MODEL TRANSFORMATION FOR DOMAIN SPECIFIC ARCHITECTURE LANGUAGES IN THE AUTOMOTIVE SOFTWARE DEVELOPMENT

Fadi Jabbour  
Software Systems Engineering  
TU Clausthal  
Julius-Albert-Straße 4  
D-38678, Clausthal-Zellerfeld, Germany  
email: fadi.jabbour@tu-clausthal.de

Andreas Rausch  
Software Systems Engineering  
TU Clausthal  
Julius-Albert-Straße 4  
D-38678, Clausthal-Zellerfeld, Germany  
email: andreas.rausch@tu-clausthal.de

Arthur Strasser  
Software Systems Engineering  
TU Clausthal  
Julius-Albert-Straße 4  
D-38678, Clausthal-Zellerfeld, Germany  
email: arthur.strasser@tu-clausthal.de

ABSTRACT
Nowadays, automotive software complexity is handled by using tool-based and model-driven development approaches. Thus, multiple tools and specialized Architecture Description Languages (ADLs) are used at different stages of the development process. This has led to process and tooling gap between these languages.

This paper aims to introduce a solution for the integration problem that exists between tool chains used in the automotive domain. The main goal of this solution is to find an effective semi-automated method that can be used to create model transformation between different automotive ADLs. Our solution provides a refinement to the current approaches of model matching and automated model transformation in order to make them more applicable in the case of complex meta-models. Furthermore, it introduce an iterative algorithm for finding correspondence element between tow meta-models.

KEY WORDS
Model Weaving, Automated Model Transformation, Automotive Industry

1 Introduction

In the distributed development process of an automotive software, software components are developed by Original Equipment Manufacturer (OEM) and TIER 1 supplier. Those software components are integrated to achieve the functional and non-functional requirements of the system. The V-model [1] is a well known development process that is used in component-based development. It defines software design steps where each step provides a specification to the following one. For example at the stage of software integration data consistency and scheduling requirements are verified. In the automotive industry standardized domain specific languages are used to exchange requirements between development steps. Each development team make use of model-based tools that have the ability to import and export the specification. Therefore different development teams exchange a specification between the development stages by tools which are compatible to exactly the same language. The compatibility to the same languages ensures the interoperability. The problem arises of importing or exporting specifications between third party developer tools. For example, the adoption of multicore control units technology in the automotive industry implies several challenges to software integration process such as communications, scheduling and suppliers’ specifics. We identified these challenges in the context of an industrial software development project for Electronic Control Unit’s (ECUs). The existing development process does not allow to efficiently address those challenges. Thus, the new architecture description language called Einheitliche Modulare Architektur Beschreibung (EMAB) was developed. While the OEM uses EMAB and several other Exchange-Formats to specify requirements for integration of software components, a TIER 1 uses tools that are dedicated to AUTomotive Open System ARchitecture (AUTOSAR) to realize the scheduling and communication of supplied and in house components. Therefore, EMAB needs to be integrated into the tool chain. To achieve the integration between those modelling tools a model-to-model transformation between EMAB and other standard formats such as AUTOSAR is one possible solution. For this purpose transformation languages such as ATLAS Transformation Language (ATL) [2][3] and Query View Transformation (QVT)[4] are used. However, developing and maintaining such transformations for a varying set of automotive specific ADLs are expensive and require a great effort. This is attributed to that, in most cases the transformation needs to be created from scratch for each new ADL, and when an ADL changes, all its transformations need to be updated. Therefore, scripts are developed by individual experts in an ad hoc manner to
satisfy the tool chain requirements. Also there are a number of approaches for automating model transformations which addresses this issue such as [5], [6] and [7] but they suffers from some drawbacks due to the complexity of modelling languages in the automotive domain.

