Advancing Traffic Efficiency and Safety through Software Technology phase 2 (ATESST2)

Report type: D2.1 – Appendix A3.3 (was I3.3.1)

Dissemination level: PU
Status: Final
Version number: 1.1
Date of preparation: 2010-05-28
Authors

Editor
Mark-Oliver Reiser

E-mail
moreiser@cs.tu-berlin.de

Authors

Andreas Abele
David Servat
Frank Hagl
Hans Blom
Henrik Lönn
Matthias Weber

The Consortium

Volvo Technology Corporation (S)  VW/Carmeq (D)  Centro Ricerche Fiat (I)
Continental Automotive (D)  Delphi/Mecel (S)
Mentor Graphics Hungary (H)  CEA LIST (F)
Kungliga Tekniska Högskolan (S)  Technische Universität Berlin (D)  University of Hull (GB)
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>08-10-27</td>
<td>[TUB_MOR] Initial version with overview and discussion of redundant artifact variation modeling approaches.</td>
</tr>
<tr>
<td>0.2</td>
<td>08-11-24</td>
<td>[TUB_MOR] Added discussion of Merge&amp;Optionality and VP/Variant approaches + description of idea to mimic VPs with Merge&amp;Opt + minor changes.</td>
</tr>
<tr>
<td>0.3</td>
<td>08-12-15</td>
<td>[TUB_MOR] Added discussion of issues (B.1) and (B.2)</td>
</tr>
<tr>
<td>1.0</td>
<td>09-02-09</td>
<td>[TUB_MOR] Finalization of layout.</td>
</tr>
<tr>
<td>1.1</td>
<td>10-05-28</td>
<td>Final corrections and adaptations to be used as appendix for D2.1</td>
</tr>
</tbody>
</table>
List of abbreviations
Table of contents

1 Introduction ................................................................................................................................. 8
2 Variability Concepts in EAST-ADL2 ........................................................................................... 9
   2.1 Overview .................................................................................................................................. 9
   2.2 Details ..................................................................................................................................... 11
3 Review of Current Status of EAST-ADL2 .................................................................................. 12
   3.1 Artifact-Level Variability I – Definition of Design Variants .................................................. 12
      3.1.1 Redundancy in EAST-ADL2 Artifact Variation Modeling .............................................. 12
      3.1.2 Impact of Choosing One of the Two Approaches in EAST-ADL2 .................................... 14
      3.1.3 Discussion and Decision ................................................................................................. 18
   3.2 Artifact-Level Variability II – Definition of When To Use Which Variant ............................... 21
      3.2.1 Basic Options ................................................................................................................ 21
      3.2.2 Discussion and Decision ................................................................................................. 22
   3.3 Review of Other Variability Concepts .................................................................................... 23
4 Conclusions .................................................................................................................................... 24
5 References ..................................................................................................................................... 25

[1] .................................................................................................................................................. 25
List of Figures

Figure 1. The two basic areas of variability management in EAST-ADL2.
Figure 2. Overview of core variability modeling concepts and their relation.
Figure 3. Two approaches of modeling artifact variation in EAST-ADL2.
Figure 4. Two alternative controllers with the Merge & Optionality approach.
Figure 5. Two alternative controllers with the Variation Point / Variant approach.
Figure 6. Complex alternative substructures with the Merge & Optionality approach.
Figure 7. Complex alternative substructures with the Variation Point / Variant approach.
Figure 8. Two consecutive variations with the Merge & Optionality approach.
Figure 9. Two consecutive variations with the Variation Point / Variant approach.
Figure 10. Mimicking two variation points with Merge & Optionality.
Figure 11. Additional hierarchical level when mimicking variation points.
Figure 12. Attaching constraints for (B.1) to the optional ADLFunctionPrototypes.
List of Tables
1 Introduction

This deliverable serves two purposes. It (a) provides a brief overview of variability concepts in the EAST-ADL2 language together with hints on where to find more information and (b) contains a detailed review of the present status of these concepts at the beginning of ATESSST2, together with a summary of the discussion on how to consolidate these concepts. The consolidation effort summarized here took place in Q4 of 2008 in WT3.3 of ATESSST2.
Variability Concepts in EAST-ADL2

2.1 Overview

The goal of the variability support in EAST-ADL2 was to provide feasible means for variability management throughout the different stages of automotive development, from early analysis on the vehicle level down to detailed design specifications of fine-grained subcomponents. Therefore, variability shows up in many parts of the ADL and, in each case, it serves a slightly different purpose or is tailored to a certain viewpoint. Therefore it is vital to first become acquainted with the big picture and the basic areas in which variability is managed within the EAST-ADL2 before going into details.

