University of Applied Sciences Wismar
University of Applied Sciences
Technology, Business and Design
Research Group Computational Engineering and Automation

Documentation

SESToPy - SES Tools Python

A Python Toolbox for Creating and Managing System Entity Structures

as of: March 27, 2021

by: Hendrik Folkerts / Research Group CEA

Contents

1 Extended SES/MB Infrastructure

Figure 1.1 depicts the extended System Entity Structure (SES)/Model Base (MB) infrastructure consisting of the extended SES/MB framework, an execution unit (EU), and an experiment control (EC).
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 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 an MB. The classic SES/MB framework defines a set of transformation methods for generating executable simulation models. Model generation is only provided in an interactive way. To support an *automated* generation and execution of models, the SES/MB approach has been extended, called the extended SES/MB (eSES/MB) approach. These extensions make the SES/MB approach more pragmatic and usable in a simulation infrastructure for engineering problems.
Although the eSES/MB is usually considered in connection with the generation of simulation models, it is generally applicable to modular-hierarchical structured software systems. Subsequently, the components of the eSES/MB infrastructure are discussed.

1.1 SES/MB Basics and eSES/MB Extensions

According to Zeigler and Hammonds, the SES is an ontology, a language with syntax and semantics to represent declarative knowledge. It is 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. The tree consists of one root node, followed by inner nodes and leaf nodes on the layers below. 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. The result of pruning is a

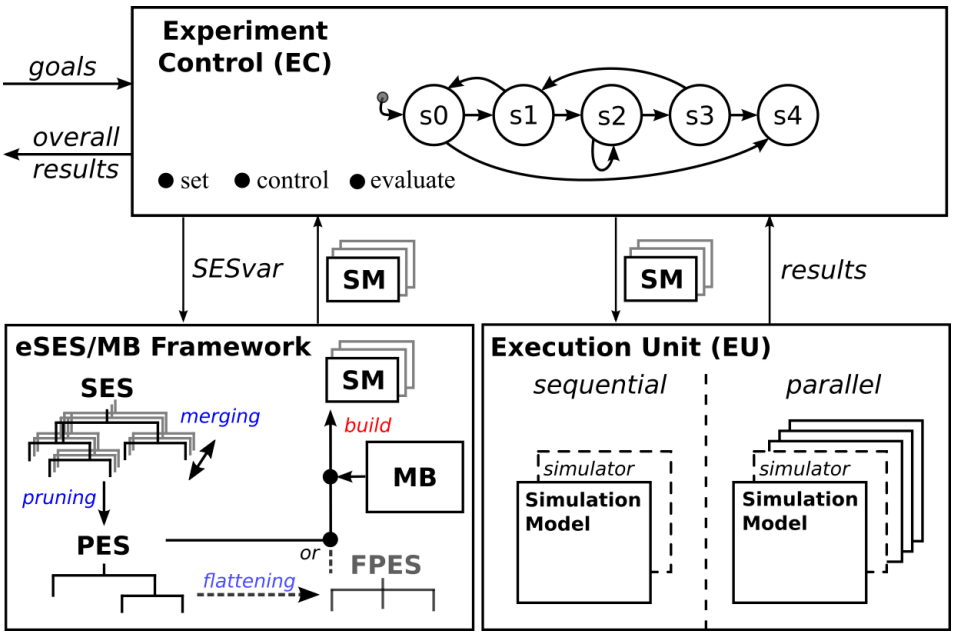


Figure 1.1: Extended SES/MB infrastructure.

reduced tree structure, called Pruned Entity Structure (PES).
The classic SES theory was extended by several researchers over the last decades. In the Research Group Computational Engineering und Automation (CEA) at the University of Applied Sciences Wismar in particular the SES/MB approach was extended by: (i) a procedural knowledge representation, (ii) a special linkage between SES leaf nodes and models in an MB, and (iii) new SES elements and software components to support the *automated* generation and an *automated* execution of simulation experiments. A brief summary of some of the key principles is given below.

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 describe how entity nodes can be decomposed into partial entities whereas the taxonomy of an entity is described by specialization nodes. Thus, aspect nodes describe a *has-a* relationship, whereas specialization nodes describe an *is-a* relationship. Multi-aspect nodes are a special case of an aspect node with all children being of the same kind.
Entity nodes can have attached variables, that represent properties of the respective

object. Additionally, the eSES/MB approach allows to define specific information at descriptive nodes. This information specifiesrelations between the descriptive node's parent node and children nodes or decisions for the pruning process. All node-specific data in the eSES/MB approach is referred to as attributes and attributes can also be defined conditionally.

Axioms The semantics of the SES are defined by axioms. The types of the nodes in a tree path 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 is allowed to occur only once. If nodes in different paths have the same name, they need to have the same attributes and isomorphic partial trees. This is called *uniformity*. Nodes with 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.

Special Attributes and Selection Constraints Specifying a composition of entities, which describes the composition of models in a modular-hierarchical manner, requires a specification of *coupling* relations. Couplings can be specified as attributes of descriptive nodes of the type aspect and multi-aspect and consist of pairs of entity names and port names, such as (source entity, source port, sink entity, sink port). In chapter 4 there are examples to get an impression on how to use couplings. Furthermore, for a multi-aspect node, a special variable, called *number of replications* (*numRep*), has to be defined determing the number of children to generate when pruning this node. For pruning rules can 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 can be used to decide 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. To specify the linkage between an SES node and a basic model in the MB, the *mb-attribute* is introduced. This special attribute is permitted just for leaf nodes. 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 as discussed in the next section. A special kind of attributes are underscore attributes. An entity node, which is a child of a multi-aspect node gets an underscore _NameOfTheNode attribute for node specific information.

SES Variables and SES Functions The concept of SES variable (SESvar) and SES function (SESfcn) has been introduced to specify complex variability within node attributes with minimal effort and to keep a lean SES tree. SESvars have a global scope and are used as input variables or auxiliary variables of an SES. The value range of SESvars can be limited by semantic conditions. Typical examples for SESfcns include the definition of varying coupling relations, varying port numbers of systems or the definition of variable parameter configurations in attributes. During pruning, SESfcns are evaluated, often with SESvars as input parameters. For effective coding of SESfcns, the implicit attributes *parent* and *children* are introduced for each SES node. They encode the parent and children node names, respectively.

Operations As depicted in Figure 1.1, there are four operations defined for the eSES/MB framework. On the SES, a *merge* operation is defined allowing two or more SES to be combined. This allows the quick reuse of an once defined SES and supports a modular problem specification on the SES level. 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 on the basis of the rules defined in their attribute. Furthermore any node specific variables and attributes are assigned. Next to the pruning method, other transformation methods are the *flattening* and the *build* methods. Applying the flattening method to a PES, all inner nodes are removed. Three layers beginning with the root node, followed by an aspect node and all leaf nodes in one layer are left. The resulting tree structure is called a Flattened Pruned Entity Structure (FPES). With the help of the build method, an executable model can be built from a PES or an FPES and basic models organized in an MB. The basic models are specific for a certain simulation software. Therefore, the build method needs to match the simulator used in the EU according to Figure 1.1.

1.2 Execution Unit and Experiment Control

The EU and EC in the eSES/MB approach are introduced for automated processing and experimentation with the set of models defined by an SES. For automatic generation of different PES, leading to a set of executable simulation model (SM), an interface to the SES is needed. This interface is established by the SESvars, which 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 SESvars. By assigning values to the SESvars, the EC determines the order and system configurations of the executable SMs, which are generated using the previously described operations of the eSES/MB framework. The *semantic conditions* of SESvars are checked before pruning and exclude certain system configurations. The resulting system design after applying the pruning method can lead to several SMs. Each SM in a set of SMs with the same structure can have a different parametrization. The EC then transmits the SM or the set of SMs to the EU. The EU links the generated SMs to the simulator, executes simulation runs in a sequential or parallel fashion and, finally, sends the results back to the EC. The results, in turn, can influence the decision of the EC on how to set the SESvars next.

2 The Python-Based eSES/MB Infrastructure

The eSES/MB infrastructure for automatic generation and execution of models has been coded in four Python applications: SESToPy, SESMoPy, SESEuPy, and SESEcPy like shown in Figure 2.1.

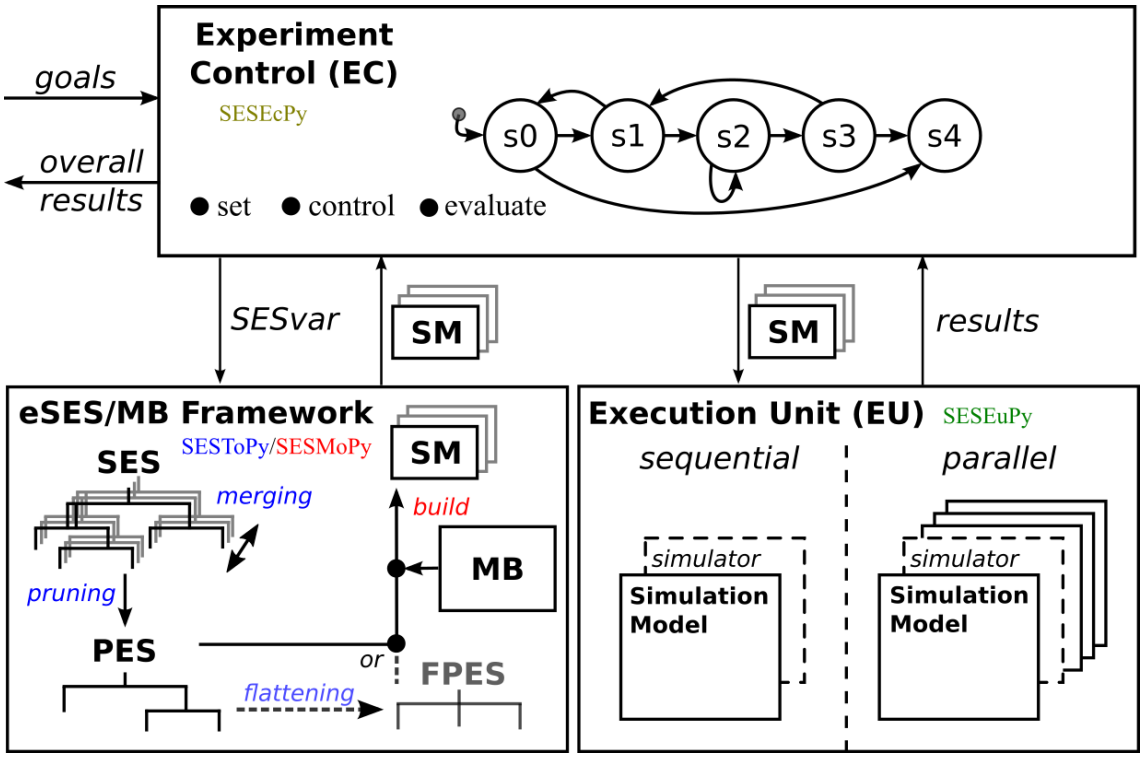


Figure 2.1: Extended SES/MB infrastructure with tools.

