Document Title	Explanatory Document for usage of AUTOSAR RunTimeInterface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	896

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R19-11

Document change history Change History			
Date	**Release**	**Changed by**	**Description**
2019-11-28	R19-11	AUTOSAR Release Management	• Initial release

Disclaimer

Table of Contents

1 Introduction

1.1 Who should read this document?

This EXPlanatory document is intended to describe the steps which are necessary for OS Vendors, RTE Vendors, Debugger Vendors and Timing Tool Vendors to implement the necessary parts to support the AUTOSAR Run-Time Interface Software Specification.

1.2 Objectives

ARTI is a set of standards for debugging and tracing the run-time behavior of embedded systems. Its origin is in the automotive sector, specifically as a concept document developed for the AUTOSAR development partnership, but its scope is not limited to purely automotive systems.

ARTI aims to make it possible for tools from multiple different vendors to collect and exchange runtime data from embedded systems in a standardized way, and hence promote competition and innovation.

ARTI describes interfaces needed to support (a) **static debugging** and (b) **dynamic tracing**.

Static debugging typically involves having an in-circuit debugger connected to the embedded system. Whenever the debugger halts the execution of the system, you can inspect the system's state (registers, stack and data). Decoding the meaning of the state is not necessarily straightforward in any medium or large scale system. ARTI allows the software be described in terms of its architectural concepts and components, so that the debugger can display a much more meaningful representation of the system state. One example is that the debugger could show a list of the tasks in the system along with their state, priority and execution time. It could also show other parts of the system, such as the inter-task messages and their values.

Dynamic tracing on the other hand operates with the embedded system running at normal speed and without interruption. The system records the points in time at which specific events occur and passes this information on to some analysis tool or viewer. As before, ARTI allows these system events to be described so that the analysis tool - viewer can interpret them in terms of architectural concepts and components. Views can be constructed showing the execution pattern of tasks, and statistics based on response times and execution times can be calculated. Dynamic tracing can be achieved with minimal or zero instrumentation of the code where an ARTI compatible in-circuit debugger is available.

The name ARTI was chosen as a nod to a previous automotive standard called ORTI. Whereas ORTI was specified focusing just on debugging embedded operating system, ARTI is intended to be capable of bringing debugging and tracing to all layers of the

software stack. This document first describes the motivation and reasoning for the ARTI specifications without going into the technical details. These are added in later sections.

1.3 Summary of use-cases

The following two use-cases should provide an idea of what is expected by the AUTOSAR Run-Time Interface to enable AUTOSAR aware debugging.

Static debugging A typical use-case is the debugging and stepping through the code execution during integration phase of an ECU. For the analysis of a failure in application software code it is always helpful to understand the OS context, meaning in which task has the error occurred.

Dynamic tracing Continuous testing ECUs on timing of the real time software is not always the standard during software development. Most of the time resource consumption is only measured statically (e.g. memory consumption). But typically some tests, especially affected by stimulating the software with the car environment sporadically fail without no obvious reason. If shifts in the scheduling and misses in timing deadlines are not considered, this source of failure can not be determined. Tracing the dynamic run-time behavior of the software architecture is therefore an important use-case for the AUTOSAR AutosarRunTimeInterface.

2 Run-Time Interface

Capturing the software architecture's dynamic run-time behavior in the field is not easy. The effort to be spend to evaluate simple performance characteristics like CPU load or timing metrics can be tremendously high, depending on the project. There are usually many parts in the tool chain involved which need to work togehter very well in order to provide meaningful and correct results.

In order to explain how the AUTOSAR Run-Time Interface is trying to overcome those generally described problems it is necessary to have a look at the techniques which need to be used to get to the intended result.

2.1 Trace Techniques

For tools to capture the software's behavior at run-time, there are many ways to do so. Each one of them has its advantages and disadvantages. But basically to capture for a longer period the software's dynamic aspects, resources are necessary. Either ECU resources such as processing time or silicon resources such as trace interface and processor pins or even bus resources. The following sub chapters should provide a brief insight into the different trace techniques and therefore an understanding, why ARTI is approached in the way it is introduced later on.

2.1.1 Hardware Tracing

In hardware tracing the main challenge is to lift the information from hardware to system level. Meaning to interpret assembly and low level instructions as execution of an OS Task or any other configuration and software architecture relevant artifacts. There are generally speaking two ways of tracing techniques with some minor silicon vendor and implementation specific differences. Program Flow Tracing (Instruction Trace) and Data Tracing.

Program Flow Tracing: with this approach assembly instructions such as jump, branch or other (vendor dependent) set of instructions create an trace event which is multiplexed and stored on the on-chip buffer or directly streamed off the target. This technique provides a fine granular insight in the code executed on the target. This comes of course at the cost of high bandwidth interfaces or limited measurement time in the on chip case. Important for the ARTI approach is to know: with instruction trace the OS internal state variables can not be seen to the trace unit and therefore the states of Tasks and ISRs can not be derived.

