Model Synchronization for Mechatronic Systems

Jan Rieke*

Heinz Nixdorf Institute & Department of Computer Science
University of Paderborn, Paderborn, Germany
jrieke@upb.de

Abstract: The development of complex mechatronic systems requires the close collaboration of different disciplines on models of the system under development. During such a model-based development, inconsistencies between the different discipline-specific models are likely to occur, which could lead to increased development time and costs if they remain unresolved. Model transformation and synchronization are promising techniques to detect and resolve such inconsistencies. However, existing model synchronization solutions are not yet powerful enough to support such an application scenario. My goal is to extend and improve model synchronization techniques so that they allow for synchronized models with multiple views and abstraction levels, as required by this development process.

1 Introduction and Problem Description

From home appliances to transportation systems, modern mechatronic systems become more and more complex and incorporate an increasing amount of software. This increasing complexity poses challenges to the development of such advanced mechatronic systems.

The design guidelines for mechatronic systems, like the VDI 2206 [Ver04], or the development methods elaborated in the Collaborative Research Center (CRC) 614 "Self-Optimizing Concepts and Structures in Mechanical Engineering" in Paderborn, propose that experts from all disciplines collaborate in a first development phase, called the *conceptual design*. Especially, they work out the so called *principle solution*, a system model that is supposed to capture all interdisciplinary concerns, but does usually not contain information relevant only to one discipline. However, in practice, the principle solution rarely captures all interdisciplinary concerns. Additionally, changes to the overall system design may become necessary later, e.g., due to changing requirements. Therefore, cross-disciplinary changes may become necessary during the discipline-specific *refinement* phase.

As an example, consider the construction of a vehicle. The systems engineer attaches a new sensor, because without it the vehicle is unsafe. This change must be propagated to the software engineer, because the sensor's data needs to be processed. Otherwise, the vehicle would not break in time, causing the risk of crashing into a preceding vehicle.

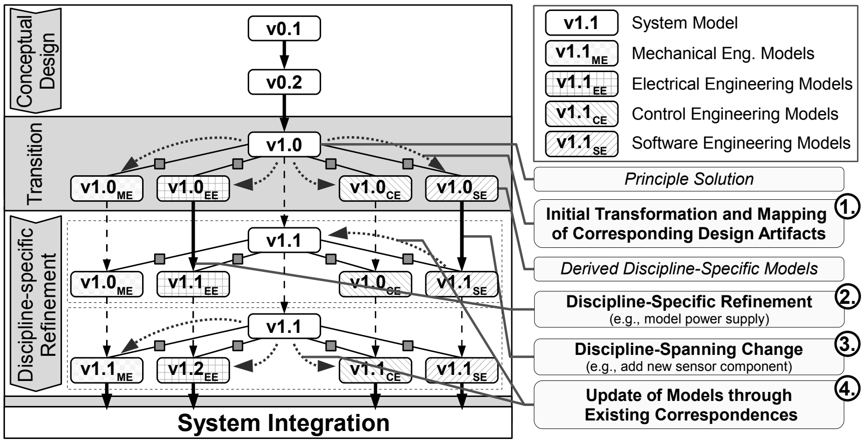
*supported by the International Graduate School Dynamic Intelligent Systems.

Model transformation and synchronization techniques are a promising approach for such synchronization scenarios. Languages and algorithms for bidirectional model synchronization are an intensively researched topic today. However, existing model synchronization techniques mainly focus on basic application scenarios where models of the same or similar expressiveness have to be kept consistent. If, like in our case, models of different abstraction levels, different scopes, or of different domains have to be synchronized, these techniques are often insufficient. Similar issues arise for many model-based development processes (e.g., MDA), too. My goal is to improve existing model synchronization techniques to be able to support such advanced requirements.

I use Triple Graph Grammars (TGG) [Sch94], a declarative transformation language similar to the OMG QVT-R standard [GK10], because we made good experiences with it during the last years and have good tool support for it. TGGs can be applied bidirectionally and allow an intuitive, graphical transformation rule specification.

2 Example and Challenges

Fig. 1 shows an exemplary process, where a cross-disciplinary change occurs. The principle solution is transformed into the different discipline-specific models (1.). The engineers from the different disciplines now start refining their models. E.g., the electrical engineers model the power supply and distribution (2.). Then, the software engineer runs a hazard analysis [GT06] and detects a flaw in the system design: A single distance sensor is not sufficiently reliable, so that a safe distance cannot be guaranteed. Therefore, he proposes to add another sensor to increase the reliability (3.). This change is relevant to other disciplines: E.g., the mechanical engineer has to attach the sensor to the chassis, and the electrical engineer has to connect it to the power supply. Thus, the principle solution has to be updated, because it should reflect all discipline-spanning concerns (4a.). Finally,



Conceptual Design
Transition
Discipline-specific Refinement
v0.1
v0.2
v1.0
v1.0 ME
v1.0 EE
v1.0 CE
v1.0 SE
v1.1
v1.1 EE
v1.1 SE
v1.1 ME
v1.2 EE
v1.1 CE
System Integration
v1.1 System Model
v1.1 ME Mechanical Eng. Models
v1.1 EE Electrical Engineering Models
v1.1 CE Control Engineering Models
v1.1 SE Software Engineering Models
Principle Solution
1. Initial Transformation and Mapping of Corresponding Design Artifacts
Derived Discipline-Specific Models
2. Discipline-Specific Refinement (e.g., model power supply)
3. Discipline-Spanning Change (e.g., add new sensor component)
4. Update of Models through Existing Correspondences

Figure 1: Possible development process with model transformation and synchronization

this updated principle solution is used to update to other discipline-specific models (4b.), preserving manual changes that occurred in the meantime.