In the modeling phase a number of system configurations are coded in an SES with the help of SESToPy. A system configuration is a structure variant with a parameterization. The formal linkage between the SES and basic models in an MB is established by the mb-attribute. Couplings describe the composition of the basic models. Set selection rules (aspectrules, specrules) and attributes (numRep, priority) with expressions dependent of SESvars. Different values for the SESvars then lead to different decisions/values and thus to a different system configuration – the SESvars are an interface of the SES. Finally create a concrete MB as discussed

in the paragraph SESMoPy and demonstrated in the documentation of SESMoPy. Thus in the modeling phase there are the system variants/configurations coded in an SES and an MB is created.
Next, an EC needs to be written. This is an experiment specific script to control the experimentation process called SESEcPy. It contains the overall goals of an experiment and how to set SESvars depending on simulation results.

In the execution phase SESEcPy sets SESvars and controls:

- Pruning: A system configuration coded in a PES is derived from the SES using SESToPy
- Flattening: An FPES is calculated by removing the hierarchy from the PES using SESToPy
- Build: An executable model for simulation is created with the information in the FPES and the basic models in the MB using SESMoPy

The resulting model is optimal taylored without any components not needed for the simulation. Finally SESEcPy adds some simulation parameters, executes the simulation of the model with the help of SESEuPy and collects the simulation results. The simulation results are evaluated and depending on the results of a simulation run different action can be taken: If the results fit the goals, the overall results are calculated and returned, else new values are set for the SESvars and the execution process is started over again. Thus the experiment run is controlled reactively.

The software components are explained in the following paragraphs. In Figure 2.2 the detailed implementation is presented.

SESEcPy The tool SESEcPy (**S**ystem **E**ntity **S**tructure **E**xperiment **c**ontrol **Py**thon) serves as control unit to control the experimentation process and define the experiment to execute and the experiment goals. It consists of scripts and calls the APIs of SESToPy, SESMoPy, and SESEuPy as shown in Figure 2.2.
For details please refer to the documentation of SESEcPy.

SESToPy SESToPy **S**ystem **E**ntity **S**tructure **To**ols **Py**thon) is the SES modeling tool implementing a graphical SES editor and all SES related methods. In the editor an SES tree can be specified interactively in a file browser view and for every node specific attributes or rules can be defined. In addition to the *pruning* method for deriving a system structure, SESToPy supports some more methods such as *merging* different SES and *flattening* for removing the hierarchy information. Applying the

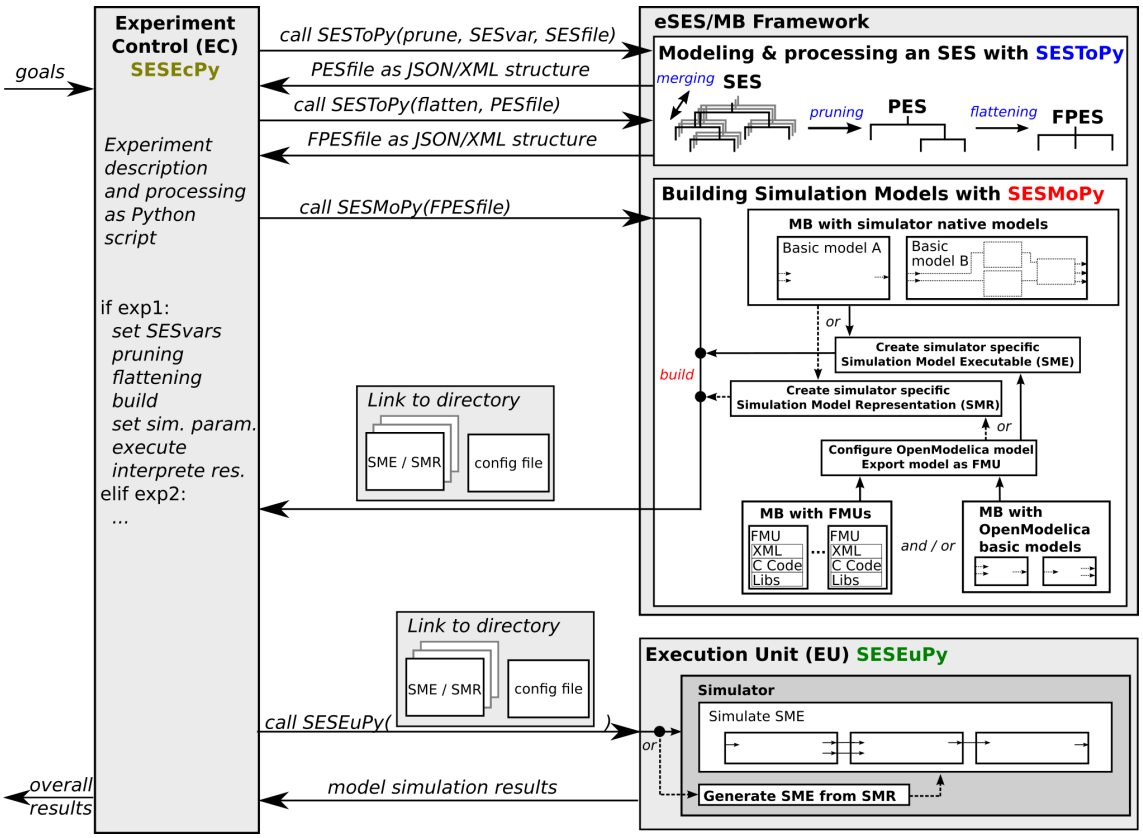


Figure 2.2: Python-based eSES/MB infrastructure.

flattening method, an FPES is derived. The pruning and flattening methods can be used interactively or as API methods. SESToPy supports to save SES, PES, and FPES as JSON or XML files.

SESMoPy For generating executable models, SESMoPy (**S**ystem **E**ntity **S**tructure **Mo**del builder **Py**thon) was developed. SESMoPy is a model builder, which supports *build* methods for several simulation environments in two ways.

(i): Currently *native models* for *MATLAB/Simulink, Dymola, and OpenModelica* can be generated using simulator native MBs organizing basic models. To be applicable to the same SES, in all simulator specific MBs there have to be the same basic models. Basic models can be atomic models or coupled models. Each basic model with the same name needs to have the same interface. According to the information in the FPES a set of simulator specific SMs is built using the basic models in the MB. Different configurations according to the FPES lead to a number of SMs. Depending on the target simulator a simulator specific file with instructions on how to build an executable SM is generated or an executable SM is created directly. In

Figure 2.2 the executable SM is called Simulation Model Executable (SME) and the file with instructions on how to build the SME is called Simulation Model Representation (SMR). For details on this please look at the documentation of SESMoPy. Via the JSON or XML interface of SESToPy other simulation environments can be integrated.

(ii): In addition, SESMoPy supports *model generation based on the Functional Mock-up Interface (FMI) definition.* The FMI standard is defined for several uses, one of which is FMI for Model Exchange. The generalized interface FMI is supported by a number of established simulators.
The MB organizes an amount of Functional Mock-up Unit (FMU) and / or OpenModelica basic models. FMUs are basic models implementing the FMI as interface. Importing the basic models from the FMU MB and / or using the basic models from the OpenModelica MB an OpenModelica model is created and configured according to the information in the FPES. This OpenModelica model is exported as FMU. Different configurations according to the FPES lead to a number of models exported as FMU. Using FMI in the modelbuilder, only one MB for all simulators is needed, since the created models as FMUs define their interface based on the FMI standard supported by simulators. Thus independence of the SES and the simulator can be achieved on simulator level. For details please look at the documentation of SESMoPy.
Depending on the target simulator a simulator specific file with instructions on how to build an executable SM is generated or an executable SM is created directly. In Figure 2.2 the executable SM is called SME and the file with instructions on how to build the SME is called SMR. For creating an SME the respective model FMU is imported in the target simulator in the modelbuilder. For details please look at the documentation of SESMoPy.

For both ways the resulting SMs (SMEs or SMRs) are created in one directory together with a configuration file with information about the created models and the target simulator. SESMoPy returns a link to the directory containing the created SMs and the configuration file.

Thus SESMoPy's build method generates SMs using the information in the FPES and basic models from a target specific (the native approach) or universal (the FMI approach) MB. Information about the way the model is created can be provided in the EC calling SESMoPy or at the SES level according to the SES enhancements. More information on the modelbuilder, especially the support of different simulators, is in the documentation of SESMoPy.

SESEuPy The Python software tool SESEuPy (**S**ystem **E**ntity **S**tructure **E**xecution **u**nit **Py**thon) acts as a general EU. It implements a kind of wrapper for the integration of different simulation environments into the framework. It receives a link to the directory containing the SMs and the configuration file. The configuration file is analyzed. If an SM comes as SME it can be executed directly in the target simulator. If an SM comes as SMR, an SME needs to be generated first by the target simulator before the SME can be executed. When using FMI and the SM comes as SMR, the respective model FMU is imported in the target simulator for generating the SME. For details please refer to the documentation of SESEuPy.

Due to the modular structure and well-defined interfaces, components for generating component-based software for non-simulation-specific environments can also be integrated into the framework.
This is shown for software in the example in Section 4.15. It cannot be used with SESEcPy and SESEuPy, since these programs are created for the use with simulators.

