Model Synchronization at Work: Keeping SysML and AUTOSAR Models Consistent

Holger Giese, Stephan Hildebrandt, and Stefan Neumann

Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
{Holger.Giese,Stephan.Hildebrandt,Stefan.Neumann}@hpi.uni-potsdam.de

Abstract. During the overall development of complex engineering systems different modeling notations are employed. For example, in the domain of automotive systems system engineering models are employed quite early to capture the requirements and basic structuring of the entire system, while software engineering models are used later on to describe the concrete software architecture. Each model helps in addressing the specific design issue with appropriate notations and at a suitable level of abstraction. However, when we step forward from system design to the software design, the engineers have to ensure that all decisions captured in the system design model are correctly transferred to the software engineering model. Even worse, when changes occur later on in either model, today the consistency has to be reestablished in a cumbersome manual step. In this paper, we present how model synchronization and consistency rules can be applied to automate this task and ensure that the different models are kept consistent. We also introduce a general approach for model synchronization. Besides synchronization, the approach consists of tool adapters as well as consistency rules covering the overlap between the synchronized parts of a model and the rest. We present the model synchronization algorithm based on triple graph grammars in detail and further exemplify the general approach by means of a model synchronization solution between system engineering models in SysML and software engineering models in AUTOSAR which has been developed for an industrial partner.

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) [1] models are employed quite early to capture the requirements and basic structuring of the whole system by system engineers, while AUTOSAR (Automotive Open System ARchitecture)[1] models are used later in the software development process to describe the

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

G. Engels et al. (Eds.): Nagl Festschrift, LNCS 5765, pp. 555–579, 2010.

concrete software architecture and its deployment. Using these different models helps in addressing each specific design issue with an appropriate notation and at a suitable level of abstraction.

For example, when going from the system design with SysML to the software design stage with AUTOSAR, today the engineers have to ensure manually that all relevant 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 (MDE) with its support for model transformation and model consistency checking is a promising direction to approach the sketched model consistency problems, which result as the models describe the system under development from different viewpoints and on different levels of abstraction capturing only partially overlapping information (cf. [35]).

Triple graph grammars (TGGs) are a formalism to declaratively describe correspondence relationships between two types of models. They were introduced in [29] and are one option to specify the required model transformations using a declarative transformation specification. In several contexts, different variants of TGGs have already been employed for model synchronization such as the integration of SysML models with Modelica simulation models [22], keeping models from the domain of chemical engineering consistent [4] and transformations from SDL models to UML models and vice versa [7].

In this paper, we report about our approach to tackle the outlined model synchronization problem. Built on top of techniques from model-driven engineering such as meta-models, consistency rules, and bidirectional model transformations resp. model synchronizations specified by TGGs, a general architecture has been developed, which allows to automate the task of keeping models consistent. We only require that the TGG rule set is deterministic and that only one of the models is changed at a time. Like described in [13], in many cases managing and tolerating inconsistencies (e.g., by allowing to manipulate different models concurrently) instead of directly removing them is desirable. In the case of the automotive domain, consistency plays a crucial role, e.g., caused by the reason that inconsistencies between previously defined requirements and the later implementation can lead to catastrophic failures. Thus, a more rigorous handling of inconsistencies like requested in [13] is adequate for our application example.

In a project with the automotive industry, we could demonstrate that our approach can be employed for model synchronization between the SysML tool TOPCASED and the AUTOSAR tool SystemDesk. Firstly, the model transformation derived from TGGs permits to automatically generate the initial AUTOSAR model from the SysML model. Secondly, consistency between both models in case of changes in one of them can be maintained by a TGG-based model

synchronization system[19]. Thus, we can synchronize both models such that changes within one are automatically transferred to the other.[2]

By making manual transformation and synchronization steps obsolete, the automatic synchronization of models reduces costs and time. This applies to the initial transitions, for example, from the SysML model to the AUTOSAR model, as well as the re-establishment of consistency in case of changes in one model. In addition, such automated synchronization steps are less error prone than manual steps as employed today. They further enable employing iterative and more flexible development processes as the costs for iterations or changes are dramatically reduced as long as parallel changes do not occur.

The structure of the paper is as follows: The current state of the art and its limitations compared to our outlined new approach are discussed in Section 2. The considered application example for our approach for synchronizing SysML and AUTOSAR models is introduced in Section 3. The new approach, its architecture, and its components are first sketched in Section 4. Then the constituent parts are presented in detail. The technique for model synchronization is explained in Section 5. The tool adapter and the techniques for consistency checks follow in Section 6 and Section 7. At the end, we discuss the suitability of our approach by looking into several typical usage scenarios, such as the initial transfer of information or change propagation, and close the paper with a final conclusion and an outlook on planned future work.

2 State of the Art

In this section, we discuss related approaches for model transformation and synchronization, and related work in the context of Model-Driven Engineering, that make use of model transformations.

2.1 Model Synchronization

An overview of model transformation and synchronization systems can be found in [9]. The paper categorizes existing approaches and briefly explains them.

As outlined in the introduction, MDE requires a bidirectional solution, which preserves model contents when synchronizing as much as possible. However, many available model transformation approaches only support classical one-way batch-oriented transformations [16]. The QVT implementations [6] and [15], and some graph transformation-based approaches such as VIATRA [33,5], the GREAT model transformation system [34], AGG [12], the core PROGRES tool [28] or the core FUJABA tool [32] are only unidirectional but partly incremental. The available relational QVT implementations [21,31] as well as BOTL [25] are bidirectional but only support a batch-oriented processing and, thus, fail to be scalable.

[2] It has to be noted that the required restriction concerning no parallel changes in the models does not result in any additional limitations in the considered application domain, as the processes currently try to exclude changes at all to avoid the cumbersome manual step to reestablish consistency.

Other existing TGG-based approaches also do not provide a comparable automatic and computational incremental solution (for a detailed discussion see [19]): The TGG transformation algorithm based on the PROGRES environment [4] is also incremental, but operates interactively, and is therefore inappropriate for the transformation of large models. In the incremental TGG transformation approach supported by ATOM^3 [20], updates are triggered by user actions like creating, editing or deleting elements and the specification of updates for all possible user actions is required. Thus, the consistency of the approach is difficult to guarantee and initial complete model transformations are not supported. Another TGG realization based on [32] is MOFLON [10]. It focuses on model integration and transformation for the MOF 2.0 standard [26] rather than incremental model synchronization.

Incremental model synchronization can also be seen as an inconsistency resolution problem. [11] describes an incremental solution for the related problem of inconsistency checking. The presented system allows a check to be made to quickly determine whether a modification has caused inconsistencies, and proposes solutions to the user. For a more detailed discussion of such solutions for model synchronization we refer to [19].

2.2 Model Integration

Model-Driven Engineering is a development paradigm, where models are the primary development artifacts. In [23] this idea is described. Models are used to describe the system under development from different viewpoints and on different levels of abstraction. During the development process, models are refined and ultimately source code is generated from these models. 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. In practice an additional challenge is that different kinds of models are normally supported by different tools and these tools use diverse technologies for representing these models. So at first models need to be accessed in an appropriate way to be able to apply model transformation techniques.

