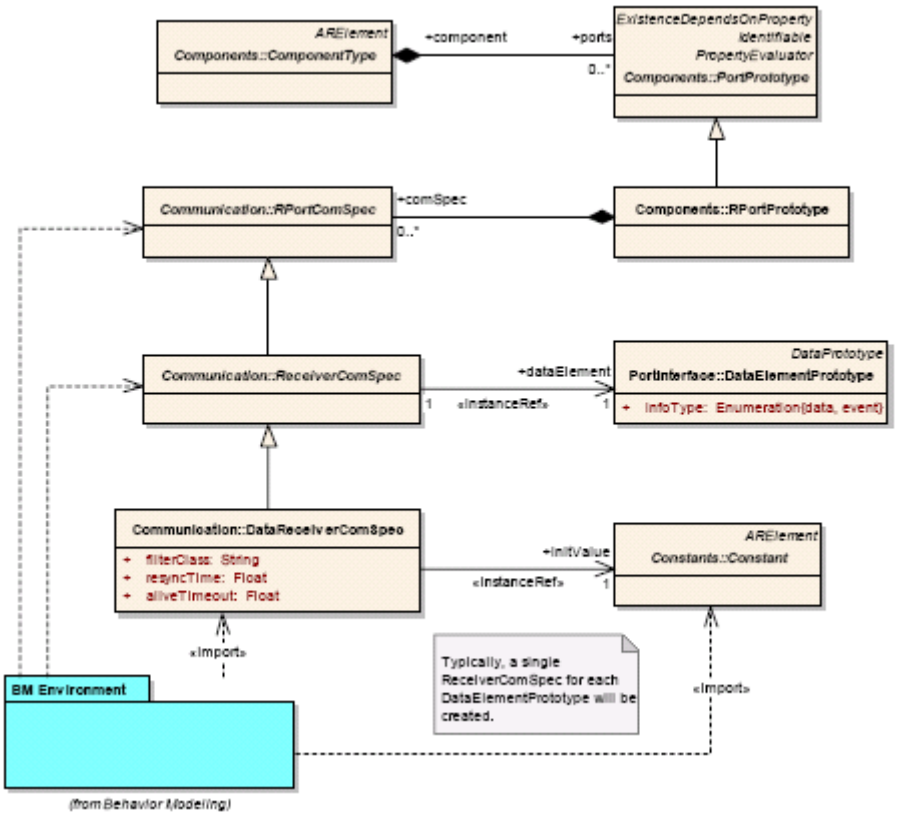

Figure 14: ReceiverComSpec

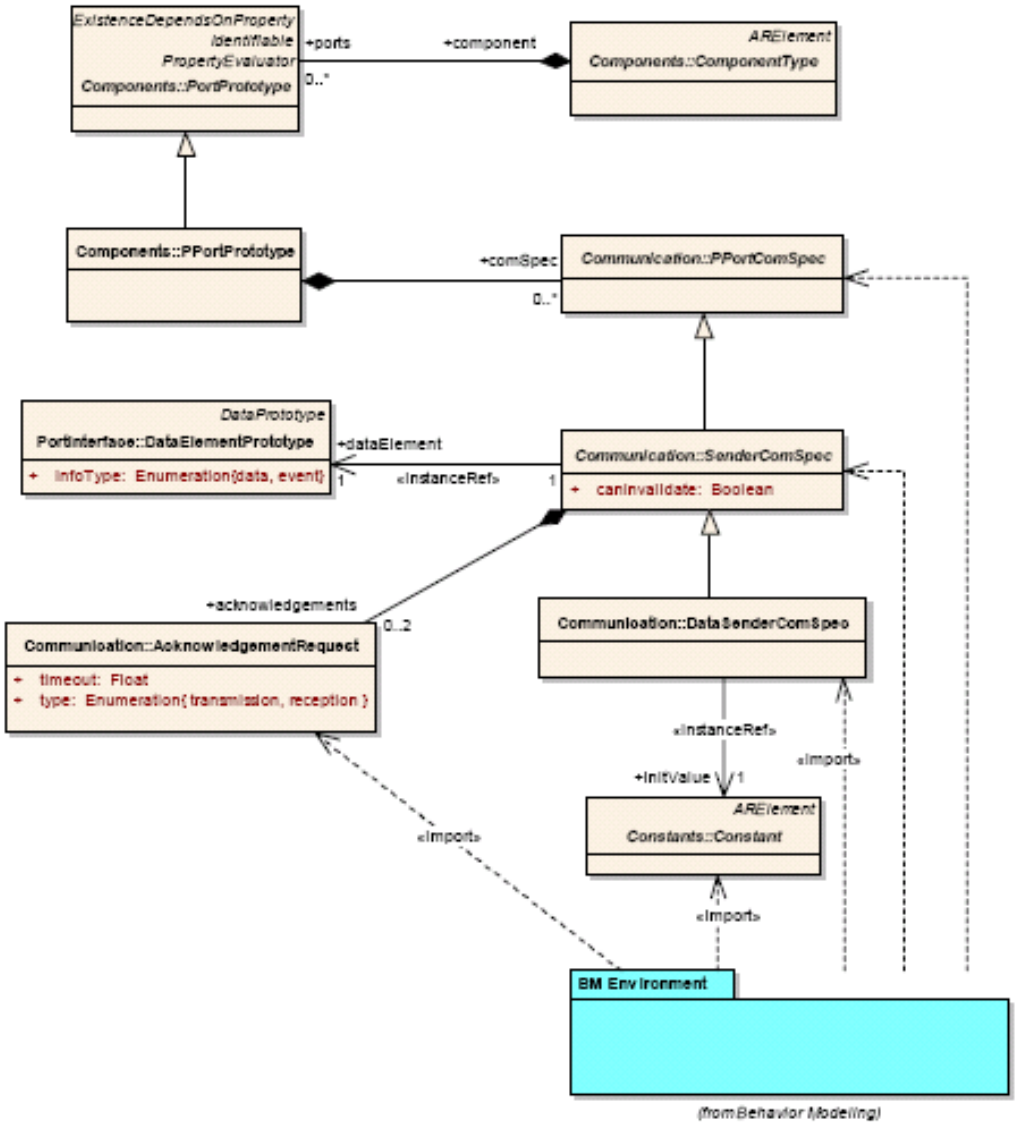

Figure 15: SenderComSpec

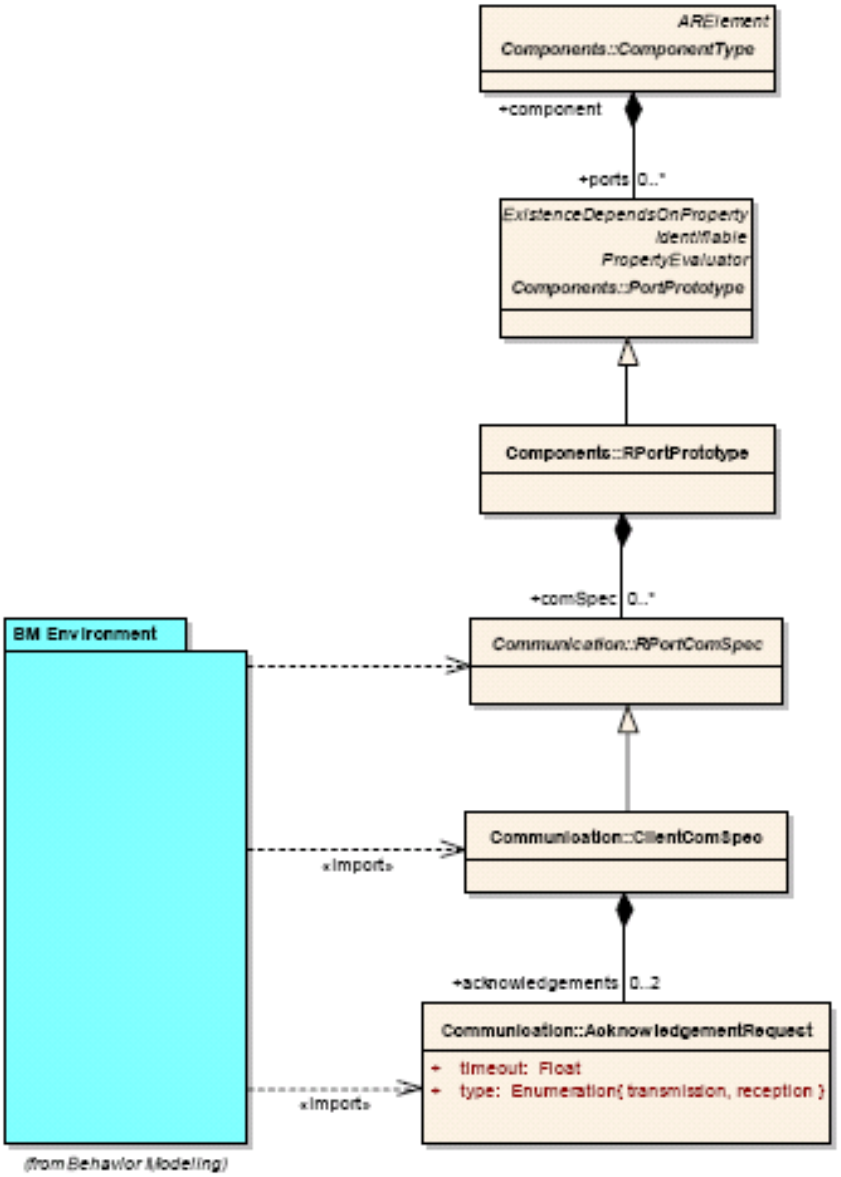

Figure 16: ClientComSpec

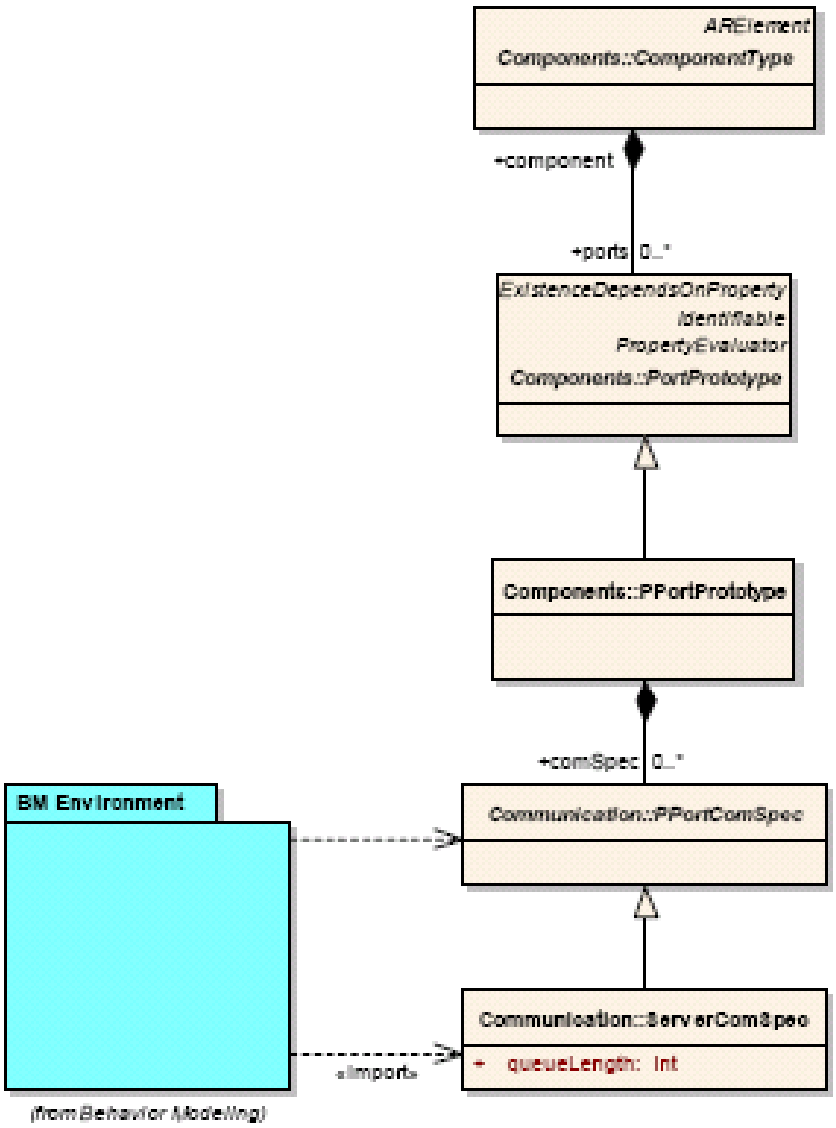

Figure 17: ServerComSpec

4.3.2 RTEEvents

RTE events implement execution triggers for runnables. The generation of RTE events within a BMT, for simulation purposes or other use, is seen to be part of the software component environment. The `RunnableEntity`, which reacts on the `RTEEvent` is part of the behavior model of the software component (see Figure 18). This distinction is made since the `RTEEvent` has no impact on the generated code of the software component. The `OperationINvokedEvent` references the operation invoked by the client and is required for client server communication. The `ModeSwitchEvent` is listening to mode changes notified by the `StateManager`. It

can be considered in a future document version when the Mode Management concept is finalized.

