Seminar "'Automotive Open Systems Architecture"'

Modeling and Development of AUTOSAR Systems using SystemDesk

Sebastian Wätzoldt

Hasso-Plattner-Institut for IT Systems Engineering
at the University of Potsdam
Prof.-Dr-Helmert-Straße 2-3
14482 Potsdam
Professorship Systemanalyse und Modellierung
Prof. Dr. Holger Giese

Tutor:
Stefan Neumann

Abstract. The development and modeling of automotive software systems is one of the most difficult challenges in the automobile industry. The increasing costs of integration problems in soft- and hardware and the interaction of different manufacturers and suppliers leads to a request of interoperability and standardization. At this time, 80 percent of today's innovations in the automotive area are realized with the help of software. One approach to handle the enormous complexity of building such systems is the upcoming AUTOSAR standard. This paper investigates the interaction of modeling and developing AUTOSAR conform systems using the special software tool SystemDesk. A look inside the development process of automotive software systems using the AUTOSAR framework is given together with a walk through a real world example in a concrete application domain.

Contents

List of Abbreviations

AUTOSAR	AUTomotive Open System ARchitecture
CAN	Controller Area Network
DBC	Description file for CAN bus
ECU	Electronic Control Unit
LDT	LIN Description File
LIN	Local Interconnect Network
RTE	(AUTOSAR) Runtime Environment
SIL	Software in the loop
SWC	Software Component
VFB	Virtual Functional Bus

1 Introduction

The enormous increasing complexity in automotive software systems leads to a new innovative standard called AUTOSAR. This paper introduces modeling and developing techniques for this upcoming standard using the software tool SystemDesk. After a short introduction of AUTOSAR and SystemDesk in the next two sections, the modeling steps and development approaches are discussed in 2. There, all necessary steps and important features of the tool are presented and at the end some interoperability issues are pointed out. The paper closes with a short summary in 3 identifying some discussion points and possible future work motivations.

1.1 The AUTOSAR framework

The **AUT**omotive **O**pen **S**ystem **AR**chitecture (AUTOSAR) is an international group of automobile manufacturer, supplier and software companies with the aim to establish a new open standard for software architecture in the automobile. This new standard should lead to the controllability of an enormous growing of complexity in modern car architectures. The main concept of the AUTOSAR framework is the replaceability of software components (SWC), which could be used in different car platforms (technical overview see [7]). By the reason of cooperation between manufacturer and many suppliers, the use of standardized interfaces between components and a methodology for the development process is highly recommended. At a conceptual view the AUTOSAR standard is a middleware divide into several layers (see figure 1). At the highest abstraction layer there are the different software components. They encapsulate variable application software running at the AUTOSAR infrastructure. Well-defined interfaces enable the interaction and communication with the components. At this level of abstraction all communication mechanism are processed over the virtual functional bus (VFB). This concept offers a virtual integration into the whole system in a very early modeling stage of the development (specification about the VFB see [8]). The VFB will be implemented later by the AUTOSAR runtime environment (RTE). For executing a software component, it must be allocated to an electronic control unit (ECU). The lowest layer encapsulates the hardware in a system. Therefore, it contains the different ECUs and busses. In the middle of that architecture there is the basic software including operating system functionality, complex device driver and several interfaces of a microcontroller abstraction, communication and services. More detailed information about concepts and implementation of AUTOSAR are in [1] (Technical Foundations for the Development of Automotive Embedded Systems) and at the AUTOSAR website [6].

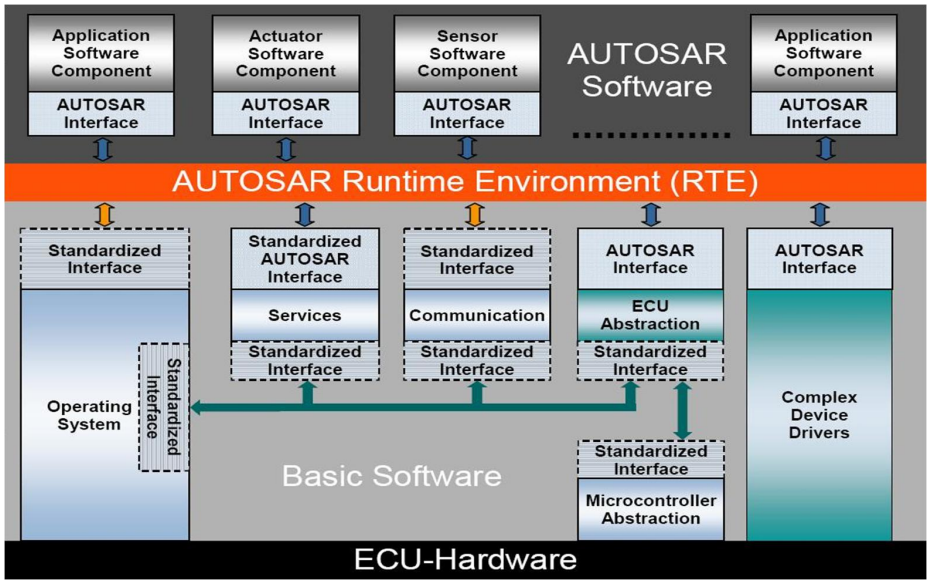


Fig. 1. The layered architecture of the AUTOSAR framework.

1.2 SystemDesk a modeling tool for AUTOSAR

SystemDesk is a software architecture tool for modeling AUTOSAR systems. It is developed by dSpace and should enable an early verification and visualizing of automotive software architectures. With the help of the tool it is possible to structure different parts of the whole system and get an impression of the communication and interaction of all components in it. dSpace is a company which develops different hardware and software solutions for mechatronic systems. It has different project centers all over the world for example in Germany, France or China. If you look at the whole methodology of AUTOSAR, SystemDesk provides one opportunity building up an AUTOSAR system. The different features and steps which are necessary to develop such a system will be introduced in the next chapter.

