Model-Based Extension of AUTOSAR for Architectural Online Reconfiguration

Basil Becker[1], Holger Giese[1], Stefan Neumann[1],
Martin Schenck[2] and Arian Treffer[2]

Hasso-Plattner-Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
[1]forename.surname@hpi.uni-potsdam.de
[2]forename.surname@student.hpi.uni-potsdam.de

Abstract. In the last years innovations in the automotive domain have more and more been realized by software leading to a dramatically increased complexity of such systems. Additionally automotive systems have to be flexible and robust, e.g., to be able to deal with failures of sensors, actuators or other constituents of an automotive system. One possibility to achieve robustness and flexibility in automotive systems is the usage of reconfiguration capabilities. However, adding such capabilities introduces even higher degree of complexity. To avoid this drawback we propose to integrate reconfiguration capabilities into AUTOSAR, an existing framework supporting the management of such complex system at the architectural level. Elaborated and expensive tools and toolchains assist during the development of automotive systems. Hence we present how our reconfiguration solution has been seamlessly integrated into such a toolchain.

1 Introduction

Today most innovations in the automotive domain are realized by software. This results in a dramatically increasing complexity of the developed software systems[1]. The objective of the AUTOSAR framework is to deal with this complexity at the architectural level. Additionally these systems need to deal with diverse situations concerning the context in which the software is operating. Such systems and especially the software, which is realizing essential functionalities of the overall system, need to be flexible to react on changes of its context. Regardless if such a system need to react on failures or on other contextual situations[2], flexibility and robustness plays an important role in today's automotive applications.

[1] The complexity concerning the size of the developed software, the functionality realized by the software system and so on.
[2] An example for such a situation, which is not related to a failure is in case the car is connected to diagnostic devices.

Reconfiguration is one possibility to facilitate the flexibility and robustness of such systems. There exist different possibilities to realize reconfiguration within automotive software. One is to realize reconfiguration mechanisms at the functional level. Because the AUTOSAR framework primarily provides mechanisms to deal with the complexity at the architectural level also the reconfiguration aspects should be available at the same level. Because deriving architectural information from the functional level could be difficult or even impossible we propose to specify reconfiguration aspects at the architectural level and to automatically derive the needed functionality based on the architectural information.

Further in a typical development scenario one has to deal with black-box components provided by third parties and elaborated information about the included functionality is not available, what also hampers the management of reconfiguration aspects at the functional level. Another possible solution is to introduce a new approach inherently facilitating reconfiguration aspects in the context of automotive systems. Today standard methods and tools already exist for supporting the development process of AUTOSAR. Because adapting existing tools or developing new once is very costly the propagation of such a new approach would be hardly suitable in practice. Summarizing we have identified the need for an development approach that is able to provide reconfiguration capabilities at the architectural level, can be seamlessly integrated into an exiting development solution and can also include third party components into the reconfigurable architecture. In this work we show how reconfiguration capabilities, which are currently not included in the existing AUTOSAR approach can be supported at the architectural level without degrading existing development solutions, tools or the standard itself. We further show how the needed functionality for realizing the reconfiguration logic can be automatically generated based on the architectural information describing the reconfiguration. The used application example for our evaluation is related to the field of fault tolerant systems and from our perspective such systems are one possible field to which reconfiguration like discussed in the remainder of this work can be applied.

The remainder of this paper is organized as follows. In Section 2 we discuss existing approaches supporting reconfiguration relevant for automotive systems and especially those approaches providing reconfiguration capabilities at the architectural level. In Section 3 we briefly introduce the existing toolchain, which builds the technological foundation for our investigation concerning the developed extension for on-line reconfiguration within the AUTOSAR framework. Subsequently in Section 4 we show how such a system is usually modeled with the given tools and how the additional reconfiguration aspects could be formulated based on the input/output of the existing toolchain. In Section 5 we show how these created additional reconfiguration aspects are automatically merged back into the original architecture and how the merged result fits into the existing tools without discarding or degrading parts of the original toolchain. Finally we give short discussion concerning the current results of our work in Section 6.

2 Related Work

In several different areas of computer science ideas have been presented, which are related to the approach we are going to present in this paper. In the field of software product lines and especially dynamic software product lines the topic of variable software has been addressed. The software architecture community has presented some work on the reconfigurability of robotic systems. Work, tailored to the automotive domain, has been done in the DySCAS project. We did some research on self-optimizing mechatronic systems.

In previous work we have presented a modeling technique called Mechatronic UML (mUML), which is suitable for the modeling of reconfigurable and self-optimizing mechatronic systems [1, 2]. However, the mUML approach differs from the one, which will be presented in this paper, in the fact that mUML uses an own code generation mechanisms and thus could hardly be integrated into existing development tool chains.