In this paper, we aim to introduce a solution to achieve the integration between a project related architecture language EMAB and other standardized automotive ADLs. For this purpose, we investigate the current methods of model matching and automated model transformation to shed the light on their limitations and hence propose the refinement required to create a transformation mechanism that is accurate, easy to use, and expendable to support other description languages. Furthermore, we introduce a new semi-automated algorithm for model matching.

The remainder of the paper is organized as follows: Section 2 motivate a transformation scenario for model matching, based on two meta-models, which shows challenges for current automated transformation approaches. Sections 3 introduces basics for automated model based transformation and 4 reviews critically current approaches in the field of ontology alignment, database schema integration and in the field of modelling languages transformation, respectively. The approach for semi-automated matching and weaving model creation are presented in Section 5 and applied in a case study in 6. The final section is the discussion and conclusion.

2 Motivating example

The main motivation of this work is to find an automated method to create transformation between EMAB and other ADLs in order to integrate EMAB into the automotive tool chain. Thus, in this section we introduce a transformation scenarios between EMAB and AUTOSAR. The meta-models of both languages are discussed in the following subsections (due to complexity of the meta-models, only a part of each meta-model is considered).

2.1 EMAB

EMAB [8] is an architecture description language that supports an efficient integration of software on different Electronic Control Units (ECUs). It allows a partial architecture description for single and multicore systems, which contains all the necessary requirements including scheduling and communication. Then this partial architecture is provided to the supplier for the integration of modules on single or multicore ECU. In this case integration means configuration and verification of module’s scheduling and communication requirements as well as the compilation of source code and linkage of object code. Thus the source or object code is the realization of a module element. Modules are not decomposable architecture elements and describe the interface for communication to other architecture elements. Some modules realize a common context which need to be hidden for reuse purposes. Therefore, groups of modules are described by Static Architecture Group (SAG) elements. The communication at runtime is relevant to ensure data consistency and scheduling requirements. Both are described by process and subprocess elements.

2.2 AUTOSAR

AUTOSAR is an industry standard for the automotive software architecture that aims to master the increasing complexity of automotive electronic architectures and allowing the reuse of software components in different vehicle platforms. It was developed by group of automotive manufacturers and stakeholders. The standard supports a design model, which comprises a set of specifications that describe the components of software architecture and define their interfaces [9] [10].
In AUTOSAR non decomposable elements are described by AtomicSWComponentType elements which in turn are realized by source or object code. A quantity of *componentType elements is composable by the CompositionSWComponentType element for the purpose of encapsulation. The PortPrototype element describe the properties for the communication between elements. A special type of composition is specified by RootSwCompositionPrototype. This type describe the top level software parts of the system which in turn are reusable in other systems. The runtime behavior of each AtomicSWComponentType element is specified by RunnableEntities. Therefore, several attributes related to event- and communication-requirements are specified, which are out of the scope of this paper [11].

Figure 2 illustrates a part from the software component template related meta-model. To describe the composition a recursion pattern is used. The pattern make use of the elements SwComponentType, CompositionSWComponentType and SWComponentPrototype. The latter two inherit the attributes of SwComponentType. Therefore, the CompositionSWComponentType references SWComponentPrototype which in turn specify the type of each composed element. For example, another composition element or an AtomicSWComponentType element. Thereby, top level compositions are referenced by RootSwCompositionPrototype. The element SwcInternalBehavior is contained by AtomicSWComponentType and describe events and datatypes for communication. Both are related to the contained RunnableEntity which in turn inherits attributes from the ExecutableEntity element [11].

