Lightweight introduction of EAST-ADL2 in an automotive software product line

Andrea Leitner, Nermin Kajtazovic, Roland Mader, Christian Kreiner, Christian Steger, Reinhold Weiß

Institute for Technical Informatics
Graz University of Technology, Austria
{andrea.leitner, nermin.kajtazovic, roland.mader, christian.kreiner, steger, rweiss}@TUGraz.at

Abstract

This paper describes the technical aspects of the transition to a software product line approach in the automotive domain. One major challenge is the current existence of two different emerging standards for this domain, AUTOSAR and EAST-ADL2. These potential standards should be borne in mind during the software product line introduction because they may someday become mandatory. In addition, the existing development process should be changed as little as possible, and one final important requirement for the software product line is the implementation of a single point of control to ensure consistency between various development artifacts.

To this end, we propose a lightweight introduction of EAST-ADL2 as a documentation tool only as an initial step. This is achieved by extracting structural information from AUTOSAR models and automatically generating the corresponding EAST-ADL2 representation. The automatic generation ensures consistency between AUTOSAR and EAST-ADL2 models. As an important side effect, variability information can be extracted in this transformation step and used to build an EAST-ADL2 compositional variability model. This model can then be mapped to the central domain model and used to configure the EAST-ADL2 documentation to the other development artifacts consistently.

In this way, we can accomplish the lightweight introduction of EAST-ADL2 in the development process through the automatic generation and use the generated variability information for configuration from a single control point.

1. Introduction

This paper describes one important research outcome of our current project. The goal of the HybConS project is to generate a generic software architecture for the electronic control unit (ECU) of Hybrid Electrical Vehicles (HEV) [1]. One part of this project is concerned with the introduction of a variability management environment in terms of a software product line (SPL) [2] for the development of automotive control software. A major requirement is to create a “single point of control” for various development artifacts. In this context, “single point of control” means one point from which all variability can be controlled consistently across the various development artifacts.

To better understand the challenges, it is important to get an overview of the development and context. In the automotive domain, model-based development with MATLAB Simulink is very common. The predominant development process is the V-model, as shown in Fig. 1 (left). To improve standardization and interoperability on the software level, the AUTOSAR consortium has developed the AUTOSAR standard [3]. Similar efforts have been made on the design level, which have resulted in EAST-ADL2 [4], an architecture description language for the automotive domain. The two concepts are described in more detail below.

The EAST-ADL2 metamodel provides a sophisticated variability package. With this package, it is possible to describe variability at different levels of detail and to derive concrete products. At the beginning of our project we evaluated the applicability of EAST-ADL2 as an SPL environment. We eventually decided to choose another approach, for reasons discussed below.

Our motivation is the need for a software product line approach in the current development process. As mentioned above, one goal is to develop a generic software architecture. Therefore, it is essential to describe variability systematically. To date, a single system development approach has been used. Variants are often built from existing products by copying and adapting existing code. This is an error-prone approach that should be changed.

Since we know that AUTOSAR is becoming a standard, and EAST-ADL2, intended as a systematic way to describe the domain, may also be adopted at some point, we want to include these concepts in our adoption process from the beginning. Currently, we are dealing with an established development process, so it is not possible to make major changes to the process during production. This is the main reason for the evolutionary and lightweight approach.

1.1. Motivation

In order to implement a single point of control it is not only necessary to describe variability, but also to control all kinds of development artifacts consistently. This means that variability mechanisms have to be integrated into all of the...
This by extracting both structural and variability information from an AUTOSAR model. Software components described with AUTOSAR are then transformed into an EAST-ADL2 Functional Design Architecture (FDA) model. This strategy can also be used as a starting point for the automatic extraction of variability information from legacy systems. Currently, this is only possible if the variability information is present in the AUTOSAR description. Variability mechanisms have been supported in AUTOSAR since version 4. This variability information can be used to integrate the EAST-ADL2 model into an existing product line project as one type of development artifact.