Figure 1 provides such a big picture. On the right side the familiar EAST-ADL2 abstraction layers are shown. Now, variability management falls into two fundamental categories: variability management on the vehicle level and variability management on the other abstraction layers of EAST-ADL2 such as FAA or FDA. The latter is called “artefact-level variability management”. The left side of the figure provides a characterization of these two areas.

Variability on the vehicle level:
• Very abstract; no design/implementation details.
• Distinction of customer vs. technical perspective.
• Modeling means: only Feature Modeling.

Variability on the „artefact level“:
• Variability of the actual requirements, design, etc.
• Only technical perspective.
• Modeling means: Feature Modeling + Variation Points inside FAA/FDA/...-Diagrams

Figure 1. The two basic areas of variability management in EAST-ADL2.

The most important difference between variability management on the vehicle level and that on the artifact level is related to the high level of abstraction of the vehicle level. On the vehicle level, no design or implementation details of the system are specified (at least in principle, exceptions may apply in practice). Only fundamental, coarse-grained functional and non-functional characteristics of the system – called features – are defined there. On this high level of abstraction, variability simply means that some of these characteristics may or may not be present in a given variant of the complete system. Means to express such variability constitute an integral part of feature modeling, and therefore no additional modeling means for variability management have to be introduced here; feature modeling is fully sufficient. However, the situation is much different on the artifact level. Here we have detailed, complex models and we have to define in detail how they vary from one complete system variant to another. This cannot be done with feature modeling alone. In addition to feature modeling, we need means to express variations in these models, for example in the component diagrams of the FDA. Often this is realized by way of variation points which are each supplied with a list of variants, small model fractions that can be inserted at the
corresponding variation point in the model. It will be shown below that these variation modeling means on artifact level were subject to final refinement in WT3.3 during 2008-Q4.

An important trait of EAST-ADL2’s basic approach toward variability management is that feature modeling is used extensively and consistently on different levels of abstraction. This naturally leads to having several feature models within a single system description, which have to be semantically related to each other. For this purpose, another modeling concept is needed, called configuration link. Such a configuration link relates one feature model to another with respect to configuration. It states how the second model, called the target feature model, has to be configured depending on a certain configuration of the first, the source feature model. In other words, a configuration link defines a transformation of any given configuration of the source feature model to a configuration of the target feature model, which means that whenever a configuration of the source model is provided, a configuration of the target can be derived from that.

In summary, we have three core variability modeling concepts in EAST-ADL2:

- Feature modeling (used on vehicle and artifact level),
- Concept for expressing variations in artifacts on the levels below the vehicle level,
- Configuration links (to link two feature models).

Another important variability concept, i.e. the multi-level concept, can be deemed optional and is therefore omitted from this overview.

The following figure gives an impression how the above three concepts relate to each other. It is not intended to give a detailed account of this and therefore many details were left out. It can be seen that the means for expressing variation in an artifact-level model are not used on the vehicle level. Furthermore, it can be observed how configuration links relate the various feature models on the different abstraction layers; given the description of configuration links above, we see that a configuration of a lower-level feature model, e.g. that of the WiperSystem ADLFunction, can be derived from a given configuration of the customer feature model, for example. The use of feature models along the hierarchy of the system is only shown for FDA, but a similar organizational pattern might be applied in FAA or HDA.

![Figure 2. Overview of core variability modeling concepts and their relation.](image-url)
2.2 Details

Since the details of the variability concepts were defined in the deliverables of ATESST1, this is not repeated here. Instead, we provide some hints on where to find more information.