Data Tracing: with the data trace approach data read and write access to most of the memory areas can create a trace event message and export the operation without run-time overhead to the on chip trace buffer or output stream. If the OS internal variables are known, with this approach Task and ISR states can be exported. In a modern AUTOSAR systems there is no state variable or other tracking mechanisms for

Runnables. Therefore, observing Runnables by means of data tracing write accesses to global data object is not natively supported by the RTE. Adding manually a variable to be used in the VFB Hooks for example, only for means of data tracing is often referred to as instrumentation. With this approach trace events are created only by scheduling behavior at run-time.

2.1.2 Software Tracing

Hardware tracing requires that the microcontroller is equipped with an on-chip trace unit/logic. With software based tracing the events can be captured and stored on target in the on-chip memory. Usually in CPU idle, the data can be send off the ECU via the connected bus interfaces, like CAN or Ethernet. The approach is therefore quite similar to data tracing with instrumentation with the difference, that the interface to send the information off the chip is most probably a bus instead of a debug interface.

2.2 Static Debugging

One very important use-case for ECU development and integration is the debugging of the embedded software. For users to debug the software, information about the current running Task or ISR can be pretty helpful. Therefore, the ARTI file provides information to the debugger about the relevant OS internal state variables.

2.3 OS Awareness

One already successful attempt to standardize the access for tracing tools to the OS internals for the OSEK platform is represented by the ORTI (OSEK Run-Time Interface) standard. Information about the Task Stack, Current Running Task, Current Running ISR and the knowledge to decode the Task State from the OS internals was stored in an .ORTI file. For AUTOSAR and especially multi-core projects the standard is not fitting anymore. With the merge from single-core OS to multi-core OS the tracking of Task and ISR states became more complex. Expressions, Pointer handling and so on are difficult to trace during the code execution. Additionally the ORTI standard is not aware about RTE features. But most importantly: Software based tracing is not standardized within ORTI.

The objective of ARTI is to extend this OS awarenss and make the debug, trace and other run-time tools aware of additional AUTOSAR modules such as RTE and SchM.

2.4 ARTI at a glance

The general idea of ARTI is more than to achieve easy "OS Awareness" for debuggers. Starting from a common understanding about scheduling state machines, with a common exchange format for debug and trace configuration. Additionally, exchanging run-time measurements up to their interpretation affects the range of the AUTOSAR/ASAM Run-Time Interface. One of the ARTI goals is to achieve a standard to configure, gather and process as well as to evaluate vendor independent AUTOSAR projects run-time behavior. To achieve this workflow and standardization aims, also a trace exchange format has to be standardized as well as common timing parameters. The Timing Parameters and the trace format is covered by the ARTI ASAM standard.

The following list describes the necessary steps to be taken for an ARTI workflow and their artifact files.

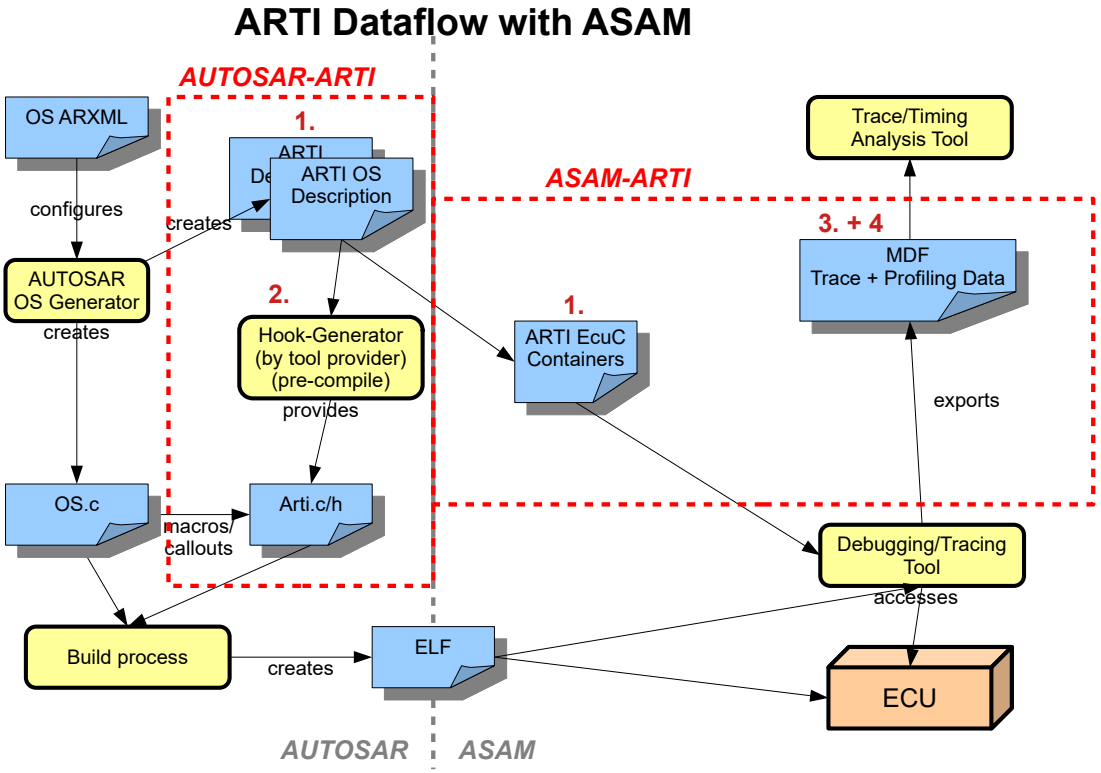

Figure 2.1: ARTI Dataflow

1. **ECU Configuration (ARXML) - AUTOSAR**

   The ARTI ECU Configuration Parameters containers fulfill rather two purposes. For once they store the Trace/Debug Configuration of the AUTOSAR project. Aside of that, the gathering of all ARTI containers replaces the information provided by an ORTI file.

