Document Title	Specification of Feature Definition of Authoring Tools
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Identification No	203
Document Classification	Auxiliary

Document Version	1.0.3
Document Status	Final
Part of Release	3.0
Revision	0001

Document Change History

Date	Version	Changed by	Change Description
31.10.2007	1.0.3	AUTOSAR Administration	• Document meta information extended • Small layout adaptations made
24.01.2007	1.0.2	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
28.11.2006	1.0.1	AUTOSAR Administration	Legal disclaimer revised
18.11.2005	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents 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 Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, 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

Table of Figures

1 Introduction

This chapter provides a brief overview of the scope and goals of the document.

1.1 Aspects of AUTOSAR Authoring Tools (non-normative)

As a first approximation for a definition, AUTOSAR Authoring Tools cover the following description formats within AUTOSAR:

- Software-Component Descriptions created according to the AUTOSAR Software-Component Template
- ECU Resource Descriptions created according to the AUTOSAR ECU Resource Template
- System Descriptions created according to the AUTOSAR System Template

In particular, AUTOSAR Authoring Tools are required to be able to read, create or modify AUTOSAR descriptions (i.e. the physical representation of AUTOSAR models).

Figure 1-1 sketches the descriptions that can be maintained by AUTOSAR Authoring Tools within the AUTOSAR methodology. 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 cover the behavior implementation. 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 1-1.

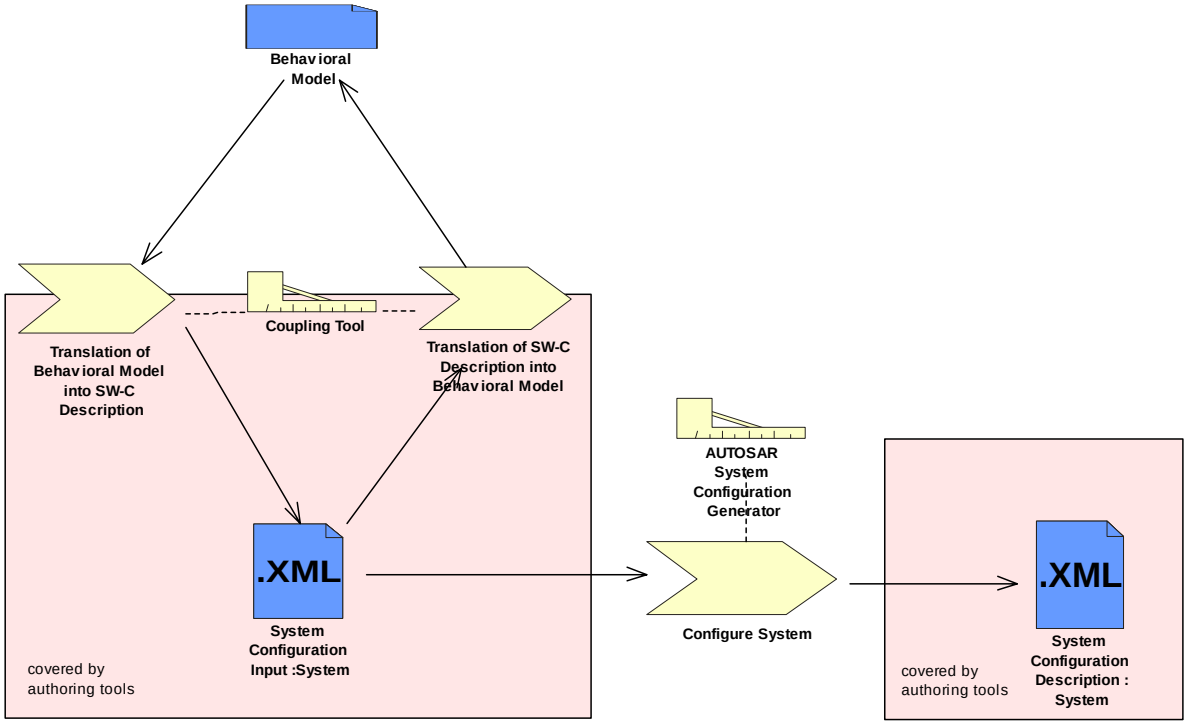

Figure 1-1: Descriptions which can be created and modified by AUTOSAR Authoring Tools

A further aspect of AUTOSAR Authoring Tools is the configuration of the System that (as sketched in Figure 1-1) produces the System Configuration Description as an output. Please note that this task can be carried out manually or (to some extent) automatically. Since the automatic system configuration (although sketched as an "AUTOSAR System Configuration Generator") has not yet been addressed within AUTOSAR, the first approximation to AUTOSAR Authoring Tools as described in this document is focused on the manual creation of a System Description.

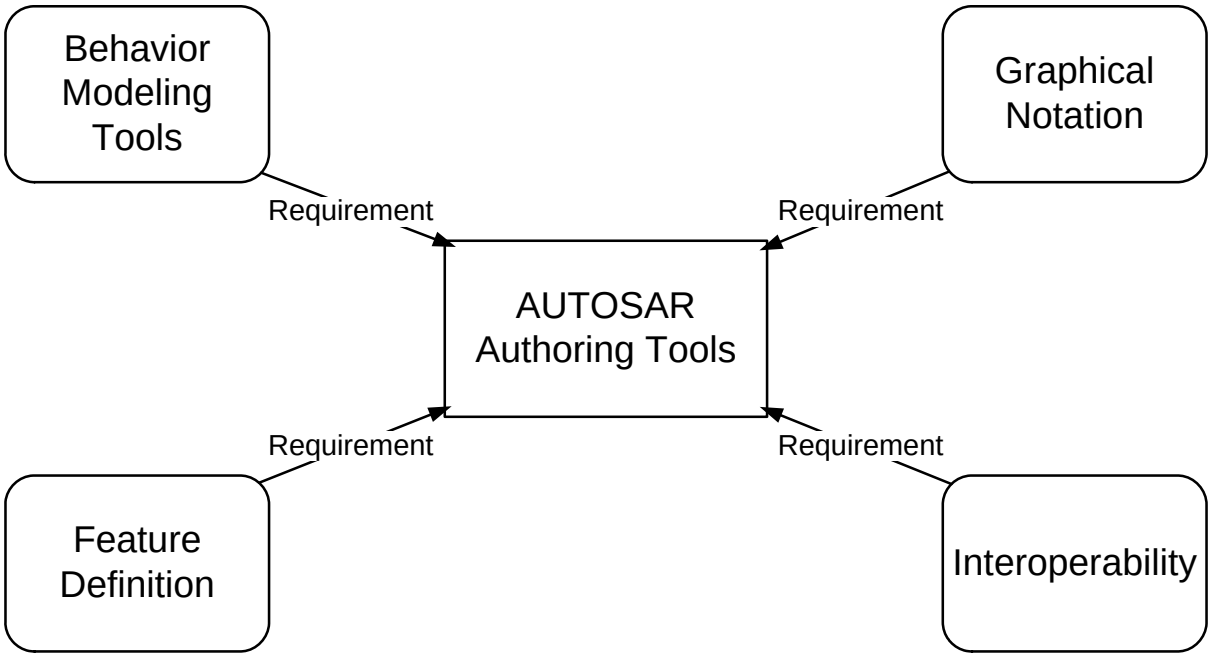

Figure 1-2: Aspects of AUTOSAR Authoring Tools

The description of AUTOSAR Authoring Tools covers several important aspects as depicted by Figure 1-2. Please note that the description of all these aspects results in the formulation of requirements to AUTOSAR Authoring Tools.
Each aspect as depicted in Figure 1-2 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 by Figure 1-2) is available (in a separate document for each aspect):

- **Specification of Interoperability of Authoring Tools** [10]
  The document “Interoperability of AUTOSAR Authoring Tools” emphasizes on issues that might come up when exchanging AUTOSAR models between different tools. After describing some basic concepts of data exchange the document sketches strategies on how these issues can be resolved. Requirements on AUTOSAR Authoring Tools for ensuring interoperability are defined.

- **Specification of Interaction with Behavior Models** [11]
  The document ”AUTOSAR Interaction with Behavioral Models” defines the mapping between AUTOSAR concepts described in the Software-Component Template and concepts in selected behavior modeling tools for some use cases. General requirements for behavior modeling tools are derived from these use cases.

- **Specification of graphical Notation** [12]
  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 properly.
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. The dependency relationships of this document are depicted in Figure 1-3.

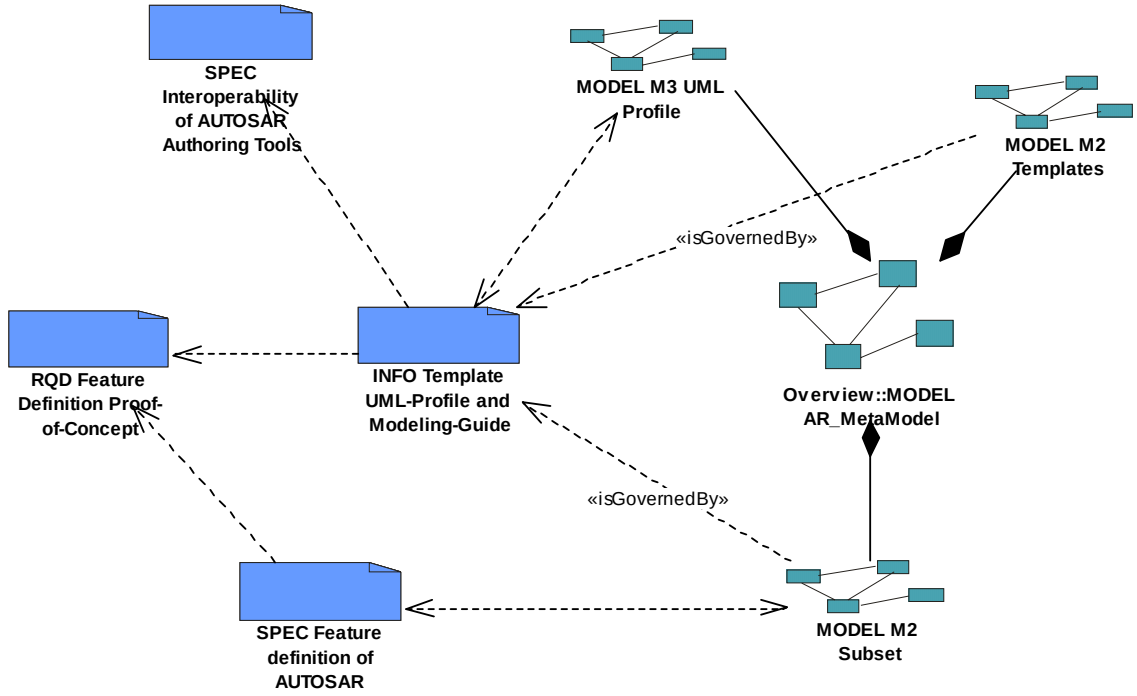

Figure 1-3: Document dependencies

According to Figure 1-3, this document maintains a mutual dependency to the subset package in the AUTOSAR meta-model [9]. Furthermore, it depends on the corresponding requirements document [8].

1.2 Origins and goals (non-normative)

The AUTOSAR initiative has been established to standardize software infrastructure in automotive ECUs as well as corresponding information exchange formats between design tools. The implementation of the concepts created in numerous workgroups is a very ambitious endeavor, especially if it is carried out in a single step.
It is only natural that some of the concepts developed within the AUTOSAR initiative impose a higher risk and/or effort than others. These concepts contribute massively to the overall risk and effort of AUTOSAR implementation.
It is therefore highly recommended to identify potentially critical parts of the AUTOSAR concept and derive a plan for a **stepwise implementation**. In other words: the strategy should be to first implement crucial parts of the concept that are supposed to be of benefit but impose only a limited and well-estimated risk[1] of failure.
This document provides a recommendation for the first step of 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.
The recommendation is based on the definition of **uses cases** identified by (mainly) OEM representatives in the WP. Please note that the first versions of this document were focused on **features**, i.e. the point of view was mainly that of a tool supplier.

[1] Admittedly, the estimated risk of specific features more or less directly corresponds to experiences made in projects where similar concepts have been implemented.

By assessing the features against the specified use cases (i.e. bringing together the different point of views) it will hopefully be possible to identify a stable and reliable collection of features that actually form the subset.
Please note that the result of the assessment could possibly consist of mostly features that have been identified as sort of critical concerning the effort for implementation and especially the risk they inject. In this case additional measures have to be taken to sort this issue out.
Please note further that the resulting subset of the AUTOSAR concept is expected to have mainly an impact on AUTOSAR authoring tools for the supported templates.
On the other hand, decisions made on the level of design admittedly will set preconditions and constraints for e.g. the first implementation of the AUTOSAR basic software. The latter, however, is intentionally left out of scope of this document[2].
The definition of different implementation steps is certainly supposed to consider migration issues, i.e. it must be guaranteed that models created by means of the first implementation step can positively be converted to models used in the second implementation step and so on.
Please note that the appendix of this document (namely chapter 11) discusses possible scenarios that could be applied to develop the feature definition beyond the scope of this document.
The work package covering the specification of the ECU Configuration Template is still in progress. Therefore, particular features related to this work package (e.g. mapping of runnables to tasks) are not yet included into this document.
Therefore, this document is consequently limited to the description of editors and "authoring tools" for carrying out sort of a **structural design**[3] within the scope of AUTOSAR. This includes mainly the design of software-components (including compositions), ECU Resource Descriptions, and System Descriptions.
Aspects of AUTOSAR RTE generation and/or the basic software are intentionally not covered. Perhaps this might be amended when the definition of the AUTOSAR RTE and basic software acquires a stable state.

1.3 Terminology

- **An XML schema** refers to XML validation mechanisms like W3C DTD (“Extensible Markup Language (XML) 1.1, *http://www.w3.org/TR/xml11*”) or W3C XML schema (XML Schema 1.1, *http://www.w3.org/XML/Schema*”)

- **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.

- **AUTOSAR XML description** describes the XML representation of an AUTOSAR model. The AUTOSAR XML description can consist of several fragments. Each fragment must validate successfully against the AUTOSAR XML schema as defined in the AUTOSAR model persistency rules.

[2] The major reason for this observation is the fact that at the time of this writing the specifications of AUTOSAR basic software didn't acquire a stable state.
[3] Maybe it would be a good idea to define this term sooner or later within the scope o AUTOSAR. This requires perhaps a closer look at the methodology. For the moment, the term "structural design" is hopefully intuitive.

- **Authoring tools** are software tools working on any form of AUTOSAR data. The scope of an authoring tool (in terms of AUTOSAR data) is all data that it requires to properly function, especially the data that it can modify.

- **Data inconsistencies** are combinations of elements or attributes of elements that can be described by the modeling language, but models that they are part of are not "valid" in the sense that systems they model would be either impossible to create or they would not work as intended.

1.4 Conformance to the AUTOSAR meta-model

Please note that this version of the document (i.e. 1.0.0) directly corresponds to the AUTOSAR meta-model [9] taken as the basis for AUTOSAR V1.0, i.e. revision 11997. The annotation of the meta-model, however, has been added after the actual meta-model has been frozen for being released as version 1.0.

1.5 Relationship to other documents

This document is to some extent related to the list of non-basic software features. The latter even references each feature defined in this document and provides an individual assessment of each feature with respect to upcoming AUTOSAR releases.
Please note that this document has been started approximately 8 months before the initial creation of the non-basic software feature list. In accordance with the original goals, this document is a bit more restrictive about features than the non-basic software feature list.
After this document has been finalized, all further activities concerning the definition of a feature set will be carried out in the context of the non-basic software feature list.
This document will not be maintained any longer. It is about the definition of a feature set for a **first** implementation of AUTOSAR Authoring Tools. After this task has been achieved (it can safely be assumed that first AUTOSAR Authoring Tools will be available early in 2006), there is no point to keep the document any longer.
Please note, however, that any further evolution of the feature set might be sketched along the possible scenarios discussed in chapter 11.

2 Requirements tracing

Please note that the feature definition itself is more or less comparable to a collection of functional requirements to AUTOSAR Authoring Tools. On the other hand, there are obviously some requirements on this document as well.
These have been defined in a separate requirements document [8]. Table 1 contains a requirement trace matrix that indicates the correspondence between a particular requirement and chapter numbers within this document in which the requirement is addressed.

Requirement	**Satisfied by**
[ATFD001] Define subset	**4**
[ATFD002] Identify features	**5, 6, 7, 8**
[ATFD003] Include key features	**5, 6, 7, 8**
[ATFD004] Limit risk	**5, 6, 7, 8**
[ATFD005] Declare criteria	**4.6**
[ATFD006] Assess priorities	**5, 6, 7, 8**
[ATFD007] Annotate meta-model	**5, 6, 7, 8**
[ATFD008] Support structural validation	**5, 6, 7, 8**

Table 1: Requirements trace matrix

3 Use cases

The purpose of this document is to identify a collection of features for a first implementation of AUTOSAR Authoring Tools. The features, on the other hand, need to be somehow motivated in order to avoid potentially subjective assessments driven by individual authors of this document.
Therefore, the authors of this document came up with the idea to define **features in exchange for use cases**. In other words: use cases are sort of a currency for the creation of this deliverable.
Any contributor could literally “buy” a specific feature by providing a comprehensive use case that clearly demonstrates the necessity to have this feature in a first implementation of AUTOSAR Authoring Tools. As a side-effect, the authors intended to implement sort of a balance between OEM/Tier-1 (who are typical contributors of use cases) on the one hand and tool vendors (who eventually would have to implement the features) on the other.

3.1 Structure of use case description

The description of a use case is started by a short table containing a short description of the use case and a reference to the initiator. Example:

Short description
What are the main characteristics of the use case?
Initiator
Who submitted the use case for the subset description initially?

After the table a detailed description of the use case is provided.

3.2 Description of use cases

Please note that although the definition of use cases to some extent has a normative character (because the estimation of features is derived from it) it is **not** implied that the definition of use cases is exhaustive.

3.2.1 Top-down hierarchical design

Short description
Carry out a hierarchical design of software-components from top-level down and allow for a later refinement (i.e. decide whether a `ComponentPrototype` is actually typed by a `CompositionType` or an `AtomicSoftwareComponentType`) of `ComponentPrototypes`.
Initiator
Uwe Honekamp (Vector)

A `CompositionType` consists of `ComponentPrototypes` typed by specific `ComponentTypes`. Therefore, a designer of a `CompositionType` might want to include several `ComponentPrototypes` typed by a specific `ComponentType`. The creation

of a CompositionType therefore has a large impact on the multiple instantiation of software-components.
It is certainly reasonable to suspect that a creator of a composition does not primarily care whether the instantiated ComponentType represent further CompositionType or else AtomicSoftwareComponentType[4].
Example: an OEM creates a CompositionType and hires a tier-one supplier for the implementation of particular ComponentPrototypes within the CompositionType. It is then up to the supplier to decide whether the software-component in question must be implemented as an AtomicSoftwareComponentType or by means of a further CompositionType.
If, for example, certain parts of the functionality already exist in the form of AtomicSoftwareComponentType there will supposedly be a high motivation to implement the software-component in question by means of a Composition and create further AtomicSoftwareComponentType for the missing functionality.
In other words: the supplier is able to **reuse** existing software-components which perfectly in line with the AUTOSAR goals. Please note that this use case has a large impact on the feature ARSubset0008.
The AUTOSAR Authoring Tool is supposed to provide a “morphing” functionality to support the switch between a CompositionType and an AtomicSoftwareComponentType and vice versa

3.2.2 Interfaces of atomic software-components

Short description
The user wants to specify the ports and interfaces of atomic software-components with their attributes.
Initiator
Thomas Ringler (DC)

This is a general use-case which does not imply the point of time in the workflow where this step is done. In opposite to this the “Top-down hierarchical design” and the “Bottom-up design of CompositionTypes” reflect the workflow.
The specification of atomic software-components and their ports and interfaces includes all aspects of single software-components that are necessary for it to be used in a software system (i.e. to be able to use it as a prototype in the system). Interfaces and software-components can be specified independently from each other.

[4] Please note that the current AUTOSAR meta-model does not allow for a postponement of the decision whether a specific ComponentPrototype is typed by a CompositionType or an AtomicSoftwareComponentType. Hint: ComponentType is an abstract meta-class.

3.2.3 Bottom-up design of CompositionTypes

Short description
Create `CompositionTypes` on the basis of existing `CompositionTypes` and `AtomicSoftwareComponentTypes`. This use case represents a re-usage of software-components.
Initiator
WP1.2

The user wants to bundle and restructure existing software-components. A composition is created based on existing software-components, which then form part of the created composition.

3.2.4 Implementation of software-components by means of behavior models

Short description
Use behavior modeling tools for software-component implementation
Initiator
Uwe Honekamp (Vector)

Having created the structure of a software-component, it is at some point in time necessary to implement the behavior. This could be done on the basis of mere C code or by means of a behavioral model.
The latter is in most cases based on a proprietary document format defined by the vendor of the corresponding behavioral modeling tool. In order to implement the behavior of the software-component the first step is therefore to somehow map the structure of the software-component to the document format of the behavior model.
The initial model created by the transformation process could then be further developed. The correspondence between the software-component and the behavioral model is expressed by means of file references stored in the component implementation.
Another possible scenario would be to take an existing behavioral model and create a software-component according to the structure of the model. This process might require some manual work to sort out structural incompatibilities.
In general, some degrees of freedom have to be resolved concerning the implementation of software-components by means of behavioral models.
For example, a behavioral model could be created either to represent the behavior of an entire component or to represent the behavior of a single runnable (please note that this alternative is currently not really sufficiently supported by the AUTOSAR meta-model). Independent of the selected alternative, the formal definition of the corresponding software-component still remains unchanged.

3.2.5 Specification of Communications

Short description
The user specifies the type and content of data exchanged by software-components.
Initiator
Andreas Graf (BMW), Mark Brörkens (VW / Carmeq)

The communications as specified in the SWC description could be derived from a functional model without regards to the actual physical communications. The technical decision that needs to be made is which data needs to be exchanged between which components and if a certain communication follows the sender/receiver paradigm or the client/server paradigm.

3.2.6 Specification of ECU Resource Descriptions

Short description
The user imports the AUTOSAR conforming descriptions of ECUs from an ECU supplier into the authoring tools in preparation for the mapping step.
Initiator
Andreas Graf (BMW)

For the definition of a topology, the ECUs that are being used have to be specified in an AUTOSAR conforming format within the authoring tools. This covers the data that is necessary to build a topology as well as the data that needs to be taken into account when mapping software to an ECU (e.g. does the binary match the ECU's CPU type)?
The data for a specific ECU will usually be collected by the supplier of that ECU and provided to the customers.
The modeler of a system imports the AUTOSAR conforming descriptions of ECUs from an ECU supplier into the authoring tools in preparation for the mapping step.
To support this kind of integration, a tool shall be able to import ECU Resource Descriptions. To ease the import step, it should be configurable so that a specific location can be specified as a default location for the ECU Resource Descriptions.

3.2.7 Specification of resources for software-component description

Short description
Specification of resources for software-component description
Initiator
Achim Seibertz (FMC)

One of the main criteria for mapping of SW-C onto ECUs is the resource requirement of the complete SW system assigned to a single ECU. In order to estimate the total amount of resources required for a software system, it is necessary to have the information for each of the assigned SW-Cs. The CPU specific resource information is:

Static memory consumption (Code, Global and Static Variables in RAM/ROM and NVRAM)

Dynamic memory consumption (Stack, Heap)

The subset of AUTOSAR SW-C description attributes has to be able to cope with these attributes in order to allow the resource estimation. Based on this functionality some example sub use cases can be derived:

- In-Cycle Action:
    - OEM adds a vehicle level function onto an existing hardware topology and has to find an optimal mapping in terms of resources
    - Supplier has to update the existing software and has to make sure that the change will not result in resource issues
    - Supplier has to replace an ECU and to verify that the HW resources will not be exceeded by the software
- New Vehicle Program:
    - In the system design process a static utilization analysis for the complete system will help to identify resource bottlenecks as well as non-used HW resources.

Please note that the overall use case can not fully be covered within the scope of this document because the final assessment of resource consumption is only feasible in the context of an ECU configuration. On the other hand, the assessment in the scope of the ECU configuration needs preparation by specifying resource claims in the scope of a software-component.
This constraint is mainly caused by the fact that the relevant basic software configuration can only be available in the scope of an ECU configuration for the first time in the workflow. It is obvious that the basic software contributes to a more or less large extend to the overall resource consumption.

3.2.8 Specification of timing resources for software-component description

Short description
Specification of resources for software-component description
Initiator
Achim Seibertz (FMC)

One of the main criteria for mapping of SW-C onto ECUs is the resource requirement of the complete SW system assigned to a single ECU. In order to estimate the total amount of resources required for a software system, it is necessary to have the information for each of the assigned SW-Cs. Concerning this use case, the relevant CPU specific resource information is the worst case execution time.
Again, please note that the overall use case can not fully be covered within the scope of this document because the final assessment of resource consumption is only feasible in the context of an ECU configuration. On the other hand, the assessment in the scope of the ECU configuration needs preparation by specifying resource claims in the scope of a software-component.
This constraint is mainly caused by the fact that the relevant basic software configuration can only be available in the scope of an ECU configuration for the first time in the workflow. It is obvious that the basic software contributes to a more or less large extend to the overall resource consumption.

3.2.9 Consider interaction with basic software

Short description
The user wants to specify the interaction of software-components with the basic software.
Initiator
Thomas Ringler (DC)

Although the actual interaction with basic software modules via the RTE is not in the scope of AUTOSAR Authoring Tools, it is certainly of interest to merely **specify** the interaction with the basic software in terms of ports of a software-component.
It must be possible to distinguish between ports used for communication among application software-components and ports used exclusively for the interaction with the basic software.
The interaction with the basic software supposedly cannot be limited to the usage of sender-receiver communication. Where applicable, it must be possible to use client-server communication as well.

3.2.10 System description of a new system

Short description
The user wants to specify the system description and system constraints of a new system, regarding system communication, system topology and system communication.
Initiator
Thomas Ringler (DC)

The user wants to specify:
- The topology: which ECUs are connected and how they are connected
- System Communication: (Communication Matrix, Frames, Signals, Gateway tables, Communication Protocols)
- Mapping Constraints, what must be mapped, together, what separated etc
- Installs the link between network signals and component interfaces.

3.2.11 System description of an existing system

Short description
The user wants to specify the system description of an existing system like mapping of SW components to ECU and a representation of the communication matrix.
Initiator
Andreas Graf (BMW), Mark Brörkens (VW / Carmeq)

This use case focuses on the description of an existing system. It should be possible to describe:
- The mapping of the data items that are being sent to the actual frames/signal on the underlying bus system. In other words: describe the current communication matrix,

- the network topology (Including bus-systems as CAN, LIN, FlexRay, Unspecified and rough description of ECUs),
- The mapping of software-components to ECUs (not mapping constraints).

3.2.12 Reuse of communication matrices

Short description
The user wants to reuse existing communication matrices
Initiator
Thomas Ringler (DC)

The user wants to use of existing communication matrix among different projects. Usually there exists a basis communication matrix for power-train domain, one basis communication matrix for the body domain, which is used in several projects.

3.2.13 Shipping of an AUTOSAR software-component

Short description
Software-components are exchanged among stakeholders
Initiator
Andreas Graf (BMW)

The term "shipping" is taken from the system team's Technical Overview Document V007, chapter 2.2.4
A fully implemented software-component is passed from one process step to the next. This can involve more than one role in the development process. The shipped component is self-contained to the extent that no modification of the implementation is necessary in the following steps (with the possible exception of compiling source files into object code).
This includes the information in the Use Case "Specification of resources for software-component description", but adds information that is necessary to deploy the system, e.g. dependencies on libraries, supported processing units etc.
The shipped component can be an atomic software-component or a hierarchy of composite types with self-contained atomic software-components as leaves. Examples of this use case are:

- delivery of a component to the OEM by the supplier
- copying of the component from a component repository

3.2.14 Designing a Sensor Actuator Component

Short description
`SensorActuatorSoftwareComponentType` is used to describe the semantics of sensors and actuators within the application software.
Initiator
Andreas Graf (BMW)

To interact with physical systems, some software-components will interface to hardware components. For these software-components it needs to be specified for which

hardware configurations they are designed and how physical values map to logical values by means of a mapping specification. That mapping specification should support very complex mappings.

3.2.15 Top-Down functional development

Short description
Before specifying actual software-components the user wants to carry out a dedicated design step that creates a set of functional blocks which are later mapped to software-components.
Initiator
Matthieu Guillemain (PSA)

When developing a system using the top-down approach the workflow as shown in Figure 3-1 could be performed.
At the stage of mapping the functions onto the Topology, the knowledge of ECU and SWC capabilities is needed. If the OEM intends to use an ECU or a SWC that has already been developed by the supplier, the associated ECU Resource Description or Software-component Description is available through the means of e.g. a central repository.
If not, the OEM makes an assumption on the capabilities of the ECU or SWC that will be used, thus creating a first instance of the ECU Resource Description or Software-component Description.
Once the functions have been mapped onto the Topology, the System Communication Matrix can be generated. The OEM is then able to create the System Configuration Description.
At the stage of defining the ECU, the OEM gives the supplier the System Configuration Description and the specification of behavior of the functions allocated to the ECU (the behavior is usually expressed by means of a model).
The outline of the ECU Resource Description and Software-component Description that may have been created by the OEM in the previous stages, are also communicated to the supplier as input specification for the ECU.

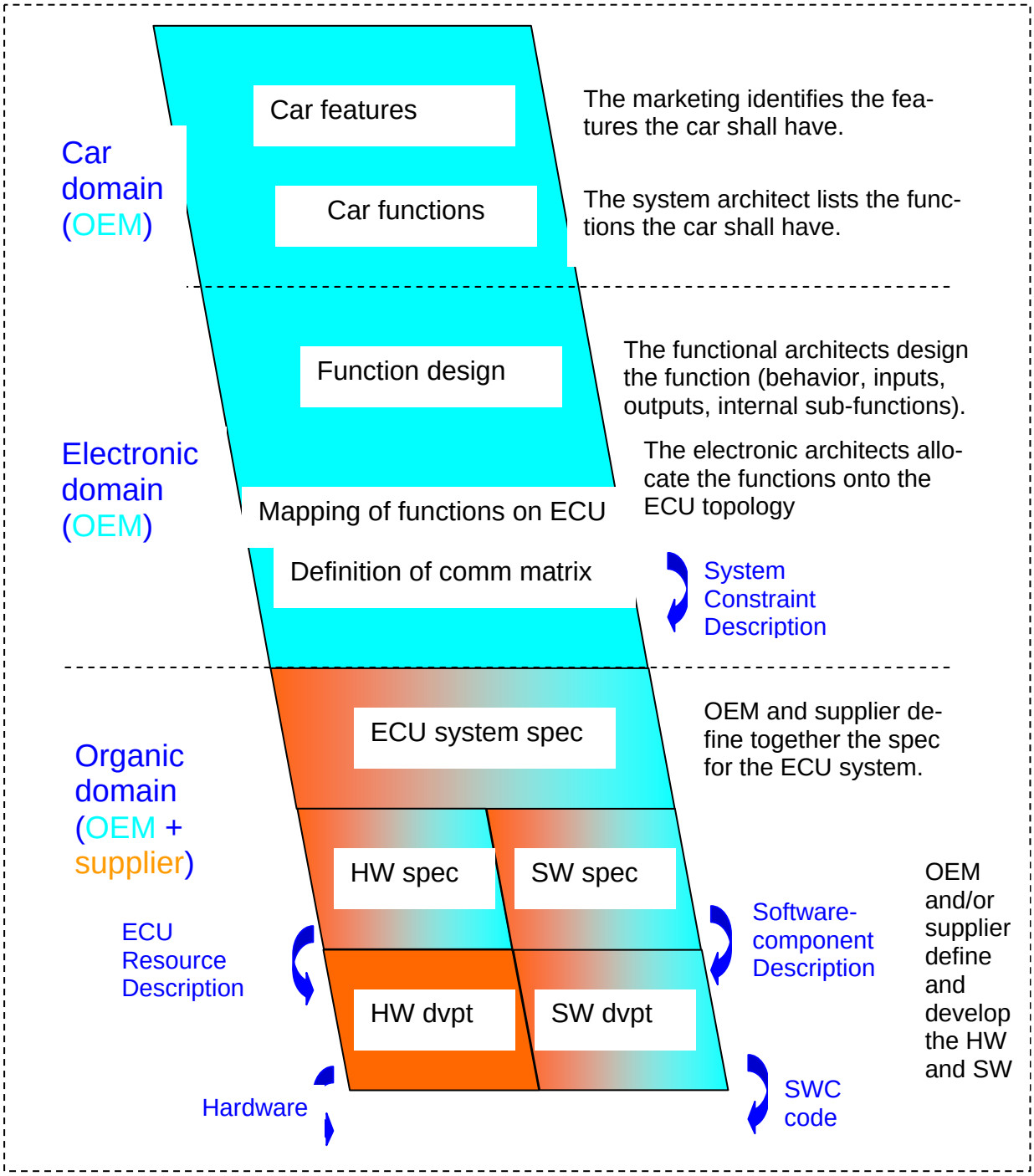

Figure 3-1: Example workflow

The existing development process is chronologically distributed over 3 phases:

1. Car domain activities (features and functions are identified by non-electronic people). These are of OEM's responsibility.
2. electronic domain activities (electronic functions are designed independently of hardware considerations in terms of behavior and interfaces ; those functions, once broken down into sub-functions at a suitable level, are then allocated onto the Topology). These are of OEM's responsibility, with input of the supplier on its catalog of ECU.

3. organic domain activities (each ECU is defined first at a system level by breaking down into hardware or software, then the hardware and software are defined and developed). These are of the supplier's responsibility, with input from the OEM on their interface needs.

The supplier then proceeds with its detailed specification of hardware and software. This is followed by the implementation stage. Once the development and required testing has been completed, the supplier updates the ECU Resource Description and Software-component Descriptions. Those can then be added in the central repository and thus made available to the OEM for its future allocations of functions on Topology.
It can be noted that the AUTOSAR data are first initiated as outlines, then refined and updated by the OEM or the supplier through successive iteration loops as the development of the network and the ECUs progresses.
The AUTOSAR data are exchanged between the OEM and supplier by means of a central repository. It is foreseen that less formal means may also be used to allow ideas to circulate between the OEM and supplier in the early stages of development.
The aforementioned example of a development process shows some points in the workflow where exchange of data is required: This exchange can be done on several levels:

- The initial AUTOSAR model might be automatically generated out of an existing proprietary database or created manually from scratch.
- The incomplete result might be edited by another tool and/or person.
- The AUTOSAR model might be created to a given level of granularity by the OEM and then passed over to another department or to a supplier.
- It might be the case that only a subset of the whole AUTOSAR model is passed to a supplier. The supplier might need to make sure that all required information is available.
- A supplier might be contracted to implement an AUTOSAR component. He needs to return a complete AUTOSAR model for the implemented component. The OEM might need to evaluate if the AUTOSAR model is complete in order to facilitate further processing in the AUTOSAR tool chain.
- At some point of time some AUTOSAR models might need to be integrated / merged. Potential collisions need to be resolved.

