Enabling of AUTOSAR system design using Eclipse-based tooling

H. Heinecke, M. Rudorfer, P. Hoser, C. Ainhauser, O. Scheickl

BMW Car IT GmbH, Petuelring 116, 80809 Munich, Germany

Abstract: AUTOSAR is a development partnership for standardisation of software architectures for the development of complex E/E systems. The software configuration process specified by AUTOSAR involves the handling of large amounts of data describing the E/E system. An efficient application of the process requires good and continuous tool support.
In this paper we propose an approach for AUTOSAR tooling, which is based on the technology and, more important, on the idea of Eclipse. Eclipse is one of the most successful open source projects of the last years with a strong influence on the industry. It provides an open development platform that can easily be extended. On top of Eclipse, the approach provides an open tool basis, which can be extended by special, free or commercial plug-ins.

Our AUTOSAR tooling approach is oriented towards the ideas behind Eclipse and focuses on reusing the success factors for a tool approach within the AUTOSAR community.

Keywords: AUTOSAR Tooling, AUTOSAR-Based Development Platform, Eclipse, AUTOSAR quick start

1 Introduction

Today's automotive system development is getting more and more complex because of several reasons (e.g. exponential increase of automotive functionality, restricted package envelope) [1]. To handle this complexity, the automotive industry has launched AUTOSAR, a development partnership for standardisation of E/E architectures. AUTOSAR specifies a highly configurable software architecture for Electrical Control Units (ECU) in the automotive domain [2].
The configuration process of the architecture's software modules involves the handling of large amounts of data describing the E/E system. Thus, for an appropriate application of AUTOSAR, a good tooling, which supports the users in handling all the configuration data during system development, is required. BMW Car IT has gained already a lot of experiences in this area by implementing a tooling prototype named Orpheus [3].

Based on the experiences collected at BMW Car IT, this paper introduces a novel tooling approach that promises to satisfy most of today's and tomorrow's AUTOSAR tooling requirements. The so-called AUTOSAR-Based Development Platform (ADP) has to be handled like an open tool framework, however limited to AUTOSAR members and partners only. Due to the fact that the AUTOSAR partnership comprises more than 100 partners and members, the layout equals to an open source activity, limited to the somewhat smaller world.
ADP shall provide as an *initial donation* the basic functionality every AUTOSAR tool suite needs (importing/exporting AUTOSAR templates, AUTOSAR meta model implementation etc.). The entire basic functionality (as depicted in Fig 2) is open to every member in the AUTOSAR development partnership and is continuously improved by contributions of the partners. On top of the basis, tool developers can provide their *area of expertise* in terms of pluggable extensions, released either as free or commercial products.
In the following sections we give a brief overview of the proposal for the promising approach. Section 2 sums up the AUTOSAR methodology. Section 3 gives the reader a survey of Orpheus and our gained experiences with AUTOSAR tooling. Section 4 introduces the motivation and the concept behind ADP and reflects the impact such a tooling platform has on the AUTOSAR development partnership.

2 The AUTOSAR methodology

Aside from the software architecture itself, AUTOSAR specifies the methodology for configuring the architecture. Each of the software modules in the AUTOSAR architecture can be configured and are partially generated. This way the software can be tailored for the special needs of the target ECU it is used on. Fig 1 shows the data flow of the configuration process, which mainly consists of the following four steps.

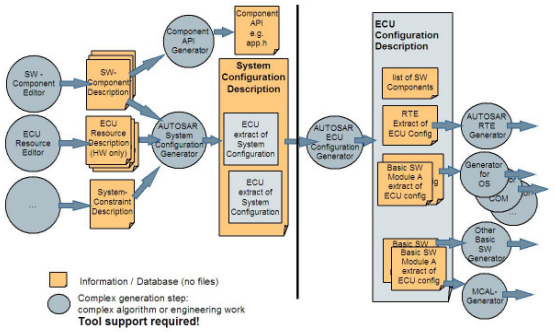

Fig 1 – AUTOSAR Design Flow

1. *System Configuration*
   In the first step, the system as a whole is being described and configured. In this step the topology of the hardware is described as well as the used application software components and their deployment on the ECUs.

2. *Extraction of ECU specific information*
   The overall system description is being split into several smaller descriptions. Each of those descriptions only contains the information needed for configuring the AUTOSAR architecture for one ECU.

