Towards Integrating SysML and AUTOSAR Modeling via Bidirectional Model Synchronization

Holger Giese, Stephan Hildebrandt and Stefan Neumann
[first name].[last name]@hpi.uni-potsdam.de
Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany

Abstract: During the overall development of complex engineering systems different modeling notations are employed. In the domain of automotive systems, for example, SysML models are employed quite early to capture the requirements and basic structuring of the whole system, while AUTOSAR models are employed later to describe the concrete software architecture. Each model helps to address the specific design issue with appropriate notations and at a suitable level of abstraction. However, when we step forward from SysML to the software design with AUTOSAR, the engineers have to ensure that all decisions captured in the SysML model are correctly transferred to the AUTOSAR model. Even worse, when changes occur later on either in the AUTOSAR or SysML model, today the consistency has to be reestablished in a cumbersome manual step. Otherwise the resulting inconsistency can result in failures when integrating the different system parts as captured by the SysML model. In this paper, we present how techniques for the model-driven development domain such as meta-models, consistency rules, and bidirectional model transformations can be employed to automate this task. The concept is exemplified by an experiment done within an industrial project.

1 Introduction

The development of complex engineering systems involves different modeling notations from different disciplines. Taking the domain of automotive systems as an example, SysML (System Modeling Language) [Sys08] models are employed quite early to capture the requirements and basic structuring of the whole system and AUTOSAR (Automotive Open System ARchitecture)[1] models are used later in the development process to describe the concrete software architecture and its deployment. Using these different model helps to address each specific design issue with an appropriate notation and at a suitable level of abstraction.

When going from the system design with SysML to the software design stage with AUTOSAR, today, the engineers have to ensure manually that all decisions captured in the SysML model are correctly transferred to the AUTOSAR model. When changes occur later on either in the AUTOSAR or SysML model, the situation is even worse: The consistency has to be reestablished in a cumbersome manual step that inspects both models and transfers all detected changes. Otherwise, the integration of the different system parts as captured by the SysML model and refined in the AUTOSAR model may fail.

Model-Driven Engineering and model transformation are a promising direction to approach this problem (cf. [Win07]). Models are used to describe the system under development from different viewpoints and on different levels of abstraction. The use of different kinds of models leads to the problem of keeping those models consistent to each other. At this point, model transformation systems play a central role.

Triple graph grammars (TGGs) are a formalism to declaratively describe correspondence relationships between two types of models. They were introduced in [Sch94]. A TGG based transformation system can perform model transformations using this declarative transformation specification.

[1] http://www.autosar.org

Figure 1: A simple meta model to describe components, ports and connectors

In several context different variants of TGGs have been employed for model synchronization such as the integration of SysML models with Modelica simulation models [JPB08], keeping models from the domain of chemical engineering consistent [BHLW07] and transformations from SDL models to UML models and vice versa [BGN+04].

This paper reports about a project with industry in which we investigated how the SysML tool TOPCASED and the AUTOSAR tool SystemDesk can be integrated. We use techniques from the model-driven development domain such as meta-models, consistency rules and bidirectional model transformation resp. model synchronization to automate the integration task. The model transformation permits to automatically derive initial AUTOSAR models from SysML models and to reestablish consistency between both models in case of changes in one of them. More specifically, our model synchronization approach [GW09] based on triple graph grammars (TGG) [Sch94] has been employed to synchronize both models such that changes within one of them are automatically transferred to the other one.

The automatic synchronization of models not only reduces the costs and time required for the transition from the SysML to the AUTOSAR model. It also reduces the cost and time to reestablish consistency in case of changes in either model. In addition, the automated synchronization is less error prone than manual labor employed today and enables to employ iterative and more flexible development processes as the costs for iterations or changes are dramatically reduced.

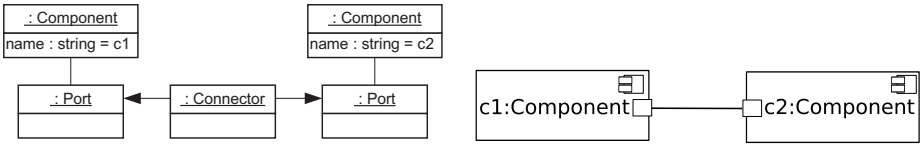
The structure of the paper is as follows: The employed concepts from MDE are outlined in Section 2. The considered modeling notations SysML and AUTOSAR are introduced in Section 3 and the model synchronization between both are presented in Section 4. The approach and its support for model synchronization are presented in Section 5 before we discuss its suitability for several typical usage scenarios, such as the initial transfer of information or change propagation, in Section 6. The paper closes with a final conclusion and an outlook on planned future work.