3.2.16 Specify FlexRay systems

Short description
The user wants to specify FlexRay Systems.
Initiator
Thomas Ringler (DC)

The user wants to specify FlexRay Systems. Obviously, this requires the specification of FlexRay-specific topology information including star topology. In AUTOSAR, star topologies are obviously modeled on the basis of a hub definition.

3.2.17 Specify CAN systems

Short description
The user wants to specify CAN Systems.
Initiator
Uwe Honekamp (Vector)

The user wants to specify communication relationships among ECU based on the CAN bus system. This requires the support for the modeling of CAN-specific communication hardware and attributes.

3.2.18 Specify LIN systems

Short description
The user wants to specify LIN Systems.
Initiator
Uwe Honekamp (Vector)

The user wants to specify communication relationships among ECU based on the LIN bus system. This requires the support for the modeling of LIN-specific communication hardware and attributes as well as LIN scheduling tables.

3.2.19 Mapping of components to ECUs

Short description
At some stage in the design process it is necessary to map software-components to ECUs and (in a second stage) signals to bus frames
Initiator
Uwe Honekamp (Vector)

The result of the mapping of software-components (technically speaking: `ComponentPrototypes` within the top-level `CompositionType` used to type the `SoftwareComposition` aggregated by the `System`) to ECUs (technically speaking: `ECUInstance`) certainly has a significant impact on the mapping of signals to bus frames.
After having mapped the software-components it becomes obvious which communication path has to be implemented by ECU-internal means and which communication path must be mapped to bus communication.

3.2.20 Data-consistency for communication among RunnableEntities

Short description
Communication among `RunnableEntities` is certainly necessary in a typical software-component. Therefore, it is indispensable that data consistency for exchanged information can be ensured.
Initiator
Uwe Honekamp (Vector)

The need for data consistency has already been addressed in the Software-Component Template. Different concepts for achieving the goal of safe data exchange have been described in the specification of the Software-Component Template.
Although the first implementation of AUTOSAR Authoring Tools will most likely not support the actual implementation of the different mechanism in code (because that would require an RTE implementation) it would perhaps be a good idea to simply be able to specify consistency requirements from the point of view of an individual software-component (i.e. an `AtomicSoftwareComponentType`).

3.2.21 Definition of physical units

Short description
It should be possible to describe physical units for specific `DataElements`
Initiator
Mark Brörkens (VW)

The definition of physical units certainly contributes to the identification and/or definition of conversion formulae in the context of the specification of data semantics. It would therefore be good if the AUTOSAR Authoring Tools were supporting the definition of physical units.

3.2.22 Definition of comments

Short description
It should be possible to add comments to specific model elements within an AUTOSAR model.
Initiator
Mark Brörkens (VW)

Although formally not relevant, the definition of comments and descriptions contributes to the clarity and consistency of an AUTOSAR model. AUTOSAR Authoring Tools should therefore support the specification of descriptive text in the context of elements of AUTOSAR models.
The specification of comments should be possible down to the smallest granularity (e.g. `CompositionType` vs. `DataElementPrototype`) of model elements.

3.3 Estimation of use case priorities

The use cases defined in this document have been formulated mainly from the point of view of the OEMs[5] contributing to the creation of this document. After the definition of use cases has been completed it was agreed that each OEM is supposed to define priorities for the individual use cases.
The result of the priority estimation is listed in Table 2. The anonymous votes of the individual OEMs have been taken as the basis for tentatively defining an overall priority for each use case.
Please note that in some cases the definition of the overall priority is very simple because the priority estimation of individual OEMs matches perfectly to each others. Certainly, there are other cases where the definition of the overall priority is debatable.

	OEM1	OEM2	OEM3	OEM4	Overall
Top-down design	P1	P1	P1	P1	P1
Bottom-up design	P1	P1	P1	P1	P1
Implementation of software-components by means of behavior models	P1	P1	P3	P2	P1
Specification of communication	P1	P1	P1	P1	P1
Specification of ECU Resource Descriptions	P2	P2	P2	P2	P2
Specification of NVRAM and other memory resources for software-components	P1	P2	P3	P1	P2
Specification of timing resources for software-components	P2	P3	P3	P3	P3
Interaction with basic software	P2	P1	P2	P1	P2
System description new	P1	P1	P1	P1	P1
Shipping of an AUTOSAR software-component	P2	P2	P2	P2	P2
System description existing	P1	P1	P1	P1	P1
Reuse of communication matrix	P3	P3	P3	P1	P3
Migration of meta-model	P3	P3	P3	P3	P3
Top-down functional development	P1	P1	P1	P1	P1
Interfaces of software-components	P1	P1	P1	P1	P1
SW-Units	P1	P1	P1	P1	P1
Comments	P1	P1	P1	P1	P1
Specification of FlexRay systems	P1	P1		P2	P1
Specification of CAN systems		P1		P1	P1
Specification of LIN systems		P1		P1	P1
Mapping of software-components		P1		P1	P1
Data consistency among runnables		P2		P2	P2

Table 2: Estimations of use case priorities

[5] Unfortunately, no representatives of Tier-1 suppliers joined the discussion. Therefore, the specific demands of Tier-1 suppliers might or might not be sufficiently addressed within this document.

Obviously, the P1 use cases outnumber the P2 and P3 use cases by far. This numerical relationship is sketched in Figure 3-2.

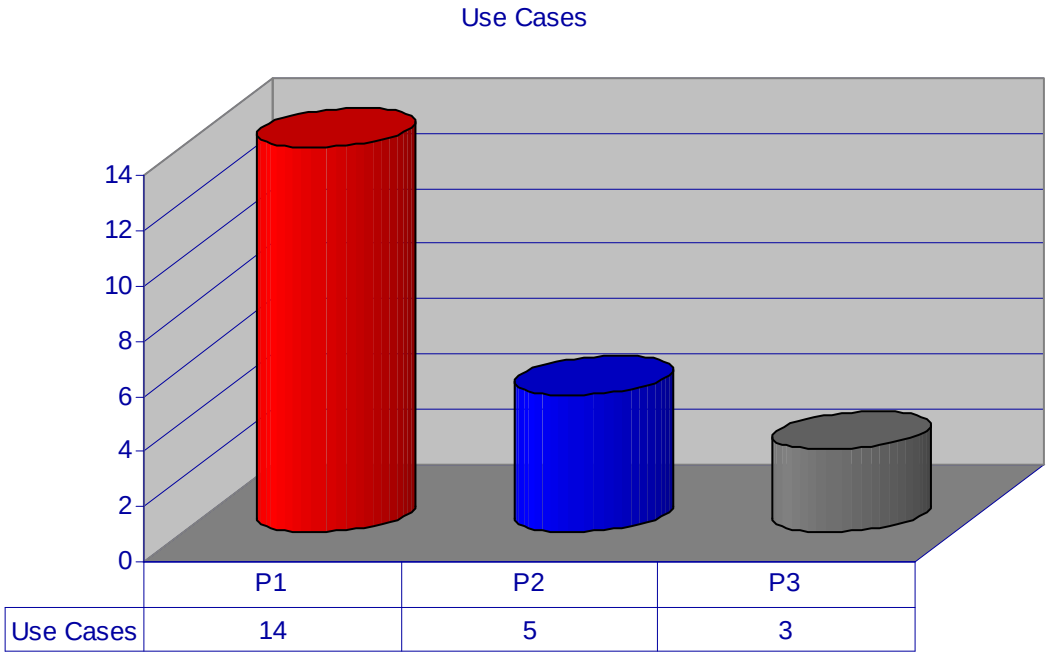

Figure 3-2: use case histogram

3.4 Tracing of use cases to features

This chapter contains a matrix that allows for a tracing of the mapping of use cases to features (the latter are described in chapters 5, 6, 7, and 8). By this means it should easily be possible to verify the coverage of use cases by features and vice versa.
A correspondence between a specific use case and a particular feature are indicated by a capital x. Empty fields indicate that the correspondence is either weak or non-existent.

	Top-down design	Bottom-up design	Behavior models	communication	ECU resource	Sensor-actuator SWC	SW-C resources	Shipping	Interaction with BSW	System Description new	System Description existing	Reuse of Comm. Matrix	Top-Down Functional Dev.	Interfaces of SW-C	Physical units	Comments	FlexRay	Component mapping	CAN	LIN	Data-consistency
ARSubset0001																					
ARSubset0002																					
ARSubset0003				X			X	X		X											
ARSubset0004								X					X								
ARSubset0005	X		X	X				X					X	X							

	Top-down design	Bottom-up design	Behavior models	communication	ECU resource	Sensor-actuator SWC	SW-C resources	Shipping	Interaction with BSW	System Description new	System Description existing	Reuse of Comm. Matrix	Top-Down Functional Dev.	Interfaces of SW-C	Physical units	Comments	FlexRay	Component mapping	CAN	LIN	Data-consistency
ARSubset0006	X	X	X	X		X	X	X		X			X								
ARSubset0007	X	X						X		X			X								
ARSubset0008	X	X	X					X		X											
ARSubset0009	X	X		X																	
ARSubset0010	X	X		X																	
ARSubset0011																					
ARSubset0012									X												
ARSubset0013																					
ARSubset0014																					
ARSubset0015				X				X													
ARSubset0016							X														
ARSubset0017						X															
ARSubset0018			X					X													
ARSubset0019				X																	
ARSubset0020				X																	
ARSubset0021																					X
ARSubset0022																					
ARSubset0023					X					X											
ARSubset0024					X																
ARSubset0025					X																
ARSubset0026					X																
ARSubset0027					X																
ARSubset0028																					
ARSubset0029					X																
ARSubset0030																					
ARSubset0031										X	X						X				
ARSubset0032																		X			
ARSubset0033																		X			
ARSubset0034																		X			
ARSubset0035										X	X							X			
ARSubset0036																					
ARSubset0037																					
ARSubset0038										X											
ARSubset0039																					
ARSubset0040																					

	Top-down design	Bottom-up design	Behavior models	communication	ECU resource	Sensor-actuator SWC	SW-C resources	Shipping	Interaction with BSW	System Description new	System Description exist-ing	Reuse of Comm. Matrix	Top-Down Functional Dev.	Interfaces of SW-C	Physical units	Comments	FlexRay	Component mapping	CAN	LIN	Data-consistency
ARSubset0041										X	X										
ARSubset0042										X	X										
ARSubset0043																					
ARSubset0044																					
ARSubset0045			X																		
ARSubset0046																					
ARSubset0047																					
ARSubset0048																					
ARSubset0049							X														
ARSubset0050																					
ARSubset0051			X					X													
ARSubset0052																					
ARSubset0053																					
ARSubset0054																			X		
ARSubset0055										X	X						X				
ARSubset0056										X	X									X	
ARSubset0057																					X
ARSubset0058																					
ARSubset0059																					
ARSubset0060										X	X	X					X		X	X	
ARSubset0061																	X		X		
ARSubset0062															X						
ARSubset0063																X					

Table 3: Tracing of use cases and features

4 AUTOSAR feature definition

4.1 Goals

The motivation for creating a subset of the AUTOSAR methodology for a first implementation of the latter has been developed in a meeting (the first Premium Member Conference) between members of the AUTOSAR PL team and tool vendor representatives. In this meeting, the evaluation (in terms of tool interoperability) of the methodology has been identified as the major objective of a first implementation of the AUTOSAR methodology.
One of the major observations in this meeting was the assumption that particular features of the AUTOSAR methodology are estimated to cause a significant effort and risk for the implementation[6].
Therefore, the participants agreed to define a subset of the AUTOSAR methodology that should provide enough features to allow for a serious evaluation of the implementation while at the same time concentrate and consequently limit on features that can be implemented within a given limit of effort and risk.
Please note that it is more or less likely that the first implementation eventually requires a further iteration of the methodology for improving its stability and fitness for purpose. It is therefore **neither intended nor advised** to use the first implementation of AUTOSAR Authoring Tools for carrying out series production projects.

4.2 Scope

Although generally referred to as "the subset", the scope of this document is **not** on the entire AUTOSAR methodology. The purpose of the subset is merely to provide a **basis for the implementation of Authoring Tools** based on AUTOSAR templates, in particular the Software-Component Template, the ECU Resource Template, and the System Template.
The AUTOSAR basic software modules as well as aspects of RTE generation are intentionally kept out of this subset definition although the subset admittedly will have one or the other concrete impact on the fist implementation of the basic software and the RTE.

4.3 What is a feature?

The term "feature" is (although it is difficult to find a formal definition of the term) commonly used in the software tool community to describe characteristics (in terms of functionality) of the software. A feature can be described precisely, i.e. it is possible to define what is in the scope of a feature and what not.
In the case of an AUTOSAR authoring tool, a feature is represented by one or many meta-classes and their attributes in the AUTOSAR meta-model. Features are used to implement use cases such that a single use case requires one or more features for implementation.
The scope of individual features can be fine-grained or coarse-grained; it would even be possible to define a hierarchical structure of features describing the functionality of a particular tool (or, in the case of AUTOSAR, a family of tools).

[6] "20% of the features cause 80% of the effort"

4.4 Representation in the meta-model (non-normative)

The definition of a formalization strategy for the representation of the subset in the AUTOSAR meta-model is within the scope of the meta-modeling team. This chapter is supposed to merely give an overview about the strategy that would favorably be used for the annotation.
The first results of the meta-modeling team suggest to define a separate package "Authoring Tools V1" and then create references with the <<import>> stereotype from the package to all meta-classes that are supposed to be a member of the subset.
For each feature, one or more separate diagram display(s) the meta-classes which are required for the support of the feature in question. Aggregations and associations between imported meta-classes are imported without additional effort for meta-model annotation, i.e. it is not necessary to import the associations as well.
In other words: all meta-classes that are referenced by the package "Authoring Tools V1" are member of the subset for the first implementation of AUTOSAR Authoring Tools.
Additional constraints can further limit the subset. For example, the subset allows for the existence of multiple top level packages but on the same time does **not** allow for the existence of sub-packages (see Figure 5-4 for a comprehensive example).
However, the membership of features in the subset is certainly not only limited to meta-classes. It could as well be possible that particular features would be represented by individual attributes within a specific meta-class. This case must also be supported by the formalization strategy.
The current approach is to add additional constraints (that deal with the handling of the respective attributes) to the package. This case can be studied in Figure 6-1.
Meta-model annotations have been created mainly for the P1 features because they will certainly be considered in the subset. The P2 features, on the other hand, are subjects to individual assessment.
Therefore, a second package "Authoring Tools P2 features" has been introduced to annotate all meta-classes that are relevant for P2 features. Please note that diagrams of this package contain references from the "Authoring Tools V1" package as well. For a comprehensive example of this pattern please refer to Figure 5-1.
However, a meta-class shall only be referenced by one of the packages. If it is already referenced by "Authoring Tools V1" there is no need to create an additional relationship to "Authoring Tools P2 features".
Finally, the annotation of each P2 feature that will be promoted to a member of the subset will be changed: the origin of the import-relationship will be switched from "Authoring Tools P2 features" to "Authoring Tools V1".
In some cases meta-classes are only referenced by a mere dependency relation (again, please refer to Figure 5-1 for a meaningful example). This annotation pattern is used to annotate a base-class that is not actually required by itself but must be considered in any case because any derived meta-class is actually a member of the subset.

4.5 Structure of feature description

The assessment results of each feature are summarized in a specifically designed table (an example is printed below). For being able to unambiguously reference particular features it is necessary to assign a unique ID to each feature. Example:

[ARSubset4242] Feature title

Short description				
What are the main characteristics of the feature?				
Initiator				
Who submitted the feature for the subset description initially?				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
References to features that must be implemented as a prerequisite are listed here.	This is a list of references to features that depend on this one.	low/ medium/ high	low/ medium/ high	P1/P2/P3

Priority values represent the result of merit estimation from the feature point of view; they have the following meaning:

- **P1**: *essential*, must be implemented in any case[7] during the first implementation step because the authoring tool could not reasonably be used without the corresponding feature.
- **P2**: *beneficial*, feature contributes to the feature set of the first implementation step. This mainly applies for features that have been identified on the one hand as important but on the other hand an authoring tool could be used sensibly without the feature.
- **P3**: *nice to have*, but does not contribute essential functionality. This category applies for features that can be derived from the meta-model but for which the interest of stakeholders did not exceed a certain threshold.

The risk for a particular feature is rated as “low”, if it fulfils all of the following characteristics:

- Stable concept, i.e. no more changes are expected to occur
- High “maturity of the underlying technology”, i.e. a lot of engineering know-how is available how to deal with the feature

A feature is rated with a high risk on the other hand, if it represents at least one of the following characteristics

- Not yet a stable concept
- New field of technology

In addition, the background of the feature is shortly outlined followed by (where applicable a discussion of open issues and potential caveats concerning the implementation of the feature.

The (admittedly) gross estimation of implementation effort boils down to the following metric:

- **Low**: feature most likely can be implemented, tested, and documented within less than a single week.
- **Medium**: feature supposedly can be implemented, tested, and documented within more than one and less than 3 weeks.
- **High**: feature presumably requires more than 3 weeks for implementation, testing, and documentation.

[7] Exception: features that would be excluded from the subset fort he sake of consolidation with the "overall non-basic software feature list".

4.6 Criteria for membership in the subset

Obviously, the understanding of the criteria that lead to the introduction of one feature to the final subset definition and the rejection of another shall be as precise as possible.
The conclusion of the authors of this document is that the membership of particular features in the AUTOSAR subset should mainly be driven by the description of relevant use cases. These are discussed in chapter 3.
Estimations about effort and risk, on the other hand, represent gross figures of certainly limited accuracy. This must be considered for the final evaluation of features concerning the membership in the subset.
On top of that, a P1 feature derived from a use case that bears high effort and risk shall anyway be given a second thought in order to avoid a potential complexity problem.

5 Common patterns

Common patterns are specific aspects of the AUTOSAR meta-model that are shared among all AUTOSAR templates.

5.1 [ARSubset0001] Properties

Short description				
Properties are used to apply design-time constraints on model structure.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	high	P3

Properties can be used to parameterize the model at design time. The existence of particular model elements on the M1 modeling level depends on the result of the evaluation of a property condition defined on the M2 modeling level.
Properties therefore have an immediate impact on the model structure. Property expressions can be evaluated at any time. Thus, the structure of an existing model might be changed as the consequence of a property evaluation.
This requires tools to immediately recognize changes of the model structures according to property evaluation. Although considered as a useful feature in the long-term this feature is not considered as crucial for a first implementation of AUTOSAR Authoring Tools.

5.2 [ARSubset0002] Grouping

Short description				
The grouping concept is used to categorize AUTOSAR model elements.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P2

The grouping concept has been introduced to provide some means for structuring and categorizing of large models. Groups may reference arbitrary identifiable model elements, even other groups. Model elements can legally be a member of different groups.

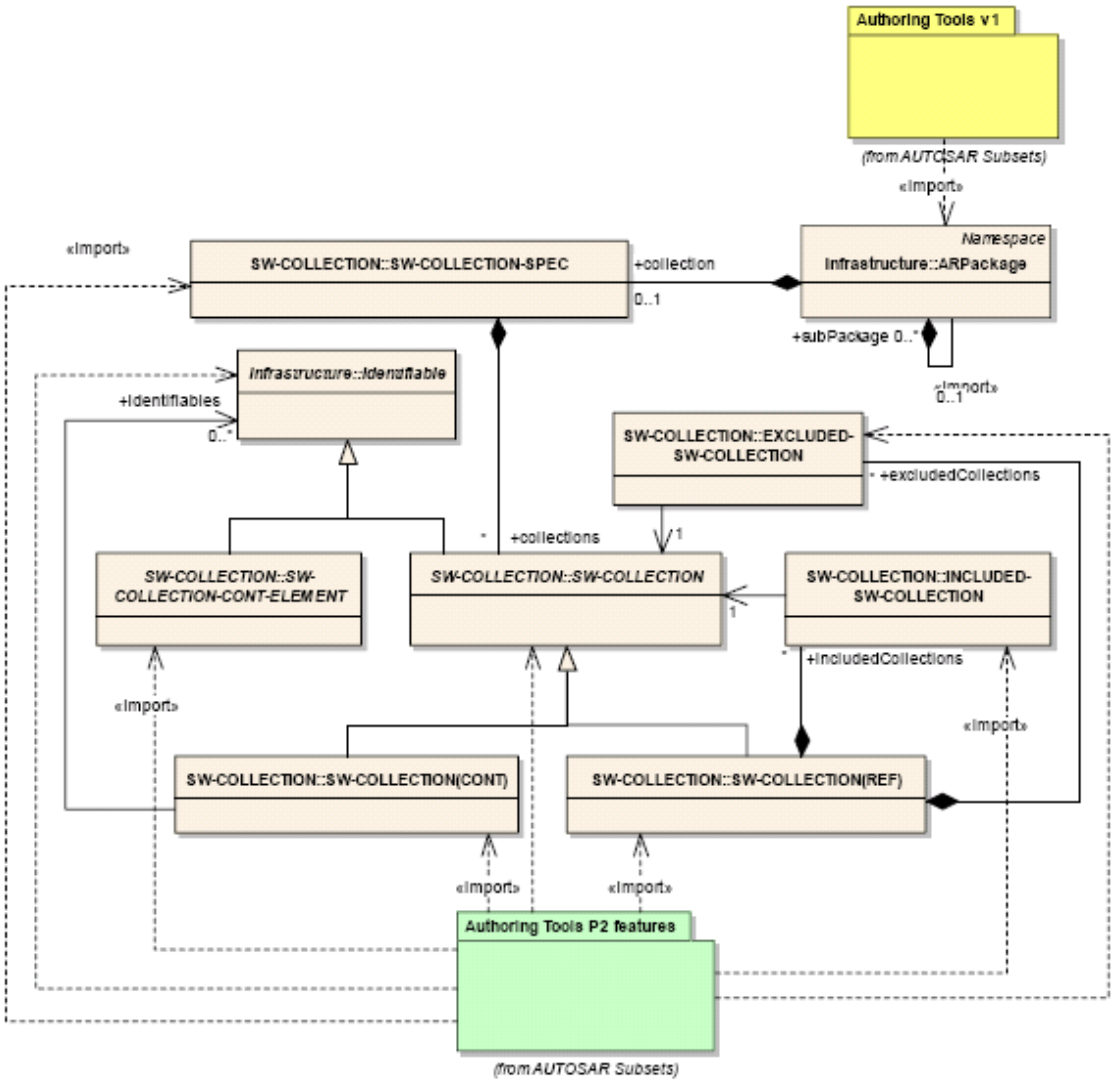

Figure 5-1: Annotation of the meta-model for the support of grouping

The representation of the grouping concept in the meta-model is mainly implemented by means of modeling elements stemming from MSR (please consult Figure 5-1 for more details), in particular the SW-COLLECTION.

5.3 [ARSubset0003] Simple data types

Short description				
Data types for model elements can be defined independent from the usage in the model.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	low	low	P1

The definition and the later usage of simple data types (integer, real, Boolean, etc.) are essential for supporting the sensible utilization of AUTOSAR templates.
As depicted by Figure 5-2, the supported simple data types are `RealType`, `IntegerType`, `OpaqueType`, and `BooleanType`. These types are derived from the meta-class `PrimitiveType`.
Although being derived from the meta-class `PrimitiveType` as well, `CharType` and `StringType` are not considered. The more or less obvious reason for this decision is discussed in ARSubset0048.

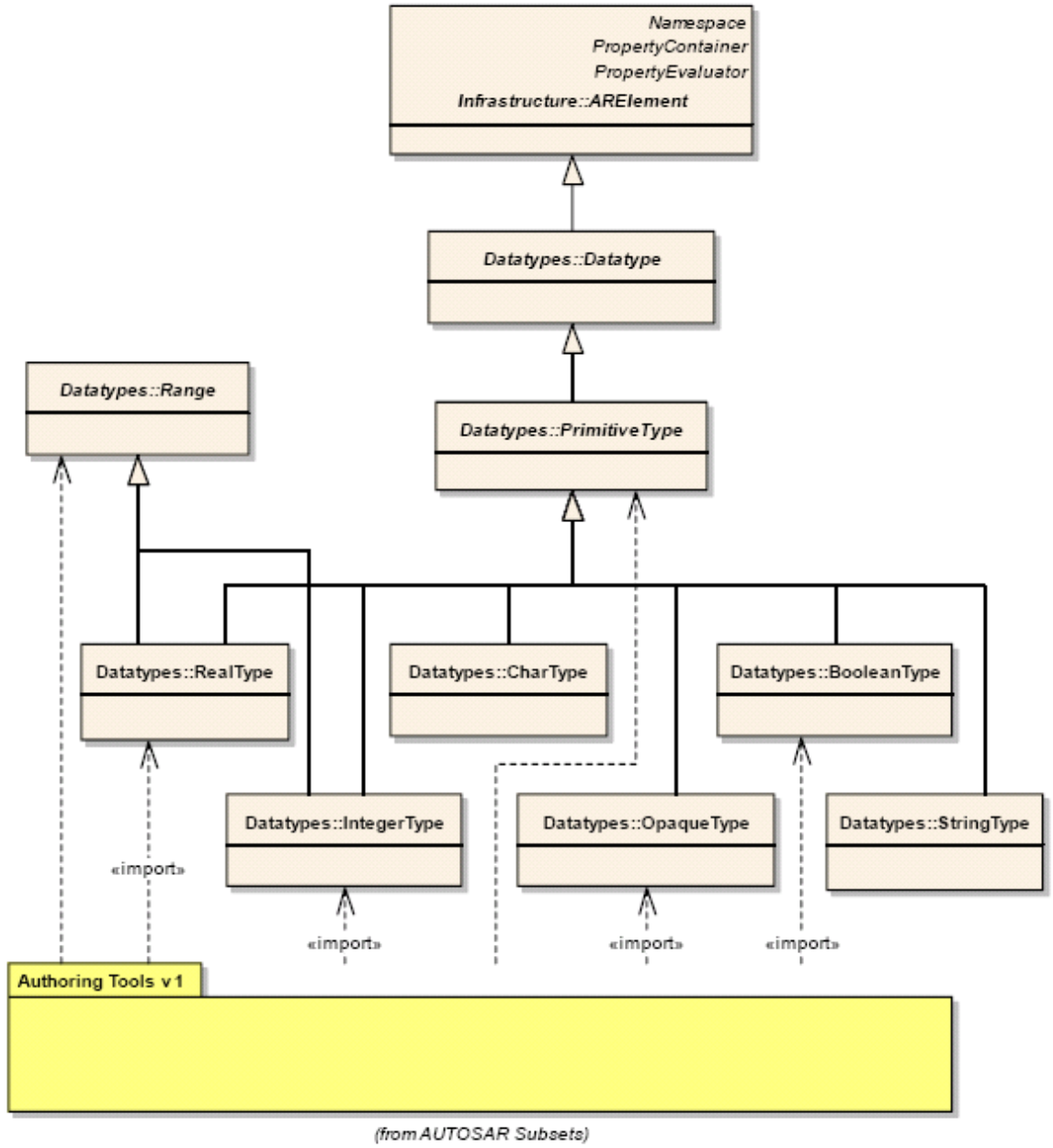

Figure 5-2: Annotation of the meta-model for the support of simple data types

5.4 [ARSubset0046] Complex data types

Short description				
Data types for model elements can be defined independent from the usage in the model.				
Initiator				
Zoltan Matyus (VCT)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	high	high	P2

The definition of `CompositeTypes` (i.e. either `RecordType` or `ArrayType`, please refer to Figure 5-3 for more details) for `DataElements` must be observed carefully. A `DataElement` that exceeds a certain size can not be mapped to a single frame of an underlying bus system. It could only be transmitted by the help of a dedicated transport protocol.

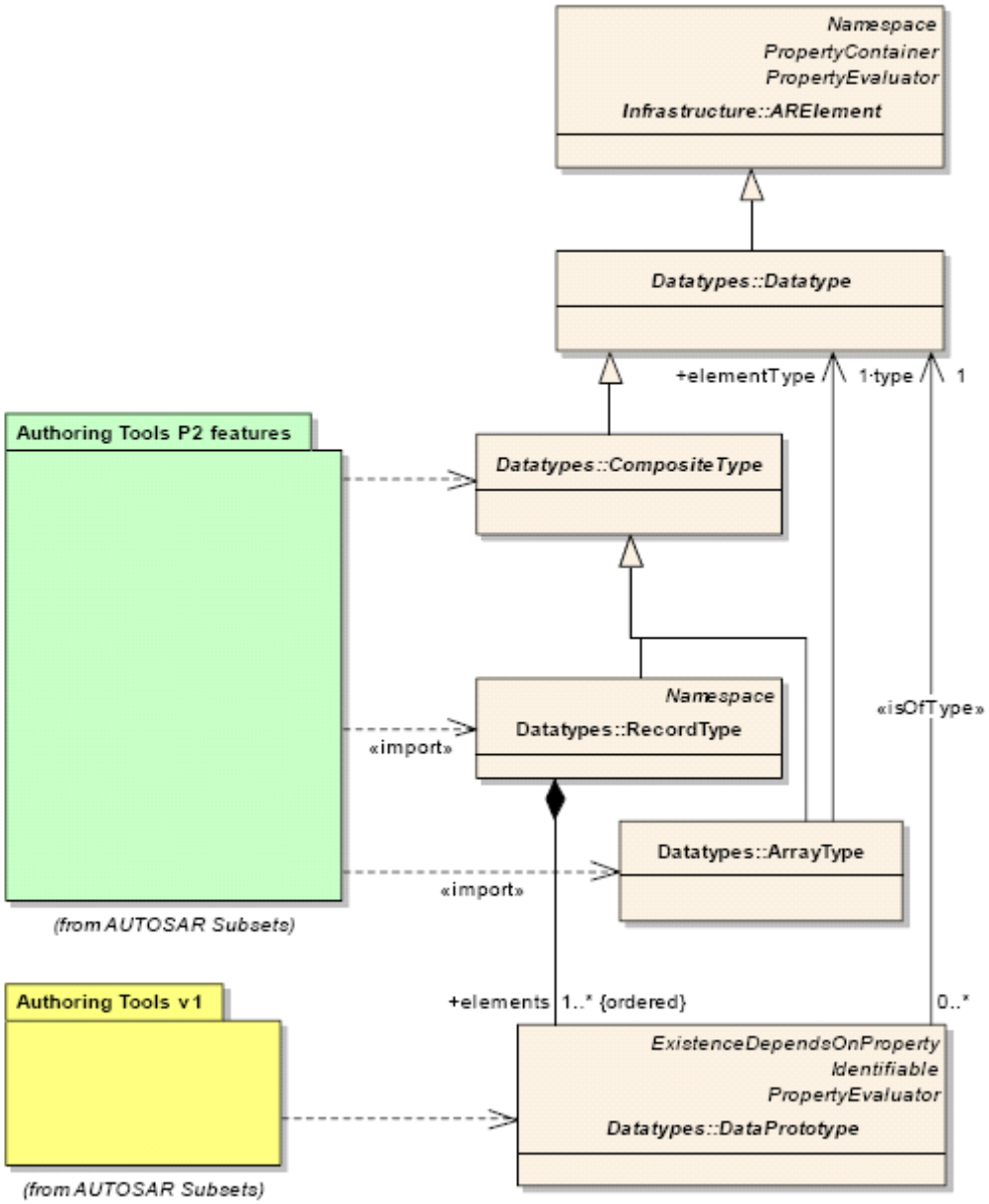

Figure 5-3: Annotation of the meta-model for the support of complex data types

The latter is not supported by AUTOSAR [7]. `DataElements` of `CompositeType` could therefore **not** be transmitted over a communication bus. This is certainly a serious limitation even for the design level.
Please note that this issue is also addressed under the ID "Restr_AR1.0_00001" in [7], i.e. the document proposes to limit the length of a PDU to the payload length of a single communication frame.
Please note further that the RTE SRS explicitly contains a requirement (RTE00091) to support marshalling of complex data types. It is not yet clear, however, whether this requirement will actually be considered in the SWS.
Decision: this feature is not part of the feature set for Authoring Tools V1.

5.5 [ARSubset0048] CharType and StringType

Short description				
The length of string is unlimited and strings can potentially be UTF-8 encoded. The same applies to a character				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P3

According to the meta-model, the `StringType` and `CharType` are primitive types. The length of a string, on the other hand, is not limited. The unlimited length basically leads to the same problems as described for complex data types (ARSubset0046).
On top of that, an upfront determination of the length of a string and a character is not possible if the string or character is encoded according to the UTF-8 standard. This is not actually critical but requires the allocation of more space in a communication frame than actually needed in most cases.

5.6 [ARSubset0004] Package

Short description				
Packages can be used for model structuring. As opposed to grouping, packages provide namespaces for the model elements contained.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P2

Although the package concept on the first view could be considered a nice-to-have feature, it is to some extent essential for the AUTOSAR concept.
For example, naming collisions could be avoided by using packages. According to the definition of the Software-Component Template, at least a single (top-level) package is required in any case. This could, however, be introduced implicitly without necessary having to make the user aware of it.

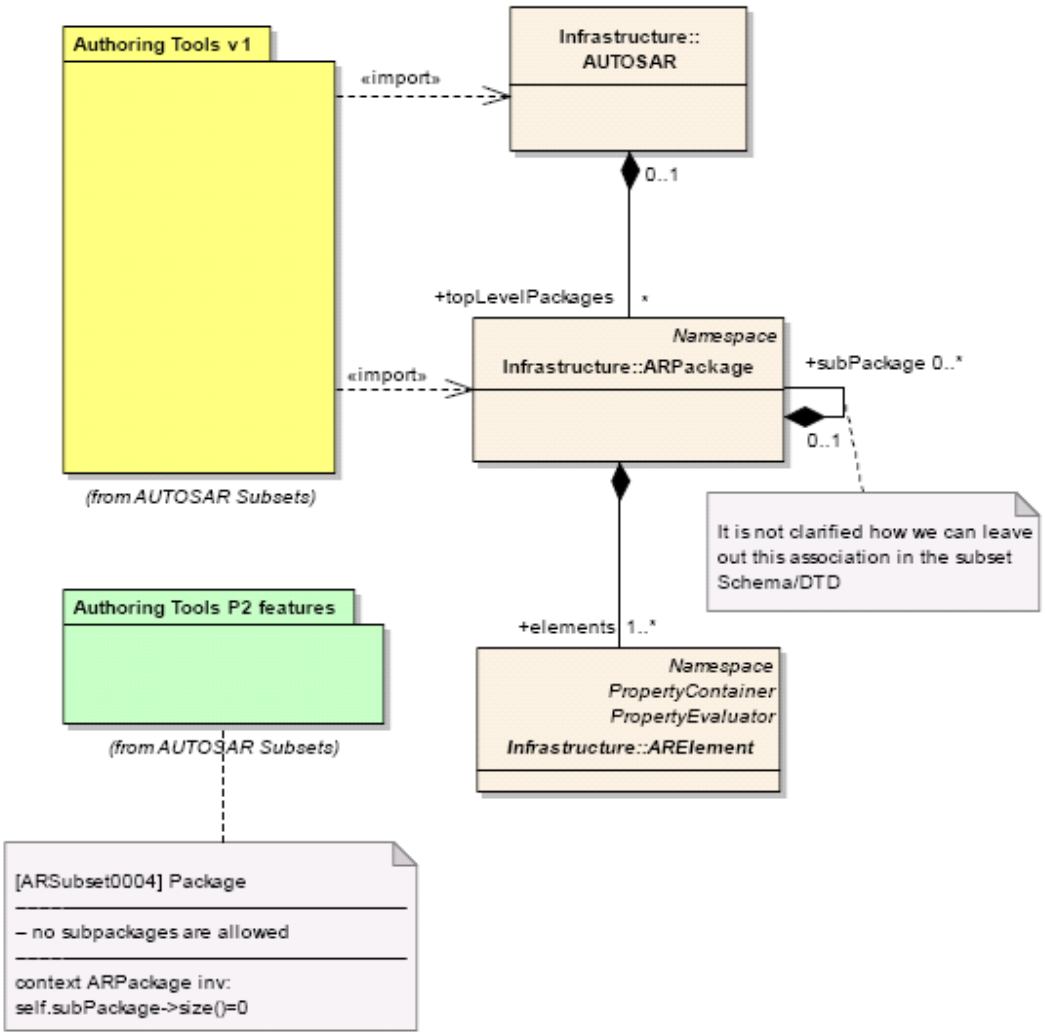

Figure 5-4: Annotation of the meta-model for the support of packages

Perhaps a fixed set of packages (e.g. unlimited number of top-level packages, but no hierarchies of packages) could be defined for the first implementation of Authoring Tools. Later, it would be possible to support arbitrary package structures.
As depicted in Figure 5-4, the annotation of the meta-model is combined with a constraint that limits the package concept to the top level: it is not allowed to create sub-packages.
Please note that (as mentioned in Figure 5-4) it is not yet clarified how the aggregation of packages for creating sub-packages can technically be left out of the creation of a DTD/XML Schema.
Please note further that the package hierarchy and the hierarchy defined by `CompositionTypes` are completely orthogonal. In other words: A CompositionType will always be stored in a single package while the `ComponentTypes` used to type `ComponentPrototypes` can be located anywhere in the package hierarchy.