2.3 Meta-Model Relations

Relations describe equivalent elements between the source and target meta-model. These need to be used for transformation between the EMAB and AUTOSAR meta-models. In our example we need to find equivalences for class elements and dependency elements between the EMAB and AUTOSAR meta-model. Therefore, elements equivalence can be determined by comparison of the structure and of the semantic. The first problem arises applying a structural comparison. There is no naming similarity between EMAB and AUTOSAR elements. Not one relation can be determined in an automated way without additional equivalence related knowledge. Using this knowledge, we get a quantity of equivalences which are represented in Figure 3. Not mapped elements are represented in the right most column. The equivalent elements are grouped according to "Root", "Static", "Dynamic". "Root" groups meta-model elements which aggregate all other elements. "Static" groups meta-model elements which describe communication and context. "Dynamic" groups all elements which are related to run-time behaviour. This solution is determined using an ATL based tool-chain which enables to define relations in a manual way for structural non similar elements. Due to the complexity of the meta-models a solution in an automated way is necessary. Therefore, a proper solution is to manually define some basic equivalences for each of the groups mentioned above and determine the others by using the structure specification from the meta-models.

Figure 3: Needed relations between EMAB and AUTOSAR meta-models for a valid transformation

3 Background

A review of some key concepts, which are directly related to this work, is presented in this section. It illustrates three
of the important elements in the field of model transformation automation.

### 3.1 Model transformation

This section will discuss model transformation because it is considered as a key element in achieving the integration of different ADLs.

Model transformations is an automatic operation in which one or multiple target models are generated from one or multiple source models, according to a transformation description [12]. The components of this operation are presented in figure 4 where a transform engine transforms a source model into a target model. Both of these models conform to different meta-models. The transformation description contains all the instructions that guide the transformation throughout the transformation process.

![Figure 4: The components of a model transformation](image)

### 3.2 Model weaving

Model weaving is an operation in which relation between source and target meta-models are defined in an abstract way. Although there is no single definition of model weaving, in [13], it is defined as "the operation for setting fine-grained relationships between models or meta-models and executing operations on them based on the semantics of the weaving associations specifically defined for the considered application domain". Furthermore, Didonet Del Fabro et al. [14] defined Model Weaving as "a generic way to establish element correspondences". Those correspondences are represented as a model that is named the weaving model, which conforms to a weaving meta-model. Normally, the weaving model contains equivalence relations between elements in the source and target meta-model that are the semantically equivalent. As such, weaving models can be used by a model transformation language in order to translate source model(s) into target model(s).

Figure 5 depicts the operational context of the proposed model weaving operation and its usage in model transformation. A model weaving operation produces a weaving model representing the mapping between the source meta-model and the target meta-model. Similar to other models, this should conform to a specific weaving meta-model. The resulting weaving model can be used for many operations such as deriving a model transformation.

![Figure 5: Weaving Model](image)

### 3.3 Model matching

As model weaving represents the first step on the way to achieve automated model transformation, the next step is the automated generation of the weaving model itself. This can be achieved through model matching. Model matching is defined, within the model weaving context, as a semi-automatic process to generate relationships between model elements via applying a number of hypotheses related to the input models [15]. The execution of a given heuristic, which seeks to establish links between the elements of the input models, is seen as the basis of the matching process. One example is that a relationship between model elements sharing the same name in the input models could be generated by a given heuristic. However, heuristics are generally inexact. In this regard, the weaving model that a matching operation produce is considered as a proposal that may be validated, invalidated or updated by a human operator. One method to improve the matching process result is by executing several heuristics independently or sequentially and combine their results. In general, currently used matching heuristics depend on the element internal properties (e.g. name, type, constraint, etc.) or on the structure similarity (e.g. elements are similar if they appear in similarly-structured groups) [16][17].

### 4 Related work

Various concepts that are related to this work have been explained and developed by many researchers. Although a number of attempts tried to solve the problem stated above, the concepts developed still fail to offer an optimal solution. This section critically reviews the relevant literature to different technologies and research attempts in the same direction of this work, including model weaving, model transformation automation and model matching. This review will help identifying the gap in the literature and hence help articulating the research problem.
A wide-range work on how to find relationships between two structures comes from the field of ontology alignment and database schema integration, where many concepts were introduced. Some of these concepts didn’t find their way into the fields of model matching and automated model transformation. Thus, investigating them can be very helpful when searching for a solution for this research problem. In the following, some of these studies are discussed in details.