3. *ECU Configuration*
   The information extracted from the system description is enriched with additional ECU specific information. This additional information is ECU specific in the sense that it only affects one dedicated ECU. This could be for example the size of a buffer or the diagnosis functionalities supported by the ECU.

4. *Code Generation*
   With all the information in place, in the final step the code for the AUTOSAR architecture's software modules is generated and deployed on the target ECU.

The shown process steps indicate, that a lot of configuration and generation tasks have to be done. Handling this process in an efficient and consistent manner, requires an adequate tooling support for the described steps. This means, that the appropriate usage of AUTOSAR in series production strongly depends on the availability of appealing AUTOSAR tools.

3 Orpheus - AUTOSAR tooling prototype

Orpheus is an AUTOSAR tooling prototype developed at BMW Car IT in the early stage of the AUTOSAR specification. In the following, we give a brief overview on Orpheus and, in particular, the experiences we collected. More details on the tool can be found in [3].

3.1 Motivation

After the initial phase, the specification of the AUTOSAR methodology had reached a state where it was ready for approval. In order to verify the feasibility of the specified process it needed to be applied to an example project.
In this first proof of concept the configuration and generation of one module of the AUTOSAR architecture was realised. The module chosen was the Runtime Environment (RTE), which is the communication middleware of the AUTOSAR architecture.

In such an early phase no AUTOSAR tools were available on the market. Orpheus was developed to fill this gap for the concept proof. Another intention of Orpheus was to give AUTOSAR users a first impression of how ECU development with AUTOSAR could be.

3.2 Technical background

Orpheus is an Eclipse-based tooling prototype.

> "Eclipse is an open source community whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle." ([4])

The reason for selecting Eclipse as a base development platform was, that it already provides a lot of functionalities needed:

- Large *core functionality* is available (e.g. basic editors for quick visual representation of the model, workspace-concept to administrate Orpheus projects)
- *OAW* [7] provides a code generation framework used for generating the RTE
- *GEF* [8] provides a powerful framework for creating graphical editors
- With the *plug-in mechanism* of Eclipse, Orpheus is easy composable and extendible.

3.3 Functional Overview

As described in the technical background section, Orpheus is based on Eclipse. On top of Eclipse, we have developed several basic functionalities, an AUTOSAR tool such as Orpheus must have:

- An AUTOSAR compatible meta model
- Import/Export of AUTOSAR compatible models
- Graphical editors to modify the model
- Model validation to guarantee a consistent and contradiction-free model
- Persistence mechanism for the storage of the model.

On top of this basic functionality, Orpheus provides some sophisticated features (e.g. the RTE generator) [3].

As Orpheus is a proof of concept, it does not cover the whole range of the AUTOSAR methodology. Nevertheless, with the comfortable extension and adoption mechanism provided by Eclipse, Orpheus can be extended easily with new features, that are currently not supported by AUTOSAR. On the one hand, the tool is continuously extended by additional

plug-ins (e.g. a Timing plug-in for the verification of the timing behavior of the developed system), on the other hand the AUTOSAR-compatible meta model of Orpheus is used to analyse, develop and to introduce new meta model features and concepts.

3.4 Experiences

The technical background section has shown the outstanding applicability of Eclipse as tool basis, since it provides a lot of desirable features (e.g. core functionality, extensibility, adaptability, …)[1].
Despite the tool basis, we had to implement a lot of AUTOSAR-specific *basic functionality* as listed in the functional overview section (e.g. persistence mechanism, input/output, core meta model, editors, …). Prototyping a first AUTOSAR RTE generator showed us, that a lot of core AUTOSAR functionality had to be implemented, before reaching the real development task. So, this fundament was needed as a prerequisite to provide the actual functionality (e.g. RTE generator). This means, that only when this basic functionality is available, the user can exploit his area of expertise (see Fig 2).

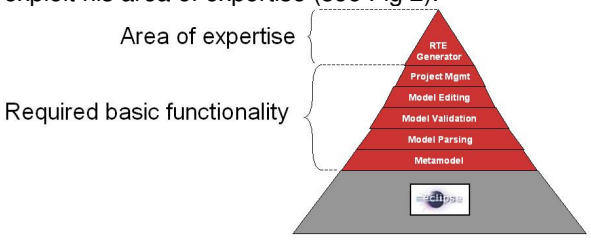

Fig 2 – The area of expertise on top of the tool

Reflecting about the basic functionality indicates an interesting relation to Eclipse:

> Eclipse represents a fundament for application development in general. The basic functionality of Orpheus represents a fundament for a specific application development, namely for AUTOSAR tool development.