   Currently there are four ARTI ECU Configuration Parameter Containers available: Arti, ArtiGeneric, ArtiHardware and ArtiOs. Depending on the use-case a different set of Parameter Container needs to be configured.

   Arti The ARTI ECUC Container takes care about storing all actual trace and debug information. It is necessary for all ARTI use-cases. It collects the names of all ARTI relevant variables, f.e. the layout of the OS Hooks with a TypeMap to map Task and ISR Ids to the names, or the task state expressions for static debugging, which are referenced from ArtiHardware.

   ArtiOs The ARTI OS container stores basically the OS configuration with a view for tracing and debug tools. It describes mainly all available Tasks and ISRs. Additionally it defines which debug or trace feature is enabled for the referenced OS configuration, while the ARTI container sums up which ARTI hooks, variables and so on are available in the project in total.

   ArtiHardware The ARTI Hardware container stores all references for the currently running Task and ISR OS variables for each core, while the actual variable is stored in the ARTI component. This container is only necessary for static debugging and establishes the connection between CurrentRunningTask, CurrentRunningIsr and the ECU core.

   ArtiGeneric The ARTI Generic container provides the possibility for ARTI OS and RTE vendors to add additional information to the ARTI files, which is not standardized. It can be used to store the start address of an Task for example. The ArtiGeneric container is not mandatory to be used in any use-case.

2. **Hook Generator - AUTOSAR**

   After configuration of the AUTOSAR project, the tracing tool vendor specific hook implementations needs to be generated. Intentionally the trace tool reads therefore the used ARTI ECUC files (split ECUC or merged) and generates out of them the C-code implementation of the ARTI hooks. After adding them to the build process the project is able to be compiled. Background of that workflow is mainly, that the hook macros can be expanded to void and therefore be switched on and off after configuration.

3. **Trace Format (MDF) - ASAM**

   After compilation of the whole project the AUTOSAR Run-Time Interface part of standardization basically ends. The customer is now able to debug or trace the project with a view on OS run-time parameters. During the recording of the Task and ISR transitions signalized by the OS hooks, the tracing vendor stores the timestamp and scheduling event which has taken place. The set of scheduling transitions is defined by the OS hooks and signalize a state transition in one of the state diagrams for the affected scheduling entity (Task, ISR).

   The trace exchange format basically stores the scheduling events with a timestamp and additionally stores the information which scheduling entity is affected and which state diagram is to be used to calculate the possible timing parameters. As exchange format itself the well known ASAM MDF (Measurement Data Format) is used for many reasons: First of all, it can store huge data amounts efficiently, it is well known in the industry as well as it stores data and the description of the data at the same time.

4. **Timing Parameters (MDF) - ASAM**

   After the tracing data is available, the information of interest can be derived. The ASAM Run-Time Interface is therefore focusing on the topic of how interpret the data a set of metrics. With such an workflow the standardization approach should help to cover finding run-time issues in AUTOSAR projects. The timing parameters are intended to be storable in the MDF file, along with the trace data.

The artifact files and the dataflow for one ARTI aware project with both standardization approaches can be seen in figure 2.1.

3 OS

To achieve a vendor independent interpretation of the run-time behavior of an AUTOSAR ECU, ARTI defines state machines. The transitions between those states signalize the OS scheduling at run-time. In other words the state transitions can be understood as the implementation of the OS Hooks. Each time an OS Task is preempted by an ISR for example, the ARTI Hook macro signalizes the change in the OS Task's state transition. This has two advantages: Software based tracing can be used as well as data tracing and the result is the same.

3.1 State Model

The OS State Model, described in the SWS OS is supported by ARTI. But for a proper view with a timing tool, additional state information is necessary. Since there is no distinction between a preempted, released and activated task state an enhanced state machine is introduced. Depending on the features provided by the OS vendor, either the standard OS model can be used or the enhanced one for detailed analysis. To distinguish both state diagrams, the hook macro and therefore the available state transitions shall go by a different state machine description (Class Name). The ARTI Class name **`AR_CP_OS_TASK`** is to be used for the standard OS Task states and **`AR_CP_OSARTI_TASK`** for the enhanced state machine.