2 Modeling AUTOSAR using SystemDesk

The focus of this chapter is, to explain different modeling steps with SystemDesk to get a whole AUTOSAR system. After presenting an overview and a concrete example, different parts of the systems modeling process are introduced. At the end some aspects of interoperability with other tools are discussed. All steps and functionalities could be reproduced using the tool and for example a given tutorial project in it.

2.1 Model and Development

To get a feeling to which category SystemDesk belongs, figure 2 shows all necessary steps and dependencies between partial results by following the AUTOSAR methodology (more information about the methodology in [4]). There are three parts (seen as a process flow) of the methodology with the intention of modeling a software architecture. The behavior modeling (shown at the top of the figure) must be done to get the application software. The next main activity in the flow would be used for building up the AUTOSAR conform System (middle layer) and at the end of the process, the basic software can be configured and generated. The main focus of the tool is modeling a software architecture. Therefore, no ECU or bus development is possible (more about that in section 2.1.3). Before we start with working in SystemDesk some kind of behavior modeling should be done to get the implementation (C code) for every SWC. Furthermore, it is also possible to import an existing SWC specification via an AUTOSAR XML file. After that the system can be build up (that is shown in the middle part with system modeling). This part of the flow will be divided again into different modeling steps provided by SystemDesk (see section 2.1.2, 2.1.3 and 2.1.4). At the end, the configuration and generation of the basic software (including operating system functionality) must be done for each ECU. The figure shows that SystemDesk has the main focus at the middle part but overlaps in the two other layers by providing interoperability interfaces (that is discuss in section 2.3).

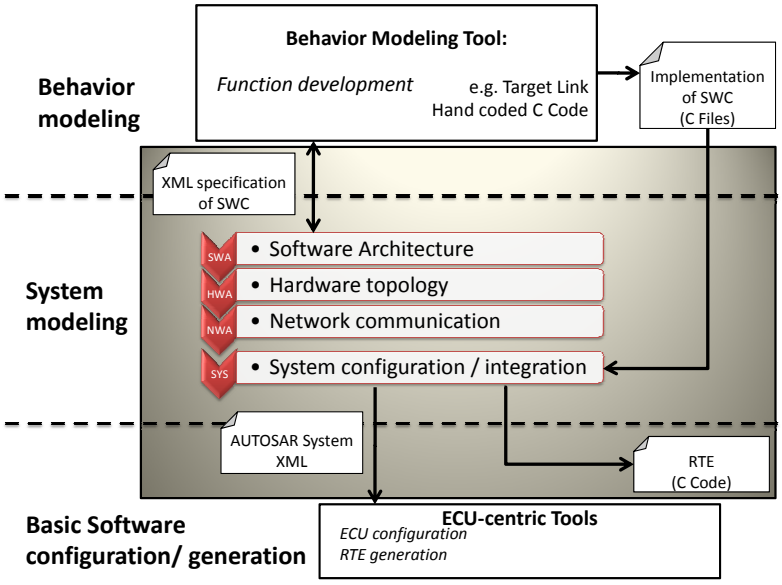


Fig. 2. Integration of SystemDesk in the AUTOSAR tool chain.

2.1.1 Direction Indicator Example In SystemDesk there is a tutorial project, which is choosen to explain the main modeling concepts and possibilities of that tool to model AUTOSAR conform systems. Figure 3 shows some important units for that example scenario. There are two sensors, one sensor indicates whether the driver wants to turn left or right (switch sensor) and the other checks if the warn light blinker should be enabled or not. In addition, we have two actuators one for the right and one for the left blinker. Different kinds of electronic control units communicate with each other using a special network infrastructure. Furthermore, there are a lot of software components, runnables, ports and interfaces in the whole system. They will be introduced in detail in the next sections. The main intention of this example is, that the central body ECU detects, with the assistance of the sensors, the decisions of the driver and forward special messages to the correct actuator ECU (left/ right blinker or both). As communication hardware a controller area network (CAN-bus) is used.

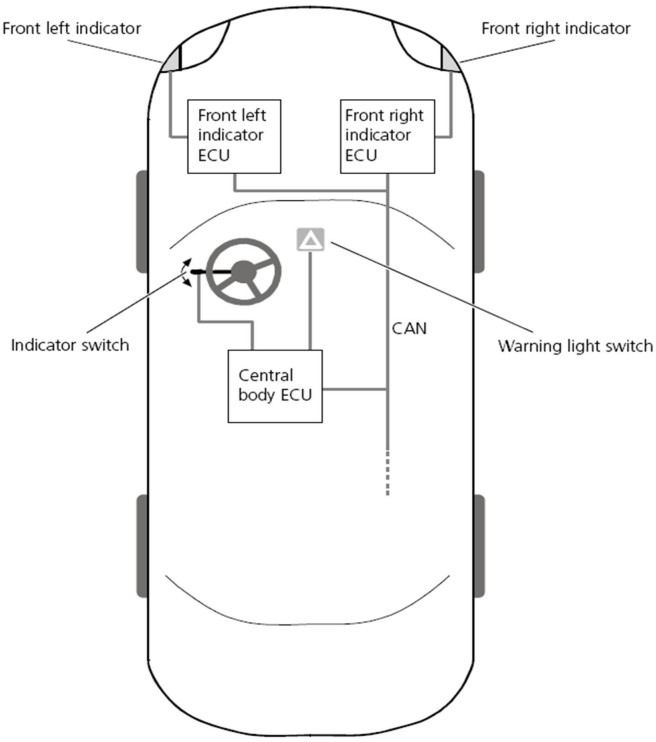


Fig. 3. Schematic representation of a simple direction indicator with different sensors and actuators.