3 Description of SESToPy

The software SESToPy supports the eSES/MB framework and interfaces for pruning, merging, and flattening as needed for the eSES/MB infrastructure and shown in Figure 3.1.

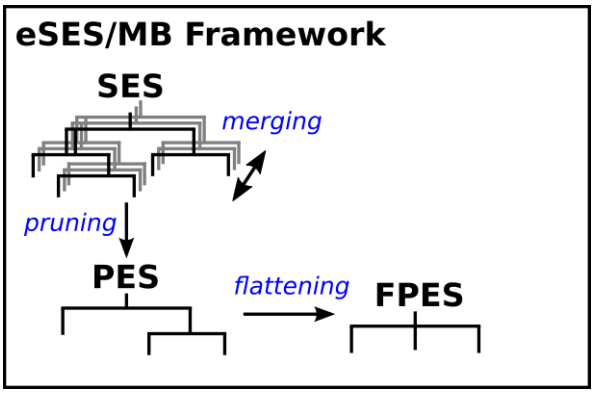


Figure 3.1: Part of the eSES/MB infrastructure supported by SESToPy.

The software is written in Python 3 using the bindings PyQt5 for the user interface. It is programmed in Python 3.4.1 with PyQt 5.5, but as of August 2018 it runs in current Python/PyQt versions as well. There are scripts for building an executable in Windows using cx_Freeze and a runnable in Linux using PyInstaller. Currently there are no more modules needed than given in the Python Standard Library.

The program window of SESToPy has a menu at the top, quick links below and is mainly a tabbed panel with 10 tabs, each representing a space, where an SES-model can be defined. Each tab is parted in three frames. In the left frame, SES-model wide settings (*Global Settings*) can be set, in the middle frame the SES tree can be defined (*Hierarchy Model*), and in the right frame node specific properties (*Node Specific Properties*) depending on the type of the node can be set. In the following sections the three frames are explained in detail. Figure 3.2 gives an overview.

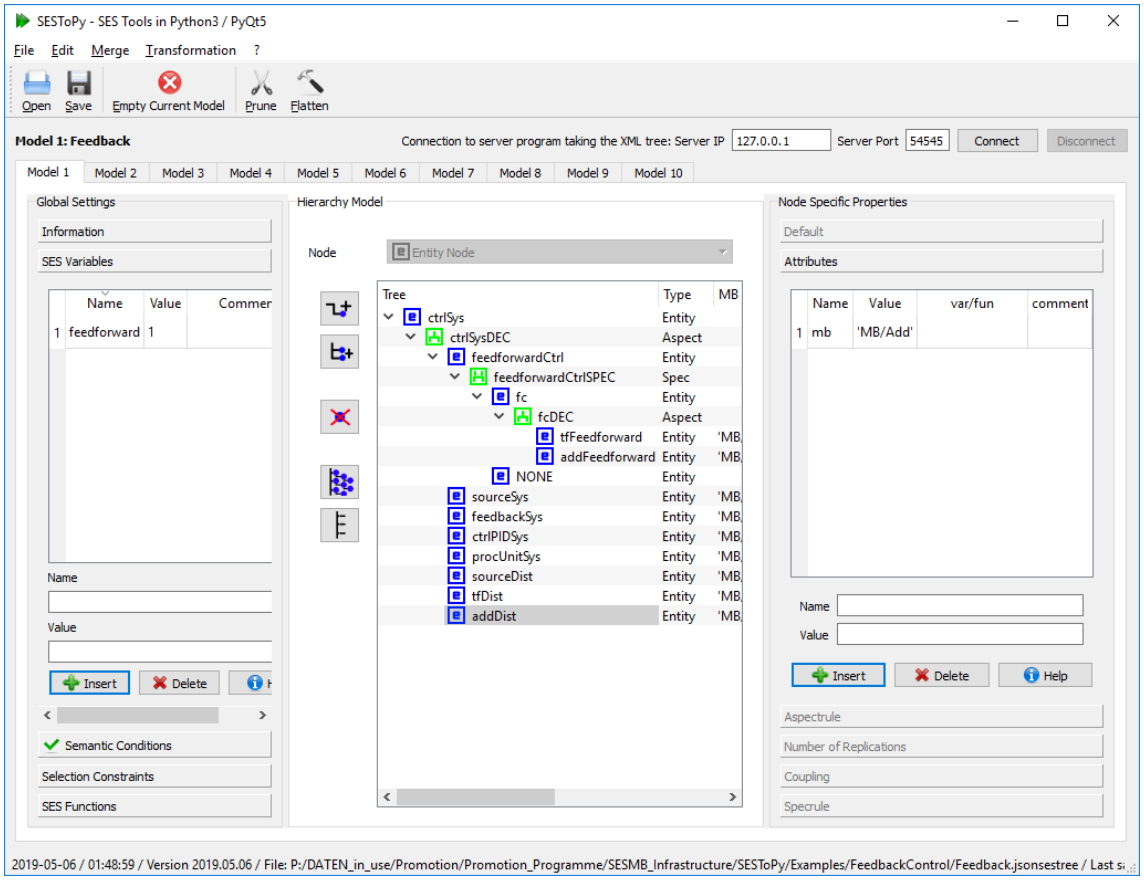


Figure 3.2: The program window of SESToPy.

The program acts as client in a server-client infrastructure exchanging the data specified in this program. The global settings, the SES tree and all node specific attributes are exported as XML string whenever a change in the data occurs. The XML string is the same as presented later in Sections 3.5.5 and 3.5.6. The server can be for example a program showing a tree view. The server needs to be started and its IP address and port is set in SESToPy. Click on the connect button connects SESToPy with the server, click on the disconnect button disconnects again.

3.1 Working with SESToPy

In this section some information on how to use SESToPy are given. For a wide variety of examples to use with SESToPy please look at Section 4.
In Figure 3.3 an overview of the graphical user interface of SESToPy is given. Numbers in red refer to the buttons described in this text.

To insert the first node press button 1. A node is inserted in the hierarchy model.

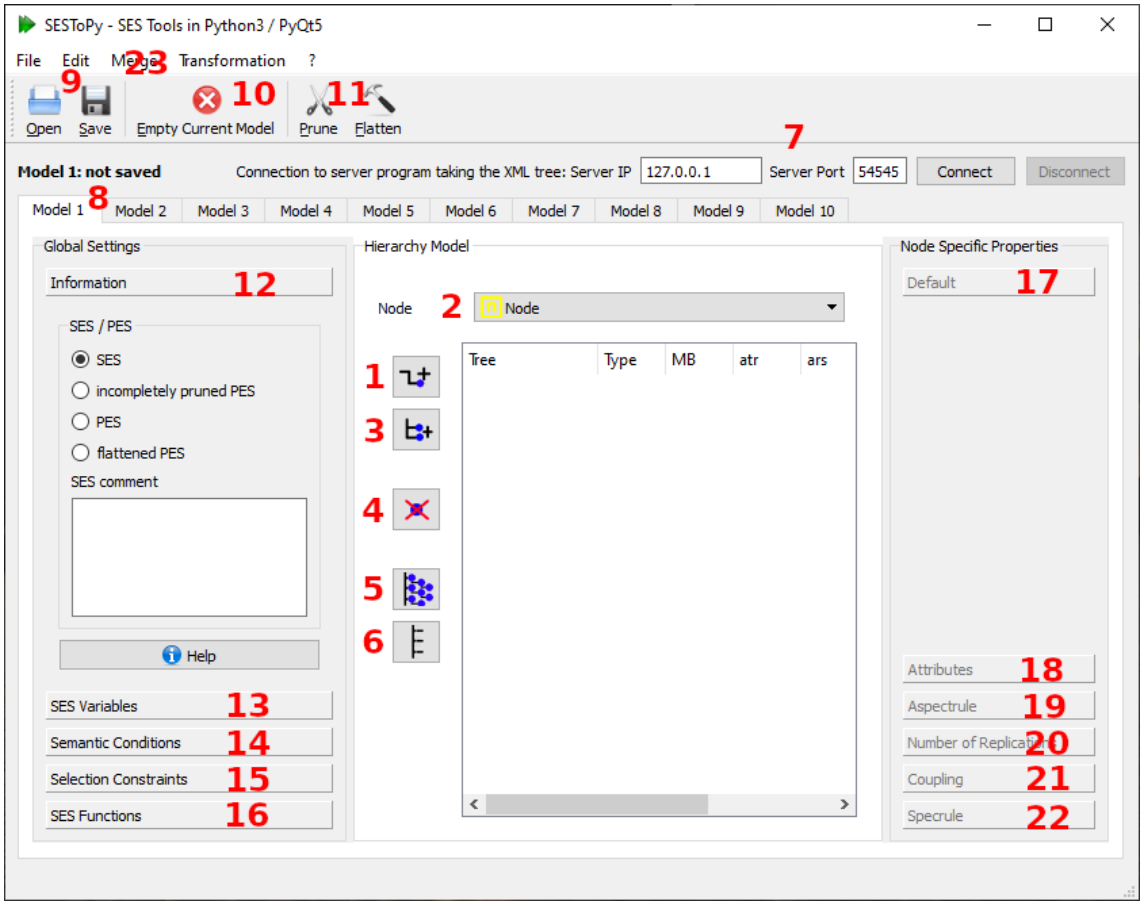


Figure 3.3: SESToPy - The graphical user interface.

The name of a node can be changed by double clicking the name of a node or by pressing the key F2 on a selected node in the hierarchy model. Underscores in names of nodes are reserved for pruning and are changed to minus signs. Press button 1 again to insert a subnode. The type of a node can be changed by setting a selection in the dropdown menu 2. Depending on the type of a node toolboxes on the right side are opened. A sibling can be inserted by clicking the button 3. A node can be deleted by clicking the button 4. The tree can be collapsed by clicking the little arrow in the hierarchy model next to the node. Clicking on button 5 expands the whole tree and clicking on button 6 collapses the whole tree. In the context menu of SESToPy (rightclick on the SESToPy's user interface) the functionality of buttons 1, 2, and 4 as well as the functionality of the drop down menu 2 can be accessed. Further information on the hierarchy view are given in Section 3.3.

