Document Title	AUTOSAR Methodology
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	068
Document Classification	Auxiliary

Document Version	1.2.2
Document Status	Final
Part of Release	3.2
Revision	1

Document Change History

Date	Version	Changed by	Change Description
27.04.2011	1.2.2	AUTOSAR Administration	Legal disclaimer revised
23.06.2008	1.2.1	AUTOSAR Administration	Legal disclaimer revised
28.11.2007	1.2.0	AUTOSAR Administration	• Subchapter “limitations of the current version” enhanced • Document meta information extended • Small layout adaptations made
31.01.2007	1.1.0	AUTOSAR Administration	• Updated chapter 5 “ECU-Design” • Updated chapter 6.1 “Relationship with Services” • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
27.04.2006	1.0.0	AUTOSAR Administration	Initial release

Disclaimer

Advice for users

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

1 Abbreviations

API	Application Programming Interface
ASAM MCD	Assosiation for Standardization of Automation- and Measuring Systems Measurement, Calibration and Diagnositics
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CPU	Central Processing Unit
DWARF	Debug With Arbitrary Record Format
ECU	Electronic Control Unit
KWP2000	KeyWord Protocol 2000
MCAL	MicroController Abstraction Layer
OEM	Original Equipment Manufacture
OS	Operating System
RTE	Runtime Environment
SW	Software
SWC	Software Component
VFB	Virtual Functional Bus
XCP	Universal Measurement and Calibration Protocol
XML	Extensible Markup Language

2 AUTOSAR Methodology

2.1 Introduction

AUTOSAR requires a common technical approach for some steps of system development. This approach is called the "AUTOSAR methodology". This document defines and describes this AUTOSAR methodology.

This document is a refinement of the "AUTOSAR Technical Overview" [Tech]. It covers all major steps of the development of a system with AUTOSAR: from the system-level configuration to the generation of an ECU executable.

2.1.1 Scope of the methodology

The AUTOSAR methodology is not a complete process description. "Roles" and "responsibilities" are not defined in this methodology.

Furthermore, the methodology does not prescribe a precise order in which activities should be carried out. The methodology is a mere work-product flow: it defines the dependencies of activities on work-products. This means that when the information specified in the methodology is available, an activity can be carried out to produce the output work-products.

This restriction implies that the AUTOSAR methodology does not define an overall time-line and does not define how and when iterations are carried out. For example during system-design, the same activity (namely configuring the system) will be carried out repeatedly with various levels of precision. There will be a first "rough" configuration and a final "precise" configuration which might depend on the feedback from the actual configuration or even implementation of ECUs. How and when such refinement steps are to be carried out is NOT defined in the methodology.

2.1.2 How the methodology is modeled

In order to promote a consistent and precise description of the "AUTOSAR methodology" across the project, a formal syntax, called "SPEM", is used to model the methodology [SPEM].

SPEM is closely integrated with the AUTOSAR meta-model. The AUTOSAR meta-model precisely defines the concepts that are used when describing systems with AUTOSAR[1]. The syntax of the exchange formats (the so-called "templates") between tools is directly generated out of this meta-model[2].

[1] The "Template UML Profile and Modeling Guide" describes the modelling approach used in the meta-model [ModGuide]. The detailed content of the meta-model is described in various other specifications, such as the "Software-Component Template", the "ECU-Resource Template" and the "System Description Template".

[2] The "Model Persistence Rules for XML" describes the relationship between the XML-based exchange formats and the meta-model [ModRules], [MetaModel].

The SPEM of the AUTOSAR methodology relates many work-products that are input or output of an activity to specific elements out of the AUTOSAR meta-model. This ensures consistency between the AUTOSAR “templates” and their application in specific steps in the AUTOSAR methodology: the AUTOSAR meta-model defines HOW something is described; the AUTOSAR methodology defines WHEN these descriptions are used in specific activities.

2.1.3 Limitations of the current version

2.1.3.1 General limitations

Over the past releases in AUTOSAR, detailed work on both template and software specifications has progressed. Hence, one might experience inconsistencies between this document and detailed specifications as referred to in Chapter 7 References. In the unlikely case of a conflict between this document and one of the referenced detailed specifications, the latter shall take precedence.

2.1.3.2 Usage of the C language

This version of the methodology description refers to the implementation language C. That means the handling of software sources in this context is explained or illustrated exemplarily for C. Basically the methodology should be independent from the implementation language and the given description should be easily adaptable to other languages.

2.2 Structure of this document

Chapter 2.3 of this document describes the syntax used in the SPEM. This chapter is a prerequisite for a precise understanding of the methodology.

The actual methodology starts with an overview and then goes into more depth according to the following structure:

- The System Configuration shows all activities taking place at system-level,
- the activities ECU Design and Configuration are taking place at ECU-level,
- and the Component Implementation highlights the methodology used at component level.

2.3 The notation used to describe the Methodology

2.3.1 SPEM

AUTOSAR describes the methodology using the Software Process Engineering meta-model, or SPEM for short. SPEM standardizes the terminology used for

describing processes. SPEM is a standard defined by the Object Management Group (OMG) and is designed to describe a concrete software development process or a family of related software development processes [SPEM]. SPEM is a UML profile, which makes it possible to integrate the AUTOSAR methodology right into the AUTOSAR meta-model.

For the purposes of describing the AUTOSAR methodology, only a very small subset of SPEM is actually used. The following sections describe the modeling-elements used for the definition of the AUTOSAR methodology. These are:

- Work-Product,
- Activity,
- Guidance,
- Flow of Work-Products,
- Dependencies between Work-Products,
- Composition of Work-Products,
- and References to elements of the meta-model.

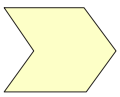
2.3.2 Work-Product

A «Work-Product» is a piece of information or physical entity produced by or used by an activity.

For the AUTOSAR methodology 4 specific kinds of «Work-Product» are defined:

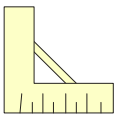
- XML-Document[3],
- c-Document (for files containing sources in the language C),
- obj-Document (for object files),
- h-Document (for files containing header files that are included in c-files).

2.3.3 Activity

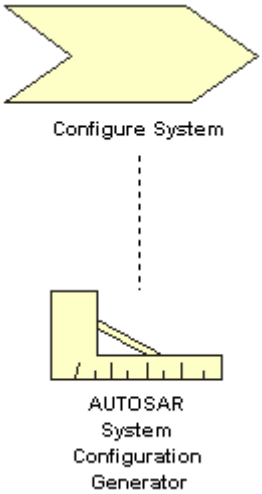


An «Activity» describes a piece of work performed by one or a group of persons: the tasks, operations, and actions that are performed by a role or with which the role may assist[4].

2.3.4 Guidance



«Guidance» elements are associated with activities and represent additional information or tools that are available to the practitioners of the activity. In SPEM, possible types of «Guidance» can for example be: Guidelines,



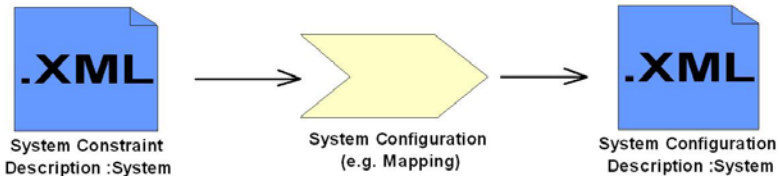
[3] Note that a single XML-document can consist of an arbitrary number of files. methodology defined in this document does NOT define the number of files that are an activity.

[4] Note that the AUTOSAR methodology does NOT define roles.

Techniques, Metrics, Examples, UML Profiles, Tool mentors, Checklist, Templates.

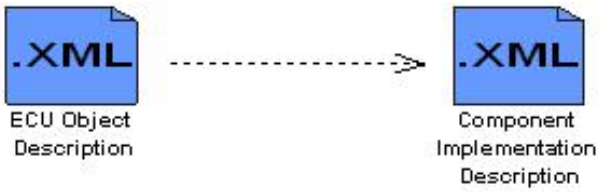
In the AUTOSAR methodology, we are using «Guidance» to model tools that are to be used to perform the activity. The example on the right shows that the activity `Configure System` is associated with the «Guidance» `AUTOSAR System Configuration Generator`. The association is represented by a dotted line and means that the tool `AUTOSAR System Configuration Generator` is used to perform the activity.

2.3.5 Flow of Work-Products



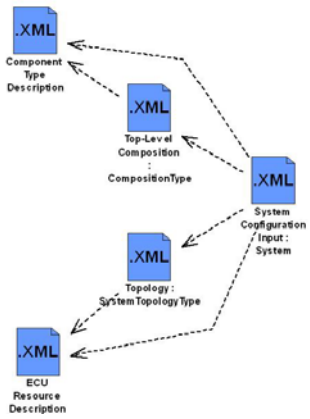
The flow of work-products is graphically represented by a line with an arrowhead. It is always directed from its source to its destination and is used to identify the input and output of an activity. In this example the activity `System Configuration` uses the work-product `System Constraint Description` as input and outputs the `System Configuration Description`.

2.3.6 Dependency



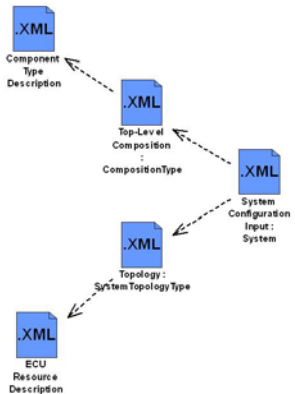
A «Dependency» is a dotted line with an arrowhead that indicates that one work-product depends on another work-product. It is a unidirectional «Dependency» and the direction of the line clarifies who depends on whom. The example shows that the XML-Document `ECU Object Description` depends on the XML-Document `Component Implementation Description`. In this context the «Dependency» can also be interpreted as a reference: the XML-document `ECU Object Description` contains references to information contained in the XML-document `Component Implementation Description`.