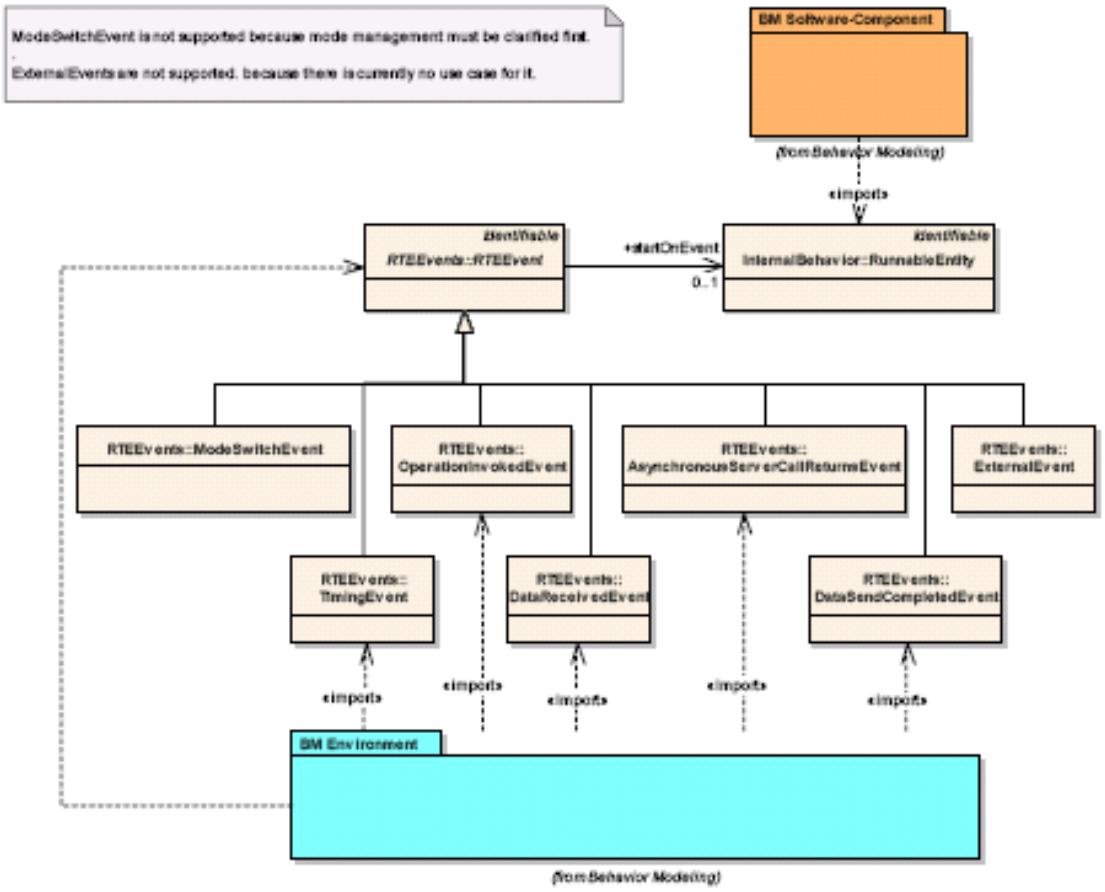

Figure 18: RTEEvents

4.3.3 Execution Constraints

Execution constraints may be used for the modeling and configuration of the software component environment, to coordinate the activation of runnable entities for simulation purposes. The `ExecutionConstraints` specifies the execution order of the runnables. The allowed and not allowed execution behaviors can be formulated by using the ordered-reference (see Figure 19).
For example it can be specified that a runnable can only execute if all other runnables have completed their execution. It is not intended for cyclic activation constraints. This concepts has some overlapping with the mode concept.

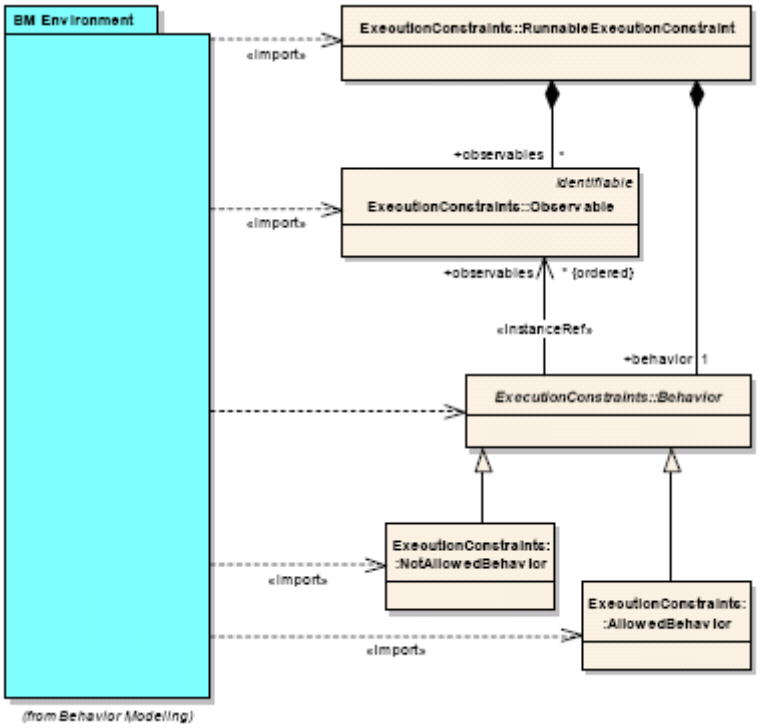

Figure 19: Execution Constraints

4.3.4 Services

Simulating the environment of a software component, may require the modelling of services. The environment simulation may provide the possibility of interaction between the software component and the service. For more information on the services see 4.2.8 and Figure 11.

