A Component Model for the AUTOSAR Virtual Function Bus

Dietmar Schreiner[1,2] and Karl M. Göschka[1]

[1] Vienna University of Technology
Institute of Information Systems, Distributed Systems Group
Argentinierstrasse 8 / 184-1, A-1040 Vienna, Austria
{d.schreiner, k.goeschka}@infosys.tuwien.ac.at

[2] University of Applied Sciences Technikum Vienna
Department of Embedded Systems
Höchstädtplatz 5, A-1200 Vienna, Austria

Abstract

To reduce cost and time to market of automotive software systems and simultaneously increase the products' quality, the component paradigm has achieved broad acceptance within the automotive industry over the last few years. This fact is reflected by upcoming domain specific software standards like AUTOSAR. In AUTOSAR application concerns are covered by software components, while infrastructural ones are handled within layered component middleware. The so obtained separation of concerns leads to an increase in application quality, reusability and maintainability, and hence to a reduction of cost and time to market. This paper contributes with the consequent application of the component paradigm to AUTOSAR's layered middleware, thereby gaining all benefits of CBSE not only for the application level, but for the whole automotive software system. The introduced component model for middleware components can flexibly be used to build resource-aware, AUTOSAR compliant, component middleware for distributed automotive software systems and can seamlessly be integrated within the AUTOSAR system architecture.

1 Introduction and Related Work

Within the last decade, the fraction of electronic building blocks within automotive systems has steadily increased. State of the art vehicles contain more than 70 electronic control units (ECUs), typically connected by up to 10 diverse bus systems [8]. Driven by market demands these numbers are going to grow rapidly, boosting the complexity of the vehicles' electronic subsystems. On this account,

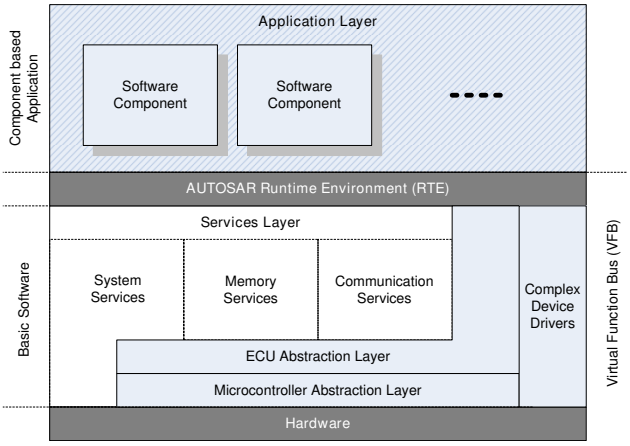

Figure 1: AUTOSAR System Architecture

software has become a key factor in cost and time to market for automotive systems. Thus it was self-evident that major companies strived for a standardization not only of software modules deployed within their vehicles but also for an appropriate software engineering process. In 2002 the AUTOSAR [9] consortium was jointly founded by automobile manufacturers, suppliers and tool developers, to create an open and standardized automotive software architecture including adequate development paradigms. The main foci of AUTOSAR are (i) to increase the quality of automotive software, its maintainability and its scalability, (ii) to support usage of commercial-off-the-shelf (COTS) components across product lines, and finally (iii) to optimize cost and time to market.

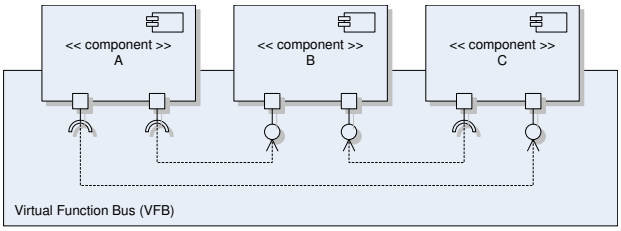

Figure 2: The Virtual Function Bus at Application Level [4]

To attain these goals, component based software engineering (CBSE) has been chosen as AUTOSAR's development paradigm. CBSE allows a clean separation of infrastructural and application concerns. All application specific implementation is assigned to AUTOSAR software components in the application layer, while any implementation related to infrastructure is located within the layered software modules below (and including) the AUTOSAR runtime environment (RTE). These software layers are referred to as *Basic Software* in AUTOSAR and among others resemble the typical functionality of component middleware.

