How to Define SES Trees for Variability Modeling

Christina Deatcu[1*], Hendrik Folkerts[1], Thorsten Pawletta[1], Umut Durak[2]

[1]Hochschule Wismar - University of Applied Sciences, Research Group CEA, Philipp-Müller-Straße 14, 23966 Wismar, Germany; **christina.deatcu@hs-wismar.de*

[2]Institute of Flight Systems, German Aerospace Center (DLR), Lilienthalplatz 7, 38108 Braunschweig, Germany

SNE 29(3), 2019, 117 – 126, DOI: 10.11128/sne.29.tn.10482
Received: May 10, 2019 (Selected ASIM SST Hamburg 2018 Postconf. Publ.), Accepted: June 10, 2019
SNE - Simulation Notes Europe, ARGESIM Publisher Vienna, ISSN Print 2305-9974, Online 2306-0271, www.sne-journal.org

Abstract. The System Entity Structure (SES) is a high level approach for variability modeling, particularly in simulation engineering, which is under continuous development. In this context, an enhanced framework is introduced that supports dynamic variability evolution using the SES approach. However, the main focus is to start a discussion about a set of design patterns, which were developed to analyze the tree design and computing aspects of System Entity Structures. As development of our MATLAB-based SES toolbox for construction and pruning of SES trees proceeded, the necessity to have some generalized examples for testing and verification came more and more into awareness. We propose a set of design patterns that, if completely representable and computable by a certain tool, support all aspects of SES theory. In addition, the patterns give users substantial support for developing SES models for other applications.

Introduction

This paper is a modified version of [1]. It details the specification of basic design patterns and introduces more advanced combined patterns.

Generally, variability modeling can be seen as an approach to describe more than one system configuration. According to Capilla, Bosch, and Kang [2], a software variability model has to describe the commonality and variability of a system at all stages of the software lifecycle. In software engineering, variability modeling is often closely associated with product lines. For software product lines the variability is described explicitly. Variation points are defined where different solutions can be derived.

Such variability mechanisms can be specified at different levels of abstraction, ranging from requirements specification to source code implementation. A popular high level approach is feature modeling by means of feature models, which were introduced as part of Feature-Oriented Domain Analysis by Kang et al. [3] and subsequently extended and used in various ways. An important further development of variability modeling has been the notion of variability in time, known as binding time in product line engineering [4]. That means that variability can be realized from design time to runtime.

In simulation engineering, the problem of variability modeling is well known from the eighties. One of the first high level approaches for variability modeling in the design phase was introduced with the System Entity Structure (SES) by Zeigler [5]. The objective was to describe a set of system configurations for a family of systems. An SES is represented by a tree structure, which describes a set of modular, hierarchical system structures, defines references to basic models in a model base (MB) and specifies various parameter settings for the referenced basic models. In addition, the approach defines several abstract transformation methods for deriving a particular system configuration and for generating an executable simulation model [6]. The entire approach was continuously further developed by Zeigler and many other researchers, such as in [7], [8], and [9].

As another approach for describing system structures and their configurations, various XML based composition schemes, which distinguish between interfaces and the concrete implementations of the models were proposed, such as by Röhl and Uhrmacher [10] or by Wang and Wainer [11]. Although that research does not explicitly address model families as the SES/MB approach does, some of the basic ideas are similar. Moreover, the ideas in [10, 11] are important for the design of reusable model components and their organization in an MB. In this paper, the reusability of components in an MB is not discussed.

It focuses on modeling system variability using the SES and the necessary software framework.

The problem of variability at runtime is known as variable or dynamic structure system modeling and simulation. Analogous to the approaches in software engineering, such a dynamic variability can be described on the level of a specific model, such as introduced by Barros in [12], or separated using a higher level model abstraction in conjunction with an appropriate software framework. Regarding this, a first theoretical approach for dynamic variability modeling using the SES/MB method was published in 1990 [13].

Based on this early idea, a prototype of a full SES/MB based modeling and simulation infrastructure has been developed and implemented within MATLAB/Simulink by the authors [14, 15], and a Python implementation is in progress. In addition, the SES theory has been enhanced. In this context, the necessity to have some generalized SES patterns came more and more into awareness. On the one hand patterns are helpful during software development and on the other hand the patterns are expected to give users substantial support for developing SES models for their applications.

After a short overview to the enhanced SES ontology and the enhanced SES/MB based infrastructure, basic design patterns for variability modeling are introduced. The description is related to the modeling capabilities provided by feature models. Then, some combined patterns are discussed exemplary to give an impression for advanced variability modeling possibilities using the SES and its extensions. Finally, the main results are summarized and an outlook to future work is given.

1 Background

According to Zeigler and Hammonds, the SES is an ontology, a language with syntax and semantics to represent declarative knowledge [8]. It is particularly suitable for describing system configurations for different application domains. An SES is represented by a directed tree structure. Objects are represented by nodes which are connected by edges. There are four node types with different properties describing the objects and their relations. Furthermore, there are axioms for defining the SES correctly. Since an SES describes a number of system configurations, the SES tree needs to be *pruned* to get one particular configuration, which is called Pruned Entity Structure (PES).

The classic SES theory was extended by several researchers over the last decades. In [14] and [16] the SES theory was extended with a procedural knowledge representation. Some of these extensions are used in this paper. A comprehensive example on how the pruning patterns proposed in this paper can be used, is demonstrated in [17].