6 Assessment of Software-Component Template

This chapter contains a discussion of relevant features of the AUTOSAR Software-Component Template.

6.1 [ARSubset0005] Interface

Short description				
`PortInterfaces` define a reusable description of the information exchanged among software-components.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
	ARSubset0006, ARSubset0012, ARSubset0015, ARSubset0050	high	high	P1

`PortInterfaces` are an important part of the AUTOSAR concept. It is (despite the risk imposed by this feature) not an option to leave them out of the first implementation step because a later migration to a meta-model implementation with `PortInterfaces` would certainly be too difficult.

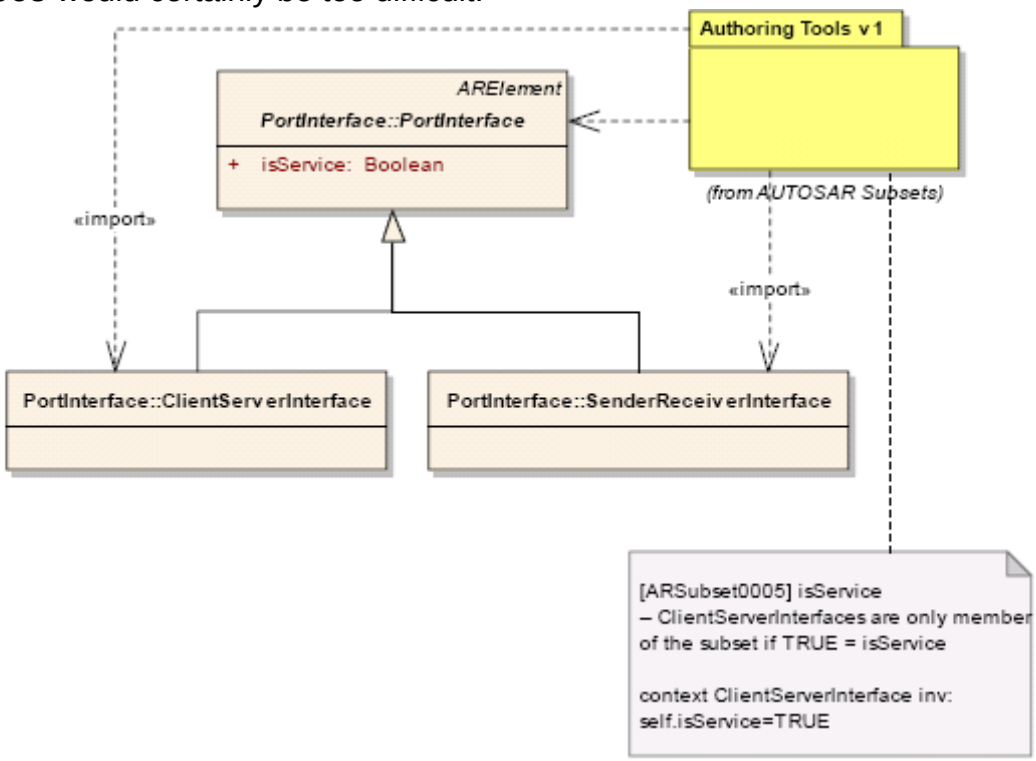

Figure 6-1: Annotation of the meta-model for the support of interfaces

As depicted by Figure 6-1, both the `ClientServerInterface` and the`SenderReceiverInterface` are supported. In the first implementation of AUTOSAR, the

former can only be used as an AUTOSAR service, i.e. the attribute `isService` must always be set to TRUE.

6.2 [ARSubset0006] Software-component

Short description				
Software-components allow for the definition of self-contained application functionality.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0005	ARSubset0007, ARSubset0008, ARSubset0012, ARSubset0014, ARSubset0016, ARSubset0017, ARSubset0018, ARSubset0045, ARSubset0047	high	high	P1

This is - without any question – a really essential part of the concept; there is no option to leave the concept of software-components out of the first implementation of AUTOSAR.
Nevertheless, some potentially important aspects (e.g. support for diagnostics) of software-components have not been properly resolved in the lifetime of the responsible AUTOSAR WP. Therefore, it is likely that the description of software-components will evolve in the future.
Obviously, the risk of software-component evolution is difficult to estimate without an impression about the add-on features that will have to be discussed in the future. A crucial part of the risk is to ensure the ability to migrate existing software-components to later versions of the AUTOSAR Software-Component Template.

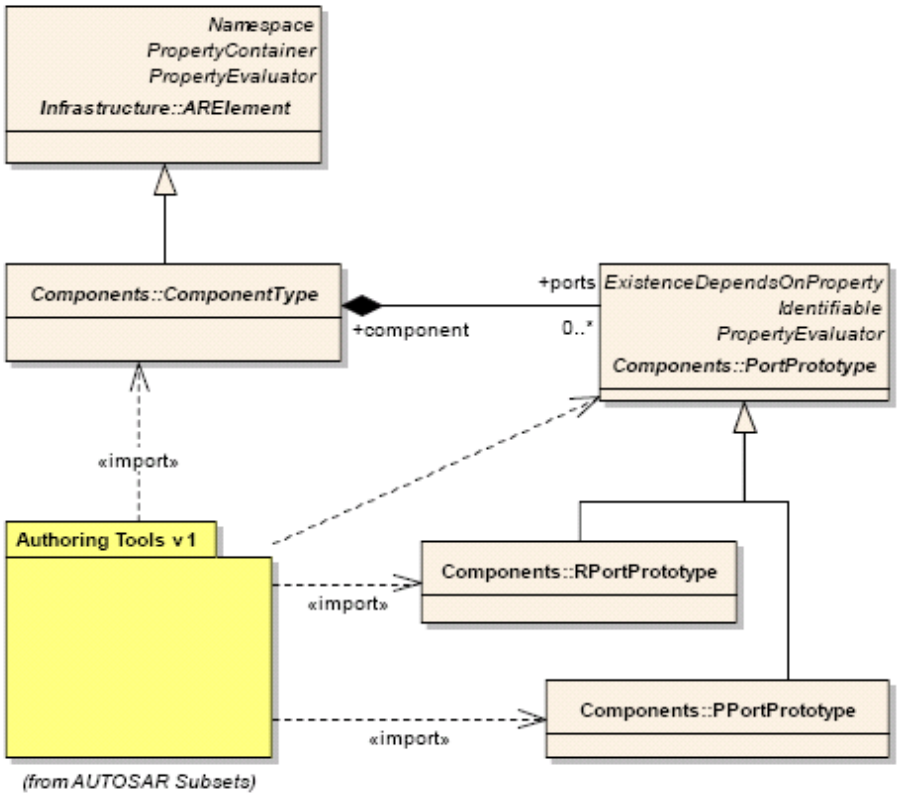

Figure 6-2: Annotation of the meta-model for the support of software-components

The necessary annotation of the meta-model, however, is quite simple (please consult Figure 6-2 for more details): relevant meta-classes are mainly `ComponentType` and `PortPrototype` as well as its subclasses `RPortPrototype` and `PPortPrototype`.

6.3 [ARSubset0007] Composition

Short description				
`CompositionType` allows for the structuring of software-components (or other compositions) to a higher hierarchical level.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006	ARSubset0008, ARSubset0009	low	high	P1

`CompositionType` is essential (and thus given a P1) for the AUTOSAR concept because it provides the basis for structuring the software configuration of a vehicle (which is in fact represented by the top-level composition).
Nevertheless, `CompositionType` is a mere architectural means that has no effect on the level of the VFB. In particular, a `CompositionType` does not add any new

functionality to what is already provided by the `ComponentPrototypes` it aggregates. Consequently, a `CompositionType` can not have runnable entities (see chapter 6.17).
Therefore, the risk for introducing `CompositionType` to the first implementation of AUTOSAR can be estimated as rather limited.

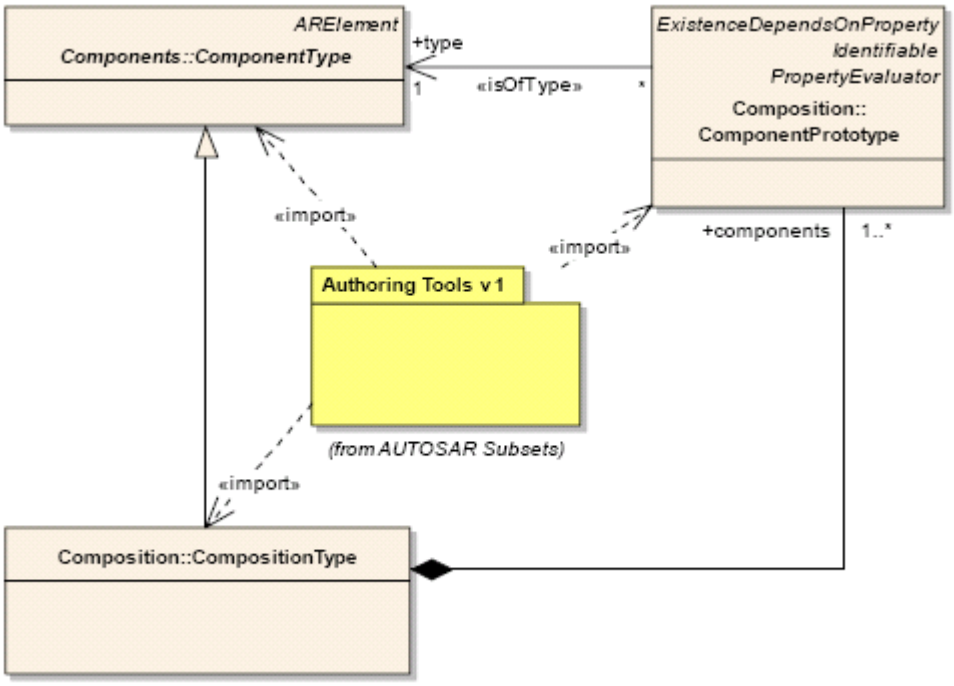

Figure 6-3: Annotation of the meta-model for the support of compositions

For the creation of a composition, the `CompositionType` as well as `ComponentType` and ComponentPrototype are required as the basis. The relation between these meta-classes is sketched by Figure 6-3.

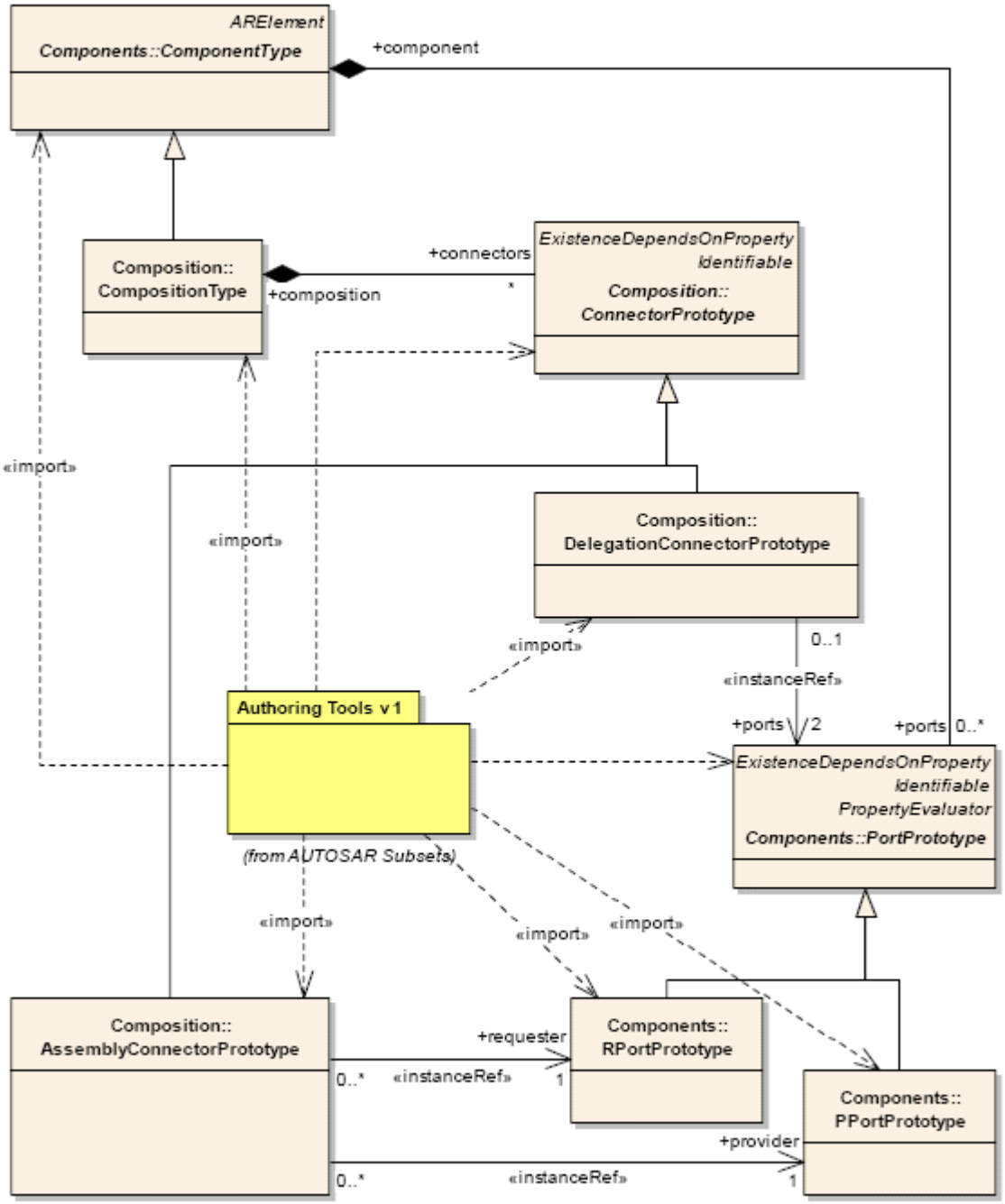

Figure 6-4: Annotation of the meta-model for the support of connectors

Of course, a `CompositionType` can only be created reasonably if `ConnectorPrototype` (i.e. both `DelegationConnectorPrototype` and `AssemblyConnectorPrototype`) is supported as well. This relationship is sketched in Figure 6-4. Furthermore, `PortPrototypes` (i.e. `RPortPrototype` and `PPortPrototype`) are required as well.

6.4 [ARSubset0008] Multiple instantiation

Short description				
The same ComponentType can be used as the basis to create multiple ComponentPrototypes within the scope of a specific CompositionType.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006, ARSubset0007	--	medium	high	P1

Multiple instantiation by reuse of ComponentTypes is one of the key features of AUTOSAR.
The support for multiple instantiation obviously has a large impact on the entire tool chain. In order to limit the effort and risk for source-code implementation, it is possible to provide means for describing multiple instantiation on the design level and use (if this is supported by the underlying layers, e.g. the RTE) a single instantiation scheme on the source-code implementation level.
On the level of Implementation, multiple instantiation has some important characteristics that need to be taken into account properly.
Please note that this feature does not require any additional annotation of the meta-model. All necessary annotations of the meta-model are already contained in Figure 6-3.

6.5 [ARSubset0009] Delegation/assembly connector

Short description				
The AUTOSAR concept provides different kind of connectors for interconnecting software-components.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006, ARSubset0007	ARSubset0010, ARSubset0011, ARSubset0053	medium	high	P1

DelegationConnectorPrototype and AssemblyConnectorPrototype have different semantics and capabilities.
The latter must be implemented properly by AUTOSAR Authoring Tools, i.e. the Authoring Tool is supposed to explicitly distinguish (according to the connected ports) whether the user is about to create a DelegationConnectorPrototype or an AssemblyConnectorPrototype.
The user interface, however, could support an automatic detection of delegation and assembly connectors such that the user would not necessarily have to indicate the type of the connection manually.
Changes of the model at least potentially could have an impact on the usage of DelegationConnectorPrototype and AssemblyConnectorPrototype. Es-

pecially for large models this could have a serious impact on the performance of the AUTOSAR Authoring Tool.
Please note that this feature does not require any additional annotation of the meta-model. All necessary annotations are already implemented in the context of feature ARSubset0007, as depicted in Figure 6-4.

6.6 [ARSubset0010] Sender/receiver relationship for data

Short description				
Establish a data-oriented communication over the VFB				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0009	ARSubset0019, ARSubset0020	medium	high	P1

Sender/receiver relationship is used for transferring data and events (see ARSubset0053) between software-components. This communication of data is considered essential for the first implementation of the AUTOSAR concept.
On the application level, the sender/receiver relationship shall be uniformly used whether data is transmitted over a specific communication bus or merely by means of inter-process communication.

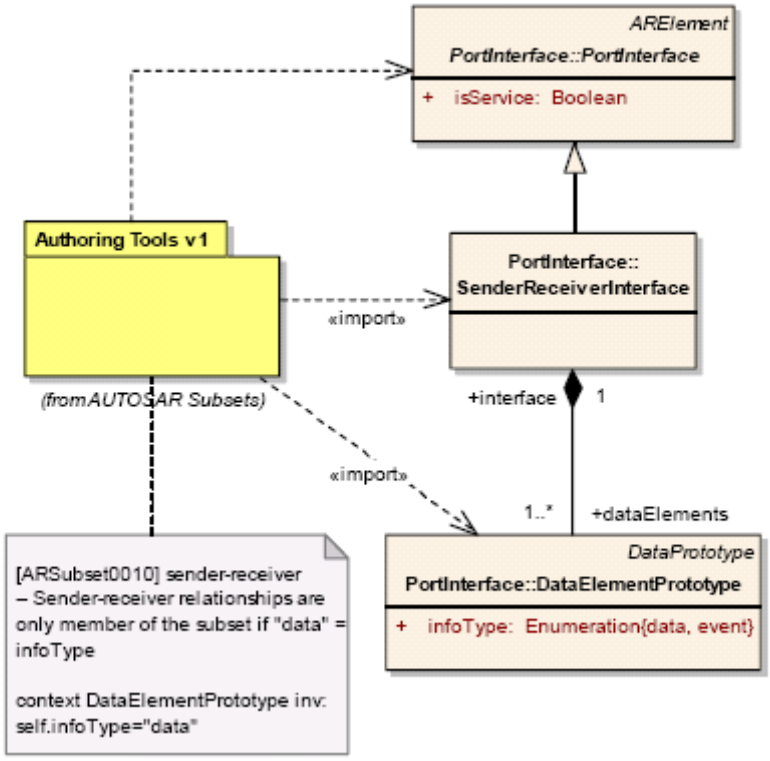

Figure 6-5: Annotation of the meta-model for the support of sender/receiver relations

From the meta-model point of view, the sender-receiver relationship is quite simple to annotate, mainly the meta-classes `SenderReceiverInterface` and `DataElementPrototype` are to be taken into account.

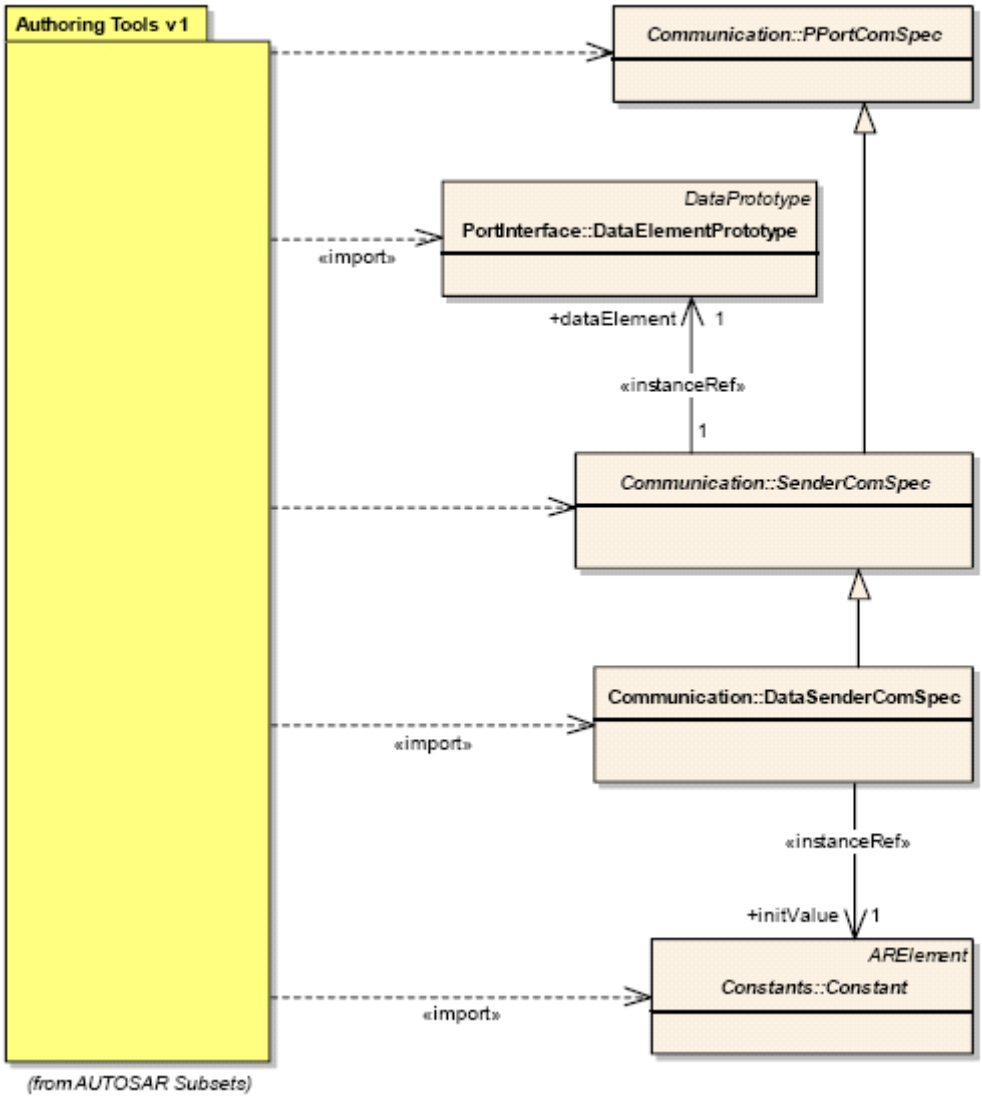

Figure 6-6: Annotation of the meta-model for the support of sender init values

Please note that the definition of initial values at both the sender and the receiver represents another aspect of the description of a sender/receiver relationship.

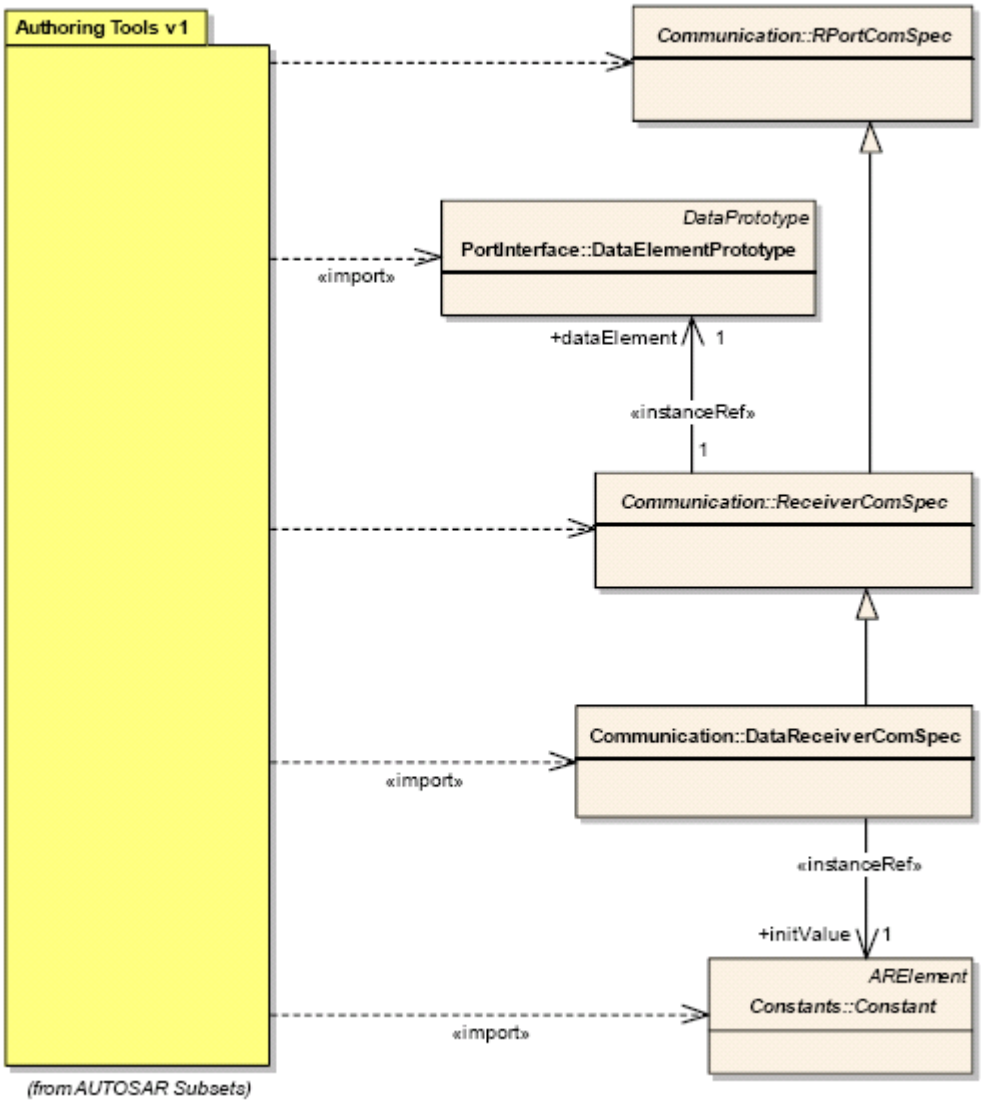

Figure 6-7: Annotation of the meta-model for the support of receiver init values

An initial value is modeled as a `Constant` that is referenced with the `initValue` role. The necessary annotation of the AUTOSAR meta-model has been documented

in Figure 6-6 and

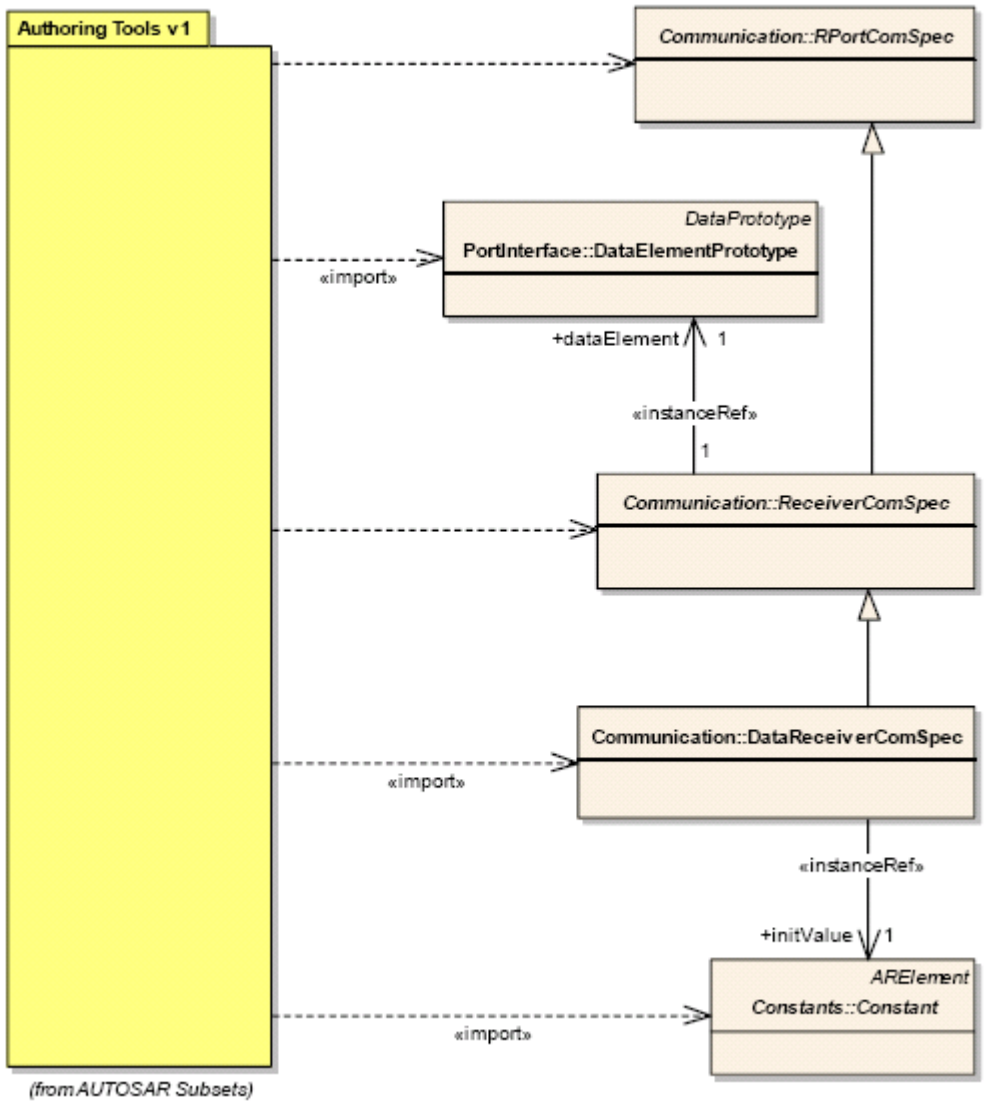

Figure 6-7.

6.7 [ARSubset0053] Sender/receiver relationship for events

Short description				
Establish a sender-receiver communication for events over the VFB				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0009	--	medium	medium	P3

A `DataElementPrototype` might represent either "data" or "events" (the latter are actually data as well, but might have a slightly different semantics). As no use case has been provided for `DataElementPrototype` being used as an "event" this feature is considered of minor importance. Hence, the priority is set to P3.

6.8 [ARSubset0011] Application-level client/server relationship

Short description				
Establish a service-oriented communication over the VFB				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0005, ARSubset0009	--	high	medium	P3

Client/server relationships can potentially be used on the application level (this feature) or for interacting with the RTE and basic software (see feature ARSubset0012). The benefits on the application level, however, are considered to be secondary for the first implementation of AUTOSAR.
Therefore, the priority is set to P3. Since risk is estimated as high and the effort as medium, consideration of this feature for the first implementation of AUTOSAR is strongly discouraged.
State-of-the art communication systems (with the possible exception of MOST, but MOST is not supported by the AUTOSAR COM layer anyway) support only sender/receiver relationships. Therefore, client/server operations must anyway be transformed to a protocol based on sender/receiver relationships.
The protocol will supposedly be very complex since both synchronous and asynchronous operations must be supported. Therefore, the protocol needs to take into account relationships among several frames on the communication bus.
On top of that, it is possible and in the majority of cases very likely that the arguments of the operation can not be mapped to a single frame. Therefore, the client/server protocol needs to implement characteristics of a transport protocol (which, in turn, is not supported [7] within AUTOSAR[8]).
Client/server relationships are mentioned [7] as well (Restr_AR1.0_00002). However, this document only addresses inter-ECU client/server relationships. In other words:

[8] "Transport Protocol cannot be used by AUTOSAR COM. That means that the length of I-PDUs used in AUTOSAR COM is limited to 8 bytes." [7]

software-components that are mapped to the same ECU are entitled to use client/server relationships and software-components on different ECUs are not.
This actually imposes a severe constraint on the mapping of software-components that represents certainly a contradiction to the intention of making software-components independent from their mapping to an ECU.
Please note that the RTE SRS contains a requirement (RTE00082) about the support of client/server communication. It is, however, not clear whether this requirement will actually be covered in the SWS.

6.9 [ARSubset0012] Client/server relationship to RTE and basic software

Short description				
Establish a service-oriented communication over the VFB				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0005, ARSubset0006	--	medium	medium	P2

Client/server relationships can potentially be used on the application level or for interacting with the RTE and basic software. For interacting with the RTE and basic software, processor boundaries shall not be crossed.
The resolution of function calls to any flavor of remote procedure calls is therefore not necessary. This reduces the complexity of the implementation dramatically. This feature could, for example, be used for implementing the support for mode management in software-components.
Please note that communication relationships between the basic software and the application are annotated as "AUTOSAR services" (i.e. for this purpose the attribute `isService` of the meta-class `ClientServerInterface` must be set to TRUE, please refer to ARSubset0005 and Figure 6-1 for more details).
This allows for a clear distinction between client/server relationships on application level one the one hand (ARSubset0011) and between application level and the basic software on the other hand.

6.10 [ARSubset0013] Data variant management

Short description				
Provide means for managing different data variants of a software-component.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	high	P3

The behavior of software-components is to some extent influenced by the values of certain characteristic data (that are formally considered in the model) of the software-

component. The entirety of all characteristic data defines a data set of the software-components.
This enables the application of data variant management to data sets by means of calibration management tools. This feature, though indispensable in series projects, will supposedly not be of real importance for the first implementation of AUTOSAR. Please note, however, that the first implementation can still be used where calibration is not an issue.