An AUTOSAR software component [2] is a unit of execution, that is described by its ports. Ports are defined to be well defined points of interaction which provide or require interfaces, thus a port may be of type require (R-port) or of type provide (P-port). An AUTOSAR component may be of atomic or composed nature. However, it has to be atomic in terms of deployment, so it can be mapped to exactly one specific ECU.

Figure 1 provides an overview of the AUTOSAR system architecture described in [1]: Application components, the AUTOSAR software components, are part of the *application layer*. They interface with the AUTOSAR RTE, that not only provides all infrastructural services for the application components, but also handles issues of component instantiation. Beneath the RTE, system-, memory- and communication services are encapsulated within the *Services Layer*, hardware abstraction is provided by the *ECU Abstraction Layer* and finally direct hardware access is provided by the *Complex Device Driver* layer.

At application level none of these sub-RTE layers is visible. Instead, components are directly connected and interact via the so called *Virtual Function Bus* (VFB) (see Fig. 2). The VFB is a logic abstraction—the application components' view—of all layers (*Basic Software* and hardware) involved in component interaction and therefore represents the system's component middleware. Its interfaces are provided by the RTE at runtime. Application components are developed using the VFB and have to implement application specific concerns only, thus are smaller in size, less error prone and reusable in different deployment scenarios.

As one can clearly see, CBSE in AUTOSAR is only applied to the application layer while the whole *Basic Software* adheres to a layered software architecture. Nevertheless, these layers are highly configurable. According to the AUTOSAR methodology, the RTE is generated [3] under consideration of configuration specific properties, defined throughout the application's development process. By this means, layered component middleware is compiled for each ECU, depending on infrastructural requirements of the ECU's application layer.

Project COMPASS [5] aims at applying the component paradigm not only to the application level but to the component middleware itself, thereby gaining all advantages of CBSE for AUTOSAR applications and AUTOSAR *Basic Software*. It is obvious, that middleware components can not rely on component middleware, thus the actual AUTOSAR component model can not be applied at *Basic Software* level. In fact, a well suited component model has to cover the needs of "low-level" middleware components, but also has to serve AUTOSAR application components in a cooperative way.

In COMPASS we accomplish this goal on the one hand by augmenting the AUTOSAR component model, to be precise, by extending the semantics of explicit component connectors defined at application level, and on the other hand by providing a separate component model for middleware components, the COMPASS component model. In addition we had to define mappings and transformations from application level connectors within the augmented AUTOSAR component model, to composed middleware structures [19, 17] within the COMPASS component model. As a consequence, software developers are able to automatically synthesize custom-tailored component middleware for their AUTOSAR applications from application models and prefabricated middleware building blocks. To specify the COMPASS component model we had to

- identify the first class architectural entities, that are subject to composition: components and explicit connectors at application level, and communication primitives and assemblies at middleware level (see Section 2.2, [18, 17]).

- define a composition standard, that particularizes how components (application and middleware level) are composed and deployed (see Section 2.3), how constraints and non-functional requirements are considered during composition time [18], how component configuration is handled, and how composition information is stored.

- define an interaction standard that particularizes how components interact, how they invoke services, and how data is propagated at runtime (see Section 2.3).

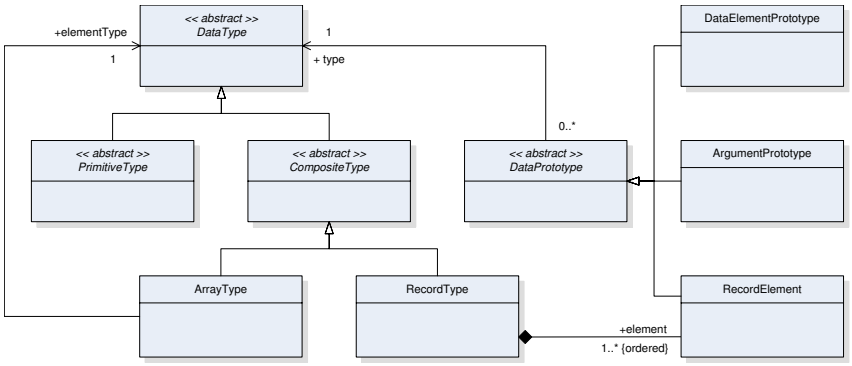

Figure 3: Meta-Model of COMPASS Data Types (simplified)

- provide a component framework for middleware components, that defines mechanisms of object handling, storage and instantiation, inter- and intra-node communication, operating-system functionality, and system specific resources (see Section 2.4).