2 Model-Driven Engineering

At the core of the *Model-Driven Engineering* (MDE) approach, models build the basis for the development of systems. Different kinds of models are used to describe the system under development from different viewpoints and on different levels of abstraction. While these models are all related to each other, they need to be kept in sync after modifications. For this purpose, model transformation and synchronization systems can be used that create a new target model from an existing source model (transformation) or modify an existing target model to make it consistent to a source model (model synchronization). More details on model transformation will be presented later in Section 4.

2.1 Meta Models

In MDE, general purpose modeling languages as well as *Domain-Specific Languages* (DSLs) are commonly used to describe problems specific to the domain where the system is to be deployed. Such languages can be textual or visual languages. Commonly, meta models are used to describe the abstract syntax of such languages. A meta model defines all the elements that can be used in a valid model, as well as the relationships between them. Figure 1 shows a simple meta model for the description of components, ports and connectors. Figure 2 shows an example instance model in abstract and concrete syntax. The abstract syntax shows all elements of the model as objects in an object diagram. The concrete syntax uses a specific graphical notation.



(a) Abstract Syntax (b) Concrete Syntax

Figure 2: An example model conform to the meta model in Figure 1

2.2 Constraints on Models

Usually, UML Class Diagrams are used to describe the structure of a meta model. However, there are some properties that cannot be expressed using class diagrams but which must be fulfilled by valid instances of a meta model. An example is a constraint on the value an attribute may take. For this purpose, the *Object Constraint Language* (OCL) can be used to express such additional constraints. In Figure 1, an OCL constraint is used to describe, that the name of a component must be shorter than twenty characters. These constraints can be evaluated on instances of the meta model to check if these are indeed valid instances.

2.3 Profiles and Stereotypes

Instead of using meta models to define a modeling language, the UML can be adapted by profiles and stereotypes. Stereotypes can be used to add new attributes to existing UML elements. They can also define constraints that must be fulfilled by instances of the stereotyped meta elements. A profile contains a set of stereotypes and can be applied to a UML model. The following chapter explains the use of stereotypes for SysML models.

3 Modeling

There exist several suitable modeling languages and notations for the development of complex systems (e. g., for embedded automotive systems), which focus on different aspects or views. In this paper, we have a look into two different languages that are used within a particular development thread going from the system engineering (including requirements as well as the HW and SW structure) down to software engineering. The application domain considered here is the development of automotive embedded systems. SysML is used to analyze and design the overall system architecture and the AUTOSAR framework is used to specify the SW architecture in more detail. AUTOSAR is a DSL, which focuses on the development of Electronic Control Systems. This is only one aspect of the overall system engineering process supported by the SysML modeling language.

3.1 SysML

A widely-used language for system engineering is SysML (System Modeling Language), which is currently available in version 1.1 (see [Sys08]). SysML supports the desing and analysis of complex systems including HW, SW, processes and more. SysML reuses a subset of the UML and adds some additional parts (e. g., the Requirement- and Parametric-Diagram) to facilitate the engineering process by providing several improvements compared to the UML concerning system engineering. The UML itself tends to be more software centric while the topic of SysML is clearly set to the analysis and design of complex systems (not only SW).

In SysML, system blocks are used to specify the structure of the system[2]. For this purpose the UML element *Class* is extended by the stereotype *block*. A block describes a logical or physical part of the system (e.g., SW or HW). Multiple of these blocks can be used for representing the structure of a system. An example for the additional capabilities of SysML is the possibility to model the flow of objects between different system elements (which are specified in form of

[2] A block describes a part of the structure of a interconnected system.