In the DySCAS[3] project dynamically self-configuring automotive systems have been studied [3, 4]. DySCAS does not provide a model based development approach, tailored to the specification of reconfiguration. Reconfiguration is specified with policy scripts, which are then evaluated by an engine at run-time (cf. [5]).

Software Product Line Engineering (SPLE) [6] aims at bringing the assembly line paradigm to software engineering. Typically a software product line is used to develop multiple variants of the same product. However, as the classical SPLE approach targets the design-time variability of software it is not comparable to the approach we are going to present in this paper. Recently a new research branch has emerged from SPLE called Dynamic Software Product Line Engineering [7]. In Dynamic Software Product Lines the decision, which variant to run, has moved from design- to run-time. Such an approach is presented in [8], where the authors describe a dynamic software product line, which is suitable for the reconfiguration of embedded systems. In contrast to our approach this one is restricted to the reconfiguration of pipe-and-filter architectures and the reconfiguration has to be given in a textual form.

In [9] a framework for the development of a reconfigurable robotic system has been presented. But the presented approach does in contrast to ours not support the model-driven development of reconfiguration. A policy-based reconfiguration mechanism is described in [10]. The authors present a powerful and expressive modeling notation for the specification of self-adaptive (i.e. reconfigurable) systems but their approach requires too much computational power and is thus only remotely applicable to embedded systems. In [11] an approach based on mode automata has been presented. However, mode automata only support switching between different behaviors internal to a component and do not cover architectural reconfiguration.

[3] http://www.dyscas.org

3 Existing Development Approach

For the development of embedded systems – especially in the automotive domain – several tools exist that provide capabilities for model-based development of such systems. Tools used by companies typically are mature, provide reliable and optimized code generation mechanisms and are as expensive as complex. Hence, any technique that claims being usable in the domain of embedded / automotive systems must be integrated into the existing toolchain. We will use this section to exemplary describe a toolchain, which might be used in the context of the AUTOSAR domain specific language.

3.1 AUTOSAR

The AUtomotive Open System ARchitecture (AUTOSAR) is a framework for the development of complex electronic automotive systems. AUTOSAR provides a layered software architecture consisting of the Application layer, the Runtime Environment and the Basic Software layer. Figure 1[4] shows the different layer of the architecture. The Basic Software layer provides services concerning HW access, communication and Operating System (OS) functionality (cf. [12]). The Basic Software provides several interfaces in a standardized form to allow the interaction between the Basic Software layer and the application layer routed through the Runtime Environment. The Runtime Environment handles the communication between different constituents of the application layer and between the application layer and the Basic Software layer (e.g., for accessing Hardware via the Basic Software, cf. [13]). The Application layer consists of Software Components, which can be hierarchically structured and composed to so called Compositions. Software Components and Compositions can have ports and these ports can be connected via Connectors (see [14] for more details). The real communication is realized through the Runtime Environment in case of local communication between Software Components (Compositions) on the same node (Electronic Control Unit) or through the Runtime Environment in combination with the Basic Software in case of communication between different nodes.

The main focus of AUTOSAR is the modeling of architectural aspects and of structural aspects. The behavior modeling (e.g., needed control functionality for reading sensor values and setting actuators) is not the main focus of the AUTOSAR framework. For modeling such behavior existing approaches and tools can be integrated into the development process of AUTOSAR. One commonly used tool for the model based development of behavior is MATLAB/Simulink (like described in Section 3.2). For executing such functionality AUTOSAR provides the concept of Runnables, which are added as a part of the internal behavior of a Software Component. Developed functionality could be mapped to Runnables and these Runnables are mapped to OS tasks. Additionally events

[4] Picture taken from http://www.autosar.org/gfx/media_pictures/AUTOSAR-components-and-inte.jpg.

can be used to decide inside an OS task if specific runnables are executed at run-time (e.g., runnables could be triggered by events if new data has been received via a port of the surrounding Software Component). For more details about the OS provided by the AUTOSAR framework see [15].

Once the modeling and configuration is done, in the current release version of AUTOSAR[5] changes at run-time concerning the structure of the application layer (e.g., restructuring connectors) are not facilitated by the framework.