2.3.7 Transitive Relations

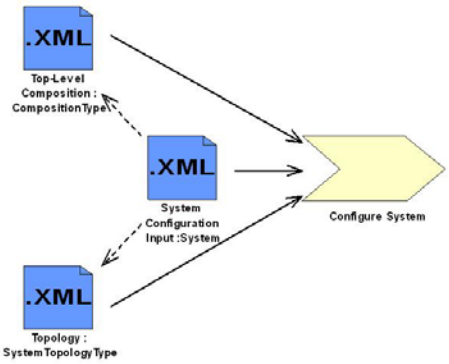


The graphic on the left shows an example where the XML-Document `System Configuration Input` depends on the XML-Documents `Top-Level Composition`, `Component Type Description`, `Topology` and `ECU Resource Description`. This is graphically represented by four single dependencies.

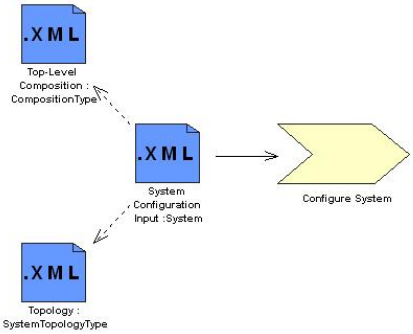
But it is possible to reduce the graphical overhead by observing the following fact: the `Top-Level Composition` depends on the `Component Type`



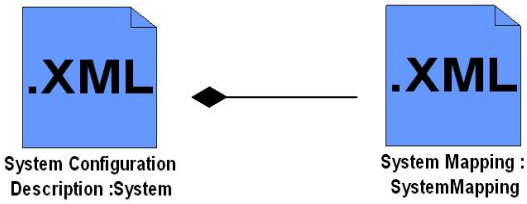
Description and the Topology depends on the ECU Resource Description. Therefore an explicit graphical representation of the dependency between the System Configuration Input and the Component Type Description is superfluous. The same applies to the dependency between the System Configuration Input and the ECU Resource Description. The graphic on the right is semantically equivalent.



The same simplification is applicable to workflows. In this example the activity Configure System has three inputs, namely Top-Level Composition, System Configuration Input and Topology. It is sufficient to show the flow between Configure System and System Configuration Input, because the last named depends on the Top-Level Composition and on the Topology. Also these both graphics are semantically equivalent.



2.3.8 Composition



A «Composition» is graphically represented by a line with a solid diamond on its end. A «Composition» is used to show that one work-product is made up of (=contains) other work-products. In this example the System Configuration Description contains a System Mapping. It is also possible to say that the System Mapping is part of the System Configuration Description.

2.3.9 Reference to elements of the meta-model

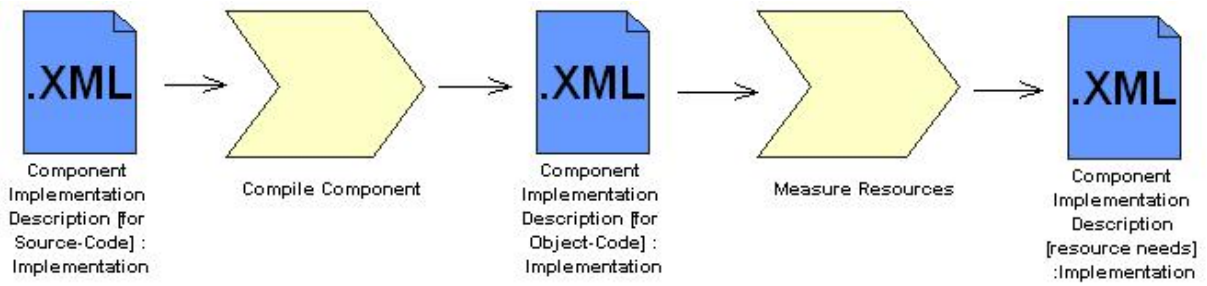
The following notation is used in the AUTOSAR methodology diagrams to indicate that a work-product uses a specific template out of the AUTOSAR meta-model [MetaModel] :

Work-product-name : Meta-class-name [5]

In this example we have an XML-Document named System Configuration Description which is an instance of the meta-class “System”. This reference to the class “System” out of the meta-model defines precisely what information *can*[6] be contained in this work-product and how this information is structured.

[5] The Work-Product is an instance of a meta-class out of the meta-model

2.3.10 Different states of work-products



The state of a work-product can change during an activity. Therefore the same work-product appears several times in the diagram, but with a different state. The name of the work-product remains the same, but the state changes. The notation is as follows:

Work-product-name [Work-product-state] : Meta-class-name

In the example above the state of the `Component Implementation Description` changes from `[for Source-Code]` to `[for Object-Code]` and then to `[resource needs]`.

[6] Through referencing the meta-model, the methodology defines more precisely what information can be contained in the work-products. The current version of the methodology however does not define formally what information MUST be contained in a work-product in order to be able to carry out a certain process step. Future versions of the methodology will also try to capture this information more precisely.

3 Methodology Overview

Figure 1 shows a rough outline of the design steps to build a system and resultant of this the ECUs and the topology with the AUTOSAR methodology.

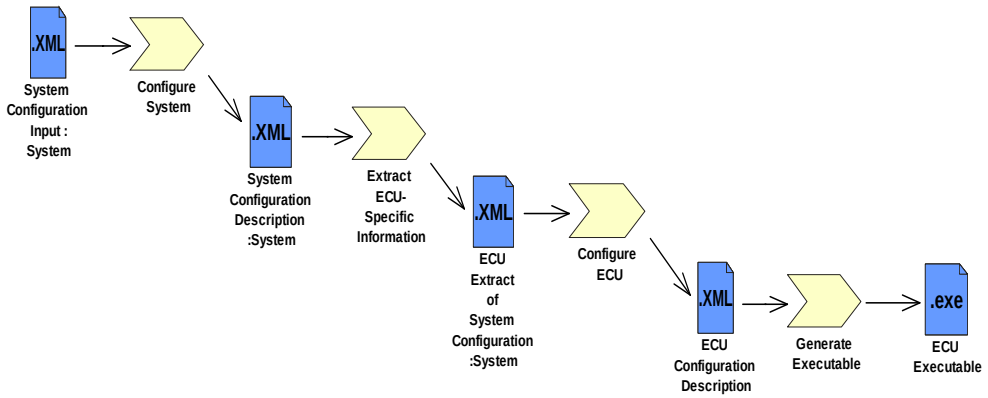


Figure 1: Overview AUTOSAR methodology

Firstly the `System Configuration Input` has to be defined. This is a system design or architecture task. The software components and the hardware have to be selected, and overall system constraints have to be identified. AUTOSAR intends to ease the formal description of these initial system design decisions via the information exchange format and the use of templates. So defining the `System Configuration Input` means filling out or editing the appropriate templates.

This addresses information of the following packages

- Software Components: each software component requires a description of the software API e.g. data types, ports, interfaces, etc., see [SWCTempl].
- ECU Resources: each ECU requires specifications regarding e.g. the processor unit, memory, peripherals, sensors and actuators, see [ECURes].
- System Constraints: this contains e.g. constraints regarding the bus signals, topology and mapping of belonging together software components, see [SysTempl].

It depends on the use case whether a template has to be filled out from scratch or whether a reuse – probably with some editing – is possible. Basically the AUTOSAR methodology allows for a high degree of reuse in this context.

The activity of the `Configure System` mainly maps the software components to the ECUs with regard to resources and timing requirements.

The output of the `Configure System` is the `System Configuration Description`. This description includes all system information (e.g. bus mapping, topology) and the mapping of which software component is located on which ECU.

The step Extract ECU-Specific Information extracts the information from the System Configuration Description needed for a specific ECU. This is then placed in the ECU Extract of System Configuration.
The activity Configure ECU adds all necessary information for implementation like task scheduling, necessary BSW (basic software) modules, configuration of the BSW, assignment of runnable entities to tasks, etc.

The result of the activity Configure ECU is included in the ECU Configuration Description, which collects all information that is local to a specific ECU. The executable software to this specific ECU can be built from this information.

In the last step Generate Executable an executable is generated based on the configuration of the ECU described in the ECU Configuration Description. This step typically involves generating code (e.g. for the RTE and the BSW), compiling code (compiling generated code or compiling software-components available as source-code) and linking everything together into an executable.

Parallel to these briefly described steps of the methodology there are several steps required to integrate the software components into the whole system, e.g. generating the components API, and implementing the components functionality. For clarity they are not depicted in Figure 1. Nevertheless the implementation of a software component is more or less independent from ECU configuration. This is a key feature of the AUTOSAR methodology.

The following sections describe the various parts of the AUTOSAR methodology in more detail. To reflect the parallelism of the several activities we don't follow the simplified sequential structure of Figure 1, but we distinguish parts of the methodology that are necessary at least once per system, per ECU, and per component.