2.1.2 Software Architecture The software architecture in one of three concurrent steps in AUTOSAR system modeling (see figure 2). SystemDesk provides a graphical environment for the software architecture. If you remember the layered AUTOSAR architecture, in this case the upper software layer is covered by the tool. At this abstraction level software components interact with each other via the virtual functional bus. By modeling the software architecture, the same view is provided. In a first step you can build up the system, creating different kinds of software components. To aggregate different systems parts together, you can structure it with grouping different SWCs into one or more compositions. Ports with interfaces are used to enable a communication between SWCs. Therefore, connectors link ports under the abstraction of the VFB. It is possible to model a specific interaction behavior of the system by defining a set of runnables in each SWC (conceptual view see figure 4).

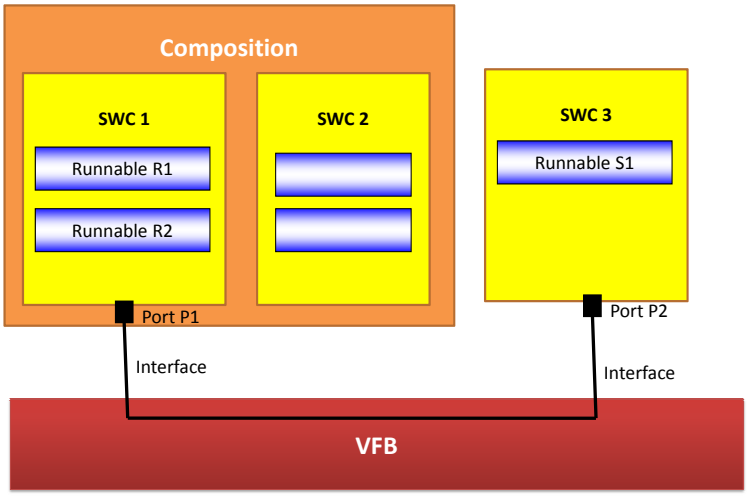


Fig. 4. Conceptual interacting of SWCs, ports, interfaces and runnables.

To get an impression how this could look like in SystemDesk, please have a view at figure 5. There are two important areas of that tool at this time. At the left hand side, you can see the project manager. It contains a library with all created components, ports and interfaces and different other parts which belongs to a complete system. One of these parts is the software architecture, but also other parts like the hardware topology and the network communication (see 2.1.3 and 2.1.4) are shown. In the middle of that figure an overview about

the whole software architecture of the system is given. It is possible to identify four SWCs. Two of them at the left side encapsulate the turn switch sensor and the warn light button. The two other SWCs are for the left and right blinker. In the middle of this overview there is a composition (orange) which contains the blinker logic of the example. Several ports are linked with connectors to enable the communication and interaction of the SWCs.

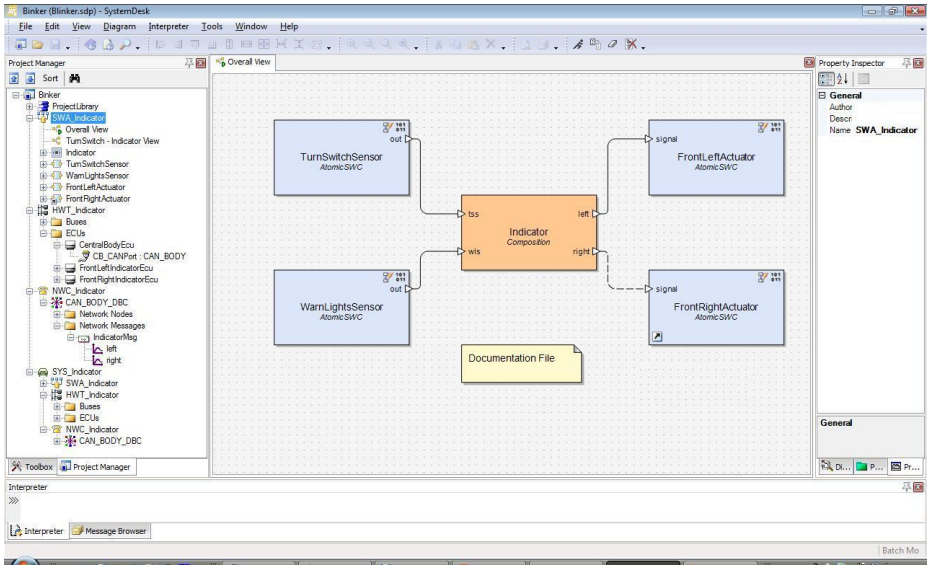


Fig. 5. SystemDesk software architecture overview.

To have a deeper look into the front left actuator see figure 6. For every SWC a name and a description can be modeled. At the left side one sender/receiver interface is shown with one data element in it. The element is derived from a boolean base data type and indicates whether the actuator should blink (value is 1) or not (value is 0). The port with the name "'signal"' at the SWC uses this interface as receiver (ingoing triangle notation).

The last step at this software architecture level is to specify the behavior. This could be done by creating runnables (see figure 7). In a standard dialog it is possible to set a name, description and the category of the runnable. Here also specific other details could be specified like some triggers when the runnable should be executed or the data access for it. For instance, in this special example the runnable has read only access for the data value in the interface. It is enough, because the actuator only reads whether he should blink or not.

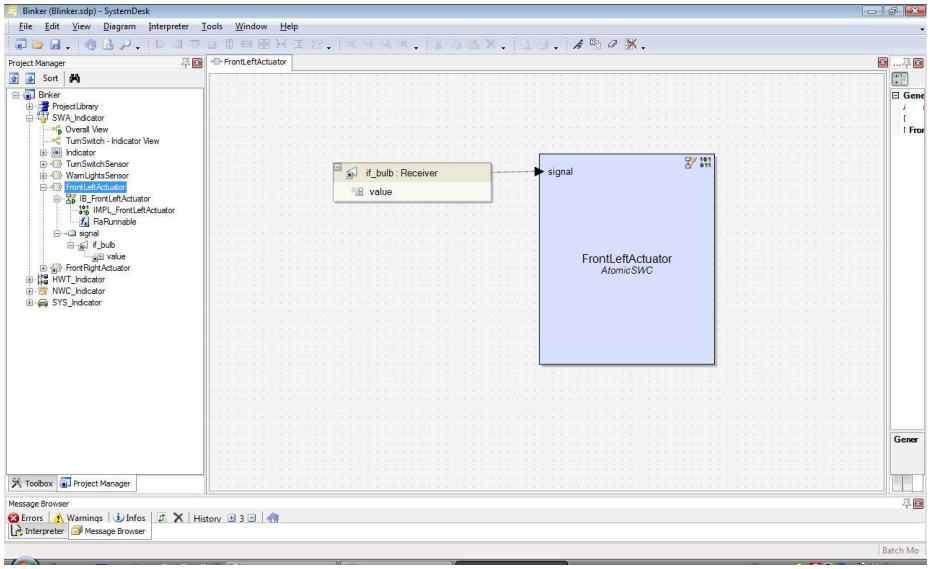


Fig. 6. A SWC with port, interface and runnable specification.