Keeping this observation in mind, we analyse now the experiences gained from developing the basic functionality:
The basic functionality was implemented by a *small group* at BMW Car IT. These plug-ins were only written as proprietary developments.
Therefore, the developers had only *feedback* from a restricted user group. For example we developed some graphical editors to model AUTOSAR entities. Those editors were good for small scenarios, but, as we realized much later, were not usable for big projects. If we published those editors earlier, we would have received user feedback and could have improved them, or, even better, some other person would have improved the editors and contributed them back to the community, which means, back to us. Obviously, this limited feedback had impact on the speed to reach quality and maturity of the tool.
As the resulting software was not shared with others, the effort that was spent on providing the basic functionality was only exploited BMW internally. *Redundant development* of similar functionality had to be done by several other AUTOSAR tool developers elsewhere. For example we had to implement a mechanism to read and write the AUTOSAR templates from and to XML. We are almost sure that in a lot of other companies, people developed the exact same functionality in a similar way. So, if we had published those Readers/Writers to the AUTOSAR community, a lot of duplicate work could have been avoided. Joining forces on the implementation of the basic functionality would have speed up the development of the overall tool.

The experiences described above (no community and therefore little feedback, redundant development) reveal that the development of the basic functionality was associated with some hurdles, decreasing the implementation quality and speed.

The next chapter now illustrates a promising approach, which caters to the above-mentioned issues. As a relation between Eclipse and the basic functionality of Orpheus was discovered before, the new approach exploits the Eclipse success factors in order to absorb the identified issues of Orpheus and, in general, of AUTOSAR tools.

4 ADP - AUTOSAR-Based Development Platform

4.1 Definition

In this chapter we propose a novel "Open" Tooling approach. The term “Open“ in the context of this paper is defined as a common core platform that is open *within* the AUTOSAR community, not to the general public. The limitation is mandatory due to contractual regulations within AUTOSAR. However, because AUTOSAR is already a large partnership (with more then 100 partners and members), this restriction does not reduce the benefits of an “open source” strategy.

4.2 Situation today

Orpheus was one of the first developed AUTOSAR IDEs and gave a first impression of how such a tool could look like, but was never planned to become a

[1] Already in the early stage of Orpheus (2004) Eclipse and its technology were a good choice for a tool basis. Actual Eclipse releases are even more attractive since the technology has been enhanced successively (e.g. GMF).

series tool and only covers a subset of the AUTOSAR functionality.
Since then, several tool vendors introduced their products to the market. However, there is currently no tool chain available which provides the complete required functionality. One of the reasons for that is, that the AUTOSAR methodology implies the handling of large amounts of information describing the E/E system. Another difficulty for tool vendors is that different OEMs with different specific needs, for example features to cope with legacy systems, force them to implement different custom-made additions to their tools, which are quite time consuming. Additionally, today's tools are proprietary and not open which prevents extension by third parties or the customer himself.
In the following, an approach that considers and neutralizes those reasons will be presented

4.3 Exploiting the Eclipse success story

Eclipse is one of the most successful (open source) projects around. It is an open development platform for almost every kind of Rich Client Software with a large community that builds and supports it.
The key benefit of Eclipse is its extensible architecture that is based on the usage of plug-ins. Eclipse consists of a small kernel and a set of basic plug-ins that provide functionality that is needed by many users. Eclipse can be extended by writing custom plug-ins, which bring new functionalities to the platform.
As Eclipse is such a convenient starting point for developing applications, more and more people use it, extend it, and themselves contribute their extensions back to the community, so that the Eclipse core grows in functionality and quality.
Those success factors of Eclipse could be applied to an Open AUTOSAR Tool Framework as well.