4 System Configuration

4.1 System Configuration Overview

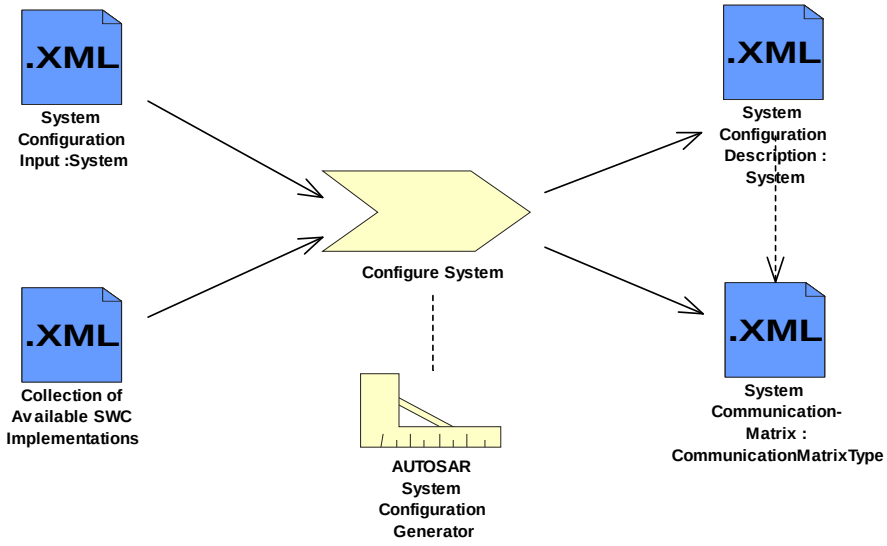


Figure 2: System configuration overview

The activity `Configure System` takes engineering decisions at system level. These decisions are based on the `System Configuration Input` and the `Collection of Available SWC Implementations`. The `AUTOSAR System Configuration Tool` supports the decisions. Output of this activity is a complete `System Configuration Description` and an associated `System Communication-Matrix`.

4.2 System Configuration Details

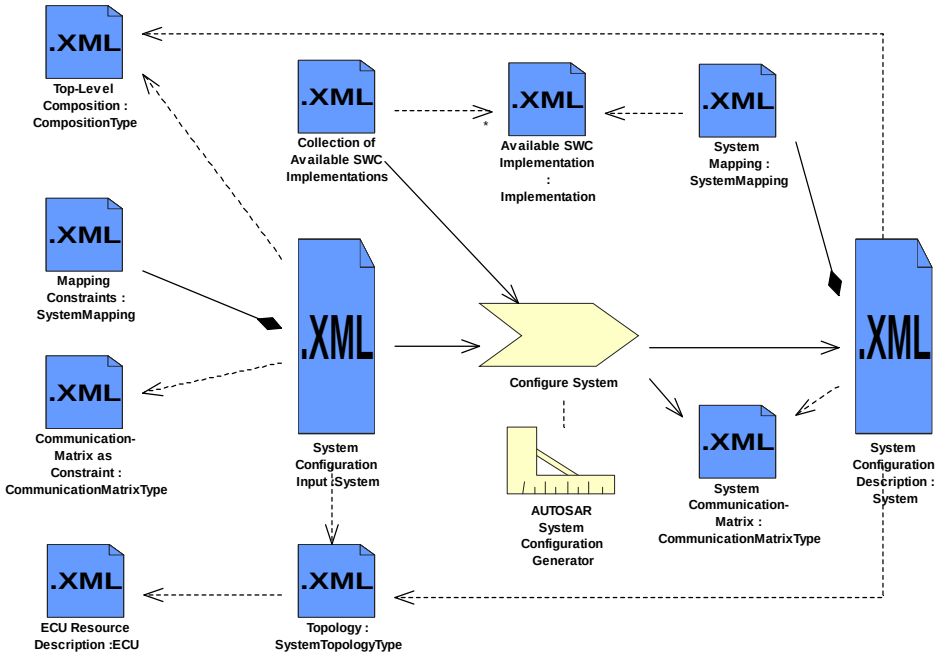


Figure 3: System configuration details

The activity `Configure System` is performed at system level. In addition to the `System Configuration Input`, this activity needs a `Collection of Available SWC Implementations`, which contains a description of the software-component implementations that can be used to realize the components required by the `Top-Level Composition`. The work-products coming out of this activity are the `System Configuration Description` and the `System Communication-Matrix`.

The `System Configuration Input` contains a reference to a `Top-Level Composition`. This `Top-Level Composition` contains a hierarchical description of all components that should be present in the system to be generated. The outgoing work-product `System Configuration Description` references the *same* `Top-Level Composition`. This means that during the `Configure System` activity the component-view on a system (which components are present) is not modified.

The `System Configuration Input` also contains a reference to the `Topology` of the system. The topology describes the ECUs that are present in the system and how they are interconnected through networks. The topology references `ECU Resource Descriptions`, which describe the hardware-resources available on individual ECUs in the topology. This topology is not modified during the activity `Configure System`; the `System Configuration Description` references the same topology.

One of the most important decisions that are taken during the Configure System activity is the "mapping": for each component (out of the Top-Level Composition), a decision must be taken on what ECU in the Topology the component runs.

As part of the mapping decisions, the Configure System activity might decide on the use of specific implementations for certain software components. These implementations are chosen from the Collection of Available SWC Implementations. Choosing an implementation at system-level might enable a more precise analysis of required and provided resources and allows the system-designer to influence more precisely what happens inside the ECU. In many cases however, such choices are not made at system-level, but are left over to the specific ECU configuration in a later activity in the AUTOSAR methodology.

The results of the decisions regarding mapping and component-implementations are documented in the System Mapping which is part of the System Configuration Description. This System Mapping contains a *complete* mapping of all components on ECUs of the topology and an *optional* choice of specific implementations for the software-components.

The System Configuration Input includes or references various constraints that should be considered during the Configure System activity. These constraints can be described as Mapping Constraints which force or forbid certain components to be mapped to certain ECUs or requires certain implementations to be used for components. In addition, these Mapping Constraints can contain resource estimations describing the net availability of resources on ECUs and thereby limiting the possible mappings. The System Configuration Input can also reference a partly incomplete Communication-Matrix as Constraint.

Finally, an important aspect of the activity is the design of the System Communication-Matrix. This System Communication-Matrix completely describes the frames running on the networks described in the topology and the contents and timing of those frames.

The tool AUTOSAR System Configuration Tool supports the activity Configure System. That means this tool is more than a generator that produces the output based on some input following a certain algorithm. It is rather an editing tool. It should help to take the aforementioned engineering decisions (e.g. via clear graphical representation), to store these decisions, and to change them later if necessary. So when an initial output was generated, the tool will be used to refine both the System Configuration Description and the System Communication-Matrix. As a consequence of such iterations during the overall system development, the tool also has to be able to read not only the inputs but the outputs as well.

4.3 Activities after System Configuration

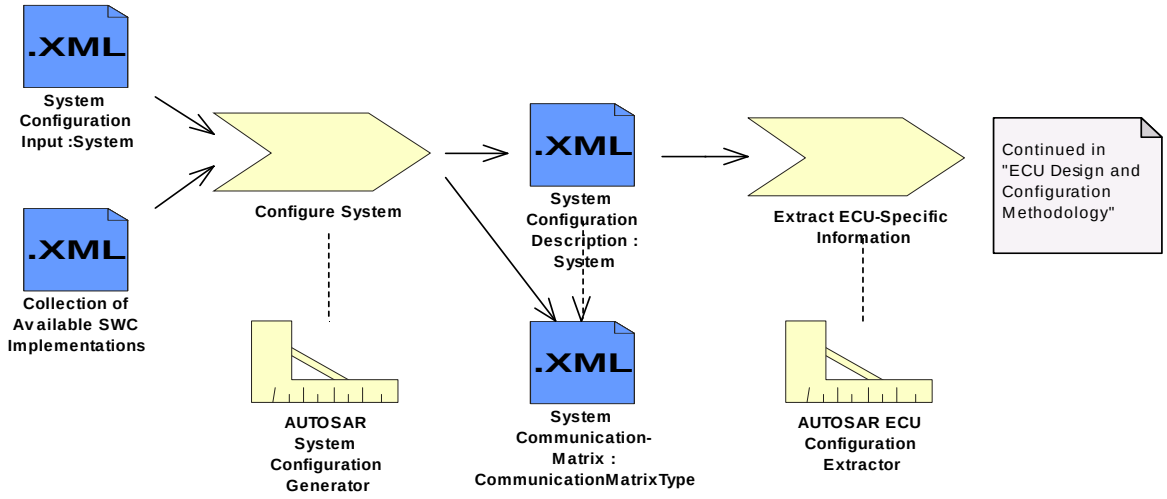


Figure 4: Activities after system configuration

The `System Configuration Description` which is output of the `Configure System` activity can be used to generate ECU-specific extracts of the system configuration. These ECU-specific work-products[7] are used as input to the design and configuration of specific ECUs as described in the following section.

[7] This includes also an ECU specific extract of the system communication matrix.

5 ECU Design and Configuration Methodology

5.1 Overview

Figure 5 shows an overview about the design steps to build an ECU with the AUTOSAR technology.

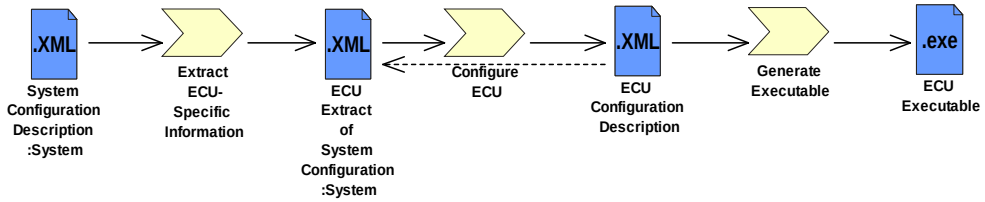


Figure 5: Overview about ECU part of the AUTOSAR methodology

The input to this phase is the `System Configuration Description`, which is created during the system configuration phase. The output of this phase is the executable ECU software.