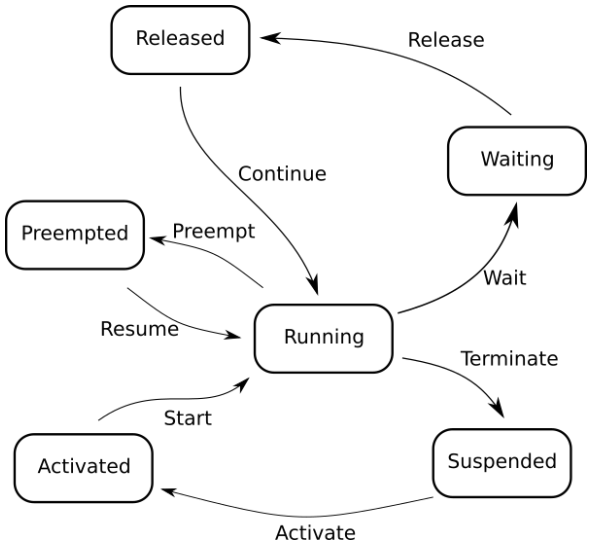

Figure 3.1: Enhanced Task State Diagram
`AR_CP_OSARTI_TASK`

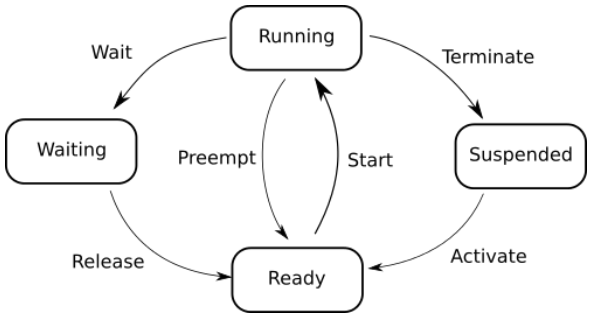

Figure 3.2: Standard Task State Diagram
AR_CP_OS_TASK

3.2 Hooks

The ARTI hook macros are intended to signalize the state transitions of an schedulable entity. By intention they are designed to be macros, rather than functions to be able to minimize the overhead added in computing. The usage of the ARTI Hook can be switched off in the affected BSW Module. For example all OS ARTI macros are switched on with the define **OS_USE_ARTI**. With AUTOSAR release 4.4 the ARTI OS switch is not implemented yet, but will follow with 19-11.

Listing 3.1: Enable or disable ARTI with one define

```
1 #ifdef OS_USE_ARTI
2 #include "arti.h"
3 #else
4 #define ARTI_TRACE(_contextName, _className, _instanceName,  _eventName,
      instanceParameter, eventParameter) ((void)0)
5 #endif
```

The layout of the ARTI Macro is quite generic. The main idea therefore is that users can also define their own macros. However, this approach is currently not fully standardized. For all standardized scheduling state machines, the **_className** maps the macro to the state machine, while **_eventName** describes the state transistion. The **_contextName** should describe whether Interrupts are disabled or not, during the reading of the macro data while execution. Three different modes are possible, **_USER** which indicates that the hook implementer can not disable interrupts and needs to provide correcting postprocessing in case of an interruption. **_NOSUSP** indicates that the macro will be executed in an context where interrupts are locked. **_SPRVSR** indicates that the hook makro can disable the interrupts itself. The **_instanceName** should give information to which OS (name) the Task belongs to.

Listing 3.2: Preprocessor conversion example for ARTI macros

```
1 #define ARTI_TRACE(_contextName, _className, _instanceName, _eventName,
      instanceParameter, eventParameter)  \
2 ARTI_TRACE__ ## _contextName ## __ ## _className ## __ ## _instanceName
      ## __ ## _eventName((instanceParameter),(eventParameter))
```

The **instanceParameter** and **eventParameter** are not literals such as all other macro parts **(_....Name)**. The **instanceParameter** is used to handle the CoreId parameter, while the TaskId can be accessed via the **eventParameter**.

Listing 3.3: ARTI trace macro for OS Task Wait

```
1 ARTI_TRACE(NOSUSP, AR_CP_OSARTI_TASK, OS, OsTask_Wait, CoreId, TaskId);
```

At compile time, the preprocessor will replace the generic macro with all specific ARTI macros, if they have been implemented with an tracing vendor specific instrumentation. The double underscore should help to parse the macros easier.

Listing 3.4: ARTI trace macro implementation for OS Task Wait

```
1 #define ARTI_TRACE__NOSUSP__AR_CP_OSARTI_TASK__OS__OsTask_Wait(CoreId,
    TaskId) {;}
```

3.3 ECUC

The following section focuses on the ECUC representation of ARTI for OS tracing use-case. To configure and store the ARTI configuration for the hook based tracing the following ECUC containers are necessary: ARTI and ArtiOs. The ARTI container stores the Id mapping for Tasks and Cores as well as all available OS hook macros (figure 3.3). The ArtiOs container stores all configured OS Tasks and references in the ArtiOsInstance EcucParamConfContainer all enabled hook macros, see therefore figure 3.4.