In the MATE project [30] an adapter has been realized to access MATLAB/Simulink models in such a way that model transformation rules can be applied, e.g., for checking guidelines while model consistency like in case of model synchronization is not the main focus.

In the ModelBus project [2] a framework has been developed, which is able to integrate different model-based development tools into a service-oriented middleware. The purpose of the ModelBus project is to provide a framework allowing several tools to be connected within a single environment. Model transformation and synchronization techniques can be potentially applied using this framework. When access to the different models is provided in an adequate form, the integration of different models using model transformation and synchronization techniques can be realized.

An approach for the integration of SysML models with Modelica simulation models has been described in [22]. The approach is also based on triple graph

grammars but uses VIATRA [8] to implement the transformation. In contrast to the system presented in this paper, synchronization of models is not supported.

3 Application Example

We have evaluated the approach presented in this work within a project organized with our industrial partner, dSPACE GmbH. dSPACE provides, besides other products, several tools for the development of embedded systems, especially for the automotive domain. Within this project we used two different modeling languages commonly used for the development of automotive embedded systems, namely the *System Modeling Language* (SysML) and the *AUTomotive Open System ARchitecture* (AUTOSAR). We used the tool SystemDesk, a professional tool for the development of complex automotive embedded systems according to the AUTOSAR standard, and TOPCASED, an open-source toolkit for supporting the modeling of SysML models. Both modeling languages (AUTOSAR and SysML) are subsequently described in more detail.

3.1 SysML

A widely used language for system engineering is SysML (System Modeling Language), which is currently available in version 1.1 (see [1]). SysML supports the design and analysis of complex systems including hardware (HW), software (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 UML concerning system engineering. 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 our application example the existing SysML profile provided by the OMG has been used, which utilizes the generic extension mechanisms of UML to customize UML elements using the concept of stereotypes. Such stereotypes can be applied to UML elements[3] and extend as well as define constraints on these elements. For expressing constraints also the *Object Constraint Language* (OCL) can be used, which not only allows the description of structural properties, but also the specification of additional constraints on the values and/or types of attributes and so on. For more information about OCL see [27]. Instances of the stereotyped UML elements must fulfill the properties defined by the applied stereotypes. The SysML profile (like any other UML profile) contains a set of stereotypes, which are applied to a UML model. In the following we explain relevant SysML stereotypes for a small application example.

In SysML, system blocks are used to specify the structure of the system[4]. For this purpose the UML meta model element *Class* is extended by the stereotype *<<block>>*. A block describes a logical or physical part of the system

[3] Elements of the UML meta model.

[4] A block describes a part of the structure of an interconnected 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 SysML blocks) by using *<<flow ports>>*. *<<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. Ports can be connected via connectors provided by the UML meta model. The elements required to connect different ports (the UML *Connector* and *ConnectorEnd*) as well as a part of the SysML meta model describing blocks, flow ports and flow specifications are shown in Figure 1 in a simplified way.

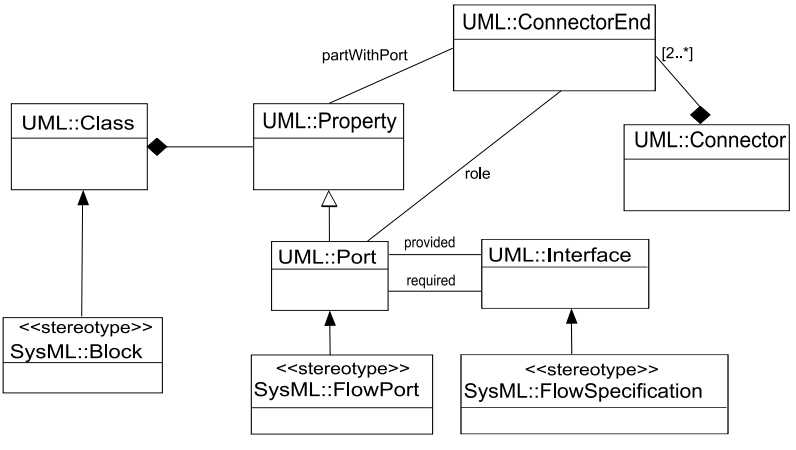

Fig. 1. Extract of the SysML metamodel

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 has been modeled using the tool TOPCASED[5] and the resulting SysML model of the engine fuel control system is shown in Figure 2. 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.

