Toward Mega Models for Maintaining Timing Properties of Automotive Systems

Stefan Neumann and Andreas Seibel

Hasso Plattner Institute at the University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
14482 Potsdam, Germany
`forename.surname@hpi.uni-potsdam.de`

Abstract. In the recent years diverse modeling tools for the development of automotive systems emerged and each tool and the associated modeling language have different strengths and weaknesses. A comprehensive solution tries to integrate multiple partially overlapping models from different tools. In general, timing properties are crucial when developing real-time system and in a complex setting minor changes of the models may lead to violations of existing timing requirements. Thus, it is crucial that relevant dependencies between models and related timing properties are explicitly captured, allowing the analysis of the impact of changes on the timing properties and timing requirements. However, current modeling tools and languages do not explicitly encode all relevant dependencies and, thus, violations may remain undetected. In this paper we propose to use the initial concept of mega models as a solution for the support of those dependencies relevant for timing properties.

1 Introduction

Over the last few years, diverse modeling tools for the development of automotive systems emerged with each has different strengths and weaknesses. Examples of professional modeling tools are TOPCASED (see `http://topcased.org`), which employs SysML (see `http://www.omgsysml.org`) as modeling language with a strong focus on modeling several types of requirements, SystemDesk (see `http://www.dspace.de`), which supports AUTOSAR (see `http://www.autosar.org`) as architectural modeling language, and MATLAB/Simulink (see `http://www.mathworks.com/products/matlab/`), which strength is modeling behavioral aspects.

A comprehensive solution has to combine the strengths of individual solutions. Thus, it has to integrating multiple partially overlapping models from different modeling tools. In the past, we have developed a tool-chain that integrates several tools (including the aforementioned tools) for modeling in the domain of automative systems. We additionally integrated a real-time simulation tool called chronSIM (see `http://www.inchron.com/chronsim.html`) for analyzing timing properties [1], which is an important part of the development process.

In general, timing properties are crucial when developing real-time system and they need to be considered at different levels of abstraction, within different models and in different development steps. In a complex setting, minor changes of the models may lead to violations of existing timing properties and, therefore, timing requirements. Thus, it is crucial that relevant dependencies between models and related timing properties are explicitly captured, which permits analyzing the impact of changes on the timing properties and timing requirements. However, in the most cases current modeling tools do not explicitly encode all relevant dependencies that have an impact on timing properties, which are dependencies between different models and even between elements in the same model. This may result in violations of timing requirements that remain undetected. In this paper, we propose the initial concept of employing mega models as a solution to encode these dependencies, which supports the explicit maintenance of timing properties. Generally, mega models are models of models and their relationships [2], which can be used to explicitly represent any kind of modeling artifacts. To also encode dependencies within models, we need more detailed relationships as proposed in [3]. There we proposed to use hierarchical modeling artifacts and relationships to encode dependencies at any level of detail.

Following in Section 2, we give an example of how different models depend on each other and how different techniques can be employed to support an overall development process. We also discuss limitations and restrictions of the employed techniques and in Section 3 a discussion concerning the proposed solution is given. Section 4 provides related work and in Section 5 we conclude our proposal.

2 Application Scenario

In our current tool-chain, we have various dependencies between elements of different models (inter-model) and even between elements of the same model (intra-model), which are insufficiently supported in a sense that timing requirements are not violated by changing individual model elements. In many cases, even detecting that crucial timing properties are potentially impacted by some modification is rarely possible. To show how different types of dependencies might look like, Figure 1 shows a simple application example with elements organized in different models.

The figure represents in a simplified form artifacts related to four different models: a SysML model, an AUTOSAR model, a task-based model and a behavioral model. The task-based model is used by chronSIM for real-time simulation purpose. The behavioral model is a specification that is modeled within MATLAB/Simulink. MATLAB and chronSIM models are depicted as dashed rectangles only because currently we only import and export these models from and to AUTOSAR. The models itself exist manifested in form of associated project or model files. The figure also shows some model elements of the provided models. The SysML model defines a hardware platform, a software component and a timing requirement. The AUTOSAR model also defines a hardware platform, a software component and a timing requirement (latency timing requirement), which are semantically equivalent to the hardware platform, the software component and the timing requirement of the SysML model because they represent equivalent parts of the system. In addition, the AUTOSAR model has a a latency timing guarantee, which is a timing property that is somehow guaranteed, e.g., by the developer. The software component of the AUTOSAR model additionally contains a runnable, which defines its behavior.