5 Requirements for the Interaction of SW-C descriptions with Behavior Models

5.1 Conversion SW-C and Behavior Models

5.1.1 Tasks of the Converter

Based on the use-cases, and identified parts of the AUTOSAR meta model relevant for functional behavior modeling, in this chapter requirements for the conversion of SW-C descriptions to functional behavior models (and vise versa) are derived.
It is important to note, that the subject is about the conversion of SW-C descriptions and functional behavior models, not the coupling of the tools, like coupling of an authoring tool with a BMT. Thus the requirements in section 5.2 and the following sections, are independent of a specific tool realization.
The conversion will be done by a tool, so called **"Converter"** (see **Figure 20**). The converter can be realized in different ways e.g., it could be

- part of an authoring tool,
- part of a BMT,
- a stand-alone tool.

In this document it is assumed, that there are two different tools, an authoring tool, where the structure of the software component is defined and a BMT where the functional behavior is modeled.
It might be the case that an authoring tool additionally offers functional behavior modeling capabilities. In this case, the conversion is implicitly covered by the tool.

5.1.2 Directions of the Conversion

Figure 20 also outlines, that there are two directions of the conversion:

- From software component description to the functional behavior model of software components.
- Form the functional behavior model to the software component description.

The converter may need additional configuration or user information for the conversion.

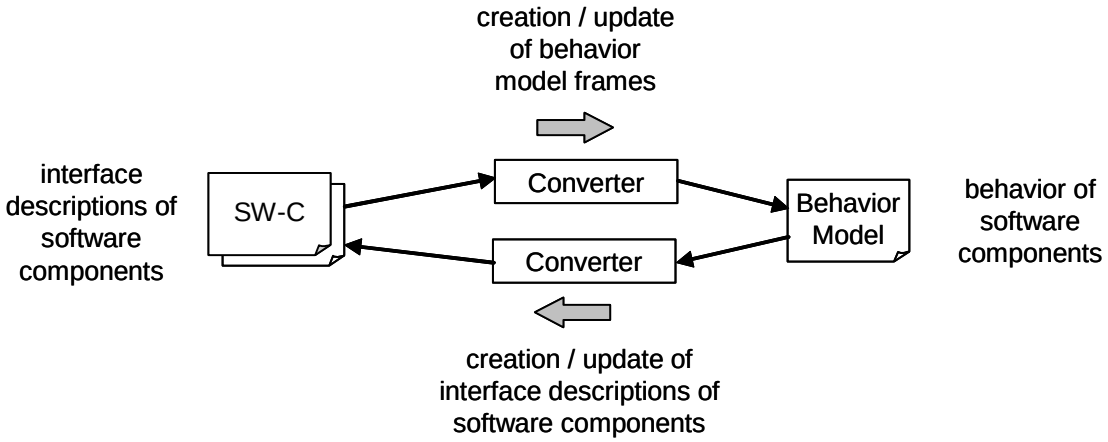

Figure 20: Coupling of SW-C with Functional Behavior Models by the Converter

For each direction the following different cases can be distinguished.

- The initial creation of an empty functional behavior model (so called model frame), based on a software component description, and the update of an already existing functional behavior model (model frame including already model functional behavior) according to the changes in the software component description (see e.g. **[UC_ATBM_00001]** and **[UC_ATBM_00018]**).
- The initial creation of a software component description, based on a legacy functional behavior model and the update of a software component description according to changes in the functional behavior model. (see e.g. **[UC_ATBM_00007]** and**[UC_ATBM_00020]**).

5.1.3 Round-trip Issue –Type concept

The requirement [TR_ATBM_00005] allows for a model-driven, forward-engineering iterative development process. The AUTOSAR model is considered as the master and the functional behavior model is derived from that.
If the requirement [TR_ATBM_00032] is fulfilled, then backward or reverse engineering is also possible. This is most often used to get started with legacy systems.
A combination of both engineering steps yields in round-trip engineering. The round-trip engineering is not a necessity for the successful implementation of AUTOSAR.

Figure 20 outlines that there are two directions of conversion. However performing a round-trip with 1) creation of a functional behavior model frame based on a SW-C description and 2) feed this functional behavior model back to the converter for creating of a SW-C description has to be handled with special care.
If the BMT does not support a proper type concept for software components the resulting SW-C description after the round-trip is different to the original SW-C, since Prototypes will be changed in types after the round-trip. Consequently inconsistencies would exist, which have to be avoided. Thus two different cases can be distinguished (see below):

- **Modeling Compositions: Design contract ensured by the BMT**
- **Restriction to Modeling Atomic Software components, export simulation experiments**

Descriptions of use-cases and requirements in this document show that unlimited round-trip engineering capabilities between different authoring tools and BMTs are desired by many users.
Due to the complexity of the AUTOSAR meta model and the fact that most of the involved tools will probably only cover parts of the meta model (and thus will be specialized for certain purposes), the tasks of converting and merging model parts from the different sources appropriately may become very complex. For this reason, tool support will most likely be feasible (or economically justifiable) only for some of the above-mentioned use-cases.

Modeling Compositions: Design contract ensured by the BMT
When modeling compositions, a BMT has to ensure, that a structural model building block representing a `ComponentPrototype` MUST NOT be changed without taking all other structural model building blocks representing `ComponentPrototyes` typed by the same `ComponentType` into account. In other words: the structural model building blocks representing the `ComponentPrototypes` typed by the same `ComponentType` MUST NOT implement different behavior. This contract must be kept during the mapping of the `CompositionType` to a functional behavior model.
Please note that the only case where the structural model building blocks would be allowed to implement a different functional behavior would obviously be the case where more than one `InternalBehavior` references a specific `AtomicSoftwareComponentType`.
The design contract must be kept for variants of this use case (e.g. functional behavior modeling of all software-components mapped to a single ECU **[UC_ATBM_00006]**, etc.) as well.