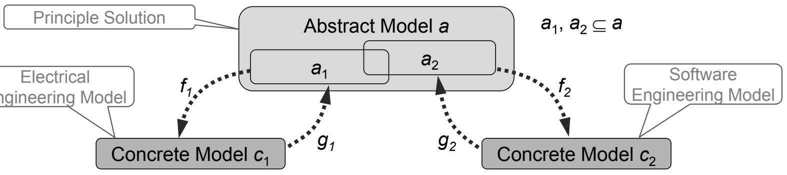


Figure 2: Abstract view of the synchronization scenario

Fig. 2 shows an abstract view of the synchronization scenario described above. There is one abstract model a, the principle solution. Only some parts of a are relevant to a specific discipline model. E.g., the part a_1 is relevant to the concrete model c_1, where a_2 is relevant to c_2. The different parts of a are typically not disjoint, as there are shared elements between the different disciplines. In the course of the development, all of these models may be subject to change. This leads to several challenges in the context of model synchronization, which will be described in the following.

2.1 Definition of abstraction and concretization functions

The abstract model a is used to initially generate the concrete models c_i. E.g., its part a_1 is transformed into c_1 by the transformation function f_1. As a_1 is more abstract, there is a semantic gap between this source and the target c_1 of the model transformation f_1. Usually, the model transformation concretizes all elements of a_1 using defaults. However, the engineers may decide not to stick to this default, but instead use another possible concretization (or refine the default one). Hence, more formally speaking, f_i is not single concretization function, but a family of functions: one abstract model a_i maps to several concrete models $c_{i,j}$. Therefore, the abstraction functions g_i can not be bijective.

Today's model transformation and synchronization techniques lack explicit support for the specification of such families of transformation functions (or non-bijective transformation functions) which are required for synchronizing models on different abstraction levels. Therefore, it is often impossible or at least difficult and non-intuitive to model different concretization rules and store these concretization decisions.

TGGs do not allow for such abstraction relations, either. Usually, TGGs only provide a single rule that translates a specific model element. As we would like to define a family of transformation functions, a first approach would be to specify additional, alternative rules for the same element. However, having different rules for the same element causes unpredictable transformation results in most existing transformation tools, because the order of the rule application is not deterministic: The transformation tool simply selects one of the rules and does not check alternatives.

Therefore, the idea is that, when applying a rule, we should also check for alternative rules that might be applicable. These could then be selected manually by the user where necessary. A critical pair analysis on the rule set could help identifying ambiguous situations (and hiding confluent alternatives). However, in our scenario, transformations should often run automatically, which means that (predefined) default concretizations should be used. Afterwards, the user should be able to search for alternative concretizations on selected elements, or even specify new concretizations which then could be used on other elements, too. Thus, the rule set itself is subject to changes during the development.

2.2 Incrementally updating without changing unaffected target model parts

Both the abstract model a and the concrete models c_i contain parts that are not subject to every transformation. For example, the implementation details of a software component are only contained in c_2, because they are not relevant to other disciplines. In a, only the part a_1 is covered by the transformation rules when translating it to c_1. Therefore, the models cannot be synchronized by running the whole transformation from scratch, because such model-specific information would be lost. Instead, the idea is to *incrementally update* the models by revoking invalid rule applications and applying new rules.

If a change occurs in a, we need to check whether this change has to be propagated to the c_i models. If, for example, an element of c_1 is altered so that the applied rule is invalid, existing model synchronization tools would update a_1 by deleting the corresponding elements (rule revocation), and then try to apply new rules. As there could be elements in a that referenced the deleted elements, this deletion could result in dangling links, i.e., an invalid model. Thus, such simple incremental updates are insufficient for our application scenario.

The idea is to propagate the *editing operations* that took place on the source model. However, specifying a complete transformation on editing operations is a complex task, as there might be a large number of possible editing operations, some of which cannot be anticipated during rule design. Thus, my idea is to emulate such editing operations by applying traditional rules more “intelligent”. For example, this can be achieved by not deleting model elements on rule revocation right away. Instead, they are *marked for removal* and could later be reused (possibly with minor modifications), as elements created by new rule application are often “equal” to some deleted elements. The challenge here is to identify which elements to modify or reuse.