<table>
<thead>
<tr>
<th>Topic</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>D4.2.1 Chapter 4 + 6.2.3</td>
</tr>
<tr>
<td>Feature Modeling</td>
<td>D4.2.1 Chapter 4.1 + 5.1</td>
</tr>
<tr>
<td></td>
<td>Slides from UML&amp;AADL workshop</td>
</tr>
<tr>
<td>Configuration Links</td>
<td>D4.2.1 Chapter 4.2 + 6.2.2</td>
</tr>
<tr>
<td></td>
<td>Slides from UML&amp;AADL workshop</td>
</tr>
<tr>
<td>Artifact Variation Modeling</td>
<td>D4.2.1 Chapter 6.1 + 6.2.1</td>
</tr>
<tr>
<td>Multi-Level Concept</td>
<td>D4.2.1 Chapter 5.2</td>
</tr>
</tbody>
</table>

In addition, the other deliverables of WP4 in ATESST1, namely D4.1.1 and D4.2.2, also contain valuable information, of course. In particular, information on applying the variability concepts in practice and on reuse w.r.t. variability can be found there.

Furthermore, the following two papers might be useful and provide additional detail:
- on organizing variability within composition hierarchies of a component design: https://www-svn-east-adl.cea.fr/trac/browser/WP7/Publications/VAMISIS09/vamisis09_FINAL.pdf
3 Review of Current Status of EAST-ADL2

As described above, the major goal of this deliverable is to review the current state of variability concepts in EAST-ADL2, consolidate them where necessary or at least identify issues to be tackled during the work in WT3.3 in ATESST2. The aim of this review is not to refine or extend the variability mechanisms but to tidy up what is already there.

A first review of the variability concepts at the beginning of WT3.3 showed, that only the variability concepts for expressing variation on the artifact level require such a clean up. They contain a certain limited conceptual redundancy which should be resolved before continuing with further refinement and extensions in ATESST2. The other concepts, in particular feature modeling and configurations links, are in a rather stable state already and only require minor adjustments, for example renaming of some attributes or associations.

For the purpose of this review, we clearly separate two major concerns of artifact-level variability:

A. How to define what variants exist in a system design?

B. How to define …

1. when a certain variant is to be selected (e.g. "use X in U.S. and Y in EMEA market")
2. how several variants can be combined (e.g. "X excludes Z").

These will be referred to as issues (A), (B.1) and (B.2) throughout this document. We first examine these two concerns separately in the next two sections and then all other concepts of variability modeling are examined together in a single section (Section 3.3).

3.1 Artifact-Level Variability I – Definition of Design Variants

As indicated above, there is a small but relevant redundancy in EAST-ADL2 with respect to the means for expressing variations in artifacts. This redundancy in current EAST-ADL2 will be described in more detail in the coming section. In the subsequent section, a discussion will be given how this redundancy could be resolved in two alternative ways. Finally, in the last section of this chapter, the pros and cons of each possibility will be described and a decision will be taken.

3.1.1 Redundancy in EAST-ADL2 Artifact Variation Modeling

In principle, there are two traditional, fundamental ways of defining variations in a software development artifact:

- Variation Point / Variant approach: explicitly modeled variation points, each supplied with several variants,
- Merging & Optionality approach: directly merging all variants into the artifact and then marking them optional (plus dependencies).

Whenever an artifact contains a variation, i.e. a certain part of this artifact is variable in the sense that different content can occur at this part of the artifact, we can either explicitly mark this part of the artifact as being variable with a dedicated modeling element (i.e. a variation point) and define the different forms the artifact can have at this location (i.e. the variants) or we can simply define the alternative content side by side directly within the artifact and mark them optional (plus defining dependencies between the optional content). In the first case, resolving variability then means to select a single variant at each variation point and in the second case it means ruling out all optional content which is not applicable.

In EAST-ADL2 both of these approaches were introduced for expressing variations in artifacts. In most cases the variation point mechanism is being used, but sometimes also the merging &
optionality approach is applied. Figure 3 shows the diagram “Artifact-Level Variation Management” from the EAST-ADL2 domain model in Enterprise Architect and highlights which modeling entities relate to the two above approaches of artifact variation modeling.