6.11 [ARSubset0014] Mode-management

Short description				
Consider mode management in the description of software-components				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006	--	high	high	P3

A common concept on mode-management has not yet agreed (currently, only a first draft of the ECU state manager specification is available) among stakeholders in AUTOSAR. Therefore, it is likely that the concept has not been finalized before the definition of an AUTOSAR subset is agreed.
On top of that, no relevant use case has been identified that requires mode-management to be supported by a first implementation of AUTOSAR Authoring Tools.

6.12 [ARSubset0015] Simple data semantics

Short description				
Provide a "physical meaning" for `DataElements`, etc., on the basis of a simple conversion (identity, linear scaling, table)				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0005	--	low	medium	P1

Simple data semantics provides an association between physical meanings of a `DataElement` to the implementation domain. This is an essential functionality and must be supported in the first implementation of the AUTOSAR concept.
Data semantics is supported according to the MSR-SW concept (i.e `SW-COMPU-METHOD`, `SW-COMPU-SCALE`). The latter allows for a complex definition of e.g. conversion rules.

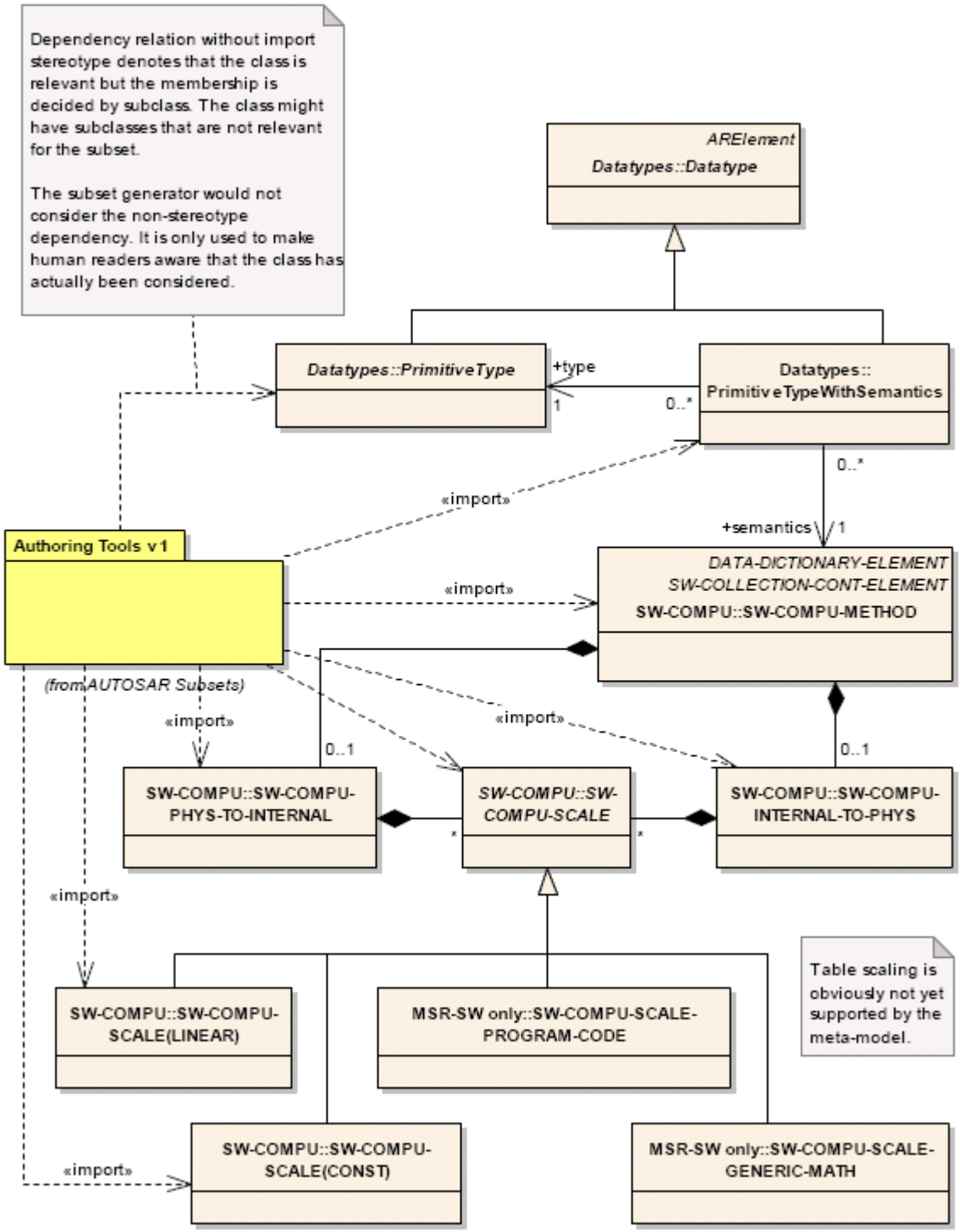

Figure 6-8: Annotation of the meta-model for the support of simple data semantics

As depicted by Figure 6-8, the annotation of the meta-model for the support of simple data semantics covers the meta-classes `SW-COMPU-SCALE(LINEAR)` and `SW-COMPU-SCAL(CONST)`. It would be nice to have a table-based conversion as well but obviously this has not been introduced to the AUTOSAR meta-model at all.

Semantics should be applicable to both the conversion from "physical-to-internal" and the conversion from "internal-to-physical". Therefore, the meta-classes SW-COMPU-PHYS-TO-INTERNAL and SW-COMPU-INTERNAL-TO-PHYS are annotated as well.

6.13 [ARSubset0050] Complex data semantics

Short description				
Provide a "physical meaning" for DataElements, etc., on the basis of a complex conversion (generic math, complex rational functions, program code)				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0005		high	medium	P3

SW-COMPU-SCALE allows for the definition of complex conversion rules, e.g. on the basis of program code or some generic math. Supposedly (at least there is no use case) this feature is not of paramount importance for the first implementation of AUTOSAR Authoring Tools. It is therefore considered a P3 feature.
As depicted in Figure 6-8, the meta-classes SW-COMPU-SCALE-PROGRAM-CODE and SW-COMPU-SCALE-GENERIC-MATH are not annotated in the meta-model and will therefore not be considered by the feature set for the first implementation of AUTOSAR authoring tools.

6.14 [ARSubset0016] Memory resource requirement

Short description				
Specify the memory resource consumption of software-components in order to compare the amount of required memory with the available memory of a particular ECU. By this means it is possible to check whether a specific set of software-components can be mapped to a specific ECU. Please note that the decision whether required memory matches provided memory can only be made in the context of the ECU Configuration.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006, ARSubset0029	--	low	medium	P2

Although it is certainly a good idea to implement preconditions like this for the mapping of software-components to ECUs, the practical significance of the entire approach heavily depends on the quality of memory resource estimation.

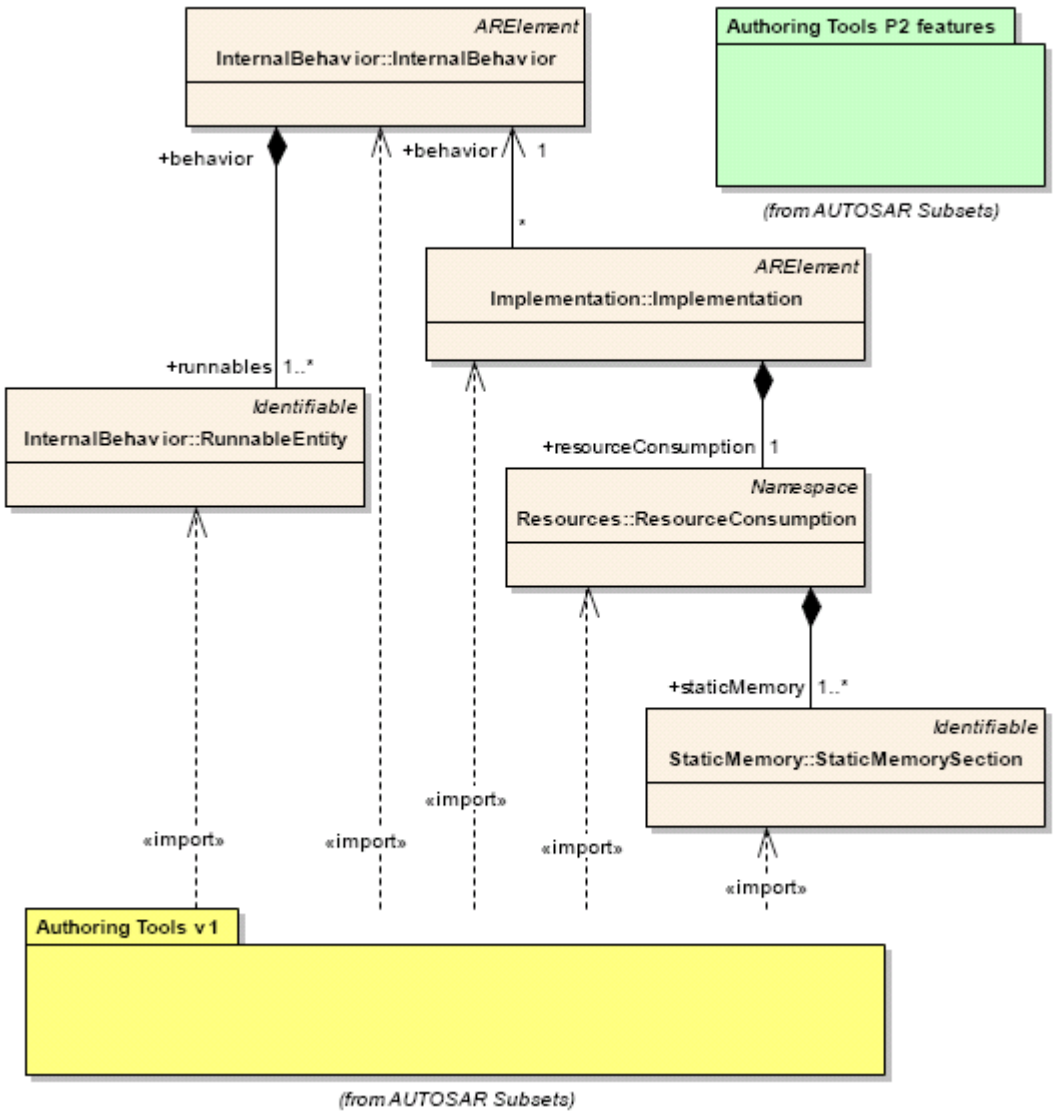

Figure 6-9: Annotation of the meta-model for the support of memory resource requirement (I)

The better the estimation of memory resource consumption can be specified, the better the estimation about the capability of software-components to be mapped to a particular ECU can be derived. Of course, the description of the target ECU is supposed to enclose a description of the available memory.

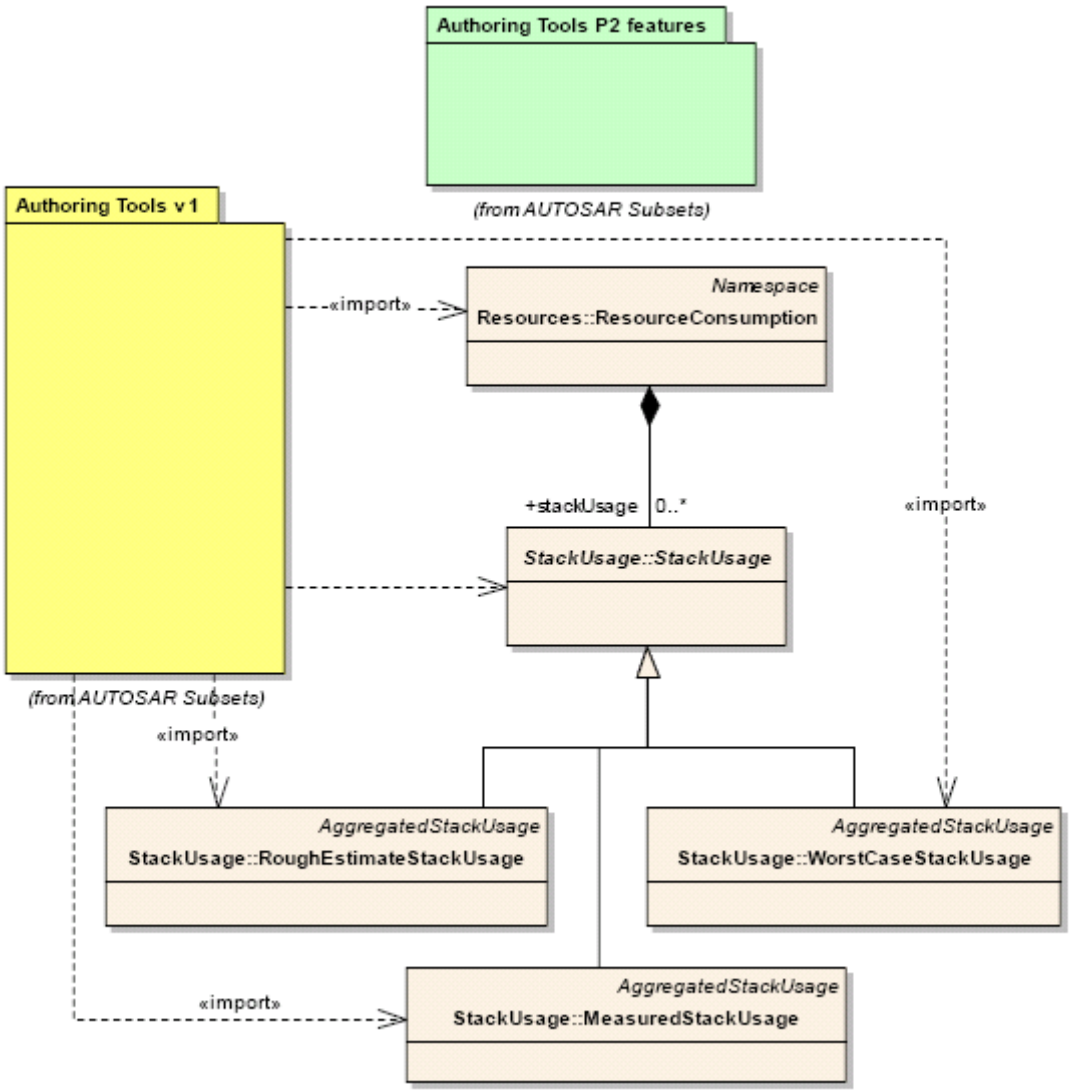

Figure 6-10: Annotation of the meta-model for the support of memory resource requirement (II)

As depicted by Figure 6-9 and Figure 6-10, this feature covers the resource consumption of static memory (represented by the meta-class `StaticMemorySection`) and the `StackUsage`. The latter can be specified as a rough estimate (`RoughEstimateStackUsage`), as a measured value (`MeasuredStackUsage`), and as a (however the value has been identified) worst-case value (`WorstCaseStackUsage`).

6.15 [ARSubset0047] Execution time resource requirement

Short description				
Specify the execution time resource consumption of software-components in order to compare the required performance with the available performance of a particular ECU. By this means it is possible to check whether a specific set of software-components can be mapped to a specific ECU. Please note that the decision whether the required execution time limit matches the performance of the underlying hardware can only be made on the level of ECU configuration.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006	--	high	medium	P3

The appropriate estimation of the Worst-Case-Execution-Time (WCET) is still a basic research problem. If there is no major break-through in research this will put serious limitations on the automation of the system configuration and integration process like automatic mapping of software-components to ECUs, signals to bus frames, etc.
Please note that the impacts of timing specification are not sufficiently addressed in AUTOSAR in general.
On top of that, it is not possible to formally check whether the required execution time can be guaranteed on the underlying ECU because the ECU Resource Template does **not** provide any information about performance.
Please note further that the execution time is actually **not** the relevant information for specifying end-to-end timing requirements. For this purpose the **response time** must be defined. The latter heavily depends on the underlying scheduling policy. It is not yet clarified how response times can be specified with the necessary amount of accuracy.
Under consideration of the described issues it is certainly advised to keep this feature out of the collection of features for a first implementation of AUTOSAR Authoring Tools.

6.16 [ARSubset0017] Coupling to sensors and actuators

Short description				
On the application level, sensors and actuators are represented by a dedicated `SensorActuatorSoftwareComponentType`.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006	--	medium	medium	P1

The `SensorActuatorSoftwareComponentType` implements the application-level behavior of sensors and actuators. A specific `SensorActuatorSoftwareComponentType`, however, must be located on the same ECU as the corresponding sensor or actuator is connected to.

The respective annotation of the meta-model is quite simple. Consult Figure 6-11 for more details. Please note that `DisplayHW` (as a special type of actuator) will not be supported by the first implementation of AUTOSAR Authoring Tools.

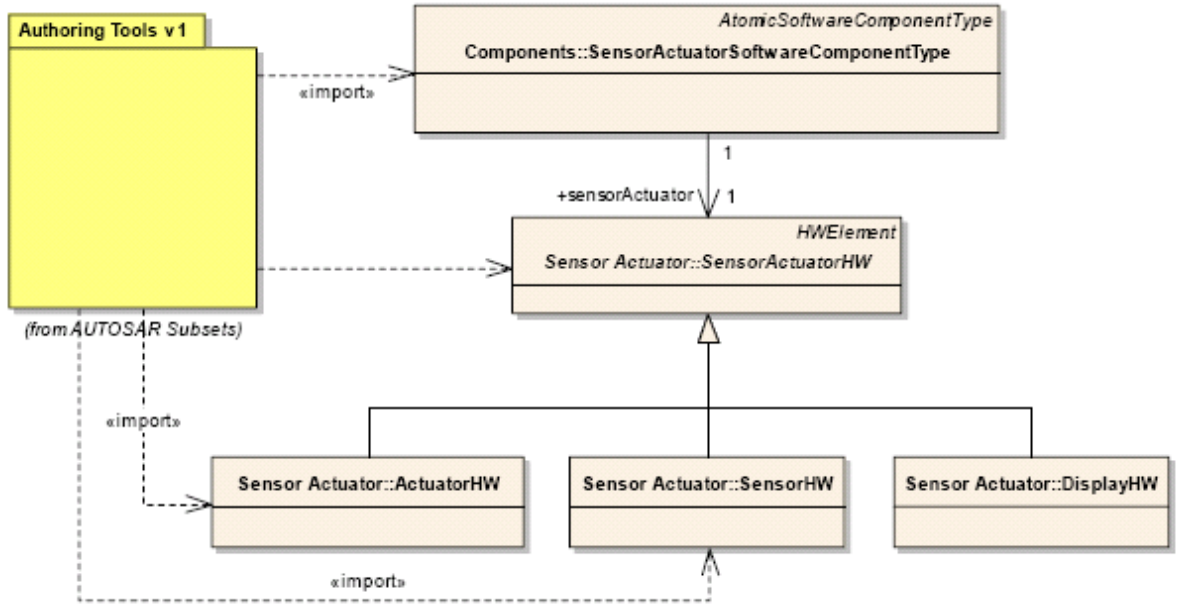

Figure 6-11: Annotation of the meta-model for the coupling of sensors and actuators

This is supposedly an acceptable limitation since AUTOSAR at the moment does not support the telematic and infotainment domains.

6.17 [ARSubset0018] Runnable entities

Short description				
`RunnableEntities` are the "substrate" for implementing the functional behavior of software-components. `RunnableEntities` are activated in response to so-called `RTEEvents`. The latter can have different semantics, e.g. timing event, data reception, etc.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006, ARSubset0051	ARSubset0021, ARSubset0022, ARSubset0057	medium	high	P1

The behavior (in terms of the AUTOSAR meta-model expressed by the dedicated meta-class `InternalBehavior`) of a specific combination of an `AtomicSoftwareComponentType` and an `InternalBehavior` can (and will be in most cases) be distributed over a number (i.e. 1..*) of `RunnableEntities`.
The latter are integrated in (different) RTOS tasks and therefore activated according to the specified policy of the underlying RTOS scheduler.
Please note that the specification of software-components can actually be executed to a certain extent (i.e. definition of `PortPrototypes` and `PortInterfaces`) with-

out having to consider RunnableEntities. In other words: the specification of RunnableEntities is not part of the structural design of AUTOSAR applications.
The implementation of AtomicSoftwareComponentTypes behavior on specific target hardware, on the other hand, is not feasible without the specification of RunnableEntities.
As mentioned before, the behavior of AtomicSoftwareComponentTypes can be implemented either directly by means of (e.g.) C source code or by means of a high-level behavioral model.
The RunnableEntities, on the other hand, correspond to Code which can be either hand-crafted or generated from a behavioral model (please consult the corresponding feature ARSubset0045 for more details).
Please note that Figure 6-12 only (to keep the diagram as readable as possible) depicts a part of the aggregated attributes of RunnableEntities. The rest of the aggregated attributes is covered by Figure 6-14 (DataReadAccess, DataWriteAccess), Figure 6-15 (DataSendPoint, DataReceivePoint), and Figure 6-16 (RunnableEntityRunsInExclusiveArea, RunnableEntityCanEnterExclusiveArea). Furthermore, Figure 6-17 covers the membership of InterRunnableVariable.

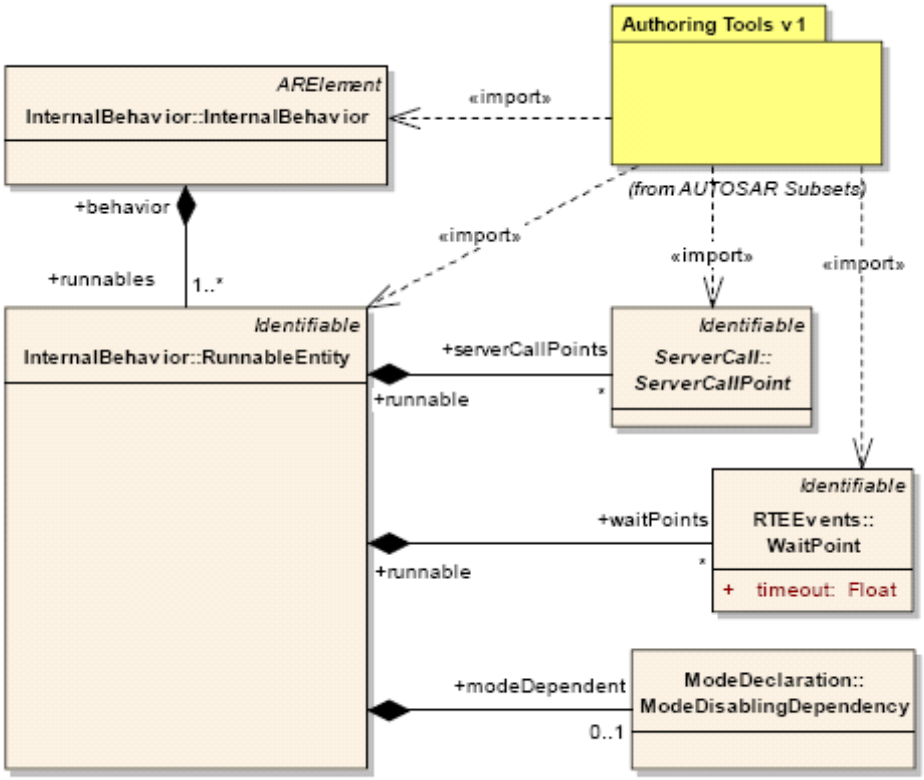

Figure 6-12: Annotation of the meta-model for the support of runnable entities

As discussed in ARSubset0014, mode management is not yet supported by the first implementation of AUTOSAR Authoring Tools.

6.18 [ARSubset0051] RTEEvents

Short description				
RTEEvents are used to describe the activation of RunnableEntities and the trigger of a WaitPoint.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0018	--	medium	medium	P1

The definition of RTEEvents is inseparably connected to the definition of a RunnableEntity. For each RunnableEntity a set of RTEEvents that are supposed to trigger the RunnableEntity can be defined.
For the first implementation of AUTOSAR, however, only a more or less small part of the collection of RTEEvents defined in the AUTOSAR meta-model is taken into account. Please consult Figure 6-13 for more details.

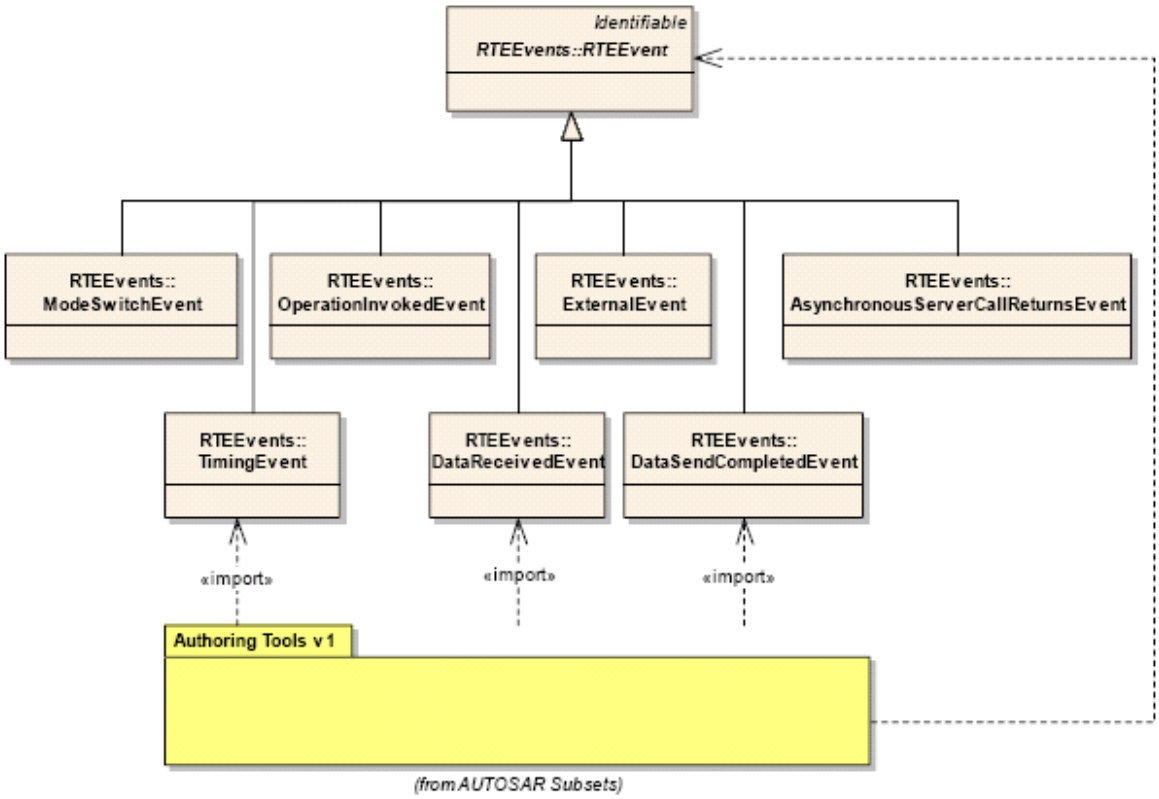

Figure 6-13: Annotation of the meta-model for the support of RTEEvents

The ModeSwitchEvent is not supported in the first implementation because mode-management in general is not a supported feature. Please refer to the description of feature ARSubset0014 for more details.
The OperationInvokedEvent is not supported because this event is only used on the server-side of a client-server relationship. This constellation will not occur in the first implementation of AUTOSAR because client-server relationships are only permit-

ted in combination with the basic software. In this case, however, the application software-component acts in all cases as the client.
The `ExternalEvent` is not supported because the implications of this type of event are not clearly defined in the specification of the Software-Component Template. The `AsynchronuousServerCallReturnsEvent` is not supported because this type of event is only necessary for application-level client-server relationships. The interaction with the basic software will always be synchronous.
On the other hand, the `TimingEvent`, `DataReceivedEvent`, and `DataSendCompletedEvent` will certainly be required for the first implementation of AUTOSAR.
It is not finally clarified whether `RunnableEntities` of category 2 will be supported in the first implementation of AUTOSAR. If this is the case (and tentatively assumed as true) then it is necessary to annotate the `WaitPoint` meta-class.

6.19 [ARSubset0019] Buffered sending and receiving of data elements

Short description				
According to the AUTOSAR concept, data elements are sent after the corresponding runnable terminates and received before the corresponding runnable starts.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0010	--	medium	medium	P1

This feature directly corresponds to the meta-classes `DataWriteAccess` and `DataReadAccess`. The concept of these meta-classes contributes to the consistency of information received from or sent to other software-components.
Received information does not change during the run-time of runnables even if a new value has actually been received over a communication bus.

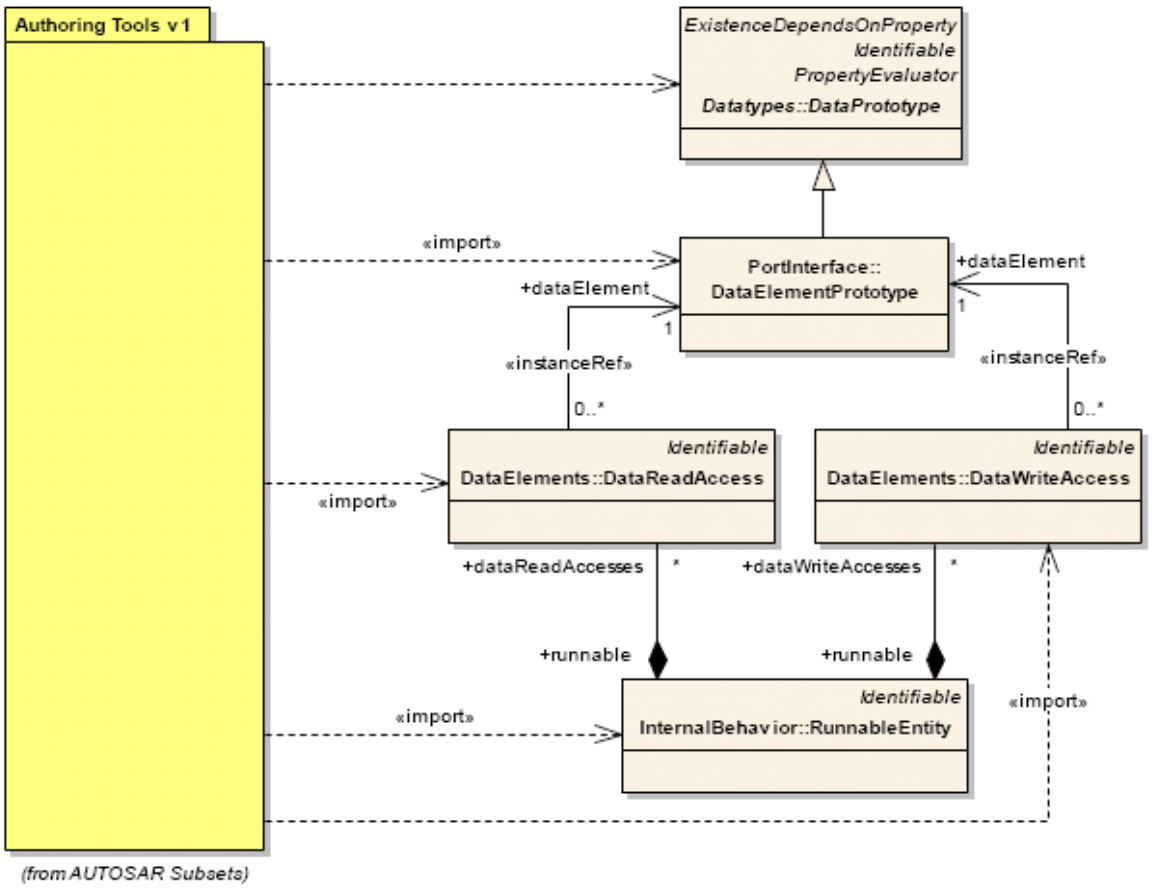

Figure 6-14: Annotation of the meta-model to support buffered sending and receiving

The sender-receiver relationship certainly needs this feature to be actually usable. It is therefore justified to set it to P1.

6.20 [ARSubset0020] Explicit sending of data elements

Short description				
Data elements can be sent explicitly after the corresponding runnable encounters a data send point.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0010	ARSubset0049	medium	medium	P1

This feature is directly connected to the meta-classes `DataSendPoint` and `DataReceivePoint`. Both are (in the code of a `RunnableEntity`) represented by API-calls that forward the values of the corresponding `DataElementPrototype` to and from the RTE.
The sender-receiver relationship certainly needs this feature to be actually usable. It is therefore without any doubt justified to set it to P1.

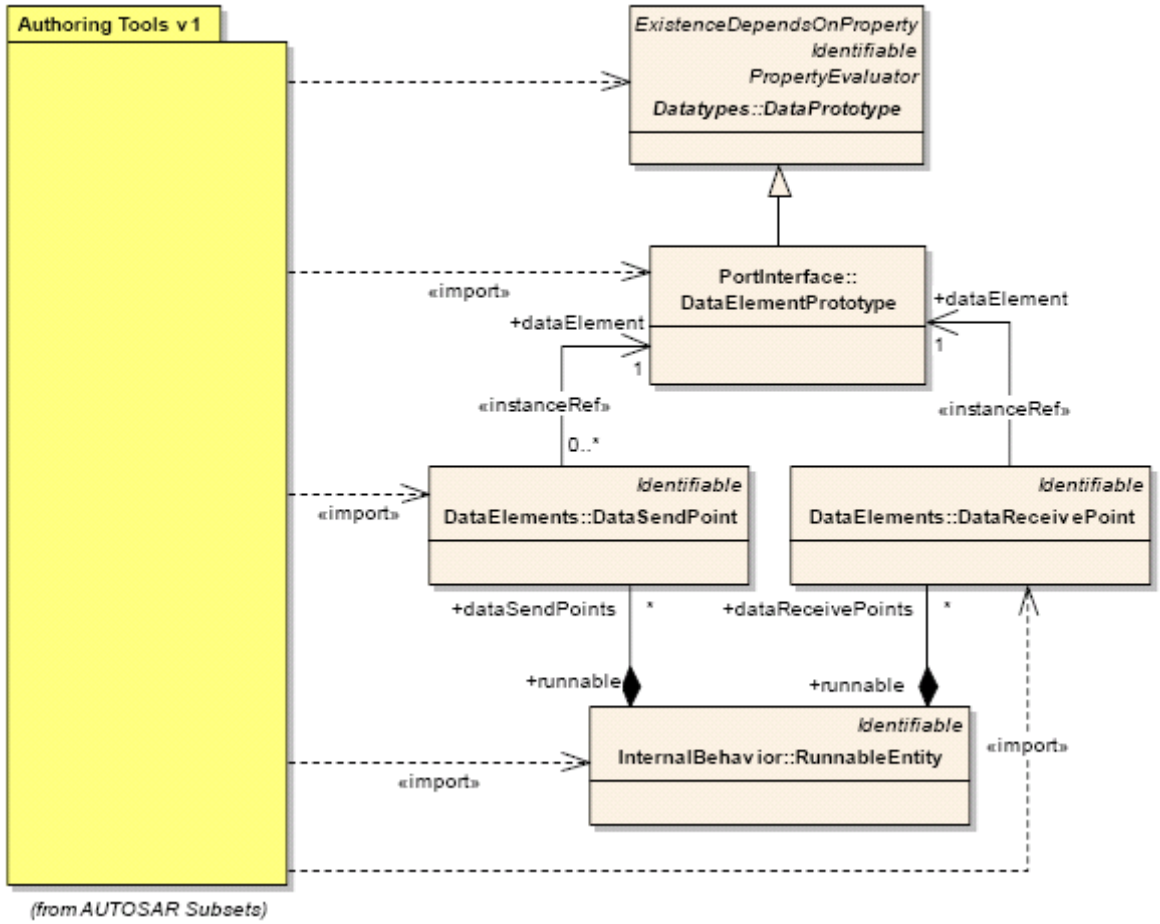

Figure 6-15: Annotation of the meta-model for the support of explicit sending and receiving

In addition, please note that this feature could be of importance for the implementation of NVRAM storage for information contained in a software-component.