1.1 Node Types

Among the four node types, there are two groups, the entity nodes and the descriptive nodes. Entity nodes describe objects of the real or the imaginary world. The root and the leaves of an SES tree are always entity nodes. Relations between the entity nodes are specified by descriptive nodes.

Descriptive nodes are the genus for aspect nodes, specialization nodes and multi-aspect nodes. Aspect nodes (name suffix *DEC*) describe how entity nodes can be decomposed in partial entities whereas the taxonomy of an entity is described by specialization nodes (name suffix *SPEC*). Multi-aspect nodes (name suffix *MASP*) are a special case of an aspect node with all children being of the same kind.

Each node or edge can have attached variables, also called attributes. For entity nodes, the variables represent properties of the respective object whereas the variables at descriptive nodes specify relations between their parent node and children nodes or decisions for the pruning process. With the extended procedural knowledge representation, values of attached variables can be assigned dynamically.

1.2 Axioms

The semantics of the SES are defined by axioms. The types of the nodes have to follow the axiom *alternating mode*. Every entity node has to be followed by a descriptive node, and vice versa. A *strict hierarchy* is needed. In every path of the tree, a name of a node may occur only once. If nodes in different paths have the same name, they need to have the same variables and isomorphic partial trees. This is called *uniformity*. Nodes on the same level of hierarchy and having the same father, called sibling nodes, have to be *valid brothers*, meaning that sibling nodes must not have the same name. The axiom of *attached variables* implies that a node must not have variables of the same name. The axiom of *inheritance* implies, that during pruning, the parent and the child of a specialization combine their attributes.

If parent and child have the same attributes, the parent's attributes are overwritten with the child's attributes and their values.

1.3 Extended SES/MB Infrastructure

The SES describing a set of system designs has been associated with the idea of model generation of modular, hierarchical systems from the very beginning [6] which led to the SES/MB approach. Each system design is defined by its system structure and parameter configuration in the SES. The core assets of all system variants are specified as a set of configurable basic models, which are organized in a Model Base (MB). The classic SES/MB framework defines a set of transformation methods for generating executable simulation models, but automated model generation is not provided. To allow automated generation and execution of models, the SES/MB approach has been extended ([14], [15], [16]). These extensions make the SES/MB approach more pragmatic for implementation and to be used in a simulation infrastructure.

Figure 1 depicts the extended SES/MB infrastructure consisting of the SES/MB framework, an *Execution Unit*, and an *Experiment Control*. Although the SES/MB approach and its extensions are usually considered in connection with the generation of simulation models, they are generally applicable to modular-hierarchical structured software systems.

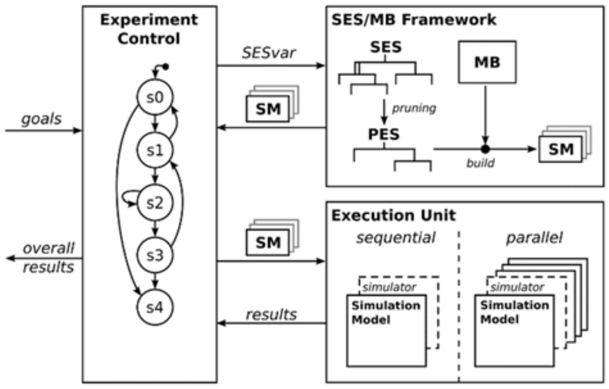

Figure 1: Extended SES/MB-based infrastructure.

Operations. On the SES, a *merge operation* is defined allowing two or more SES to be combined. This allows the quick reuse of a once defined SES. The essential operation on the SES is the *pruning method*. To extract one particular system structure and configuration, the SES needs to be trimmed to a PES. During the pruning process, decisions have to be taken at descriptive nodes.

Therefore, rules need to be defined at aspect, multi-aspect and specialization nodes. The *specialization rule* (specrule) associated with a specialization node determines which child entity shall be selected. *Aspect rules* (aspectrule) associated with aspect or multi-aspect nodes on the same hierarchy level determine which of the siblings is to be chosen. Furthermore, cross-tree relations can be expressed by *selection constraints*. Selection constraints can be used to select a certain entity based on decisions taken anywhere else in the SES tree. Next to the pruning method, another transformation method is the *build method*. With the help of the build method, an executable model can be built from a PES and basic models organized in an MB. The basic models are specific for a certain simulation software. Therefore, the build method needs to match to the simulator used.

Execution Unit and Experiment Control. For automated and reactive processing of SES models, an execution unit and an overall experiment control unit are added to the framework, as depicted in Figure 1. For automatic generation of different PES, leading to different simulation models, an interface to the SES is needed. This interface can be established by global variables of the SES, called *SES Variables* (SESvar), which can affect the decisions taken in descriptive nodes during pruning. Thus, a particular system configuration derived from an SES depends on the current settings of the SES variables. The value range of SES variables can be limited by defining *semantic conditions*, which are checked before pruning to exclude certain system configurations. By assigning values to the SES variables, the experiment control determines the order and system configurations of executable simulation models (SM) to generate from the SES with the pruning and build operations. Thereby different variants of the executable simulation models are generated. The experiment control then transmits the SM to the execution unit. The execution unit links the generated simulation model to the simulator, executes a simulation run and, finally, sends the results back to the experiment control. The results, in turn, can influence the decision of the experiment control on how to assign the SES variables next.