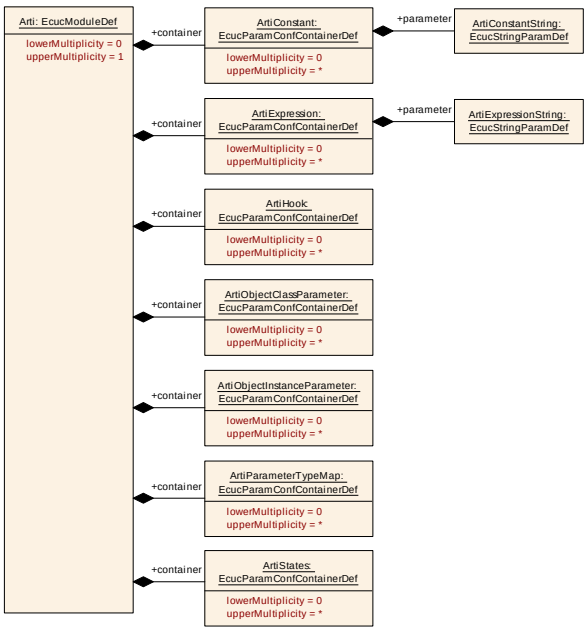

Figure 3.3: Arti EcucParamConfContainer Class Diagram

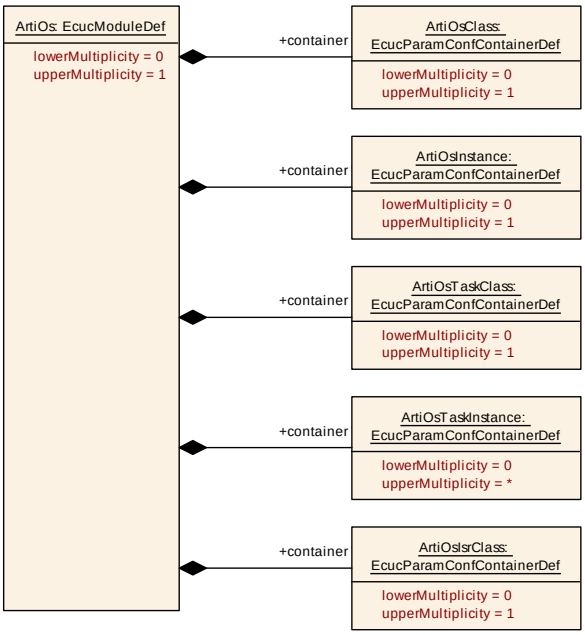

Figure 3.4: ArtiOs EcucParamConfContainer Class Diagram

Examples for this use-case are available for the Arti EcucParamConfContainer in listing 5.4, as well as for the ArtiOs EcucParamConfContainer in listing 5.5.

class Arti	
ECUC Container Value	**Note**
ArtiConstant	
ArtiExpression	Contains the OS expression string, such as Task State expression
ArtiHook	Model representation of the ARTI Hook macros
ArtiObjectClassParameter	Describes the properties of an ArtiObject, such as the CurrentRunningTask needs to references the TypeMap to evaluate the Id or Expression
ArtiObjectInstanceParameter	Instantiates an ArtiObject, like CurrentRunningTask variable for Core 0
ArtiParameterTypeMap	TypeMap for translating Task Ids to Task Names for example

Arti Class ECUC Container Values

class ArtiOs	
ECUC Container Value	**Use-Case**
ArtiOsClass	
ArtiOsInstance	OS Expressions, such as Task State
ArtiOsTaskClass	Model representation of the ARTI Hook macros
ArtiOsTaskInstance	Instantiates a Task, with reference to the OS

ArtiOs Class ECUC Container Values

Current Limitations

The ECUC Container description will change with AUTOSAR Release 19-11. With the class diagrams here AUTOSAR version 4.4 is described. In this version there are still some shortcomings which are not modeled. There is currently no ARTI representation of an ISR.

4 RTE

An ARTI compliant RTE specification is currently in development. Most probably ARTI will use the VFB Tracing Hooks as an own trace client. The release of such an approach is planned for AUTOSAR release 20-11.

5 Examples

The following two examples are intended to provide an example ARTI ECUC configuration for two use-cases.

5.1 Static Debugging

The ARTI ECU Configuration Parameter Container are intended to be configurable in such way that not all container parameters are necessary to be configured. The follwing listings show a minimal example how the three containers can be configured.

ARTI ECUConfiguration Parameters Container Arti

The Arti container stores the available Os variables or expressions to track the task states.

Listing 5.1: ARTI ECUC Container ARXML Listing for Arti

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0 AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>StaticDebugging_ARTI_ECUC</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>Arti</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Arti</
              DEFINITION-REF>
          <IMPLEMENTATION-CONFIG-VARIANT>PRECONFIGURED-CONFIGURATION</
              IMPLEMENTATION-CONFIG-VARIANT>
          <CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiExpression_Core0_CurrentTask</SHORT-NAME>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/Arti/ArtiExpression</DEFINITION-REF>
              <PARAMETER-VALUES>
                <ECUC-TEXTUAL-PARAM-VALUE>
                  <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/
                      EcucDefs/Arti/ArtiExpression/ArtiExpressionString</
                      DEFINITION-REF>
                  <VALUE>OsCfg_Trace_OsCore_Core0_Dyn.CurrentTask</VALUE>
                </ECUC-TEXTUAL-PARAM-VALUE>
              </PARAMETER-VALUES>
            </ECUC-CONTAINER-VALUE>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>ArtiObjectClassParameter_ArtiHwCore_CurrentTask</
                  SHORT-NAME>
              <DESC>
```