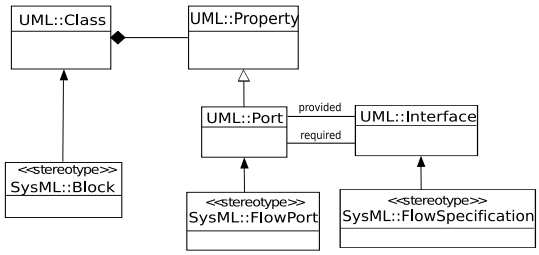


Figure 3: Extract of the SysML metamodel

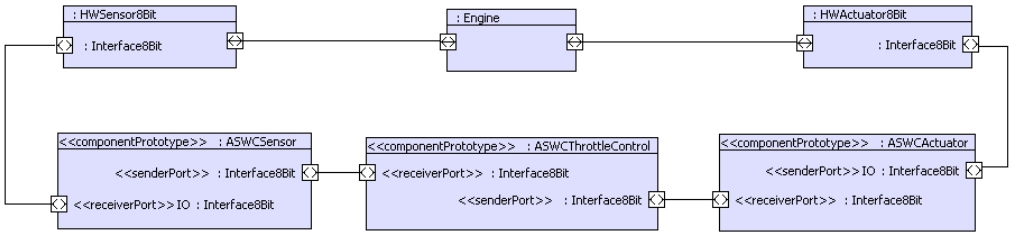


Figure 4: Application example of an SysML model created in Topcased

SysML blocks) by the usage of *flow ports*. A *flow port* is a stereotype for the UML element *Port* and allows the modeling of an object flow between SysML blocks. For the specification of objects and data, which flow over a flow port the stereotype *flow specification* is applied to the UML element *Interface* in SysML. The SysML meta model describing blocks, flow ports and flow specifications is shown in Figure 3.

When analyzing and designing automotive systems, the HW/SW-structure can be described using SysML blocks, ports (e. g., flow ports) and appropriate interfaces (e. g., flow specifications). In this paper, we use a simplified version of the structural constituents taken from an application example of an engine-fuel control system consisting of actuators and sensors for the throttle position and the control software. The control software evaluates the sensor values, computes appropriate throttle position values and sends them to the actuator of the throttle.

The system structure including HW and SW parts is modeled using the tool TOPCASED[3] and the resulting SysML model of the engine fuel control system is shown in Figure 4. The example consists of six different types of blocks, three of them represent hardware parts like the engine, a HW actuator and a HW sensor for setting and measuring the throttle position of the engine. The HW sensor (*HWSensor8Bit*) is connected to a SW block (*ASWCSensor*), which reads in data from the HW (e. g., by using driver functionality) and sends these measured values to a SW block, which realizes the control functionality (*ASWCThrottleControl*) and computes an output signal. This output signal is send to a SW block (*HWActuator*), which realizes the access to the HW actuator, which is represented through the block *HWActuator8Bit*. The *HWActuator* interacts with the representation of the physical engine.

When such a system is designed several restrictions have to be considered concerning the used HW sensor blocks in combination with the software blocks. A typical restriction is that a connector could only connect ports, which implement the same interface. In the shown example, e.g., the flow ports of the blocks *ASWCSensor* and *HWSensor8Bit* over which these two blocks are connected, have to implement the same interface. Such a constraint can be expressed in form of the following OCL constraint for the type connector:

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

```
context Connector inv:
self.end->forAll(e:self.end->get(0).role.type == e.role.type)
```

Only three of the blocks (*ASWCThrottleControl*, *ASWCSensor* and *ASWCActuator*) described above are relevant for the SW architecture. We use stereotypes to be able to identify the definition and the usage of SW blocks like described in Section 2. In our implementation, stereotypes are defined for identifying, e.g., the definition of SW blocks (*atomicSoftwareComponent*) as well as for the usage of the defined SW blocks (*componentPrototype*) like shown in Figure 4. In the following section, we show how these constituents can be represented in a DSL, which focuses on the development of automotive software systems.

3.2 AUTOSAR

AUTOSAR (Automotive Open System ARchitecture) is a framework for the development of complex electronic automotive systems. The purpose of AUTOSAR is to improve the development process for ECUs (Electronic Control Units) and whole systems by defining standards for the system and software architecture. The AUTOSAR standard[4] defines a meta model, which describes a DSL for the development of automotive embedded systems. This meta model is described in [AUT07] in form of an UML profile. We use a stand-alone meta model for AUTOSAR, which is realized accordingly.

As defined by the AUTOSAR meta model the software architecture is build of Components (e. g., *AtomicSoftwareComponents* (ASWC)). These ASWC are derived from the type *ComponentType* and can communicate using two different categories of ports, required and provided ports (represented through *RPortPrototype* and *PPortPrototype*). Both types are derived from the same abstract class *PortPrototype*. An *RPortPrototype* only uses data or events, which are provided by other ports of type *PPortPrototype*. A port of type *RPortPrototype* or *PPortPrototype* can implement an interface of type *PortInterface*. This *PortInterface* is refined by *ClientServerInterface* and *SenderReceiverInterface*. The AUTOSAR meta model for SWCs and for the different types of ports is shown in Figure 5.