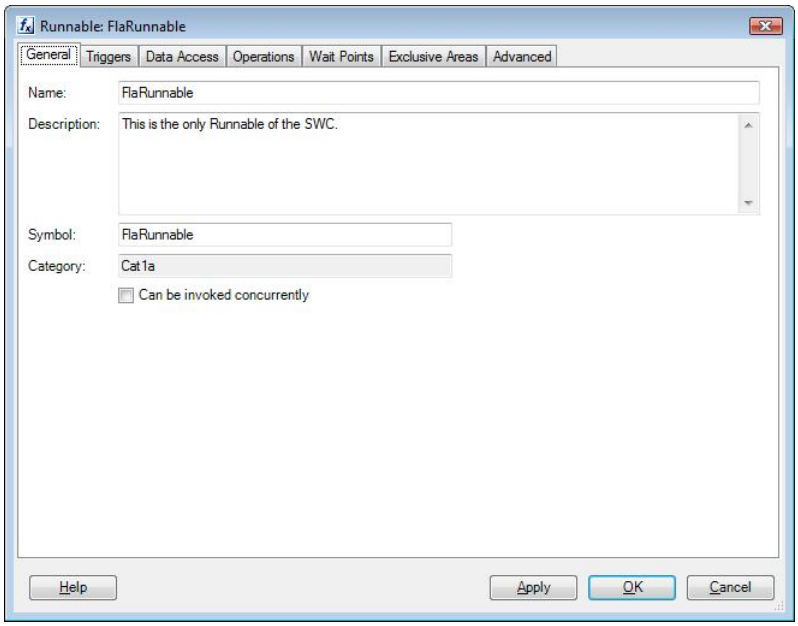


Fig. 7. The standard dialog for a runnable in SystemDesk.

The implementation of the SWC can be imported from external C code files. A possible implementation for a runnable in that file could look like this code fragment:

```
1  // One possible implementation of the Runnable:
2
3  void FlaRunnable(void)
4  {
5      // read the data value from interface
6      value = Rte_IRead_FlaRunnable_signal_value();
7
8      // do something with value
9     ...
10 }
```

Line 3 defines a function with the same name of the runnable. The implementation could be very different, here a RTE function call is done (line 6) to read the data value from the specified interface.

2.1.3 Hardware Topology Remember the AUTOSAR layered architecture, at the lowest level there is the hardware. For constructing a hardware topology in SystemDesk an appropriated set of ECUs must be choosen and different communication mechanisms have to be selected. After that, it is possible to build up the hardware structure via connecting busses and ECUs. At the end, specific hardware settings can be configured. A valid combination of ECUs and busses could look like figure 8.

The first part at hardware modeling is the communication infrastructure. SystemDesk provides combinations of CAN and LIN busses. For each, it is possible to specify in a standard dialog for example a name and a speed. For interacting with ECUs each bus must provide communication ports. The second part of the hardware are the ECUs. It is not possible to construct a new ECU from the bottom, SystemDesk provides a list with a lot of controller types. Therefore, a configuration of each ECU running at different special platforms can be set. Connect the ECUs to the busses over the modeled communication ports in a matrix finishes the hardware modeling.

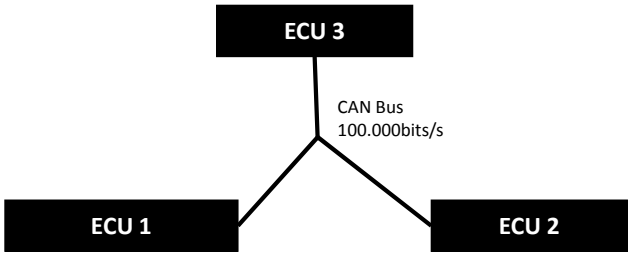


Fig. 8. Harware topoloy example consists of a combination of three ECUs and a CAN bus.

2.1.4 Network Communication A third view on the system completes the modeling process. Conceptually, the communication matrix is independent of the underlying physical bus. Therefore, SystemDesk handles it independently of the hardware topology. The network architecture consists of a communication matrix, network nodes and messages with signals. It is possible to describe different kinds of messages with variable signals in it and map them to the network nodes. Each node is an abstraction of the physical bus structure below and is interconnected with one of them. There are different advantages for this abstraction. At the one hand, network nodes are able to send and receive a set of messages. So it is possible setting up special access rights for example to reject messages and therefore process different communication scenarios. At the other hand SystemDesk provides an import interface for DBC (standard for CAN bus) and LDT (LIN description file) files to automate this step (more information about DBC and LDT see for example [9]). If the communication matrices (such as DBC files) are already available (for example, if the OEM did the network planning and a supplier uses SystemDesk to develop the software for one ECU) they can be imported. In this case only additions like missing messages or signals must be modeled.

2.1.5 System Configuration and Integration After modeling the software architecture, specifying the hardware topology and creating a network communication infrastructure, you can join these three parts together to a whole system. The first step combining the parts is building a system configuration element in SystemDesk and map all needed SWCs to the ECUs where they should run (see figure 9 for conceptual view). Choosing the correct implementation files for each SWC, a communication matrix and a mapping of messages (including the signals) to ports and interfaces are steps which are required for a correct system behavior.