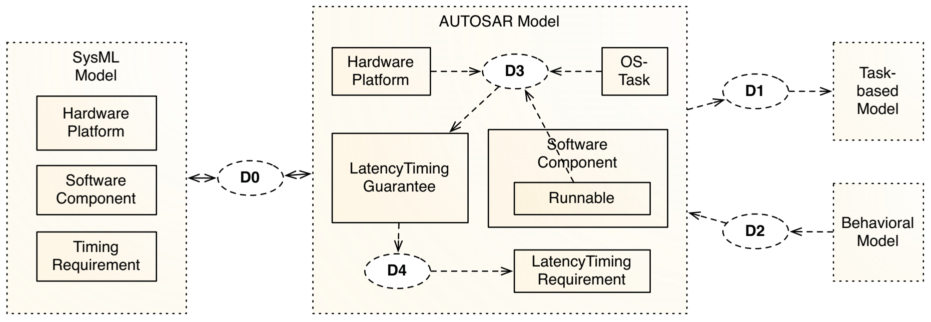

Fig. 1. Simplified application example

The dashed ellipses D0-D4 denote dependencies that implicitly exist between different models (D0, D1 and D2) as well as within models (D3 and D4). D0 reflects the bi-directional dependency between the SysML Model and the AUTOSAR model, which is currently realized by a bi-directional model-synchronization (cf. [1]). It synchronizes the overlapping model elements of both models. D1 denotes a dependency between the AUTOSAR model and the task-based model, which is currently realized in one direction by a model transformation implemented directly in Java. After simulating the task-based model, the simulation results must be propagated back to the AUTOSAR model, which is currently done manually. D2 denotes a dependency between the behavioral model and the AUTOSAR model. This is currently realized by generating code from MATLAB/Simulink and subsequently mapping the generated code manually into the AUTOSAR model. D3 is an implicit dependency between the OS-Task, the hardware platform, the runnable and the latency timing guarantee. The timing guarantee is an end-to-end timing property of the software component. Thus, if one of the dependent elements has changed the timing guarantee is potentially invalidated. D4 depicts the dependency that implicitly exists between the latency timing guarantee and the latency timing requirement. In the case the timing guarantee changes the timing requirement may be violated. Currently, those exemplary dependencies cannot explicitly defined in our tool-chainbecause we have no model for inter-model dependencies nor the AUTOSAR metamodel supports defining dependencies (D3 and D4) between all related elements explicitly.

3 Employing Mega Models in the Automotive Domain

In this position paper, we overcome the aforementioned issue of not being able to express inter-model and even intra-model dependencies by employing the notion of mega models. Our proposed solution employs mega models for defining relationships between modeling artifacts as fist-class entities. Therefore, models are explicitly represented as modeling artifacts in a mega model and dependencies between models are explicitly encoded by means of relationships between modeling artifacts. Additionally, a mega model supporting a flexible level-of-detail, as shown in [3], allows the definition of arbitrary relationships also between single elements of models. Thus, we can overcome the problem of defining intra-model-dependencies by encoding required dependencies between model-elements as relationships in the mega model without changing the metamodel (e.g., in the case of dependency D2 of the AUTOSAR model). In addition, the semantic of relationships can be expressed by arbitrary model operations, like in the case of a model-synchronization, etc. Figure 2 shows a high-level view of our current tool-chain captured by a mega model.

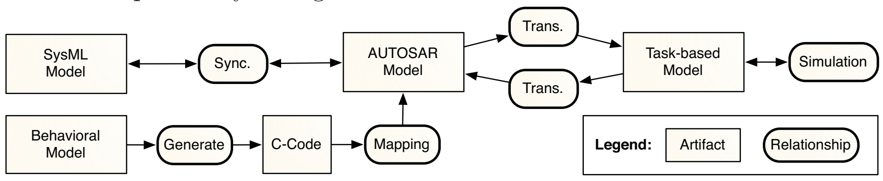

Fig. 2. High-level view of the mega model

The figure shows five models, which are now represented as modeling artifacts of the mega model. Between these modeling artifacts we can define relationships, which reflect the required dependencies of our application example in Figure 1. The SysML model has a bi-directional synchronization relationship with the AUTOSAR model. The behavioral model has a generation relationship to C-Code, which is mapped to the AUTOSAR model expressed by the mapping relationship. The AUTOSAR model has a transformation relationship to the task-based model. For each direction there is a distinct relationship because it is not a bi-directional synchronization. The simulation relationship denotes that the task-based model is in a simulation dependency with itself. When simulating the model, the results are stored in the same model.