In summary, this proposal has two main contributions:

1) The lightweight introduction of EAST-ADL2 in the development process requires a transformation from AUTOSAR to EAST-ADL2. The detailed mapping is described in the following sections.

2) The implementation of a single point of control with respect to variability. Since EAST-ADL2 does not cover all of the steps of the current development process, we treat the EAST-ADL2 model as a development artifact. This means that we integrate variability mechanisms, but control them from one central model. Basic variability information can be automatically extracted in the transformation step.

Sec. 2 provides a short introduction to the concepts of EAST-ADL2 and AUTOSAR and summarizes related work. Sec. 3 describes the concepts of the two major parts of this proposal: Sec. 3.1 describes the basic mapping strategy used for the transformation, and Sec. 3.2 describes the integration of the model in an SPL. Finally, Sec. 4 describes the implementation of the two parts, and Sec. 5 evaluates the results of our transformation approach.
2. Related Work

2.1. EAST-ADL2 overview

The Electronics Architecture and Software Technology - Architecture Description Language (EAST-ADL) is an architecture description language for the automotive domain. It has been developed within the scope of the EAST-EEA (EAST - Embedded Electronic Architecture), ATESST and ATESST\(^2\) (Advancing Traffic Efficiency and Safety through Software Technology) projects. Its primary goal is to provide a detailed description of the entire system and to improve communication in the development environment. This is achieved through the representation of a system on different layers of abstraction (see Sec. 2.1.1) combined with additional modeling aspects, such as variability, requirements modeling, feature modeling, environment modeling, and system level analysis [4].

2.1.1. Structure. As illustrated in Fig. 2, an EAST-ADL2 system model consists of four layers.

On the vehicle level, an abstract description of the entire system in terms of features (as defined in [7]) is provided. This abstract representation serves as a basis for communication with stakeholders (e.g. customers). On the next level, the analysis level, abstract functions are defined by breaking down the requirements and features. This abstract functional description corresponds to a domain concept. For example, it can describe an environment model, devices interacting with an environment, or functions [8]. In the next level, components are further divided into either hardware (e.g. sensors, actuators) or software components. In addition, middleware is modeled to connect device-specific functions to design functions. Design functions form the Functional Design Architecture (FDA), which is the model that this paper focuses on. An abstract description of the hardware is captured in the Hardware Design Architecture. The implementation level is not explicitly defined in the EAST-ADL2 metamodel. Instead, some of the concepts have been aligned to the AUTOSAR specification.

2.2. AUTOSAR overview

AUTomotive Open System ARchitecture (AUTOSAR\(^4\)) represents a standardized, open automotive software architecture developed by a group of more than 150 companies, consisting of automotive manufacturers and suppliers. The demand stems from the growing complexity in automotive software development due to extensive innovations in E/E systems [9].

The main advantages of the AUTOSAR approach are the separation of hardware and software and the resulting reusability of components. The promotional slogan is “Co-operate on standards - compete on implementation”. This means that it should become possible for OEMs and other suppliers to exchange components without revealing any implementation details [10]. The basic AUTOSAR architecture is shown in Fig. 3.

2.2.1. ECU Software Architecture. One essential design concept of AUTOSAR is the separation between ECU-specific and ECU-independent software, i.e. basic software and application software, respectively. An intermediate layer, the virtual function bus, acts as an abstract representation of the communication infrastructure for all software components. Since the communication specification is ECU-independent, high levels of modularity, scalability, exchangeability and reusability are achieved [11]. All this is located at the system level of the AUTOSAR methodology. The

---


implementation of the virtual function bus is provided by the runtime environment (RTE), which is the core of the AUTOSAR architecture [3].

2.3. General literature

Since EAST-ADL2 and AUTOSAR are two relatively new concepts, few attempts for transformations and mappings have been described in the literature. Only one approach has described a possible mapping strategy at the implementation level. In [12] and [13], the project EDONA is introduced. The aim of this project is to integrate heterogeneous tools into one platform in order to support the cooperative development of embedded automotive systems. The most interesting part of this project in the present context is the transformation block called ARGateway. It transforms the software design described in EAST-ADL2 into the AUTOSAR software component description. Since the target in this case is the implementation level, both structural and behavioral information are necessary. This requires a detailed mapping strategy.