Special Attributes. Combining basic models from the MB leads to the creation of *coupled models*. In order to describe the structure of the executable model, some nodes need to define *couplings*. Couplings are properties of descriptive nodes of the type aspect and multi-aspect and consist of pairs of entity names and port names. Figure 2 gives an impression of what a definition of couplings may look like. Furthermore, for a multi-aspect node, a special variable, *numRep*, has to be defined representing the number of children to generate when pruning this node. To specify the basic model from the MB an entity node refers to, the *mb-attribute* is introduced. This special attribute is permitted just for leaf nodes. Finally, for some cases, it is necessary to define priorities for supporting decisions among descriptive nodes on the same level of hierarchy in the *priority attribute*. All values of attributes can be defined by constants or set via SES variables or SES Functions.

SES Functions. The concept of *SES Functions* (SESfcn) has been introduced to specify complex variability within node attributes with minimal effort and to keep a lean SES tree. Typical examples include the definition of varying coupling relations, varying port numbers of systems or the definition of variable parameter configurations in attributes. During pruning, SES functions are evaluated, often with SES variables as input parameters. For effective coding of SES functions, the implicit attributes *parent* and *children* are introduced for each SES node. They encode the parent and children node names, respectively.

1.4 Software Tools for the Extended SES/MB Infrastructure

In the Research Group CEA, a prototype tool for the SES/MB infrastructure was developed, *The SES Toolbox for Matlab/Simulink* [15]. Currently, SES trees can be defined via a graphical user interface and a concrete variant can be extracted by pruning. The toolbox supports the modeler with plausibility test during SES construction, graphical representation of the SES, automatic generation of HTML documentation, and other features. The pruning process can be started from the graphical user interface and, in addition, is implemented to function automatically. Automatic pruning is necessary when using an SES constructed with the toolbox together with the experiment control. Furthermore, there is a prototype Matlab function implementing a build method for the simulation software Simulink, including SimEvents and Simscape, the MatlabDEVS toolbox [18], and for Modelica models. The SES is linked to the appropriate MB with the special mb-attribute of the leaf entity nodes.

Another software tool based on Python3/PyQt5 is under development. The aim is to be more independent from a computing environment and to support a greater number of simulators for building executable simulation models.

2 Basic Design Patterns

A few elementary design patterns that are required for design, implementation, and test of a pruning algorithm for an SES are presented in the following subsections. The proposed patterns for modeling of system structures are necessary, particularly if one aims to use the SES tool for automated model generation in the context of the extended SES/MB approach. In analogy to the semantics of feature models ([2], [3]) and mathematical logical expressions, we try to classify the first patterns according to their purpose.

In the context of feature modeling, four kinds of features are used: (i) mandatory features (logical AND), (ii) alternative features (logical XOR), (iii) optional features and, (iv) OR-features (logical OR). At first sight, it seems to be obvious and simple to decide which SES constructs can be used to define tree sections satisfying the feature categories above, but there is often more than one way to express the logical relations among tree sections, components or entities. However, the simplest patterns consist of just one descriptive node with its parent and children.

Besides patterns fitting into the classification according to feature modeling, we identified some patterns useful to illustrate and test SES tools dealing with issues like inheritance, the special attributes, such as couplings and priorities, and evaluation order.

2.1 Mandatory Tree Sections

Mandatory are those sections which need to be existent in each system belonging to a family of systems. For an SES describing a family of systems, this means that all possible PES representing a certain system variant will also include those parts. The corresponding logical expression is the AND and we can call the linkage a '*has-a*' relationship. Design patterns of this type are the aspect node itself, a multi-aspect node or specialization siblings.

Design Pattern #1 - Aspect Node. Figure 2 depicts the simplest case of a design pattern for mandatory sections, the aspect node itself.

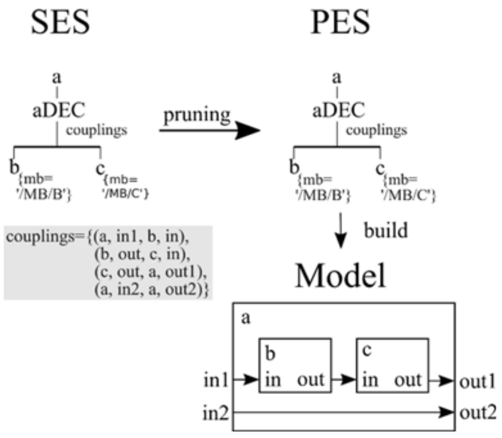

Figure 2: First design pattern for mandatory sections.

Each coupled system *a* consists of an entity *b* and an entity *c*. For model generation, aspect nodes need to define the special attribute for couplings, while the leaf nodes have the mb-attribute attached. Note that the derived PES is identical to the SES. The resulting abstract, simulator independent model on the right side is given to illustrate what kind of model structure would result from the PES. The following examples will not include detailed coupling definitions and models since couplings can be formulated analogously and concrete models depend on the chosen simulation environment.

Design Pattern #2 – Multi-Aspect Node. In Figure 3, a similar case to design pattern #1 is given. With this pattern we introduce the usage of SES variables, SES functions and the possibility to define multi-sets for setting values of attributes. Since the system *as* consists of a certain number of children of type *b*, multi-aspect nodes need to define two special attributes, the attributes numRep and couplings. When pruning a multi-aspect node, the numRep attribute is evaluated and an aspect node with children of the same type is created. In this example, the numRep attribute is defined by the current value of the SESvar VAR which is restricted to two or three by the semantic condition, but the value could also be hardcoded. Based on the number of children the couplings may vary, too. For an effective coding and to keep a lean SES tree, couplings are set via an SESfcn here. Based on the value of the SESvar VAR either cpl1 or cpl2 is chosen. The structure of cpl1 and cpl2 needs to be defined as it is presented for static couplings in Figure 2.

