Interoperable AUTOSAR tooling with Artop

Sebastian Benz, Michael Rudorfer, and Christian Knüchel

BMW Car IT GmbH,
Petuelring 116, 80809 München, Germany
{sebastian.benz,michael.rudorfer,christian.knuechel}@bmw-carit.de
http://www.bmw-carit.de

Abstract. Developing tools for the automotive standard software architecture AUTOSAR is challenging due to the complexity and heterogeneity of the underlying domain. Artop is our approach to solve these challenges by providing a common, extensible tool platform with an implementation of basic functionalities required by all AUTOSAR tools. Artop thereby enables the tailoring of Artop-based tools to its user's needs by providing means to integrate and combine different functionalities from different tool vendors. In this paper, we demonstrate the application of Artop for a case study from BMW Group.

Keywords: model-based tooling, AUTOSAR, automotive

1 Introduction

AUTOSAR has been launched by the automotive industry to handle the increased complexity of E/E systems. Its main objective is to create a basis for industry collaboration on basic functions and to increase reuse of software across different *Electronic Control Units (ECUs)*. AUTOSAR achieves this by standardizing a common middleware that is independent of the underlying hardware: the *Runtime Environment (RTE)*. AUTOSAR systems are developed by a multi step modeling-process that is described by the AUTOSAR methodology. Due to the size of AUTOSAR systems, tooling is necessary for developing, maintaining, and sharing development artifacts. Currently tools face the challenge of having to handle huge models on different abstraction levels and from different points of view.

In this paper, we first give an overview on common challenges for AUTOSAR tooling. Then we present Artop, the AUTOSAR tool platform, as our approach to solve these problems in a community. Finally, we demonstrate the application of Artop for a case study at the BMW Group.

2 Challenges in AUTOSAR Tooling

The metamodel standardized in AUTOSAR is complex and requires tools that provide abstractions that ease the development of AUTOSAR models. The AUTOSAR methodology describes the different modeling phases of an AUTOSAR

system. Each phase comprises of the creation of data models that describe different system aspects, such as ECU hardware configuration, system topology description, and software component descriptions. The methodology results in the generation of configuration parameters and production code, which implements the APIs of the software components. Due to the variety of systems that must be supported, the whole metamodel is large and complex (the current AUTOSAR release consists of 881 classes). Thus, one important task of AUTOSAR tooling is to guide and support the user in working with the complex data models that are created as part of the AUTOSAR methodology.

The AUTOSAR standard is constantly evolving in order to cope with technical evolution[1]. A new release introduces new concepts to the standard and includes maintenance changes to the previous releases. These changes often result in incompatibilities between different releases, on RTE as well as data model level. However, the life cycle of a car is quite long (up to seven years of production plus the duty to provide spare parts for up to 10 years after end of production). In this period existing software must be maintained with the consequence that tooling is required that supports older AUTOSAR releases. Thus, when implementing an AUTOSAR tool one must find ways for supporting different releases at the same time.

AUTOSAR systems are specified in a collaborative and distributed manner which requires tools to support this by means of revision control, model merging, and model exchange. For example, first an OEM specifies the overall system structure, then a supplier, which is responsible for implementing one of the software components, enriches the component descriptions. Finally, these changes must be merged back into the original model of the OEM. Even though, there are well established approaches for revision control on textual level, when using graphical notations there are still open issues.

The AUTOSAR standard covers different aspects of automotive software and hardware. The diversity of the covered aspects requires different representations of the underlying models. For example, communication channels between software components can be represented intuitively in a graphical way, but other aspects might be better represented using textual, or form-based editors. The challenge is to be able to edit and view specific aspects of a model in the most appropriate manner while at the same time keeping the underlying model consistent. For example, when a software component is changed using a graphical editor, changes must be propagated to textual editors showing the same software component. Furthermore, non-functional modeling concerns, such as timing or safety specifications, crosscut traditional decomposition criteria, e.g. the decomposition into software components. This results in these specifications being cluttered across the whole model, which opposes separation of concerns and thereby hinders maintainability.