This redundancy in variability concepts is unsatisfactory and introduces unnecessary complexity in EAST-ADL2 variability management. Therefore it was discussed in WT3.3, if one of the two approaches can be sorted out (summarized in the next two sections). For several reasons this decision is rather important for the overall EAST-ADL2 and not trivial to take, mainly because it has a great impact on

- The overall structuring and organization of the EAST-ADL2 domain model and profile,
- The implementation of EAST-ADL2 (for example in an editor),
- The modeling experience from an end-user perspective.
3.1.2 Impact of Choosing One of the Two Approaches in EAST-ADL2

In the previous section, two approaches of artifact variation modeling were identified and it was shown that both are included in EAST-ADL2, leading to a certain redundancy. Now, this section investigates what it would mean to choose each of these approaches as the sole means of expression for that purpose.

To achieve this, we consider a small, largely self-explanatory example of a wiper system containing a trivial variation (an optional rain sensor) and examine how this could be modeled according to each of the aforementioned approaches. We advance toward this question in three steps: first we only consider a very simple architecture of the wiper system with an alternative core controller component, then we examine what it would mean to have a complex internal structure being alternative and finally we deal with modeling two consecutive variations in the wiper system’s architecture.

3.1.2.1 Alternative Component

First we consider modeling the simple case of two subcomponents being alternative with respect to each other.

It is important to note that the above figures do not show EAST-ADL2 diagrams but schematic illustrations of what entities are involved in artifact variation modeling. In particular, the variants of
a certain variation point need not necessarily be painted inside the graphical representation of the variation point, as shown above.

The figures show substantial differences between the two artifact variation modeling approaches. When explicitly modeling variation points, a lot more entities are involved: the variation point itself requires its own ports and an association of the point’s ports to those of the components that represent the variants. This introduces more complexity and causes more modeling effort. The ports of the variation point may, however, be defined and created implicitly by deriving them from the variants’ ports; but in that case we have additional complexity in the domain model definition (i.e. the rules for deriving a variation point’s ports) and automatically updating such derived structural entities would constitute a non-trivial task in all implementations of the EAST-ADL2 (for example, consider the on-the-fly update required when an additional port would be added to the definition of ControllerB in the library of ADLFunctionTypes). Such update algorithms are particularly challenging in editor implementations, substantially increasing their complexity and making them more difficult to maintain and debug.

On the other hand, the Merge & Optionality approach is not only less complex from a conceptual and implementation point of view, it is also more clear from an end-user perspective. In this case, the diagram presentation in an editor could come close to the schematic illustration provided above in Figure 4. It would be immediately applicable and even easier to read and understand (at least in this example, for a discussion of more complex cases see below).

### 3.1.2.2 Alternative Substructures

Let us now consider complex substructures instead of the simple components ControllerA and B.

![Figure 6. Complex alternative substructures with the Merge & Optionality approach.](image)

![Figure 7. Complex alternative substructures with the Variation Point / Variant approach.](image)
This works almost equally well for the Merge & Optionality approach. Only an additional level of grouping is required to denote which components belong to each single alternative. But since the variation groups are orthogonal elements this does not introduce too much complexity.

The Variation Point / Variant approach, on the other hand, introduces in this case even more additional complexity compared to the Merge & Optionality approach. Here, dedicated ADLFunctionTypes have to be created for each alternative to aggregate the subcomponents that belong to this alternative. These are named Var1 and Var2 in the above example.

This means that also in this case the Merge & Optionality approach seems preferable.

### 3.1.2.3 Consecutive Variations

Unfortunately, it is just the other way round when considering several consecutive variations within a single structure of subcomponents. In the following example, we have the variation of a simple and rain-aware controller as before, but in addition there is an adaptor component behind the controller that somehow adapts the wiping speed. In an optional configuration the wiper’s speed is adapted to the vehicle speed.

![Figure 8. Two consecutive variations with the Merge & Optionality approach.](image1)

![Figure 9. Two consecutive variations with the Variation Point / Variant approach.](image2)
Here we can see that in the case of Merge & Optionality the communication connectors become far too complex due to a combinatory effect. If we would have more than two alternatives per variation, the modeling would soon become completely infeasible. The Variation Point / Variant approach, on the other hand, clearly shows its strength here. It is immediately perceivable that there is no complex semantic hidden behind the countless communication links in the other figure but that instead the situation is quite simple and almost all connectors are applicable to all configurations, with the only exception of the connector for the external rain sensor, which is actually optional.