SESToPy supports to define ten different SES. Between them can be switched clicking on the respective tab (number 8). With the buttons 9 an SES can be saved in SESToPy's JSON-based format and opened again. The current tab can be emptied

completely by clicking on button 10.

In the file menu left of 9 the functionality of the buttons 9 and 10 can be accessed as well. Furthermore in this menu an SES can be saved with another name or be exported or imported in an XML-based format. The last opened files are listed here as well and they can be opened by clicking the respective entry.

Please refer to the respective chapters for detailed information:

- 7: Paragraph below Figure 3.2
- 11: Sections 3.6 and 3.7
- 12: Section 3.2.1
- 13: Section 3.2.2
- 14: Section 3.2.3
- 15: Section 3.2.4
- 16: Section 3.2.5
- 17: Section 3.4.1
- 18: Section 3.4.2
- 19: Section 3.4.3
- 20: Section 3.4.4
- 21: Section 3.4.5
- 22: Section 3.4.6
- 23: Sections 3.5.7 and 3.5.8

The interface is explained in detail in the next sections.

3.2 Frame: Global Settings

In the Global Settings there are five toolboxes, in each of which SES-model wide settings can be taken.

3.2.1 ToolBox: Information

In this toolbox can be selected, whether a model is an SES, an incompletely pruned PES, a PES or a flattened PES. Usually, an SES is entered and after the pruning process could be finished completely (all decisions could be resolved) the radio button is set indicating a PES. If a decision cannot be found, the radio button is

set indicating an incompletely pruned PES. If the radio button indicates the current model to be a PES, via flattening a FPES can be generated. Furthermore a comment describing the SES can be entered.

3.2.2 ToolBox: SES Variables

An SESvar consists of a name and a value. Insert a unique name for the variable name. The variable name must begin with a letter and can be followed with any alphanumeric. The variable name must not be the name of an SESfcn, 'PARENT', 'CHILDREN', 'NUMREP' or 'PATH'. Insert a value using Python syntax. You can insert any Python construct. If you want to define a string, please use " or ' but do not mix them. The name and the value can be edited by double clicking on them. When finished editing and a syntax error is found the variable is deleted.
In the right field for every SESvar a comment can be inserted.

Examples for the value:

```
1   1.5   'abc'   [1, 2, 2.5]   ['a', 1, 'b', 4.5]   {1: 'a', 2: 'm'}
```

3.2.3 ToolBox: Semantic Conditions

For every SESvar, a semantic condition should be defined limiting the value range of the variable. Insert a boolean expression for the SESvars using Python syntax. After input the program tries to interprete the expression using the SESvars. If the SESvar(s) used in the semantic condition can not be found the background of the corresponding line is colored red and no result is printed. If the SESvar(s) used in the semantic condition can be found the background of the corresponding cell is colored green. The expression is evaluated and depending on the result (True / False) the result cell in the corresponding line is colored green or red. Furthermore, if all semantic conditions are okay, a green hook is set on the ToolBox bar.

Examples (a and b are SESvars):

```
a==b   a==1   a>b   a=='n'   a==['n', 4, 5]   a=={1: 'n', 2: 'm'}
a.get(2)=='m'   0<a and a<2   0<a<2
```

3.2.4 ToolBox: Selection Constraints

For cross-tree selections, selection constraints can be used. Choose a startnode from the tree. Choose one or more targetnode(s) from the tree. It is watched that only allowed nodes can be selected. If you want to change your selection use the clear selection button. When inserting the chosen constraints a color for the constraints must be selected. The chosen nodes are colored in the hierarchy model respectively. If a name is changed in the hierarchy model the selection constraints are updated. If a node defined in the selection constraints is deleted in the hierarchy model the regarding selection constraint is deleted.
The selection constraints can be replaced by usage of SESvars in combination with aspectrules / specrules.

3.2.5 ToolBox: SES Functions

With SESfcns, there is the posiibility to define procedural knowledge in the SES. Choose a file containing a correctly defined Python function. The name of the function is extracted from def name(...) and the syntax of the function is verified. The function name must not be the name of an SESvar, 'PARENT', 'CHILDREN', 'NUMREP' or 'PATH'.
An SESfcn can be copied with the key combination Ctrl+C.

3.3 Frame: Hierarchy Model

In this frame the SES tree can be defined like in a file structure browser. At the top, the node type can be selected, if allowed. On the left side there are five buttons: 'Add SubNode', 'Add SiblingNode', 'Delete Node', 'Expand All', and 'Collapse All'. The first node of a tree needs to be created clicking 'Add SubNode'. It cannot have a sibling. By double clicking the name, the name can be changed. Inserting a second node, a type (Aspect / Maspect / Spec) needs to be set in the selection field above (Maspect = multi-aspect, Spec = specialization). With clicking 'Add SubNode' or 'Add SiblingNode', the tree is built. 'Sub' and 'Sibling' is always based on the currently highlighted node.
The arrow in front of a node, lets the subtree collapse and expand. With the buttons 'Expand All' and 'Collapse All', the whole tree can be expanded and collapsed.
With the button 'Delete Node' the currently highlighted node and its subtree are deleted.

Next to the name, there are fields containing hints to the properties set for a node. When selecting a node, in the frame on the right side possible ToolBoxes, in which are properties that can be set, are made accessible.
The alternating mode is watched when entering a node. Entity nodes are followed by descriptive nodes, and vice versa. Furthermore the axioms of strict hierarchy and uniformity are watched. When entering a node with the same name like another node in the current path, a hint is given. If it is a different path, it can be chosen that a subtree like the other node with the same name has, is created. Nodes with the same name and the same parent must not have the same name, as the axiom of valid brothers urges. No node may have the same variables like demanded by the axiom of attached variables.
Nodes may have the name NONE. This is an extension of the SES. Please look in chapter 4 for further information.
The underscore _ is reserved for pruning and may not be used. Underscores are converted to minus signs.

3.4 Frame: Node Specific Properties

In the Node Specific Properties there are six toolboxes, which are greyed out at first. Depending on the node type selected, some ToolBoxes are made accessible.

3.4.1 ToolBox: Default

If no node is selected, this ToolBox is shown. Nothing can be created here.

3.4.2 ToolBox: Attributes

This ToolBox is accessible for entity nodes.

Entity nodes following multi-aspect nodes get an underscore attribute _Name-OfTheNode. This attribute is an internal variable of the program and can neither selected nor edited. The underscore attribute is a variable initialized with “0“ in the SES and during pruning this attribute gets the number of the generated child of a multi-aspect node as string value. These underscore attributes can only be accessed by the keyword “PATH“ from attributes, aspectrules, priority, number of replications, couplings, and specrules – e.g. as parameters of an SESfcn. In Chapter 4 examples are shown.

Insert a unique name for the variable name. The variable name must begin with a letter and can be followed with any alphanumeric. The variable name must not be 'PARENT', 'CHILDREN', 'NUMREP' or 'PATH'. Insert a value using Python syntax. You can insert any Python construct. If you want to define a string, please use " or ' but do not mix them. The name and the value can be edited by double clicking on them.
When finished editing and a syntax error is found the variable is deleted. If the attribute shall refer to an SESvar enter the sesvarname without " or ' in the value field. If the attribute shall refer to an SESfcn enter sesfunname() without " or ' in the value field. If you want to pass parameters to the function, you can do so passing them in the round bracket. sesvarname and sesfunname must be alphanumeric not beginning with a number. A reserved parameter is "PATH". PATH is a dictionary containing the names of the underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. Please look further in this subsection and in the next sections of this documentation for a comprehensive explanation of underscore attributes.
It is checked that the name of the SESvar or SESfcn exists and the SESfcn is executable. The value field is colored red if the SESvar or SESfcn does not exist or the SESfcn is not executable e.g. parameters for the SESfcn are missing, otherwise green. Therefore if several SESvars are referenced in an SESfcn all variables must exist.
In the comment field, any comment can be placed describing the attribute in detail. A special variable is a variable with the name "MB" (not case-sensitive). This variable creates the link to a basic model in a model base. It usually has the name "mb" and the value "Name_of_MB/basic_model". In Chapter 4 examples are shown.

Please refer to the documentation of the modelbuilder software SESMoPy for further information on the mb-attribute. An attribute configures a parameter in the basic model, to which the entity node refers with the mb-attribute, when building a model. The SES shall be independent of the simulator the model is later built for. For more information on attributes as parameters of basic models when using SESMoPy, please refer to documentation of the modelbuilder software SESMoPy.

Examples for the value:

```
1   1.5   'abc'   [1, 2, 2.5]   ['a', 1, 'b', 4.5]   {1: 'a', 2: 'n'}
```

`sesvarname PATH sesfunname() sesfunname(4.5) sesfunname(sesvarname) sesfunname(sesvarname1, sesvarname2)`

3.4.3 ToolBox: Aspectrule

This ToolBox is accessible for aspect and multi-aspect nodes. For every aspect node or multi-aspect node one single aspectrule can be defined. This needs to be an expression using Python syntax which can be evaluated to a boolean expression. If a variable shall refer to an SESvar enter the sesvarname without " or ' in the condition field. If a variable shall refer to an SESfcn enter sesfunname() without " or ' in the condition field. If you want to pass parameters to the function, you can do so passing them in the round bracket. sesvarname and sesfunname must be alphanumeric not beginning with a number. A reserved parameter is "PATH". PATH is a dictionary containing the names of the underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. Please look in the Section 3.4.2 and in the next sections of this documentation for a comprehensive explanation of underscore attributes.
It is checked whether the name of the SESvar or SESfcn exists. If several SESvars are referenced in an SESfcn all variables must exist. The whole line is colored red if the condition can not be interpreted e.g. due to an missing SESvar or function. If the condition can be interpreted but the result is false only the condition field is colored green, the other fields are colored red. If the condition can be interpreted and the result is true the whole line is colored green. Using PATH in the condition may lead to a wrong result in the SES, since underscore attributes are initialized with "0" and are recalculated during pruning.
When selecting an aspect or multi-aspect node it is searched for brothers of the same type. For every brother node of the same type as the selected node a line is created in which you can define an aspectrule. Only the line belonging to the selected node can be changed, the other lines are disabled and greyed out. Changing the node selection in the hierarchy model enables other lines.