Two design patterns for tool integration are presented in [14]. One pattern describes the integration via a data model, whereas the other integrates the process flow. Our approach is based on the ideas of the second pattern. We implemented tool adapters for Papyrus and AUTOSAR, as well as a semantic translator (Mapping API). Since there are only two tools and two languages involved in the transformation, a tool backplane is not necessary here.

In [15], a method for transformations between different model formats is presented. One of the two use cases describes the transformation from MATLAB Simulink to EAST-ADL2. The mapping is described in a so-called structural bridge, which preserves the semantics of the original model. A technical space bridge is used to access the original model data. In contrast to our approach, it is not possible to realize variability representations.

In [16], the transformation between different AUTOSAR metamodel versions is described. This is necessary to ensure interoperability.

A summary of common Model-to-Text and Model-to-Model transformation approaches is provided in [17]. The transformation implementation in this work is achieved as a Direct-Manipulation Approach.

In [18], three fundamental aspects of transformation mechanisms are introduced: scope, direction, and stages. The scope restricts which parts of the program are affected by the transformation. In our case, these are the model elements which should be contained in the target model. The direction of a transformation determines whether the structure of the source or the target program drives the transformation process. In our case, the target model (EAST-ADL2) is the driving part, since only structural and variability information is required there. The stage aspect of a transformation determines the necessary iterations. In our case, two iterations are necessary. This is due to multiple structural dependencies.

A good overview of the terminologies and technologies used for model transformations is given in [19]. Our approach can be classified as both Reverse engineering and Migration.

Domain modeling based on legacy systems is a common approach. Manual modeling using guidelines (e.g. [20]), or semi-automated knowledge extraction (e.g. [21]) can be used. Another interesting approach can be found in [22]. Here, an existing software framework serves as a basis for domain description in the solution space. The metamodel for a consistent problem-space DSL is extracted from this description. This in turn guarantees that there is a generator that can execute all concrete product descriptions using the original framework.

Haugen et. al. [23] describe a separated language approach for specifying variability in domain-specific language models. They propose a Common Variability Language (CVL) and corresponding variability resolution mechanisms embedded in the OMG metamodel stack. This allows for the description of variability in potentially all MOF-based languages, including UML, as well as MOF- and UML profile-based domain-specific languages. Although this represents a general purpose, clean approach for handling variability and providing a single point of control, it is not applicable for the representation of variability in Simulink-based implementations.

3. Design considerations

As mentioned before, our approach consists of two major parts. The first step is the transformation from the AUTOSAR component description to higher level description in EAST-ADL2. Sec. 3.1 provides a detailed description of this transformation step. Second, the EAST-ADL2 representation should be used as a development artifact, which can be controlled from one single point. Sec. 3.2.1 describes the extraction of variability information and the connection to the common domain model.

3.1. AUTOSAR to EAST-ADL2 Mapping

Our mapping strategy defines how elements from AUTOSAR are transformed into the EAST-ADL2 Functional Design Architecture. The FDA does not describe the software architecture from an implementation point of view, but rather from the design perspective. Since the two models describe software on different levels of abstraction, it is necessary to analyze different mapping strategies in order to generate a correct model (documentation) and to reduce losses in the transformation process.
The EAST-ADL2 documentation suggests two types of mappings:

**Detailed mapping** is used if behavioral information from the target model is required. In this case, the function corresponds to AUTOSAR **runnable**.

**Black-box mapping** is adequate if only structural information is required. In this case, the function is mapped to an atomic or composite software component in AUTOSAR.