To avoid misunderstandings it should be emphasized that the `ECU Executable` described in the methodology is not always the executable which will be used finally in the production line. The methodology does not define when and how iterations take place. Thus in practice the executable likely will change during development, e.g. due to optimizations or to consider calibration.

The major activities in this phase are the extraction of ECU-specific information from the `System Configuration Description`, the configuration of the ECU and the generation of the executable ECU software. The following sections will describe these activities in more detail.

5.2 Extract ECU-Specific Information

The `System Configuration Description` is an instance of the `System` element of the AUTOSAR meta-model. The tool `AUTOSAR ECU Configuration Extractor` extracts the information from the `System Configuration Description` needed for a specific ECU. This is a one to one copy of all elements of the `System Configuration Description` that are appointed to this specific ECU. Hence the activity `Extract ECU-Specific Information` can be completely automated. The result is primarily the `ECU Extract of System Configuration`.

There are some additional outputs generated in this activity that are for clarity neglected in the overview of Figure 5. These are the `ECU Extract of Communication Matrix`, the `ECU Extract of Topology`, and the `ECU`

Extract of Top-Level Composition. The input System Configuration Description simply references the information to be extracted. The ECU Extract of System Configuration can reference all these outputs.

5.3 Configure AUTOSAR Services

The ECU Extract of the System Configuration by means of the ECU Extract of Top-Level Composition contains all information about which components are mapped to a specific ECU. This Information is used in the Generate ECU SW Composition activity illustrated by Figure 6: A new work product ECU Software Composition is created which represents the overall software composition on a particular ECU, forming the basis for RTE generation. On one hand it references the application ComponentPrototypes via the ECU Extract and on the other hand contains newly generated ServiceComponentPrototypes describing the Services required by the application component: For each mapped ComponentPrototype of type AtomicSoftwareComponentType, the PortPrototypes requiring a Service are collected. Based on this information, ServiceComponentTypes are created exactly once per service per ECU with the corresponding number of PortPrototypes, thus that all service-type Port-Prototypes on the Application Components have their PortPrototype counterpart on the ServiceComponentType. Additionally, in order to connect the services to the application components, ServiceConnectorPrototypes are generated in this activity, directly connecting the application port with the service port. In order to describe the Service completely with regard to RTE generation an InternalBehavior and Implementation is created for each Service-ComponentType.

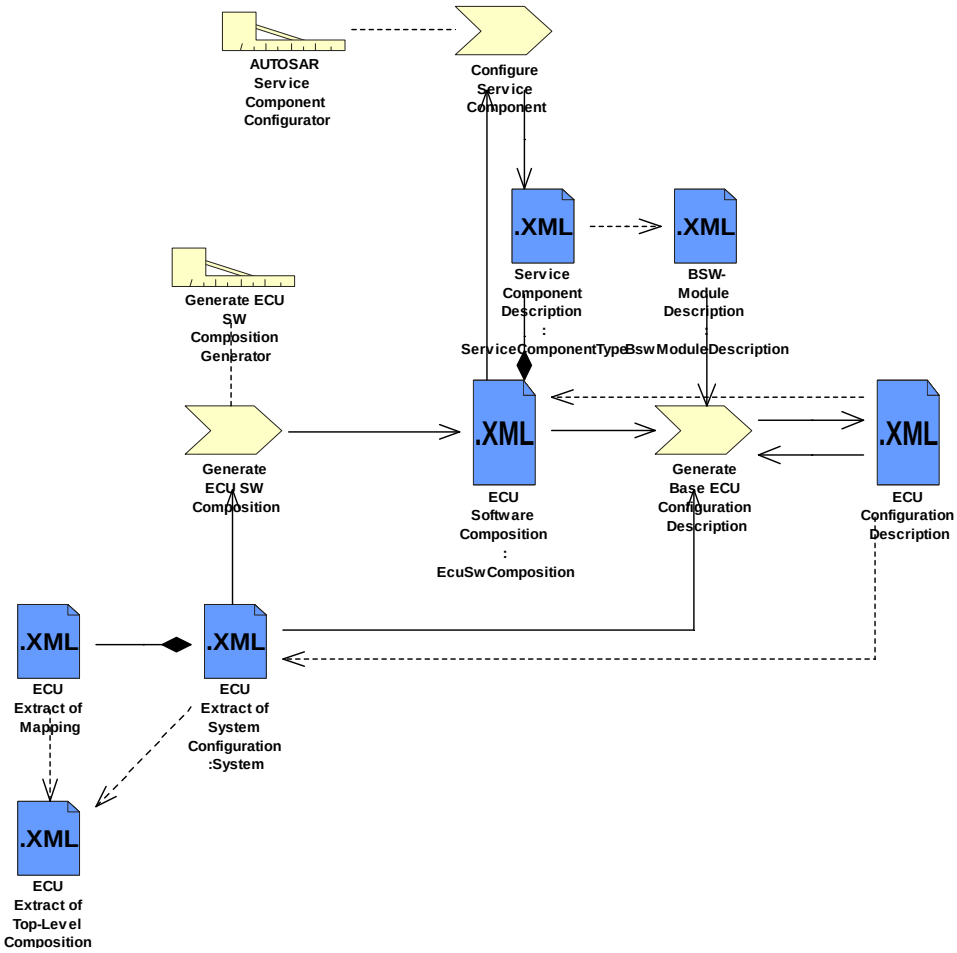


Figure 6: Per ECU Service configuration

The activity `Configure Service Component` adds all missing information relevant for RTE generation to the `InternalBehavior` associated with each `ServiceComponentType`. In particular, the port defined argument values required for the usage of some service interfaces are configured, and the required `RunnableEntities` and `RTEEvents` necessary for the RTE generation are set up.

`EcuSoftwareComposition` together with the `ECU Extract of the System Configuration` then serves as input for generating the `Base ECU Configuration Description`. Further parameter configuration of the BSW module implementing the service is being done in ECU Configuration phase, as explained in the next chapter.

5.4 Configure ECU

In contrast to the extraction of ECU-specific information, the configuration of the ECU is a non-trivial design step, which requires complex design algorithms and engineering knowledge. This step deals with e.g. the detailed scheduling information

or the configuration data for the needed Basic Software modules. With respect to the complexity of the configuration firstly there will be a perfunctory view on it, followed by a more detailed description. For background information and even more details on this topic (e.g. distinguish configuration at pre-compile time, at link time, or at post-build time) refer to [ECUConf].

Figure 7 depicts an overview of the ECU configuration. The activity `Configure ECU` tends to deliver a usable `ECU Configuration Description` for the following generation step. The configuration activity is based on the input work products `ECU Extract of System Configuration`, `Collection of Available SWC Implementations`, and `BSW Module Description`. The latter contains the `Vendor Specific ECU Configuration Parameter Definition` which defines all possible configuration parameters and their structure. This is necessary because the output `ECU Configuration Description` has a flexible structure which does not define a fixed number of configuration parameters a priori. The `BSW Module Description` is assumed to consist of single descriptions delivered together with the appropriate used BSW module.

For details in the ECU configuration the `ECU Configuration Description` has to be able to reference the `BSW Module Description`, and also the `ECU Extract of System Configuration`.

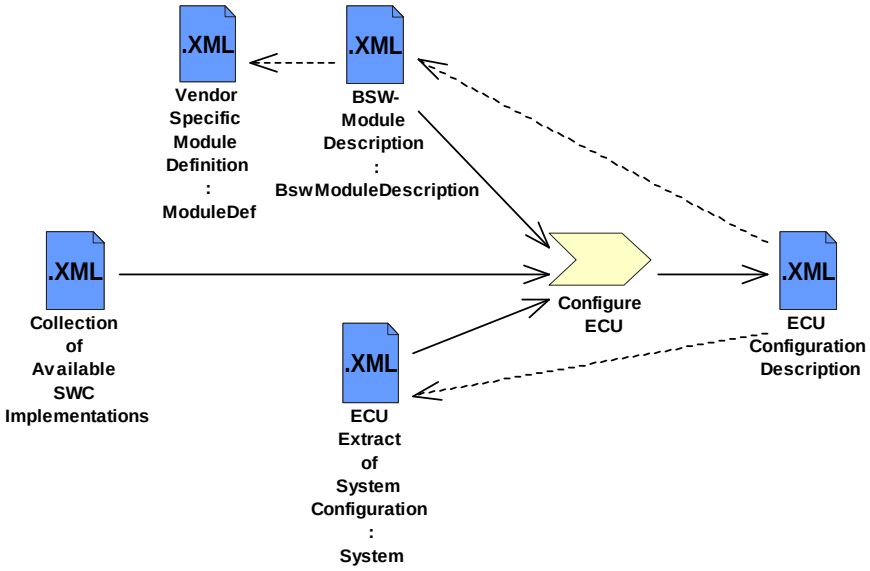


Figure 7: Overview about ECU configuration

Figure 8 shows the details of the ECU configuration. In regard to the time flow of the ECU configuration, this starts with the activity `Generate Base ECU Configuration Description` which is supported by the tool `AUTOSAR Base ECU Config Generator`[8]. This activity takes the input `ECU Extract of System Configuration`, the `BSW Module Description` and if existing a previously generated `ECU Configuration Description`.

[8] Probably integrated in any other ECU configuration editing tool.