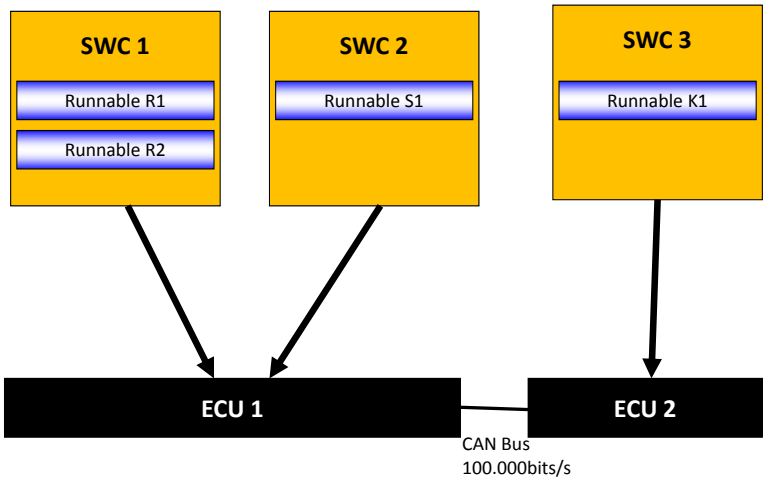


Fig. 9. Mapping SWCs to ECUs to have a unambiguous allocation for the runnables.

The major part for executing the functionality of SWC is still missing at this time. Therefore, the definition of tasks is needed. Figure 10 shows how runnables from the mapped SWCs could be uses to construct different tasks. Each OS task has among other things a category (basic or extended), a priority, a period and scheduling information for the OS.

After specifying the tasks and global system constraints like network issues it is possible to configure the OS in some ways. For example global resources like events or counters, memory, alarm clocks or messages can be specified. All the information are captured from SystemDesk using standard dialogs (see figure 11 for the OS configuration).

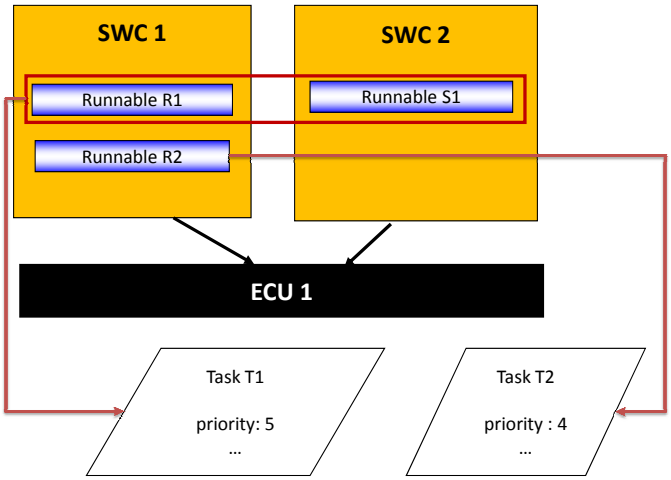


Fig. 10. Building up OS tasks with runnables from the allocated SWC.

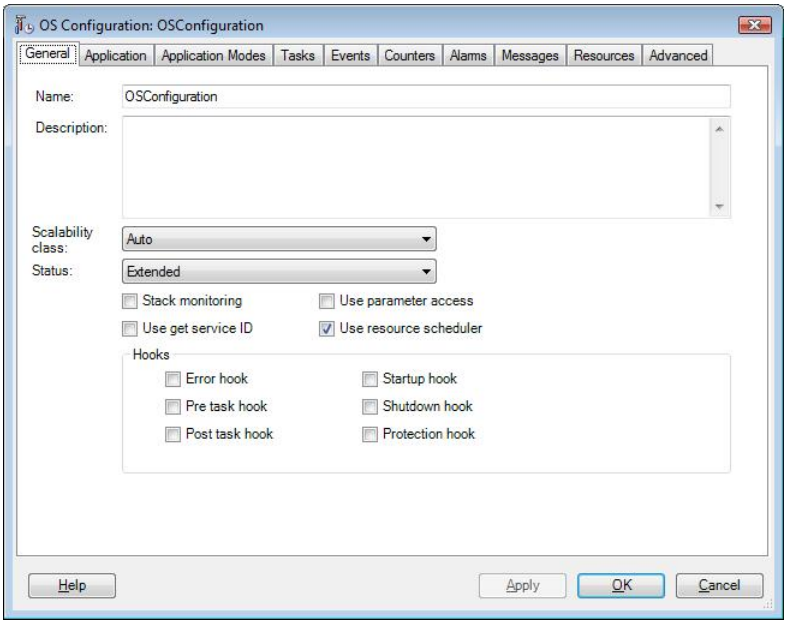


Fig. 11. The dialog of SystemDesk to capture information about OS settings.

2.1.6 Runtime Environment and COM Generation A great benefit of modeling a whole system architecture doing all the steps described above is the possibility of executing the complete system afterwards. Therefore, it is necessary to come down from the view of the VFB to a more concrete / real view of interaction and communication with the help of the runtime environment (RTE). At this point SystemDesk provides several automatic code and configuration generation possibilities. For more information about steps in the methodology of generation the RTE please read [4] and [2].

At the generation of the COM configuration (see figure 12), each signal which is send over a port gets a unique id. That is important to match that signal (and the sent value in the signal) exactly via a COM function call (explained more in detail see figure 15). In SystemDesk it is also possible to aggregate signals from different sources like network nodes and messages and refer them to different groups for the PDU router to perform for example different communication scenarios crossing the COM stack (see [3]).