6.21 [ARSubset0021] Exclusive area

Short description				
Goal: make sure that information exchanged between `RunnableEntities` on the same ECU is consistent within the run-time of the corresponding `RunnableEntities`. This approach is based on the assumption that `RunnableEntities` belonging to a specific `ExclusiveArea` would not interrupt each others				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0018	--	medium	low/high	P2

During run-time of an AUTOSAR software application, the exchange of information is not limited to the communication among specific software-components. Especially the `RunnableEntities` of a particular software-component certainly need to ex-

change information for proper operation. The consistency of this information obviously must be guaranteed as well.

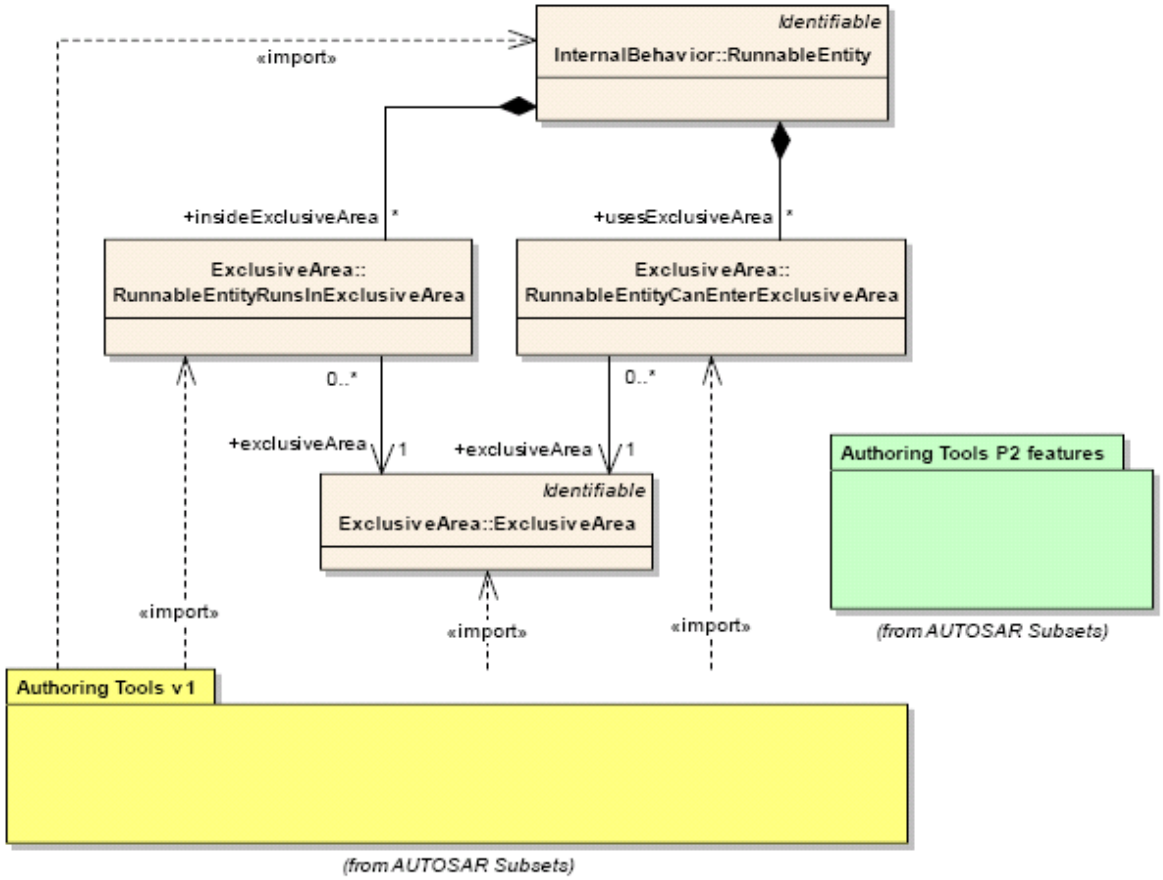

Figure 6-16: Annotation of the meta-model for the support of exclusive areas

The specification of the Software-Component Template [1] describes several concepts for ensuring the data consistency of inter-runnable communication. In addition, it would perhaps be possible to **either** implement application-level data consistency algorithms **or** (and this should certainly not be the preferred solution) use the VFB-mechanisms for inter-runnable communication.

6.22 [ARSubset0057] Inter-runnable variable

Short description				
Goal: make sure that information exchanged between `RunnableEntities` on the same ECU is consistent within the run-time of the corresponding `RunnableEntities`. In this case the consistency protection is based on the formal description of information exchanged among `RunnableEntities`.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0018	--	medium	low/high	P2

The meta-model provides a dedicated meta-class `InterRunnableVariable` for implementing this feature.

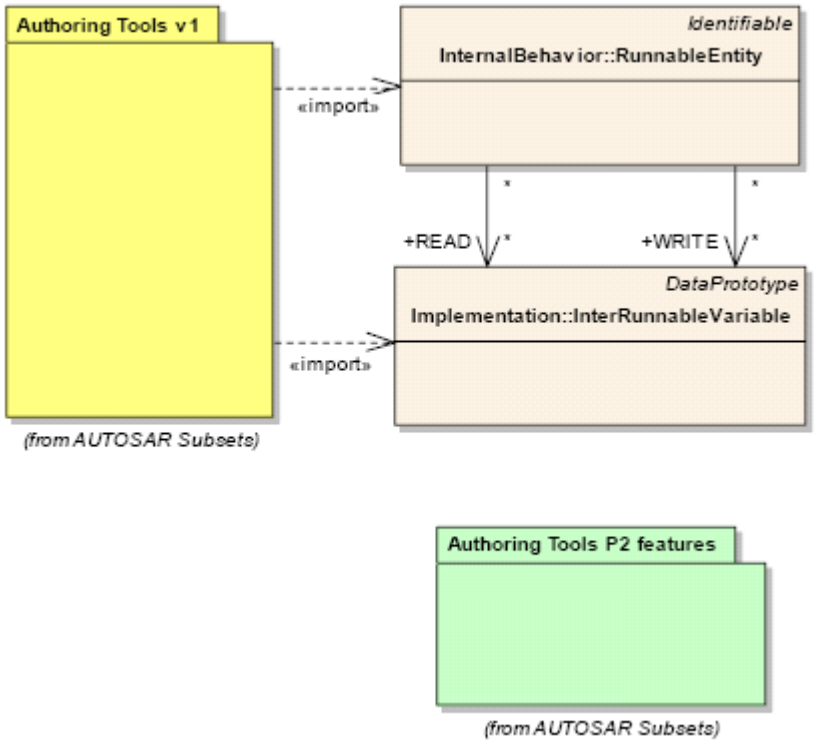

Figure 6-17: Annotation of the meta-model for the support of inter-runnable variables

Therefore, the annotation of the meta-model is comparably easy to implement, see Figure 6-17.

6.23 [ARSubset0022] Execution order of runnable entities

Short description				
Specification of a deterministic execution order of `RunnableEntities` of a specific software-component.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0018	--	medium	medium	P3

The execution order of `RunnableEntities` is primarily determined by the data flow among `RunnableEntities`. It is obvious that a `RunnableEntity` that provides data would have to be executed preceding a `RunnableEntity` that consumes the information.
Nevertheless, there are constraints about the execution order of `RunnableEntities` that can not be expressed by means of data flow criteria. For example, certain `RunnableEntities` could be used for initialization purposes while others could be specially implemented to terminate the functionality of a software-component.

As a rule, the determination of a feasible execution order of `RunnableEntities` at compile time is directly linked to the Worst-Case Execution Time (WCET) estimation problem.
During initialization, no `RunnableEntity` of the "normal operation" kind is supposed to be executed. This constraint can be described by means of the execution order of `RunnableEntities`.
In the first implementation of the AUTOSAR concept, however, these constraints can be implemented manually without having to be formally expressed. Therefore, the feature is given a P3.

6.24 [ARSubset0049] Specification of NVRAM resource consumption

Short description				
The Software-Component Template allows for the specification of resource consumption for NVRAM storage.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0020	--	medium	medium	P2

The specification of resource requirements for NVRAM storage is different (in terms of attributes in the AUTOSAR meta-model) from the specification of other memory resources. A tool needs to take the different attributes and their semantics into account.

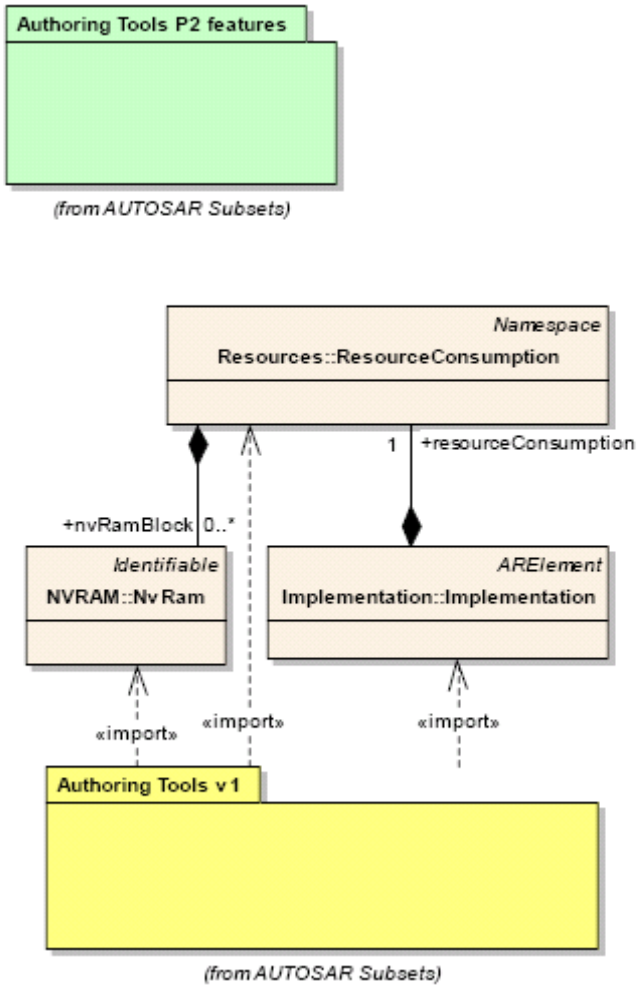

Figure 6-18: Annotation of the meta-model for the specification of NVRAM consumption

The necessary annotation of the AUTOSAR meta-model is depicted in Figure 6-18. Besides the inclusion of `ResourceConsumption` it is necessary to annotate the meta-class `NvRam` as well.

6.25 [ARSubset0062] Definition of physical units

Short description				
The definition of DataElements is extended by the definition of a physical unit for each DataElement				
Initiator				
WP1.2				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P1

The consideration of physical units is yet another feature based on the MSR-part of the AUTOSAR meta-model. The necessary meta-classes are sketched in Figure 6-19.

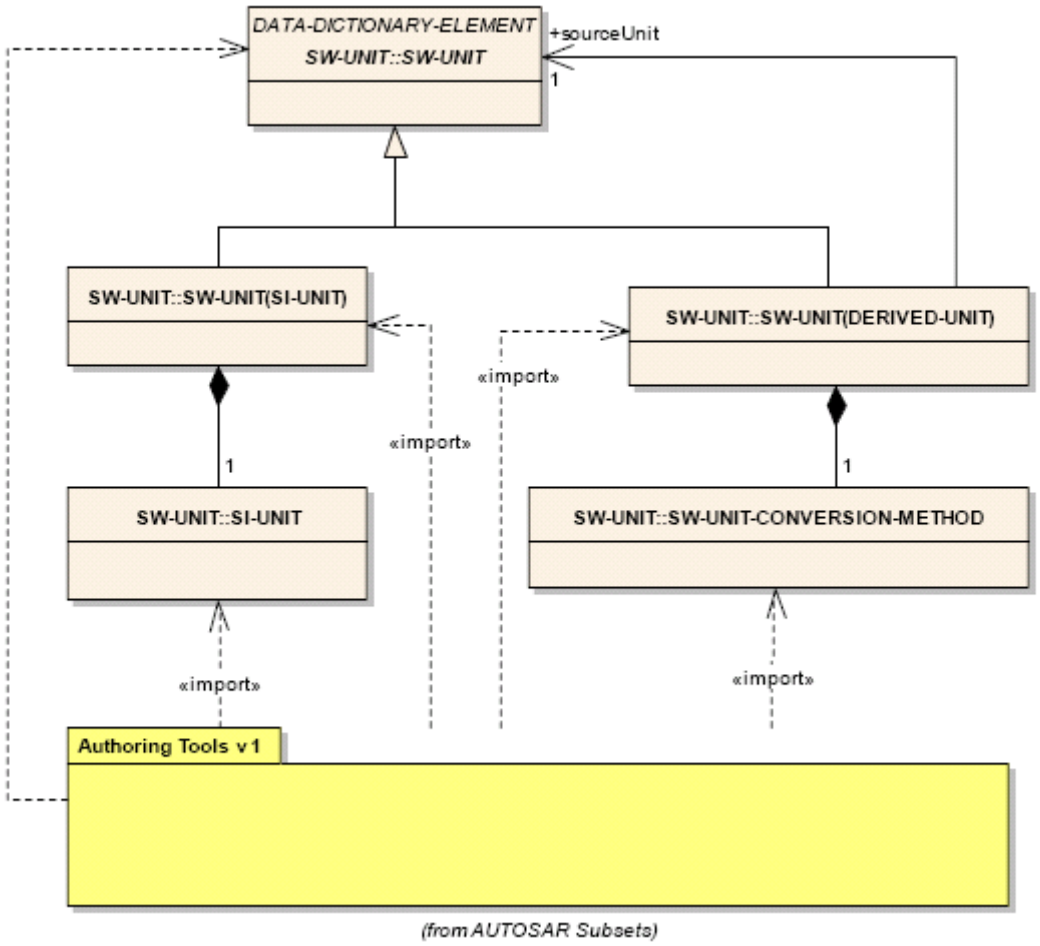

Figure 6-19: Annotation of the meta-model for the support of physical units

The consideration of physical units has relationships to the definition of conversion formulae as described primarily by ARSubset0015 and ARSubset0050.

6.26 [ARSubset0063] Comments

Short description				
During the creation of AUTOSAR models it is possible to add comments to identifiable model elements.				
Initiator				
WP1.2				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	low	low	P1

The specification of comments is currently not covered by the AUTOSAR meta-model. However, the formalization as described in [5] foresees the introduction of a description field in the DTD/XML Schema.
In other words, it is possible to add comments to identifiable model elements, but this feature cannot be derived from the meta-model. The creation of an annotated meta-model is consequently not possible.

7 Assessment of the ECU Resource Template

This chapter contains an evaluation of hardware-related features that are described in the ECU Resource Template. Please note that the ECU Resource Template provides means for describing an ECU and the devices mounted to it (in extreme cases down to the gory detail).
While there are certainly cases where the detail is needed to properly describe the characteristics of hardware, it is assumed (and that's precisely the reason why ARSubset0058 is only a P2 feature) that a first implementation of the AUTOSAR concept can be created without having to take into account a really detailed description of hardware.
Therefore, it is assumed that for each feature described below the level of detail is limited to the really essential aspects. For example: the power supply of an ECU and its devices can be neglected for the first step without losing too much functionality.

7.1 [ARSubset0058] Hierarchical hardware description

Short description				
The ECU Resource Template provides means for the hierarchical description of hardware structures. The number of hierarchies is not limited by the modeling paradigm.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0026	high	high	P1

The hierarchical description of hardware at arbitrary levels of hierarchies requires support for the meta-class `HWContainer` as well as for delegation connectors represented by `DelegationHWConnection` and `MemoryMappedDelegationHWConnection`.
The annotation of the meta-model is sketched in Figure 7-1. Please note that the basic description of an ECU as described in ARSubset0023 represents some sort of a hierarchy as well.
In contrast to the totally generic approach, feature ARSubset0023 only supports a very basic hierarchy that consists of an ECU and the next hierarchy level consisting of a PU, memory, peripherals, etc.

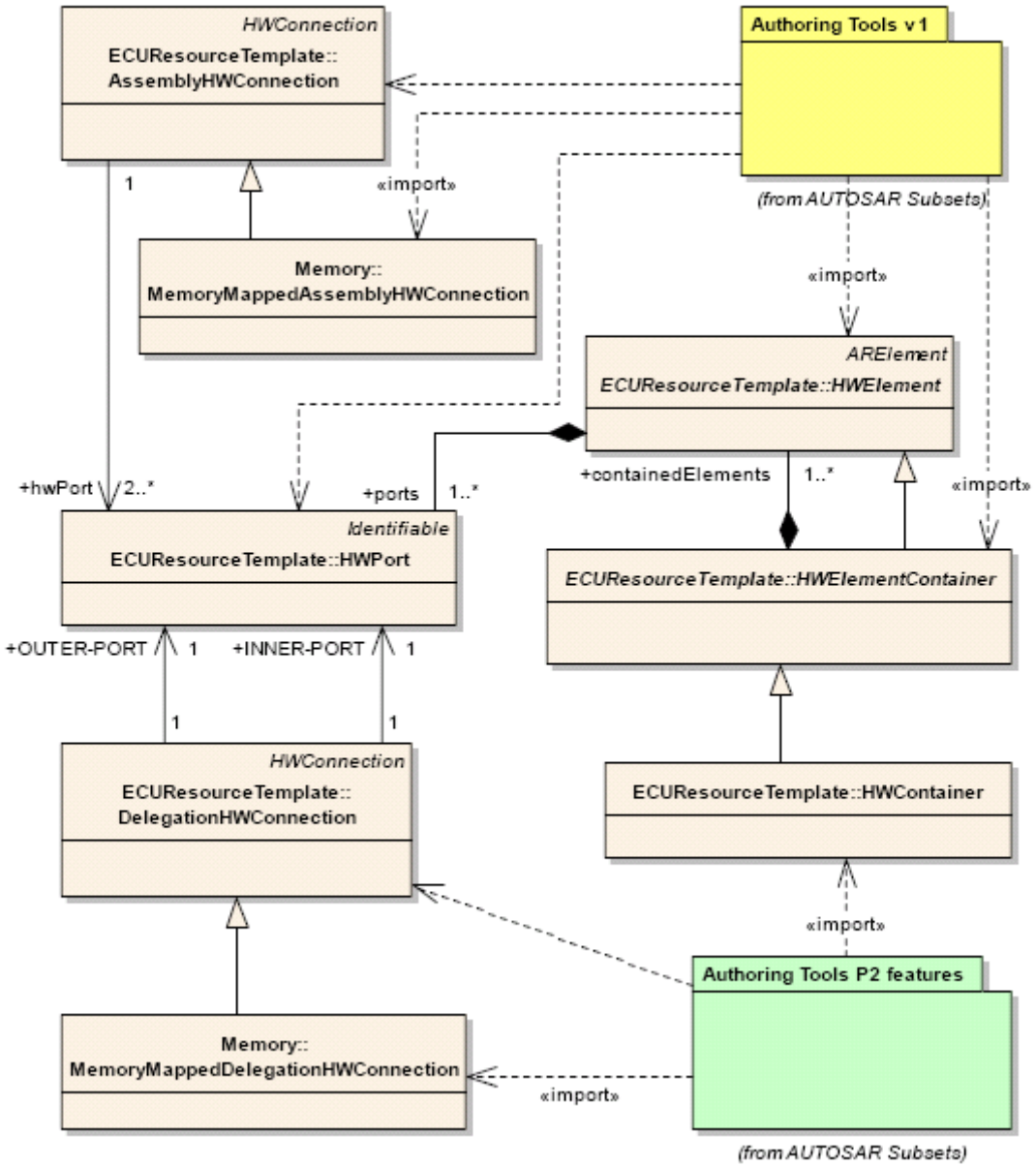

Figure 7-1: Annotation of the meta-model for the hierarchical description of hardware

Therefore, the creation of arbitrary levels of hierarchies is not actually mandatory to be capable of describing entire ECUs. It is a supplementary feature that will most likely be used where AUTOSAR hardware is described to a great level of detail.

7.2 [ARSubset0023] Definition of ECUs

Short description				
Provide means for defining ECUs for AUTOSAR applications				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0024, ARSubset0025, ARSubset0026, ARSubset0027, ARSubset0028, ARSubset0029, ARSubset0030, ARSubset0052	medium	medium	P1

According to the AUTOSAR ECU Resource Template, an ECU is a mere container for devices residing on it. Every ECU must be equipped with at least a single `ProcessingUnit`. Therefore, it could be justified to combine the definition of an ECU directly with the definition of a `ProcessingUnit` residing on it.

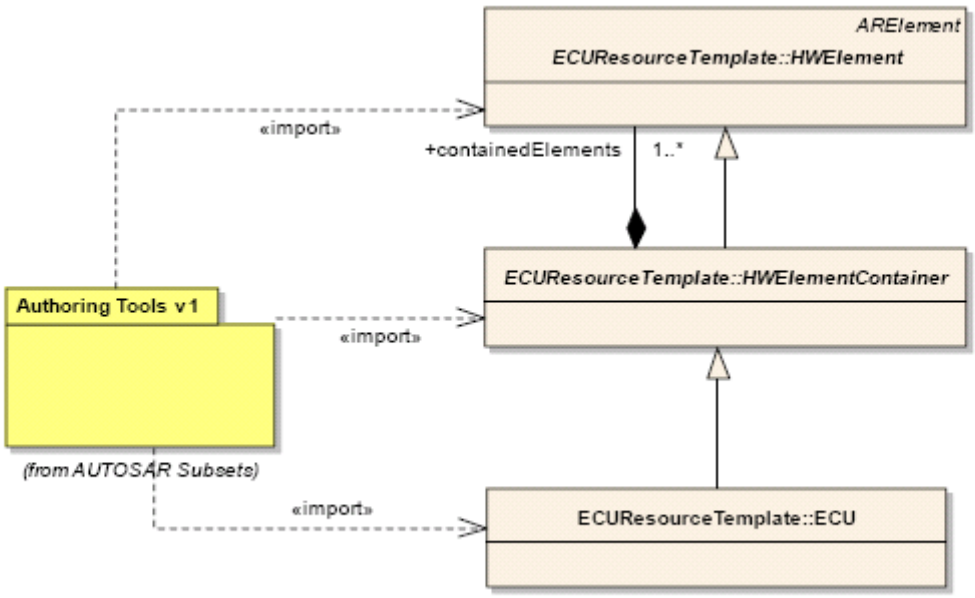

Figure 7-2: Annotation of the meta-model for the support of ECU definitions

As per description of feature ARSubset0030, ECUs with more than a single `ProcessingUnit` would most likely not be supported in the first implementation step.

7.3 [ARSubset0024] Definition of communication peripherals

Short description				
Add the definition of CommunicationPeripherals to the description of an ECU				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P1

CommunicationPeripherals are required to physically exchange information with other ECUs.

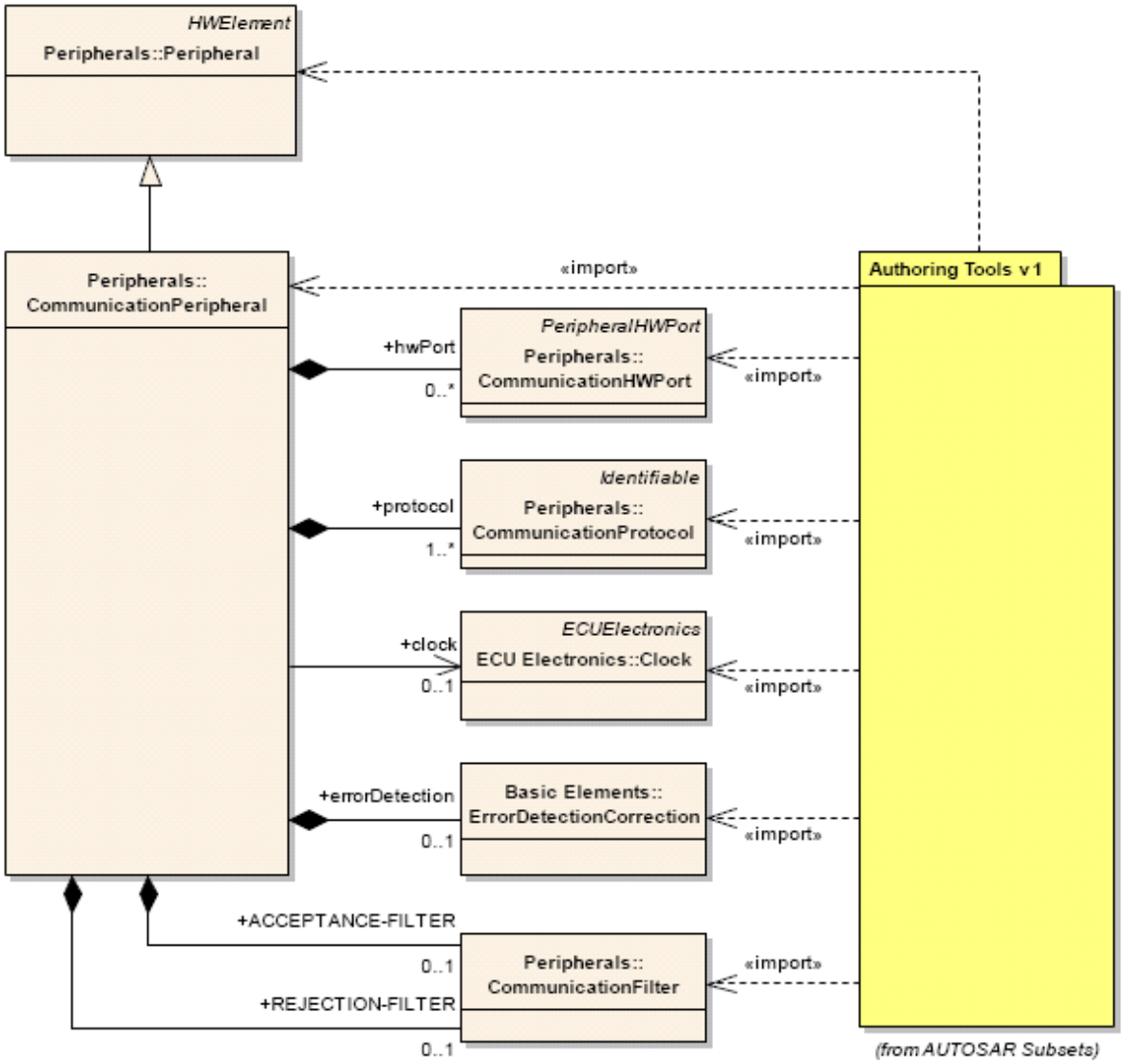

Figure 7-3: Annotation of the meta-model for the support of communication peripherals

A single ECU can have several CommunicationPeripherals of different kinds, e.g. CAN, LIN, etc.

7.4 [ARSubset0025] Definition of I/O peripherals

Short description				
Add the definition of general purpose input/output peripherals to the description of an ECU				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P1

The counterpart to `CommunicationPeripherals` for communication is the GPIO peripheral to connect `SensorHW` and `ActuatorHW` to the `ProcessingUnit`.

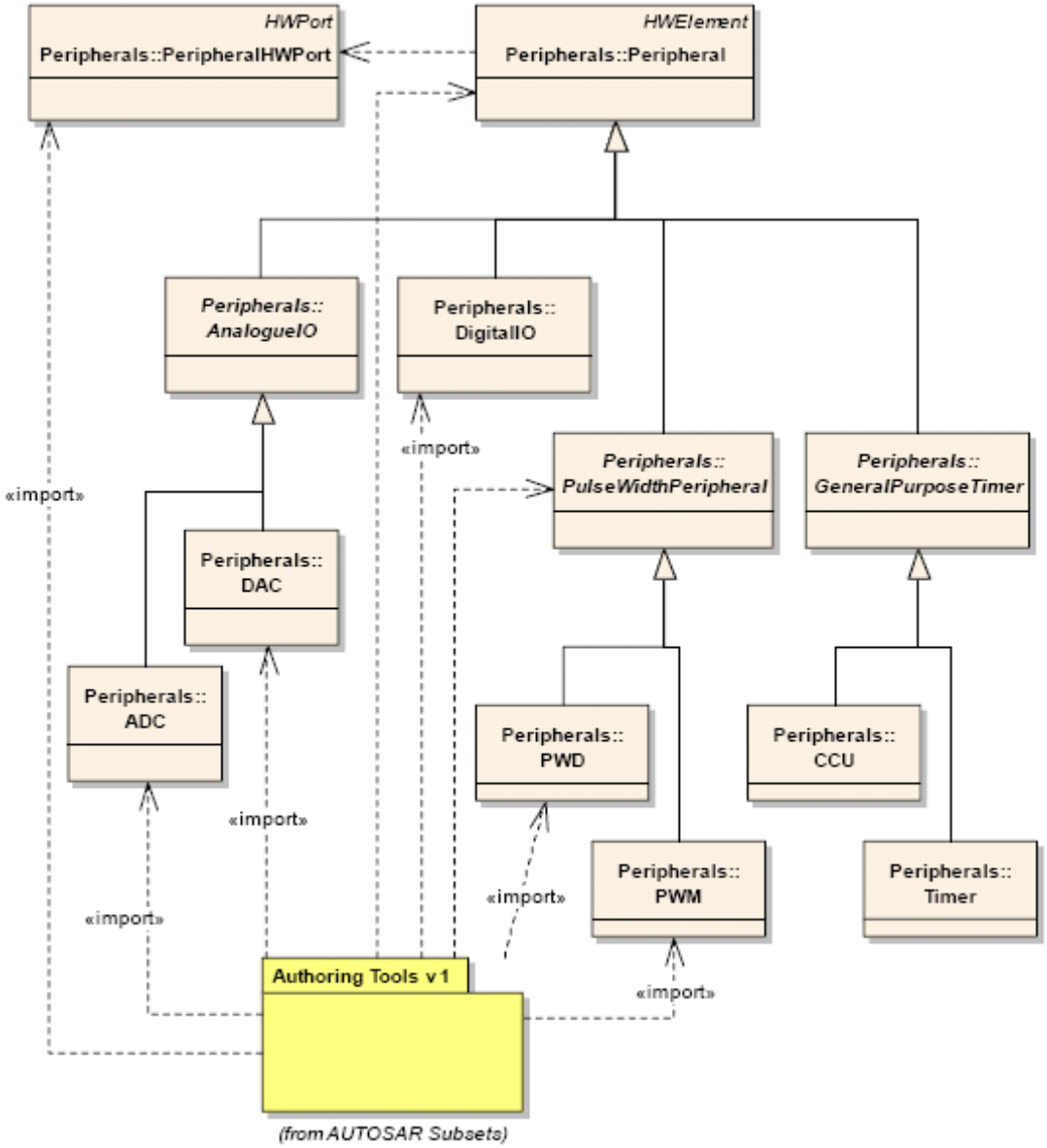

Figure 7-4: Annotation of the meta-model for the support of I/O peripherals

7.5 [ARSubset0026] Definition of ECU electronics

Short description				
Add the definition of ECUElectronics to the formal description of an ECU				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023, ARSubset0058	--	medium	medium	P2

The formalized description of the peripheral electronic components of an ECU (in particular: Oscillator, CommunicationTransceiver, and power electronics) is necessary for the detailed description of ECU hardware.
An overview of the meta-model annotation for the support of ECU electronics is depicted in Figure 7-5. The meta-classes PowerSupplyHWElement and PowerDriverHWElement are of a more or less complex structure that suggests the creation of two additional figures, i.e. Figure 7-6 and Figure 7-7.

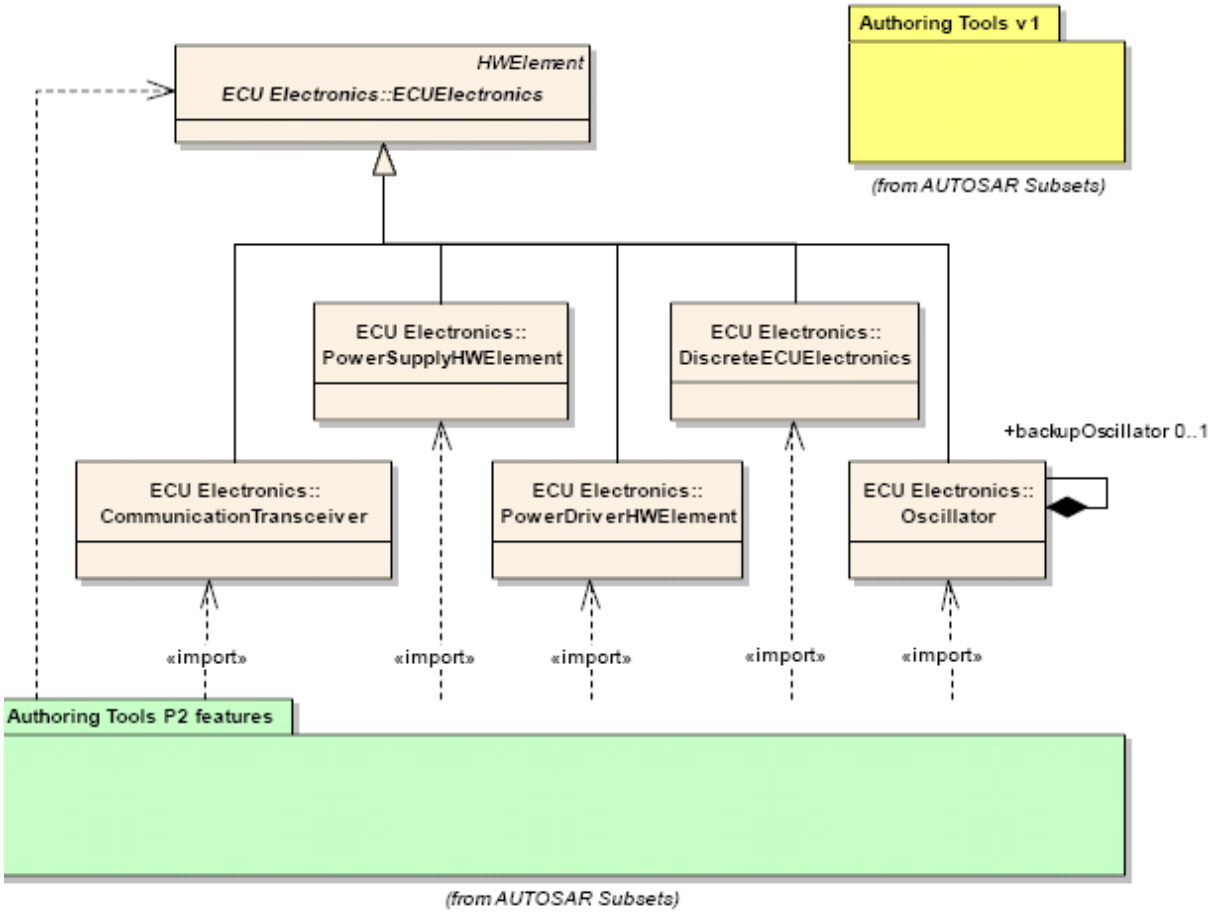

Figure 7-5: Annotation of the meta-model for the support of ECU electronics

The PowerDriverHWElement aggregates the PowerDriverProtection and PowerDriverNotification. Both meta-classes must be annotated as a member of a P2 feature (see Figure 7-6).