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

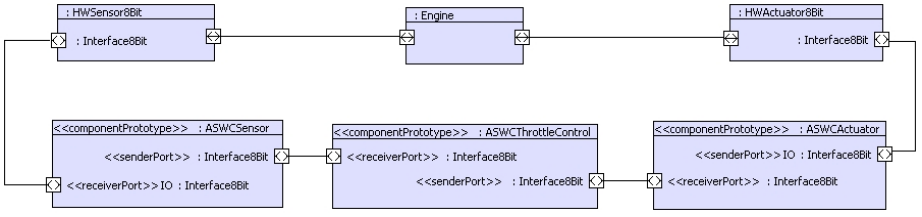

Fig. 2. Application example of an SysML model created in Topcased

The HW sensor (*HWSensor8Bit*) is connected to a SW block (*ASWCSensor*), which reads 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 can 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 the form of the following OCL constraint for the type connector:

```
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. 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 2. In the following section, we show how these constituents can be represented in AUTOSAR.

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 defines a meta model, which describes a DSL for the development of automotive embedded systems. The part of the meta model relevant for the present work is described in [3] in the form of a UML profile. We use a stand-alone meta model for AUTOSAR, which is realized accordingly.

As defined by the AUTOSAR meta model (an excerpt is shown in Figure 3), the software architecture is built from 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 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 2 can also be described using ASWCs, ports and interfaces, which are defined within the extract of the AUTOSAR meta model shown in Figure 3. Figure 4 shows the same SWCs modeled with the tool SystemDesk[6].

In case of the SysML example, the SW blocks, ports and connectors can be described directly within such an AUTOSAR model in the form of ASWCs. In case of the blocks describing HW, such a mapping is not desired by our industrial partner. HW components in AUTOSAR are represented on completely different levels of abstraction than in SysML. Therefore, the blocks, ports and connectors concerning HW in the SysML model have not been reflected in the AUTOSAR model in our application example. Also the connectors, which exist in the SysML model between the ports of a SW block and a HW block have not been transformed to AUTOSAR.

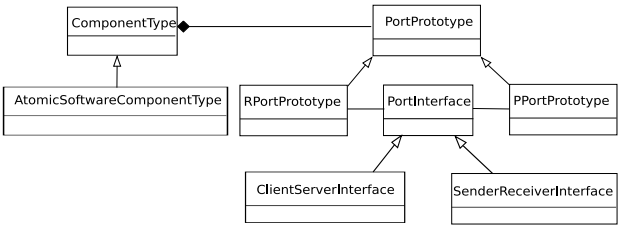

Fig. 3. Extract of the AUTOSAR meta model

3.3 Common Constituents

The elements in Figure 2 are tagged with stereotypes, e.g., <<*senderPort*>> and <<*receiverPort*>>. This is necessary because there are different types of ports in AUTOSAR but only one type in SysML. Another example are software components. AUTOSAR supports atomic and composite software components.

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

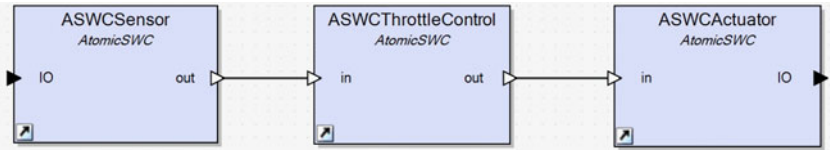

Fig. 4. AUTOSAR model derived from the SysML model

Both are represented as blocks in SysML. To distinguish both types, the stereotypes <<*atomicSoftwareComponentType*>> and <<*compositionType*>> have to be used. These stereotypes have been defined in a small profile for SysML, which we created for this purpose. In the remainder of this work we describe how to support consistency of such semantically identical elements in different models using transformation and synchronization techniques.

4 Approach

4.1 General Architecture

The generic architecture to integrate model transformation and synchronization with existing modeling tools is shown in Figure 5. The transformation system only supports EMF-compatible (Eclipse Modeling Framework [7]) models. Therefore, the source and target models need to be provided in that format. This is done by tool adapters, which translate the models from and to EMF if necessary. If the modeling tool itself is based on EMF, such an adapter can be realized easily. If the modeling tool is not based on EMF, the tool adapter has to provide an EMF representation on the fly that the transformation system can modify, and has to synchronize it with the actual model in the modeling tool. More information on how such an adapter could be realized can be found in Section 6.

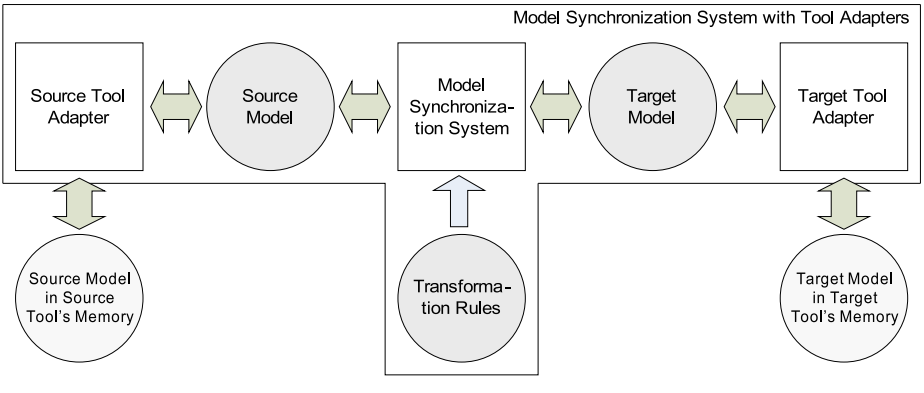

Fig. 5. General Architecture

[7] http://www.eclipse.org/modeling/emf/

4.2 Architecture Example

In the project realized with our industrial partners, we have established an architecture like described above, which integrates the tools TOPCASED and SystemDesk. For each tool an adapter has been realized, that provides an EMF representation of the model to the transformation system. Figure 6 shows the concrete architecture developed in that project.

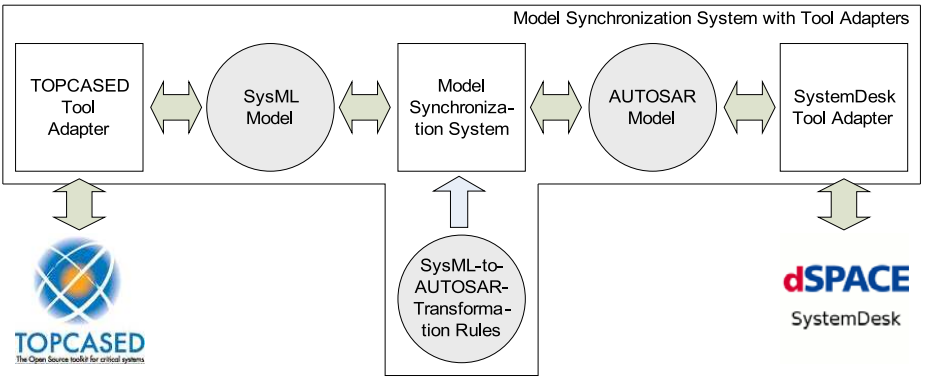

Fig. 6. Integration architecture for TOPCASED-SystemDesk integration

The transformation system, the tool adapters, and TOPCASED are based on the Eclipse platform. TOPCASED already uses EMF as its underlying modeling infrastructure. Therefore, access to these models from the transformation system can be realized without great effort. SystemDesk is a stand-alone Windows application that provides a COM interface for access to its models. Accessing SystemDesk's models is more difficult, because the technology gap between Eclipse/EMF and SystemDesk/COM must be bridged. For this purpose, we have developed a dedicated adapter. While the transformation system creates and modifies an AUTOSAR model in EMF representation, the SystemDesk adapter takes care of reading and writing the model to and from SystemDesk. For more details concerning the realization of the SystemDesk adapter see Section 6.

5 Model Synchronization System Based on Triple Graph Grammars

The model transformation system is based on triple graph grammars [29]. It is able to perform model transformations in both directions, i.e. create a target model from a source model and vice versa. Furthermore, it can synchronize both models after modifications have occurred. In the following sections, triple graph grammars are briefly introduced and the transformation and synchronization algorithm is explained.

5.1 Triple Graph Grammars