In the work of Mitra et al. [18], a semi-automatic ontologies integration method was introduced. Their approach depends on semantic knowledge articulation tool (SKAT), which is based on simple lexical and structural matching. The domain expert supplies the tool with positive and negative matching rules. Depending on those initial rules, the tool suggests new matches that need to be approved by the domain expert. Then the process is repeated in an iterative manner. In case of complex and large ontologies, they suggested that parts of the ontologies can be matched against each other. The main drawback of this work approaches is that the domain expert needs to supply the tools with too many rules which are in most cases not sufficiently general in nature. Noy and Musen [19] developed a more complex algorithm that needs less interaction from the user. They guided the user in each step of the ontologies merging process by suggesting possible actions and determining conflicts. Their iterative algorithm starts with the linguistic-similarity matches for the initial comparison. Then concentrate on finding clues based on the structure of the ontology and users actions.

Ehrig et al. [20] argued that manually defined ontology alignment methods such as the methods proposed by previous solutions ([18], [19]) rarely constitute an optimal configuration of sub-strategies from which they have been built. Therefore, they proposed machine learning techniques to select among a set of methods, and not to create methods as suggested in other solutions. This is a complementary approach and share the same drawback with all other machine learning based-algorithms, which is training takes a long time and requires a lot of data. However, the machine learning techniques could be enhanced to support this study in future work.

Apart from the ontology alignment, many techniques in elements match was introduced in the field of database schema integration. Bernstein et al. [17] have classified and summarized most of the work done in this field. Mainly, the matching techniques work by matching element in different schemas depending on their internal features and structure such as linguistic matching, constraint-based matching, structure-based matching or graph matching. Then a strategy is used to combine a list of matches together. For example, independently or sequentially execute matchers and combine their results. While some of those techniques are schema specific, many of them are general in nature and can be used in the field of automotive ADL matching.

In the field of modelling languages transformation, Fabro et al. [5] suggested using different model matching methods to create a weaving model between meta-models. Those methods executed consequently in order to reach a final matching result. Then the model transformation executable code is automatically generated for the weaving model.

The Fabro et al. [5] algorithm works by creating links based on the Cartesian product between all the elements in matched meta-models. Then multiple similarity methods are executed to find a similarity value for each link. Each of these methods has a specific weight (given by the user) that determine how each matching method affect the total similarity value. For instance, the weight of elements names similarity method can be 0.8, and 0.2 for the similarity method that compares elements types. This means that the elements are considered more similar if they have the same name as the same type [5]. In the next step, the similarity value for each link is corrected depending on it’s neighbors similarities in order to take into account the structure similarity of the meta-models. That is done by using suitable algorithm such as the similarity flooding algorithm [21]. In the last step links are filtered by keeping only the one with a similarity value higher that a threshold given by the user.

Although the Fabro et al. algorithm work fine on relatively similar meta-model, it suffers from drawbacks when applying it in matching meta-model in the automotive domain. The algorithm result can be explained due to the meta-models complexity and the lack of any naming, type or cardinality similarity between the elements. The drawbacks of the algorithm can be summarize in three main points. First, the user have to choose a weight for each similarity method from a wide-range of possible combinations. Each choice can lead to a different set of correct/wrong matching links. Therefore, in most cases the user have to go through a very long trial and error loop before finding the best combination. Second, The user can’t easily affect the algorithm result by passing positive/negative matching rule. Despite that fact that the user can step in and adjust the weaving model after each step of the algorithm, it’s still a very tedious task due to the very high number of links created by the Cartesian product in the first step. Third, The algorithm does not support meta-model partitioning which is very useful in case of large meta-model because many names and structures are repeated over the meta-model. That can lead to some very negative matching result where class in package A is matched with a class in package B, while it’s super class match with a class in another part of the target meta model due to a naming similarity.