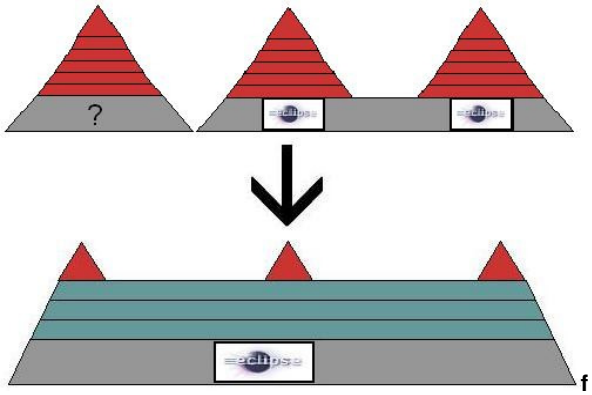

Fig 3 – Common tool basis on top of Eclipse

For such an approach, some requirements have to be fulfilled. We need an *open tool basis* that is used by a large number of developers. This basis has to provide a set of core functionalities, needed by almost every AUTOSAR tool. The core platform has to be *extensible* in order to add custom functionality. Fig 3 shows the effort needed for developing AUTOSAR tools with and without such a common AUTOSAR development platform.
Last but not least, it needs a *community*. Such a community cannot be created or established. This community has to form itself because of the benefit that such an approach has.

The following sections outline a rough idea, how the structure of an AUTOSAR-Based Development Platform could look like and some thoughts on the open platform, extensibility and community.

It leaves and creates enough headroom for free as well as commercial extensions of the core functionality. It is a chance rather than a threat to commercial tool vendors as they can focus on their expertise and are freed from coding the same basic functionality over and over again.

4.4 Structural Overview

The basis for the AUTOSAR-Based Development Platform is Eclipse with its plug-in mechanism. As illustrated in Fig 4, the architecture of the development platform defines three layers on top of Eclipse.

- MDSD Layer
- AUTOSAR Layer
- Custom Layer

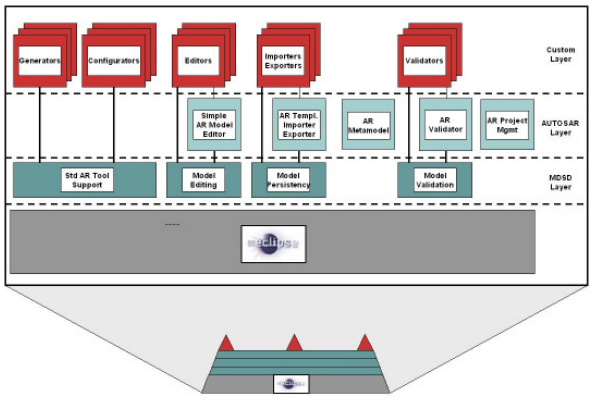

Fig 4 – The structural overview of ADP

The *MDSD Layer* provides rudimentary concepts and functionality needed for building a Model Driven Software Development (MDSD) tool. The services located in the MDSD Layer provide mechanisms for loading, saving, accessing, manipulating and validating models. For most of these mechanisms, the core functionality is already available in today's Eclipse distributions:

- Ecore for designing the meta model

- OAW for code generation
- EMF for persistence mechanism
- GMF, GEF for graphical editors
- OCL for model validation

The concepts and functionalities are not specific to AUTOSAR. They build the basis for the AUTOSAR specific services in the AUTOSAR Layer and for custom services, which reside in the Custom Layer.

On top of the MDSD layer the *AUTOSAR Layer* is located. It provides plug-ins, which extend the basic services of the MDSD Layer with AUTOSAR specific services. The core plug-in is the meta model plug-in, which provides an implementation of the most recent AUTOSAR meta model. Further plug-ins provide functionalities of reading and writing AUTOSAR conform XML files, validating an AUTOSAR model and viewing AUTOSAR models. The AUTOSAR Layer makes use of various AUTOSAR specifications. This is the reason why the usage of the AUTOSAR-Based Development Platform has to be limited to AUTOSAR partners and members only (see 4.1).

The *Custom Layer* is the layer where the actual tool functionality is located. These functionalities can access and build on the services provided by the MDSD and AUTOSAR Layer and are not part of the tool basis. They are implemented by tool vendors or even might be implemented by the ECU developers themselves if some OEM specific proprietary extension is needed. Examples for custom plug-ins provided by tool vendors would be a configurator for an AUTOSAR module or any fancy editor with sophisticated support for editing AUTOSAR models.

The three layers mentioned above represent the architecture of the proposed approach. In the following sections, the benefits of such an approach will be illustrated and discussed.