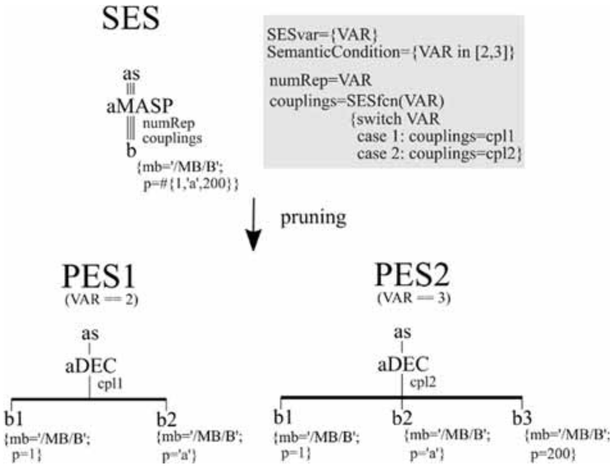

Figure 3: Design pattern for mandatory sections with a multi-aspect node.

The attribute p is set by dint of a multi-set variable. Although after pruning all children of *as* are of the same type *b* and are referencing the same basic model B in the MB, their parametrization can be different. Children's names are generated by appending a number to the entity name *b* to ensure that the resulting siblings fulfill the axiom of valid brothers.

Design Pattern #3 - Specialization Siblings. If two or more specialization nodes are on the same hierarchy level, they will all be evaluated. For the pattern depicted in Figure 4, this means that *a* will specialize into one of *b* or *c* AND into one of *d* or *e*. Which child of the specialization is taken depends on the values of the SES variables and the specialization rules.

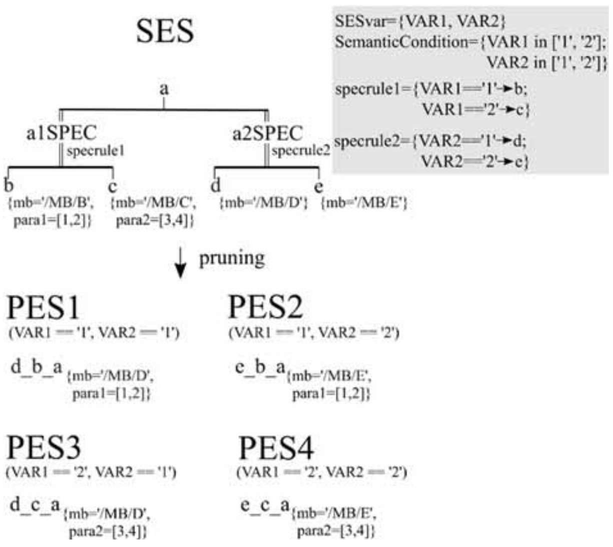
SES
SESvar={VAR1, VAR2}
SemanticCondition={VAR1 in ['1', '2']; VAR2 in ['1', '2']}
specrule1={VAR1=='1'→b; VAR1=='2'→c}
specrule2={VAR2=='1'→d; VAR2=='2'→e}
a
a1SPEC
specrule1
a2SPEC
specrule2
b {mb='/MB/B', para1=[1,2]}
c {mb='/MB/C', para2=[3,4]}
d {mb='/MB/D'}
e {mb='/MB/E'}
pruning
PES1 (VAR1 == '1', VAR2 == '1')
d_b_a {mb='/MB/D', para1=[1,2]}
PES2 (VAR1 == '1', VAR2 == '2')
e_b_a {mb='/MB/E', para1=[1,2]}
PES3 (VAR1 == '2', VAR2 == '1')
d_c_a {mb='/MB/D', para2=[3,4]}
PES4 (VAR1 == '2', VAR2 == '2')
e_c_a {mb='/MB/E', para2=[3,4]}

Figure 4: Design pattern for specialization siblings.

If, for example, VAR1 is set to 1 and VAR2 is set to 2, at specialization *a1SPEC* the left child *b* is selected and, at specialization *a2SPEC*, the right child *e* is selected. During pruning, it depends on the pruning algorithm which of the two specializations is evaluated first. For this example, we assume that the left specialization node *a1SPEC* is evaluated first. The evaluation order influences what is the resulting value of the mb-attribute since, according to the inheritance axiom, attributes with the same name are overwritten in the parent node.

With this example, we also demonstrate the use of SES variables for defining specialization rules.

2.2 Alternative Tree Sections

Alternative are those sections from which exactly one part needs to be existent in each system belonging to a family of systems. A system described by a PES is of the type which is selected in a specialization or consists of the children of aspects chosen at aspect siblings. The corresponding logical expression is the XOR and we can call the linkage an '*is-a*' relationship. Design patterns of this type are the specialization node itself, aspect siblings, multi-aspect siblings or siblings containing an aspect node and a multi-aspect node.

Design Pattern #4 - Specialization Node. The simplest pattern to specify alternatives is the specialization node shown in Figure 5. Based on the specialization rule, exactly one child needs to be selected to construct a valid variant. The specialization rule evaluates the SESvar VAR to allow a selection. After pruning, the resulting system can either be of type *b* OR of type *c*. Valid names are constructed by preceding the fathers name *a* with either *b* or *c* and attributes are inherited as defined by the SES axioms.

Design Pattern #5 - Aspect Siblings. If two or more aspect nodes are on the same hierarchy level, exactly one of them has to be selected by evaluating the aspect rules of the aspect brothers.