```
        <L-2 L="EN">Current Running AUTOSAR Task.</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
          /EcucDefs/Arti/ArtiObjectClassParameter</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiObjectClassParameter/
              ArtiParameterTypeMapRef</DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
              StaticDebugging_ARTI_ECUC/Arti/
              ArtiParameterTypeMap_TaskExpr</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>ArtiObjectInstanceParameter_Core0_CurrentTask</
          SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
          /EcucDefs/Arti/ArtiObjectInstanceParameter</DEFINITION-REF
          >
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
              EcucDefs/Arti/ArtiObjectInstanceParameter/
              ArtiExpressionRef</DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
              StaticDebugging_ARTI_ECUC/Arti/
              ArtiExpression_Core0_CurrentTask</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
  </CONTAINERS>
</ECUC-MODULE-CONFIGURATION-VALUES>
```

ARTI ECUConfiguration Parameters Container ArtiHardware

The ArtiHardware container is necessary to describe all availabe cores for debugging. Additionally each core can reference the core dependend CurrentRunningTask variable.

Listing 5.2: ARTI ECUC Container ARXML Listing for ArtiHardware

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>ArtiHardware</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/
      ArtiHardware</DEFINITION-REF>
  <IMPLEMENTATION-CONFIG-VARIANT>PRECONFIGURED-CONFIGURATION</
      IMPLEMENTATION-CONFIG-VARIANT>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VectorArtiHwCoreClass</SHORT-NAME>
```

```
        <DESC>
          <L-2>Description</L-2>
        </DESC>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
            /EcucDefs/ArtiHardware/ArtiHwCoreClass</DEFINITION-REF>
        <REFERENCE-VALUES>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                EcucDefs/ArtiHardware/ArtiHwCoreClass/
                ArtiCurrentTaskClassRef</DEFINITION-REF>
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                StaticDebugging_ARTI_ECUC/Arti/
                ArtiObjectClassParameter_ArtiHwCore_CurrentTask</VALUE
                -REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </ECUC-CONTAINER-VALUE>
      <ECUC-CONTAINER-VALUE>
        <SHORT-NAME>VectorArtiHwCore_0</SHORT-NAME>
        <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
            /EcucDefs/ArtiHardware/ArtiHwCoreInstance</DEFINITION-REF>
        <REFERENCE-VALUES>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                EcucDefs/ArtiHardware/ArtiHwCoreInstance/
                ArtiCurrentTaskInstanceRef</DEFINITION-REF>
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                StaticDebugging_ARTI_ECUC/Arti/
                ArtiObjectInstanceParameter_Core0_CurrentTask</VALUE-
                REF>
          </ECUC-REFERENCE-VALUE>
          <ECUC-REFERENCE-VALUE>
            <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                EcucDefs/ArtiHardware/ArtiHwCoreInstance/
                ArtiEcucCoreRef</DEFINITION-REF>
            <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/EcuC/
                EcucHardware/EcucCoreDefinition_C0</VALUE-REF>
          </ECUC-REFERENCE-VALUE>
        </REFERENCE-VALUES>
      </ECUC-CONTAINER-VALUE>
    </CONTAINERS>
  </ECUC-MODULE-CONFIGURATION-VALUES>
```

ARTI ECUConfiguration Parameters Container ArtiOs

The ArtiOs container in this simple example is just necessary to describe which task should be tracked for Os debugging in this example and references the task in the Os container. This is basically a duplication of information, but used to substitute the ORTI file.

Listing 5.3: ARTI ECUC Container ARXML Listing for ArtiOs

```
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>ArtiOs</SHORT-NAME>
          <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/ArtiOs</
              DEFINITION-REF>
          <CONTAINERS>
            <ECUC-CONTAINER-VALUE>
              <SHORT-NAME>VendorArtiOsTaskInstance_IdleTask_C0</SHORT-NAME>
              <DESC>
                <L-2 L="EN">ARTI representation of EcuC Task &quot;
                    IdleTask_C0&quot;.</L-2>
              </DESC>
              <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
                  /EcucDefs/ArtiOs/ArtiOsTaskInstance</DEFINITION-REF>
              <REFERENCE-VALUES>
                <ECUC-REFERENCE-VALUE>
                  <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                      EcucDefs/ArtiOs/ArtiOsTaskInstance/ArtiOsTaskEcucRef</
                      DEFINITION-REF>
                  <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/Os/
                      IdleTask_C0</VALUE-REF>
                </ECUC-REFERENCE-VALUE>
              </REFERENCE-VALUES>
            </ECUC-CONTAINER-VALUE>
          </CONTAINERS>
        </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

5.2 OS Task Tracing

ARTI ECUConfiguration Parameters Container Arti

The Arti container stores the TypeMaps for TaskId and CoreId, as well as the available OS Hooks and their layout.

Listing 5.4: ARTI ECUC Container ARXML Listing for Arti

```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0" xmlns:xsi="http://www.w3.
    org/2001/XMLSchema-instance" xsi:schemaLocation="http://autosar.org/
    schema/r4.0␣AUTOSAR_00046.xsd">
  <AR-PACKAGES>
    <AR-PACKAGE>
      <SHORT-NAME>OsTaskTracing_ARTI_ECUC</SHORT-NAME>
      <ELEMENTS>
        <ECUC-MODULE-CONFIGURATION-VALUES>
          <SHORT-NAME>Arti</SHORT-NAME>
```