Pruning an SES can lead to two or more aspect or multi-aspect nodes in the same level. In order to prune it is necessary to sequence the nodes by giving a priority. A higher priority means the node is selected preferred. An expression with SESvars and/or SESfcns must evaluate to a number. Coloring the line and possible values

is like described in Section 3.4.4. Using PATH may lead to a wrong result in the SES, since underscore attributes are initialized with “0“ and are recalculated during pruning.

Examples for the condition in an aspectrule:

```
a==1    b==[1, 2]    sesvarname==1   sesfunname()==’a’
sesfunname(4.5)==’m’   sesfunname(sesvarname)==sesvarname2
sesfunname(sesvarname1, sesvarname2)==’i’ and 1==1   "abc"==A and True
True   False   sesfunname(PATH)==1
```

3.4.4 ToolBox: Number of Replications

This ToolBox is accessible for multi-aspect nodes. Enter an integer >=1 . If a variable shall refer to an SESfcn enter sesfunname() without “ or ’ in the condition field. If you want to pass parameters to the function, you can do so passing them in the round bracket. sesvarname and sesfunname must be alphanumeric not beginning with a number. A reserved parameter is “PATH“. PATH is a dictionary containing the names of the underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. Please look in Section 3.4.2 and in the next sections of this documentation for a comprehensive explanation of underscore attributes.
It is checked whether the name of the SESvar or SESfcn exists. If several SESvars are referenced in an SESfcn all variables must exist. The line is colored red, if the SESvar or SESfcn does not exist or the result is no integer >=1, otherwise green. Using PATH may lead to a wrong result in the SES, since underscore attributes are initialized with “0“ and are recalculated during pruning.

Examples for the value:

```
1   12   sesvarname   sesfunname()   sesfunname(4.5)   sesfunname(sesvarname)
sesfunname(sesvarname1, sesvarname2)   sesfunname(PATH)
```

3.4.5 ToolBox: Coupling

This ToolBox is accessible for aspect and multi-aspect nodes. Couplings describe, how the parent of the current node is coupled to a child of the current node or the

children of the current node are coupled to each other. Couplings can be defined using a coupling iterator list or using SESfcns. For multi-aspect nodes a coupling function has to be used since for every child a coupling iterator list needs to be defined. For the definition of a coupling the source node and the sink node have to be selected from the dropdown menus. The source port and the sink port have to be defined in a list in the line below the dropdown menus. Several ports for a source node and a sink node can be defined at once. For details see examples. Please always decide for a direction of flow, do not mix source ports and sink ports. When couplings are inserted, the name and the type of the ports can be adapted by double clicking the “port name / type“ field of the inserted node in the couplinglist.

If the coupling is defined by an SESfcn a list of the SESfcns can be selected from a dropdown menu. The selected function is inserted in the line below. If the function can not be interpreted e.g. since parameters are missing the line is colored red, otherwise green.
As function parameters it is possible to pass the node specific values “PARENT“, “CHILDREN“, “NUMREP“, and “PATH“. PARENT refers to the parent's name of the current node, CHILDREN passes a list of the children's names of the current node and NUMREP refers to the number of replication of the current node, and PATH lists the underscore attributes in the path from the current node to the root. The PARENT is a string with the parent's name of the current node, the CHILDREN is a list consisting of the children's names of the current node, [child1name, child2name, ...], whereas the NUMREP is just an integer value containing the number of replications. PATH is a dictionary containing the names of the underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. Please look in Section 3.4.2 and in the next sections of this documentation for a comprehensive explanation of underscore attributes.

If the coupling is defined by an SESfcn the coupling iterator list is greyed out and vice versa. The result of the coupling function is printed below the entry of the coupling function. Make sure, that all used SES variables and SES functions exist and that the result is something like
[[“sourcenodename“,“sourcenodeport / porttype“,“sinknodename“,“sinknodeport / porttype“,“comment“]]
or
[[“sourcenodename“,“sourcenodeport / porttype“,“sinknodename“,“sinknodeport /

porttype“,“comment“], [“sourcenodename“,“sourcenodeport / porttype“, “sinknodename“,“sinknodeport / porttype“,“comment“],...]
and make sure that the sourcenodename and the sinknodenames equal one parent's or child's name. Usage of the PARENT and CHILDREN attributes are helpful for this (see examples). Furthermore, make sure, the sourceport and sinkport are strings and no arrayindex refers to a nonexistent element (e.g. in the childrenlist when passing CHILDREN). Note the whitespaces and the / separating the portnames from the porttypes (the types of the ports of basic models, which are connected by the couplings). Some possible porttypes and their meaning are listed in Table 3.1. Types of physical ports are not listed here, but they begin with “PP“. Please see the help in the coupling ToolBox in the SESToPy program to see possible types.

Table 3.1: Possible porttypes and their meaning.

Porttype	Meaning
SPR	SignalPortReal
SPI	SignalPortInteger
SPB	SignalPortBoolean
PPx	PhysicalPortXXX

Using PATH may lead to a wrong result in the SES, since underscore attributes are initialized with “0“ and are recalculated during pruning.

The SES shall be independent of the simulator the model is later built for. Information on how to name the coupling ports of the model components (the mb-attribute of entity nodes refers to) in the SES for model generation with SESMoPy can be found in the documentation of the modelbuilder software SESMoPy.

Examples for the coupling iterator list, the selection of a source node and a sink node is assumed:

```
[1, 2]      [Out1, In2][2, 1]
```

Examples for couplings using SESfcns:

```
cplfcn(PARENT,CHILDREN,NUMREP)   cplfcn(PATH, PARENT)
```

3.4.6 ToolBox: Specrule

This ToolBox is accessible for specialization nodes. For every child of a specnode a single specrule can be defined. This must be an expression using Python syntax

which can be evaluated to a boolean expression. If a variable shall refer to an SESvar enter the sesvarname without “ or ’ in the condition field. If a variable shall refer to an SESfcn enter sesfunname() without “ or ’ in the condition field. If you want to pass parameters to the function, you can do so passing them in the round bracket. sesvarname and sesfunname must be alphanumeric not beginning with a number. A reserved parameter is “PATH“. PATH is a dictionary containing the names of the underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. Please look at Section 3.4.2 and in the next sections of this documentation for a comprehensive explanation of underscore attributes.
It is checked whether the name of the SESvar or SESfcn exists. If several SESvars are referenced in an SESfcn all variables must exist. The whole line is colored red if the condition can not be interpreted e.g. due to an missing SESvar or function. If the condition can be interpreted but the result is false only the condition field is colored green, the other fields are colored red. If the condition can be interpreted and the result is true the whole line is colored green. Using PATH in the condition may lead to a wrong result in the SES, since underscore attributes are initialized with “0“ and are recalculated during pruning.
When selecting a spec node, for every child a line is created in which you can define the child specific specrule.

Examples for the condition:

```
a==1     b==[1, 2]     sesvarname==1   sesfunname()==’a’
sesfunname(4.5)==’m’   sesfunname(sesvarname)==sesvarname2
sesfunname(sesvarname1, sesvarname2)==’i’ and 1==1   "abc"==A and True
True   False   sesfunname(PATH)==1
```

3.5 Menus

Via the menus it is possible to save and open a file, empty a SES-model, export as XML, merge, prune, and flatten an SES. This is described in detail in the next chapters.

3.5.1 File: Open

SESToPy saves all internal data in a JSON format with the fileending ".jsonsestree". A file of this format created with SESToPy can be opened and is inserted in the current tab. If the current tab already has data, SESToPy asks to empty the data in the current tab before inserting data. An example for the JSON structure is given in the appendix.

3.5.2 File: Save

The data of the SES-model of the current tab are saved in a JSON structure. The file gets ".jsonsestree" as ending. If a model is saved the first time, the place and name to store are selectable. After saving the name of the model is shown above the tabs and the whole path is shown in the statusbar. An example for the JSON structure is given in the appendix.

3.5.3 File: Save As

Just like Section 3.5.2, but the place and name to store are chooseable every time it is selected.

3.5.4 File: Empty Current Model

The data of the current tab is emptied completely.

3.5.5 File: Import from XML

An XML file containing an SES readable for SESToPy can be inserted in the current tab. The XML file contains the global variables and the tree with attributes. If the current tab already has data, SESToPy asks to empty the data in the current tab before inserting data. An example for an importable XML structure is given in the appendix.

3.5.6 File: Export as XML

The SES-model of the current tab can be saved as XML. An example for an exported XML structure is given in the appendix.

3.5.7 Merge: Add SubSES

A saved sub SES can be appended below the currently highlighted node. The currently highlighted node needs to have the same name as the node the sub SES begins with. Usually the name of the first node of the sub SES to insert is coded in the filename as described in Section 3.5.8. It is then replaced by the sub SES. Warnings are given, if a variable is found in the sub SES which already exists in the SES.

3.5.8 Merge: Save SubSES

The data beginning from the currently highlighted node are saved as a JSON structure in a file with the ending ".jsonsestree". The filename is suggested as <name_of_the_first_node>_subSES.jsonsestree. All global settings (SESvars, Semantic Conditions, ...) are saved as well. If there are selection constraints and only some nodes of the selection constraint would be saved, there is a warning and the file is not saved.

3.5.9 Merge: Import SubSES from XML

Just like Section 3.5.7, but the data is exported as XML.

3.5.10 Merge: Export SubSES to XML

Just like Section 3.5.8, but the data is exported as XML.

3.5.11 Transformation: Prune

Starts the pruning method of the current SES-model. See Section 3.6 for details.

3.5.12 Transformation: Flatten

Starts the flattening method of the current SES-model. See Section 3.7 for details.

3.6 Pruning

An SES describes a number of system configurations. Getting a specific system configuration, a PES needs to be created. Since information are cut away during pruning, please make sure, to always start pruning with an SES, never with an incompletely pruned PES.