Restriction to Modeling Atomic Software components, export simulation experiments
If a BMT cannot ensure the above design contract the modeling of functional behavior has to be limited to `AtomicSoftwareComponentTypes`. For creating simulation experiments the functional behavior of a composition can be derived by (automatically) creating a functional behavior model on the basis of the topology of the `CompositionType`. Please note that this is a one-way-approach: it would not be possible to feed back this functional behavior model into the `CompositionType` again.

5.1.4 Generation of the Software Component Environment by the Converter

Optionally the converter should generate the software component environment for simulation purposes (see **Figure 21**). Optionally the converter can use system configuration information to perform additional tasks, e.g. the mapping of data elements to system signals.

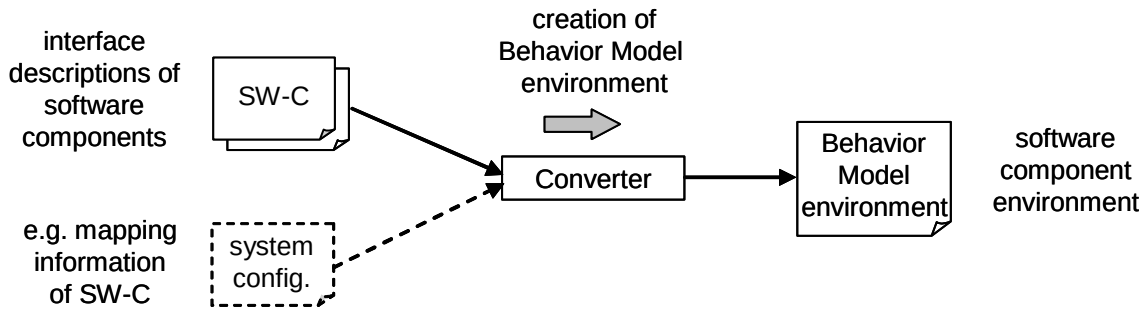

Figure 21: Generation of the Software Component Environment by the Converter

5.2 Creation of functional behavior models

The remainder of this chapter contains the requirements on interaction with functional behavior models. Some of the requirement specifications are further illustrated with graphical descriptions. For these figures only simple and abstract graphical elements are used – instead of any specific AUTOSAR graphical notation. Thus, the figures are intentionally not precise, but they have the same expressive style for both, AUTOSAR models and functional behavior models.

5.2.1 [TR_ATBM_00001] Creation of functional behavior model frames

Initiator:	DC
Date:	23.06.2005
Importance:	High
Requirement:	Creation of functional behavior model frames
Description:	The **converter** shall create model frames for a specific BMT based on the AUTOSAR meta model classes, defined in the parts of the AUTOSAR meta model relevant for functional behavior modeling (see chapter 4). The meta model classes should be mapped to a functional behavior model, according to the modeling heuristics of the underlying BMT and depending on the capabilities of the BMT. • The converter shall link different modeling elements together: e.g. the converter shall link the representation of ports to the representation of software components. • The converter should link the VFB-view with the RTE-view: The representation of Runnable Entities should to be linked to the representation of their software components. • The converter shall connect the functional behavior model of the software components to the software environment model. • The converter should allow to create a model frame without runnable entities in order to satisfy use-case [**UC_ATBM_00019**].
Rationale:	AUTOSAR templates cover interface descriptions of software components. According to the AUTOSAR methodology, the behavior of software components is implemented in one of the supported programming languages C, C++ or Java. Optionally, it can be modeled with BMTs from which these implementations can be generated.
Use Case:	**[UC_ATBM_00001]**: The user aims to have a software component description being converted into a functional behavior model. [**UC_ATBM_00019**]: The user wants to have an interface description of software components converted to a functional behavior model frame (see **[UC_ATBM_00001]**). Then he wants to e.g. define the internal behavior (runnables) in the BMT based on simulation experiments. From the resulting functional behavior model a description of the functional behavior shall be converted **[UC_ATBM_00007]** and merged with the interface description of software components.
Dependencies:	TR_ATBM_00003, TR_ATBM_00004, TR_ATBM_00005, TR_ATBM_00030
Conflicts:	--
Supporting Material:	--
Comment:	This requirement captures the conversion of all the meta classes defined in chapter 4.

Conversion in case of BMT-Limitations

In some cases BMTs are not able to provide a proper type-instance-pattern, as used in the AUTOSAR meta model. The lack of this pattern has influence on the mapping of AUTOSAR software components to functional behavior models. The following issues address some limitations of existing BMTs:

- If the BMT does not provide for `PortInterface`:
  - The **converter** shall eliminate the `PortInterface` and shall directly map data elements to the inputs and outputs of the software component representations.

- If the BMT does not support multiple internal behaviors which reference the same atomic software component:

  - The **converter** shall only convert the software component description of a specific combination of an `AtomicSoftwareComponentType` and an `InternalBehavior`.

  - The **converter** shall support selecting specific combinations of an atomic software component together with its internal behavior.

5.2.2 [TR_ATBM_00005] Update of functional behavior model frames

Initiator:	BMW
Date:	06.09.2005
Importance:	High
Requirement:	Update of functional behavior model frames
Description:	The **converter** should update model frames for a specific BMT based on the changes in the linked AUTOSAR meta model classes. The converter should update the functional behavior model so that all conditions of [TR_ATBM_00001] still hold after the update.
Rationale:	It is likely that the software development process is not sequential but iterative. To avoid that the user needs to build the functional behavior model from scratch for each iteration, the converter should be able to deal with incremental creation of model frames.
Use Case:	**[UC_ATBM_00018]**: The user wants to update functional behavior models according to changes in the software component definition.
Dependencies:	TR_ATBM_00001, TR_ATBM_00003
Conflicts:	--
Supporting Material:	--
Comment:	--

5.3 Conversion of Software Components

Regarding the conversion of software components by the converter the following distinct cases are identified and explained in this section:

I. conversion of one atomic software component

II. conversion of a flat user-defined selection of atomic software components, in this case the atomic software components are not grouped in an hierarchical manner

III. conversion of hierarchically structured software components by means of compositions

5.3.1 [TR_ATBM_00003] Creation of behavior model frames of an atomic software component

Initiator:	DC
Date:	20.01.05, update: 13.05.05
Importance:	high
Requirement:	Creation of behavior model frames of an atomic software component
Description:	The **converter** shall convert the `AtomicSoftwareComponentType` in combination with a specific `InternalBehavior` into structural model building blocks according to the modeling heuristics of the underlying BMT.
Rationale:	As a minimum requirement the **converter** shall convert one atomic software component into a functional behavior model frame (see Figure 22).
Use Case:	**[UC_ATBM_00001]**: The user aims to have a software component description being converted into a functional behavior model. **[UC_ATBM_00002]**: The user aims to model and simulate the functional behavior of one selected atomic software component in combination with an internal behavior by means of a BMT.
Dependencies:	TR_ATBM_00001, TR_ATBM_00005
Conflicts:	--
Supporting Material:	--
Comment:	The goal here is to create the equivalent of a software component type within the BMT.