```
<DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/Arti</
   DEFINITION-REF>
<IMPLEMENTATION-CONFIG-VARIANT>PRECONFIGURED-CONFIGURATION</
   IMPLEMENTATION-CONFIG-VARIANT>
<CONTAINERS>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>ArtiHook_ArtiOs_TaskRelease</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
       /EcucDefs/Arti/ArtiHook</DEFINITION-REF>
    <PARAMETER-VALUES>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/ArtiHookContext</DEFINITION-REF
           >
        <VALUE>NOSUSP</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/ArtiHookClass</DEFINITION-REF>
        <VALUE>AR_CP_OS_TASKSCHEDULER</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/ArtiHookEventName</DEFINITION-
           REF>
        <VALUE>OsTask_Release</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/ArtiHookInstance</DEFINITION-
           REF>
        <VALUE>VectorOsOs</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
    </PARAMETER-VALUES>
    <REFERENCE-VALUES>
      <ECUC-REFERENCE-VALUE>
        <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/ArtiHookEventParameterTypeRef</
           DEFINITION-REF>
        <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
           OsTaskTracing_ARTI_ECUC/Arti/
           ArtiParameterTypeMap_TaskId</VALUE-REF>
      </ECUC-REFERENCE-VALUE>
      <ECUC-REFERENCE-VALUE>
        <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
           EcucDefs/Arti/ArtiHook/
           ArtiHookInstanceParameterTypeRef</DEFINITION-REF>
        <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
           OsTaskTracing_ARTI_ECUC/Arti/
           ArtiParameterTypeMap_CoreId</VALUE-REF>
      </ECUC-REFERENCE-VALUE>
    </REFERENCE-VALUES>
  </ECUC-CONTAINER-VALUE>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>ArtiParameterTypeMap_CoreId</SHORT-NAME>
```

```
<DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
    /EcucDefs/Arti/ArtiParameterTypeMap</DEFINITION-REF>
<SUB-CONTAINERS>
  <ECUC-CONTAINER-VALUE>
    <SHORT-NAME>Core0</SHORT-NAME>
    <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
      AUTOSAR/EcucDefs/Arti/ArtiParameterTypeMap/
      ArtiParameterTypeMapPair</DEFINITION-REF>
    <PARAMETER-VALUES>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
          /EcucDefs/Arti/ArtiParameterTypeMap/
          ArtiParameterTypeMapPair/
          ArtiParameterTypeMapPairInput</DEFINITION-REF>
        <VALUE>0</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
      <ECUC-TEXTUAL-PARAM-VALUE>
        <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
          /EcucDefs/Arti/ArtiParameterTypeMap/
          ArtiParameterTypeMapPair/
          ArtiParameterTypeMapPairOutput</DEFINITION-REF>
        <VALUE>OsCore_Core0</VALUE>
      </ECUC-TEXTUAL-PARAM-VALUE>
    </PARAMETER-VALUES>
  </ECUC-CONTAINER-VALUE>
</SUB-CONTAINERS>
</ECUC-CONTAINER-VALUE>
<ECUC-CONTAINER-VALUE>
  <SHORT-NAME>ArtiParameterTypeMap_TaskId</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
    /EcucDefs/Arti/ArtiParameterTypeMap</DEFINITION-REF>
  <SUB-CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>INVALID_TASK</SHORT-NAME>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
        AUTOSAR/EcucDefs/Arti/ArtiParameterTypeMap/
        ArtiParameterTypeMapPair</DEFINITION-REF>
      <PARAMETER-VALUES>
        <ECUC-TEXTUAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
            /EcucDefs/Arti/ArtiParameterTypeMap/
            ArtiParameterTypeMapPair/
            ArtiParameterTypeMapPairInput</DEFINITION-REF>
          <VALUE>22</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
        <ECUC-TEXTUAL-PARAM-VALUE>
          <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
            /EcucDefs/Arti/ArtiParameterTypeMap/
            ArtiParameterTypeMapPair/
            ArtiParameterTypeMapPairOutput</DEFINITION-REF>
          <VALUE>INVALID_TASK</VALUE>
        </ECUC-TEXTUAL-PARAM-VALUE>
      </PARAMETER-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>IdleTask_C0</SHORT-NAME>
```