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 atomic software component in AUTOSAR. This illustration also combines the left-hand side (LHS) and right-hand side (RHS) of the rule. The black elements belong to the LHS and the RHS of the rule, i.e. they form the application context. 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. [29]). The correspondence model is used to explicitly store correspondence relationships between corresponding source and target elements. It allows the target model elements corresponding to a given source model element to be found quickly. The correspondence nodes are connected to each other and form a directed acyclic graph. Although, there is no link visible in Figure 7 between *CorrPackage* and *CorrASWC*, that link is created implicitly and is not shown in the TGG rule to ease modeling of TGG rules.

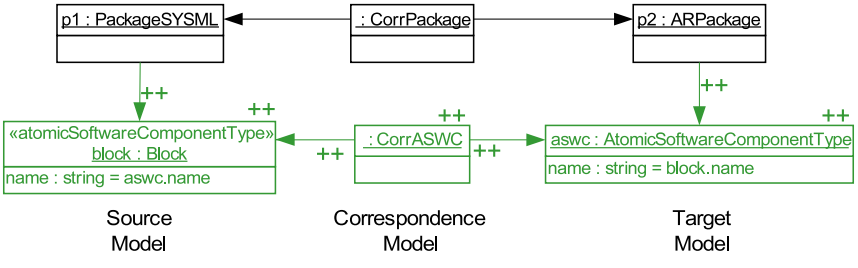

Fig. 7. TGG rule for the transformation of a block to an atomic software component

Fig. 8. Axiom of the triple graph grammar for the transformation from SysML to AUTOSAR

Like every other graph grammar, a triple graph grammar has a start graph which serves as the starting point of the transformation. In the context of TGGs it is called *axiom*. Figure 8 shows the axiom for the transformation from SysML to AUTOSAR. The whole grammar for this transformation contains many more rules, which are not shown here due to space limitations.

5.2 Model Transformation

TGG rules are declarative by nature. To execute them they can be either interpreted by a dedicated TGG interpreter like presented in [24], or other executable artifacts can be derived. In our case, Story Diagrams [14][8] are generated, which are executed by a Story Diagram interpreter [18].

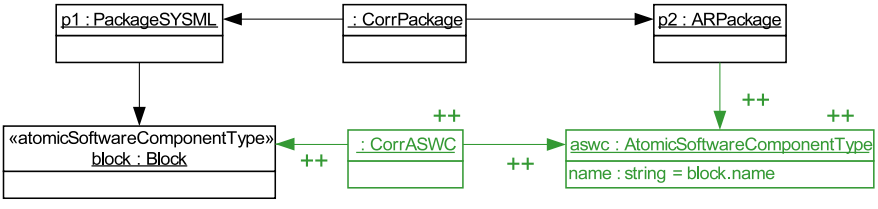

Fig. 9. Operational rule for the forward transformation of a block to an atomic software component

The transformation system consists of two major parts, the transformation engine and the operational transformation rules. The engine is independent from specific source or target models and invokes the rules. The transformation system supports both transformation directions, i.e. creating the right model from the left model and vice versa. Furthermore, synchronization of models is also supported. This is explained in Section 5.3. Therefore, separate operational rules for each direction are required. Conceptually, the operational rules are derived by adding all elements on the source model side to the rule's application context. Figure 9 shows the conceptual forward transformation rule derived from the rule in Figure 7. In practice, the operational rules have to do much more. Therefore, four separate operations are generated for each rule and for each direction, that perform

1. transformation of elements (*transformation()*)
2. synchronization of elements (*synchronization()*)
3. synchronization of attributes (*synchronizeAttributes()*, called by 2)
4. reconstruction of broken structures (*repairStructure()*, called by 2)

The overall operation principle of the transformation engine and the *transformation()* operation is explained in the following, the others are described in the next section.

The operation principle of the transformation engine is depicted in Figure 10. To execute a model transformation, the engine is started with the root elements of the source and target models as parameters, as well as the desired transformation direction (i.e. forward or backward). First, the axiom is executed to transform the root node (1). The correspondence node that was created by

[8] Story Diagrams combine UML activity diagrams with graph transformation rules to describe behavior.

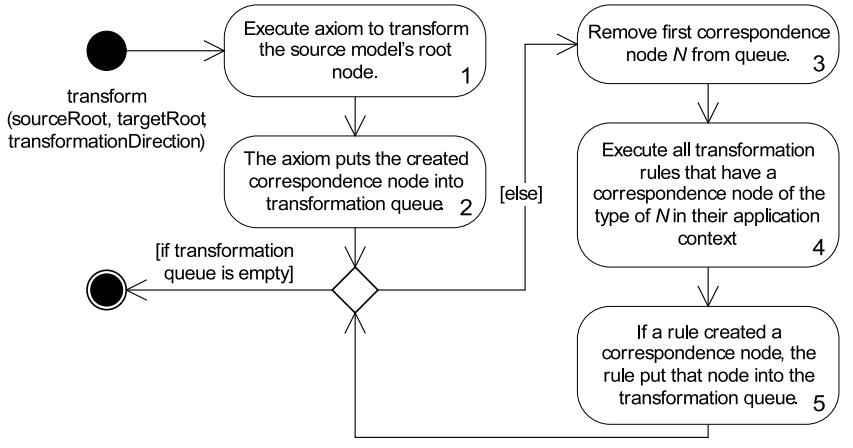

Fig. 10. Operation principle of the model transformation engine

the axiom is put into the transformation queue of the engine (2). This step is done by the axiom itself. The transformation queue contains the correspondence nodes that need to be processed. After that, the first correspondence node in the queue is removed (3) and all transformation rules are executed that expect such a correspondence node in their application contexts (4). If a rule has successfully transformed an element, the associated correspondence node is put into the transformation queue (5). Steps 3 to 5 are repeated until the transformation queue runs empty. Then the transformation is complete.

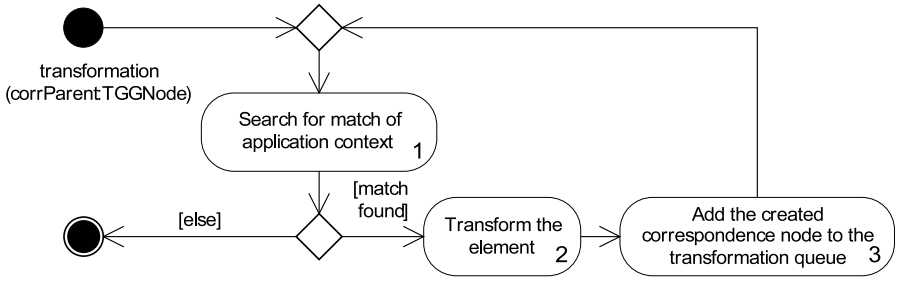

Fig. 11. Operation principle of the *transformation()* operation

Figure 11 shows the operation principle of the *transformation* operation[9]. The operation's parameter is the parent correspondence node (in case of the rule in Figure 9, the *CorrPackage* node). This node is the starting point of the search for other elements of the application context of the rule (*PackageSYSML*, *ARPackage* and *Block* in Figure 9). If these elements can be found and if they were

[9] Axioms are only special kinds of rules. Therefore, the operation principle of axioms is virtually the same.

not transformed before (1), the correspondence node and target elements are created according to the TGG rule (2). After that the newly created correspondence node is added to the transformation queue of the transformation engine (3). This process is repeated as long as new matches for the rule's application context can be found. After that, the control flow returns to the transformation engine as described above.

5.3 Model Synchronization

The model transformation system can also perform a synchronization between the models after an initial transformation. For efficiency, the system only visits those nodes that were actually modified. To detect modifications, an event listener is registered at each element of the source and target models. If an element is modified, its associated correspondence node is put into the transformation queue. There is an additional flag associated with each node in the queue[10], that marks whether the consistency of a correspondence node should be checked (flag is true), or new elements should be transformed (flag is false) when this correspondence node is processed by the engine. The notification listener always sets this flag to true. The transformation rules (see Section 5.2, step 3) always set it to false.