### 3.1.2.4 Relation to Other Parts of EAST-ADL2

The question of following either only the Merge & Optionality approach or only the Variation Point / Variant approach is closely related to the detailed semantics of ports and connectors. If the Merge & Optionality approach was chosen, alternatives in the internal structure of an ADLFunctionType would be merged and directly modeled side by side of one another within the same model. This may lead to illegal EAST-ADL2 models if 1:n and n:1 connections are not allowed in all cases.

This section summarizes an email-Discussion on that topic. There also exists a related Trac ticket, available under [https://www-svn-east-adl.cea.fr/trac/ticket/20](https://www-svn-east-adl.cea.fr/trac/ticket/20).

Considerations on individual cases of connecting several ports:

- **FlowPort 1:n**
  --> signals broadcasted to all consumers (no problem)

- **FlowPort n:1**
  --> n producers fill the FIFO of the single consumer (no problem IF(!) we have FIFO semantics on in-FlowPorts, I think we do - but where is this documented?)

- **ClientServer 1:n**
  --> all servers will provide a response. This might be problematic:
    - We need a FIFO at the client for the server-responses,
    - What about blocking? is the client blocked until the first response comes in or until all responses are received,
    - Does the client have the option to choose among the n servers ? or is he even required to do so ? (probably: no / no)

  At least the first two are required but are not usual client/server semantics (if I'm not mistaken) and both are probably not in the domain model right now.

- **ClientServer n:1**
  --> true, this is the standard case

Relation to MARTE and other methods:

- 1:n and n:1 is is okay for UML and therefore also for MARTE (because MARTE has no constraints on UML in this respect)
- But it has to be clarified what is meant with this—this is a semantic variation point in UML.
- FlowPort is not part of UML. In MARTE it is also okay for FlowPorts (as there are no restrictions on the connectors).
- In AR: 1:n is not possible for ClientServer Ports.
- In MARTE there is the RTService (HLAM) which is dedicated to deal with concurrent access to services.
3.1.3 Discussion and Decision

In this chapter, we will first summarize the pros and cons of the two approaches, i.e. Merge & Optionality and VP/Variants. Following this, a conclusion and decision will be presented.

3.1.3.1 Advantages and Disadvantages

The main advantages and disadvantages of the two approaches can be summarized as follows. For Merge & Optionality we have:

- Pro: intuitive in simple cases.
- Pro: very simple from a user-perspective.
- Pro: very lean and elegant with respect to definitions in domain model; keeps domain model simple.
- Con: does not scale well; not feasible for complex cases. In particular, when having consecutive variations within a single ADLFunctionType’s internal structure, then a combinatorial explosion occurs; countless communication Connectors need to be defined.
- Con: bad readability in complex cases. It is no longer immediately perceptible that all combinations of several consecutive variations are existent.

For VariationPoint/Variants we have:

- Pro: more intuitive (arguably).
- Pro: less know-how has to be put in by modeler; more know-how is “hard-wired” in the ADL’s entities (i.e. variation points and variants).
- Pro: reduces complexity of models in complex cases by avoiding combinatorial explosion (cf. case “consecutive variations” above).
- Con: more things need to be defined, particularly in simple cases (i.e. a mapping needs to be defined from the variation point’s ports to the ports of the ADLFunctionTypes used as variants).
- Con: relatively complex definition in the domain model (esp. rules for deriving the ports of the variation point, including a rule for equality of ports of those ADLFunctionTypes that are used as variants).
- Con: many implicitly defined entities, i.e. ports of variation points, which are actually required in the model, for example to define communication Connectors to/from such implicitly defined ports of variation points. This leads to the problem that they have to be created and deleted automatically by any implementation of the EAST-ADL2, which introduces several challenges.

3.1.3.2 Conclusion

Neither solution is fully satisfying. Overall, the Merge & Optionality seems favorable, due to its simplicity and readability. However, this approach is not applicable to complex cases.

A third solution was therefore examined during the review, leading to a compromise between the two approaches:

- Only Merge & Optionality is provided in the domain model:
  - Subcomponents can be optional,
  - Communication Connectors are merged, leading to n:m,
• Communication Connectors to/from optional subcomponents are implicitly defined as optional (same for delegating ports),
• When resolved, Connectors to/from deselected optional subcomponents are removed,
• A tool can dynamically highlight which Connectors are optional.