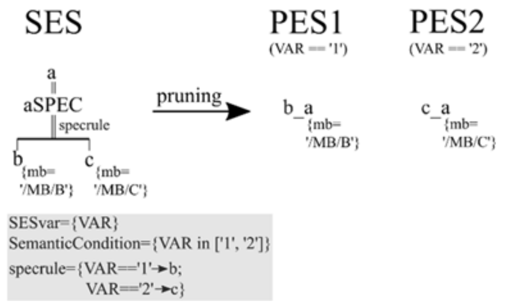

Figure 5: A simple specialization node.

This is presented in Figure 6. The system *a* consists either of *b* and *c* or of *d* and *e*. The aspect rules one and two define which branch is selected during pruning.

Design Pattern #6 – Multi-Aspect Siblings. In the pattern shown in Figure 7, two multi-aspect nodes are on the same layer. In the first pruning step, the multi-aspect nodes are resolved leading to two aspect nodes. After this step, the resulting intermediate PES can be finally resolved using pattern #5 for aspect siblings, as previously described. The system *a* consists either of *b1*, *b2*, and *b3* or of *c1* and *c2* depending on which child is selected based on the aspect rules.

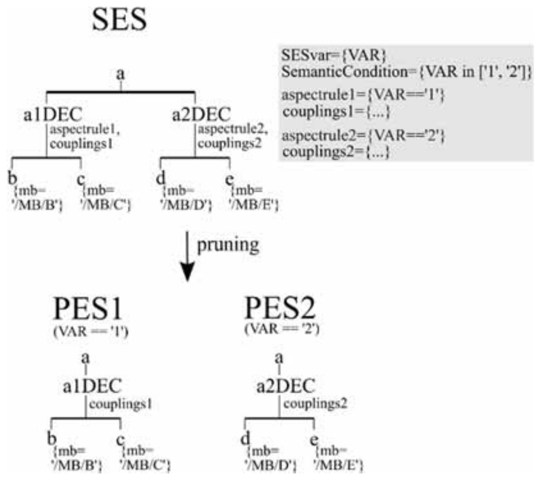

Figure 6: Aspect siblings result in an alternative selection.

Design Pattern #7 - Aspect and Multi-Aspect Siblings. If there are more than one aspect nodes and multi-aspect nodes on the same hierarchy level, the behavior is like aspect siblings after the multi-aspect node is resolved (see Figures 6 and 7).

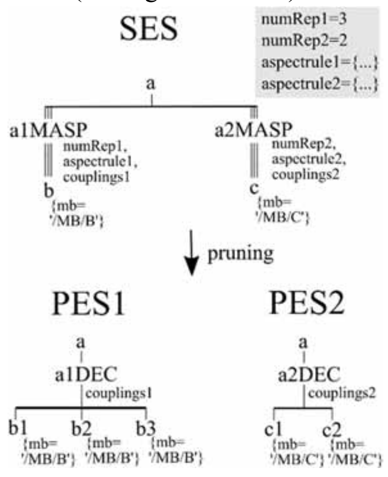

Figure 7: Multi-aspect siblings behave like aspect siblings.

2.3 Optional Tree Sections - Design Pattern #8

Optional sections can be contained in the resulting system structure, but they do not have to be. This can be done using an extension of the SES, the NONE element. A NONE element for a leaf entity node means that, if the NONE branch is selected, the entity is not included at all. In the pattern shown in Figure 8, the specialization has a child which is a NONE element. Hence, this SES can evaluate to NONE during pruning based on the specialization rule. The system *a* can either be of type *b* or not existent at all.

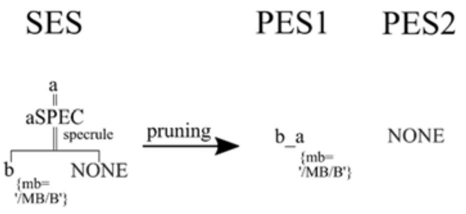

Figure 8: Optional sections expressed by specialization nodes with the NONE element.

2.4 OR Tree Sections - Design Pattern #9

Logical OR means that one or more entities or tree sections need to be included to get a valid variant. This can be expressed by an aspect node whose children are followed by specializations. Each specialization contains a NONE element as one child. The corresponding design pattern is shown in Figure 9. At least one specialization has to evaluate to a node not being NONE. By defining the specialization rules reasonably, the user has to ensure, that this is guaranteed. This pattern is composed by pattern #1 describing mandatory tree sections in combination with pattern #8 for optional elements. After pruning, the system *a* consists of *b* and *c*. System *b*, in turn, is of type *bs* or not existent while *c* is of type *cs* or not existent.

Couplings have to be adjusted when the tree changes by evaluating nodes. Since the couplings are defined at the aspect node *aDEC*, it is obvious, that there is a need for the possibility to define variable couplings. Variable couplings can be defined via SES Functions as introduced with design pattern #2.

3 Combined Design Patterns

In the previous section, the elementary design patterns were discussed in particular. During tool development, the necessity of testing combinations of basic patterns (combined patterns) was recognized.