Probably the biggest challenge in AUTOSAR tooling is the diversity of the AUTOSAR standard with its different abstraction levels, different development phases, and different domains. The diversity manifests in a wide variety of differ-

[1] Specifications are available for information only via http://www.autosar.org

ent workflows that must be supported by an AUTOSAR tool depending on the context of the current user. For example, an AUTOSAR tool for software developer needs completely different functionalities than a tool for a system architect. Our experience is that there will never be "the AUTOSAR tool" that covers all different aspects and functionalities. What is required is a tooling environment that is extensible and that can be tailored to the needs of different users. In the remainder of this paper we will introduce our approach for such a tooling environment: Artop.

3 Artop - the AUTOSAR tool platform

To handle the variety of AUTOSAR modeling workflows, one must find a way to tailor a tool for specific workflows by adding required features and removing unnecessary ones. There are two main premises for such a tooling environment: firstly, functionality must be portable, meaning that existing features can be used across different tools. Secondly, the effort of implementing existing tools with new features must be as low as possible such that missing features can be added easily. Our goal is to have an ecosystem for AUTOSAR tools that consists of a platform that provides common base functionalities required by all tools (e.g. metamodel implementation) and which is complemented by individual features from various providers.[2] To reach that goal the Artop User Group[2], a group of licensed users of the AUTOSAR standard, has been founded.[1] It was launched in October 2008 and provides Artop - the AUTOSAR tool platform. It is run by the Artop Design Members which currently are BMW Car IT, Continental Engineering Services, Geensys and Peugeot Citron Automobiles.

One of the key prerequisites for a successful ecosystem is a common technical platform that allows for easy adoption of the platform and for easy extension of the commodities by the differentiating features of the individual company. Eclipse[3] is an open source community, whose projects are focused on building an open development platform and that clearly fulfills this requirement. Amongst other functionalities Artop comes with metamodel implementations of the AUTOSAR standard in versions 2.0, 2.1, 3.0, 3.1 and 4.0. One of the technology projects that complement Eclipse is the Eclipse Modeling Framework (EMF). EMF, is based on Ecore, a language for defining the abstract syntax of modeling languages. There exists a wide variety of frameworks based on EMF that support different use cases such as, building graphical or textual editors, model transformation languages, or code generation frameworks. The AUTOSAR metamodels are implemented using EMF which takes tool interoperability one step further: Where former tools were only able to exchange data in form of the AUTOSAR XML files on the file level, the EMF model allows different features from different authors to work in parallel on the exact same model in memory. In the next section we demonstrate how we used the functionalities provided by Eclipse, EMF, and Artop for tailoring an Artop-based IDE to our needs.

[2] http://www.artop.org
[3] http:www.eclipse.org

4 Case Study

This section presents a case study, in which we tailored an Artop-based IDE to the needs of an AUTOSAR software developer at BMW Group. A developer faces the following tasks when implementing a new AUTOSAR software component:

Specify software components: An AUTOSAR model must be created that specifies the software components, the required and provided interfaces, as well as the runnables that comprise this software component.

Generate interface code: AUTOSAR standardizes a C API for software components and runnables. The API code is generated from the software component specification that is defined in the previous task.

Implement software component behavior: The actual behavior of a software component is implemented in C, which requires a C development IDE.

Write unit tests: Software components should be developed in a test-driven manner. This requires that software components can be tested inside of the IDE without deploying the software to the actual target, which inhibits instant feedback during development.

Currently there is no AUTOSAR tool available that supports all these use cases within the same tool. However, by using an Artop-based tool we were able to create an IDE for AUTOSAR software components that fulfills exactly these use cases by combining existing features with newly created ones.

4.1 Specifying software components with ARText

Textual languages provide a fast and efficient way of creating models. Artop provides ARText, a framework for defining textual languages for AUTOSAR[4]. Part of Artop is the ARText-based software component language (ASCL). We use ASCL for specifying software components. Models described in ASCL are abstractions of the actual AUTOSAR metamodel. This has the advantage that the language is easy to understand and that software components are specified in an AUTOSAR release independent way. Part of ASCL are model transformers that transform ASCL models into instances of different AUTOSAR releases and thereby provides release independent development of AUTOSAR models. Figure 1 shows an example specification of a software component.