The solution developed in this work will build upon the latter solution [5] through extending it to handle more complex transformation structure and refining it using ideas from ontology alignment and database schema integration fields in order to guarantee getting better matching results.
5 Matching algorithm

In this section we introduce our approach for finding correspondence elements between two meta-models. This approach was developed to overcome the shortcoming of the current approaches when they were used in the context of automotive domain. The main goal of the approach is to offer an easy way to create a weaving model between relatively complex meta-models, which later can be interpreted into an executable transformation code.

Due to the fact that even the best matching algorithms are error prone, and particularly fully-automatic algorithms [17], we decided to develop an iterative algorithm because in matching complex meta-models scenarios a human interaction is needed in the loop. As such, the user needs to validate the matching result after each step. This interaction prevents the errors occurred in the early stages of the algorithm from propagating in later stages. It will also simplify the user task as validating and correcting small part of the solution after each step requires less effort that validating the final result.

Due to the fact that model transformation in industrial setting neither total nor injective [22], our algorithm does not aim to create a weaving model that contains the correspondence elements in the source an target meta-models in a disjoint way and then translate it into one-to-one transformation rules. Instead, we aims to create a weaving model that is complete in nature and can handle complex transformation rules that can be translated directly to model transformation without any needs to modify the transformation code. Thus, filling the gaps between disjoint matched element is one of our main goals.

The matching algorithm is shown in figure 6. It runs in five step as follows.

1. The goal of algorithm’s first step is to find some of the most obvious equivalence links between the classes of source and target meta-models. Let’s call them base links. Those links can be found manually by the user in case of complex meta-model or by using an automated solution such as the one presented by Fabro et al. [5] or other solutions from the field of schema matching then selecting the best n match of it’s result. In case of the automated matching, the user needs to validate the result. This set of correct links is going to be used in the next steps of the algorithm as stating points for the matching process. The number of base links and how they are distributed over the meta-model depends on the matched meta-models complexity.

2. In the second step, the algorithm propagates base links in order to fill the gap between them and complete the weaving model. It uses links found in the first step as starting points to match their super/sub-classes and references. In this step no threshold input is needed. Instead, for each super/sub-class or reference that is connected to the source class in a base link, we search for the best match in the super/sub-classes and references connected or near to the target class of that base link. Notice that the target reference does not have to be directly connected to the target class. In this case the base link is extended to an one-to-many link. The best match is found using a modified version of the similarity flooding algorithm introduced by Melnik et al. [21]. The reason behind modifications done on the similarity flooding algorithm is that in some cases the algorithm will significantly reduces the similarity of base links and gives a small similarity value for elements that is directly connected to them. That is because as mentioned earlier with complex meta-models normally there is no naming or other type of similarity between elements. In other words, elements outside the base links have a similarity value of 0. So instead of using the similarity of all neighbors to calculate an element similarity, only base links similarities are spread all over the meta-models.

3. The new links found in the previous step are validated by the user.

4. The new links get approved by the user enter the base. Next, go back to step two and repeat the process until no new user approved links found or the user decided to stop the algorithm.

5. In the last step, the attribute of each link source class/classes are matched with the attribute of this links target class/classes.

Although our algorithm does not offer the same level of automation provided by other solutions, it has its own advantages which are as follows.

- The main advantage of our solution is that unlike many other algorithm, it can be useful when matching large and complex meta-models that lack linguistic and overall structure similarities, which is the case in most real industrial settings.
- The algorithm supports matching source elements with a complex structure in the target meta-model and

Figure 6: Matching algorithm steps
to the best of our knowledge, such links are not supported by any other solution.

- The base links themselves represent an meta-model partitioning, because each of them can be considered as the center of a meta-model part.
- They can help improve the graphical representation of the matching process by representing each one on different tap.
- No weight or threshold values needs to be provided by the user, but the user needs to validate the result, which is the case in most other solution.