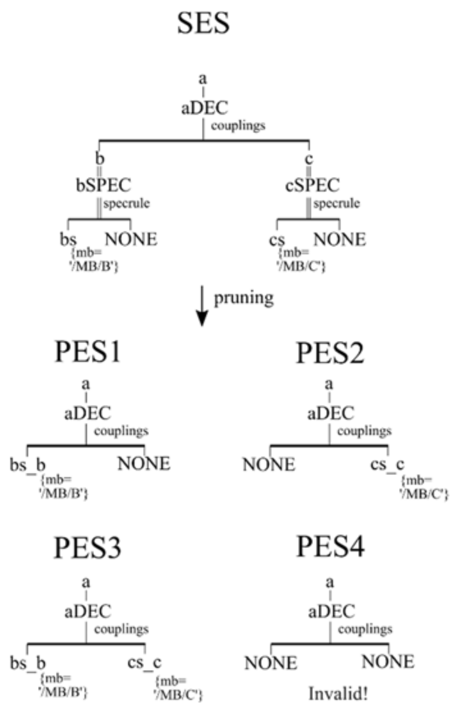

Figure 9: OR expressed by aspect nodes followed by specializations with NONE elements.

Issues addressed with these patterns are inheritance, evaluation order and priorities. In the next sections, a few combined patterns are introduced.

3.1 Design Pattern #10 - Two Specialization Nodes in One Path

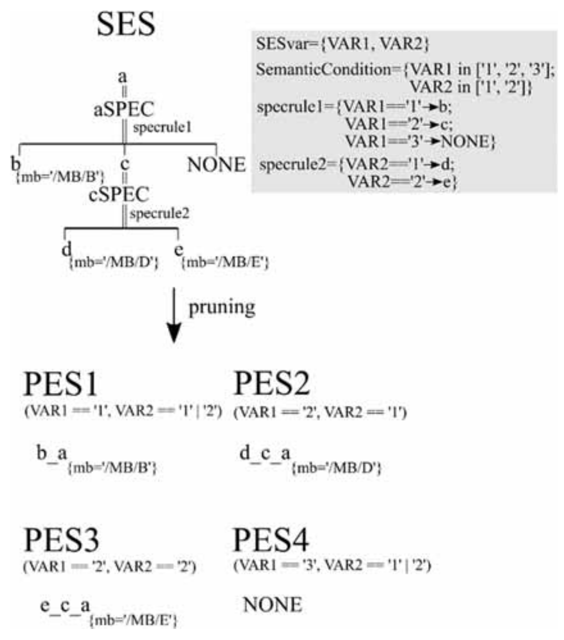
SES
SESvar={VAR1, VAR2}
SemanticCondition={VAR1 in ['1', '2', '3']; VAR2 in ['1', '2']}
specrule1={VAR1=='1'→b; VAR1=='2'→c; VAR1=='3'→NONE}
specrule2={VAR2=='1'→d; VAR2=='2'→e}
a
aSPEC
specrule1
b {mb='/MB/B'}
c
NONE
cSPEC
specrule2
d {mb='/MB/D'}
e {mb='/MB/E'}
pruning
PES1 (VAR1 == '1', VAR2 == '1' | '2')
b_a {mb='/MB/B'}
PES2 (VAR1 == '2', VAR2 == '1')
d_c_a {mb='/MB/D'}
PES3 (VAR1 == '2', VAR2 == '2')
e_c_a {mb='/MB/E'}
PES4 (VAR1 == '3', VAR2 == '1' | '2')
NONE

Figure 10: Inheritance of attributes at two specialization nodes in one path.

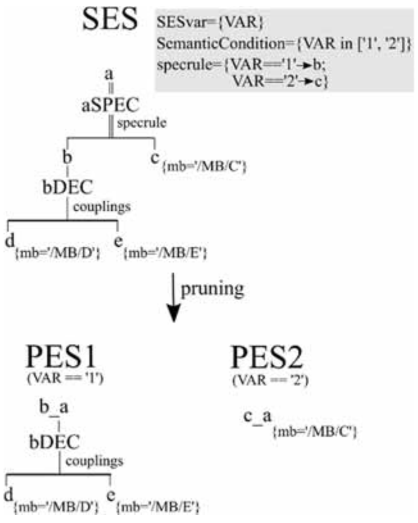

Figure 11: A specialization with a succeeding aspect.

Specialization nodes inherit the attributes of the selected child and append them to the father's attributes, as previously described. In Figure 10, there are two specialization nodes in one path. Additionally, the NONE element is used. Thus, this can be classified as an optional feature and shall clarify the axiom for attribute inheritance. During pruning, firstly *aSPEC* is evaluated. In case the child *c* is selected, another decision for child *d* or child *e* is taken. In that case, the attributes of the child node of *cSPEC* are inherited to the first node.

3.2 Design Pattern #11 – Specialization with Succeeding Aspect

Figure 11 depicts a specialization node with a succeeding aspect node. This pattern is a combination of a specialization node followed by a single aspect node. Therefore, it is a combination of alternative and mandatory tree sections. The system *a* can be of type *b* or *c*, the *b* can be decomposed in *d* and *e*.

3.3 Design Pattern #12 - Specialization and Aspect Siblings

When aspect nodes and specialization nodes are brothers, the specialization node has to be resolved first during pruning. If, additionally, an aspect node is below the specialization node, during pruning two aspect nodes will become siblings. Since, in this case, the occurrence of aspect siblings is not known until the first pruning step, aspect rules could not be formulated beforehand. In order to tackle this, the priority attribute for aspect and multiaspect nodes was introduced. Throughout the whole SES, these nodes get a unique number. If, during pruning, aspect nodes become brothers, a decision about which to choose can be made.