The SW blocks (*ASWCSensor*, *ASWCActuator* and *ASWCThrottleControl*) defined within the SysML model described above can also be specified within an AUTOSAR model. The blocks shown in Figure 4 can also be described using ASWCs, ports and interfaces, which are defined within the extract of the AUTOSAR meta model shown in Figure 5. Figure 6 shows the same SWCs modeled with the tool SystemDesk[5]

In case of the SysML example, the SW blocks, ports and connectors can be described directly within such an AUTOSAR model in form of ASWCs. In case of the blocks describing HW, such a mapping is currently not realized in our system. Therefore, the blocks, ports and connectors concerning HW in the SysML model do not exist in the AUTOSAR model. Also the connectors, which exist in the SysML model between the ports of a SW block and a HW block are not transformed to AUTOSAR. In the next project phases, also these HW constituents will be considered. The AUTOSAR model described in this specific case is a subset of the elements existing within the SysML model. One possibility to derive the AUTOSAR model shown in Figure 6 from the SysML model is to manually transfer the relevant parts. Such a manual activity is expensive and error-prone. Furthermore, to manually keep both models consistent when changes occur is even more difficult. Another possibility is to use techniques, which allow to automatically derive one model from another or even to synchronize two existing models. Subsequently we describe a technique, which supports model transformation as well as model synchronization.

4 Model Synchronization

Model transformation systems can be used to transform one model into another model using a set of transformation rules. These rules are defined on the meta models of the source and

[4] Information can be found at: http://www.autosar.org

[5] http://www.dspace.com/ww/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm

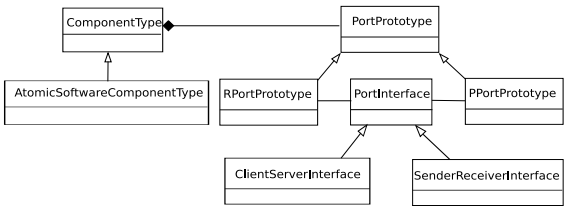


Figure 5: Extract of the AUTOSAR meta model

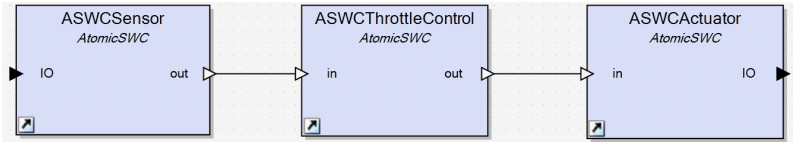


Figure 6: AUTOSAR model derived from the SysML model

target models of the transformation. They describe, what pattern of target model elements has to be created if the source model contains a certain element pattern. The model transformation system analyzes the source model and creates a target model according to the transformation rules. Examples of model transformation systems are ATL[JABK08], VIATRA[CHM+02] and systems based on *Triple Graph Grammars*[GW09] (TGG), as well as the QVT standard[OMG], which describes a language for expressing model transformation rules but does not provide an implementation.

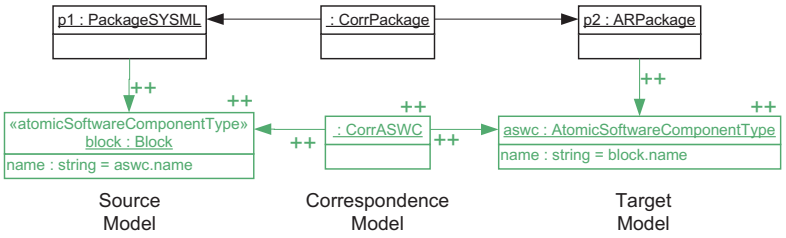


Figure 7: TGG rule for the transformation of a block to an atomic software component

Besides meta models also graph grammars can be used to describe a language. A graph grammar contains a set of graph grammar rules and a start graph, which defines the basic elements that must be contained in a model. The rules of a graph grammar consist of a *Left-Hand-Side* (LHS) and a *Right-Hand-Side* (RHS). The LHS defines the context, in which the rule can be applied, i.e. the elements that must already exist in the model. If the LHS of the rule can be matched to existing elements, these elements are replaced by the elements of the RHS. If the elements of the LHS are also part of the RHS, they are usually left untouched. Elements that occur only on the RHS are added to the model, elements that occur only on the LHS are deleted. Using a graph grammar, a model can be built by applying the rules successively, starting with the start graph.