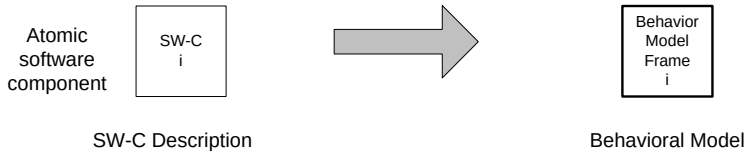

Figure 22: Creation of functional behavior model frames of an atomic software component

5.3.2 [TR_ATBM_00030] Creation of behavior model frames of a selection of atomic software components

Initiator:	DC
Date:	20.07.2005
Importance:	High
Requirement:	Creation of behavior model frames of a selection of atomic software components
Description:	In the case of a defined selection of atomic software components: • The **converter** should create functional behavior model frames as parts of one functional behavior model, where each software component is represented as a functional behavior model frame (see Figure 23). • The **converter** should inter-connect the software components, through their r- and p-ports, according to the software component description.
Rationale:	--
Use Case:	**[UC_ATBM_00003]**: The user aims to model and simulate the functional behavior of several atomic software components together in a BMT, in order to do simulation experiments, where the interaction of the functional behavior of different atomic-software components can be evaluated. **[UC_ATBM_00005]**: The user wants to model and simulate the functional behavior of selected atomic software components, which need not to belong to one composition. **[UC_ATBM_00006]**: The user wants to model and simulate the functional behavior of all or a set of atomic software components, which are allocated on the same ECU but do not necessarily belong to the same composition, in order to evaluate the interoperability of "functions" that will be mapped to the same ECU.
Dependencies:	TR_ATBM_00001, TR_ATBM_00003, TR_ATBM_00004
Conflicts:	--
Supporting Material:	--
Comment:	Compared to TR_ATBM_00003 this requirement is about component prototypes because only component prototypes can be interconnected. Please refer also to section 0 on design constraints and to section 4.2.2 on restraints about type concepts.

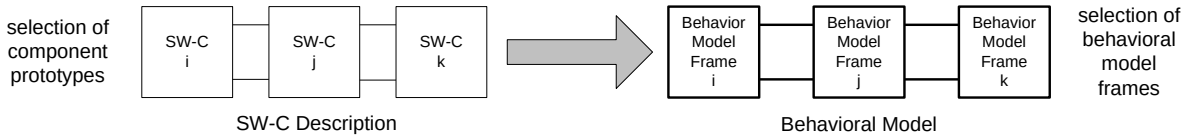

Figure 23: Creation of functional behavior model frame for a selection of atomic software components

5.3.3 [TR_ATBM_00004] Creation of a hierarchical model frame structure

Initiator:	DC
Date:	20.01.05, update: 13.05.05
Importance:	Medium
Requirement:	Creation of a hierarchical model frame structure
Description:	The **converter** should create a structured functional behavior model based on the hierarchical description of a `CompositionType` with its contained `ComponentPrototypes`, where the hierarchical structure of the `CompositionType` is converted into a structured functional behavior model (see Figure 24).
Rationale:	--
Use Case:	**[UC_ATBM_00004]**: The user wants to model and simulate the functional behavior of the atomic software components of a selected composition by means of BMTs, where the hierarchical structure of the composition is converted into a structured behavior model.
Dependencies:	TR_ATBM_00001, TR_ATBM_00003, TR_ATBM_00030
Conflicts:	--
Supporting Material:	--
Comment:	As for any other requirements dealing with prototypes the conversion should not break the design contract while creating hierarchical compositions, see also section 0. For the conversion of an `AtomicSoftwareComponentType` in combination with a specific `InternalBehavior` see also TR_ATBM_00003.

In the case of compositions the **converter** shall conserve the structure of the composition within the functional behavior model: There is no use-case in which a hierarchically composed software component structure is transformed to a flat functional behavior model (see Figure 24):