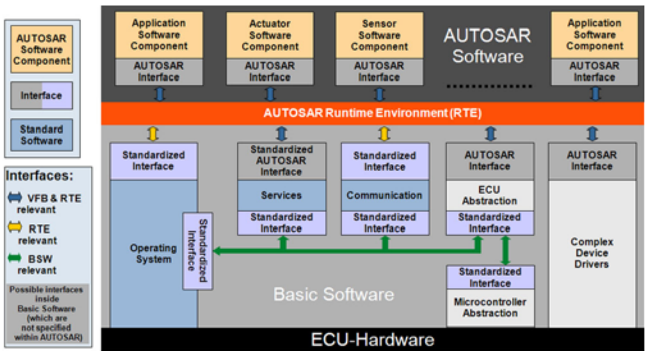


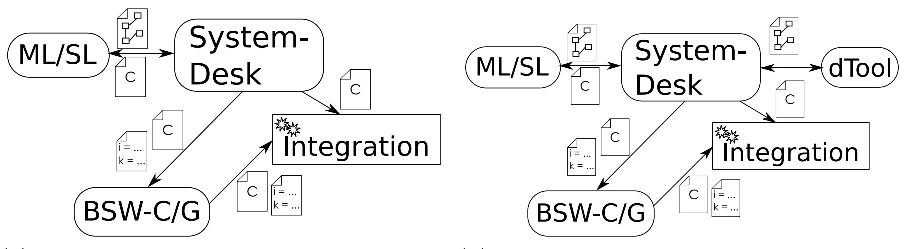
Fig. 1. The AUTOSAR layered architecture

3.2 Existing Toolchain

The scheme in Figure 2(a) shows one possible toolchain for the development of AUTOSAR systems. Rectangles with rounded corners represent programs, rectangles with cogwheels stand for processes. The arrows indicate exchange of documents, the type of the document (i.e. models, C-code or parameters) is annotated to the arrows. The system's architecture (i.e. components, ports and connectors) is modeled in SystemDesk[6]. Together with the architecture SystemDesk also supports the modeling of the system's deployment to several ECUs. The components behavior is specified using Matlab with the extension Simulink. For Matlab/Simulink (ML/SL) special AUTOSAR block sets exist, which allow the import of components specified in SystemDesk into Matlab and following the development of the component's functionality.

Further SystemDesk supports the generation of optimized C-Code, which conforms to the AUTOSAR standard concerning the Runtime Environment (cf. Subsection 3.1). Together with the C implementation of the software components modeled in SystemDesk the generated output also contains a configuration for the basic software layer. This layer is generated from specialized tools (e.g. Tresos

[5] Release 3.1
[6] http://www.dspace.de



(a) Exemplary toolchain for development with AUTOSAR (b) Tool chain for modeling reconfigurable AUTOSAR architectures

Fig. 2. The current and the extended toolchain for the development with AUTOSAR

by ElectroBit, abbreviated as BSW-C/G in Figure 2) and is specific to the system modeled in SystemDesk and the available hardware.

At the integration step a build environment compiles the generated C-Code and builds the software running on each ECU.

3.3 Evaluation Example

The used application example for showing the reconfiguration capabilities that are supplemented to the existing AUTOSAR framework in our approach is the reconfiguration of a set of adjacent aligned distance sensors. The discussed evaluation example allows reacting on sensor failures in the manner that the failure of individual sensor instances is compensated.[7]

Such adjacent aligned sensors are commonly used in a modern car, e.g., in case of a parking distance control. Such a parking distance control uses sensors (e.g., ultrasonic sensors) embedded in the front or rear bumper for measuring the distance to nearby obstacles.

Additionally in Section 5.3 we discuss the evaluation results of experiments we have made on an evaluation platform using the techniques described in Section 4.

4 Modeling Reconfiguration

In order to make an AUTOSAR system architecture reconfigurable, some additional concepts are needed. The toolchain needs to be extended in a certain way that extensions do not make the existing toolchain invalid. From our perspective the best way is to integrate an optional tool that can be plugged into the existing toolchain.

[7] For our application example we assume that a sensor failure can be observed at the level of Software Components.

4.1 Extended Toolchain

Our modeling approach is currently restricted to the modeling of AUTOSAR software architectures. The toolchain in Figure 2(b) shows our approach of extending the existing toolchain by another tool without degrading existing ones. By using this proposal the developer is free to choose, whether he wants to use our given enhancement or not. He can either model an architecture, that does not provide any reconfiguration or he can use our tool in addition and empower himself to specify and realize reconfiguration aspects. The advantages are obvious: better control and overview due to the diagrammatic depiction.

SystemDesk SystemDesk is a tool provided by *dSPACE*[8] supporting the modeling of AUTOSAR conform systems. Among other things it supports the modeling of the AUTOSAR HW and SW architectures. For modeling the SW architecture Software Components, compositions as well as ports, interfaces and connectors are provided as modeling artifacts. These artifacts can be used to describe the architectural aspects of a concrete SW architecture for a specific system like shown in Figure 3.[9] Besides modeling the architecture in SystemDesk, the tool also allows the linking of the Software Components to their behavior, written in C-Code or given in form of MATLAB/Simulink models.