2.3 Concurrent modifications, conflict detection and resolution

In a distributed development environment, several engineers from different disciplines work on their models independently. As it is well-known in software development with source code, this can lead to conflicts. Some of these conflicts can be resolved automatically, but user interaction is still necessary in many cases.

The problem even enlarges when working with interconnected models: The conflict could be due to a change in another discipline. In such a case, manual conflict resolution is ideally done by (a) the systems engineer using the principle solution, or (b) the discipline expert using his own models. As the conflict affects several models, its resolution has to be consistently applied to all models. Thus, the results from a model differencing and merging tool should be included in the model synchronization process.

3 Related Work

Due to lack of space, I focus on the most related work. There is more related work, e.g., on model merging and on non-graph-based approaches.

Giese and Wagner [GW09] present an approach how TGGs can be applied for incrementally updating a target model when changes to a source model occurred, basically by revoking (deleting) and re-applying rules. Giese and Hildebrand [GH09] improved this algorithm so that it removes less elements, but this is still insufficient for my application scenario, because their algorithm still causes unnecessary deletions for more complex editing operations, leading to possible damage of model-specific information.

Ráth et al. [RVV09] propose a solution which does not define the transformation between models any more, but maps between model manipulation operations. This is a fundamentally different approach, as the rule design differs significantly from traditional, declarative model transformations, and its design methodologies still have to be elaborated. Furthermore, designing a complete transformation based upon editing operation is more extensive than using traditional declarative rules. However, I would like to investigate how their ideas could be included in my approach, e.g., to avoid user interaction ambiguous cases.

In the context of chemical process modeling, Körtgen [Kör09] developed a synchronization tool for the case of a simultaneous evolution of both (source and target) models. She defines several kinds of damage types that may occur and gives abstract repair rules for these cases. At runtime, these general repair rules are used to derive concrete repair operations for a specific case. In this way, changes can be propagated with repair operations that delete less elements, i.e., affect less elements that are not subject to the transformations. The approach also includes means for processing alternative rules. However, the solution relies on user-interaction during the transformation, which should, in my application scenario, be avoided where possible.

Xiong et al. [XSHT09] present a synchronization technique that also allows for the simultaneous evolution of both models in parallel. Basically, they run a backward transformation into a new source model, and then use model merging techniques to create the updated final source model. The same is done in forward direction. This technique mainly relies on the capabilities of the model merger. Furthermore, if the model transformation damages model-specific elements during update, their technique cannot avoid information loss, neither. Additionally, they do not incorporate advanced support for conflict detection and resolution, a major requirement in practical scenarios.

4 Summary

Model synchronization is promising technique to support the model-based development of mechatronic systems by automating consistency management tasks, which are time-consuming and error-prone if done manually. However, existing model synchronization techniques mainly focus on simple application scenarios where models of the same or similar expressiveness have to be kept consistent. If models of different abstraction levels or of different scopes have to be synchronized, these techniques are often insufficient.

My aim is to improve existing model synchronization techniques to be able to support such advanced requirements. My hypothesis is that such advanced techniques could greatly improve the development of mechatronic systems as well as other model-based development approaches. I plan to provide a prototype and to apply the developed techniques in the context of a larger example from the CRC 614, in order to evaluate the benefits and possible disadvantages.

References

[GH09] Holger Giese and Stephan Hildebrandt. Efficient Model Synchronization of Large-Scale Models. Technical Report 28, Hasso Plattner Institute at the University of Potsdam, 2009.

[GK10] J. Greenyer and E. Kindler. Comparing relational model transformation technologies: implementing Query/View/Transformation with Triple Graph Grammars. *Software and Systems Modeling (SoSyM)*, 9(1), 2010.

[GT06] H. Giese and M. Tichy. Component-Based Hazard Analysis: Optimal Designs, Product Lines, and Online-Reconfiguration. In *Proc. of the 25th Int. Conference on Computer Safety, Security and Reliability (SAFECOMP), Gdansk, Poland*, pages 156–169, 2006.

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

[Kör09] Anne-Thérèse Körtgen. *Modellierung und Realisierung von Konsistenzsicherungswerkzeugen für simultane Dokumentenentwicklung*. PhD thesis, RWTH Aachen University, 2009.

[RVV09] I. Ráth, G. Varró, and D. Varró. Change-driven model transformations. In *Proc. of Model Driven Engineering Languages and Systems*. Springer, 2009.

[Sch94] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In G. Tinhofer, editor, *20th Int. Workshop on Graph-Theoretic Concepts in Computer Science (WG'94)*, volume 903 of *Lecture Notes in Computer Science (LNCS)*, Heidelberg, 1994. Springer Verlag.

[Ver04] Verein Deutscher Ingenieure. Design Methodology for Mechatronic Systems, 2004.

[XSHT09] Yingfei Xiong, Hui Song, Zhenjiang Hu, and Masato Takeichi. Supporting Parallel Updates with Bidirectional Model Transformations. In *Proc. of the 2nd Int. Conference on Theory and Practice of Model Transformations (ICMT '09)*. Springer-Verlag, 2009.