Triple Graph Grammars combine three conventional graph grammars to describe the correspondence relationships between elements of two types of models. Two graph grammars describe the two models and a third grammar describes a correspondence model. Figure 7 shows a TGG rule for the transformation of a SysML block to an ASWC in AUTOSAR. This illustration also combines the LHS and RHS of the rule. The black elements belong to the LHS and the RHS of the rule. The elements marked with **++** (and printed green) belong only to the RHS and are created when the rule is applied. Rules that delete elements are not used in the context of model transformation with TGGs (cf. [Sch94]). The correspondence model is used to explicitly store correspondence

relationships between corresponding source and target elements. It allows to quickly find the target model elements corresponding to a given source model element.

Furthermore, TGGs allow bidirectional model transformations. The target model can be created from a source model (forward transformation) and vice versa (backward transformation). Besides model transformation, where a new target model is created, model synchronization is also supported. This means, that an existing target model is modified to make it consistent with a source model again. Modifications made to the target model are now retained and not overwritten[6]. Another advantage of synchronization is, that it can be performed much faster than a complete model transformation, especially if the models are large and only small modifications have to be synchronized.

The TGG rules are declarative and cannot be executed right away. Instead, operational transformation rules are derived for the forward and backward transformation. These operational rules are executed by a transformation engine to perform the model transformation. More information can be found in [GW09].

5 Architecture

In the industrial project, an architecture has been established, which integrates the tools TOPCASED and SystemDesk using the Eclipse platform. The tools are incorporated within the Eclipse platform in a way that both types of models (SysML and AUTOSAR models) exist in form of an EMF representation. Based on this EMF representation, the transformation and synchronization techniques described in Section 4 are realized. Subsequently, we describe this architecture in more detail.

5.1 Overall Architecture

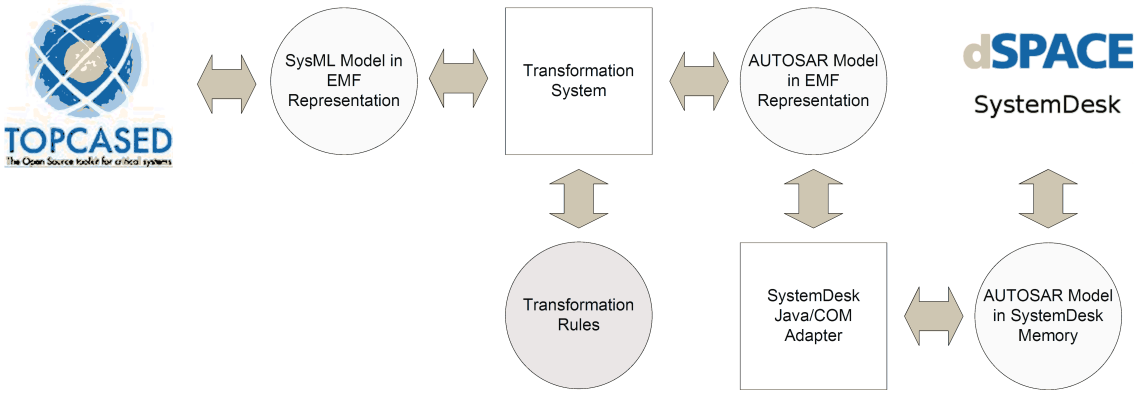


Figure 8: Overall system architecture

Figure 8 shows the overall architecture of the system. The core is the model transformation system, which implements the transformation and synchronization functionality. For this purpose, the transformation system needs to have access to the existing AUTOSAR and SysML models in EMF representation. In our architecture, this is possible in two different ways, in a file-based manner by reading and writing XML files, or alternatively by accessing the models directly in the modeling tools' memories. TOPCASED already uses EMF as its underlying modeling infrastructure and access to these EMF models from the transformation system can be realized without great effort. Accessing SystemDesk's models is more difficult because the technology gap between Eclipse/EMF and SystemDesk must be bridged. For this purpose, we developed a dedicated

[6] Unless they collide with changes in the source model. In this case, the changes of the target model are overwritten.

adapter, that reads an AUTOSAR model in EMF representation and writes it to SystemDesk, and vice versa. While the transformation system creates and modifies an AUTOSAR model (e.g., by a transformation or synchronization) in EMF representation, the SystemDesk Adapter takes care of reading and writing the model to and from SystemDesk. The model transformation and synchronization functionality is realized within the transformation system, which has access to the EMF models.