2 COMPASS Component Model

A component model provides the semantic framework of what components are, how the are constructed, composed, deployed and used [22, 10], therefore it provides the foundation for any component based application.

Within the next sections we will specify the COMPASS component model by providing UML 2.0 [15] meta-models. These meta-models formally define the vocabulary, required to describe COMPASS model elements and their dependencies. When instantiating a meta-model element[1], the resulting element is a COMPASS model element. When instantiated in turn, this model element finally leads to a specific implementation. To give an example, a meta-model element *ComponentType* can be used to instantiate a model element of type *Component*. Finally, a COMPASS compliant component architecture may contain a specific instance (implementation) of this model element (e.g. the *ErrorLogger* component, implementing a centralized error logging). Abstract elements, (stereotyped $\ll$ *abstract* $\gg$), must not be instantiated and are typically used to represent all of the element's specializations.

2.1 Data Types

Data types are of great importance for describing component interaction. Therefore, we consider two levels of abstraction when defining data types (both can also be found in the AUTOSAR standard[2]): (i) the *structural level* and (ii) the *implementation level*. At the structural level, common interface definition languages typically specify their data types by combining predefined primitive data types to form various data structures, the user-defined types. At implementation level the mapping of data types and data structures to bits and bytes is of primary concern.

When it comes to heterogeneous component interaction, both levels of abstraction have to be taken into account. ECUs of distinct type might encode data in different ways, so data types that match at structural level could mismatch at implementation level. Therefore, COMPASS provides rules for data conversion, that are based on the data type meta-model provided in Figure 3. This model is equivalent to the one defined by the AUTOSAR standard [2]. In fact, COMPASS applies the component paradigm to AUTOSAR *Basic Software*, thus inventing a new—perhaps incompatible—data type model was deliberately avoided.

Within the depicted model, class *DataType* is an abstract generalization of all data types of the COMPASS component model. The abstract classes *PrimitiveType* and *CompositeType* are both derived from *DataType* whereas *PrimitiveType* is the abstract superclass for all primitive data types. These are not shown in Figure 3 for simplicity but are named here:

- ***IntegerType*:** represents all integer types within the COMPASS component model (e.g. int, word).
- ***FloatType*:** represents all floating point types within the COMPASS component model (e.g. double, float).
- ***BooleanType*:** represents all boolean types within the COMPASS component model (e.g. bool).

[1] Classes in meta-models represent building blocks of models. Therefore instantiating a class within a meta-model may again produce a class, but at a lower level (model level).

[2] The AUTOSAR standard defines three levels of abstraction: (i) the *Data Semantics Level*, (ii) the *Data Structure Level* and (iii) the *Data Implementation Level*.