Additionally the HW architecture including the used types of ECUs (Electronic Control Unit), the deployment of Software Components to these ECUs as well as additional information concerning the configuration (e.g., configuration concerning communication and the OS) can be specified. Based on this information SystemDesk automatically generates code, which can be compiled for the specified platform. Besides the code for the application layer SystemDesk also generates source code realizing the Runtime Environment functionality.

Figure 3 shows the relevant part of the SW architecture concerning our application example modeled in SystemDesk. Like depicted on the right side of Figure 3 the composition consists of four Software Components representing the distance sensors[10] connected to another composition *SensorLogic* evaluating the sensor values to a single value provided by the port *ShowDistanceOut*.[11]

The above mentioned elements (Software Components, ports and connectors) are used to describe the software when no reconfiguration is intended. Some additional elements shown in Figure 3 are described in more detail in the following section. These elements (*Interpolation*, *Reconfiguration* and the unused ports of the sensors) are used later to realize the reconfiguration functionality.

[8] www.dspace.de

[9] For the realization of control functionality other constituents can be imported into SystemDesk, e.g. in form of C-Code or Matlab/Simulink models, to realize the implementation of internal behavior of Software Components.

[10] The ports accessing the HW via the Runtime Environment and Basic Software are not shown here because they are not object of reconfiguration.

[11] To allow a better understanding *SensorLogic* calculates a single output value based on the different input values. Potentially also several output values can be computed.

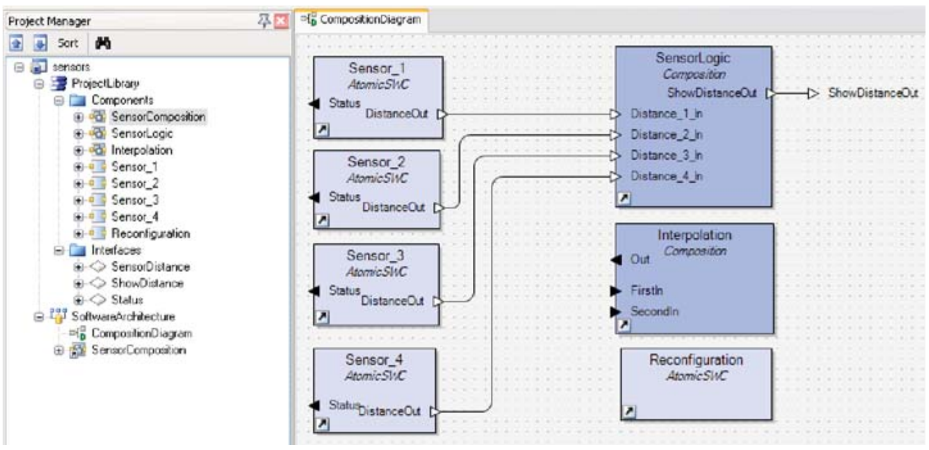


Fig. 3. Configuration in SystemDesk

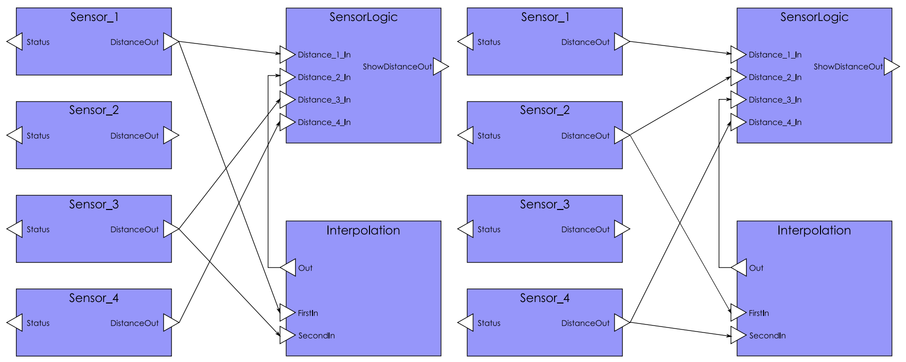
dTool The usual modeling procedure is not altered until the modeling in SystemDesk[12] is initially done like described above. After the model from SystemDesk is exported in form of an XML file[13] and loaded into the *dTool* the constituents concerning the reconfiguration could be specified. Using the dTool we are now able to model two different aspects, relevant for the reconfiguration. On the one hand our tool allows creating new configurations, which differ from the initial one. Such differences are alternative connections (in form of connectors) between components and/or compositions. Which parts of the architecture are relevant concerning reconfiguration is indicated by the Software Component *Reconfiguration* included in the original SystemDesk model. Alternatively the dTool allows to manually choosing relevant parts of the imported architecture. On the other hand our dTool allows to model an automaton, which specifies how to switch between the modeled configurations.