The upper left part of Figure 8 depicts the work products the `ECU Extract of System Configuration` has to access (either as contained part or via reference) for generating the base configuration. These are ECU specific extracts of the top-level composition, the mapping and the communication matrix.

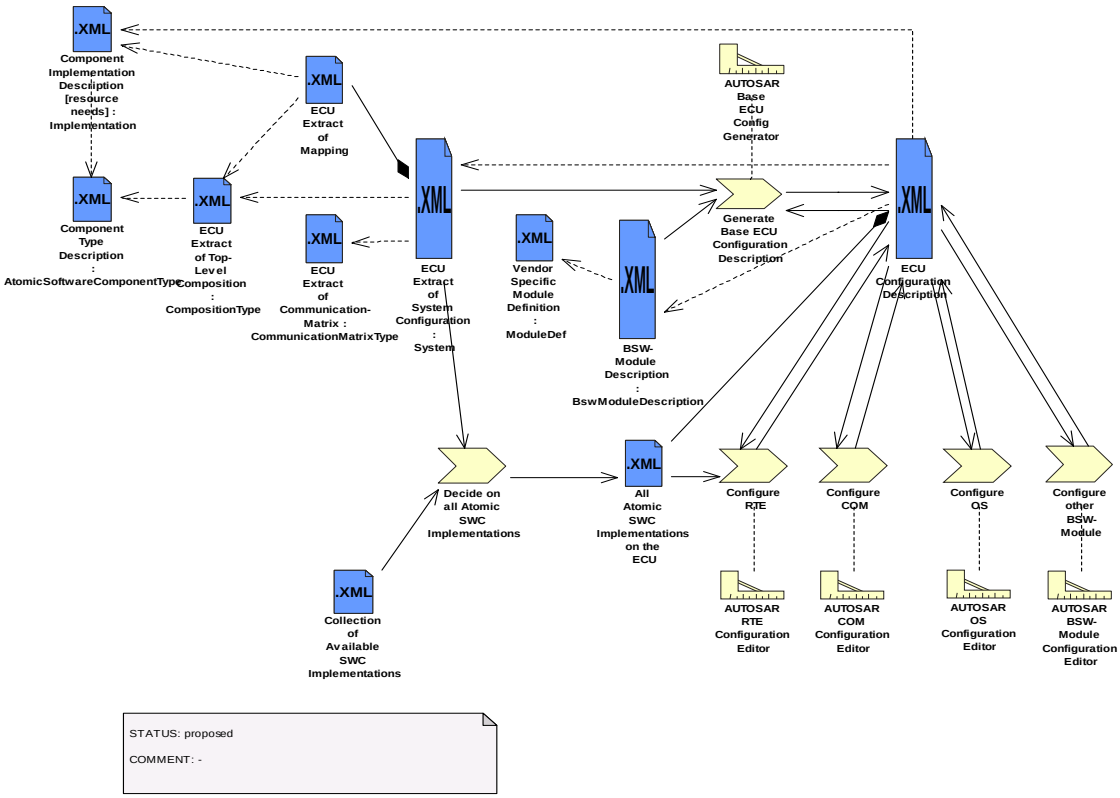


Figure 8: Details of ECU configuration

When there is a base ECU configuration, the real configuration can be performed. This is mainly editing work on the `ECU Configuration Description` which is typically supported by an editing tool. In practice this will require iterations and/or parallel work to configure the RTE and all participating BSW modules. For clarity Figure 8 distinguishes only the configuration activities `Configure RTE`, `Configure COM`, `Configure OS`, and `Configure other BSW Module`, all supported by an appropriate configuration editor[9].

The methodology does not prescribe a certain order of these configuration steps. The `ECU Configuration Description` which was produced by one activity can be read by another activity (e.g. `Configure RTE` generates a description and `Configure COM` reads this).

[9] This editor may also be a generic tool which is able to configure any parameter in the `ECU Configuration Description`. In addition such a generic editor may contain the capabilities of the `AUTOSAR Base ECU Config Generator`.

Usually the configuration activities for the BSW modules (inclusive COM and OS) read and write the ECU Configuration Description.

The configuration of the RTE is more complex. This additionally needs the work product All Atomic SWC Implementations on ECU. As this description may change frequently (e.g. because software components have been moved to or from other ECUs or simply another implementation of a software component has been selected), the configuration of the RTE has to be repeated as well.

The All Atomic SWC Implementations on ECU is the output of the activity Decide on all Atomic SWC Implementations. That means based on the Collection of Available SWC and on the Service SWC Description an implementation is selected for each Atomic Software Component.

This leads to the configuration of AUTOSAR services in the ECU context as depicted in the lower left of Figure 8. The activity Configure AUTOSAR Interface of Service results in a Service SWC Description for the service. Thus it reads the Requirements on Service which reference the Service Description, and a Template for Service Configuration Description. The Service SWC Description must be generated, because it depends on the configuration of the service.

5.5 Generate Executable

After the ECU has been configured, software for several parts of the ECU can be generated. This refers to the Basic Software, the RTE and (if the implementation of all necessary software components is available) the linking of the components resulting in the executable code of the ECU. The following sections describe these generation steps in more detail.

5.5.1 Basic Software Generation

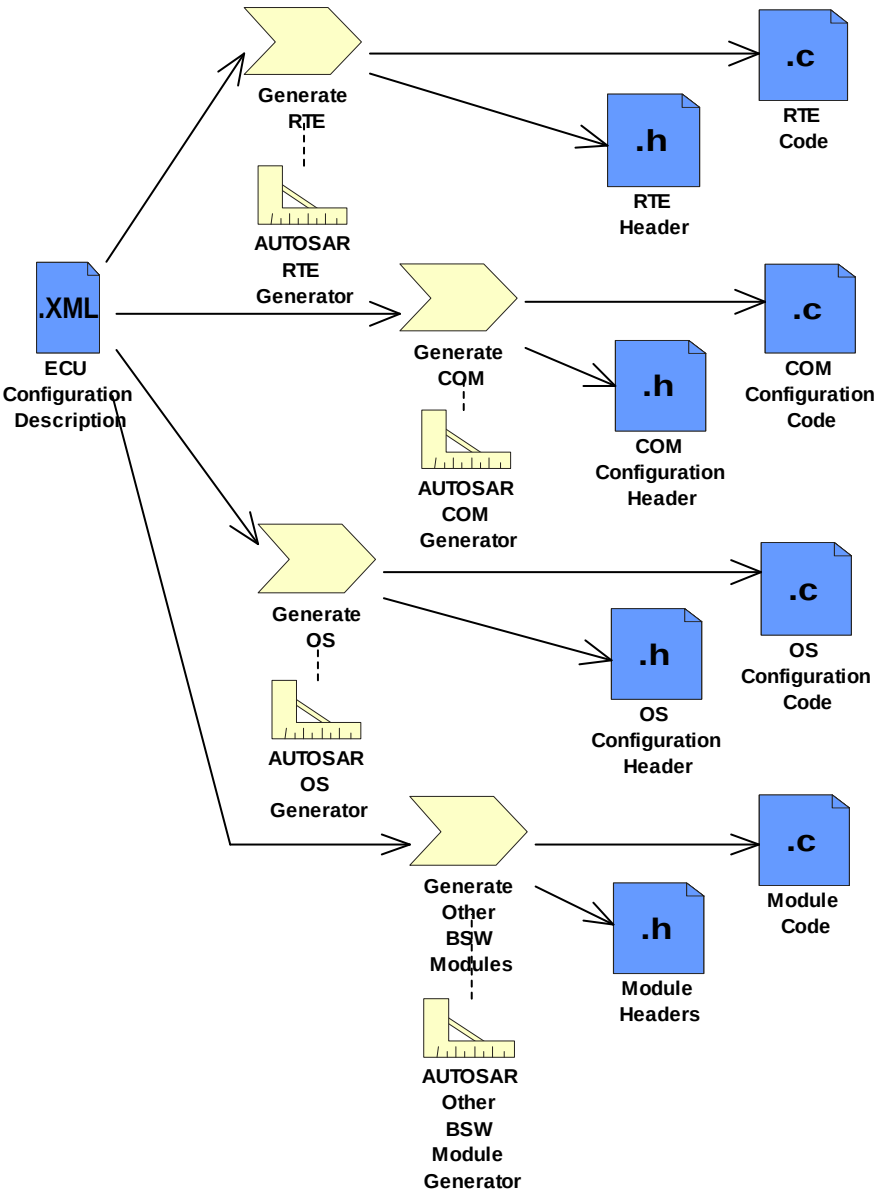


Figure 9: Per-ECU Basic Software generation

Figure 9 shows the generation of Basic Software. For each module of the BSW a generator reads the relevant parameters from the `ECU Configuration Description` and creates code that implements the specified configuration. For sake of clarity Figure 9 distinguishes only generation activities `Generate RTE`, `Generate COM`, `Generate OS`, and `Generate Other BSW Modules`. Appropriate generation tools[10] should support all these generation activities.

[10] These tools don't have to be stand-alone tools. Probably a generator for a certain BSW module will be integrated in the related configuration editor. It is also possible to aggregate the generation for several modules in a generic generator.

The configuration of the RTE and BSW modules, like COM or OS, reflects via specifically generated C code.