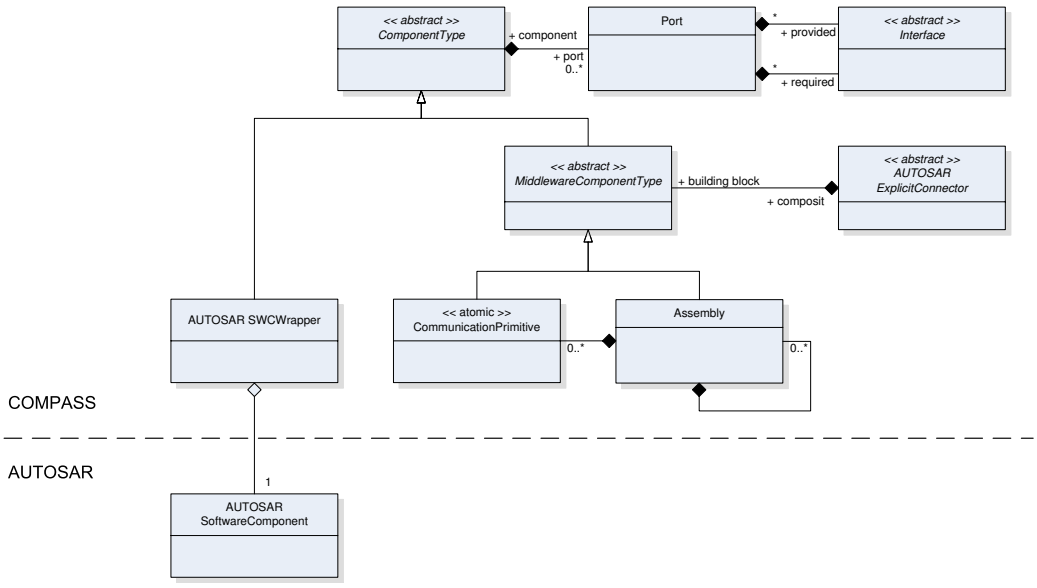

Figure 4: Meta-Model of COMPASS Components

- ***CharType*:** represents all character types within the COMPASS component model (e.g. char, TCHAR, unichar).
- ***StringType*:** represents all string types within the COMPASS component model (e.g. string, chararray).

All of them are concrete classes, thus can be instantiated to generate COMPASS data types. In addition, the COMPASS data type meta-model contains two classes for composite data types (represented by their abstract superclass *CompositeType*): *ArrayType* and *RecordType*. As depicted, an *ArrayType* is associated with exactly one *DataType* for all of its elements, while a *RecordType* contains an ordered set of *RecordElements*. *RecordElements* are concrete specializations of the *DataPrototype* class, that is associated with an arbitrary data type (via the generalized *DataType* class).

2.2 Model Elements

In contrary to component models like OMGs CORBA Component Model [16] or Microsofts COM [13], we consider not only components to be first class architectural entities, but also explicit connectors, as they represent interaction-specific attributes and implementations within a component architecture.

2.2.1 Components

In general, components are small, trusted units of execution [12, 21] that interact only by their well defined interfaces according to contractual guarantees [11] and strictly contain no other external dependencies. They can be independently deployed and composed without modification and, as a result, are well suited for reuse [14] and third-party composition.

Within this paper, application components adhere to the AUTOSAR component model, while middleware components comply to the COMPASS component model. As application components are connected to the VFB, they have to interact with the RTE. The RTE is built from middleware components, so COMPASS middleware components have to be aware of AUTOSAR components. For this reason COMPASS defines three concrete classes of components as shown in Figure 4. Two of them are true middleware components (both are derived from the abstract superclass *MiddlewareComponentType*). The third one is used to represent AUTOSAR-COMPASS interface adapters):

- ***Communication Primitives*:** The basic building blocks of COMPASS middleware are called *communication primitives* and are instantiated from the meta-model class *MiddlewareComponentType*. Note that the class *CommunicationPrimitive* is stereotyped $\ll atomic \gg$. Communication primitives are atomic in terms of deployment (connected communication primitives have to be deployed within the same address space) and in terms of execution (services of communication primitives must not be suspended by a scheduler). Since atomicity always has to hold under both aspects, one single stereotype can be used to describe them. Communication primitives are sim-

ilar to application components and mainly differ by their life-cycle. In contrary to application components, communication primitives do not exist within platform independent composition specifications at application level, but are created by model transformations during the middleware synthesis. We distinguish two types of communication primitives: (i) Passive Communication Primitives, that are executed within the thread of execution of the client, that is requesting their service. (ii) Active Communication Primitives, that own at least one thread of execution, thus have to provide at least one interface for activation by infrastructural services like operating system schedulers or interrupt service routines (ISRs). Communication primitives may also be stereotyped $\ll$ *singleton* $\gg$ indicating that the specific primitive must not exist more then once within a component architecture.

- ***Assemblies*:** The COMPASS component model is a hierarchical model. Composed structures may be treated like basic building blocks, if viewed as black box. These composed structures are called *assemblies* and are represented by the class *Assembly* in the COMPASS meta-model. An assembly may contain composed communication primitives and other assemblies in a recursive manner.