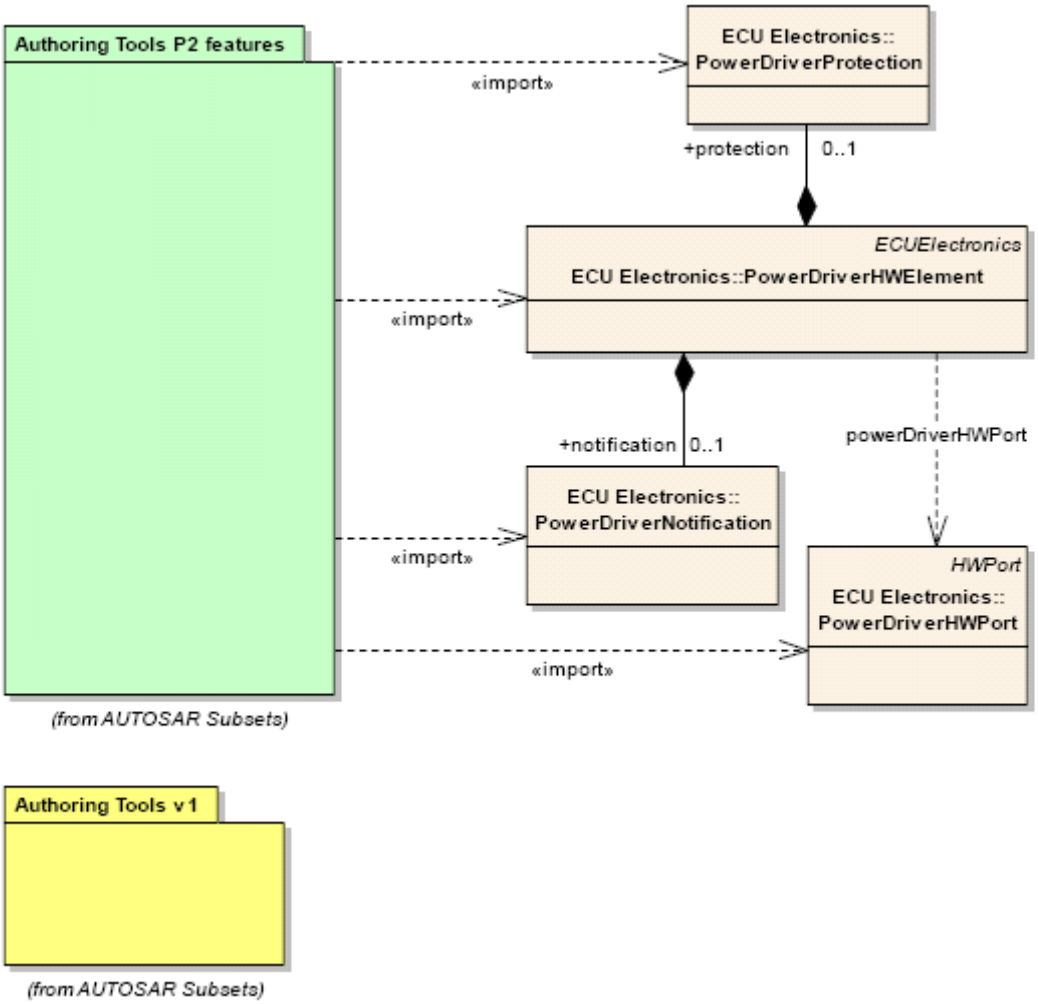

Figure 7-6: Annotation of the meta-model for the support of power driver electronics

Obviously, in addition to the `PowerDriverHWElement` it is mandatory to add the `PowerDriverHWPort` to the collection of annotated meta-classes.
Concerning the description of power supply electronics, it is indispensable to annotate the `PowerSupplyHWElement` and the corresponding `PowerSupplyHWPort`. On top of that, the `PowerSupplyVoltageNotification` as well as the `PowerSupplyCurrentNotification` are certainly of interest for modeling.

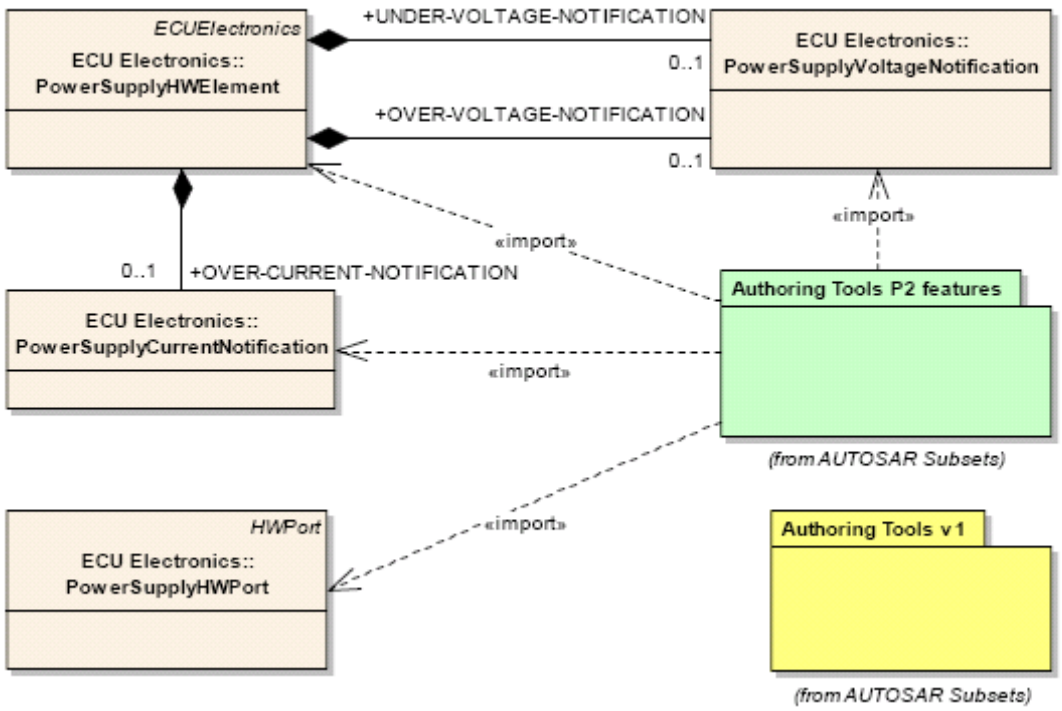

Figure 7-7: Annotation of the meta-model for the support of power supply electronics

Since the creation of a hierarchical hardware structure (i.e. ARSubset0058) has been given a P1, this feature must at least be considered for implementation in the first release of AUTOSAR Authoring Tools. It is therefore tentatively rated with a P2 priority.

7.6 [ARSubset0027] Definition of sensors/actuators

Short description				
Add the definition of `SensorHW` and `ActuatorHW` to the description of an ECU				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P1

Like `CommunicationPeripherals`, `SensorHW` and `ActuatorHW` are indispensable for the operation of AUTOSAR ECUs.

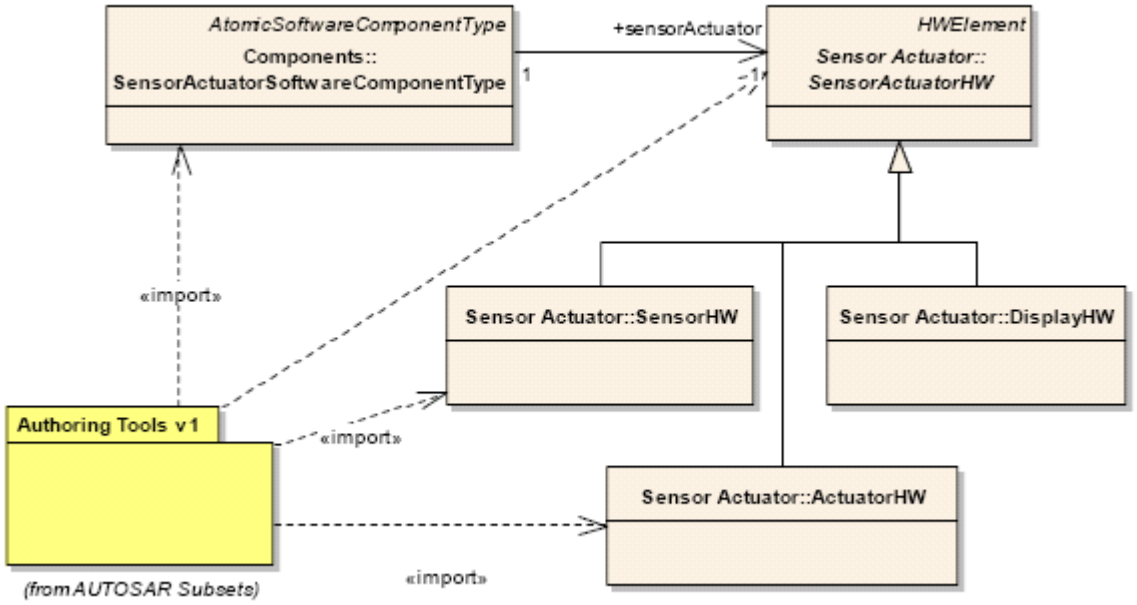

Figure 7-8: Annotation of the meta-model for the definition of sensors and actuators

The number of `SensorHW` devices and `ActuatorHW` devices attached to a single ECU is in principle **not** limited by the AUTOSAR concept.

7.7 [ARSubset0052] Definition of displays

Short description				
Add the definition of displays to the description of an ECU				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P3

It is assumed (as documented in Figure 7-8 of ARSubset0027) that `DisplayHW` is not of importance for the first implementation of AUTOSAR Authoring Tools. Therefore, the meta-class `DisplayHW` is not annotated.

7.8 [ARSubset0028] Signal transformation

Short description				
Provide the description of `SignalTransformation` within the chain from `SensorHW` to the `ProcessingUnit` and/or from there to an `ActuatorHW`.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P3

The description of SignalTransformation is helpful for creating sensor/actuator-relevant software (i.e. SensorActuatorSoftwareComponentType). It certainly helps the software developer to understand the main characteristics of the SensorHW or ActuatorHW.
Furthermore, parts of the low-level software of an ECU could probably automatically be configured by dedicated code generation tools within the AUTOSAR scope.

7.9 [ARSubset0029] Available memory

Short description				
The description of available memory provided the basis for comparing the resource requirements of the software with the resources available at the hardware.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	ARSubset0016	medium	medium	P2

The specification of ProvidedMemorySegments allows for a check whether the requirements of a collection of software-components in combination with the necessary basic software can be met by the underlying ECU. Please note that the check can only be carried out on the level of an ECU configuration.
Please note that the description of this feature (see Figure 7-9 for more details) contains the description of both volatile (represented by the meta-class ProvidedMemorySegment) and non-volatile (represented by the meta-class ProvidedNVMemorySegments) memory.
In general, the MemoryMappedHWPort is not required for this feature because its attributes do not have any relevance for the mere description of available memory.
On the other hand, the ProvidedMemorySegment is required to provide at least a single HWPort. Therefore, the annotation of the MemoryMappedHwPort is applied for technical reasons only.
Please note that for the computation of the overall memory capacity it is **not** feasible to simply sum up the capacities of the individual memory segments because the usage of each segment must be taken into account properly.
As a matter of fact, the usage of each segment is determined by the MemoryMappedHWConnection to the segment. Therefore, the meta-class MemoryAccess is included because it provides information about the usage of a specific access to a memory segment. The MemoryMappedAssemblyConnection provides information whether the segment is used for either code or data.

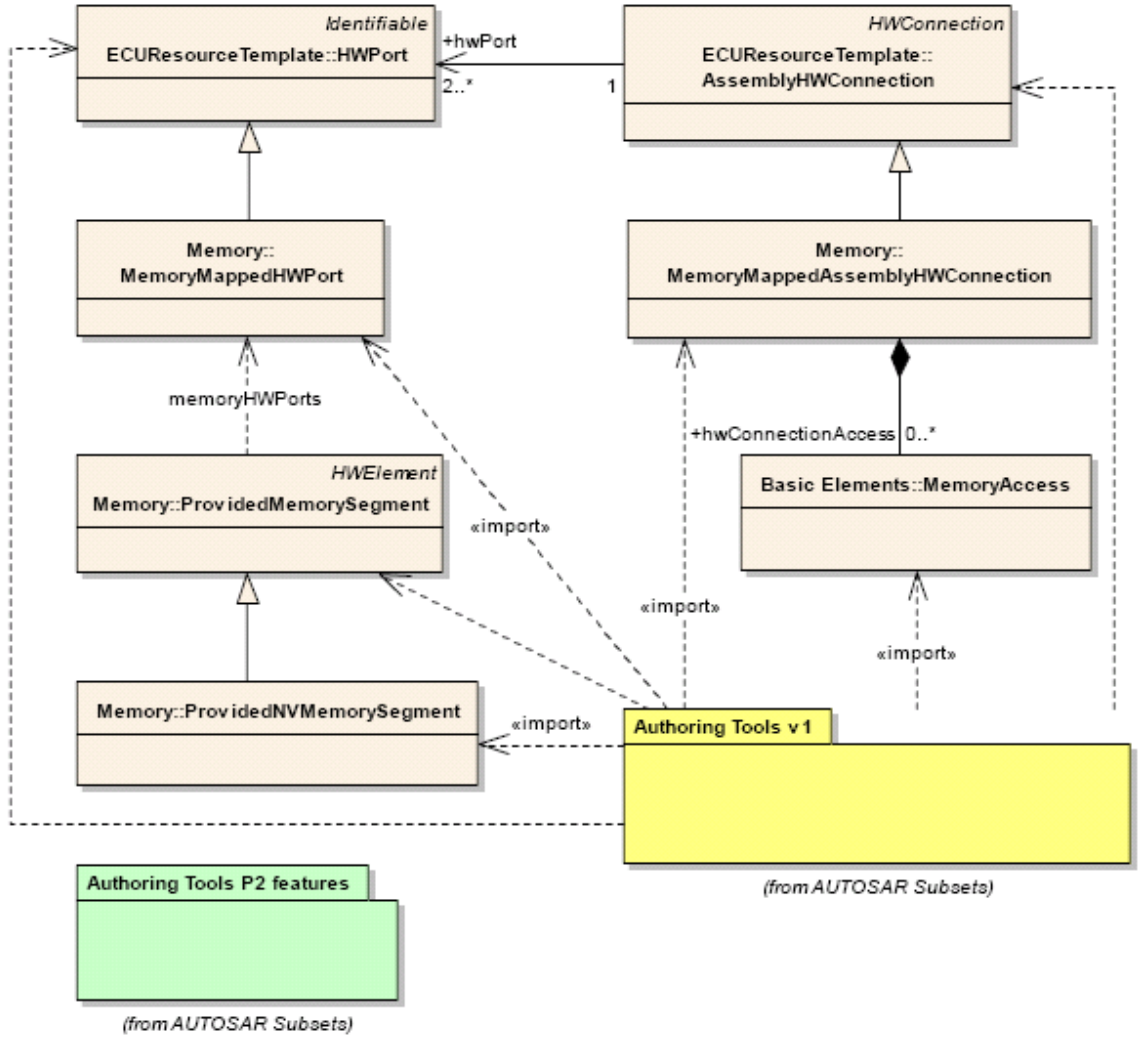

Figure 7-9: Annotation of the meta-model for the description of available memory

It is obvious that the decision about the membership of this feature in the AUTOSAR subset for Authoring Tools V1.0 is strongly interlinked with the specification of memory resource requirements of software-components (ARSubset0016).

7.10 [ARSubset0030] ECUs with multiple Processing Units

Short description				
The ECU Resource Template allows for the definition of more than a single `ProcessingUnit` mounted to a specific ECU.				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0023	--	medium	medium	P3

The usage of ECUs with more than a single `ProcessinUnit` is not exactly an exotic case in typical automotive systems. Nevertheless, it can be safely neglected for the

first implementation of AUTOSAR Authoring Tools because its importance for the evaluation of the methodology is rather low. Consequently, this feature is given a P3.

8 Assessment of System Template

The System Template describes mainly communication aspects (including the definition and usage of particular bus types, frame characteristics, etc.) and the mapping of application software to specific hardware nodes.

8.1 [ARSubset0054] Support for CAN

Short description				
Use CAN bus for describing a communication connection in a system description				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0037, ARSubset0041	low	medium	P1

The CAN bus certainly is the most used communication bus in contemporary automotive systems. It is therefore without any question justified to make the general support of CAN a P1 feature.
Technically, the `CANBus` is derived from the abstract meta-class `CommunicationCluster`. The latter contains a description of `PhysicalChannels` that in turn consist of a description of `PhysicalMediumSegments`. Each `PhysicalMediumSegment` references a collection of `ECUCommunicationPortInstances`, in the case of CAN a `CANCommunicationPortInstance` is actually referenced.

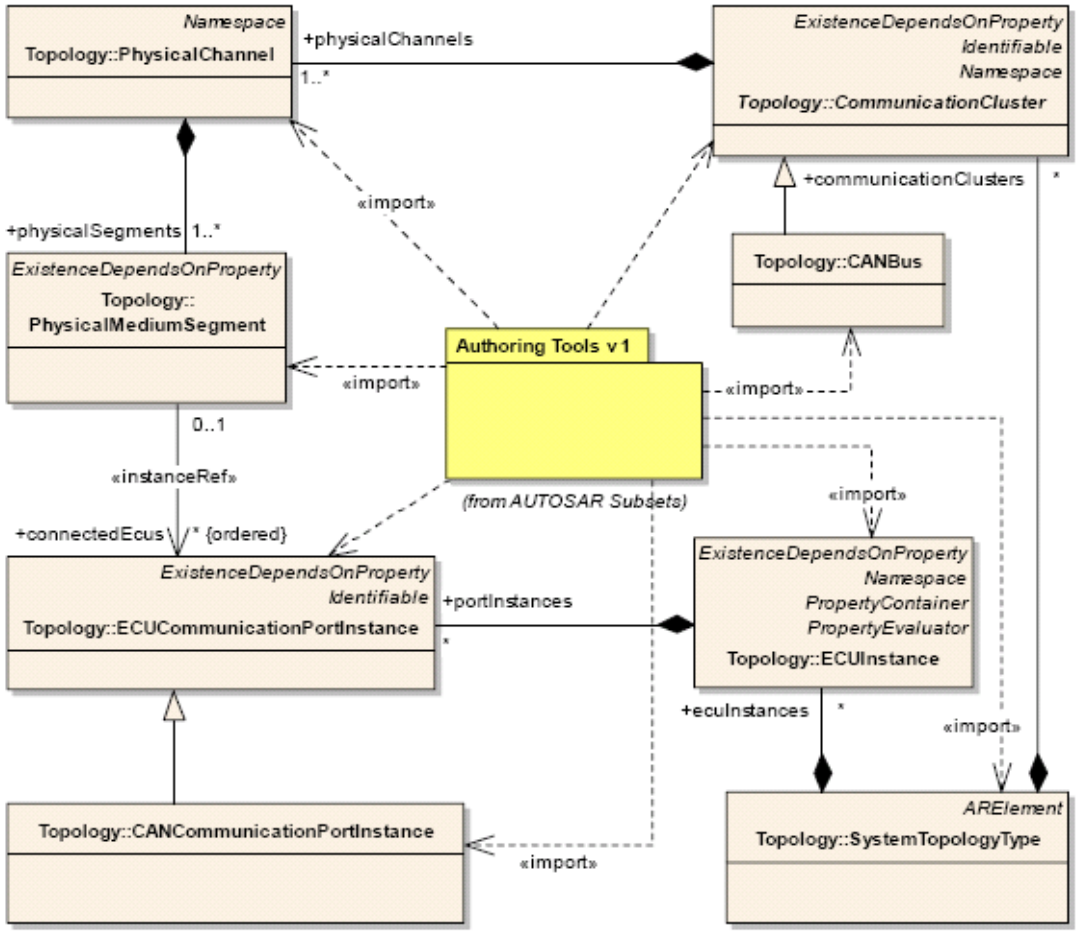

Figure 8-1: Annotation of the meta-model to support CAN buses

This relationship is depicted in Figure 8-1. The consideration of frame-related information is documented in ARSubset0041.

8.2 [ARSubset0055] Support for FlexRay

Short description				
Use the FlexRay bus for describing a communication connection in a system description				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0041	low	medium	P2

Although FlexRay has been identified as an important feature the support within the work groupp for the provision of this feature even for the first implementation of AUTOSAR tools was not very strong. Therefore the feature tentatively has been given a P2.

The meta-classes required to support FlexRay in general are very similar to the meta-classes required for CAN (as described in ARSubset0054). That does not mean that the implementation of this feature is similar to ARSubset0054.

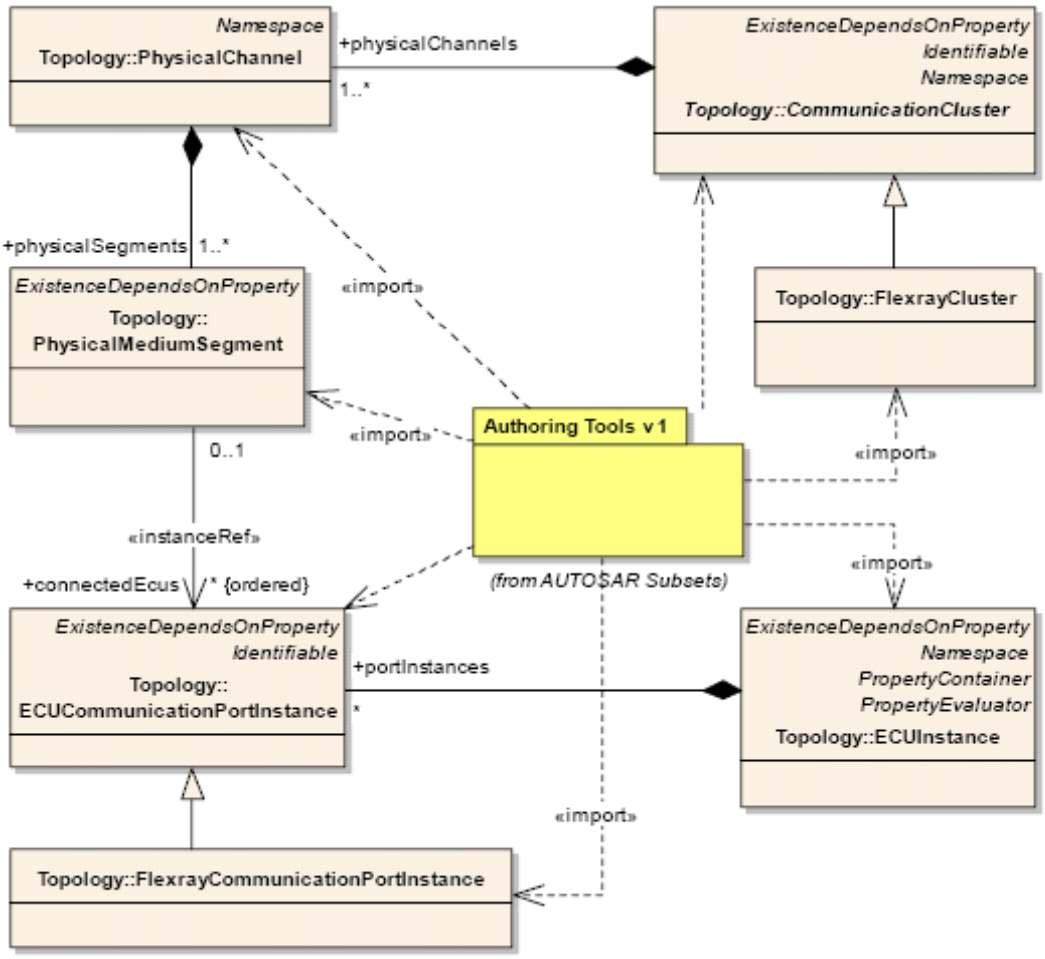

Figure 8-2: Annotation of the meta-model for the support of FlexRay

Only the `CANBus` that is depicted in Figure 8-1 would be replaced by a `Flexray-Cluster` (see Figure 8-2). Again, the consideration of frame-related information is documented in ARSubset0041.

8.3 [ARSubset0056] Support for LIN

Short description				
Use the LIN bus for describing a communication connection in a system description				
Initiator				
Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0041, ARSubset0042	low	medium	P1

Like in the case of CAN, support for LIN is certainly required for the first implementation of AUTOSAR Authoring Tools. Again, the annotation of the meta-model (see Figure 8-3) is very similar to the support for CAN (ARSubset0054) and FlexRay (ARSubset0055).

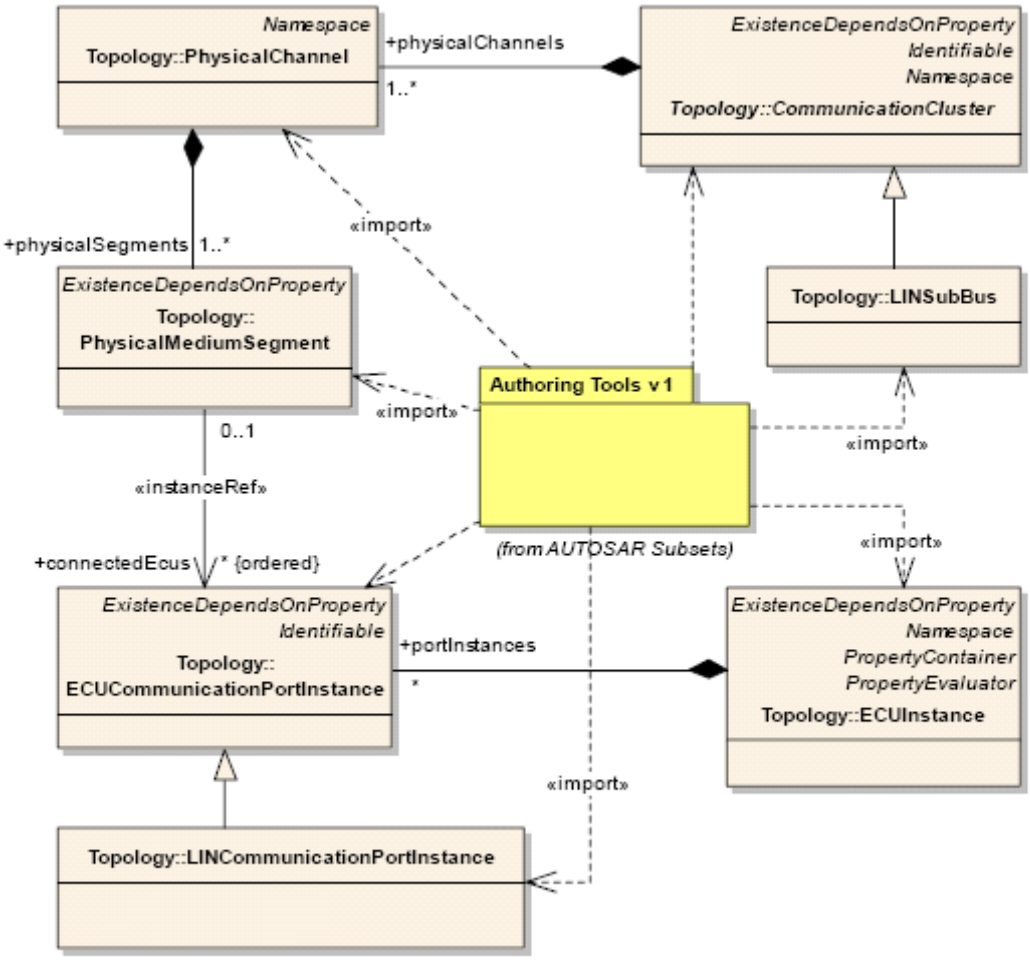

Figure 8-3: Annotation of the meta-model for the support for LIN

And again, the meta-model annotation does not mean that the implementation can be carried out along the CAN example.

8.4 [ARSubset0031] Modeling of buses

Short description				
Modeling of buses by referring ECUs				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	ARSubset0032, ARSubset0036, ARSubset0041, ARSubset0044, ARSubset0061	low	medium	P1

This feature is mainly about the description of physical interconnections of bus segments. For example, a Hub could or could not be used to model a specific communication bus.

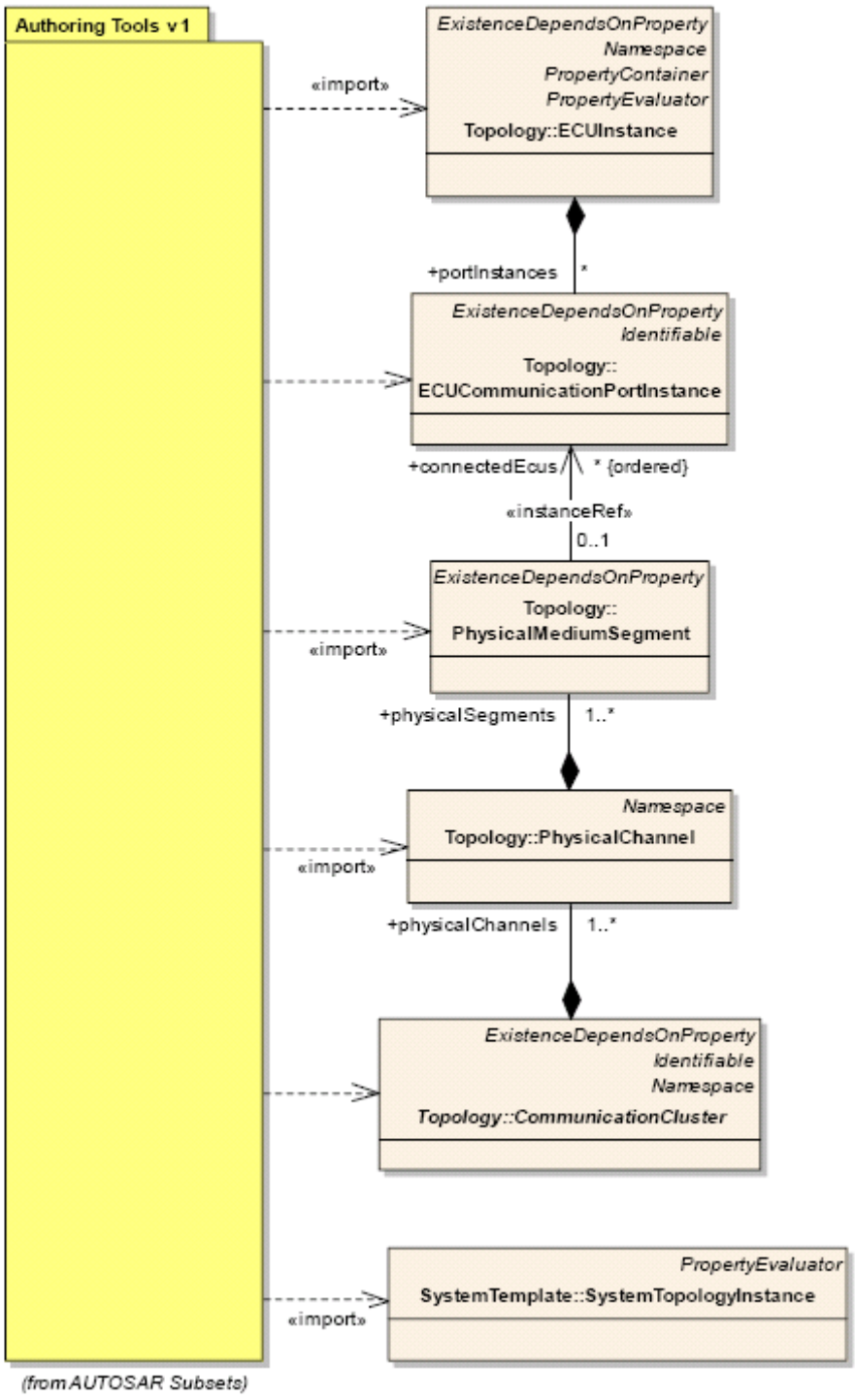

Figure 8-4: Annotation of the meta-model for the support of buses and simple topologies

This feature is the basis for the description of a distributed system. It is therefore inevitable for the first implementation of AUTOSAR Authoring Tools.

8.5 [ARSubset0061] Modeling of simple topologies

Short description				
Modeling simple topologies by means of Hubs.				
Initiator				
WP1.2				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0031	ARSubset0044	medium	medium	P2

Hubs are used to create simple topologies of buses, i.e. bus segments are simply connected to each others by means of a Hub. Please note that this feature does not support the creation of star etc. by means of a Hub.

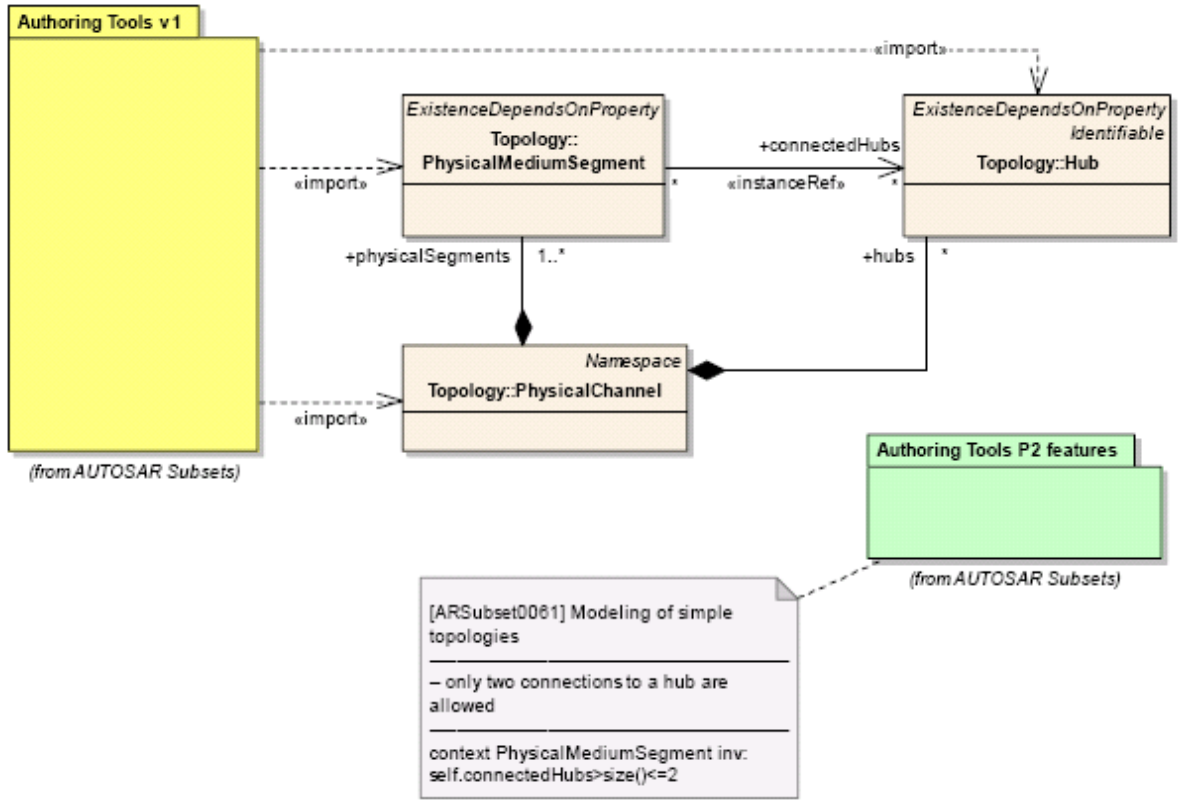

Figure 8-5: Annotation of the meta-model for the support of simple topologies

Therefore, the annotation of the meta-model is accompanied by a constraint concerning the number of PhysicalMediumSegments that reference a specific Hub.