Figure 4(a) depicts the configuration (modeled in the dTool) associated with the state that sensor two is broken. In the shown configuration the value of the port *DistanceOut* from the broken sensor *Sensor_2* is not available. Consequently the value sent to the port *Distance_2_In* of the composition *SensorLogic* is interpolated from the to sensor values of the first and the third sensor via the additional composition *Interpolation*.

Figure 4(b) shows the configuration associated with the state that sensor four is broken and the value sent to the port *Distance_3_In* of the composition *SensorLogic* is interpolated based on the sensor values of the second and the fourth sensor.

[12] http://www.dspace.de/ww/en/ltd/home/products/sw/system_architecture _software/systemdesk.cfm

[13] The AUTOSAR framework specifies XML-Schemes for exchanging AUTOSAR models in a standardized form.



(a) Configuration in case Sensor_2 is broken

(b) Configuration in case Sensor_3 is broken

Fig. 4. Two configurations of the architecture for two different scenarios

The composition *Interpolation* used here provides some functionality for interpolating two different sensor values. This functionality has been added specifically for our application example.[14] This interpolation functionality is used to approximate the value of a broken sensor based on the values of two adjacent sensors. It is potentially possible to integrate this functionality into an existing Software Component, but for a better understanding, we decided to introduce a new Software Component for this purpose.

The second part, which could be modeled in the dTool relevant for the reconfiguration is the automaton shown in Figure 5 specifying how to switch between different configurations. The automaton consist of the initial state *initial*, where all four sensors work correctly, the state *sensor2broke* where the second sensor is broken, the state *sensor3broke* where the third sensor is broken and state *allfail* where the first or the fourth sensor or more than one sensor is broken. Transitions between these states specify which reconfiguration is applied at runtime. The transitions are further augmented with guards. These guards are expressions over the values provided by components within the reconfigurable composition, which provide information relevant for the reconfiguration (in our case these information are provided via the *Status*-ports of the four Sensor-Software Components). An example for such a guard is shown at the transition from state *initial* to state *sensor2broke* requiring that the status port of the Software Component *Sensor_2* provides the value 0 (indicating a broken sensor).

For the application example we assume that such status ports of the Software Components representing the sensors exist as we otherwise were not able to observe each sensors' status.[15]

[14] In our application example this functionality has been realized using Matlab/Simulink.

[15] Alternatively an observer could be realized in form of an additional Software Component evaluating the sensor values over time and providing the status ports. If the

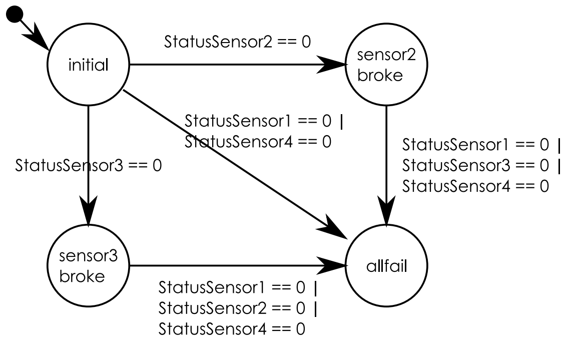

initial
StatusSensor2 == 0
sensor2 broke
StatusSensor1 == 0 | StatusSensor4 == 0
StatusSensor3 == 0
StatusSensor1 == 0 | StatusSensor3 == 0 | StatusSensor4 == 0
sensor3 broke
allfail
StatusSensor1 == 0 | StatusSensor2 == 0 | StatusSensor4 == 0

Fig. 5. Reconfiguration automaton in the dTool

5 Merge

In its current version the AUTOSAR standard does not support reconfiguration as a first class modeling element. Thus, SystemDesk also does not support modeling of diagrams that represent different variations of one composition. Hence the direct import of the reconfiguration, we have modeled in the *dTool*, is impossible. Nevertheless we want to make use of SystemDesk's elaborated and AUTOSAR standard conform code generation capabilities. We had to find a way to translate the reconfiguration behavior into a SystemDesk/AUTOSAR model. This is done by merging all configurations to one final model. In the final model, the reconfiguration logic will be encapsulated by two components, the RoutingComponent and the StateManager.