In the current version of SESToPy selection constraints are neglected in the pruning process.

During pruning, decisions are taken at aspectrules and specrules. For aspect / multi-aspect brothers, one brother needs to be selected. In specnodes, one child needs to be selected, the name of the selected child is united with the parent of the current node and attributes are inherited as urged in the axiom of inheritance. Same attributes in both nodes are overwritten with the child ones. Furthermore, for multi-aspect nodes children according to the number of replications are created, the node is changed to an aspect node and the childrens' names are extended with a number to fulfill the axiom of valid brothers. The nodes have internal variables, such as an unique identifier (uid), some of which need to be adjusted during pruning. Furthermore during pruning the underscore attributes in children of multi-aspect nodes are assigned. A number according to the number of the created child is given.

The pruning algorithm is presented as an Activity Diagram in the appendix and comprehensible examples are given in chapter 4. The basic ideas shall be explained here. There is nearly no recursion in the algorithm.

Pruning Algorithm The pruning algorithm is based on two lists of the nodes in a different format. One list describes the properties of each node, but has no direct information on the hierarchy. Anyway, the depth of the layer of the node is stored. It is called *nodelist*. In the second list the nodes up to the root node for every leaf node are stored. This is a list containing all *paths* in the SES. Thus, this list holds information on the parent and children of every node.

Pruning can be started from the program's graphical user interface or using the program's shell interface (see later in this section).
When using the graphical interface the nodelist, SESvars, semantic conditions, SES-fcns, and paths are taken from the current active model in the user interface. When

entering a rule in the program's graphical user interface the expression in the condition is evaluated directly. The result of an evaluated condition is written into the node. This result is taken for pruning. If the expression in the condition has a PATH variable, the condition cannot be evaluated correctly in the SES. A PATH variable gives a dictionary with all underscore attributes from the current node to the root. The PATH variable is the only way to access underscore attributes. Underscore attributes can only be assigned during pruning multi-aspect nodes. Thus any expressions with the PATH variable need to be recalculated during pruning.
When using the shell interface the SES is given as JSON file and some variable/-value pairs for SESvars are passed. The nodelist, SESvars, semantic conditions, and SESfcns are read from the file. The paths are calculated and the semantic conditions, aspectrules and specrules are calculated with the new values for the SESvars, if the expression in the condition for an aspectrule or specrule does not have a PATH variable.
The next steps are the same for usage of the graphical user interface and the shell interface. In attributes, number of replication, and priority any SESvars and SESfcns in the expressions are evaluated and replaced by the calculated value, if the expression does not refer to PATH. It is checked, that the SES starts and ends with an entity node in all paths. Furthermore the semantic conditions (defined for giving a value range for the SESvars and therefore limiting possible variants of the SES) are checked. Changes during pruning are inserted in the nodelist. Therefore every entry in the list describing a node gets a field defining, whether the node shall be in the PES. For pruning, the paths list is travelled and for every node, which is not excluded from the PES yet, the pruning is done as described in the next paragraphs. Changes are written to the nodelist.

Start the main pruning sequence by checking, if the node is deactivated. If not, get the type of the node and do the node specific steps as described in the next paragraphs.

Pruning an Entity In case the current node is an entity node, no decision needs to be taken. Expressions in attributes referring to PATH are calculated, other attributes are calcuated before.

Pruning an Aspect In case the current node is an aspect node, expressions in aspectrules, the priority, and couplings referring to PATH are calculated and evaluated. The number of brothers is counted. In case there

are no brothers, the node will stay in the PES, just pass it. If there are brothers, it is looked for brothers of the type specialization, since they need to be pruned first. Prune any brother of type specialization as described in the next paragraph. If there are still aspect or multi-aspect brothers, there needs to be a decision by aspectrules, which of them is selected.
There are mainly two cases: On the one hand, the aspect / multi-aspect brothers are brothers in the SES, on the other hand these nodes can have become brothers by resolving a specialization node as described in the next paragraph.
For the first case exactly one node needs to be selected by aspectrules. The other nodes and their children are deactivated in the nodelist. Otherwise a warning is given and the type of the resulting PES is set to an incompletely pruned PES. No nodes are deactivated in the nodelist.
For the second case, the decision for one node is made by the priority attribute. It is important, that no two nodes share the highest priority. If so, the node with the highest priority is kept, the others are deactivated. If two nodes share the highest priority, a warning is given and the type of the resulting PES is set to an incompletely pruned PES. No nodes are deactivated in the nodelist. A higher value for the priority attribute means a higher priority. An example for this case is given in Section 4.12.

Pruning a Specialization In case the current node is an specialization node, expressions in specrules referring to PATH are calculated and evaluated. The selected child needs to be found in a specialization node.
If no child is selected, the node and its children are not integrated in the PES and the PES is declared as an incompletely pruned PES. If a child is selected, deactivate the specialization node and the selected child, before the not selected other children are deactivated including their subtrees. For the selected child get the name and the attributes and place the attributes in the father node of the specialization node overwriting existent attributes with the same name. Rename the parent of the specialization node connecting the selected child's name and the parent's name with an underscore. In case, NONE is selected as child, the father gets the name NONE. In the nodelist the parent's name and depth information need to be adapted for the subtree of the selected node.

Pruning a Multi-Aspect In case the current node is a multi-aspect node, expressions in aspectrules, the priority, couplings, and number of replications (numRep) referring to PATH are calculated and evaluated. The numRep variable and the children including their subtrees of the current multi-aspect node are fetched. They are put in a separate list, where they are multiplied according to numRep. Several internal variables, such as uids of the nodes, need to be corrected. Adjust the name by appending the children's number to the name, so the axiom of valid brothers is satisfied. Assign underscore attributes, which are attributes of entity node being children of multi-aspect nodes, with the number of the created child. Finally put the newly created nodes in the nodelist again and convert the multi-aspect node to an aspect node.
By creating the aspect node, aspect siblings could still be on the same layer. Therefore, start over with checking on deactivation of the current node again.

Finally, the excluded nodes are removed from the nodelist and the field added before to indicate whether the node shall stay in the PES is removed. If the pruning was started from the graphical user interface, the nodelist is inserted in a model in the graphical user interface. If the pruning was started from the shell interface, the nodelist is encoded as filestring and written to a file.

Pruning Interface The shell interface to the pruning method shall be described here.
The pruning method can be started from the command line calling SESToPy with the -p option as seen in Figure 3.4. The input SES .jsonsestree file is specified and the values of the SESvars for the pruning are set in the format
[SESvarname1=value1,SESvarname2=value2]. In case there is an SESvar containing a string, the whole list of SESvars needs to be placed in quotes. If the strings in the list are quoted with “, the list has to be quoted with ’ and vice versa. E.g.
’[SESvarname1=value1,SESvarname2=“value2“]’ or
“[SESvarname1=value1,SESvarname2=’value2’]“
An output filename is generated, and if all conditions are met for pruning, an output PES file is saved.
As shown in Figure 3.5, the output file can be specified with the -o option behind the values for the SESvars.

Figure 3.4: Pruning interface of SESToPy.

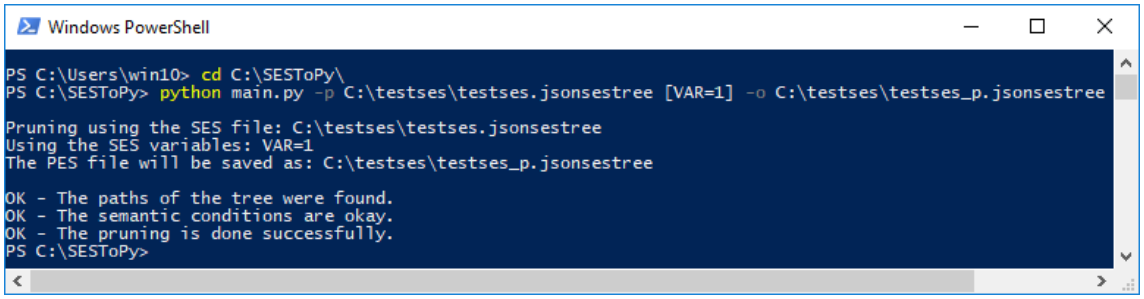


Figure 3.5: Pruning interface of SESToPy - specifying the output file.

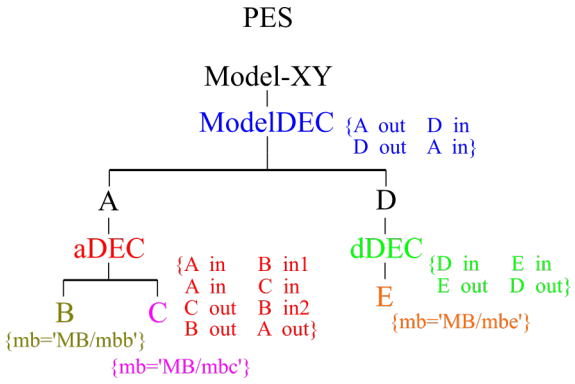
3.7 Flattening

Flattening is only allowed for a PES. During flattening all inner nodes are cut away leaving just the root, followed by an aspect and the leaves. When it comes to model generation, inner nodes have no influence on the model output (simulation result), since they just describe the structure of a model with coupling relations. They describe the hierarchy of entity nodes, which are linked to model components, in the SES. Leaf nodes can have the mb-attribute, which is the link to a basic model in the model base and can have further attributes for configuration of the basic model. When creating a model, it depends on the modelbuilder, whether an FPES needs to be created.

In the flattening process, all nodes of the PES except for the nodes of the first and second layer and leaf nodes are removed. All leaf nodes are placed on the third layer of the created FPES. Couplings referring to removed nodes are replaced by couplings referring to leaf nodes. A comprehensible example is given in Figures 3.6 and 3.7 and the algorithm is shown.