- ***AUTOSAR Software Component Adapters*:** As mentioned before, AUTOSAR software components have to be connected to their middleware. COMPASS components make up that middleware, so the third component class is used to join the two component models. At model level, application components are connected via explicit connectors. These connectors are transformed into composed middleware structures, when synthesizing the communication middleware. In addition, interface adapters have to be generated between the generic connector structures (middleware assemblies) and the application components. These adapters form the *Basic Software's* RTE and are of type *AUTOSAR SWCWrapper*.

All three classes are siblings of the abstract superclass *ComponentType*, that specifies the look-alike of COMPASS components: A COMPASS component may own an arbitrary number of ports. In contrary to the AUTOSAR standard, COMPASS ports may expose multiple interfaces of different types[3]. Ports are typically used to (i) model distinct states of a component's interfaces, to (ii) express semantic relations between them (e.g., call-back interfaces) or (iii) to provide a higher-level of abstraction for points of interaction. Interface states bound to ports play an important

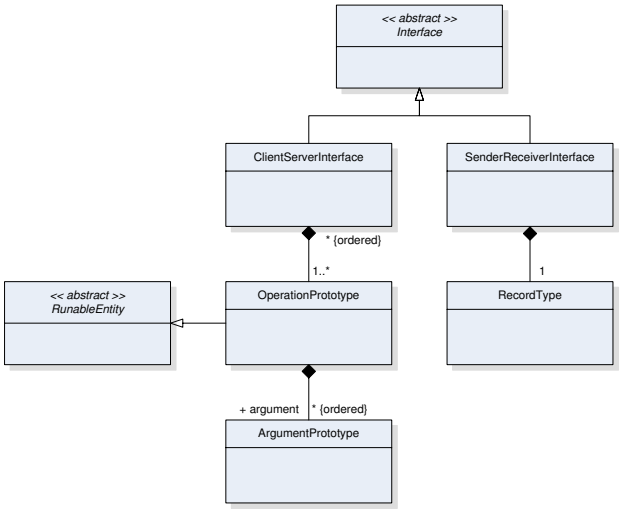

Figure 5: Meta-Model of Component-Interfaces

role in describing and verifying runtime behavior of composite structures [20], logic grouping is applied to increase a model's comprehensibility and higher-level abstraction is used within meta-models like the structural designs developed in [17].

An interface is a named set of services provided or required by a component. A component has to provide at least one interface, but may own multiple, distinct ones, so called facets. Interfaces specify the dependencies between the services provided by the component and the services required to fulfill this task. As shown in Figure 5, the COMPASS component model specifies two classes of interfaces: (i) the Client-Server Interface and (ii) the Sender-Receiver Interface[4]. The Sender-Receiver interface type utilizes the *RecordType* data type (see Figure 3), thus an instance of a Sender-Receiver interface is specified by the data structure sent or received. The Client-Server Interface type contains an ordered set of *OperationPrototypes*, that contain another ordered set of *ArgumentPrototypes*. According to this specification, a Client-Server Interface consists of an ordered set of operations, that are identified by their signature.

2.2.2 Connectors

At application level, we consider explicit connectors to be first class architectural entities. They represent middleware functionality and therefore are hot-spots of interaction. It is important to mention, that explicit connectors are model elements at application level, so they are not architectural entities in the COMPASS component model, but within an augmented AUTOSAR model. Nevertheless, ex-

[3] AUTOSAR component ports expose only one type of interface (required or provided) and therefore are classified as R-Ports or P-Ports

[4] This limitation was introduced in accordance to the AUTOSAR standard, that supports client-server and sender-receiver interaction only.

plicit connectors play a key-role in synthesizing the application's communication middleware. Hence their meta-model is provided here, too. Figure 6 shows all types of explicit connectors and describes their dependency to middleware components.

An explicit connector's type is defined by two dimensions:

- ***Communication Paradigm*:** AUTOSAR supports two communication paradigms: client-server and sender-receiver communication. On this account, the abstract superclass *ExplicitConnector* is specialized by two (also abstract) classes, class *ClientServerConnector* and class *SenderReceiverConnector*.
- ***Component Deployment*:** The second dimension is the outcome of how the connected components are deployed relative to each other. Two connected components may be deployed (i) within the same address space (IP), (ii) on the same ECU, but within different address spaces (XP) and (iii) on different ECUs (XN). Note these prefixes in Figure 6. Using this information, three concrete classes of types of explicit connectors are available by specialization. In [17] we specified structural designs at model level for each concrete connector type. When transforming the application level connector to middleware, these designs become the transformation's cornerstones.