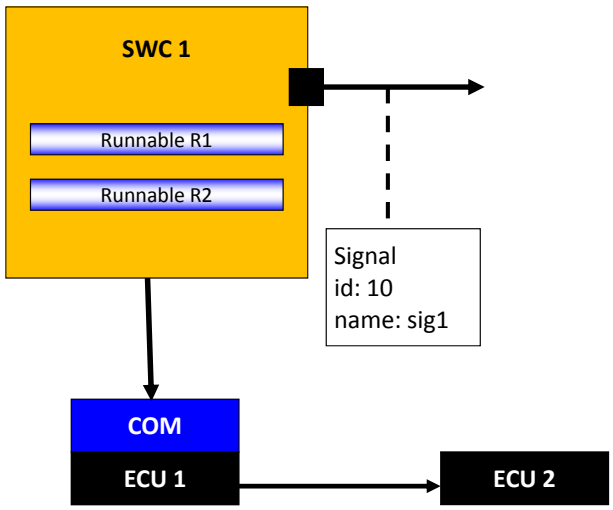
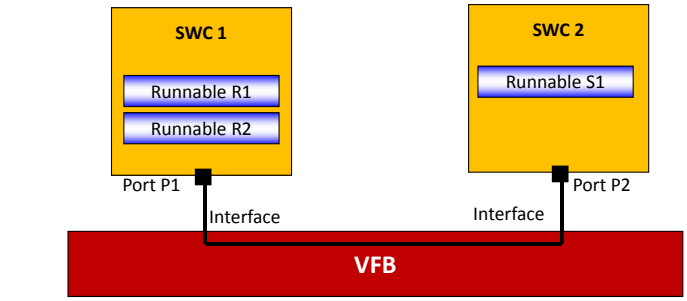


Fig. 12. Mapping unique IDs to each signal crossing a port.

Figure 13 shows the implementation of the behavior of different SWCs. The programmer (or another code generation tool) can implement the runnables using high level function calls (VFB view). In this concrete example a runnable R1 writes a value, a second runnable S1 could read afterwards. Both function calls

do not care about the underling hardware infrastructure. The RTE generator now implements these high level functions but considers the hardware situation.




```
void R1() {
 //…
 RTE_IWrite_R1_myvalue(value);
 //…
}
```

```
void S1() {
//…
value = RTE_IRead_S1_myvalue();
//…
   }
```

Fig. 13. User code for programming at the VFB view.

There are two major possibilities, first it could be that both SWCs are lying at one ECU and an intra communication could happen. The implementation of this scenario would look like it is shown in figure 14. In this case a global value can be written and read, because both SWC uses the same memory resources at the ECU. This implementation should avoid unnecessary overhead traversing the communication stack.

A second scenario could look like figure 15. Each SWC runs on different ECUs and a communication is only possible by sending messages over the bus. In this alternative the RTE function is implemented calling a function of the communication stack. Here the id (number 10 in the picture) takes an important role. Only knowing the id of each signal (the mapping was done at the COM configuration) allows a correct sending and receiving behavior listing at the right port for the id. So, in this second case a full using of the communication stack is necessary to communicate between different ECUs (inter communication).

All these decisions and steps of generating the correct code of the RTE is done full automatically in SystemDesk. The system engineer can now take this code and investigate the modeled AUTOSAR architecture by executing it.

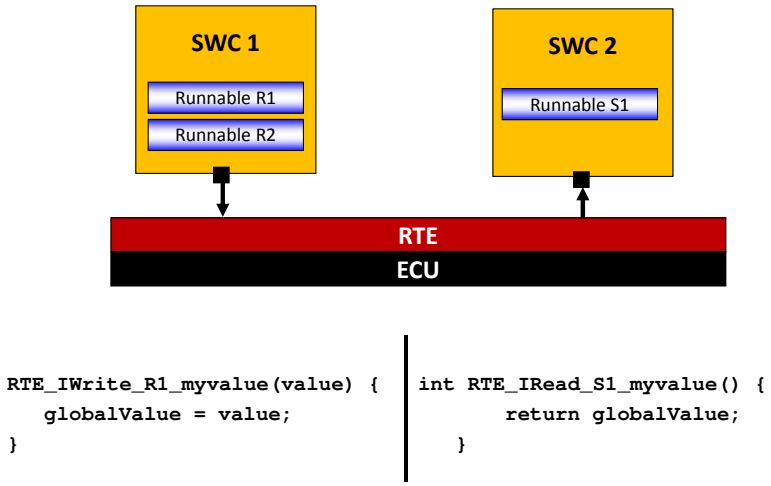


Fig. 14. Implementation of the RTE function if both SWC have an intra communication. Reading and writing of a global value.

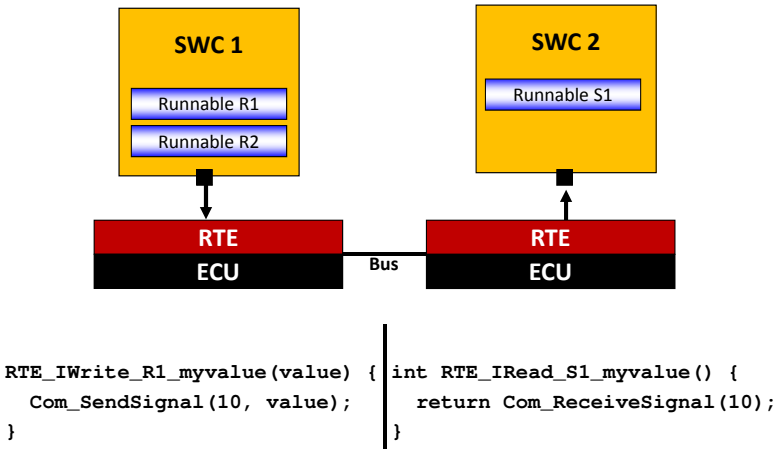


Fig. 15. Implementation of the RTE function if both SWC have an inter communication. Reading and writing of using the communication stack.