5.5.2 RTE Generation

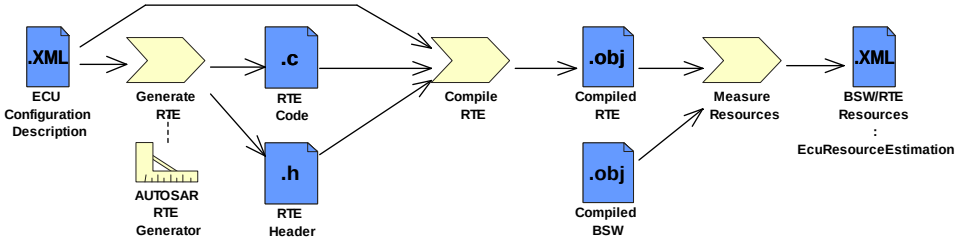


Figure 10: Per-ECU RTE generation

Figure 10 shows the details of the RTE generation process. After the RTE code and headers are generated by the activity `Generate RTE`, the generated RTE code is compiled (`Compile RTE`). Some resources, such as required space for code and data, can already be measured (`Measure Resources`).

5.5.3 Generation of Executable Code for ECU

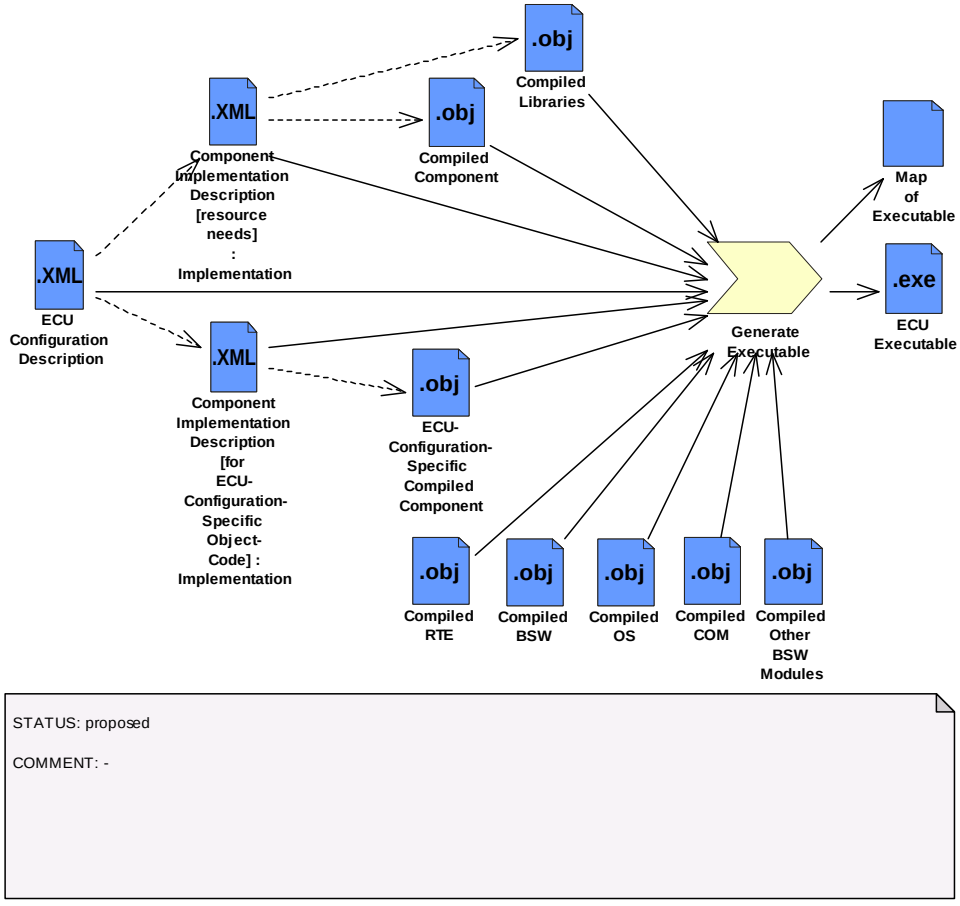


Figure 11: Generation of executable code for ECU

The remaining steps to generate the code for an ECU resemble today's development practice. However, it is important to note that the `Generate Executable` activity is more than a simple linker step. Information from the `ECU Configuration Description` might be used to generate specially configured executable software. The `ECU Configuration Description` is needed as input to the `Generate Executable` activity, because it contains the information which BSW modules and SWC implementations are used to create the executable.

The output of this activity is the `ECU Executable` and the `Map of Executable` (which is typically the log file from linking the `ECU Executable`).

Furthermore it has to be synchronized with the component implementation (as described in section 6). That means the `Compiled Component` and the `Compiled Libraries` must be available.

5.6 Measurement and Calibration

The AUTOSAR methodology supports the use of measurement and calibration. Thus it is necessary that data characteristics, which should be measured or changed by means of a calibration tool, are described in the standard format of ASAM MCD, see [ASAM]. The ASAM MCD description collects the information of how to physically interpret binary data on certain ECU memory addresses. The related file format is called A2L.

Basically measurement and calibration can be performed for each ECU separately. Figure 12 depicts the steps to generate an A2L file usable by a calibration tool.

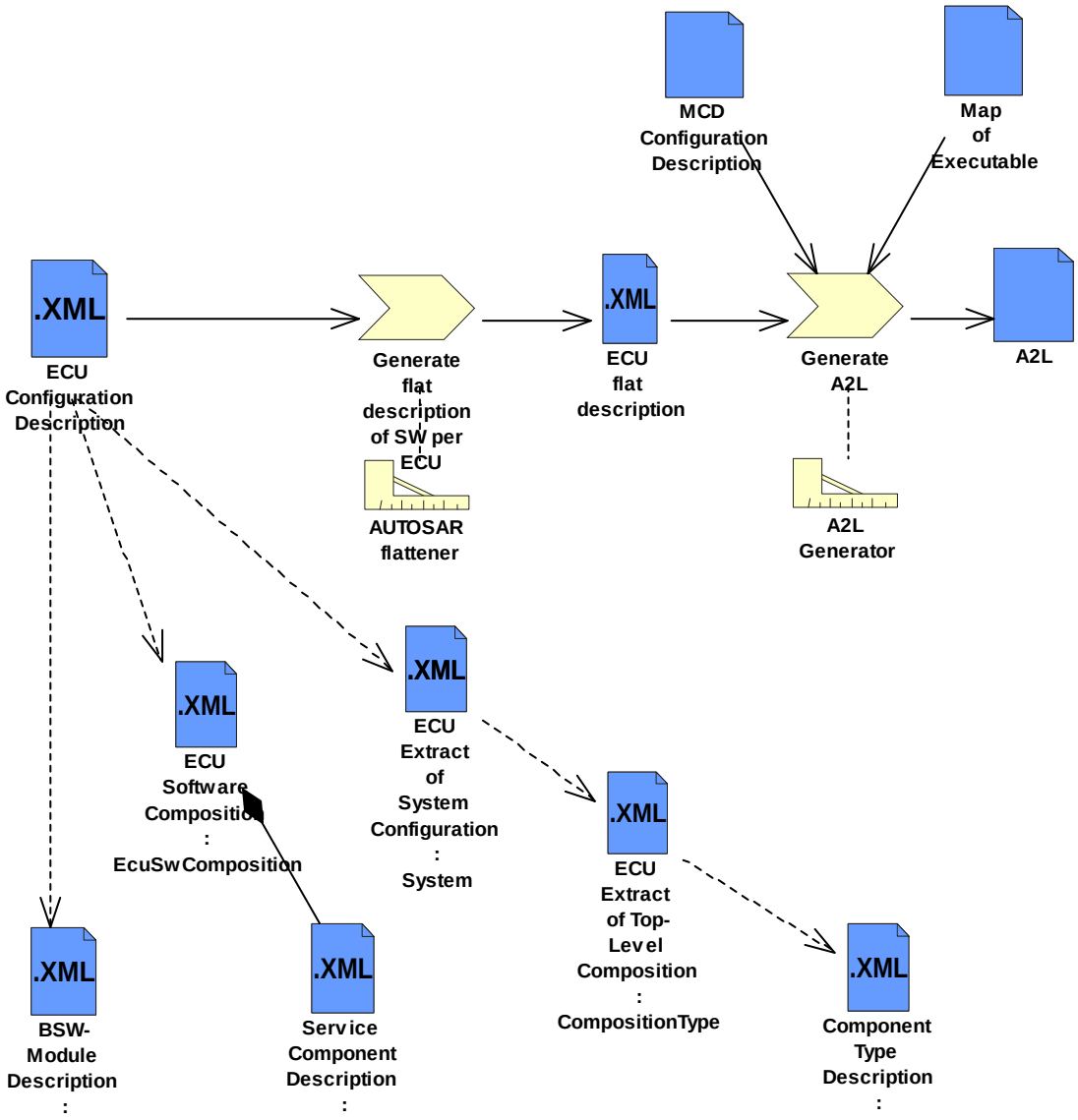


Figure 12: Measurement and calibration – A2L generation

The basic information about which data to measure or calibrate and how to interpret it is stored in the description of each software component. However, the information

on C-symbols required to identify the data is not directly available in the the Component Type Description. We need an intermediate work product per ECU, called ECU flat description, which contains this information. Note that this work product could be an intermediate format for the RTE generation activity, so it makes sense to combine the ativity Generate flat description of SW per ECU with the RTE generator. The activity generate flat description of SW per ECU reads the Component Type Description of all the software components mapped to the ECU, the Service Component Description of all AUTOSAR Services configured for the ECU and the BSW Module Description of all BSW modules integrated on the ECU. It gets access to these data via the ECU Configuration Description.