This algorithm is supported with an extended weaving meta-model that can represent one-to-many links and can reconnect the model parts in case of new structure arrangement in the target model. It is also supported with a model transformation that interprets the weaving model into executable ATL transformation code. The extended weaving meta-model and the model transformation is going to be discussed in future works.

6 Case study

In order to have a better understanding on how the latter algorithm works, we discuss a matching scenario between the previously mentioned automotive specific ADLs; EMAB and AUTOSAR.

The lack of linguistic similarity between the element in those meta-models is noticeable despite the fact that both ADLs share the same semantics. They also have very limited structure similarity. Thus, current methods of model matching are not applicable in this case study.

The matching steps between two parts of EMAB and AUTOSAR meta-models applying the algorithm presented in this work are as follows.

1. In the first step, the user provide the algorithm with base links. In this example, there is an obvious equality relation between the Process in EMAB meta-model and the RunnableEntity in AUTOSAR meta-model. Other base link is the equality relation between SAE and SwComponentPrototype.

2. In the second step, the algorithm tries to find the best match for each sub/super class near the sources classes in the base links. That is done using the updated version of the similarity flooding algorithm [21]. As a result, the following new matches are presented to the user in order to validate them: (SAG, CompositionSwComponentType), (Module, AtomicSwComponent) and (IAG, AtomicSwComponent). Here the user can accept the first two and ignore the third one. Because the matched classes are not directly connected to the class SwComponentPrototype, the base and newly founded links are extended to one-to-many links and became (SAE matches SwComponentPrototype that references an object of the type SwComponentType), (SAG matches SwComponentPrototype that references an object of the type CompositionSwComponentType) and (Module matches SwComponentPrototype that references an object of the type AtomicSwComponent). By using the same process starting from the second base link, the class PE matched with ExecutableEntity and Subprocess matched with RunnableEntity.

3. In the next step, the same method is used to find the best match for each reference in the source classes of base links. For example, the best match for the reference referencedProcesses is the runnables. This match will lead to extend the already found matching link so that a SAE is matched with the structure SwComponentPrototype that reference an object of the type AtomicSwComponent that contains an object of the type SwInternalBehavior.

4. After the user validation, the second and third steps are repeated in order to match the whole meta-models.

The result of the case study is summarized in the following: The algorithm creates a weaving model containing 17 of 20 correct and 2 incorrect equivalences. The description of the algorithm above consider only some, but not all matched elements. To get the correct result according to figure 3. The user needs to add 5 and remove 2 equivalences. Therefore 7 additional change operations on the resulting weaving model are necessary. On the other side a version of the algorithm presented by Fabro et al. [5] (performing the steps: 1.cartesian product, 2.naming similarity, 3.similarity flooding, 4.select-threshold of 0.3, 5.link rewrite) creates 1 of 20 correct and 2 incorrect equivalences. To get a correct result according to figure 3. The user need to apply 21 change operations. As input both algorithms use a weaving model that contains 2 correct user defined equivalences.

7 Conclusions

In this paper we briefly reviewed the integration problem found between tool chains used by different parties in the automotive industry. We also shown how model transformation represent a potential solution for this interoperability problem. Thus, in an attempt to find a suitable approach for creating those transformations, we reviewed the current approaches of model transformation automation and discussed their limitation in creating transformation between the automotive domain architecture languages. Furthermore, we reviewed the matching approaches used in the fields of ontology alignment and database schema integration. Finally, we introduce a matching algorithm that can be used in matching complex meta-models. Although this algorithm requires a high level of user interaction and does not offer the same level of automation provided by other solution, we argue that it is more applicable in an actual industrial setting and overcome some of the current heuristic based matching limitations. The goal of this algorithm is to
create a complete weaving model and not a group of one-to-one links, so this model can be directly used to create the model transformation without the need to manually modify and extend the transformation code. The weaving model structure and the transformation generation processes is going to be discussed in a future work.

References