Therefore, the actual operation principle of the transformation engine is slightly more complicated than described in Figure 10. In step 4 of Figure 10, first the flag is checked. If the flag is false, the *transformation* operations are executed. If the flag is true, only the *synchronization* operation (see below) is executed that belongs to the rule that created the current correspondence node. The synchronization rule is responsible for re-establishing consistency between the associated source and target model elements. Furthermore, in case of synchronization, the axiom is not executed beforehand because the root model elements already exist.

The synchronization operations are used in an attempt to preserve existing target elements as far as possible. Many modifications, like moving elements, can be synchronized by adjusting some links in the target model instead of deleting and retransforming elements. This reduces the number of necessary modifications in the target model and allows even large models to be synchronized very quickly in many cases.

Figure 12 depicts the operation principle of a synchronization rule. First, the rule checks if the structure of the source, target and correspondence elements complies with the rule. If the structure is valid, the *synchronizeAttributes()* operation is called (1). This operation compares the attribute values, synchronizes them if necessary, and returns whether attribute synchronizations were performed. If attribute values were actually modified, the subsequent correspondence nodes are put into the queue with their flags set to true to check their consistency as well (2). If no attribute synchronizations were necessary, the subsequent correspondence nodes do not need to be checked. In any case, the current

[10] For simplicity, this has been omitted in Section 5.2.

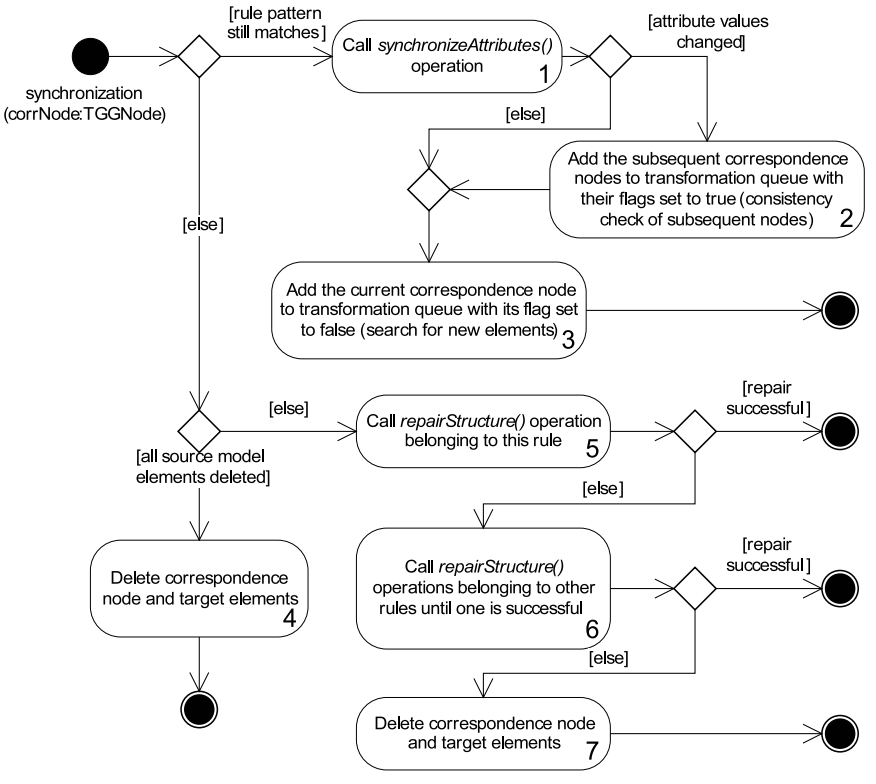

Fig. 12. Operation principle of the *synchronization()* operation

correspondence node is also added to the transformation queue with its flag set to false to search for new elements to transform (3).

In case the rule pattern does not match, a check is made to determine whether all source elements of the correspondence node were deleted from the model. Then the correspondence node and the target model elements are obsolete and can be deleted, as well (4). Note that this implies that all subsequent correspondence nodes and their target elements have to be deleted, too. If at least one of the source elements is still part of the model, a repair is attempted. First, the *repairStructure()* operation (see below) belonging to the same rule is executed (5). In case the repair is successful, the operation is finished. If it fails, the repair operations of the other rules are tried (6). As soon as one of them succeeds, the operation terminates. Note that the *repairStructure()* operation adds correspondence nodes to the transformation queue if necessary. Should all repair attempts fail, the correspondence and target elements are deleted (7). The modified source element cannot be synchronized with the available transformation rules.

The *repairStructure()* operation is the key to minimizing the number of write operations on the target model to re-establish consistency. Its general operation principle is shown in Figure 13. Prerequisites for a successful repair are that

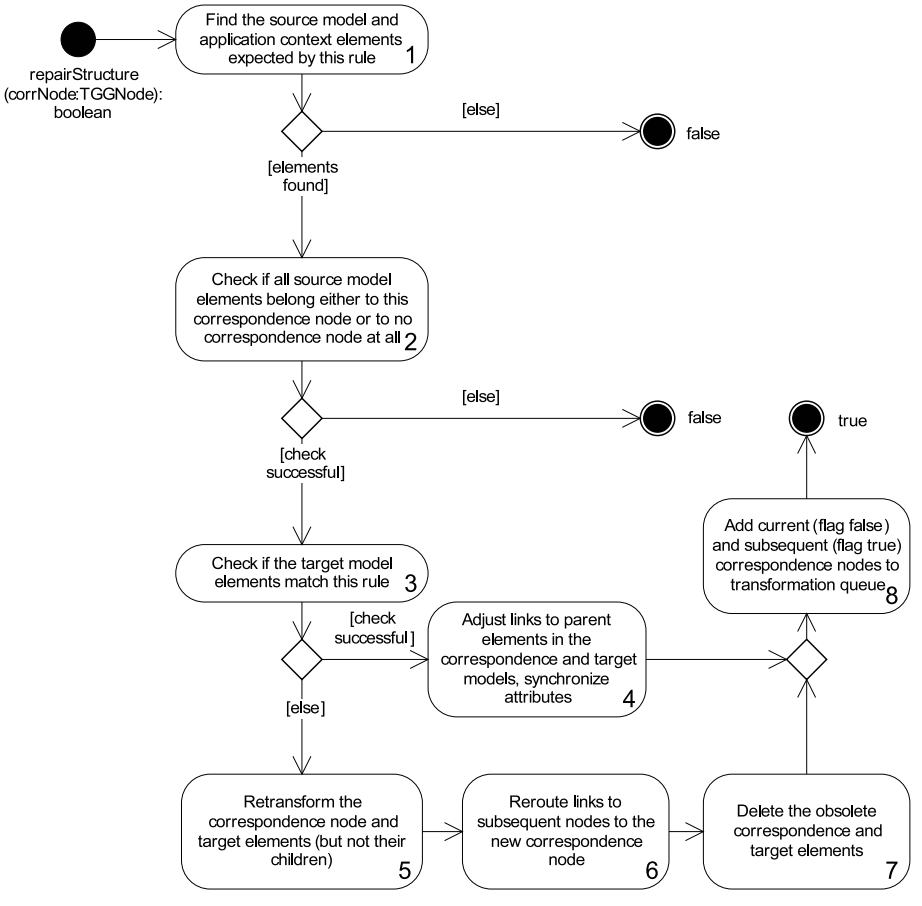

Fig. 13. Operation principle of the *repairStructure()* operation

the source model and application context elements exist (1) and that the source model elements are not connected to any other correspondence nodes (2).[11] If these conditions are met, the target element pattern is checked (3). If it complies with this rule, consistency can be re-established easily by re-adjusting the links to the elements of the application context of the rule (4). For the rule in Figure 7 this means to delete the existing link between *aswc* and the *ARPackage* it is currently linked to, and create a new link between *aswc* and the *ARPackage* that was matched in step 1. Moved elements can be synchronized mostly with this simple repair action.

More complex modifications can lead to the applicability of a different rule. In this case, the target elements do not meet the expected pattern because they were created by another rule. Then the correspondence node and target elements

[11] If connected, this means that these elements were already transformed by a different rule.

have to be created according to that rule (5). The links to subsequent correspondence nodes are rerouted to the newly created correspondence node (6), and the obsolete correspondence node and target elements are deleted (7). Finally, the current (or newly created) correspondence node is added to the transformation queue with its flag set to false to search for new elements to transform. Also the subsequent correspondence nodes are added to the transformation queue with their flags set to true to check their consistency. Depending on whether the structure could be repaired or not, true or false is returned by this operation.

This synchronization algorithm has some major advantages. First, the repair of broken structures minimizes the number of required write operations on the target model to synchronize modifications. While the previous version of our algorithm ([19]) would discard the target elements and retransform them, additional details that are not reflected in the source model would be discarded, as well. By preserving target elements, those details are preserved, too. This is important if the connected models have different levels of detail. Of course, also the performance is much higher if only links are changed and elements are not recreated. Second, the synchronization starts directly at those correspondence nodes where a modification took place. Those parts of the models that were not changed are not checked by the system. In addition, the synchronization stops checking correspondence nodes if a modification does not have any effects on them. For example, moving an element in the model usually does not influence its child elements. The algorithm first synchronizes the movement of the parent element. Then, its direct children are checked. If they have not been affected by the modification, the remaining indirect children are not checked. All in all, these optimizations make synchronization effort mostly independent from the overall model sizes. The size of the modifications (number and severity of modifications) has the largest impact on synchronization performance.