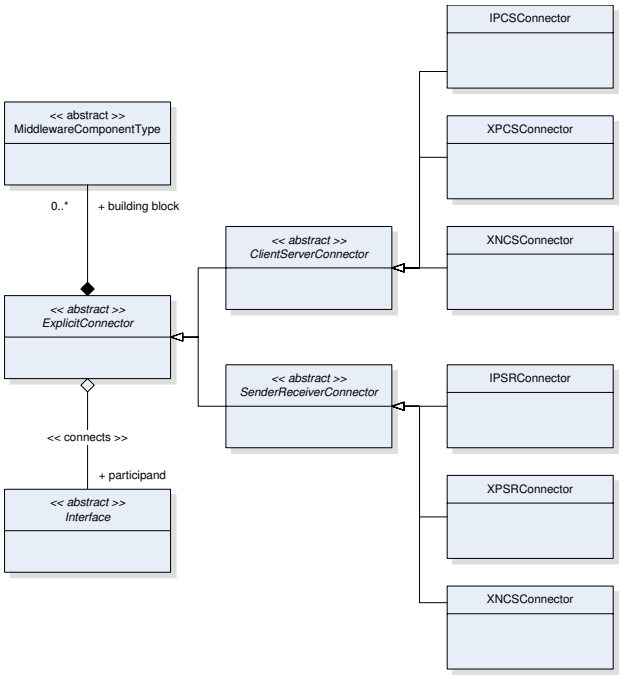

Figure 6: Meta-Model of Explicit Connectors

2.3 Composition and Interaction Standard

To incorporate the COMPASS component model into the AUTOSAR development process, the AUTOSAR component model has to be slightly extended. As AUTOSAR allows component connectors to be tagged (attributed), no syntactic modifications have to be made. The connectors only have to be granted "first-class entity" status and have to be extended by attributes, specifying the dimensions of an explicit connector.

At application level, AUTOSAR components may be composed, if their required and provided interfaces match. So (i) their connected interface are of the same type (communication class, interface elements etc.) and (ii) the connection is established by an explicit connector, that has a valid mapping to an available middleware structure (COTS middleware assembly).

In the COMPASS component model no explicit connectors exist. Middleware components—communication primitives and assemblies—are connected by local, implicit client-server connectors (local procedure calls). A connection is valid, iff (i) the connected interfaces are of the same type (interface elements etc.) and (ii) the communication primitives are connected in accordance to a valid structural design for explicit connectors.

2.4 Component Framework

The component framework provides infrastructural resources for the application's component architecture. The COMPASS component model is situated within the AUTOSAR infrastructure and is meant to build component middleware. Therefore, all infrastructural services (e.g., operating system abstraction, scheduler, etc.) have to be encapsulated within middleware components. As described in [17], communication primitives may expose an optional infrastructure port for connecting these infrastructural service components. COMPASS components are statically bound at compile time, instantiation and initialization at runtime has to be performed at application startup by one dedicated startup-component.

3 Conclusion and Future Work

In this paper we have described a component model for AUTOSAR component middleware that seamlessly cooperates with the AUTOSAR component model. Our component model can be used to improve the quality of AUTOSAR applications, as AUTOSAR applies the component paradigm to its application layer only. AUTOSAR

middleware, the *Basic Software*, adheres to a layered software architecture, and thus can not participate in the assets of CBSE, available at application level. By applying our component model to AUTOSAR *Basic Software*, the automatic synthesis of custom tailored, light-weight, component middleware from AUTOSAR application models and prefabricated communication primitives—the middleware components—is enabled.

To proof our concept, we have sliced the AUTOSAR *Communication Stack*, to be more precise, the *FlexRay* [7] *Communication Services* stack, and have built appropriate communication primitives from those slices. We then designed a component based application—a simple speed-aware door-lock application—and synthesized a component based component middleware (*Basic Software* and RTE) using our transformations and communication primitives. When comparing the so generated, application, and ECU-specific, middleware to a layered version, we achieved software that was smaller in size (30% smaller memory footprint) and of better runtime performance (10% less CPU usage)[6].