8.6 [ARSubset0032] Component mapping

Short description				
Assignment of software-components (or more specifically, a combination of a `ComponentPrototype` and an `Implementation`) to a specific `ECUInstance`				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0031	ARSubset0033, ARSubset0043	high	high	P1

This is an essential part of AUTOSAR; therefore it is a P1 feature. Risk and effort are rated as high, since the component mapping has a lot of influence on other parts of the model, esp. the communication. Various consistency conditions in the data model have to be ensured.
The annotation of the meta-model is more or less trivial: the meta-class `SwCompToEcuMapping` references the already mentioned meta-classes `Implementation`, `ComponentPrototype`, and `ECUInstance` (onto which software-components can be mapped).

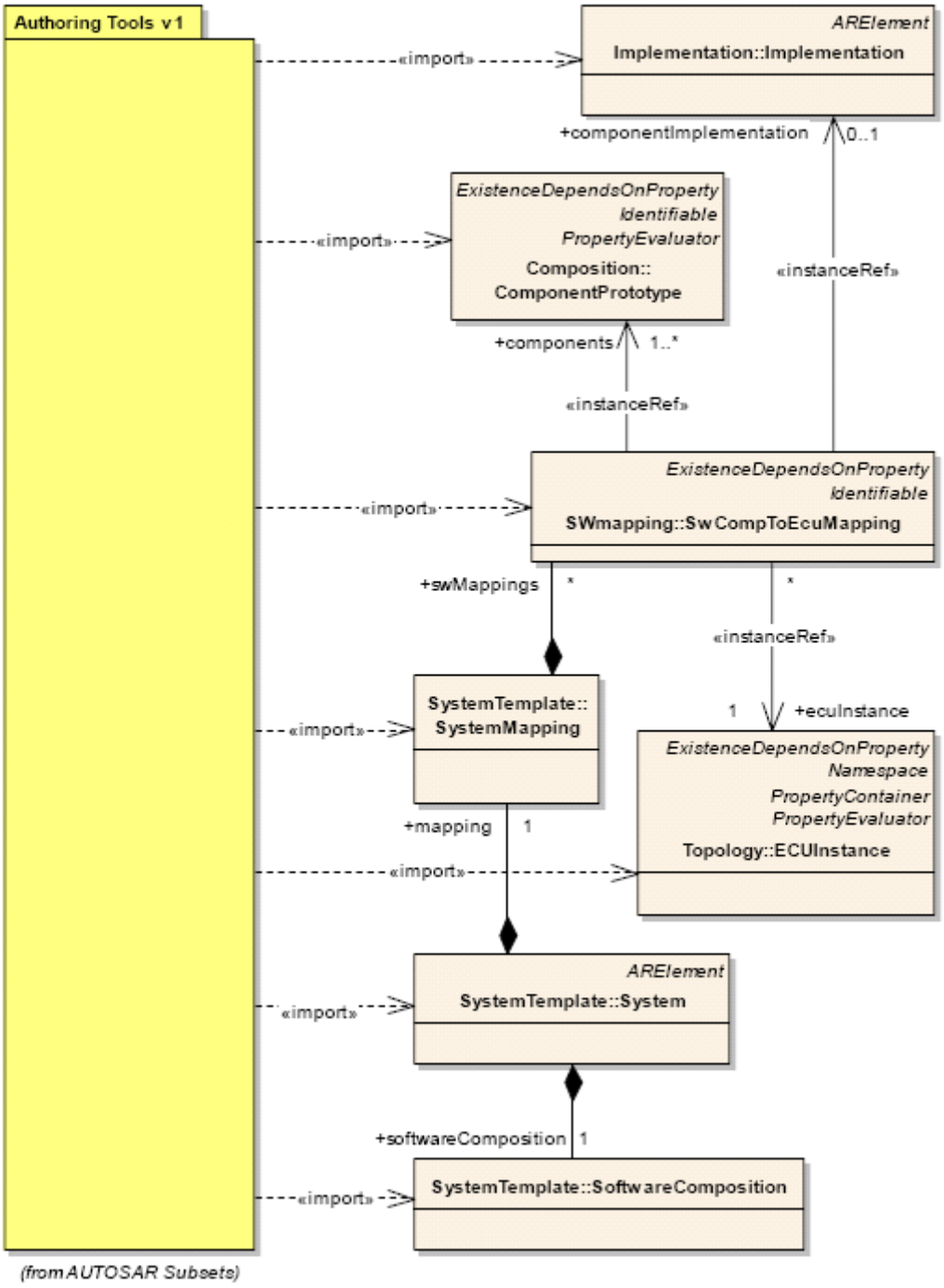

Figure 8-6: Annotation of the meta-model for the support of component mapping

The `SoftwareComposition` represents the instance of a `CompositionType` that actually makes up the top-level vehicle software architecture. This is the root for all component mapping activities.

8.7 [ARSubset0033] Communication mapping

Short description				
Definition of the bus communication (frames, frame layout)				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0032	ARSubset0034, ARSubset0035, ARSubset0037, ARSubset0038, ARSubset0039, ARSubset0040	low	medium	P1

The dependency on the component mapping mainly exists, if a top-down development process has to be supported. In this case, the bus communication depends on the interfaces and distribution of the components.

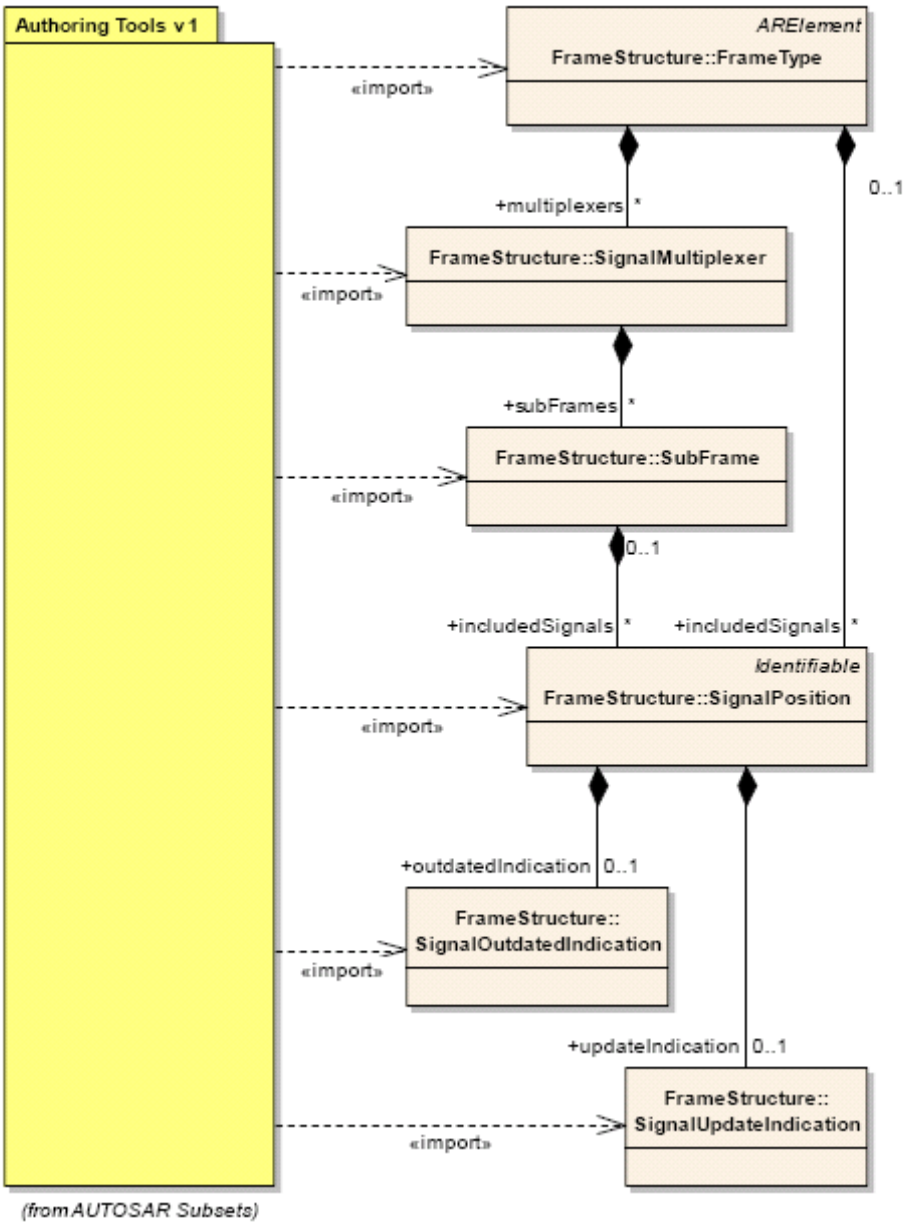

Figure 8-7: Annotation of the meta-model for the support of communication mapping (1)

This feature is (in terms of meta-classes) rather complex. The depiction of the necessary annotation of the AUTOSAR meta-model is therefore distributed over Figure 8-7 and Figure 8-8.

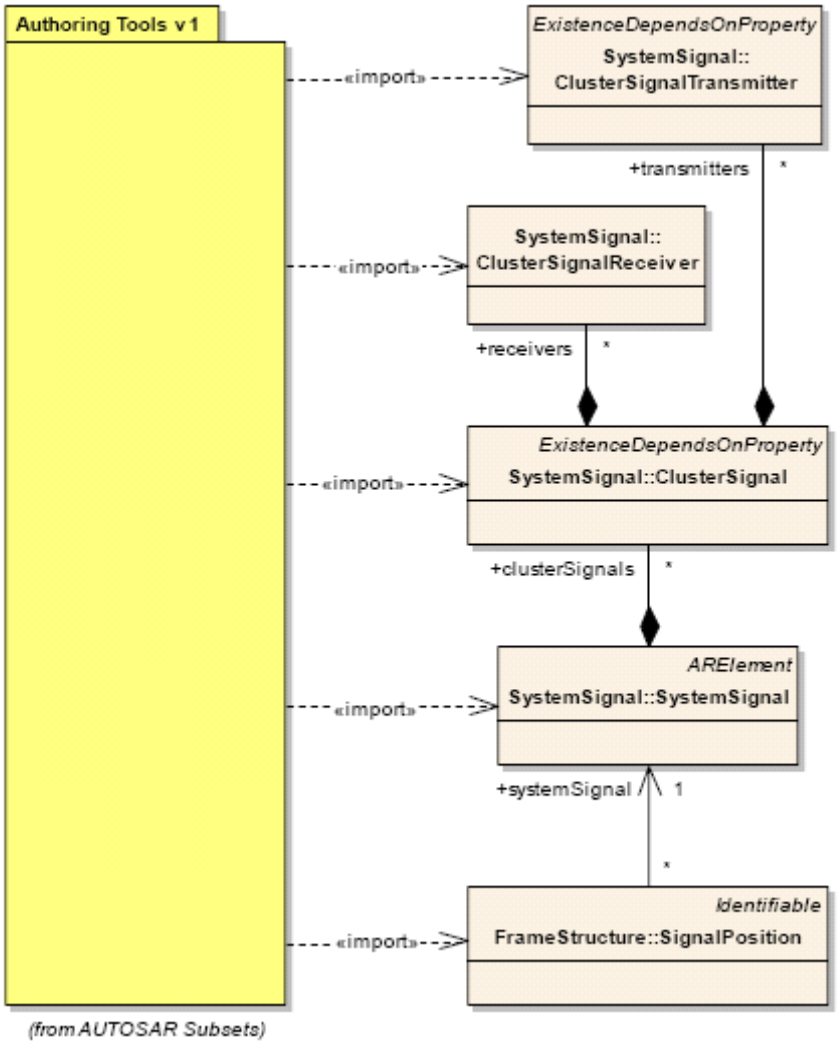

Figure 8-8: Annotation of the meta-model for the support of communication mapping (2)

Please note that the `SystemSignal` as well as the `ClusterSignal` has recently become a very central concept of the System Template. This fact is emphasized mainly in Figure 8-8.

8.8 [ARSubset0034] Data mapping

Short description				
Installs the link between network signals and component interfaces				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033	--	high	medium/-high	P1

The complexity of this feature depends on the communication patterns, which have to be supported. The complexity is not very high for sender-receiver pattern, but quite high for client-server pattern.

The risk is rated as high, since it is the “central link” between SW structure design and communication design. Therefore, the data mapping must ensure various consistency conditions.
Both alternative ways to design the data mapping have to be supported: a “forward way” (especially requested by one Tier-1 supplier stakeholder) from SW structure to communication design is required for a top-down development process; a “reverse way” from communication design to SW structure design is required for a bottom-up process.
Please note that `SenderReceiverToProtocol` and `ClientServerToProtocolMapping` mapping is not part of the subset because modeling of protocols in general is not considered (ARSubset0039).

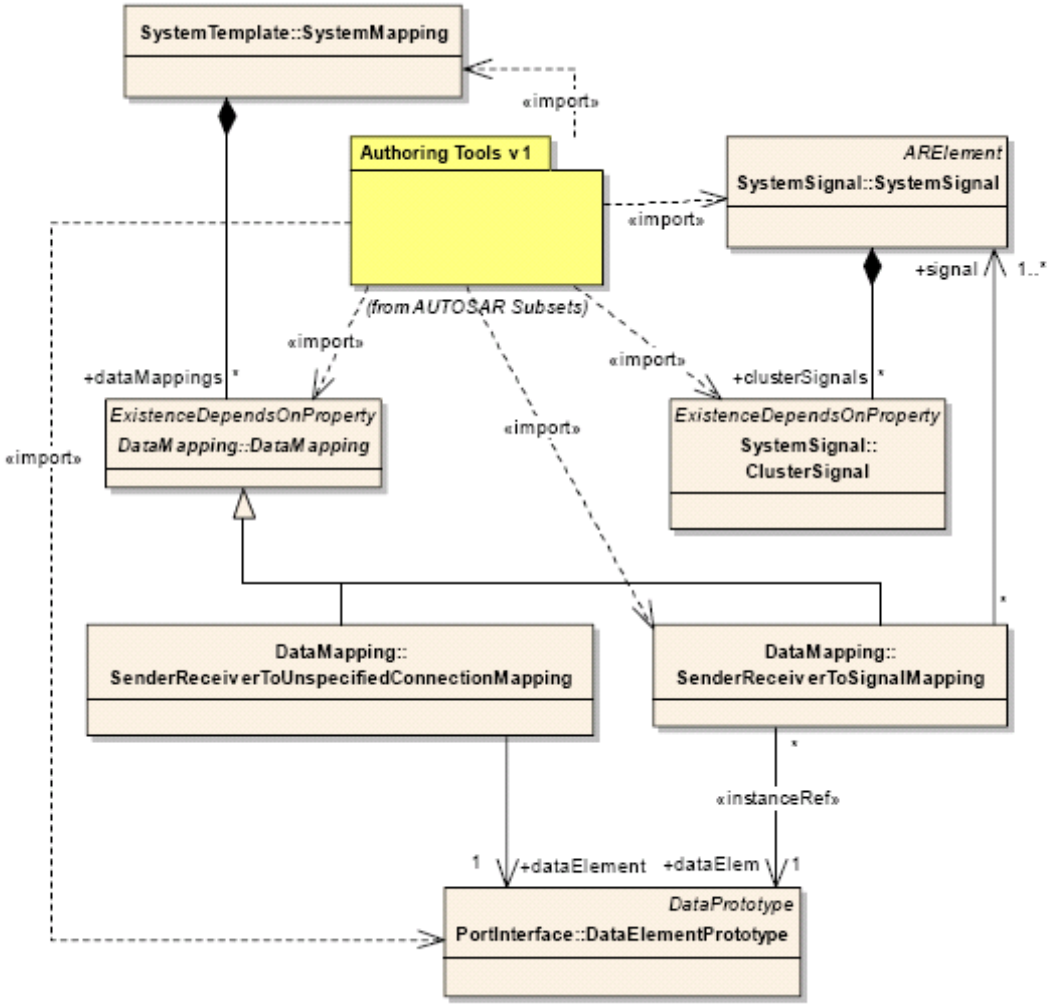

Figure 8-9: Annotation of the meta-model for the support of data mapping

On the other hand, the `SenderReceiverToUnspecifiedConnectionMapping` is considered because it is used for a direct interconnection of ECUs with an unspecified protocol, a feature that is obviously considered important by the OEMs.

8.9 [ARSubset0035] Frame timing

Short description				
Defines the dynamic behavior of bus frames.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033	--	medium	medium	P1

The specification of the frame timing may be done in a local scope of a frame. Nevertheless, the consistency of the frame timing with the port attributes (signal age) must be ensured.
The `UnspecifiedTiming` meta-class is tentatively considered in the feature set.
The `RequestControlledTiming` is currently not covered by a dedicated use case as well and will therefore not be considered (see Figure 8-10). Without question, the `EventControlledTiming` and `CyclicTiming`, on the other hand, represent standard use cases for bus communication. Therefore, the meta-classes are annotated for being members of the feature set of AUTOSAR Authoring Tools V1.0.
The `StartCondition` and `ActiveCondition` aggregated at the `CyclicTiming` meta-class is intentionally excluded from the feature set because the consideration of both meta-classes only makes sense in combination with a suitable mode-management concept (which, in turn, is **not** supported, see ARSubset0014).
Of course, the consideration of `FrameInstance` and `FrameType` is self-evident for the support of frame timing in AUTOSAR Authoring Tools V1.0.

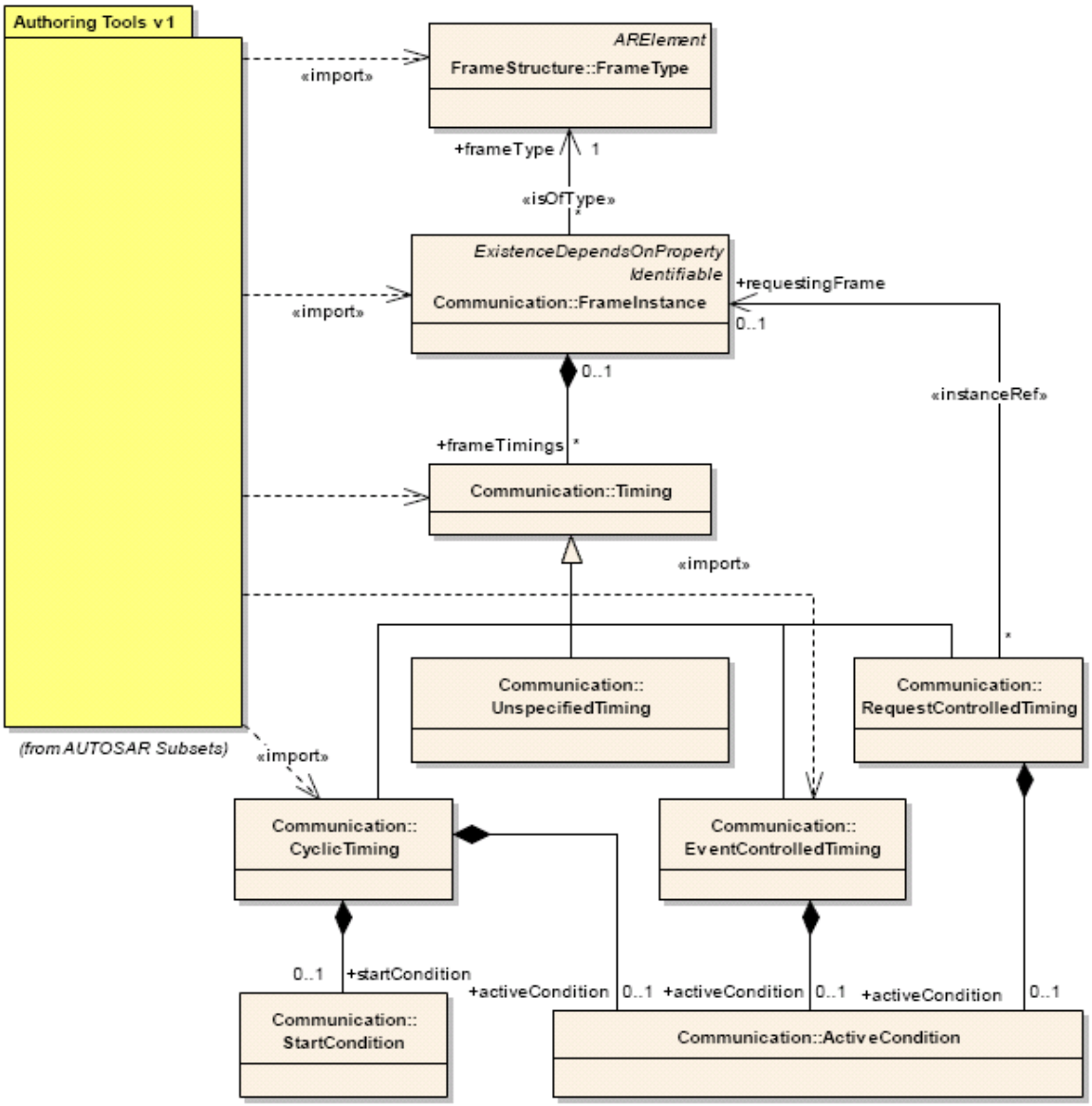

Figure 8-10: Annotation of the meta-model for the support of frame timing

Please note that although this feature has been given a P1 priority it will **not** be included in the feature subset. This decision has been taken in the course of harmonizing this document with the "overall non-basis software feature list".

8.10 [ARSubset0037] Multiplexed signals

Short description				
Multiplexing is used to transport different signals at the same position of a single frame.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033, ARSubset0054	--	medium	medium	P2

The meta-class `SignalMultiplexer` represents the possibility to transmit different signals at the same position within a `FrameType` depending on the value of the multiplexer.
Please note that although this feature has been considered to have priority P2 it is obvious (see Figure 8-11 for more details) that all required meta-classes are already covered by P1 features (especially ARSubset0033). It is therefore not necessary to add any further annotation for "Authoring Tools P2 features".

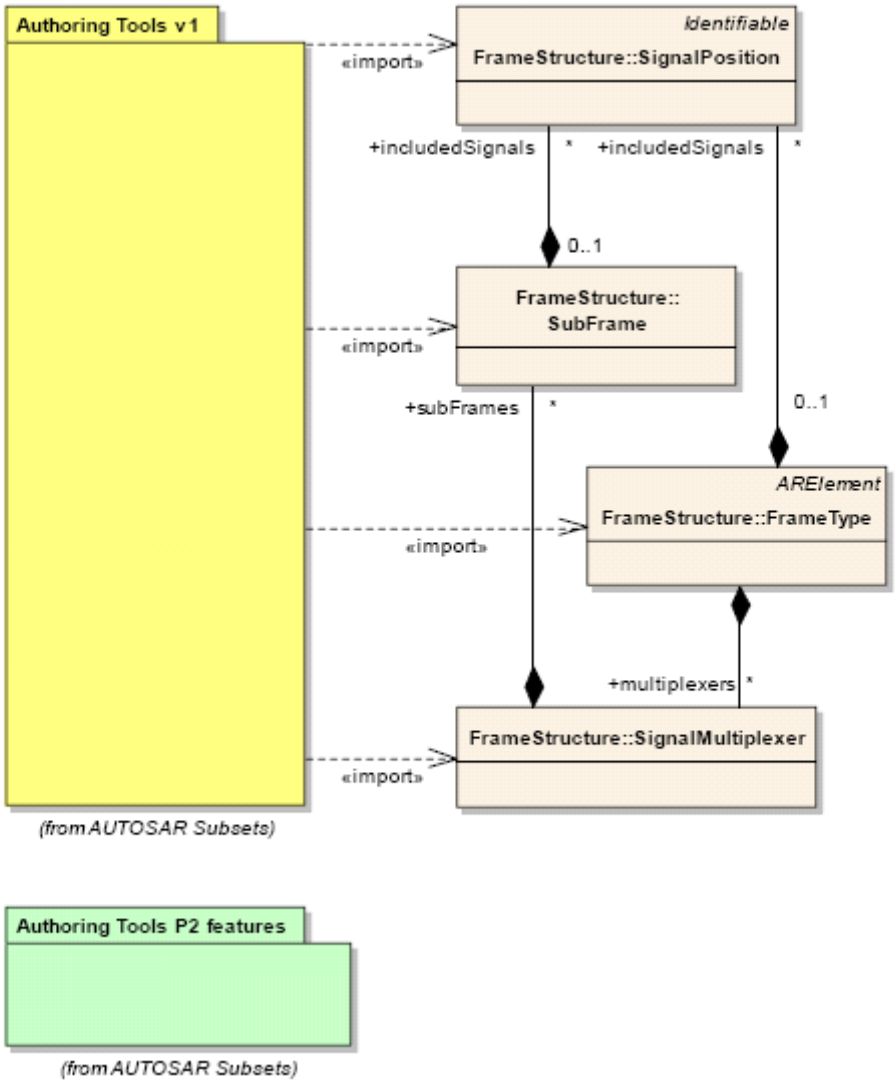

Figure 8-11: Annotation of the meta-model for the support of signal multiplexing

Please note that the fact that a tool supports all necessary meta-classes anyway does not automatically mean that the corresponding feature is supported as well. The tool obviously needs to introduce the feature-specific usage of the meta-classes plus the user interface in order to provide users with means to actually use a feature.

8.11 [ARSubset0038] Gateway functionality

Short description				
Specification of the functionality of gateway ECUs in terms of gateway table entries with according timing specification.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033	--	medium	medium	P1

Please note that the definition of gateway functionality (as expressed in terms of meta-model annotation in Figure 8-12) is not to be confused with the topological definition of gateway ECUs! The latter feature is covered by ARSubset0031.

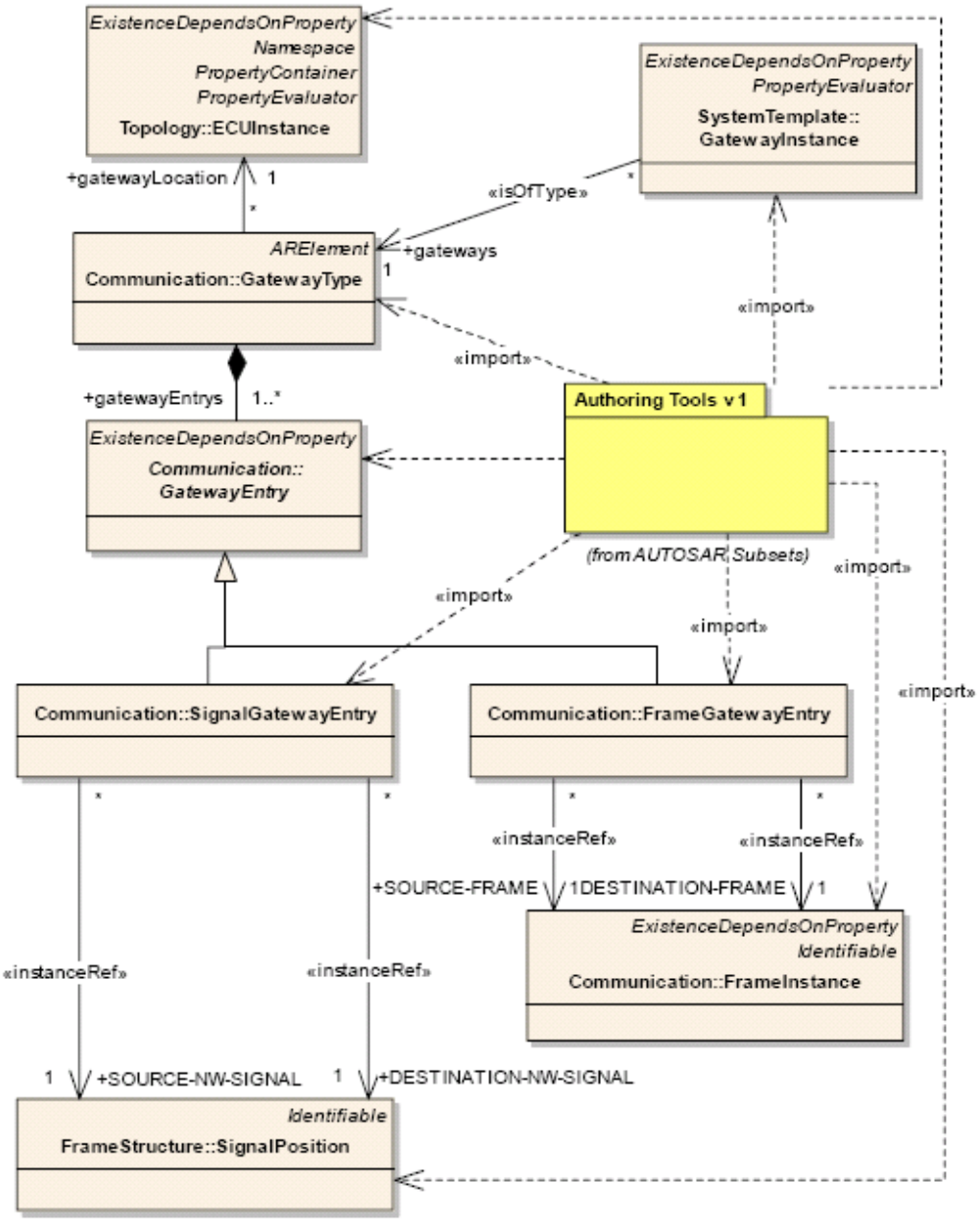

Figure 8-12: Annotation of the meta-model for the support of gateway functionality

Gateways can be defined both on the level of frames (i.e. `FrameGatewayEntry`) and on the level of signals (i.e. `SignalGatewayEntry`). In real life, gateways usually implement a mixture of both concepts. It does therefore not make sense to define separate feature descriptions for frame-based and signal-based gateway functionality.

In the meta-model, the gateway concept has been introduced following the type-instance-pattern, i.e. the meta-classes `GatewayType` and `GatewayInstance` are connected by a relationship with stereotype `<<isOfType>>`. Both meta-classes are imported in the package "Authoring Tools v1".

8.12 [ARSubset0039] Protocol modeling

Short description				
Explicit modeling of communication protocols (e.g. TP) with reference to the affected network signals and their role within the protocol.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033	--	high	medium	P3

The arbitrary definition of communication protocols is certainly a novel technology that is not used in current automotive design environments. Therefore, the risk has to be rated as high.

8.13 [ARSubset0040] Frame instance vs. frame type

Short description				
Definition of frame types with re-usable signal layout.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0033	--	medium	medium	P3

This feature can, for example, be used for defining frame gateways. Please note that although this feature is considered to have only priority P3 both `FrameInstance` and `FrameType` are already considered for the annotation of the meta-model.
These meta-classes are required e.g. for the explicit support of frame timing (covered by feature ARSubset0035).

8.14 [ARSubset0041] Bus-specific frame properties

Short description				
Each bus-type (e.g. CAN, LIN, FlexRay, etc.) provides a different paradigm for assembling bus frames. This feature represents the capability of Authoring Tools to deal with different frame structures according to the bus type.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0031, ARSubset0054, ARSubset0055, ARSubset0056	ARSubset0042	low	low	P1

Currently, AUTOSAR supports bus-specific frame properties only for CAN (i.e. `CanFrameInstance`), LIN (i.e. `LinFrameInstance`), and FlexRay (i.e. `FlexrayFrameInstance`).

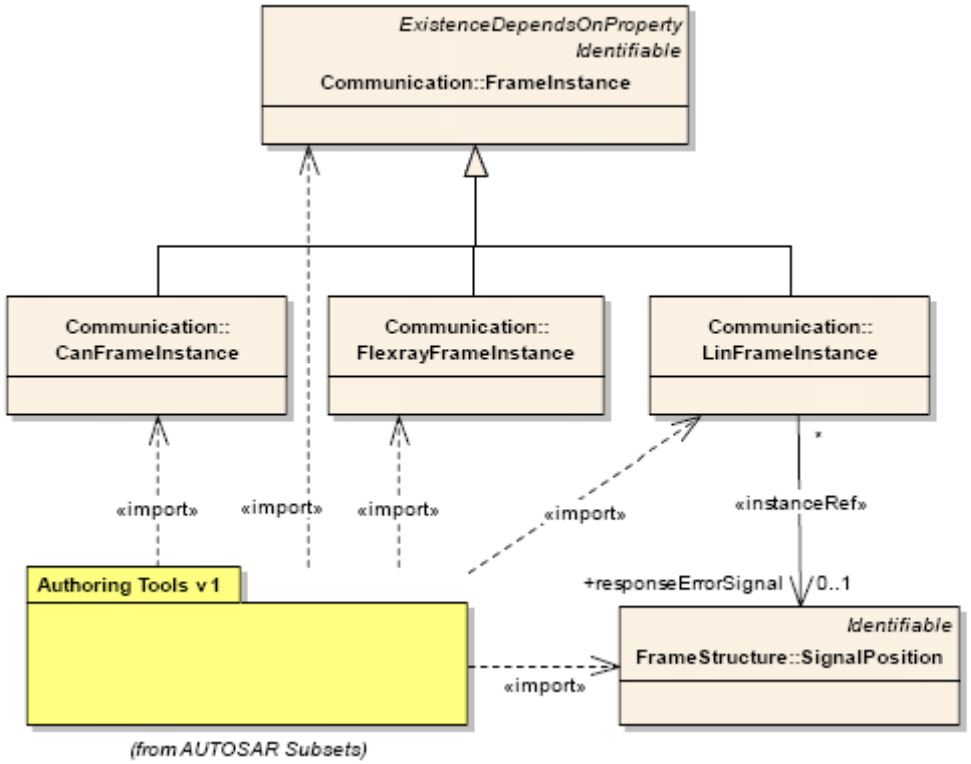

Figure 8-13: Annotation of the meta-model for the support of bus-specific properties

The annotation of the meta-model for the support of bus-specific frame properties is depicted in Figure 8-13.

8.15 [ARSubset0042] LIN scheduling table

Short description				
Definition of the message scheduling on a LIN bus				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0041, ARSubset0056	--	low	high	P1

The `LinSchedulingTable` consists of `LinSchedulingTableEntries` that can either be sent unconditionally (`UnconditionalFrameSlot`), in response to an event (`EventTriggeredFrameSlot`), or sporadically (`SporadicFrameSlot`).
Please note that, according to [7] (Restr_AR1.0_00006 and Restr_AR1.0_00007), the first implementation of AUTOSAR will be restricted to the support of LIN2.0. According to [7], On the other hand, LIN will first appear in release 2 of the AUTOSAR basic software implementation.