4.5 Open Tool Basis

As the success story of Eclipse suggests, an open tool basis that is used by a large number of users is required. The basis should provide basic functionality that most of the tool developers need before they can provide their expertise. The section before has depicted the basic functionality such an open tool basis should contain.
With the help of this basis a lot of benefits can be exploited. Firstly, not every tool developer has to implement the basic functionality on his own; *redundant implementation* can be avoided. The developers can concentrate their effort on their area of expertise and extend the basis. This leads to an increased tool *development speed and quality.*
Furthermore, the platform has great benefits for tool developers as well as end users:

For a tool developer it is attractive because it *lowers the initial hurdle* of entrance to the AUTOSAR tooling market. They can immediately start to work on their area of expertise and do not need to implement first the basic functionality offered by the basis.
For the end users of the tool, like ECU developers, the open basis lowers the hurdle of entrance of AUTOSAR since they have a free and immediate access to it. This results in a better integration of AUTOSAR into the corporation's processes.
The open tool basis is an auspicious approach to resolve current issues in the system development tooling chain. The basis represents a kind of interface for every extension. Therefore the *interoperability* between the different extensions is easy to manage and the coupling between them can be performed without time loss.
As the section before has depicted, the meta model plug-in is one of the core plug-ins provided by the basis. It is an implementation of the most recent AUTOSAR meta model. Hence, the AUTOSAR-Based Development Platform is always synchronized with the AUTOSAR specification and the *migration to new releases* is made centrally in one plug-in. This agility allows furthermore an *early validation of new releases*, because it is immediately in use.
In the end, the common basis platform is also useful for the communication across development partners. Misunderstandings caused by the usage of different tools can be avoided and increase therefore the effectiveness of communication between development partners.

4.6 Extensibility

Having this open tool basis with its core functionality brings us to the next point – extensibility.
There are various reasons or scenarios for an extension of the core:

- *Providing special functionality or replacing core functionality by sophisticated, improved functionality:* An RTE generator is an example that we do not see inside the core platform. A party that specializes on RTE generator can provide them as a plug-in.
  Another party specializes on graphical editors with supreme usability, good performance. Although the open tool basis contains a simple SWC editor, users may decide to buy and use those sophisticated graphical editors without having to abandon the rest of their tool chain.
- *Adding functionality on top of AUTOSAR:* Another party may be specialized in a certain field of research, for example timing analysis and wants to port their product to be compatible with AUTOSAR. They can use the ADP and build their software on top of that. They can improve and

sell their product without spending time on re-implementing the AUTOSAR core.

- *Adding configuration tool for special AUTOSAR software, e.g. a single AUTOSAR BSW component:* Yet another software party is selling AUTOSAR basis software components. Those BSW components have to be configured, using part of the information that is present in the AUTOSAR model. The can write a small configurator plug-in that is integrated into ADP.
- *Self-made plug-ins for project specific requirements:* In a ECU project where AUTOSAR is used, some modeling steps have to be performed over and over again. This is time-consuming and error-prone and could be automated. Scripting comes to ones mind. Instead of writing a script in a special scripting language, one could also implement this as a simple plug-in on top of ADP.
- *Adding OEM specific functionality, e.g. connectivity:* Last but not least, a lot of OEMs and Tier-1s have special requirements when integrating the whole of AUTOSAR into their companies' processes and backend-systems. This integration is so company specific that it can and will never be addressed by the AUTOSAR standard. This connectivity to proprietary backend systems could be realized as OEM specific ADP plug-ins.

All those examples show how extensibility can improve the benefit of the overall platform. Those extensions can be either free, part of another commercial product or a commercial product itself.
Core functionality is implemented once and is reused. This frees resources to concentrate on the area of expertise.
Migration scenarios are addressed, as proprietary parts of the tool-chain can be used in the first place, later replaced by standard-conform plug-ins without having to change the rest of the tooling.
The possibility, to exchange certain plug-ins opens the market for a competition on plug-in level, not on the whole tool suites. This allows the user to select the plug-ins which are best suitable for him (freedom of choice) and allows the tool vendor, to differentiate on quality and development speed from his competitors.

4.7 Large Community

A vital factor for the success of Eclipse as a tooling platform is the active community that evolved around it. The formation of such a community is one of the key concepts of the AUTOSAR-Based Development Platform. As mentioned at the beginning of this chapter, the community would consist of members and partners of the AUTOSAR development partnership.