5.2 SystemDesk Adapter

SystemDesk provides an API, which is implemented in form of a Component Object Model (COM) and allows to access objects within SystemDesk from any COM-compatible application. Using this API also the AUTOSAR models within SystemDesk can be read and written. Based on the provided API, we have implemented an adapter, which is able to translate the model elements of the AUTOSAR Model in SystemDesk to EMF conformant model elements and vice versa. This adapter is used in our overall architecture to realize the bridge between SystemDesk and Eclipse/EMF like shown in Figure 8. In our current implementation, the adapter only partially supports the update and synchronization of AUTOSAR models in SystemDesk due to technical challenges. In the next version, we will implement the use of Unique Identifiers (UIDs), which are provided by the tool SystemDesk to fully support the update and synchronization of AUTOSAR models in SystemDesk.

5.3 Rules

The core transformation system of our architecture uses TGG rules like described in section 4 to realize the transformation and synchronization of SysML and AUTOSAR models. An example is the rule shown in Figure 7. This TGG rule transforms a SysML Block with the appropriate stereotype (*atomicSoftwareComponentType*) to an AUTOSAR ASWC. Such rules for the transformation and synchronization of the defined ports, interfaces and other constituents described in the meta model cutouts for SysML and AUTOSAR shown in Section 3 (and for the opposite direction) are also used within the transformation system.

6 Usage Scenarios

The described architecture supports several scenarios where, e.g., an initial AUTOSAR model is derived from an existing SysML model.

Additionally, the described architecture allows the synchronization of existing models by updating only changed model elements in the target model, without overwriting the whole model each time changes occur. Such a synchronization can be executed in both directions. Following, we describe different usage scenarios, in which the shown architecture allows an enhanced development process using model transformation and synchronization techniques.

6.1 Transformation from SysML to AUTOSAR

After the SysML model has been constructed, it needs to be transformed into an AUTOSAR model to get from the system design to an initial model for the software design. Design decisions concerning the software, which were defined in the SysML model have to be taken over to the AUTOSAR model. With the presented system such an initial AUTOSAR model can be automatically derived by a forward transformation. The automatic transformation is much faster than a manual transformation and there is less risk of introducing errors into the AUTOSAR model. A transformation in the other direction is also possible (backward transformation).

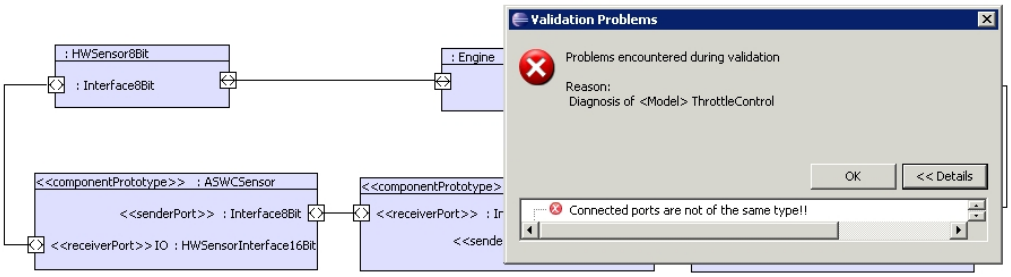


Figure 9: Screenshot of the OCL validation dialog in TOPCASED

6.2 Repeated Forward Synchronization from SysML to AUTOSAR

After the AUTOSAR model has been derived from the SysML model, modifications can still be made to the SysML model. These modifications have to be transferred to the AUTOSAR model, too. While the AUTOSAR model already exists, a complete retransformation is unnecessary. Therefore, only the modifications are synchronized. Furthermore, the AUTOSAR model might also have been modified, e.g., by changing the type of the IO port of the ASWC *ASWCSensor* shown in Figure 6. A complete retransformation would discard these modifications. Basically, our system supports such a synchronization. But due to restrictions of the current implementation of the adapter (compare Section 5) the models in SystemDesk are always overwritten. In the future, we will extend the adapter to allow the modification of the existing SystemDesk model.