6 Tool Adapter

The tool adapter already mentioned in Section 4 is responsible for providing access to a modeling tool's in-memory model to the transformation system. Of course, the modeling tool has to provide a means to access its model, e.g. a COM interface. This allows models to be synchronized without indirection via files.

The transformation system works only on EMF-based models. Therefore, the tool adapter has to provide an EMF-based model. If the modeling tool is also based on EMF, like TOPCASED, such an adapter is quite simple. It only has to get the model's root element from the model editor and provide it to the transformation system (left side of Figure 6). The transformation and synchronization take place directly on the model.

If the modeling tool is not based on EMF or not even on the Eclipse platform, a tool adapter becomes much more complicated. It has to provide an EMF-based version of the model to the transformation system in addition to the model of the modeling tool. Before and after the transformation system reads or writes the EMF-based model, the adapter has to synchronize it with the model in the

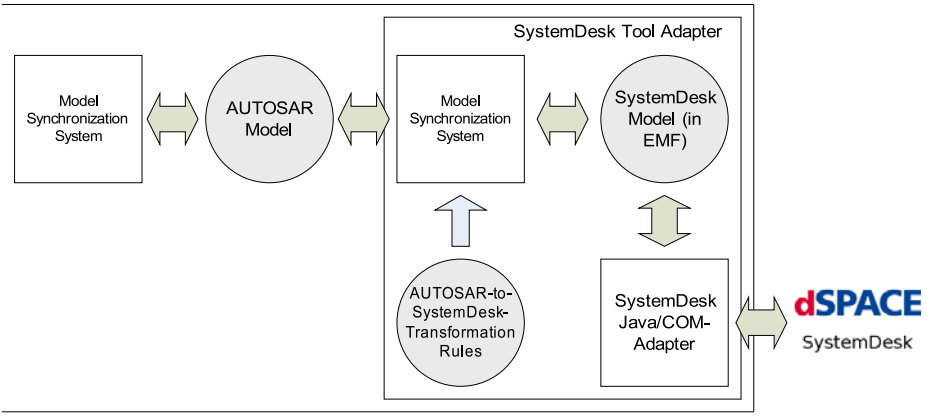

Fig. 14. Structure of the SystemDesk adapter

external modeling tool. This problem is explained in the following using *dSPACE SystemDesk* as an example.

dSPACE SystemDesk is a proprietary modeling tool for AUTOSAR modeling. It offers a *COM/.NET* interface to access the model that is currently loaded in SystemDesk. Therefore, a special *Java/COM* adapter is required to connect the Java-based transformation system to SystemDesk. The architecture of the SystemDesk adapter is shown in Figure 14. Another problem is SystemDesk's meta model, which is different from the AUTOSAR meta model. For example, SystemDesk provides a *Project Library* that contains the types of software components, ports, etc. Instances of these types can be used in the system model. However, it is also possible to create a type directly in the system model. Furthermore, a SystemDesk model contains several predefined root packages for the software, hardware and system configurations. These correspond to ordinary compositions in AUTOSAR which are not contained in any other composition. This problem is aggravated because SystemDesk's meta model was not available to us, and had to be reconstructed from the COM interface specification.

The tool adapter has to handle these SystemDesk specific issues and produce a standard-conformant EMF-based AUTOSAR model. First, we have tried to do the translation directly in the adapter code [17]. However, this has led to a very complex and hardly maintainable adapter code. While this transformation is essentially a model transformation from a SystemDesk to an AUTOSAR model, we have now used the model transformation system a second time to perform this transformation. The COM interface is used to synchronize SystemDesk's model with a corresponding EMF-based model. The model transformation system synchronizes this model with the AUTOSAR model that the adapter provides to the transformation system. This solution makes maintenance of the adapter much easier. Most of the adapter's logic is encoded in model transformation rules that can easily be adapted and extended. Maintainability is an important issue due

to the enormous complexity of the AUTOSAR meta model and its constant advancement.

We have encountered another problem, which regards object identity. It is not possible to directly reference an object in SystemDesk's memory. Instead a corresponding Java object has to be created. In an earlier version, SystemDesk did not support UUIDs. Therefore, it was hard to match Java objects to SystemDesk objects. This problem has been circumvented by always creating a second EMF-based copy of the current SystemDesk model, comparing it to the first copy, and merging the differences into the first copy using EMF Compare[12]. However, EMF Compare does not work very reliably without UUIDs. Now, SystemDesk supports UUIDs and matching corresponding objects in Java and SystemDesk is easy.

7 Model Consistency

Model consistency plays an important role, not only regarding consistency between different models or model elements of different models, but also between the elements of one model. In the following, we describe why model consistency is a crucial aspect, especially when model synchronization techniques are applied.

When model synchronization techniques are used, normally several model elements of two different synchronized models describe the same thing. These model elements can be synchronized using model synchronization techniques like TGGs. Model consistency concerning these semantically identical elements of different modeling languages is supported by the synchronization itself, as changes of model elements included in one model are carried over to the other model.

In most cases where synchronization techniques are used, the property holds that not all elements of a model are also reflected in a corresponding synchronized model. Normally, this is the case because different modeling languages, and consequently also different types of models, are synchronized. Such modeling languages have a specific purpose and, thus, different properties with different semantics are expressed by different languages. Therefore, dependencies or other properties can exist within the same model between synchronized and non-synchronized elements, which are not reflected by the synchronization mechanism itself.

In Figure 15, the two big circles on the left and right side represent two different models. In each model, a subset (represented through a cloud) is synchronized via a model synchronization system with the semantically identical elements of the corresponding model. Thus, consistency between elements from the cloud of the left side and on the right side is maintained by the synchronization system itself. Like denoted by the arrow with the exclamation mark on top, dependencies can also exist within the same model between synchronized and non-synchronized elements. Such properties can be invalidated when a model element is updated by a synchronization activity.