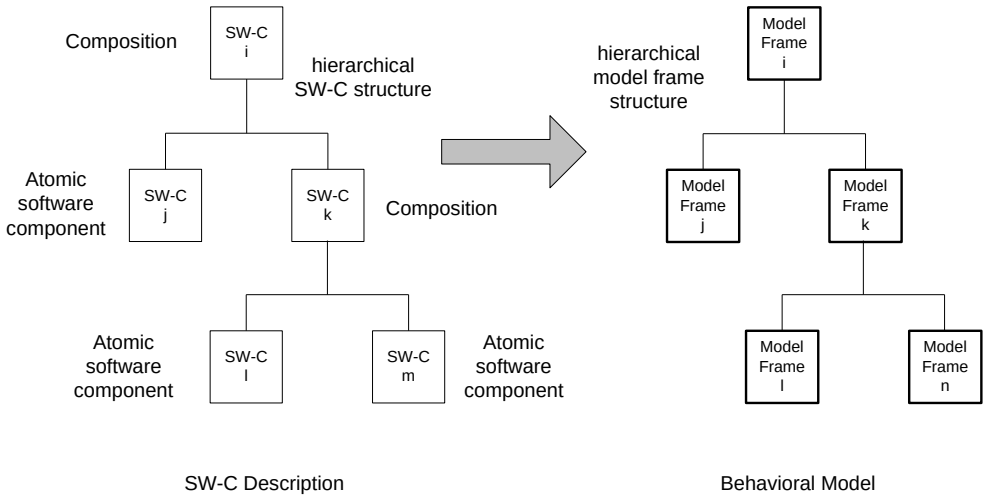

Figure 24: Creation of a hierarchical model frame structure

5.4 Creation of software component descriptions

5.4.1 [TR_ATBM_00002] Creation of interface description of software components

Initiator:	DC
Date:	23.06.2005
Importance:	high
Requirement:	Creation of interface description of software components
Description:	The **converter** shall create an interface description of a software component from a functional behavior model, modeled in a specific BMT, based on the parts of the AUTOSAR meta model relevant for functional behavior modeling, as defined in chapter 4.
Rationale:	--
Use Case:	**[UC_ATBM_00007]**: The user aims to convert a functional behavior model to a software component description.
Dependencies:	TR_ATBM_00007, TR_ATBM_00008, TR_ATBM_00028, TR_ATBM_00031
Conflicts:	--
Supporting Material:	--
Comment:	This requirement captures the conversion of all the meta classes defined in chapter 4. Conversion in case of limitations of the BMT: Some BMTs only support the port concept, but not the concept of PortInterfaces. In this case the converter shall create the required PortInterfaces when importing functional behavior models.

5.4.2 [TR_ATBM_00032] Update of interface description of software components

Initiator:	BMW
Date:	06.09.2005
Importance:	Medium
Requirement:	Update of interface description of software components
Description:	The **converter** should update an existing software component description from a linked functional behavior model, modeled in a specific BMT, based on the parts of the AUTOSAR meta model relevant for functional behavior modeling, as defined in chapter 4.
Rationale:	It is likely that the software development process is not sequential but iterative. During functional behavior modeling the software component might have to be restructured. To facilitate the process, an update of the software components from the functional behavior model should exist.
Use Case:	**[UC_ATBM_00020]**: The user aims to update a software component description which has already been converted according to **[UC_ATBM_00007]**.
Dependencies:	[TR_ATBM_00001]
Conflicts:	--
Supporting Material:	--
Comment:	--

5.4.3 Legacy Models

The creation of software component descriptions from functional behavior models depends on the functional behavior model. Either it is a legacy model which shall be transferred to an AUTOSAR description, or it already conforms to AUTOSAR behavior model building blocks, and shall be transferred to an AUTOSAR software component description.

5.4.3.1 [TR_ATBM_00007] Creation of an encapsulated atomic software component description from legacy models

Initiator:	DC
Date:	20.01.05, update: 13.05.05
Importance:	High
Requirement:	Creation of an encapsulated atomic software component description from legacy models
Description:	The **converter** should create an encapsulated atomic software component description based on a structured legacy functional behavior model, modeled by a specific BMT. The interfaces from the top level functional behavior model should be used as the interfaces of the `AtomicSoftwareComponentType` (see Figure 25).
Rationale:	--
Use Case:	**[UC_ATBM_00009]**: The user aims to take an existing legacy functional behavior model, which does not necessarily correspond to the AUTOSAR modeling concepts, and attempts to create the description of components. **[UC_ATBM_00010]**: The user aims to convert the legacy functional behavior model to only one atomic software component in combination with one component functional behavior. The structure of the functional behavior model is flattened.
Dependencies:	TR_ATBM_00002, TR_ATBM_00008
Conflicts:	--
Supporting Material:	--
Comment:	As a result of the conversion a software component type is created.

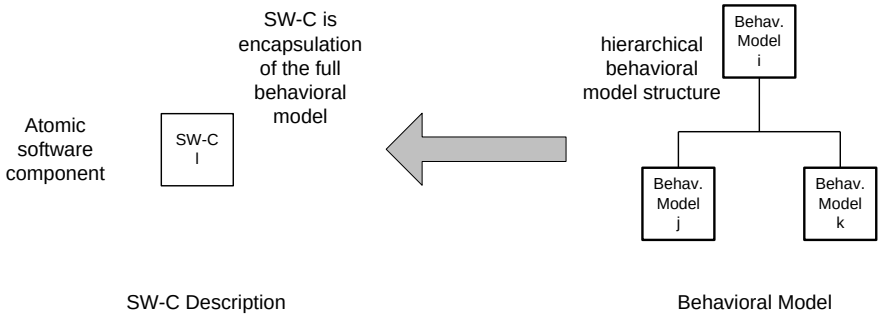

Figure 25: Creation of an encapsulated atomic software component description

5.4.3.2 [TR_ATBM_00008] Creation of hierarchical software component descriptions from legacy models

Initiator:	DC
Date:	20.01.05, update: 13.05.05
Importance:	Medium
Requirement:	Creation of hierarchical software component descriptions from legacy models
Description:	The **converter** should create a hierarchical software component description from a hierarchically decomposed legacy functional behavior model, modeled by a specific BMT (see Figure 26). The **converter** should transform the hierarchical functional behavior model structure into a software component description consisting of compositions and atomic software components. The **converter** should allow the user to specify the depth of the hierarchy, at which a subset of the hierarchy tree needs to be imported.
Rationale:	--
Use Case:	**[UC_ATBM_00009]**: The user aims to take an existing legacy functional behavior model, which does not necessarily correspond to the AUTOSAR modeling concepts, and attempts to create the description of components. **[UC_ATBM_00011]**: The user aims to convert a hierarchical legacy functional behavior model to a composition, i.e. the original structure of the functional behavior model is conserved by converting all structural elements of the functional behavior model to AUTOSAR software components.
Dependencies:	TR_ATBM_00002, TR_ATBM_00007
Conflicts:	--
Supporting Material:	--
Comment:	As a result of the conversion both component types and component prototoypes have to be created. Should the BMT offer a type concepts for functional behavior models, this should be taken into account for the conversion, see also the related explanation on type concepts in section 4.2.2.