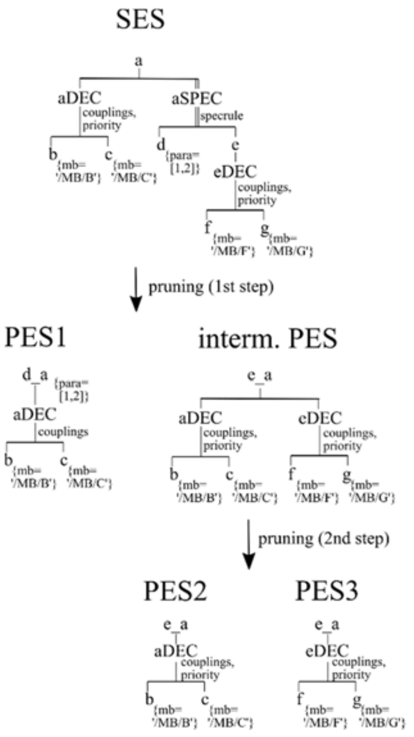

Figure 12: Aspects can become siblings by resolving a specialization.

In Figure 12, an example is given. The system *a* consists of *b* and *c*, if it is of type *d*. If it is of type *e*, it can consist of *b* and *c* or of *f* and *g*. In case the system is of type *e*, a decision as to which decomposition to take is made by interpreting the priority attribute.

3.4 Design Pattern # 13 - Several Multi-Aspects in a Path

If a multi-aspect node is followed by a second or even more multi-aspects, complexity of resulting structures grows considerably. During pruning compliance with the SES axioms has to be ensured and therefor some renaming operations are necessary.

Figure 13 depicts two successive multi-aspects where the numRep variable of *bMASP* defines a multiset. Renaming of *b* to *b1* and *b2,* as necessary for children of multi-aspects and explained in pattern #2, results in renaming of *bMASP* to *b1MASP* and *b2MASP*. Generally, one can say that renaming an entity node always calls for renaming the following descriptive node, too. The first pruning step resolves the multi-aspect *aMASP* to an aspect *aDEC* with two child nodes of type multi-aspect. In a second step the child nodes are resolved to aspect nodes, too. Since the numRep variable of *b1MASP* and *b2MASP* was set via the multiset, variable couplings are necessary.

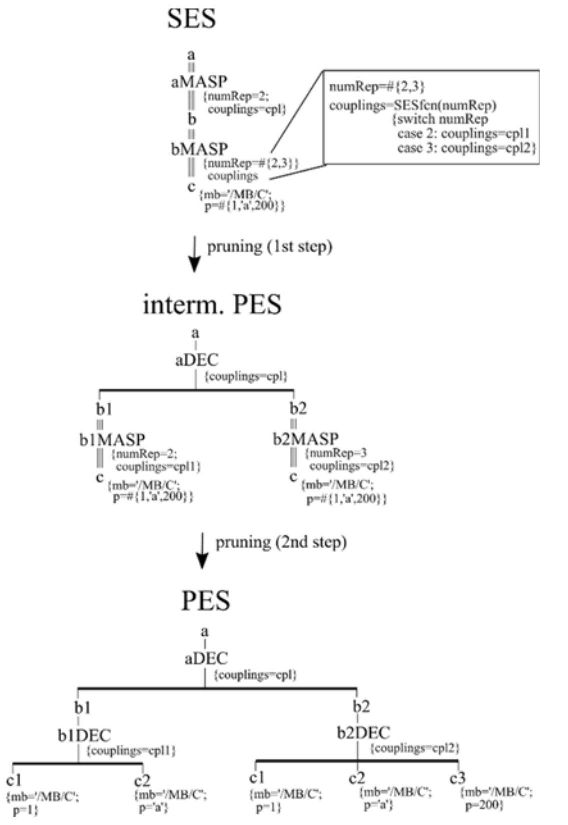

Figure 13: Successive multi-aspects with variable number of replications.

Couplings are set with the help of an SESfcn which chooses cpl1 or cpl2 based on the current value of numRep.

3.5 Design Pattern #14 – Selection Constraints

A modification of pattern #3 (specialization siblings) is shown in Figure 14. With this pattern the use of selection constraints and semantic conditions to limit the possible valid structures is pointed out. The corresponding constructs in feature models are known as *require* and *exclude* [2].

The dotted line from leaf node *c* to node *d* in Figure 14 depicts a selection constraint which means that, if *c* is selected at *a1SPEC*, *d* has to be chosen at *a2SPEC*. The resulting PES is PES3. Note, that current value of VAR2 does not matter for the selection, if VAR1 is set to 2.

Another way to control possible variants are the semantic conditions. The semantic conditions for value combinations of VAR1 and VAR2 interdict the selection of *d*, if *b* is already selected.

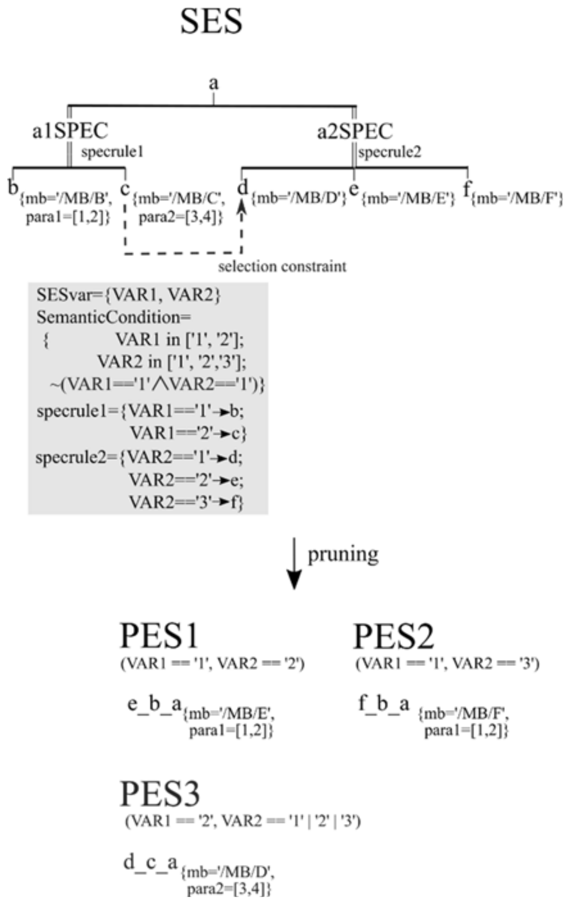

Figure 14: Variant restriction with selection constraints and semantic conditions.