The flat description is not yet usable by a calibration tool because it does not contain address information. So this will be fetched in the next activity Generate A2L. Here the A2L Generator tool reads the A2L and gathers the appropriate addresses from the Map of Executable. In addition some information about the measurement and calibration configuration is needed. This information, e.g. of the used XCP, CCP, or KWP2000 driver is provided by the MCD Configuration Description. Based on these inputs the A2L Generator generates the A2L output, which should be directly usable with calibration tools.

The A2L Generator tool needs some kind of parsing intelligence to correctly interpret the file syntax of the Map of Executable (which is typically the log file from linking the Executable). As this is highly dependent on the used compiler/linker tool chain the A2L Generator is not a standardized AUTOSAR tool.

6 Component Implementation

This section describes the workflow and the necessary activities in terms of the AUTOSAR methodology to start the development of an application software component and to integrate it later into the system. The workflow shall allow a more or less independent development of the software component's core functionality. These activities have to be performed for every application software component; hence it is also called "per component".

Figure 13 depicts the per component part of the AUTOSAR methodology. For clarity and easier understandability, this addresses only a basic workflow without any ECU-configuration-specific optimizations. It is assumed that such optimizations, as described in section 6.2, will be rather the default case in practice.

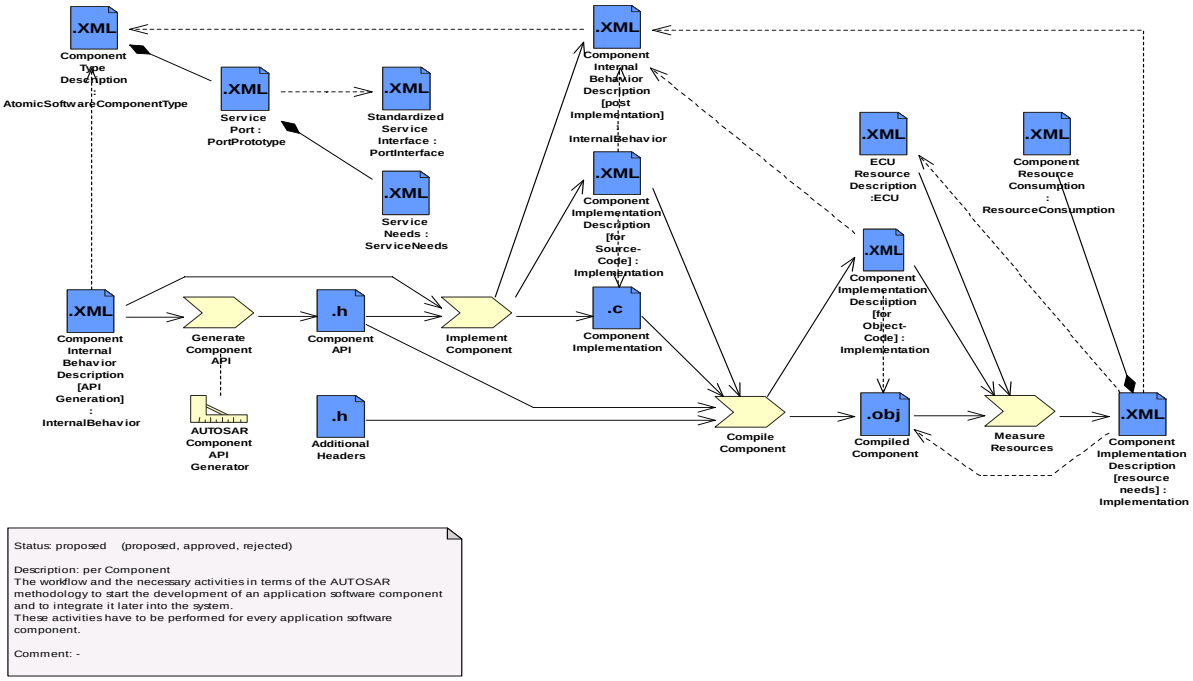


Figure 13: Per component part of the AUTOSAR methodology

The main workflow in Figure 13 runs from the left to the right. The initial work in this context starts with providing the necessary parts of the software component template [SWCTempl]. That means at least the `Component Internal Behavior Description` as part of the software component related templates has to be filled out. The internal behavior describes the scheduling relevant aspects of a component, i.e. the runnable entities and the events they respond to. Furthermore, the behavior specifies how a component (or more precisely which runnable) responds to events like received data elements. However, it does not describe the detailed functional behavior of the component. In practice an AUTOSAR authoring tool will support editing the `Component Internal Behavior Description`.

Afterwards Generate Component API has to be performed. This is a tool-based activity. The AUTOSAR Component API Generator[11] reads the Component Internal Behavior Description of the appropriate software component and generates the Component API accordingly. The Component API contains all header declarations for the RTE communication. There isn't any further engineering or configuration expected in this activity. All input is located in the Component Internal Behavior Description or referenced by it.

Next Implement Component means the functional development of the component. With the Component Internal Behavior Description and the Component API a software developer can implement (i.e. developing, programming, testing) the component vastly independent from the other system design. This implementation basically is outside the scope of AUTOSAR. The results of the implementation will be the Component Implementation (i.e. typically the C-sources), a refined Component Internal Behavior Description, which contains now additional implementation specific information, and a Component Implementation Description, which contains information about the further build process (e.g. compiler settings, optimizations, etc.).

The following activities address the integration of the previously provided component. Compile Component uses the Component Implementation Description for compiling the Component Implementation together with the Component API and the Additional Headers. This yields the Compiled Component and a refined Component Implementation Description. This contains additional new build process information (mainly linker settings) and the entry points.

Now a first Measure Resources basing on the Compiled Component, the Component Implementation Description, and the ECU Resource Description yield a refinement of the Component Implementation Description. Typical measures here refer to memory resources, e.g. RAM, ROM or stack usage.

This description does not mean that a component implementer always has to deliver the component as object code for further integration into an ECU. In AUTOSAR also a component shipment of source code will be supported. In that case the compiling has to be performed within the scope of the integration into an ECU.

6.1 Relationship with Services

Some parts of the aforementioned component implementation depend on the services the component requires. Figure 14 shows how a Component Type Description is used to describe the requirements of a software component on the services of the basic software: The connection points to AUTOSAR Services are described via Port Prototypes which are typed by Standardized AUTOSAR

[11] The AUTOSAR Component API Generator does not have to be a stand-alone tool. The functionality propably is included in the AUTOSAR RTE Generator.

Interfaces. Additional annotations to those ports[12], called Service Needs, are used to describe specific needs to the services, which need to be known by the ECU integrator and cannot be deduced from the Port Interfaces alone. Note that the same method can be applied to describe the connection of sensor/actuator component to the ECU Abstraction, with the only difference, that that the Port Interfaces are not standardized by AUTOSAR in this case.

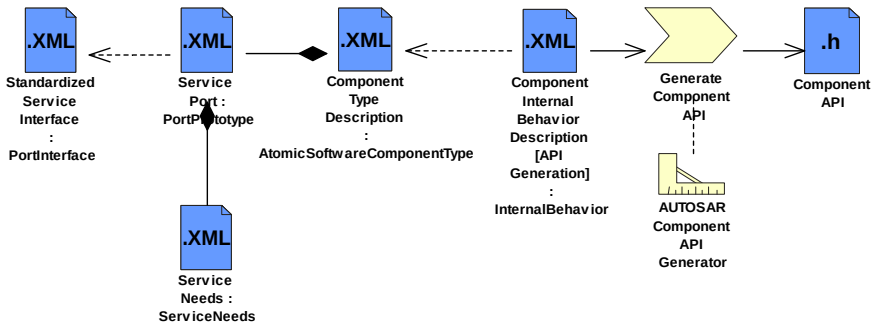


Figure 14: Per service part of AUTOSAR methodology

The services (including ECU abstraction) which shall be used in a certain component affect the API of the component. As depicted in Figure 14, the Component Internal Behavior Description references the Component Type Description which contains the Service Ports. Hence in the activity Generate Component API the generator produces a Component API tailored to the required services. For completeness it should be noted, that the Service Needs give additional information to the integrator which does NOT influence the Component API and is not an input to the later RTE generation.

6.2 ECU-Configuration-Specific Optimizations

In practice the integration of an application software component has to consider some optimizations to meet performance or resource requirements. The Component API might be much more efficient, if it will be generated particularly adapted to the concrete ECU configuration, e.g. via using macro definitions instead of function calls for some RTE interaction. In fact this should not change the Component Implementation (i.e. the C-sources).

[12] This outlines only the general approach. In special cases, requirements on a service may include additional information which cannot be attached to a port, for example the requirements to the NVRAM service may include the description of PerInstanceMemory used as RAM mirror blocks. These special cases are modeled explictly in the Software Component Template. But also in this case, the requirements on the service are described as part of a Component Type Description.

This workflow is shown in Figure 15. It is an optional, additional parallel set of activities, which can start only after the Implement Component as depicted in Figure 13.