2.2 Using more features of SystemDesk

This section will numerate some more features of the modeling tool SystemDesk. All steps, how they are described above, can be automated using a COM (common object model) port provided by the tool. The integrated python interpreter can read and execute external scripts so that parts or a whole system (also RTE generation) can be automated. To get an impression how a script could look like, see the code snipe 1.1. There are three functions for the indication of the expressiveness using the SystemDesk library. The first one (line 1 - 47) creates the software architecture of the systems, the second (line 49 - 69) defines some interfaces at the ports and the last one (line 71- 84) generates the RTE for one specific ECU. After creating the software architecture (line 8, 9) different SWC are specified. For example the "TurnSwitchSensors" component with one port (line 11 - 15) or the indicator component with 4 ports (see line 23 -29). After creating all single components the top level diagram from figure 5 is set up (SWC elements with port connections ll. 33 - 46).

The second function collects at first all SWCs and compositions from the tool (ll. 58 - 60), for adding different kinds of interfaces from the library to them (ll. 63 - 68).

The last function shows how a RTE generation for one ECU is possible. After getting the specific ECU from the project (ll. 79 - 81) a simple function call (l. 84) should do all the work (the precondition to do that is, that all necessary modeling steps are done before, otherwise the tool would throw an error).

source code 1.1. Python code snipe of three little functions creating a software architecture, interfaces and generating the RTE.

```
1  ##########################################
2  ### Create the software architecture ######
3  ##########################################
4  def CreateSoftwareArchitecture():
5      """
6      Creates the software architecture of the indicator system.
7      """
8      TutorialProject.SoftwareArchitectureName = "SWA_Indicator"
9      softwareArchitecture = TutorialProject.SoftwareArchitecture
10
11     # Create SWC "TurnSwitchSensors".
12     turnSwitchSensorSwc = softwareArchitecture.AtomicSoftwareComponents.Add("
           TurnSwitchSensor")
13     turnSwitchSensorSwc.Ports.Add("out")
14     if TutorialProject.Options.WithIo:
15         turnSwitchSensorSwc.Ports.Add("io_tss")
16
17     # Create SWC "WarnLightSensors".
18     warnLightsSensorSwc = softwareArchitecture.AtomicSoftwareComponents.Add("
           WarnLightsSensor")
19     warnLightsSensorSwc.Ports.Add("out")
20     if TutorialProject.Options.WithIo:
21         warnLightsSensorSwc.Ports.Add("io_wls")
22
23     # Create a composition for the "Indicator".
24     indicatorComposition = softwareArchitecture.Compositions.Add("Indicator")
25     indicatorViewTssInPort = indicatorComposition.Ports.Add("tss")
26     indicatorViewWlsInPort = indicatorComposition.Ports.Add("wls")
27     indicatorViewLeftOutPort = indicatorComposition.Ports.Add("left")
28     if TutorialProject.Options.WithRight:
```

```
29          indicatorViewRightOutPort = indicatorComposition.Ports.Add("right")
30
31      ...
32
33      # Create the top-level diagram.
34      overallView = softwareArchitecture.CompositionDiagrams.Add("Overall View"
            )
35
36      grTurnSwitchSensorSwc = overallView.AtomicSoftwareComponents.AddElement(
            turnSwitchSensorSwc)
37      Utilities.SetFillColor(grTurnSwitchSensorSwc, 'lightblue')
38      Utilities.SetPositionAndSize(grTurnSwitchSensorSwc, 80, 60, 180, 100 )
39
40      ...
41
42      overallView.PortConnections.Add(grTurnSwitchSensorOutPort,
            grIndicatorTssInPort)
43      overallView.PortConnections.Add(grWarnLightsSensorOutPort,
            grIndicatorWlsInPort)
44      overallView.PortConnections.Add(grIndicatorLeftInPort,
            grFrontLeftActuatorSignalInPort)
45
46      ## title = overallView.Texts.Add("Overall View")
47      ...
48
49  ##########################################
50  ### Create the interfaces at the ports.####
51  ##########################################
52  def CreateInterfaces():
53      """
54      Create the interfaces.
55      """
56      libFolder = TutorialProject.Project.Library
57
58      # Compositions and SWCs.
59      indicatorComposition = TutorialProject.SoftwareArchitecture.Compositions.
            Item("Indicator")
60      indicatorAtomicSwc = indicatorComposition.AtomicSoftwareComponents.Item("
            IndicatorAtomic")
61      ...
62
63      # Add interfaces to the delegation ports of the Indicator composition.
64      GetInterface("if_tss", indicatorComposition.Ports.Item("tss").
            ReceiverInterfaces, libFolder)
65      GetInterface("if_wls", indicatorComposition.Ports.Item("wls").
            ReceiverInterfaces, libFolder)
66      GetInterface("if_bulb", indicatorComposition.Ports.Item("left").
            SenderInterfaces, libFolder)
67      if TutorialProject.Options.WithRight:
68          GetInterface("if_bulb", indicatorComposition.Ports.Item("right").
                SenderInterfaces, libFolder)
69      ...
70
71  ##########################################
72  ### Create the RTE for one ECU       ######
73  ##########################################
74  def GenerateCentralBodyEcuRte():
75      """
76      Generates the RTE for the CentralBodyEcu.
77      """
78
79      system = TutorialProject.System
80      ecuConfiguration = system.EcuConfigurations.Item("CentralBodyEcuConfig")
81      print "Generating RTE for %s" % ecuConfiguration.Name
82
83      # Start the RTE generation now.
84      ecuConfiguration.StartRteGeneration()
```

The use of python scripts could reduce modeling work, if there are for example standard components which are needed in other systems, too. Utilize a script is only one possibility. Furthermore, SystemDesk provides an internal exchange format for saving and reusing system parts (see next section).

Another feature in the newest version of the tool is, doing software in the loop (SIL) simulations to investigate the interaction and behavior of the system. For more information about simulation in context of AUTOSAR you can read [5].