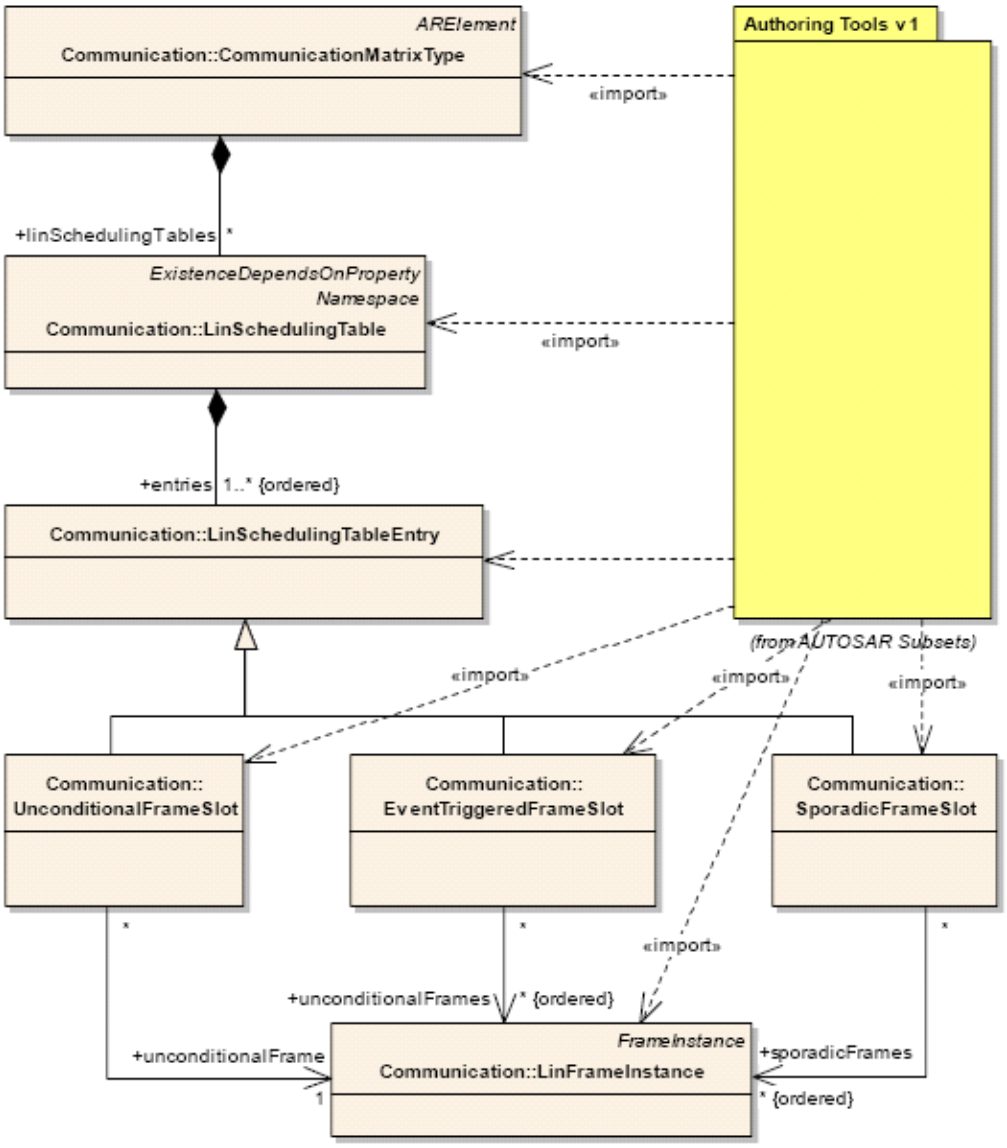

Figure 8-14: Annotation of the meta-model for the support of LIN scheduling tables

The effort for the realization of this feature is estimated as high, since the definition of a LIN schedule table requires a sophisticated tool support.

8.16 [ARSubset0043] Component mapping constraints

Short description				
Definition of constraints (cluster, separation, etc.) for the assignment of components to ECUs.				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0032, (ARSubset0044)	--	low	medium	P3

The dependency on ARSubset0044 only exists, if mapping constraints regarding the bus topology have to be defined (e.g. “Component1 and Component2 must be mapped to bus segments, which are separated by an active hub”).

8.17 [ARSubset0044] Modeling of complex bus topologies

Short description				
Refined definition of bus topologies like RING or STAR				
Initiator				
Matthias Wernicke (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0061	(ARSubset0043)	medium	medium	P3

The effort for this issue depends on the requirements on the design tools (e.g. graphical editor required?)
Note: The definition of bus communication is possible also without detailed modeling of the bus topology.

8.18 [ARSubset0059] Unspecified connection

Short description				
The System Template explicitly supports an `UnspecifiedConnection`				
Initiator				
Mark Brörkens (Carmeq)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P3

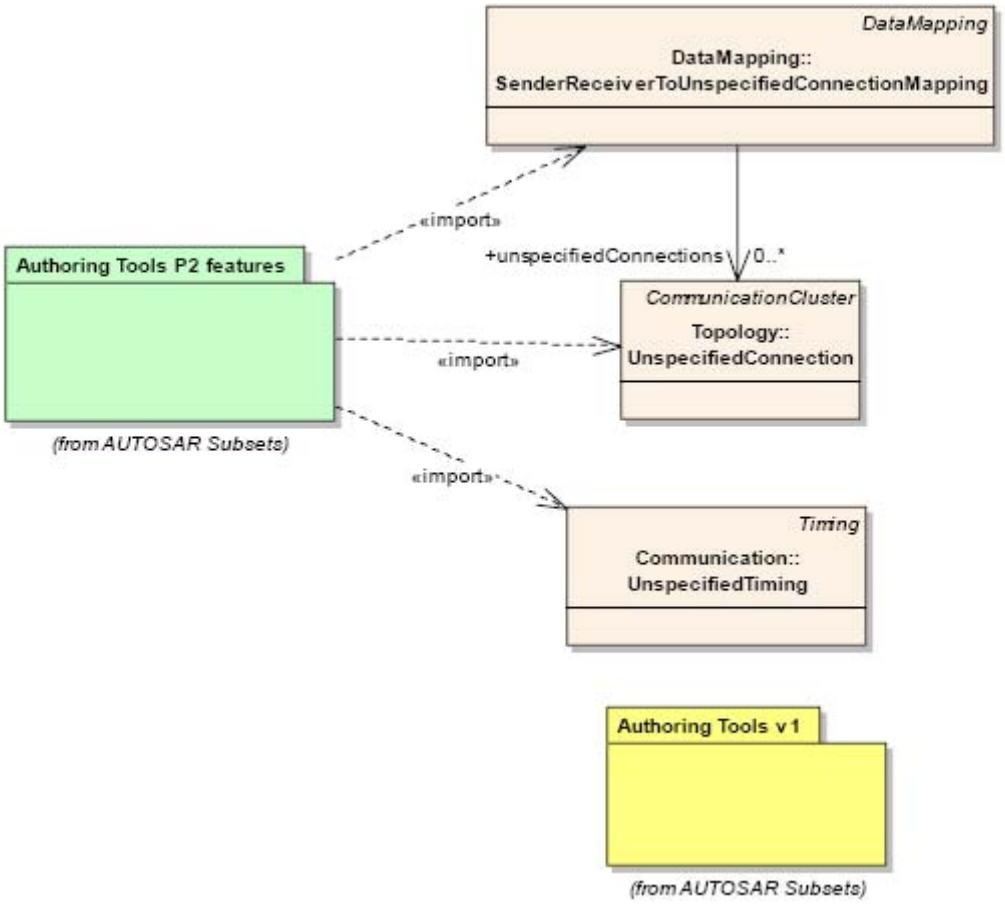

Figure 8-15: Annotation of the meta-model for the support of unspecified connection

This is used for directly connecting ECUs (by a simple wire) e.g. redundancy purposes

8.19 [ARSubset0060] Communication matrix

Short description				
The System Template explicitly supports a `CommunicationMatrixType`				
Initiator				
Mark Brörkens (Carmeq)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
--	--	medium	medium	P1

The meta-class CommunicationMatrixType allows for a re-usage of a specific communication matrix among different projects.

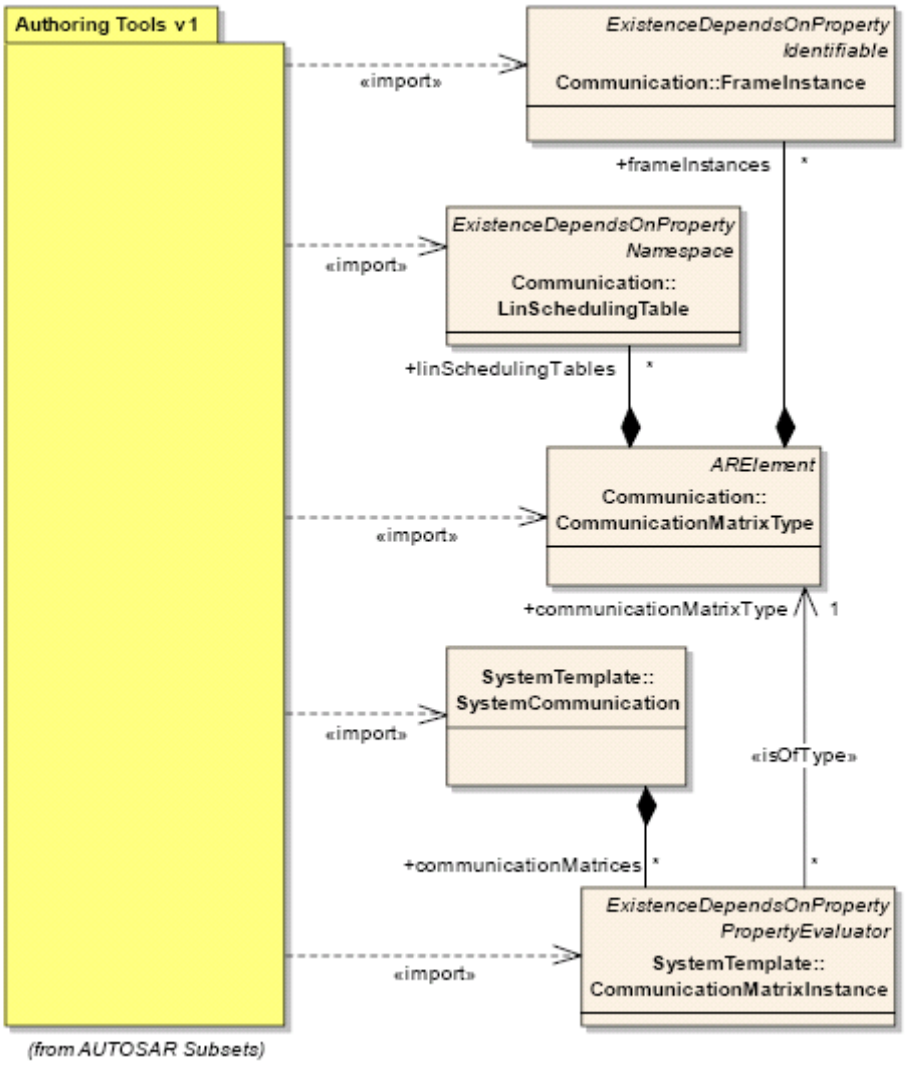

Figure 8-16: Annotation of the meta-model for the support of communication matrix

9 Miscellaneous

9.1 [ARSubset0045] Coupling of behavioral models

Short description				
Provide interfaces to graphical behavior modeling tools like MATLAB/Simulink, ASCET, etc.				
Initiator				
Matthias Wernicke, Uwe Honekamp (Vector)				
Depends on	**Is prerequisite for**	**Risk**	**Effort**	**Priority**
ARSubset0006	--	medium	high	P1

The implementation of the behavior of a combination of a `AtomicSoftwareComponentType` and an `Implementation` by means of high-level behavioral modeling tools is supposed to be accepted among the stakeholders of a typical AUTOSAR development project.
The main use cases in this context are the creation of model frames on the basis of a software-component description or, the reverse direction, the "conversion" of behavioral models to AUTOSAR software-components.

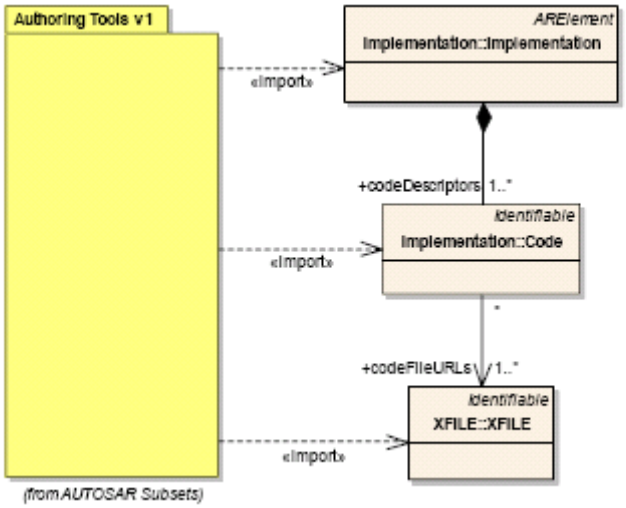

Figure 9-1: Annotation of the meta-model for the coupling of behavior models

Technically, it is possible to attach multiple files to a specific `Code`. Furthermore, the meta-class `XFILE` allows for the specification of a responsible `TOOL` (in this case: Simulink, ASCET, etc.) and a `NOTATION` (i.e. the file format, e.g. Simulink model file).

10 Definition of the AUTOSAR subset for first implementation

This chapter represents a definition of membership for particular features in a meta-model subset for a first implementation of AUTOSAR Authoring Tools. All features with priority P1 are members of the subset anyway.
It is clear that P3 features will certainly not be considered. P2 features, on the other hand, have been assessed on an individual basis. In other words: a part of the P2 features is included in the subset (please consult Figure 10-3 for more details).
Please note that Table 4 intentionally does not contain a line representing ARSubset0036 because this feature is dismissed.

ID	Priority	Feature	In Subset
ARSubset0001	P3	Properties	N
ARSubset0002	P2	Grouping	N
ARSubset0003	P1	Simple data types	Y
ARSubset0004	P2	Package	Y
ARSubset0005	P1	Interface	Y
ARSubset0006	P1	Software-component	Y
ARSubset0007	P1	Composition	Y
ARSubset0008	P1	Multiple instantiation	Y
ARSubset0009	P1	Delegation/assembly connector	Y
ARSubset0010	P1	Sender/receiver relationship for data	Y
ARSubset0011	P3	Application-level client/server relationship	N
ARSubset0012	P2	Client/server relationship to RTE and basic software	Y
ARSubset0013	P3	Data variant management	N
ARSubset0014	P3	Mode-management	N
ARSubset0015	P1	Simple data semantics	Y
ARSubset0016	P2	Memory resource consumption	Y
ARSubset0017	P1	Coupling to sensors and actuators	Y
ARSubset0018	P1	Runnable entities	Y
ARSubset0019	P1	Buffered sending and receiving of data elements	Y
ARSubset0020	P1	Explicit sending of data elements	Y
ARSubset0021	P2	Exclusive area	Y
ARSubset0022	P3	Execution order of runnable entities	N
ARSubset0023	P1	Definition of ECUs	Y
ARSubset0024	P1	Definition of communication peripherals	Y
ARSubset0025	P1	Definition of I/O peripherals	Y
ARSubset0026	P2	Definition of ECU electronics	N

ID	Priority	Feature	In Subset
ARSubset0027	P1	Definition of sensors/actuators	Y
ARSubset0028	P3	Signal transformation	N
ARSubset0029	P2	Available memory	Y
ARSubset0030	P3	ECUs with multiple Processing Units	N
ARSubset0031	P1	Modeling of buses	Y
ARSubset0032	P1	Component mapping	Y
ARSubset0033	P1	Communication mapping	Y
ARSubset0034	P1	Data mapping	Y
ARSubset0035	P1	Frame timing[9]	N
ARSubset0037	P2	Multiplexed signals	Y
ARSubset0038	P1	Gateway functionality	Y
ARSubset0039	P3	Protocol modeling	N
ARSubset0040	P3	Frame instance vs. frame type	N
ARSubset0041	P1	Bus-specific frame properties	Y
ARSubset0042	P1	LIN scheduling table	Y
ARSubset0043	P3	Component mapping constraints	N
ARSubset0044	P3	Modeling of complex bus topologies	N
ARSubset0045	P1	Coupling of behavioral models	Y
ARSubset0046	P2	Complex data types	N
ARSubset0047	P3	Execution time resource requirement	N
ARSubset0048	P3	CharType and StringType	N
ARSubset0049	P2	Specification of NVRAM resource consumption	Y
ARSubset0050	P3	Complex data semantics	N
ARSubset0051	P1	RTEEvents	Y
ARSubset0052	P3	Definition of displays	N
ARSubset0053	P3	Sender/receiver relationship for events	N
ARSubset0054	P1	Support for CAN	Y
ARSubset0055	P1	Support for FlexRay	Y
ARSubset0056	P1	Support for LIN	Y
ARSubset0057	P2	Inter-runnable variable	Y
ARSubset0058	P1	Hierarchical hardware description	Y
ARSubset0059	P3	Undefined connection	N
ARSubset0060	P1	Communication matrix	Y
ARSubset0061	P2	Modeling of simple topologies	Y
ARSubset0062	P1	Definition of physical units	Y

[9] This feature is not part of the subset as a measure for consistency with the "overall non-basic software feature list".

ID	Prior-ity	Feature	In Subset
ARSubset0063	P1	Comments	Y

Table 4: Membership in feature collection

It would perhaps be interesting to carry out a trivial statistical analysis of the findings contained in Table 4.

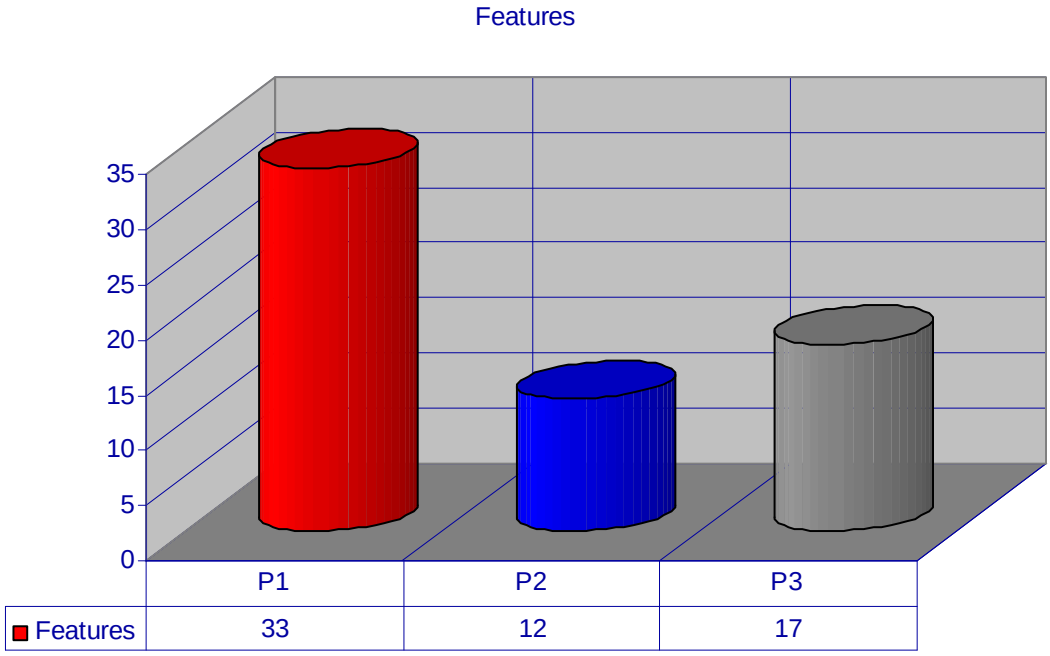

Figure 10-1: Histogram of feature priorities

As indicated by Figure 10-1, the number of P1 features slightly outnumbers the P2 and P3 features. In combination with the ratio of features in the subset compared to features excluded from the subset (see Figure 10-2) this is a sure indication that the subset still implies some complexity.

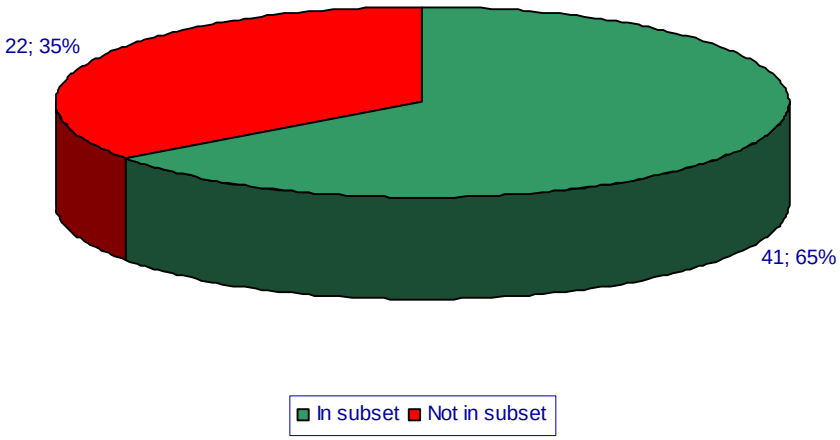

Figure 10-2: Ratio of included and excluded features

In addition, the number of included P2 features is larger than the number of excluded P2 features (as sketched by Figure 10-3).

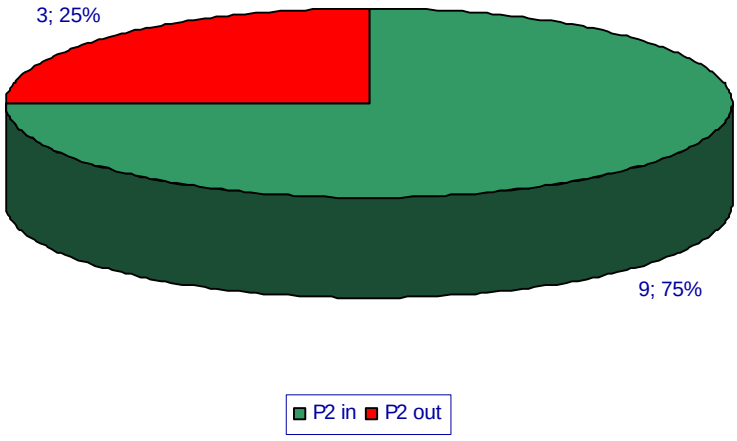

Figure 10-3: ration of included and excluded P2 features

Conclusion: the feature definition for a first implementation of AUTOSAR Authoring Tools still bears a significant amount of complexity.

11 Possible evolution of the feature definition (non-normative)

The definition of features for AUTOSAR authoring tools as described in this document is certainly only the first step on a roadmap of AUTOSAR tools in general.
Therefore, this chapter contains a mere (and perhaps incomplete) collection of scenarios that describe a possible evolution of the feature definition for the period after the features defined in this document have been implemented.

11.1 Scenario 1

In this scenario, the feature definition is extended to cover the whole methodology while the coverage of the meta-model remains as it is. This possible evolution step is depicted in Figure 11-1.

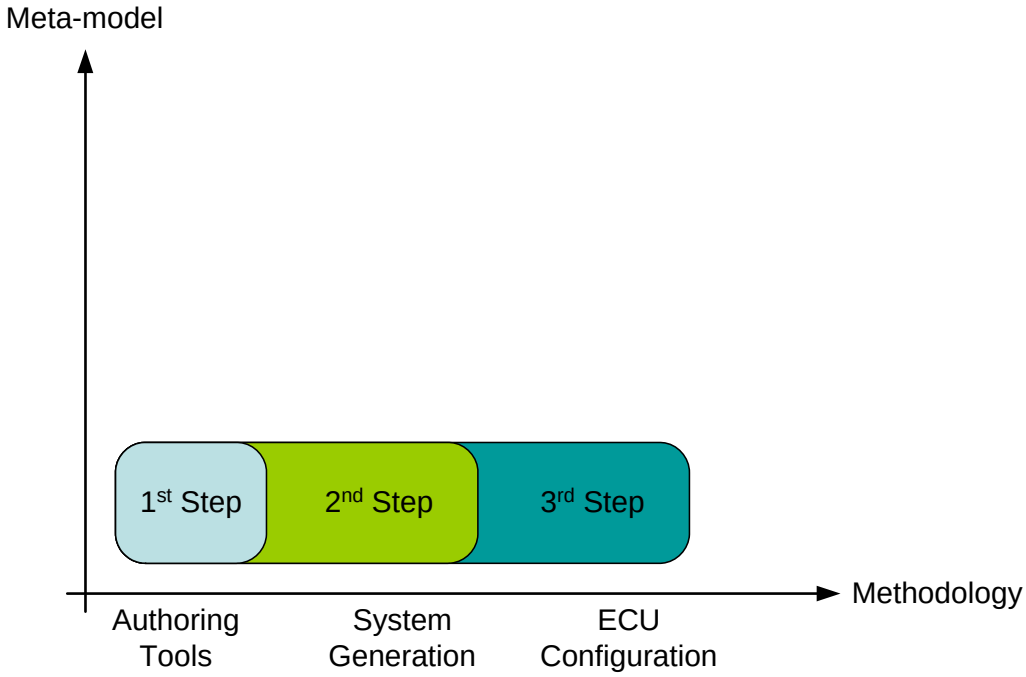

Figure 11-1: graphical sketch of scenario 1

This approach would most likely be taken if the goal is to ensure the applicability of the feature definition for authoring tools over the entire tool chain and methodology. The risk of this approach is supposedly limited since the complexity (in terms of meta-model) is limited as well.

11.2 Scenario 2

Another approach could be to extend the feature set towards a greater coverage of the meta-model but still limit the applicability to authoring tools. In other words: authoring tools would be capable of providing more sophisticated modeling means.
This would certainly contribute to the stability of the meta-model itself since it will be explored very thoroughly in the process of feature definition and annotation.

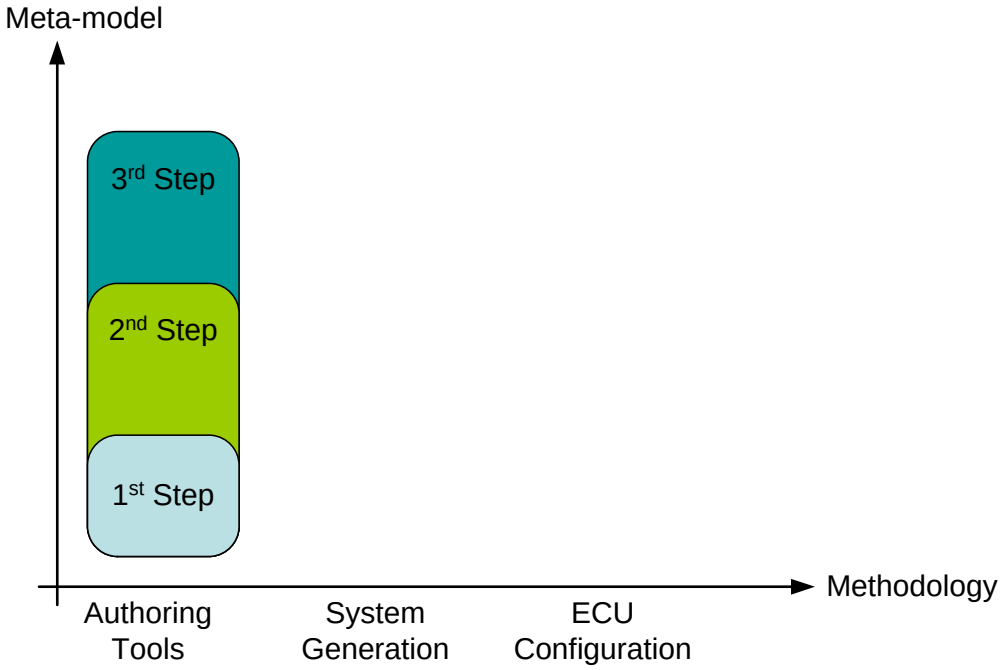

Figure 11-2: Evolution of the feature definition according to scenario 2

Please note that this scenario (as sketched in Figure 11-2) could be implemented as a preparation for scenario 3 (chapter 11.3). In this case the risk should not be too high as well.

11.3 Scenario 3

This scenario is a mere extension of scenario 2 (chapter 11.2). After the meta-model has been explored to a certain extent and corresponding features have been defined for authoring tools the further evolution aims to support the methodology step-by-step.
Nevertheless, this process is carried out in a single step but stepwise as well. Therefore, the next step in the methodology (in this example: system generation) is first addressed with a limited complexity in terms of the meta-model.
The complexity would then be increased stepwise until approximately[10] the same complexity as already achieved for authoring tools is reached. This approach is depicted in Figure 11-3.

[10] The complexity reached after step 6 does not necessarily have to be the same as after step 3.

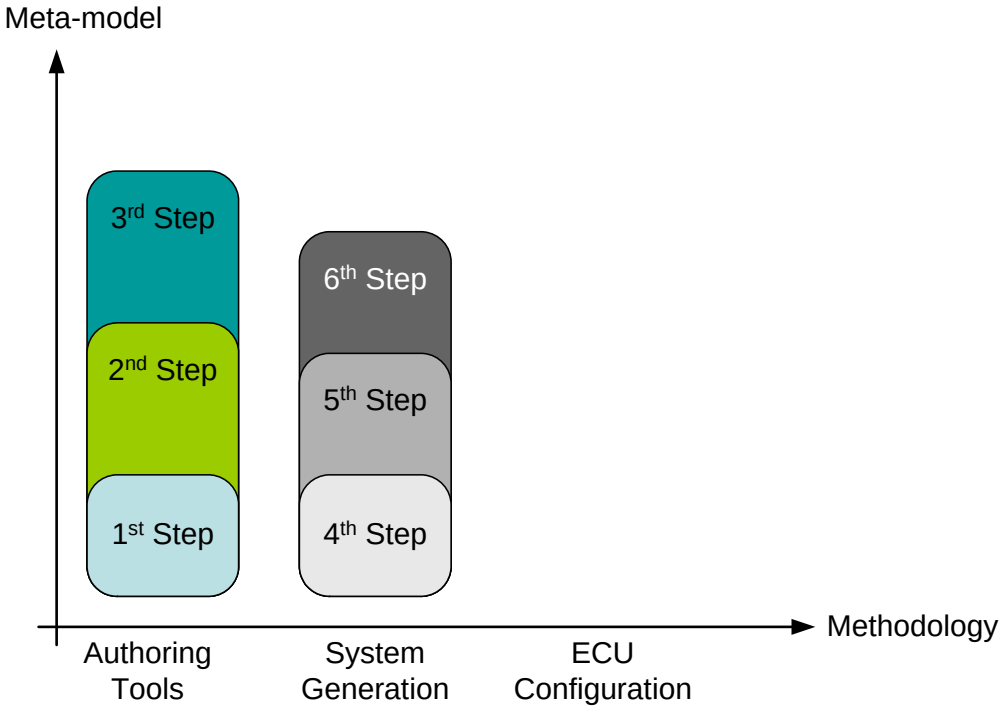

Figure 11-3: Scenario 3

Of course, the effort for this approach would be considerable since it consists of several small steps that must be stabilized before moving on to the next step. On the other hand, it is obvious that this approach is optimally suited to keep the overall risk at a minimal value.

11.4 Scenario 4

This approach is positively the most ambitious of all discussed scenarios since evolution of feature definition is driven towards a greater complexity **and** the coverage to the methodology is increased **at the same time**.
On the first view, this seems to be the approach that leads to the goal of having a full implementation of AUTOSAR in a minimal amount of time. On the other hand, this evolution model unquestionably bears a high risk of failure because of the small number of "official[11]" milestones for stabilization of the tool chain.

[11] The tool development team will certainly perform more integration steps than the official roadmap foresees for this approach.

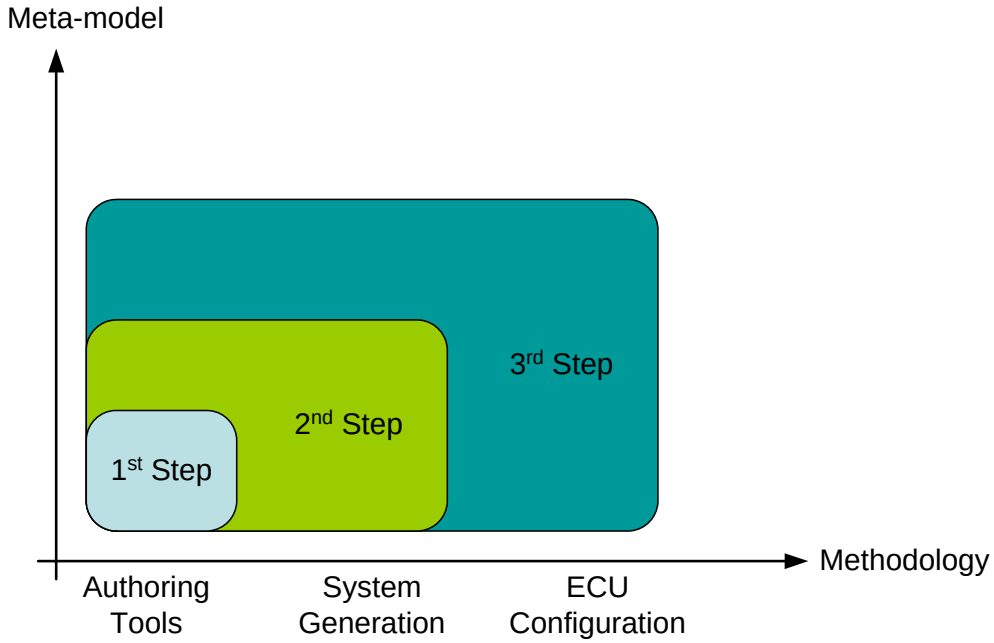

Figure 11-4: Scenario 4

12 References

12.1 Normative References

[1] Software Component Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_SoftwareComponentTemplate.pdf

[2] Specification of ECU Resource Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_ECU_ResourceTemplate.pdf

[3] Specification of System Template
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_SystemTemplate.pdf

[4] Template Modeling Guide
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_TemplateModelingGuide.pdf

[5] Template Formalization Guide
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_TemplateFormalizationGuide.pdf

[6] Specification of the Virtual Function Bus
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_VirtualFunctionBus.pdf

[7] Specification of Communication Stack Types
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_SWS_ComStackTypes.pdf

[8] Requirements on Feature Definition of Authoring Tools
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_RS_FeatureDefinition.pdf

[9] Meta-Model
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_MetaModel.EAP

[10] Specification of Interoperability of Authoring Tools
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_InteroperabilityAuthoringTools.pdf

[11] Specification of Interaction with Behavior Models
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_InteractionBehavioralModels.pdf

[12] Specification of Graphical Notation
https:/svn2.autosar.org/repos2/22_Releases
AUTOSAR_GraphicalNotation.pdf