4.2 Generating production code:

The communication between different software components is performed by the AUTOSAR RTE. This requires the actual implementation of a software component to implement a corresponding API that is generated from the software component description. We integrated an existing code generator for the software component interface code into Artop. The code generator takes as input the AUTOSAR models that are generated by ASCL and is triggered from within the IDE.

[4] Screencast introducing ARText http://vimeo.com/channels/artext

```
package arpSafetyCar

interface clientServer ILifecycle {
  operation changeVehicleMode {
    in EVehicleMode vehicleMode out tBoolean success
  }
}

component application ModeManager {
  ports {
    receiver rMode requires IVehicleMode
  }
}
```

Fig. 1. Software component specification in ARText software component language.

4.3 Implementing software components using CDT

For implementing the actual behavior of the software components, we used CDT[5], which is a fully functional C and C++ development environment based on the Eclipse platform (see Figure 2 for an example). Because Artop and CDT are based on Eclipse, CDT can be easily integrated into an Artop-based IDE. This results in having a modeling and programming environment integrated into the same IDE which eases development by enabling a seamless workflow.

```
#include "Rte_Type.h"    // generated
#include "Rte_ModeManager.h"  // generated

void ModeManager_run(void)
{
  // Mode manager implementation
}
```

Fig. 2. Software component implementation in C using CDT.

4.4 Unit testing software components with JAC

The goal is to develop software components in a test-driven manner. Currently, there exists no unit testing framework for AUTOSAR components, which is why we developed the Java AUTOSAR Components testing framework (JAC). In JAC tests can be specified in Java (or any other JVM-based language) using Junit, which makes test case implementation more efficient than writing tests in C (see Figure 3 for an example test case). Part of JAC is a code generator for a test-specific-RTE which can be triggered from test cases written in Java. The effort of developing an own testing environment was considerably low, because

[5] http://www.eclipse.org/cdt/

Artop and Eclipse enabled us to use existing frameworks for unit testing (Junit[6]), model transformation (QVT[7]), code generation (Xpand[8]).

```
public ModeManagerTestRte rte = new ModeManagerTestRte();

public void testModeManager() throws Exception {
  rte.setRequestedMode(EVehicleMode_Maintenance);
  rte.testModeManager_run();
  assertTrue(modesInBuffers(EVehicleMode_Maintenance));
}
```

Fig. 3. Test specification in Java with Junit.

5 Conclusion

Artop demonstrates how to build a model-based tooling ecosystem hat supports close interoperability of tools needed to develop systems following the development path of the AUTOSAR standard. Our case study exemplifies the application of Artop for creating a tooling environment that supports model development, production code generation, test harness generation, and test case specification. Such an integrated development environment enables a continuous workflow and thereby greatly benefits developer productivity. This has been possible because of the usage of an open and extensible platform that enables the integration of different tooling fragments and thereby tailoring it to the specific needs of an AUTOSAR developer. A key factor for the success of Artop can be attributed to the existing ecosystems it is built upon, such as Eclipse and EMF. Users of Artop can greatly benefit from the existing functionalities offered by these ecosystems.

Acknowledgments. We thank Tilmann Ochs, Dana Wong, and Stefan Schmierer for their input and helpful comments on this paper.

References

1. Harald Heinecke, Michael Rudorfer, Paul Hoser, Christoph Ainhauser, and Oliver Scheickl. Enabling of autosar system design using eclipse-based tooling. European Conference of Embedded Real-Time Software, 2008.
2. Michael Rudorfer, Christian Knchel, Stephan Eberle, Romain Sezestre, Stefan Voget, Robert Kiss, and Aldric Loyer. Artop an ecosystem approach for collaborative autosar tool development. European Conference of Embedded Real-Time Software, 2010.

[6] http://www.junit.org
[7] http://www.eclipse.org/m2m/
[8] http://www.eclipse.org/modeling/m2t/?project=xpand