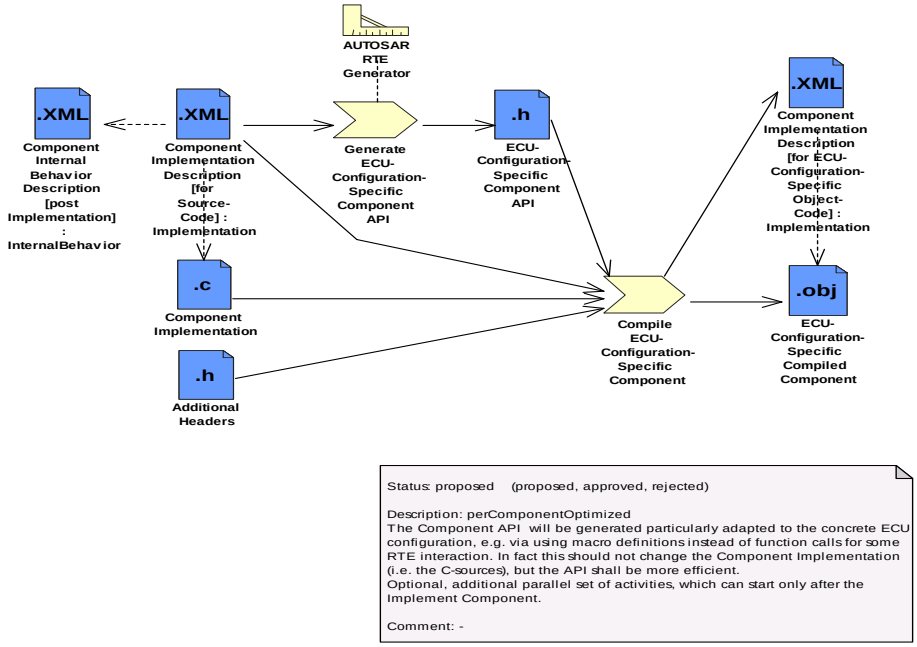


Figure 15: Per component part with ECU-configuration-specific optimization

The Generate ECU-Configuration-Specific Component API reads the Component Implementation Description [for source code], which refers to the Component Implementation and the Component Internal Behavior Description [post Implementation]. This activity is again tool-based. The AUTOSAR RTE Generator generates the ECU-Configuration-Specific Component API. That means now we have a different set of component headers, which include the ECU-configuration-specific optimizations.

The compilation activity has to consider the optimized parts. Hence the Compile ECU-Configuration-Specific Component bases on the inputs Component Implementation, on the Component Implementation Description [for source code], on the ECU-Configuration-Specific Component API and on the Additional Headers. The results of this activity will be the Component Implementation Description and the ECU-Configuration-Specific Compiled Component.

7 References

[ASAM] ASAM MCD Assosiation for Standardization of Automation- and Measuring Systems Measurement, Calibration and Diagnositics,
http://www.asam.net/

[ECUConf] Specification of ECU Configuration,
AUTOSAR_ECU_Configuration_Specification.pdf

[ECURes] Specification ECU Resource Template,
AUTOSAR_ECU_ResourceTemplate.pdf

[Glossary] Glossary,
AUTOSAR_Glossary.pdf

[MainReq] Main Requirements,
AUTOSAR_MainRequirements.pdf

[MetaModel] Metamodel,
AUTOSAR_MetaModel.eap

[ModGuide] Template UML Profile and Modeling Guide,
AUTOSAR_TemplateModelingGuide.pdf

[ModRules] Model Persistence Rules for XML,
AUTOSAR_ModelPersistenceRulesXML.pdf

[SPEM] OMG Object Management Group: Software Process Engineering Metamodel Specification,
http://www.omg.org/technology/documents/modeling_spec_catalog.htm

[SWCTempl] Software Component Template,
AUTOSAR_SoftwareComponentTemplate.pdf

[SysTempl] Specification of System Template,
AUTOSAR_SystemTemplate.pdf

[Tech] Technical Overview,
AUTOSAR_TechnicalOverview.pdf

8 Appendix

This appendix contains some detailed methodology figures without any explanation. They are collected here only for completeness. These figures don't help a reader very much with understanding the methodology (the authors mainly use them).

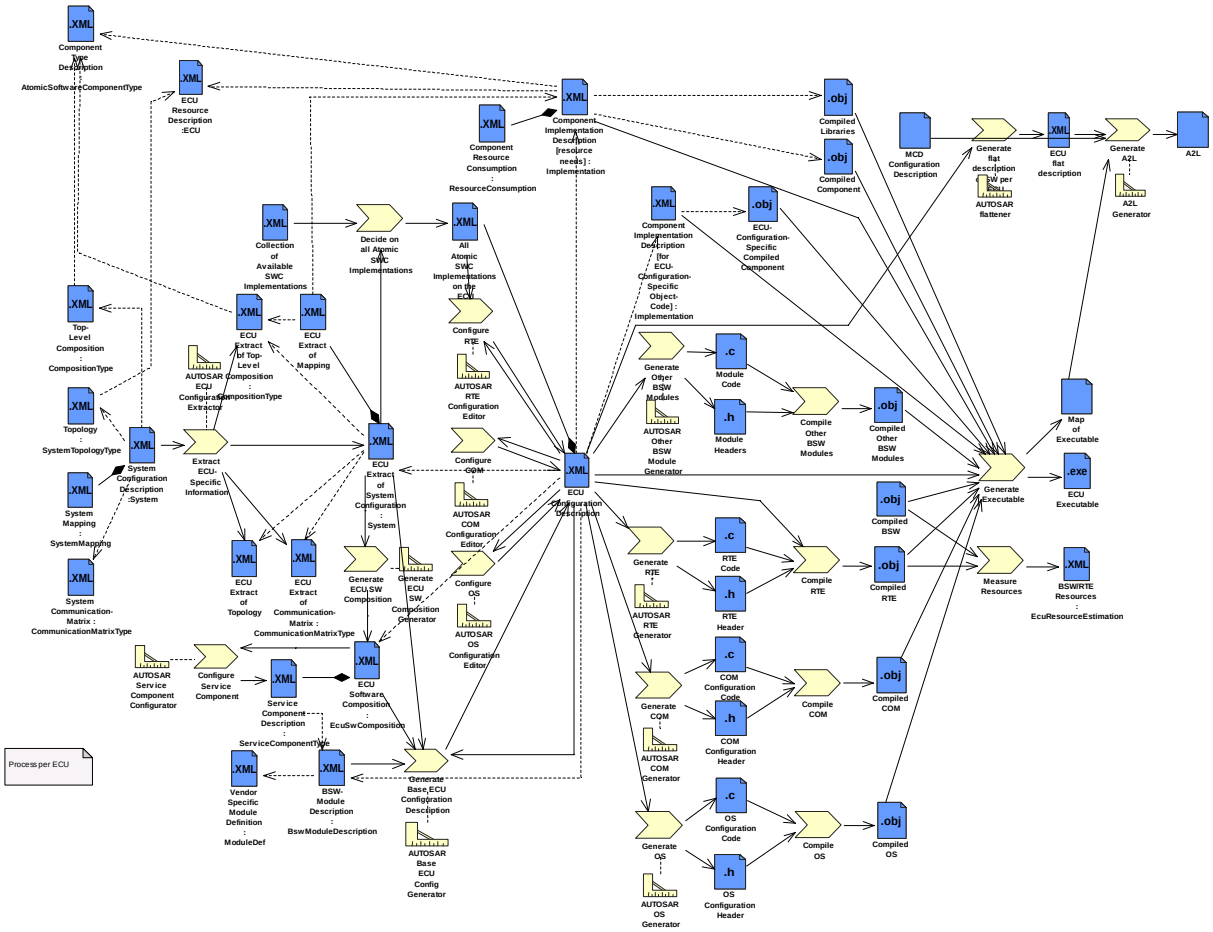


Figure 16: Detailed view of ECU part

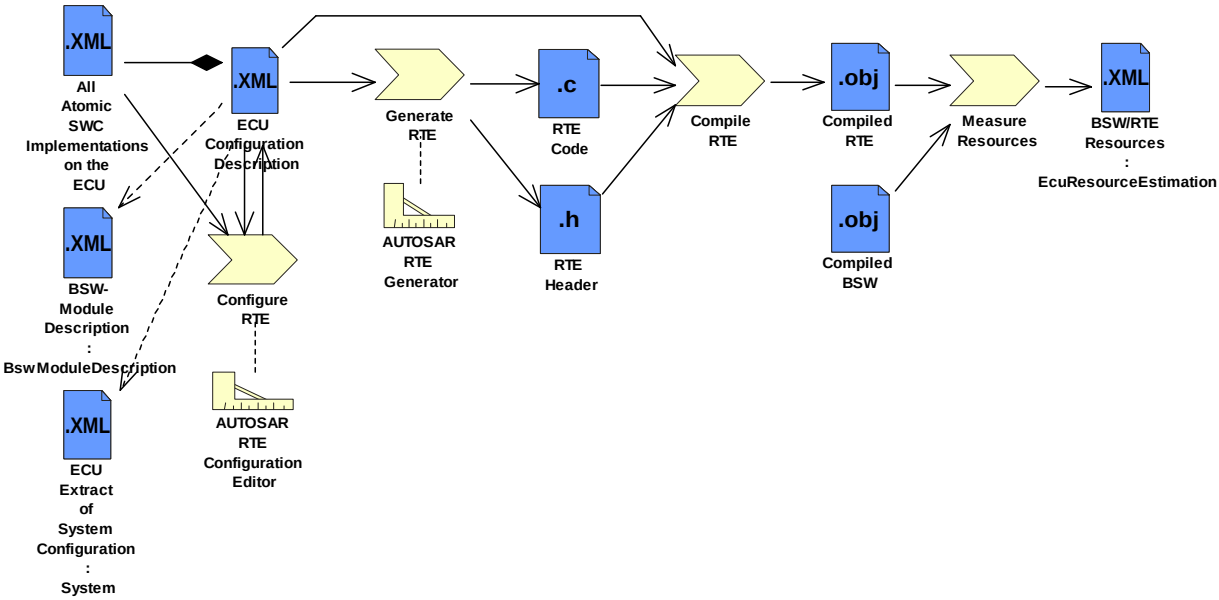


Figure 17: RTE aspects of ECU part