### 3.1.1. Mapping strategy definition

A mapping proposal in [8] defines the FDA as a functionality of the application software architecture in AUTOSAR. It also gives very important and helpful suggestions about how to execute structure and behavior mapping in detail. However, it is necessary to first decide which of the two mapping concepts proposed in EAST-ADL2 to use. Since, in this case, structural information is more important than behavioral, the black-box solution is chosen. This level of detail excludes the mapping of runnable entities and reduces the effort required to implement the transformation. To the best of our knowledge, no detailed mapping strategy between EAST-ADL2 and AUTOSAR has been described. Nevertheless, there is sufficient information about similarities between models and possible mapping solutions, which can be used to define a detailed mapping strategy.

Next, we need a detailed specification that describes how elements and their properties are mapped. Starting with the description of a complete information flow from a hardware sensor to its software representation, the most important (or all) groups of different software components are present and can be analyzed.

Fig. 5 shows a sample information flow which captures the vehicle’s velocity. The lower layer shows the software representation, and the upper layer represents the corresponding hardware description.

The physical value of the velocity is captured by the sensor and converted here to **current**. Typically, this signal is converted to a microcontroller input type (e.g., voltage) by the **ECU Electronics** component. The signal is then handed over to the standardized **Hardware Abstraction Layer Driver** and is thereafter available in the software. The next steps describe the reverse conversions on the software layer. The **ECU Abstraction** transforms the value received from the **Hardware Abstraction Layer Driver** to **current** and delivers it to the software representation of the sensor. The sensor software component then transforms **current** to the software representation of the physical value of the velocity [24].

We used the documentation of metamodel elements in combination with use cases, such as the one described above, to find analogies to EAST-ADL2 model elements. The detailed mapping specification is also based on material collected from [8], [24], [12], and [13].

### 3.2. Single point of variability control

As mentioned before, not all parts of the development process can be represented in EAST-ADL2. In particular, the representation of variability in the implementation is currently not possible in a scenario with EAST-ADL2 as the software product line core. In our scenario, the EAST-ADL2 model is used as a development artifact similar to implementation and test, which are controlled from one single point, as shown in Fig. 4B. This single point of control is a feature model represented in pure::variants. To control variability from this common domain model, a connector between Papyrus (EAST-ADL2) and pure::variants has to be implemented. The implementation is described in further detail in Sec. 4.2.1. EAST-ADL2 variability concepts are described below.

#### 3.2.1. Extracting variability information

We first assume that it is not important to document why variability occurs. Therefore, all variants are handled as internal variants. For further propagation towards the analysis block, the developer must decide what will be visible to the customer.

The variability part of the EAST-ADL2 metamodel contains two types of traces. First, domain assets are traced to variation points on the artifact level, which is known as artifact-level variability. Different options of these variation points are handled by special metamodel elements,
called variable elements, which have direct traces to domain assets. Second, domain assets are traced to features and feature models. These feature models are orthogonal to the assets model and only have references to them. In EAST-ADL2, feature models are composed in a hierarchical order, whereby the root feature model is called the technical feature model. It is configured by a single-vehicle-level feature model. The combination of both levels of variability makes it possible to define the product line and to configure the system in EAST-ADL2.

To extract variability information, the variable elements in the target model must first be identified. There is a predefined set of prototypes in the FDA which may vary:
- FunctionPrototype
- FunctionPort
- FunctionConnector
- HardwareComponentPrototype
- HardwarePort
- ClampConnector

Combining these prototypes makes it possible to define variability at various levels of granularity, such as subsystems, software components, cross-cutting characteristics, specializations and functionality. In an initial proof-of-concept only the first two options have been implemented.

### 3.2.2. Connecting variability representations

Fig. 6 illustrates the separation of variability concerns. The figure is aligned to the concept of pure::variants, which consists of four major parts. In the problem space, the domain is abstractly defined in a feature model. There, the commonalities and variabilities of different products are described, and the domain knowledge is made explicit. Concrete product descriptions are derived from this feature model. In the solution space, technical realizations are described in so-called family models. There is one family model for each type of artifact or tool under variability control. These models are consistently controlled from the feature model. In our scenario, we generate a vehicle-level feature model in EAST-ADL2, which is used as the family model for the configuration of the software architecture described in EAST-ADL2. Since the two representations have a similar structure, the import functionality is straightforward. The remaining manual task is the mapping between the problem and the solution space in pure::variants, which introduces the links for the configuration.