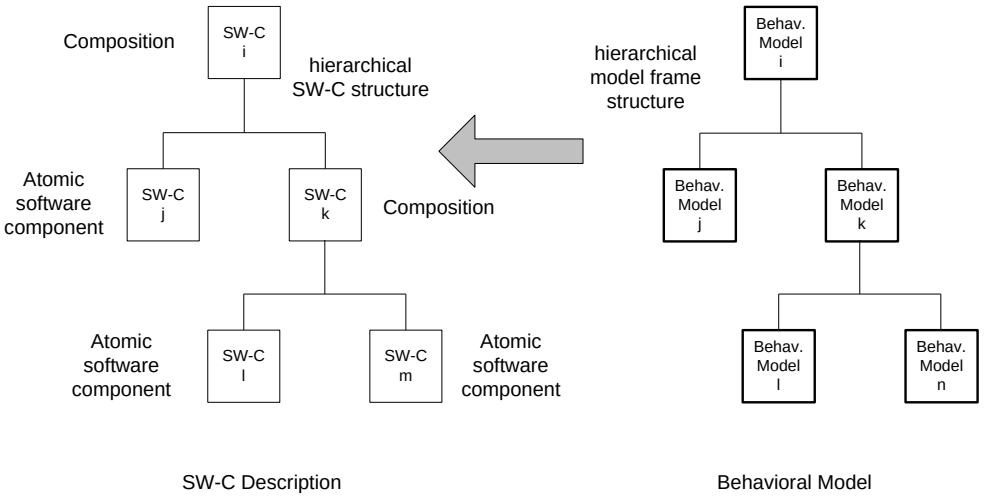

Figure 26: Creation of hierarchical SW-C descriptions

5.4.4 AUTOSAR conform behavior models

5.4.4.1 [TR_ATBM_00028] Creation of software component descriptions from AUTOSAR conform behavior models

Initiator:	DC
Date:	20.01.05, update: 20.07.05
Importance:	Medium
Requirement:	Creation of software component descriptions from AUTOSAR conform behavior models
Description:	The **converter** should convert a functional behavior model that contains BMT-specific model elements, which correspond to AUTOSAR modeling concepts to a structured composition of atomic SWC types.
Rationale:	The converter should be able to conserve the structure for the functional behavior model, since the structure has already been defined in the functional behavior model according to the AUTOSAR modeling concepts.
Use Case:	**[UC_ATBM_00008]**: The user aims to convert a functional behavior model with BMT specific model elements that correspond to AUTOSAR modeling concepts to a structured software component composition, i.e. the original structure of the functional behavior model is conserved by converting all structural elements of the functional behavior model to AUTOSAR software components. [**UC_ATBM_00021**]: The user wants to create an AUTOSAR conform functional behavioral model from the scratch for simulation purposes without basing it on a given SW-C description.
Dependencies:	TR_ATBM_00002, TR_ATBM_00031
Conflicts:	--
Supporting Material:	--
Comment:	--

5.4.5 [TR_ATBM_00031] Creation of runnable description from an AUTOSAR conform behavior model

Initiator:	WP Authoring Tools
Date:	06.09.2005
Importance:	Medium
Requirement:	Creation of runnable description from an AUTOSAR conform behavior model
Description:	The **converter** should create a description of runnables from an AUTOSAR conform functional behavior model, in order to allow the merge of interface description provided by authoring tool with runnable description from BMT.
Rationale:	--
Use Case:	[**UC_ATBM_00019**]: The user wants to have an interface description of software components converted to a functional behavior model frame (see **[UC_ATBM_00001]**). Then he wants to e.g. define the internal behavior (runnables) in the BMT based on simulation experiments. From the resulting functional behavior model a description of the functional behavior shall be converted **[UC_ATBM_00007]** and merged with the interface description of software components.
Dependencies:	TR_ATBM_00002, TR_ATBM_00028
Conflicts:	--
Supporting Material:	--
Comment:	The merging itself is outside the scope of this requirement.

5.5 Creation of the Software Component Environment

5.5.1 [TR_ATBM_00025] Creation of the Software Component Environment Model

Initiator:	DC
Date:	20.07.05
Importance:	Medium
Requirement:	--
Description:	The converter should create a software component environment model based on the AUTOSAR meta model classes, defined in the parts of the AUTOSAR meta model relevant for functional behavior modeling (see chapter 4).
Rationale:	As written in chapter 4 there is a distinction between meta model classes of the software component template which are relevant for functional behavior modeling and classes that are relevant for modeling the software component environment, used to perform simulation experiments. This requirement captures the classes for the second case.
Use Case:	**[UC_ATBM_00001]**: The user aims to have a software component description being converted into a functional behavior model. **[UC_ATBM_00002]**: The user aims to model and simulate the functional behavior of one selected atomic software component in combination with an internal behavior by means of a BMT. **[UC_ATBM_00003]**: The user aims to model and simulate the functional behavior of several atomic software components together in a BMT, in order to do simulation experiments, where the interaction of the functional behavior of different atomic-software components can be evaluated.
Dependencies:	TR_ATBM_00001
Conflicts:	--
Supporting Material:	--
Comment:	This requirement captures the conversion of all the meta classes defined in chapter 4. As a result of the conversion a model frame for the environment model is being created containing information about RTEEvents and ComSpec.

6 Appendix

6.1 References

6.1.1 Normative References

[1] Glossary
AUTOSAR_TR_Glossary.pdf

[2] Methodology
AUTOSAR_MOD_Methodology.pdf

[3] Software Component Template
AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[4] Specification of ECU Resource Template
AUTOSAR_TPS_ECUResourceTemplate.pdf

[5] Specification of System Template
AUTOSAR_TPS_SystemTemplate.pdf

[6] Metamodel
AUTOSAR_MMOD_MetaModel.eap

[7] Template UML Profile and Modeling Guide
AUTOSAR_TemplateModelingGuide.pdf

[8] Template Formalization Guide
AUTOSAR_TemplateFormalizationGuide.pdf

[9] Specification of the Virtual Function Bus
AUTOSAR_EXP_VFB.pdf

[10] Requirements on Interaction with Behavioral Models,
AUTOSAR_RS_InteractionWithBehavioralModels.pdf

[11] Specification of Feature Definition of Authoring Tools
AUTOSAR_FeatureDefinition.pdf

[12] Interoperability of Authoring Tools
AUTOSAR_TR_InteroperabilityOfAutosarTools.pdf

[13] Specification of Graphical Notation
AUTOSAR_TR_GraphicalNotation.pdf

[14] Model Persistence Rules for XML
AUTOSAR_TR_XMLPersistenceRules.pdf

6.1.2 Normative References to External Documents

[15] ASAM-MCD-2MC V2.0 / MSRSW V2.2.0. 2001.
http://www.msr-wg.de/ medoc/download/msrsw/v222/msrsw-tr-intro/msrsw-tr-intro.pdf