In Figure 3.6 a model is built from the PES. It is assumed, that the MB holds the basic models with the names mbb, mbc, and mbe. In the PES the couplings (without types of the ports of basic models which the couplings connect (porttypes)) and the mb-attribute as link to the MB for the respective nodes are given. Different colors express the affiliation. The SES shall be independent of the simulator the model is later built for and therefore generalized names (e.g. for ports of basic models) shall be used in the SES. For more information on how to define portnames of couplings, please refer to the documentation of the modelbuilder software SESMoPy. For more information on how to define parameter names of basic models (which are attributes of entity nodes), please refer to the documentation of the modelbuilder software SESMoPy. It can be seen, that the entities A and D are just "containers" for the entities B, C and E, which are linked to basic models. The system is called a coupled system. The entities A and D can be removed without losing information in the model (except for hierarchy information). This is called flattening.

When executing the flattening, leaf nodes that were not brothers before (they were



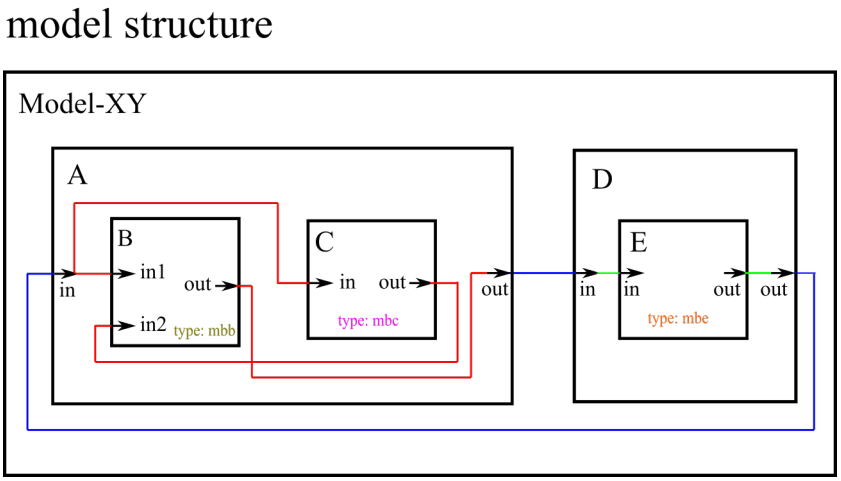


Figure 3.6: The PES and the model created from it.

on different paths) can become brothers since the hierarchy is resolved. This would be a breach of the axiom of valid brothers. Therefore, on execution of the flattening method for nodes of the same name an underscore followed by an unique number is appended to the name of the respective node in the FPES.
Figure 3.7 depicts the FPES and the model built of it. Again, it is assumed, that the MB holds the basic models with the names mbb, mbc, and mbe. As it can be seen, the inner nodes are cut away, all couplings are united in the aspect node on the second layer and there is no coupling referring to a removed node. The resulting model is the same as before, but lacks the "containers" A and D.

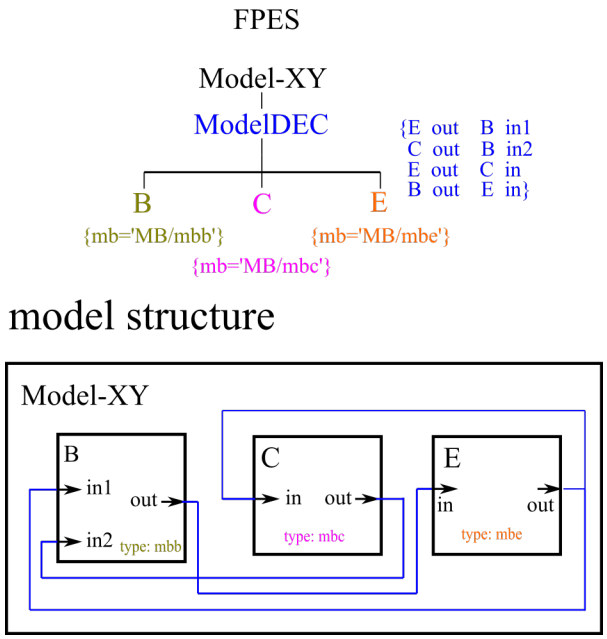


Figure 3.7: The FPES and the model created from it.

Adjusting the Couplings The algorithm for adjusting the couplings during flattening is demonstrated here using the example given in Figures 3.6 and 3.7. All couplings of the SES are collected. Every coupling has a source part and a sink part. Comparing Figures 3.6 and 3.7, it may be noticed that e.g. instead of coupling the " out " port of A (source) to the " in " port of D (sink), the " out " port of B (source) could be coupled to the " in " port of E (sink) without losing model information (except for the hierarchy), since A and D act as "containers" only. Source ports should stay sources and sink ports should stay sinks, as made clear in the algorithm.

So the basic idea is: 1) Get a list with all couplings of the PES and 2) go through the list and if one part of a coupling like “ A out “ refers to an inner node (which is removed during flattening), try to replace all occurences of “ A out “ with another coupling containing “ A out “.
It is started with the couplings of the example in Figure 3.6 shown in Table 3.2. The colors show the steps taken next. Note, that the porttypes are left away to keep matters simple.

Table 3.2: Couplings of the PES in Figure 3.6.

source		sink	
A	out	D	in
D	out	A	in
A	in	B	in1
A	in	C	in
C	out	B	in2
B	out	A	out
D	in	E	in
E	out	D	out

Travel the source couplings first. Take the first coupling, “ A out “ is identified as inner coupling (on the source side) -> try to replace all occurrences of “ A out “. Found in the 6th coupling, replace with “ B out “ -> Keep in mind, that a source should stay a source and a sink should stay a sink, so the replacement for “ A out “ should be found on the source side (and “ A out “ in the replacement coupling pair is then on the sink side like in the 6th coupling).
Remove the used 6th couplings from the list. Result in Table 3.3:

Table 3.3: Couplings of the PES in Figure 3.6 after the first step.

source		sink	
B	out	D	in
D	out	A	in
A	in	B	in1
A	in	C	in
C	out	B	in2
D	in	E	in
E	out	D	out

The node D will not be in the FPES, so “ D out “ is an inner coupling, which needs to be replaced. “ D out “ can be replaced with the 7th coupling. Remove the 7th coupling. Result in Table 3.4:

Table 3.4: Couplings of the PES in Figure 3.6 after the second step.

source		sink	
B	out	D	in
E	out	A	in
A	in	B	in1
A	in	C	in
C	out	B	in2
D	in	E	in

“ A in “ will not be in the FPES, but it is twice in the third and fourth coupling -> “ A in “ is replaced by “ E out “ and the 2nd coupling is deleted. Result in Table 3.5:

Table 3.5: Couplings of the PES in Figure 3.6 after the third step.

source		sink	
B	out	D	in
E	out	B	in1
E	out	C	in
C	out	B	in2
D	in	E	in

The node C will stay in the FPES, so the fourth coupling is okay. In the fifth coupling “ D in “ is found. Node D will not be part of the FPES, but it can be found in the first coupling. Replace and delete the first coupling. Result in Table 3.6:

Table 3.6: Couplings of the PES in Figure 3.6 after the fourth step - like the FPES in Figure 3.7.

source		sink	
E	out	B	in1
E	out	C	in
C	out	B	in2
B	out	E	in

The couplings of the FPES like in Figure 3.7 are evaluated.

If the PES is more complex, couplings might be at first replaced with couplings referring to nodes which are not in the FPES. This is resolved later, since the coupling referring to nodes not in the FPES will appear later again.

However, the ability to build models is not included in SESToPy. It is covered in the modelbuilder software SESMoPy.

Flattening Interface There is a shell interface to the flattening method.
The flattening method can be started from the command line calling SESToPy with the -f option as seen in Figure 3.8. The input PES .jsonsestree file is specified. An output filename is generated and the output FPES file is saved.

As shown in Figure 3.9, the output file can be specified with the -o option.

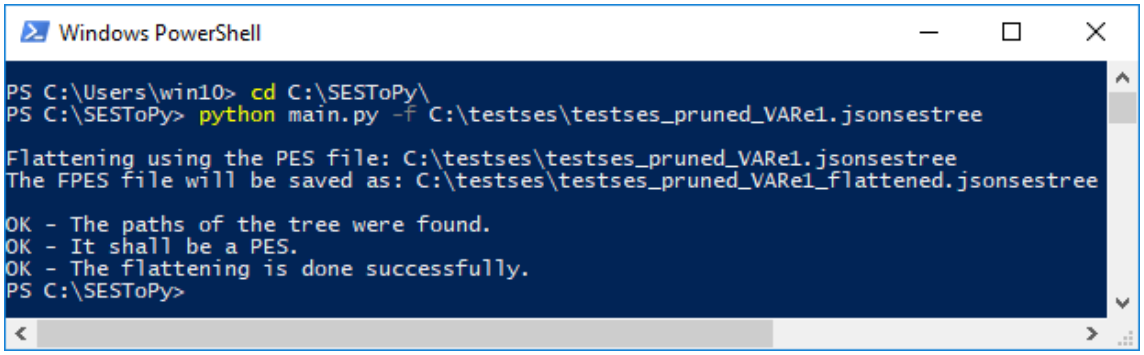


Figure 3.8: Fattening interface of SESToPy.

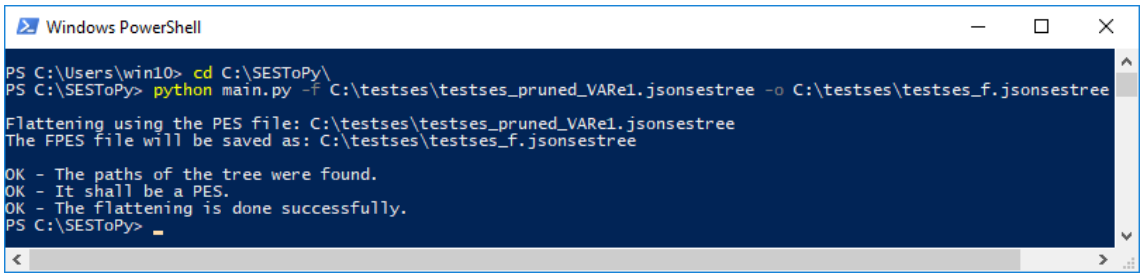


Figure 3.9: Flattening interface of SESToPy - specifying the output file.