In complex cases, when the benefits of the VP/Variants approach are required (esp. when having consecutive variations), variation points and their variants are mimicked with the Merge & Optionality mechanisms: (a) a normal ADLFunctionType is introduced as a variation point (called “pseudo variation point” or “pseudo-VP” below) and (b) the variants are defined as normal, optional and alternative subcomponents of this newly introduced ADLFunction by way of the standard Merge & Optionality mechanisms (called “pseudo variants”).

The following Figure 10 shows how the two variation points from Figure 9 above can be mimicked with the Merge & Optionality mechanism. Compare this to Figure 8 which showed the same situation with applying Merge & Optionality directly, not for mimicking the variation points; it can be seen that now the picture gets much clearer that way. Please note that “Controller_PseudoVP” and “Adaptor_PseudoVP” are two standard ADLFunctionTypes; the substring “PseudoVP” was only used in their name to highlight that they play the role of variation points here.

A first evaluation of this idea showed that it incorporates all benefits of the Merge & Optionality approach and provides sufficient means to deal with complex cases. However, two major remaining drawbacks were identified:

1. **Additional Level of Composition ⇒ Unnecessary and Undesired Overhead**
   The pseudo-VP, i.e. the additional ADLFunctionType, introduces an additional level of composition within the design. Since all ADLFunctionTypes have their own public feature model to organize variability within this hierarchy of composition, this means we would also introduce an additional feature model and hierarchy level for each (pseudo) variation point. While this might be beneficial in rare cases of very high complexity, this additional overhead will usually be useless and therefore undesired.
2. **More Demanding to Apply**

By not providing the modeling entities for the VariationPoint/Variants approach in the domain model and saying they should be mimicked somehow when needed, we in fact shift responsibility to the end-user of the ADL, i.e. the modeler. He has to know (a) *when* to use pseudo-VPs (i.e. when to mimic the VP/Variants approach) and (b) *how* to do that.

The following figure illustrates problem no. 2 by comparing the use of variation points by way of dedicated modeling entities to mimicked variation points. The dashed arrows in red ink going from the (pseudo-)variants to the public feature model of the corresponding ADLFunction show how a direct reference, i.e. direct arrow is now replaced by a two-step reference via the public feature model of the pseudo-VP.

![Diagram](image)

**Figure 11.** Additional hierarchical level when mimicking variation points.

A detailed investigation of these issues reveals, that both problems can be solved quite straightforwardly:

Ad 1.: A concept might be introduced to allow ADLFunctionTypes without a public feature model and to hereby skip them in composition hierarchy with respect to variability. Lower-level variations can be defined in terms of higher-level functions’ public feature models.

Ad 2.: This might be solved with specific tool support for creating and editing pseudo-VPs (at least problem 2.(b) can be avoided this way).
### 3.2 Artifact-Level Variability II – Definition of When To Use Which Variant

While having examined in the previous section the model elements for defining what design variants are actually available during system configuration (called issue (A) above), we now turn to the means of expression for defining (B.1) when a certain variant is to be selected (e.g. "use X in U.S. and Y in EMEA market") and (B.2) how several variants can be combined (e.g. "X excludes Z").

In the coming section, several basic options for realizing the two issues (B.1) and (B.2) will be described, before a decision will be taken in Section 3.2.2.

#### 3.2.1 Basic Options

There are a multitude of options and alternatives for solving the issues (B.1) and (B.2) identified above. One thing, however, they have all in common: the information of when to use a certain variant and how several variants may be combined is formulated as a logical constraint of some sort which is then attached to one or more EAST-ADL2 modeling elements. Accordingly, the options and alternatives to be considered can be grouped into the following two categories:

1. Where to put the logical constraint?
2. What formalism to use for expressing this logical constraint?

Let us first examine the question of where to attach the logical constraint. There are two basic options: either we can attach the constraint to the contained ADLFunctionPrototypes which were marked optional or we attach them to the containing ADLFunctionType. In Figure 12 the first solution is illustrated with the two red logical expressions on the optional ControllerA and ControllerB referring to the public feature model of the WiperSystem; the second solution would lead to a similar illustration, but the expressions would be attached to WiperSystem as part of its configuration link which links its public feature model to the public feature models of the lower level ADLFunctions.