```
            <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
               AUTOSAR/EcucDefs/Arti/ArtiParameterTypeMap/
               ArtiParameterTypeMapPair</DEFINITION-REF>
            <PARAMETER-VALUES>
              <ECUC-TEXTUAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
                   /EcucDefs/Arti/ArtiParameterTypeMap/
                   ArtiParameterTypeMapPair/
                   ArtiParameterTypeMapPairInput</DEFINITION-REF>
                <VALUE>10</VALUE>
              </ECUC-TEXTUAL-PARAM-VALUE>
              <ECUC-TEXTUAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
                   /EcucDefs/Arti/ArtiParameterTypeMap/
                   ArtiParameterTypeMapPair/
                   ArtiParameterTypeMapPairOutput</DEFINITION-REF>
                <VALUE>IdleTask_C0</VALUE>
              </ECUC-TEXTUAL-PARAM-VALUE>
            </PARAMETER-VALUES>
            <REFERENCE-VALUES>
              <ECUC-REFERENCE-VALUE>
                <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
                   EcucDefs/ArtiOs/ArtiOsTaskInstance/
                   ArtiOsTaskEcucRef</DEFINITION-REF>
                <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
                   OsTaskTracing_ARTI_ECUC/ArtiOs/
                   VendorArtiOsTaskInstance_IdleTask_C0</VALUE-REF>
              </ECUC-REFERENCE-VALUE>
            </REFERENCE-VALUES>
          </ECUC-CONTAINER-VALUE>
          <ECUC-CONTAINER-VALUE>
            <SHORT-NAME>notask</SHORT-NAME>
            <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/
               AUTOSAR/EcucDefs/Arti/ArtiParameterTypeMap/
               ArtiParameterTypeMapPair</DEFINITION-REF>
            <PARAMETER-VALUES>
              <ECUC-TEXTUAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
                   /EcucDefs/Arti/ArtiParameterTypeMap/
                   ArtiParameterTypeMapPair/
                   ArtiParameterTypeMapPairInput</DEFINITION-REF>
                <VALUE>0xff</VALUE>
              </ECUC-TEXTUAL-PARAM-VALUE>
              <ECUC-TEXTUAL-PARAM-VALUE>
                <DEFINITION-REF DEST="ECUC-STRING-PARAM-DEF">/AUTOSAR
                   /EcucDefs/Arti/ArtiParameterTypeMap/
                   ArtiParameterTypeMapPair/
                   ArtiParameterTypeMapPairOutput</DEFINITION-REF>
                <VALUE>NO_TASK</VALUE>
              </ECUC-TEXTUAL-PARAM-VALUE>
            </PARAMETER-VALUES>
          </ECUC-CONTAINER-VALUE>
        </SUB-CONTAINERS>
      </ECUC-CONTAINER-VALUE>
    </CONTAINERS>
  </ECUC-MODULE-CONFIGURATION-VALUES>
```

ARTI ECUConfiguration Parameters Container ArtiOs

The ArtiOs container in this simple example is just necessary to describe which task should be tracked for OS tracing in this example and references the task in the OS container. This is basically a duplication of information, but used to substitute the ORTI file.

Listing 5.5: ARTI ECUC Container ARXML Listing for ArtiOs

```
<ECUC-MODULE-CONFIGURATION-VALUES>
  <SHORT-NAME>ArtiOs</SHORT-NAME>
  <DEFINITION-REF DEST="ECUC-MODULE-DEF">/AUTOSAR/EcucDefs/ArtiOs</
     DEFINITION-REF>
  <CONTAINERS>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiOsInstance</SHORT-NAME>
      <DESC>
        <L-2 L="EN">Actual values of the Vector OS</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
         /EcucDefs/ArtiOs/ArtiOsInstance</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
             EcucDefs/ArtiOs/ArtiOsInstance/ArtiOsEcucRef</
             DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/Os/
             OsOS</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
             EcucDefs/ArtiOs/ArtiOsInstance/ArtiOsTaskHookRef</
             DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/
             OsTaskTracing_ARTI_ECUC/Arti/
             ArtiHook_ArtiOs_TaskRelease</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
    </ECUC-CONTAINER-VALUE>
    <ECUC-CONTAINER-VALUE>
      <SHORT-NAME>VendorArtiOsTaskInstance_IdleTask_C0</SHORT-NAME>
      <DESC>
        <L-2 L="EN">ARTI representation of EcuC Task &amp;quot;
           IdleTask_C0&amp;quot;.</L-2>
      </DESC>
      <DEFINITION-REF DEST="ECUC-PARAM-CONF-CONTAINER-DEF">/AUTOSAR
         /EcucDefs/ArtiOs/ArtiOsTaskInstance</DEFINITION-REF>
      <REFERENCE-VALUES>
        <ECUC-REFERENCE-VALUE>
          <DEFINITION-REF DEST="ECUC-REFERENCE-DEF">/AUTOSAR/
             EcucDefs/ArtiOs/ArtiOsTaskInstance/ArtiOsTaskEcucRef</
             DEFINITION-REF>
          <VALUE-REF DEST="ECUC-CONTAINER-VALUE">/ActiveEcuC/Os/
             IdleTask_C0</VALUE-REF>
        </ECUC-REFERENCE-VALUE>
      </REFERENCE-VALUES>
```

```
            </ECUC-CONTAINER-VALUE>
          </CONTAINERS>
        </ECUC-MODULE-CONFIGURATION-VALUES>
      </ELEMENTS>
    </AR-PACKAGE>
  </AR-PACKAGES>
</AUTOSAR>
```

6 Outlook

With ongoing development, ARTI shall provide more and more features for users to understand their AUTOSAR project's run-time behavior. Besides OS and RTE, ARTI is aiming to support in future OS protection Hooks and other fault recognition interfaces.

Apart from that, Adaptive Platform will also be addressed.

7 Document Information

Known Limitations

- Initial Version, no known limitations yet.

7.1 Related documentation

7.1.1 Input documents & related standards and norms

[1] Specification of AUTOSAR Run-Time Interface
AUTOSAR_SWS_ClassicPlatformARTI

[2] Specification of Operating System
AUTOSAR_SWS_OS

[3] Specification of RTE Software
AUTOSAR_SWS_RTE

7.1.2 Related specification

This document should explain the standard addressed in [1, SWS ClassicPlatformARTI]. ARTI focuses heavily on scheduling and run-time analysis and therefore affects [2, SWS OS] and [3, SWS RTE].