### 4. Implementation

#### 4.1. Implementation of the transformation process

Fig. 7 depicts the transformation process.

The implementation consists of three basic components: a SAX Parser, an AUTOSAR Model Builder and a Transformation engine. The following sections describe these components in greater detail.

**4.1.1. Parser.** The transformation process starts with the selection of a parser. For each AUTOSAR schema version there is a SAX parser, which is internally selected. The model parsing process results in an element table, which is used as a temporary storage for the calculated absolute path for each component and the references of this component. First, path information is necessary to identify the references between elements.

**Resolving Paths.** Component paths are stored as relative paths in the AUTOSAR model. For further processing, absolute paths have to be recovered.

The absolute path in AUTOSAR is built by concatenating all package names from the top to the corresponding element. This is the first information stored in the element table.

**Resolving References.** In the next step, references for each component have to be resolved. References in AUTOSAR can be expressed in two ways: (1) with absolute paths and (2) with relative paths. Absolute paths always start with a `/` followed by the root package. They are used as a unique identifier for a package. Capturing absolute references does not require any post-processing effort. Relative references in AUTOSAR are the same as those found in typical file systems. They represent the “outer right excerpt” of an absolute reference and are related to the containing package. To identify the target element, this path needs to be converted into an absolute path. For this calculation, the schema element ReferenceBase is used. There are four possible contents of the ReferenceBase. In the best case, it contains the left part of the absolute path. In the worst case, it contains a reference to some other reference base.

4.1.2. AUTOSAR Model Builder. The AUTOSAR Model Builder takes the element table as its input and builds a version-independent intermediate model. In this way, there is more flexibility in the implementation of the source model, although it causes minor, insignificant performance losses.

4.1.3. Transformation Engine. In the last step, the Transformation Engine builds the EAST-ADL2 model according to a defined mapping strategy. For this purpose, a Mapping API has been implemented in Java to represent the mapping strategy.

The mapping has to follow a specific sequence, since some mappings depend on previous information. Types are mapped first. An example of a type is a composite software component containing prototypes. It is typically referenced by a prototype which is typed by this composite. Therefore, types have to be readily available at the time prototypes are mapped.

In the next step, prototypes are mapped. Since AUTOSAR provides constructs to express references to the root software components explicitly, these references are mapped as root prototypes within the Functional Design Architecture, i.e. they are first-level functions in the FDA. Ports and port interfaces are mapped when all software components already exist in EAST-ADL2. Connectors must also be mapped at the end of the mapping process, since they refer to ports.

4.2. On single point of control implementation

4.2.1. Variability extraction. The first step towards a single point of control is the extraction of variability information. AUTOSAR provides four different mechanisms to describe variation points [3]: aggregations, associations, attribute values and property sets.

For our prototype implementation, the aggregation pattern suffices to enable variability in sub-systems and components. The main challenge in this sub-process is the generation of variability logic for the captured common and variable model assets. Metamodel elements from the EAST-ADL2 variability package are generated and interrelated to each other. The process generates artifact-level and vehicle-level elements. The vehicle level provides the interface for the system configuration. Artifact-level variability, on the other hand, is used for the internal configuration. EAST-ADL2 uses a so-called Compositional Variability Model [25]. After completing the model transformation, the root component is identified, and the variability extension is created by traversing the component tree in top-down order.

The first activity in the process takes an element and checks whether it is an elementary or a composite software component. If it is elementary, it has to be checked to see if it “varies”. If it does, a construct VariableElement is created within a variability extension and referenced to this model asset. Variability is not represented directly in the model, but rather in an orthogonal way. This makes the model and the variability representation independent from each other.