![Figure 12. Attaching constraints for (B.1) to the optional ADLFunctionPrototypes.](image)

In a situation as in the above figure, it does not make a big difference on which element the constraint is attached. When considering pseudo variation points, however, we must take into account that the optional ADLFunctionPrototypes are no longer directly contained in the ADLFunctionType “WiperSystem” but they are contained in some other ADLFunctionType as described above (cf. Figure 11). Attaching the logical constraint to the optional prototype now becomes a problem, because on the lower level it is not clear in which higher level
ADLFunctionType the pseudo variation point will be used (maybe it will even be used in several higher level ADLFunctionTypes) and therefore it is not clear to which public feature model the logical constraint may refer.

For item 2 above, i.e. the question of which formalism to use for expressing the logical constraints, the following solutions were proposed by the participants of WT3.3:

- OCL
- SysML-Constraints (cf. SysML-Blocks)
- InclusionCriteria (as defined for configuration decisions)
- new formalism

On a conceptual, methodological level, it does not make a big difference which formalism is applied. It is only important to find a formalism which can express all necessary cases and to not unnecessarily introduce anything new.

3.2.2 Discussion and Decision

The advantages and disadvantages of attaching the constraints to the prototype or the type can be summarized as follows:

- Attaching the constraints to the prototype seems more natural: when adding an optional sub-function to an ADLFunctionType it is more straightforward to directly define on this new element, when to use this optional sub-function.

- Attaching the constraints to the prototype leads to problems when skipping several hierarchical levels, i.e. when you want to directly define when to use an optional sub-function in terms of the public feature model not of the directly containing ADLFunctionType but of a type further up the containment hierarchy (as is the case with pseudo variation points).

Since the current review does not have the goal to introduce any new extensions to the EAST-ADL2 but only to consolidate and possibly simplify what is already there, a solution was targeted which is as simple as possible and does not introduce any additional formalisms and concepts.

Therefore the following solution is proposed here:

- (B.1) is realized within the configuration link defined for each ADLFunctionType. Currently, this configuration link only relates the type’s public feature model to the public feature models of all contained sub-functions. Now, it is also used to relate the type’s public feature model to the contained prototypes which were marked optional. These can be seen as implicitly defined optional features in this respect.

- To allow for skipping hierarchical levels (see above), each ADLFunctionType can publish a direct view on its contained optional prototypes.

- (B.2) is realized with variation groups which are annotated with a type that states the dependency implied on the contained elements. To realize more complex dependencies, variation groups can be nested (cf. Figure 6).

This solution can be implemented in the domain model quite straightforwardly, which is planned for January/February 2009.
3.3 Review of Other Variability Concepts

As described above, the other variability concepts are largely in a well-consolidated state. The following list summarizes the minor changes that have to be implemented in the EAST-ADL2 domain model with respect to these other concepts.

- In the entities related to Configuration Links, replace the old wording with “Product…” by the new terminology with “Configuration…”.
- Minor refinements of the entities realizing the multi-level concept (terminology).
- Check the base class of all entities (i.e. Class vs. Package).
- Check containment of entities for artifact level variability (i.e. containment in ADLFunctionType).
- Adapt documentation of domain model entities in EnterpriseArchitect to the changes described in this document.

These changes will be implemented into the domain model together with the more detailed changes described in the previous chapters in January and/or February in 2009.

NOTE: The above itemization does not comprise such refinements and extensions which were identified as major actions of WT3.3 in ATESST2, because they lie outside the limited scope of the initial review of EAST-ADL2 in WT3.3 in Q4 of 2008.
4 Conclusions

In this deliverable, a detailed review and discussion of EAST-ADL2 variability concepts was provided and several remaining issues were identified, especially regarding artifact-level variability management. In addition, possible solutions to these issues were described and discussed and the most promising solution was identified. The actual implementation of these solutions into the domain model of the EAST-ADL2 as well as the EAST-ADL2 tool platform is planned for Q1 and Q2 of 2009 respectively.
References

[1]