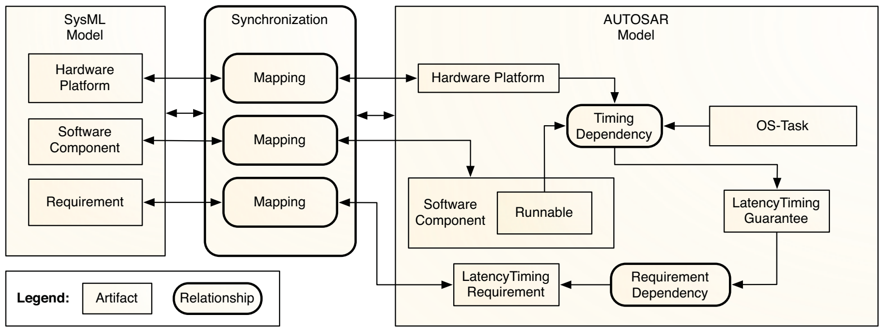

Fig. 3. Lower-level view of the mega model

As shown in [3], a mega model can also represent lower-level modeling artifacts, e.g., elements of models. This refinement also holds for relationships. Figure 3 shows a view on the mega model with details of the SysML model, the synchronization relationship and the AUTOSAR model. The synchronization relationship contains three mapping relationships, which denote the correspondence of the synchronized modeling artifacts. Within the AUTOSAR model, we can now explicitly encode the dependencies of the elements that directly impact the latency timing guarantee (timing dependency) as well as the dependency that directly impacts the latency timing requirement (requirement dependency).

A prerequisite for implementing this solution is a platform like Eclipse in combination with EMF (see `http://www.eclipse.org/emf`), which is responsible for hosting all models in central workspace. If all required artifacts are accessible in that workspace, we can easily represent all required modeling artifacts within the mega model and relate them appropriately.

4 Related Work

Due to the lack of space, we will only briefly discuss two different approaches related to our proposal. In [4] an approach called ModelBus is described that integrates tools through adapters to bridge different technologies. Their focus is on orchestrating the development process through modeling services provided by diverse tools. Thus, their approach is process oriented but not model oriented. We want to focus on a generic model (mega model) that represents modeling artifacts of different tools and further maintains relationships between these models on the basis of the generic model. Nevertheless, ModelBus can potentially be combined with our approach by employing its adapter capabilities.

DUALLY [5] is a framework for the support of language and tool interoperability by providing mechanisms to specify dependencies between different (meta) models. Based on these dependencies model transformations can be automatically derived. While we understand model transformation and also model-synchronization (like described in [6]) as a key concept to be used to obtain interoperability between different models, intra-model dependencies are rarely supported by the transformations used in DUALLY.

5 Conclusions

In this paper, we proposed a shift from a tool oriented solution to a model oriented solution to manage the dependencies concerning real-time properties between models in different tools as well between model elements included in the same tool or model. Therefore, we suggest applying mega models for capturing modeling artifacts and dependencies in between. On top of the mega models, we can formally reason about dependencies but also apply impact analysis, etc.

References

1. 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)
2. Bézivin, J., Jouault, F., Valduriez, P.: On the Need for Megamodels. In: Proc of the OOPSLA/GPCE: Best Practices for Model-Driven Software Development workshop, 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. (2004)
3. Seibel, A., Neumann, S., Giese, H.: Dynamic Hierarchical Mega Models: Comprehensive Traceability and its Efficient Maintenance. Software and System Modeling **009**(s10270) (2009)
4. Aldazabal, A., Baily, T., Nanclares, F., Sadovykh, A., Hein, C., Ritter, T.: Automated model driven development processes. In: ECMDA - Tools and Process Integration Workshop, Berlin, June. (2008)
5. Malavolta, I., Muccini, H., Pelliccione, P., Tamburri, D.A.: Providing Architectural Languages and Tools Interoperability through Model Transformation Technologies. IEEE Transactions on Software Engineering **36**(1) (January/February 2010)
6. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling **8**(1) (1 February 2009)