The most obvious benefit is the improvement of quality and functionality through *user feedback*. If software is used by a large number of users and developers it is tested more extensively as if only a small group uses it. This leads to a better test coverage of the ADP's code.
Better test coverage is only one benefit of the user's feedback. The community acts as an innovation catalyser by giving feedback on missing functionality and inspiring new functionality.
Another benefit of a large and active community are *contributions*, which are made to the ADP by members of the community. Tool developers who build ADP-based tools might want to share some features they implemented which are not part of their core competences. They could do so by contributing those features to the ADP. This leads to a continuous extension of ADP's functionality.
A large number of users also means that a lot of experience has been made with the framework. The community can provide *support* to ADP users who need help with the framework.

4.8 Challenges and Prerequisites of ADP

The previous sections have discussed some thoughts regarding the AUTOSAR-Based Development Platform. They outlined the potentials and benefits of such a framework. However, in order to introduce the ADP some challenges have to be faced and prerequisites have to be fulfilled.
As a very important prerequisite of the platform, a group of interested parties must provide the tool basis as an *initial donation*. Every member of the community needs this basic functionality before they can start with developing tool extensions for their own area of expertise.
One of the greatest challenges ADP has to face is the *reaching of a critical mass* of active participants. Only if a certain amount of participants establish, maintain and extend the tool basis, the platform can reach the required quality and maturity.
In order to permit an appropriate development of the platform, some prerequisites concerning the *infrastructure* of the development environment must be fulfilled. For example, an issue tracking system (e.g. bugzilla, jira), collaboration tools (e.g. cvs, svn) and bulletin boards are needed for those infrastructure issues.

5 Conclusion

In this paper we propose the approach of an open development platform for AUTOSAR tools. The platform shall provide basic functionalities common to all AUTOSAR tools (importing/exporting AUTOSAR templates, AUTOSAR meta model implementation etc.). It can be freely used by the partners and members of the AUTOSAR

development partnership. We propose not only to base the AUTOSAR development platform on Eclipse but also to apply Eclipse's success factors to it.

Similar to Eclipse the ADP shall provide an *open and extensible architecture.* This leads to an increase of the development speed for AUTOSAR tools. Furthermore, it eases the creation of a continuous and homogenous tool chain. Another important aspect of our proposal is the exploitation of Eclipse's most vital success factor: its *large community*. The community helps Eclipse to continuously improve its quality and extend its functionality. Applying this success factor implies encouraging actively the growth of a large community around the AUTOSAR development platform and maintaining it.

As desirable all these advantages are, they will not come all by themselves. The initiation of such an open development platform faces some challenges. The most obvious challenge is reaching a critical mass of interested partners who commit themselves to the idea and support it actively. However, as the success story of Eclipse has shown, once the critical mass of interested parties is reached, the benefits and advantages to the community are numerous and overweigh the initial efforts.

Summarizing, ADP is a proposal for applying the Eclipse success story to the AUTOSAR tooling environment in order to provide AUTOSAR users with the tools they need.

6 References

[1] H. Heinecke: "Automotive system design - challenges and potential", DATE Conference 2005, Munich Germany, 2005.

[2] AUTOSAR development partnership: "AUTOSAR: Technical Overview", http://www.autosar.org/download/AUTOSAR_TechnicalOverview.pdf, 2005

[3] C. Salzmann: "Erfahrungen mit der technischen Anwendung einer AUTOSAR Runtime Environment", VDI-Kongress "Elektronik im Kraftfahrzeug", Baden-Baden Germany, 2005.

[4] Eclipse: "*Eclipse – An Open Development Platform*", http://www.eclipse.org/, 2007

[5] Eclipse Ecosystem: "*About the Eclipse Foundation*", http://www.eclipse.org/org/, 2007

[6] Eclipse EMF: "Eclipse Modeling Framework", http://www.eclipse.org/emf/, 2007

[7] Eclipse OAW: "openArchitectureWare", http://www.eclipse.org/oaw/, 2007

[8] Eclipse GEF: "Eclipse Graphical Editing Framework", http://www.eclipse.org/gef/, 2007

7 Glossary

IDE: Integrated Development Environment

ADP: AUTOSAR-Based Development Platform

OEM: Original Equipment Manufacturer

ECU: Electronic Control Unit

RTE: Runtime Environment, Communication Middleware

VFB: Virtual Function Bus

OAW: OpenArchitectureWare

GEF: Graphical Editing Framework

GMF: Graphical Modelling Framework

SWC: Software Component Template