4 Conclusion and Future Work

This paper gives an overview of essential patterns needed to describe and test the pruning process of an SES. With the described patterns, the implementations and tests of the software tools for the extended SES/MB infrastructure developed in the Research Group CEA could be done.

However, while developing and testing the software tools, more combined patterns were found and tested. Some pitfalls in the pruning algorithm were discovered. Therefore, additional attributes, such as the priority attribute, were introduced.

However, e.g. for specialization siblings, there has to be found a way to decide which sibling is evaluated first. Formalized definitions of pruning algorithms are essential part of future work.

Future works regarding the SES software tools will especially cover enhancements of the model builder. For a full software support of the SES/MB-based infrastructure, it is necessary to have a fully functional model builder, which supports the generation of hybrid models with basic systems that are modeled using different paradigms. In addition, the Python implementation will allow to support a set of model builders for different simulators.

References

[1] Deatcu, C., Folkerts, H., Pawletta, T., and Durak, U.. *Design Patterns for Variability Modeling Using SES Ontology*. SpringSim-Mod4Sim 2018, April 15-18 2018, Baltimore, MD, USA, Society for Modeling & Simulation International (SCS), pp. 528-539. (DOI: 10.22360/SpringSim.2018.Mod4Sim.004)

[2] Capilla, R., J. Bosch, and K. C. Kang (Editors). *Systems and Software Variability Management*. Springer, Berlin Heidelberg, Germany, 2013.

[3] Kang, K. C., S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report, Software Engineering Institute Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[4] Capilla, R., and J. Bosch. *Binding Time and Evolution*. In Systems and Software Variability Management, edited by R. Capilla, J. Bosch, and K. C. Kang, pp. 57-73. Springer, Berlin Heidelberg, Germany, 2013.

[5] Zeigler, B. P.. *Multifaceted Modelling and Discrete Event Simulation*. Cambridge, Academic Press, 1984.

[6] Zeigler, B. P., T. G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. 2nd ed. San Diego, CA, USA, Academic Press, 2000.

[7] Rozenblit, J. W., and B. P. Zeigler. *Representing and Constructing System Specifications Using the System Entity Structure Concepts*. Proceedings of the 1993 Winter Simulation Conference, pp. 604-611.

[8] Zeigler, B. P., and P. E. Hammonds. *Modeling and Simulation-Based Data Engineering*. San Diego, CA, USA, Academic Press, 2007.

[9] Zeigler, B. P., and H.S. Sarjoughian. *Guide to Modeling and Simulation of Systems of Systems*. Springer, London, UK, 2013.

[10] Röhl, M., A. M. Uhrmacher. *Composing simulations from XML-specified model components*. Proceedings of the 2006 Winter Simulation Conference, pp. 1083-1090.

[11] Wang, S., G. A. Wainer. *Semantic selection for model composition using SAMSaaS*. Proceedings of the 2015 Spring Simulation Conference - TMS-DEVS, pp. 25-32.

[12] Barros, F. J.. *The Dynamic Structure Discrete Event System Specification Formalism*. SIMULATION: Transactions of The Society for Modeling and Simulation International. 13(1996)1, pp. 35-36.

[13] Zeigler, B. P., and H. Praehofer. *System Theory Challenges in the Simulation of Variable Structure and Intelligent Systems*. Computer Aided Systems Theory — EUROCAST '89. EUROCAST 1989. Lecture Notes in Computer Science, edited by F. Pichler and R. Moreno-Diaz., vol 410, pp. 41-51. Springer, Berlin, Heidelberg, 1990.

[14] Pawletta, T., A. Schmidt, B. P. Zeigler, and U. Durak. *Extended Variability Modeling Using System Entity Structure Ontology within MATLAB/Simulink*. Proceedings of the 2016 Spring Simulation Conference – ANSS, pp. 62–69.

[15] RG CEA. *The SES Toolbox for MATLAB / Simulink Website*. http://www.cea-wismar.de/tbx/SES_Tbx/sesToolboxMain.html. Accessed Nov. 22, 2017.

[16] Schmidt, A., U. Durak, and T. Pawletta. *Model-Based Testing Methodology Using System Entity Structures for MATLAB/Simulink Models*. SIMULATION: Transactions of The Society for Modeling and Simulation International. 92(8), pp. 729-746, 2016.

[17] Folkerts, H., T. Pawletta, C. Deatcu, and U. Durak. *Variability Modeling for Engineering Applications*. Simulation Notes Europe vol. 27(4),2017, pp. 167–176.

[18] RG CEA. *The PDEVS Toolbox for MATLAB Website*. http://www.cea-wismar.de/tbx/DEVS_Tbx/MatlabDEVS_Tbx.html. Accessed Nov. 29, 2017.