6.3 Backward Synchronization from AUTOSAR to SysML

However, modifications may also be made to the AUTOSAR model in order to adjust the structure during refinement of the software architecture, e.g., to reuse an already existing component. Therefore, modifications also have to be propagated back to the SysML model. While most model transformation approaches are only permit unidirectional transformations, TGGs are bidirectional. So most changes are preserved in the SysML model.

How such a propagation of changes within the shown architecture using bidirectional transformation techniques supports the development process is demonstrated by the following scenario. When the type of the IO port of the ASWC *ASCWSensor* from Figure 6 is changed within the AUTOSAR model the TGG rules are triggered within the transformation system and the corresponding SysML IO port shown in Figure 4 is updated accordingly without overwriting the whole SysML model. When the SysML model is updated, the OCL constraint described in Section 3.1 is violated and an error message is automatically generated in TOPCASED (see Figure 9) that a SysML connector is connected to ports, which have a different type.

6.4 Iterative and Flexible Processes

The usage scenarios outlined in sections 6.1, 6.2 and 6.3 demonstrate that our approach can handle changes occurring in either model in any order. Therefore, the approach enables not only a strict sequential ordering, where first the SysML model is specified and thereafter the AUTOSAR model is derived from it (section 6.1). It also allows, that changes in the SysML model are propagated to the already existing AUTOSAR model (section 6.2) and that necessary changes in the AUTOSAR model are also accordingly adjusted in the SysML model (see 6.3). Therefore, instead of a rigid sequential process, also iterative and more flexible processes can be supported. Parallel development in the different phases is supported, changes in the different models can be synchronized and potential conflicts can be detected. Later changes of the AUTOSAR model will be reflected back to the SysML model after a synchronization. Such changes in an AUTOSAR model can lead to the violation of constraints existing in SysML model like described beforehand.

7 Conclusion & Future Work

SysML models employed early on and AUTOSAR models employed later in the process can be kept consistent using presented approach thanks to the use of model synchronization techniques. It has been outlined, that usage scenarios are feasible when employing our approach, and that additional flexibility concerning the process and in particular iterative development can be achieved.

As future work, we plan to further extend the coverage and also address other development artifacts. We also want to investigate how multiple models connected via model synchronization can be efficiently managed.

Acknowledgement

We thank the dSPACE GmbH for their support to develop the presented results and Oliver Niggemann, Joachim Stroop, Dirk Stichling and Petra Nawratil for their support in setting up and running the project.

References

[AUT07] AUTOSAR. *UML Profile for AUTOSAR*, January 2007. AUTOSAR GbR.

[BGN+04] Sven Burmester, Holger Giese, Jörg Niere, Matthias Tichy, Jörg P. Wadsack, Robert Wagner, Lothar Wendehals, and Albert Zündorf. Tool Integration at the Meta-Model Level within the FUJABA Tool Suite. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(3):203–218, August 2004.

[BHLW07] Simon M. Becker, Sebastian Herold, Sebastian Lohmann, and Bernhard Westfechtel. A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. *Software and System Modeling*, 6(3):287–315, 2007.

[CHM+02] György Csertán, Gábor Huszerl, István Majzik, Zsigmond Pap, András Pataricza, and Daniel Varró. VIATRA: Visual Automated Transformations for Formal Verification and Validation of UML Models. In Julian Richardson, Wolfgang Emmerich, and Dave Wile, editors, *Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering*, pages 267–270, Edinburgh, UK, 23 September 2002. IEEE Press.

[GW09] Holger Giese and Robert Wagner. From model transformation to incremental bidirectional model synchronization. *Software and Systems Modeling*, 8(1), 1 February 2009.

[JABK08] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, 2008.

[JPB08] T. Johnson, C. Paredis, and R. Burkhart. Integrating Models and Simulations of Continuous Dynamics into SysML. 2008.

[OMG] OMG. *MOF QVT Final Adopted Specification, OMG Document ptc/05-11-01*. http://www.omg.org/.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Proc. of the* 20^th *International Workshop on Graph-Theoretic Concepts in Computer Science*, Herrschin, Germany, 1994. Spinger Verlag.

[Sys08] Systems Modeling Language v. 1.1, November 2008.

[Win07] Hans Windpassinger. Modellierungssprache für die Kfz-Software Entwicklung. Elektronik Praxis, 2007. http://www.elektronikpraxis.vogel.de/themen /embeddedsoftwareengineering/analyseentwurf/articles/95528/.