As shown in Fig. 8, for each composite, the following parts are created:

- a public feature model, which contains features that reference the content of a composite,
- an internal binding, which specifies the rules for how a variable content element is affected by selecting features from the public feature model, i.e. configuration decisions.

The public feature model is only visible for the composite container, i.e. for the composite in the next highest level of the hierarchy. The composite’s internal binding defines how this public feature model is configured. This is how the
configuration is propagated over the whole hierarchy. For each feature of a public feature model, one configuration decision corresponding to the concept of transfer functions is created. A collection of these configuration decisions related to one public feature model is thereby packaged into one internal binding. The internal binding and public feature model are the most important constructs, since they contain the whole logic for the system configuration for a specific composite software component.

If a composite contains other composite software components, they need to be processed recursively. The processing of a composite software component is done after the generation of the variability extension for all its elementary and composite parts. Finally, both the internal binding and public feature model must be packaged into a configurable container. This construct is created and referenced to the composite software component.

The last two activities of the process are part of vehicle-level variability. Here, the vehicle-level feature model and a bridge connecting the vehicle level and artifact level in EAST-ADL2 are generated.

In our case, the vehicle-level feature model is generated by simply cloning the root technical feature model. With this simplification, the step can be automated. Further conceptual abstraction would require manual intervention. Still, as a pragmatic approach, it propagates variability not only to the technical feature model, but also up to the vehicle-level feature model. The only difference between the two abstraction layers in our case is that features inside the vehicle-level feature model are represented by an element VehicleFeature, which is a specialization of an element Feature. It is extended for attributes such as cardinality, isOptional, etc. The bridge to the artifact level is created by mapping these vehicle-level features to the features of the root feature model. In fact, they are identical, but the feature model on artifact level is not visible for the configuration.

4.2.2. Variability connector. The single point of control is achieved with Eclipse plugins. This is easily done, since both pure::variants and Papyrus are Eclipse-based tools. Fig. 9 shows the structure of the variability connector architecture. The HybConS Architecture Generator represents the transformation engine described above. The Architecture plug-in is used to import the EAST-ADL2 vehicle-level feature model into the pure::variants solution space and to configure the system from the common domain model in pure::variants. Reading from and writing to EAST-ADL2 models in Papyrus is performed with a built-in plugin that provides this functionality.

The EAST-ADL2 vehicle-level feature model is configured from the pure::variants feature model. The detailed configuration of the composite variability representation in EAST-ADL2 is performed using the built-in configuration capabilities of Papyrus.

5. Evaluation

The prototype does not cover the whole set of AUTOSAR and EAST-ADL2 metamodel elements. Approximately 14% of the AUTOSAR metamodel and 64% of the EAST-ADL2 metamodel have been implemented for this prototype. The relatively limited coverage of the AUTOSAR metamodel in this first version can be explained by the fact that only basic structural elements are necessary in our project scenario.

First, we wanted to evaluate the accuracy of the transformation. It is not easy to automate this test, since this would require a second independent mapping process. Therefore, the accuracy of the generated EAST-ADL2 representation was assessed manually. Some UML tools can generate diagram information for a graphical representation of given model information. This feature was used to visualize the generated model. These graphical representations were compared manually.

As a second evaluation criterion, we measured the performance of transformations. Tab. 1 gives an overview of the
results for seven different use cases. For runtime estimation the following configuration was used: CPU Intel RT2400, 1.83GHz, 2.00 GB RAM, Windows 7 32-bit, JDK 1.6.

The use cases differ in size, as shown in the first part of the table. It shows both the number of model elements and model sizes. The parts of the transformations are types, prototypes, ports and connectors. The last parameter describes the file size.

The lower part of Tab. 1 shows the runtimes for the 3 main processing steps. The total shows that the transformation process is completed within an acceptable amount of time for an average sized model. The third part of the table shows the time required to extract variability information. This process is the slowest, but it has to be performed only once.

The last part of the evaluation shows the automation capabilities provided by this approach. Fig. 10 gives an overview of the automatic and manual tasks. The implementation of automotive software is still a manual task. Some tools can generate AUTOSAR description automatically. The description serves as input for our transformation engine. The transformation step is performed automatically.