5.1 Merging configurations

Our modeling approach only allows the reconfiguration of connections between components but is not suitable for the addition and removal of components at run-time[16]. Hence, a merged configuration consists of all components, which have been modeled in SystemDesk at the early stages (cf. Subsection 4.1). Connections, which do not exist in all configurations, are redirected via a special component, called RoutingComponent. Therefore, the first step is to build the intersection of all configurations. Connections found here are directly inserted into the merged model. Next the RoutingComponent is added.

Generating the RoutingComponent The RoutingComponent intersects every connection, which is not invariant to the reconfigurable composition. Follow-

measured values of consecutive points in time repeatedly have improper values (too big differences) a malfunction can be deduced.

[16] Please note that the dTool allows to modeling configurations, which do not contain all components. The semantic is that the components are hidden, a dynamic loading of components is not supported by AUTOSAR.

ing the RoutingComponent has to know at each point in time, which configuration is currently the active one. Which configuration is active, is determined by the evaluation of the current configuration and the valuation of the variables used in the guards of the reconfiguration automaton (cf. Figure 5). As an evaluation of the automaton at each point in time a value is sent to the RoutingComponent, is much too expensive we have implemented a different strategy.

```
38 switch (configuration_0) {
39   // Routing for configuration initial:
40   case 0:
41     Rte_IWrite_Distance_2_In_0_1_sndr_Distance(distance_1);
42     break;
43   // Routing for configuration allfail:
44   case 3:
45     break;
46   // Routing for configuration sensor2broke:
47   case 1:
48     break;
49   // Routing for configuration sensor3broke:
50   case 2:
51     Rte_IWrite_Distance_2_In_0_1_sndr_Distance(distance_1);
52     Rte_IWrite_FirstIn_1_0_sndr_Distance(distance_1);
53     break;
54 }
```

Listing 1. Excerpt of the RoutingComponent's code

The configurations modeled in the dTool get a unique number each. The RoutingComponent receives the number of the currently active configuration via a special input port. Using this information the RoutingComponent can be implemented as a sequence of switch statements. The computation of the current active configuration is done in a second component – the StateManager. The dTool automatically generates a runnable for the RoutingComponent containing the described behavior. An excerpt of the RoutingComponent's implementation is shown in Listing 1. The variables configuration_0 and distance_1 hold the values of the current configuration and the second sensor's distance respectively. The excerpt is responsible for routing the value provided by the second distance sensor. In configuration allfail (cf. line 44) and sensor2broke (cf. line 47) no routing takes place. In the initial configuration the sensor's distance value is simply forwarded (cf. line 41) and in case the third distance sensor broke down, the value is forwarded as in initial (cf. line 51) but it is also sent to the Interpolation component (cf. line 52).

StateManager The StateManager – as briefly mentioned above – is responsible for the computation of the currently active configuration. Therefore, it has to be connected with all ports that provide values, which are used in the guards of reconfiguration automaton. Each time the StateManager receives an update on its ports, it has to evaluate the automaton again and change the value of the currently active configuration accordingly.

```
49 configuration_0 = Rte_IrvRead_configuration();
50 switch (configuration_0) {
51   // State change logic for configuration
52   //       initial
53   case 0:
54     // Transition to CGConfiguration#sensor2broke(id: 1,
         name: sensor2broke)
55     if (StatusSensor2_10 == 0) {
56       configuration_0 = 1;
57       Rte_IrvWrite_configuration(configuration_0);
58       Rte_IWrite_conf_out_configuration(configuration_0);
59     }
60   break;
```

Listing 2. Excerpt from the StateManager's implementation

Updates to the StateManager's ports are signaled by events, which then trigger the StateManager's evaluation function.[17] A small part of this evaluation function is shown in Listing 2. At line 49 of the listing the currently active configuration is read, which then is used as input for the switch statement in the following line. In case the second distance sensor is broken (identified by StatusSensor2_10 equals zero) the configuration is changed (cf. line 56). Then the changed configuration is written to the StateManager's internal configuration variable (cf. line 57) and provided to other components through the conf_out port (cf. line 58).[18]

5.2 Final SystemDesk project

Figure 6 shows the Sensor-Composition after exporting the merged model to SystemDesk again. The components for the distance sensors are all connected to the RoutingComponent, which is named *Reconf* in this diagram. The system modeled in our application example does not allow an interpolation for the sensor components one and four. Following these components are always directly connected with the SensorLogic component and are not handled by the RoutingComponent. Nevertheless they also have to be connected to the RoutingComponent as the sensor values are used to interpolate the second respective third sensor in case of a failure.