2.3 Interoperability

At the end of the discussion about SystemDesk and AUTOSAR systems some possibilities about interoperability are pointed out. The use of python scripts like it was explained above gives only a reduced interaction between different suppliers. To overcome these limitations SystemDesk provides the export of the software architecture, single components and compositions and the whole system configuration to a standardized AUTOSAR XML format ([12]). The exported parts of the system could look like the XML listing 1.2. In this part of a XML specification of a system you can see the definition of the SWC "FrontLeftActuator" (ll. 9 -20), a runnable named "FlaRunnable" (ll. 25 -30) and an interface of the SWC (ll. 33-44).

This XML file can be understand from each other tool which is able to read the standardized AUTOSAR XML format. So you can exchange parts of a system and import and working with them in SystemDesk.

source code 1.2. Part of an exported standardized XML file from SystemDesk.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <AUTOSAR xmlns="http://autosar.org/2.1.4"
3  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4  xsi:schemaLocation="http://autosar.org/2.1.4 autosar.xsd">
5    <TOP-LEVEL-PACKAGES>
6      <AR-PACKAGE>
7        <SHORT-NAME>Blinker</SHORT-NAME>
8          <ELEMENTS>
9            <ATOMIC-SOFTWARE-COMPONENT-TYPE>
10             <SHORT-NAME>FrontLeftActuator</SHORT-NAME>
11             <PORTS>
12               <R-PORT-PROTOTYPE>
13                 <SHORT-NAME>signal</SHORT-NAME>
14                 <REQUIRED-COM-SPECS>
15                 ...
16                 </REQUIRED-COM-SPECS>
17                 <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-INTERFACE">/
                       Binker/if_bulb</REQUIRED-INTERFACE-TREF>
18               </R-PORT-PROTOTYPE>
19             </PORTS>
20           </ATOMIC-SOFTWARE-COMPONENT-TYPE>
21         </ELEMENTS>
22         ...
23         <INTERNAL-BEHAVIOR>
24           <RUNNABLES>
25             <RUNNABLE-ENTITY>
26               <SHORT-NAME>FlaRunnable</SHORT-NAME>
27               <CAN-BE-INVOKED-CONCURRENTLY>false</CAN-BE-INVOKED-CONCURRENTLY
                     >
28               ...
```

```
29                <SYMBOL>FlaRunnable</SYMBOL>
30              </RUNNABLE-ENTITY>
31            </RUNNABLES>
32          ...
33           <SENDER-RECEIVER-INTERFACE>
34             <SHORT-NAME>if_bulb</SHORT-NAME>
35            ...
36             <IS-SERVICE>false</IS-SERVICE>
37             <DATA-ELEMENTS>
38               <DATA-ELEMENT-PROTOTYPE>
39                 <SHORT-NAME>value</SHORT-NAME>
40                 <TYPE-TREF DEST="BOOLEAN-TYPE">/Binker/dt_bulb</TYPE-TREF>
41                 <IS-QUEUED>false</IS-QUEUED>
42               </DATA-ELEMENT-PROTOTYPE>
43             </DATA-ELEMENTS>
44           </SENDER-RECEIVER-INTERFACE>
45         </AR-PACKAGE>
46       </TOP-LEVEL-PACKAGES>
47 </AUTOSAR>
```

Remember the flow process as a part of the methodology in figure 2. SystemDesk is able to load in SWCs and system descriptions given in an AUTOSAR XML. Furthermore, DBC and LDT files can be imported for the network architecture. Finally external code files are used and integrated into the generated RTE to run the AUTOSAR system.

3 Summary

Looking back to all steps in modeling and developing an AUTOSAR conform system shows the complexity and the effort a developer must deal with. The benefits of simulating the system in an early development phase to identify errors and requirements violations or to test different systems alternatives increase the productivity and the quality of the end product. SystemDesk tries to help the developer at each step of the modeling process. Providing at the one side different integrated functionalities like automatic RTE generation and a standardized exchange concept using the AUTOSAR XML format at the other side, it helps a lot of avoiding interoperability problems and coding errors.

All in all you can say the main focus of SystemDesk is to provide a tool building up a software architecture of an AUTOSAR conform system. Of course you can also model hardware and network issues, but there you can only use existing types of ECUs and busses. Therefore, looking at the AUTOSAR layered framework in picture 1, SystemDesk concentrates its efforts to the software layer. It would be very easy developing applications with other tools and integrate this code in a whole AUTOSAR systems to find weaknesses and problems in a simulation for example. If the focus of the developer is to create new kinds of ECUs or communication mechanism like busses then SystemDesk should not be the tool to choose.

Looking at the whole methodology of AUTOSAR with all the steps and tools you need to get a proper architecture and a correct system, SystemDesk manages it to get a seamless integration into this process. Some future work could be building up a complex real world example with different software architectures, hardware topologies and network alternatives, generate different system combinations out of them and investigate the interaction and behavior of the resulting real time system. It would be very interesting to investigate the benefits of using such modeling techniques provided by SystemDesk instead of complicated and incomprehensible solutions of today's development processes.

4 Literature

References

1. Schlegel: Technical Foundations for the Development of Automotive Embedded Systems. University Potsdam, HPI, 2009.
2. N. Naumann: Runtime Environment & Virtual Function Bus. University Potsdam, HPI, 2009.
3. J. Gosda: AUTOSAR Communication Stack. University Potsdam, HPI, 2009.
4. R. Hebig: AUTOSAR - Methodology and Template Based System Design. University Potsdam, HPI, 2009.
5. A. Krasnogolowy: Simulation of Automotive Systems in the Context of AUTOSAR. University Potsdam, HPI, 2009.
6. The official AUTOSAR website: http://www.autosar.org/
7. AUTOSAR Development Partnership: AUTOSAR technical Overview 2.2.2, 2008.
8. AUTOSAR Development Partnership: Specification of the Virtual Functional Bus 1.0.2, 2008.
9. Vector: Steuergeräte optimal kalibrieren. www.vector-worldwide.com
10. SystemDesk Guide Release 6.2. July 2008.
11. dSpace website: http://www.dspace.de
12. AUTOSAR XML schema (autosar.xsd)