[12] http://www.eclipse.org/modeling/emft/

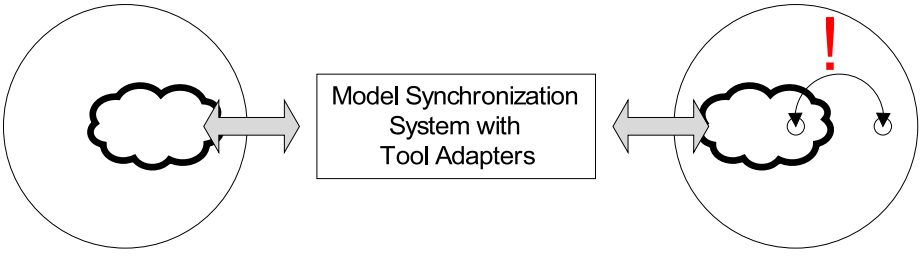

Fig. 15. Synchronization of parts of the models can lead to inconsistencies with other parts

An example of such a situation, where additional properties need to be checked is described subsequently. The SysML model shown in Figure 2 consists of six SysML blocks, but only the three lower blocks represent software. In Figure 4, these three SysML blocks are reflected in the form of semantically identical elements of an AUTOSAR model. The other three SysML blocks, which represent hardware, are not present in the AUTOSAR model. In case these two models are synchronized, only the constituents representing software are synchronized between the SysML and AUTOSAR model. Changing the bit width of the *IO* port of the *ASWCSensor* of Figure 4 is a valid operation in the AUTOSAR model, but applying these changes via the synchronization to the corresponding SysML model leads to a violation of a property of the SysML model. This is the case, because connectors in SysML are only allowed to connect ports with the same bit width. Changing the bit width of the *IO* port of the SysML block *ASWCSensor* to 16 bits, e.g. by a synchronization, without changing the bit width of the corresponding port of the block *HWSensor8Bit* leads to a violation of this property while the AUTOSAR model is still consistent.

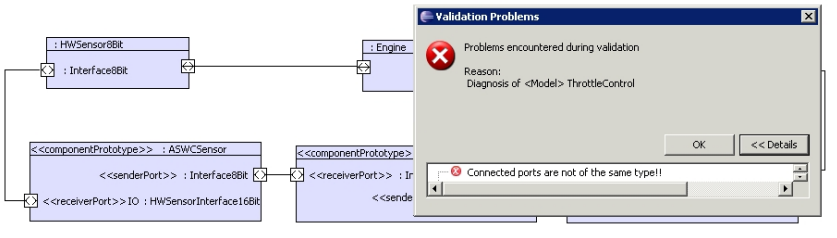

Fig. 16. Screenshot of the OCL validation dialog in TOPCASED

Ideally, the modeling tool should provide a mechanism to check models for syntactical and semantical correctness, not only to check models after synchronizations, but also to aid the user. TOPCASED, respectively EMF, provides such a validation mechanism. Validating the situation described above results in an error message like shown in Figure 16.

Currently, these constraint checks are not invoked automatically, but only on the user's request. The reason is that constraint evaluation in TOPCASED can take a very long time, because it always analyzes the whole model. Of course, it is desirable to check a model immediately after a synchronization for better usability.

8 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. In the following, we describe different usage scenarios in which the shown architecture allows an enhanced development process using model transformation and synchronization techniques.

8.1 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 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).

8.2 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 *ASWC-Sensor* as described in Section 7. A complete retransformation would discard these modifications.

8.3 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 many model transformation approaches only permit unidirectional transformations, our approach works bidirectionally and incrementally. Additional details in the SysML model are preserved which would otherwise be lost.

How such a propagation of changes 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 4 is changed in the AUTOSAR model, the transformation system updates the corresponding SysML IO port shown in Figure 2 accordingly without overwriting the whole SysML model. When the SysML model is updated, the OCL constraint described in Section 3.1 is violated (see Figure 16) because the SysML connector is connected to ports, that have different types.

When elements have been added to the AUTOSAR model that are not relevant for the SysML model (e.g., on a more detailed abstraction level), these elements are ignored by the transformation system. This is the case, because no transformation rules have been defined for these elements.

8.4 Iterative and Flexible Processes

The usage scenarios outlined in Sections 8.1, 8.2 and 8.3 demonstrate that our approach can handle changes occurring in either model in any order. Therefore, the approach enables not only a strict sequential ordering, i.e. the SysML model is specified first and the AUTOSAR model is derived from it (Section 8.1). It also allows, that changes in the SysML model are propagated to an already existing AUTOSAR model (Section 8.2) and that necessary changes in the AUTOSAR model are also accordingly adjusted in the SysML model (see 8.3). Therefore, instead of a rigid sequential process, also iterative and more flexible processes can be supported. 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 of the SysML model like described before.

9 Conclusion and Future Work

During the development of complex engineering solutions, several models are employed to capture the design decisions of different disciplines. We have presented an approach that supports synchronizing these models when they overlap with regard to the captured information. The solution enables that the interplay between the different development activities in different disciplines and the overarching system engineering can be kept consistent at minimal costs even though we do not forbid changes in the different models, which might impact each other or could lead to inconsistencies. The only limitation is that parallel changes in the different models are not supported. Although the underlying transformation

system can only synchronize two models, chains of transformations can be built to connect more than two models.[13]

We have further demonstrated that SysML models employed early on by system engineers and AUTOSAR models employed later on in the software development process can be kept consistent using our approach thanks to the use of model synchronization techniques and additional consistency rules. It has been further outlined that flexible usage scenarios, and in particular iterative development, become manageable when employing our approach.

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

Acknowledgement

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

References