The StateManager is depicted below the RoutingComponent and is connected to the RoutingComponent through the *Conf* ports, which provide information about the currently active configuration. As defined in the reconfiguration automaton (cf. Figure 5) the decision which configuration to use, depends on the values of the sensor components' status ports. Following the StateManager is connected to those ports. As the reconfiguration automaton does not

[17] Event mechanisms in form Runtime Environment events provided by the AUTOSAR framework have been used to trigger the runnable realizing the functionality of the StateManager. More information about Runtime Environment events can be found in [13].

[18] E.g., provided to the RoutingComponent.

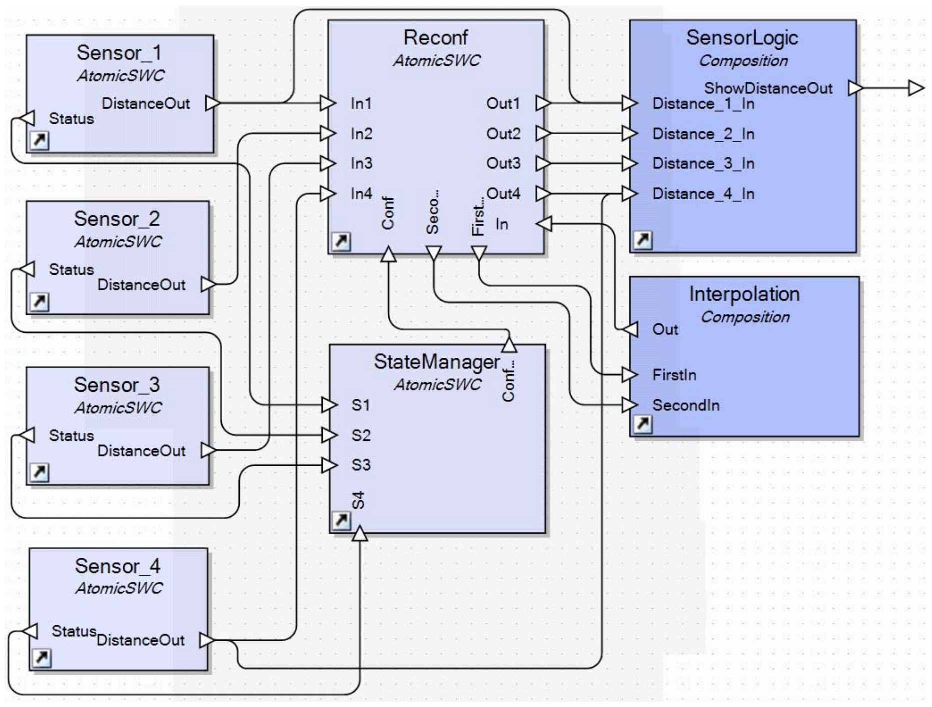


Fig. 6. Resulting merged SW Architecture in SystemDesk

rely on any values provided by the Interpolation or SensorLogic component the StateManager is not connected with them.

5.3 Evaluation Results

The above described approach for the modeling and realization of reconfiguration aspects has been evaluated within a project arranged at Hasso-Plattner-Institute in collaboration with the dSPACE GmbH.

As an evaluation platform for the shown approach the Robotino robot[19] has been used, which provides an open platform for running C/C++ programs (among others) on a Real-Time Operating System (RTOS). The RTOS is provided in form of RTAI[20], which is a real-time extension for the Linux operating system. To be able to evaluate the developed concepts on this platform an execution environment has been realized based on the existing RTAI Linux, which allows to compile and execute the outcome of the above described extended toolchain including the resulting parts of the reconfiguration functionality.

The robot provides nine distance sensors uniformly distributed around its chassis. In the context of our evaluation experiments we modeled the reconfig-

[19] http://www.festo-didactic.com/int-en/news/learning-with-robots.htm
[20] For more details see https://www.rtai.org.

uration of distance sensors accordingly to the above used evaluation example using nine instead of four sensors.[21]

The generated source code of the different tools has been compiled and executed on the platform to show the applicability of our approach. In addition we analyzed the overhead resulting from the reconfiguration functionality added by our approach in comparison to the original functionality without any reconfiguration capabilities. For this purpose we measured the execution time of the generated reconfiguration automaton included in the added StateManager in combination with the parts resulting from the routing functionality realized in the additional RoutingComponent (both components are shown in Figure 6).

In case of the nine sensors provided by the robot we measured execution times of the relevant parts concerning the reconfiguration functionality between 20 and 100 microseconds depending on the type of reconfiguration (react on the defect of one or several sensors at the same point in time). The tests have been realized on the equivalent execution platform on which the real functionality has been executed when running the application example on the robot.[22] While the robot provides a more powerful processor like it is the case for the most Electronic-Control-Units (ECUs) used within a modern car, even by using a platform or processor, which has only a tenth of the computation power we will not reach an overhead concerning the reconfiguration leading to an execution time much greater than one millisecond.