The transformation step is the slowest, but it has to be performed only once.

The last part of the evaluation shows the automation capabilities provided by this approach. Fig. 10 gives an overview of the automatic and manual tasks. The implementation of automotive software is still a manual task. Some tools can generate AUTOSAR description automatically. The description serves as input for our transformation engine. The transformation step is performed automatically.

The last part of the evaluation shows the automation capabilities provided by this approach. Fig. 10 gives an overview of the automatic and manual tasks. The implementation of automotive software is still a manual task. Some tools can generate AUTOSAR description automatically. The description serves as input for our transformation engine. The transformation step is performed automatically.

For the implementation of a single point of control, the user has to add variability information to the AUTOSAR description. If variability information exists, it can be extracted automatically to build a compositional variability model in EAST-ADL. The import of the generated EAST-ADL vehicle-level feature model as a pure::variants family model can be automated as well. The last step, the mapping between the pure::variants feature model and the family model that represents the EAST-ADL configuration, must be performed manually.

With this approach, the variability of existing products can be extracted and propagated to a higher level, thereby serving as a base for the domain modeling process. Of course, a higher level of abstraction requires manual work.

In the application engineering process, the configuration for a concrete product is described in pure::variants. The bridge between pure::variants and EAST-ADL/Papyrus enables the automatic propagation of the configuration through all levels.

<table>
<thead>
<tr>
<th>Model sizes [# model elements]</th>
<th>File s. [KB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Types</td>
<td>9</td>
</tr>
<tr>
<td>Prototypes</td>
<td>6</td>
</tr>
<tr>
<td>Ports</td>
<td>12</td>
</tr>
<tr>
<td>Connectors</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>29</td>
</tr>
<tr>
<td>Process</td>
<td>Run time [s]</td>
</tr>
<tr>
<td>SAX Parser</td>
<td>0.14</td>
</tr>
<tr>
<td>Model Build</td>
<td>0.03</td>
</tr>
<tr>
<td>Transformation</td>
<td>1.11</td>
</tr>
<tr>
<td>Total</td>
<td>1.29</td>
</tr>
<tr>
<td>Variability</td>
<td>0.26</td>
</tr>
</tbody>
</table>

Table 1. Transformation performance metrics

6. Conclusion

This paper focuses strongly on the automotive domain. For this setting, we suggest adopting a software product line approach, while bearing in mind that a new standard for architecture description is evolving in this domain. This future standard has its own variability modeling capabilities, which are not used here due to the lack of single point of control mechanisms.

We suggest an initial transformation from AUTOSAR to an EAST-ADL Functional Design Architecture, which enables a lightweight introduction of EAST-ADL2 in a standard automotive software development process. With this approach, the existing development process does not change. Software implemented with MATLAB Simulink can be transformed into an AUTOSAR description. This information is used by our transformation engine to automatically generate a documentation in EAST-ADL2. We have defined a mapping strategy and implemented this strategy in a transformation engine.

The second part of this paper described the use of this transformation to extract variability information. We used the gathered variability information to automatically build a compositional variability model in EAST-ADL2 notation. The extracted information can be used in the same way as other development artifacts, thereby enabling a single point of control.

To date, a prototype showing the basic functionality has been implemented. In future development, this prototype should be enhanced to support the full range of elements. In our opinion, it is not possible to automate the variability extraction process further because some kind of variability information has to be provided at some point in time.
point. A more advanced approach would be to compare existing implementations and analysis to extract potential variability. This logic could easily be integrated into our approach.

Acknowledgment: The authors would like to acknowledge the financial support of the “COMET K2 - Competence Centres for Excellent Technologies Programme” of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economy, Family and Youth (BMWFJ), the Austrian Research Promotion Agency (FFG), the Province of Styria and the Styrian Business Promotion Agency (SFG).

We would furthermore like to express our thanks to our supporting industrial project partner, AVL List GmbH.

References