1. Systems Modeling Language v. 1.1 (November 2008), http://www.sysml.org
2. Aldazabal, A., Baily, T., Nanclares, F., Sadovykh, A., Hein, C., Ritter, T.: Automated model driven development processes (2008)
3. AUTOSAR: UML Profile for AUTOSAR (January 2007), aUTOSAR GbR
4. Becker, S., Herold, S., Lohmann, S., Westfechtel, B.: A graph-based algorithm for consistency maintenance in incremental and interactive integration tools. Software and Systems Modeling (SoSyM) 6(3), 287–315 (2007), http://dx.doi.org/10.1007/s10270-006-0045-5
5. Bergmann, G., Ökrös, A., Ráth, I., Varró, D., Varró, G.: Incremental pattern matching in the viatra model transformation system. In: GRaMoT 2008: Proceedings of the Third International Workshop on Graph and Model Transformations, pp. 25–32. ACM, New York (2008)
6. Borland Together Architect, http://www.borland.com/
7. Burmester, S., Giese, H., Niere, J., Tichy, M., Wadsack, J.P., Wagner, R., Wendehals, L., Zündorf, A.: 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 (2004), http://www.upb.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/2004/STTT-BGN+04.pdf
8. Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D.: VIATRA: Visual Automated Transformations for Formal Verification and Validation of UML Models. In: Richardson, J., Emmerich, W., Wile, D. (eds.) Proc. ASE 2002: 17th IEEE International Conference on Automated Software Engineering, September 23, pp. 267–270. IEEE Press, Edinburgh (2002)

[13] An example is the chain SysML model=>AUTOSAR model=>SystemDesk model, although this chain of transformations was not originally intended.

9. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM System Journal 45(3) (July 2006)
10. Darmstadt, T.U.: Moflon (2007), http://www.moflon.org
11. Egyed, A.: Fixing Inconsistencies in UML Design Models. In: Proceedings of the 29th International Conference on Software Engineering (ICSE), Minneapolis, MN, USA, pp. 292–301. IEEE Computer Society, Los Alamitos (May 2007)
12. Ermel, C., Rudolf, M., Taentzer, G.: The AGG Approach: Language and Environment. In: Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, vol. 2. World Scientific Publishing, Singapore (1999)
13. Finkelstein, A.: A Foolish Consistency: Technical Challenges in Consistency Management. In: Ibrahim, M., Küng, J., Revell, N. (eds.) DEXA 2000. LNCS, vol. 1873, pp. 1–5. Springer, Heidelberg (2000)
14. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story Diagrams: A new Graph Rewrite Language based on the Unified Modeling Language. In: Ehrig, H., Engels, G., Kreowski, H.-J., Rozenberg, G. (eds.) TAGT 1998. LNCS, vol. 1764, pp. 296–309. Springer, Heidelberg (2000), http://www.upb.de/cs/ag-schaefer/Veroeffentlichungen/Quellen/Papers/1998/TAGT1998.pdf
15. France Telecom: SmartQVT, http://smartqvt.elibel.tm.fr/
16. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: Review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards final Standard. OMG, 250 First Avenue, Needham, MA 02494, USA (2003), http://www.omg.org/docs/ad/03-08-02.pdf
17. Giese, H., Hildebrandt, S., Neumann, S.: Towards Integrating SysML and AUTOSAR Modeling via Bidirectional Model Synchronization. In: 5th Workshop on Model-Based Development of Embedded Systems (MBEES) (2009)
18. Giese, H., Hildebrandt, S., Seibel, A.: Improved Flexibility and Scalability by Interpreting Story Diagrams. In: Magaria, T., Padberg, J., Taentzer, G. (eds.) Proceedings of the Eighth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT 2009) (2009)
19. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling 8(1) (February 1, 2009), http://www.springerlink.com/content/j716245824112n27/
20. Guerra, E., de Lara, J.: Event-Driven Grammars: Towards the Integration of Meta-Modelling and Graph Transformation. In: Ehrig, H., Engels, G., Parisi-Presicce, F., Rozenberg, G. (eds.) ICGT 2004. LNCS, vol. 3256, pp. 54–69. Springer, Heidelberg (2004)
21. ikv++ technologies ag: medini QVT (2007), http://www.ikv.de
22. Johnson, T., Paredis, C., Burkhart, R.: Integrating Models and Simulations of Continuous Dynamics into SysML (2008)
23. Kent, S.: Model driven engineering. In: Butler, M., Petre, L., Sere, K. (eds.) IFM 2002. LNCS, vol. 2335, pp. 286–298. Springer, Heidelberg (2002)
24. Kindler, E., Rubin, V., Wagner, R.: An Adaptable TGG Interpreter for In-Memory Model Transformation. In: Proc. of the Fujaba Days 2004, Darmstadt, Germany, pp. 35–38 (September 2004)
25. Marschall, F., Braun, P.: Model Transformations for the MDA with BOTL. In: Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications. Technical Report TR-CTIT-03-27, Univeristy of Twente (June 2003)
26. Object Management Group: Meta Object Facility (MOF) 2.0 Core Specification (January 2006), document ptc/06-11-01

27. OMG, O.M.G.: Object Constraint Language (May 2006), http://www.omg.org/spec/OCL/2.0/PDF
28. Schürr, A., Winter, A.J., Zündorf, A.: The PROGRES Approach: Language and Environment. In: Handbook of Graph Grammars and Computing by Graph Transformation: Applications, Languages, and Tools, vol. 2, pp. 487–550. World Scientific Publishing Co., Inc., River Edge (1999)
29. Schürr, A.: Specification of Graph Translators with Triple Graph Grammars. In: Mayr, E.W., Schmidt, G., Tinhofer, G. (eds.) WG 1994. LNCS, vol. 903. Springer, Heidelberg (1995)
30. Stürmer, I., Kreuz, I., Schäfer, W., Schürr, A.: Enhanced Simulink Stateflow Model Transformation: The MATE Approach. In: Proc. of MathWorks Automotive Conference (MAC 2007). Dearborn (MI), USA (2007)
31. Tata Consultancy Services: ModelMorf (2007), http://www.tcs-trddc.com/ModelMorf/index.htm
32. University of Paderborn. Fujaba Tool Suite, Germany, http://www.fujaba.de/
33. Varró, D., Varró, G., Pataricza, A.: Designing the Automatic Transformation of Visual Languages. Science of Computer Programming 44(2), 205–227 (2002)
34. Vizhanyo, A., Agrawal, A., Shi, F.: Towards Generation of Efficient Transformations. In: Karsai, G., Visser, E. (eds.) GPCE 2004. LNCS, vol. 3286, pp. 298–316. Springer, Heidelberg (2004)
35. Windpassinger, H.: Modellierungssprache für die Kfz-Software Entwicklung. Elektronik Praxis (2007), http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/analyseentwurf/articles/95528/