6 Conclusion

In this paper we have presented an approach to extend AUTOSAR architectures with reconfiguration capabilities. The approach fits into existing toolchains for the development of AUTOSAR systems and allows reusing tools, which where currently used. The overhead added to the resulting reconfigurable architecture has been shown to be minimal but the developer rewards an easier development of reconfiguration logic, which otherwise has to be done manually at the functional / implementation level. We have successfully shown that it is possible to use high-level architectural modeling techniques without generating massive run-time overhead.

Although our approach has only been evaluated in the context of AUTOSAR it should be applicable to almost any component based development approach.

For the future we plan to also support the reconfiguration of distributed compositions. From an architectural point of view a distributed composition does not differ from a local one, as AUTOSAR completely hides the communication details in the Runtime Environment-layer from perspective of the application layer. Anyway, a distributed scenario contains enough challenges such as timing delays, Basic Software configuration, deployment decisions concerning Routing-Components, just to name a few. Further the high-level architectural modeling

[21] For a better understanding we decided to only show four sensors in the previous sections.

[22] The robot is equipped with 300 MHz processor.

we have introduced in this paper also allows the verification of the modeled systems. First attempts in these directions have been very promising and we are looking forward to look into the details.

Acknowledgment We thank the dSPACE GmbH and especially Dirk Stichling and Petra Nawratil for their support in setting up the project. We want to thank the participants of the student project “A run-time environment for reconfigurable automotive software” : Christian Lück, Johannnes Dyck, Matthias Richly, Nico Rehwaldt, Thomas Beyhl, Thomas Schulz and Robert Gurol.

References

1. Burmester, S., Giese, H., Münch, E., Oberschelp, O., Klein, F., Scheideler, P.: Tool Support for the Design of Self-Optimizing Mechatronic Multi-Agent Systems. International Journal on Software Tools for Technology Transfer **10**(3) (2008) 207–222
2. Giese, H., Burmester, S., Schäfer, W., Oberschelp, O.: Modular design and verification of component-based mechatronic systems with online-reconfiguration. In: Proc. SIGSOFT ’04/FSE-12, New York, NY, USA, ACM Press (2004) 179–188
3. Feng, L., Chen, D., Törngren, M.: Self configuration of dependent tasks for dynamically reconfigurable automotive embedded systems. In: Proc. of 47th IEEE Conference on Decision and Control. (2008) 3737–3742
4. Anthony, R., Ekeling, C.: Policy-driven self-management for an automotive middleware. In: HotAC II: Hot Topics in Autonomic Computing on Hot Topics in Autonomic Computing, Berkeley, CA, USA, USENIX Association (2007)
5. DySCAS Project: Guidelines and Examples on Algorithm and Policy Design in the DySCAS Middleware System, Deliverable D2.3 Part III. (February 2009) Available online: http://www.dyscas.org/doc/DySCAS_D2.3_part_III.pdf.
6. Pohl, K., Böckl, G., van der Linden, F.: Software Product Line Engineering. Foundations, Principles, and Techniques. Springer, Berlin Heidelberg New York (2005)
7. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic Software Product Lines. Computer **41**(4) (2008) 93–95
8. Kim, M., Jeong, J., Park, S.: From product lines to self-managed systems: an architecture-based runtime reconfiguration framework. In: DEAS ””05: Proc. of the 2005 workshop on Design and evolution of autonomic application software, New York, NY, USA, ACM (2005) 1–7
9. Kim, D., Park, S., Jin, Y., Chang, H., Park, Y.S., Ko, I.Y., Lee, K., Lee, J., Park, Y.C., Lee, S.: SHAGE: a framework for self-managed robot software. In: Proc. SEAMS ’06, Shanghai, China, ACM (2006) 79–85
10. Georgas, J.C., Taylor, R.N.: Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In: Proc. SEAMS ’08, New York, NY, USA, ACM (2008) 105–112
11. Talpin, J.P., Brunette, C., Gautier, T., Gamatié, A.: Polychronous mode automata. In: EMSOFT ’06: Proc. of the 6th ACM & IEEE International conference on Embedded software, New York, NY, USA, ACM (2006) 83–92
12. AUTOSAR GbR: List of Basic Software Modules. Version 1.3.0.
13. AUTOSAR GbR: Specification of RTE. Version 2.1.0.
14. AUTOSAR GbR: Specification of the Virtual Functional Bus. (2008) Version 1.0.2.
15. AUTOSAR GbR: Specification of Operating System. (2009) Version 3.1.1.