To improve our component model, ongoing research deals with the integration of hardware components into the COMPASS component model and with the association of explicit contracts with component model elements for automatic model validation and verification.

4 Acknowledgements

This work has been partially funded by the FIT-IT [embedded systems initiative of the Austrian Federal Ministry of Transport, Innovation, and Technology] and managed by Eutema and the Austrian Research Agency FFG within project COMPASS [5] under contract 809444.

References

[1] AUTOSAR GbR. *Layered Software Architecture 2.0.0*. http://www.autosar.org/download/AUTOSAR_LayeredSoftwareArchitecture.pdf.

[2] AUTOSAR GbR. *Software Component Template 2.0.1*. http://www.autosar.org/download/AUTOSAR_SoftwareComponentTemplate.pdf.

[3] AUTOSAR GbR. *Specification of RTE Software 1.0.1*. http://www.autosar.org/download/AUTOSAR_SWS_RTE.pdf.

[4] AUTOSAR GbR. *Technical Overview 2.0.1*. http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf.

[5] COMPASS. *Component Based Automotive System Software*. http://www.infosys.tuwien.ac.at/compass.

[6] W. Forster, T. M. Galla, C. Kutschera, and D. Schreiner. Technical report wp 2.4: Profiling, testbed measurements and candidate selection. Technical Report 2.4, COMPASS, 2007.

[7] T. Führer, F. Hartwich, R. Hugel, and H. Weiler. FlexRay – The Communication System for Future Control Systems in Vehicles. In *Proceedings of the SAE 2003 World Congress & Exhibition*, Detroit, MI, USA, Mar. 2003. Society of Automotive Engineers.

[8] P. Hansen. New S-Class Mercedes: Pioneering Electronics. *The Hansen Report on Automotive Electronics*, 18(8):1–2, Oct. 2005.

[9] H. Heinecke. AUTomotive Open System ARchitecture An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. In *Proceedings of the Convergence International Congress & Exposition On Transportation Electronics*, Detroit, MI, USA, 2004.

[10] K.-K. Lau and Z. Wang. A taxonomy of software component models. In *EUROMICRO-SEAA*, pages 88–95, 2005.

[11] B. Meyer. Applying "design by contract". *IEEE Computer*, 25(10):40–51, 1992.

[12] B. Meyer. The grand challenge of trusted components. In *ICSE*, pages 660–667, 2003.

[13] Microsoft. *COM (Component Object Model)*, 2007. http://msdn2.microsoft.com/en-us/library/ms680573.aspx.

[14] O. Nierstrasz and D. Tsichritzis, editors. *Object-Oriented Software Composition*. Object-Oriented Series. Prentice-Hall, Dec. 1995.

[15] OMG. *UML 2.0 Superstructure Specification*, 2005. http://www.omg.org/cgi-bin/doc?formal/05-07-04.

[16] OMG. *CORBA Component Model Specification Version 4.0*, 2006. http://www.omg.org/docs/formal/06-04-01.pdf.

[17] D. Schreiner and K. M. Göschka. Building component based software connectors for communication middleware in distributed embedded systems. In *Proceedings of the 2007 ASME/IEEE International Conference on Mechatronic and Embedded Systems and Applications (MESA07)*, ASME/IEEE, 2007. to appear.

[18] D. Schreiner and K. M. Göschka. Explicit connectors in component based software engineering for distributed embedded systems. In *SOFSEM 2007: Theory and Practice of Computer Science, Proceedings*, volume 4362 of *LNCS*, pages 923–934. LNCS, Springer, Jan 2007.

[19] D. Schreiner and K. M. Göschka. Synthesizing communication middleware from explicit connectors in component based distributed architectures. In *Proceedings of the 6th International Symposium on Software Composition (SC 2007)*, LNCS. Springer, 2007. to appear.

[20] B. Selic. Protocols and ports: Reusable inter-object behavior patterns. In *ISORC*, pages 332–339. IEEE Computer Society, 1999.

[21] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Jan. 1998.

[22] R. Weinreich and J. Sametinger. Component-based software engineering. chapter Component Models and Component Services: Concepts and Principles, pages 33–48. Addison Wesley, 2001.