Extended Architecture Analysis Description Language for Software Product Line Approach in Embedded Systems (Extended Abstract)

Youngseok Oh, Dan Hyung Lee, Sungwon Kang, Ji Hyun Lee
SSoftware Technology Institute, Information and Communications University, Korea
{youngseok.oh, danlee, kangsw, puduli}@icu.ac.kr

Abstract - Describing architecture variabilities explicitly and precisely is important in the software product line approach for software development since it helps product derivation as well as modeling and managing the variabilities. The SAE AADL is an industry standard architecture analysis and design language for the automotive community, which originally was not intended to be used for software product line. In this paper, we propose EAADL a software product line architecture description language for the automotive domain that extends the SAE AADL. By incorporating orthogonal variability model into it, EAADL offers traceability with requirement engineering as well as the implementation process that is essential in software product line engineering.

I. INTRODUCTION

The software product line introduces and adapts several concepts, such as, commonalities, variabilities, core assets, production plan, etc. Dealing with variabilities is one of the major issues in the industry and academia. To effectively deal with variability, one important essential means is the Architecture Description Language (ADL) for software product line architecture. We can, of course, extend the standard software modeling notation UML for that purpose[1][2][3]. However, in contrast with the generality of UML being languages with the purpose of describing architecture, ADLs allow formal description of architectures by means of the notations such as components, interfaces, connectors, systems, and properties as well as analysis of the architecture based on its formal description.

There are two ADLs that are widely adopted in the software product line community[4][5][6]. One such ADL is Koala. Koala has the notion of switch connector that can be used to introduce the variation point. xADL 2.0 is based on the xArch and defined as a set of XML schemas. It has the notions of optional, variants, and version features to support the software product line architecture. However, none of these ADLs has the capability of explicitly modeling variability information such as dependencies, constraints, binding time, etc., which are essential for expressing and analyzing variability at the architecture level.

In this paper, we show how to support description of variations of a system by means of the abstraction components with interfaces and component implementations. This is done based on the industry standard Architecture Analysis Description Language (AADL) for the automotive community, which was developed by the Society of Automotive Engineers (SAE)[7][8][9].

II. EXTENDED SAE AADL FOR VARIABILITY MODELLING

The SAE AADL is a textual and graphical language used to model and analyze the real time performance critical embedded software systems. It supports the component-based development approach including the component type, implementation, and extensions for variability modeling. The SAE AADL supports the variability realization mechanisms but it still leaves open the issue of variability modeling.

The Extended SAE AADL (EAADL) supports the variability model within the context of the SAE AADL. The variability model information to be added to the EAADL can be obtained from the orthogonal variability model (OVM)[10]. It allows the variability to be separated from the artifacts. The variability model information has to be incorporated with the component type and implementation of SAE AADL in order to provide an explicit traceability with the variability model.

The SAE AADL allows defining new property set. For example, it is possible to declare new property, which is a list of enumeration values and applied to any component categories. The EAADL make use of this feature in order to include the variability information. The enumeration values are VP (Variation Point), V (Variant), mandatory, optional, internal, and external. This means that a component type or implementation may have a property defining a component as a variation point or variant, etc. Other variability information can be declared in the property set in the same way. Table 1 shows how the variability information can be declared in the property set.

<table>
<thead>
<tr>
<th>Variability information</th>
<th>OVM notation</th>
<th>EAADL property set</th>
</tr>
</thead>
<tbody>
<tr>
<td>VP, V, mandatory, optional, internal, external</td>
<td>list of enumeration (VP, V, mandatory, optional, internal, external) applies to (all)</td>
<td></td>
</tr>
<tr>
<td>requires / excludes</td>
<td>RequireRelation: aadlstring applies to (all) ExcludeRelation: aadlstring applies to (all)</td>
<td></td>
</tr>
<tr>
<td>min..max</td>
<td>Alternative: list of aadlstring applies to (all)</td>
<td></td>
</tr>
</tbody>
</table>

The variability has six enumeration values, and a component may have one or three of them. For example, (VP,
mandatory, external), (V, optional). The first one implies that the component is an mandatory external variation point. A constraint depending models the relationship among variation points and variants. The “A requires B” means the component A requires the component B. The “aadlstring” allows any textual name, so that the component names can be listed. An alternative choice allows the cardinality of optional variants; so to speak, it indicates how many optional variants are allowed to be bound in the application. Those property extensions are declared in the component as Fig. 1.

In Fig. 1, new properties are declared in the properties section. It implies that the component is an external mandatory variation point. Since it is a variation point, it has variants “A.impl1” and “A.impl2” with cardinality 1..1, which means an exclusive OR alternative choice of either A.impl1 or A.impl2.

![Fig. 1 EAADL: Application of properties for variability model](image)

**III. CONCLUSION**

When comparing the approach with other approaches, the most important aspect is the support of variability, which includes modeling, representation, management, and instantiation. The second criterion is the semantics of the model represented by the description languages. The third one is whether the quality requirements are represented or not. Fourthly, the analysis capability is compared. Table 2 summarizes the comparison results.

<table>
<thead>
<tr>
<th>Item</th>
<th>EAADL for Software Product Line</th>
<th>Other approaches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variability Modeling</td>
<td>Supported</td>
<td>Supported with other traceability means</td>
</tr>
<tr>
<td>Variability Dependency</td>
<td>Supported (simple)</td>
<td>Supported (complicated)</td>
</tr>
<tr>
<td>Architecture instantiation</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Management</td>
<td>Supported with OVM (simple)</td>
<td>Supported (complicated)</td>
</tr>
<tr>
<td>Semantics for embedded systems</td>
<td>Good</td>
<td>Not good</td>
</tr>
<tr>
<td>Quality requirements</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Analysis</td>
<td>Supported</td>
<td>Supported with profiles</td>
</tr>
</tbody>
</table>

One of the biggest advantages of the EAADL in the software product line will be the easiness of the variability management comparing to other approaches[1][2][3]. The UML approach has at least two or three architecture views, in which the variabilities are dependent and constrained together. The KobrA[2] suggests the decision model to solve this issue, and Gomaa[3] does the traceability table, in which the reference is the feature model and the features are associated with other models (views). However, the SAE AADL model places architecture viewpoints in the component implementation, and it can be structured in the topology form. It is thus much easier than other approaches to manage the variability.

EAADL introduced in this paper helps incorporate the variability management information and make an association with the orthogonal variability modeling. The implementation of EAADL is based on the orthogonal variability model, which provides the meta-model with the variability modeling requisites. As a domain specific architecture description language, EAADL also offers the benefits that SAE AADL already provides: precise semantics together with the capability of modeling and analyzing performance and schedulability.

**ACKNOWLEDGMENT**

This work was supported by the Industrial Technology Development Program funded by the Ministry of Commerce, Industry and Energy(MOCIE, Korea)

**REFERENCES**