3.8 Building an Executable/Runnable of SESToPy in Windows and Linux

For Windows and Linux there is the possibility to build an executable / runnable of the program. There is no installation of Python3 / PyQt5 needed on machines for executing these compiled files.

Executable in Windows In the main program directory of SESToPy is the file *CreateRunnableWindows.cmd*. This file is a Windows-Command script to execute in order to compile an .exe file of this program. Please note, that the executable is dependent of the processor architecture and the operating system. It is not platform independent anymore. A folder containing the .exe file and all for the execution necessary libraries is created. Compilation is done with cx_Freeze 4.3.4. The program cx_Freeze needs a configuration file. This configuration is stored in the file *setup-windows.py* the same directory as the file *CreateRunnableWindows.cmd* is located.

Runnable in Linux In the main program directory of SESToPy is the file *CreateRunnableLinux.sh*. This file is a Linux Shellscript to execute in order to compile a runnable file of this program. Please note, that the runnable is dependent of the processor architecture and the operating system. It is not platform independent anymore. A .tar.gz archive containing the runnable file and all for the execution necessary libraries is created. Compilation is done with PyInstaller.

4 Examples

In engineering, Feature Modeling is widely used. 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). In analogy to the semantics of feature models and mathematical logical expressions, Sections 4.1, 4.2, and 4.7 can be classified as mandatory (logical AND). Sections 4.3, 4.4, 4.5, and 4.6 can be seen as alternative (logical XOR). For expressing optional like 4.8 and OR features like 4.9, an extension of the SES is used. Sections 4.10, 4.11, and 4.12 are combinations of the previous mentioned basic examples.
Next to these basic and combined examples for creating SES in the Sections 4.13 and 4.14 complex examples including flattening are shown. While these examples present simulation-specific component-based systems, in Section 4.15 an example for a non-simulation-specific component-based system is presented. In Section 4.16 examples for usage of underscore attributes and PATH are given. Please note, that any code in this chapter is given as pseudocode.
These examples are released with SESToPy in the JSON format. They are stored in the program directory of SESToPy in a folder called “Examples“.
To keep matters simple, any underscore attributes in children of multi-aspect nodes are left away in the figures up to the example in Section 4.15.

4.1 Example #01: Aspect Node

An SES tree with a single aspect is the simplest example given in Figure 4.1. A 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 referring to a basic model. Note, that the types of the ports of basic models, which the couplings connect, are left away to keep matters simple. Also note, that the derived PES and FPES are 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 FPES. But be aware, that model generation is not included in SESToPy. It is covered in the modelbuilder

software SESMoPy. In the following chapters coupling definitions and the FPES are not given.

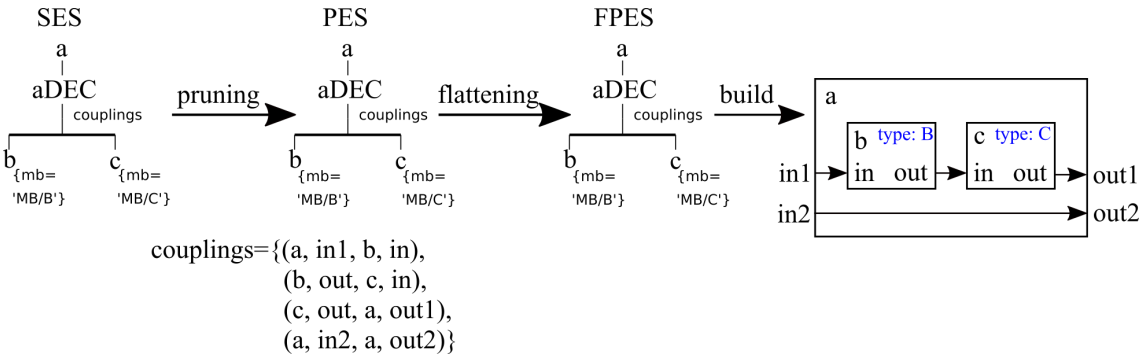


Figure 4.1: Example 01 - Aspect node

4.2 Example #02: Multi-Aspect Node

In Figure 4.2, a similar case to 4.1 is given. The system *as* consists of three children of type *b*. Multi-aspect nodes need to define two special attributes, the attributes numRep and couplings. In this example, the numRep attribute is filled with a hardcoded three, but the value could also be assigned dynamically by using an SESvar or SESfcn. When pruning a multi-aspect node, the numRep attribute is evaluated and an aspect node with children of the same type is created. The derived PES is similar to the PES in Figure 4.1, except that there is a third brother, *b3*, and all children of *as* refer to the same model B in the model base. 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.

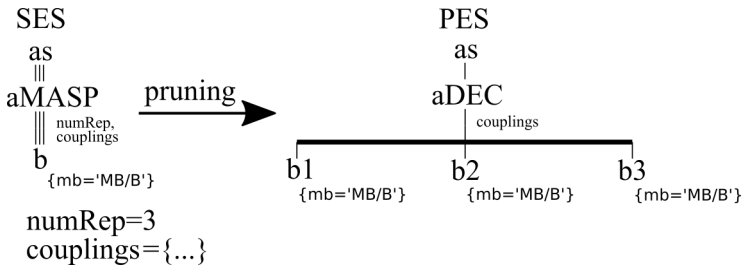


Figure 4.2: Example 02 - Multi-aspect node

4.3 Example #03: Specialization Node

An SES with a specialization node is shown in Figure 4.3. Based on the specialization rule, exactly one child needs to be selected to construct a valid variant. Thus, after pruning, the resulting system can either be of type *b* OR of type *c*. The father inherits the attributes of the child.

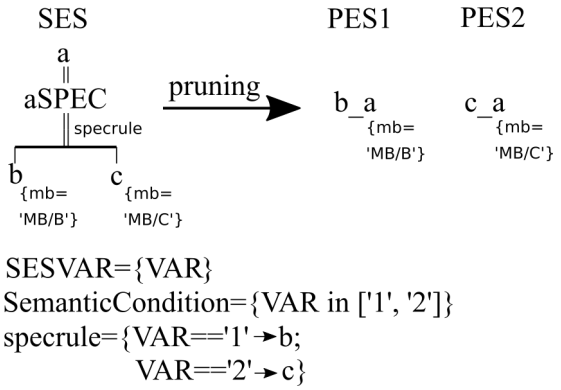


Figure 4.3: Example 03 - Specialization node

4.4 Example #04: 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. This is presented in Figure 4.4. 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.

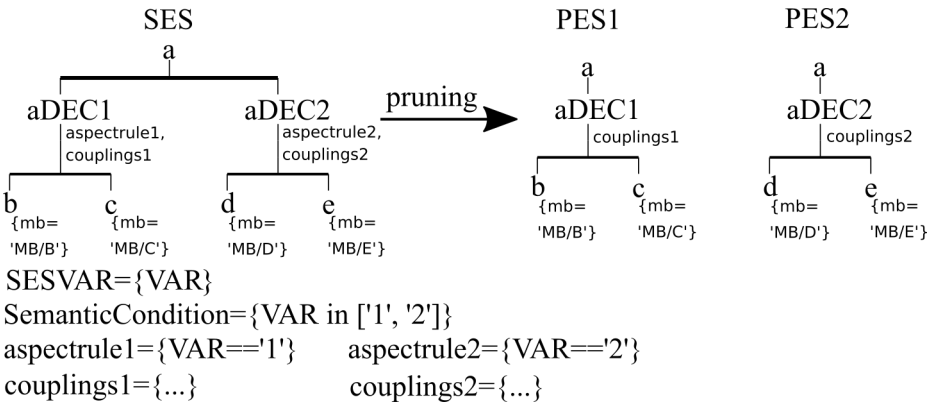


Figure 4.4: Example 04 - Aspect siblings

4.5 Example #05: Multi-Aspect Siblings

In the pattern shown in Figure 4.5, 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 as in Section 4.4 for aspect siblings 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.

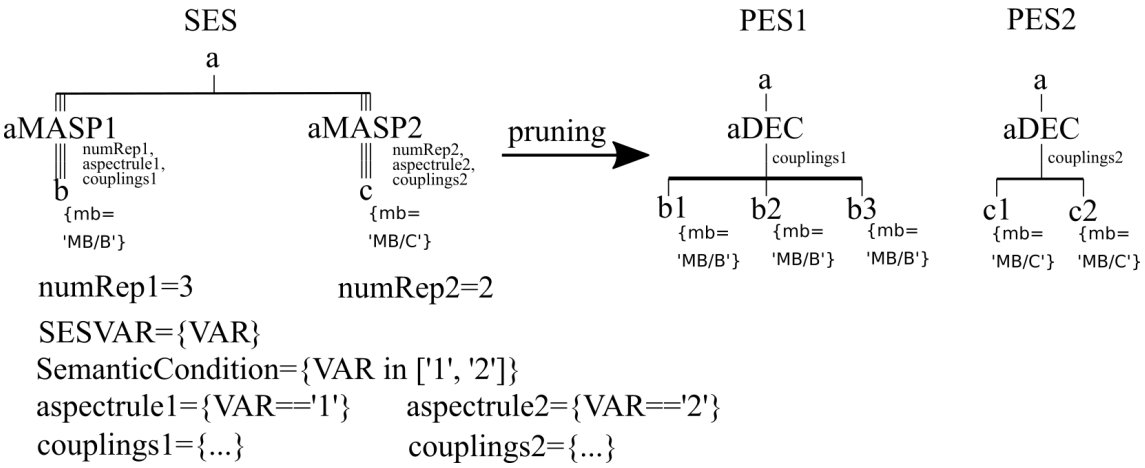


Figure 4.5: Example 05 - Multi-aspect siblings

4.6 Example #06: 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. The example is shown in Figure 4.6. After pruning, the system is built up as before in Section 4.5.

4.7 Example #07: Specialization Siblings

If two or more specialization nodes are on the same hierarchy level, they will all be evaluated. For the example depicted in Figure 4.7, 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 SESvars and the specialization rules. If, for example, VAR1 is set to 1 and VAR2 is set to 2, at specialization *aSPEC1* the left child *b* is

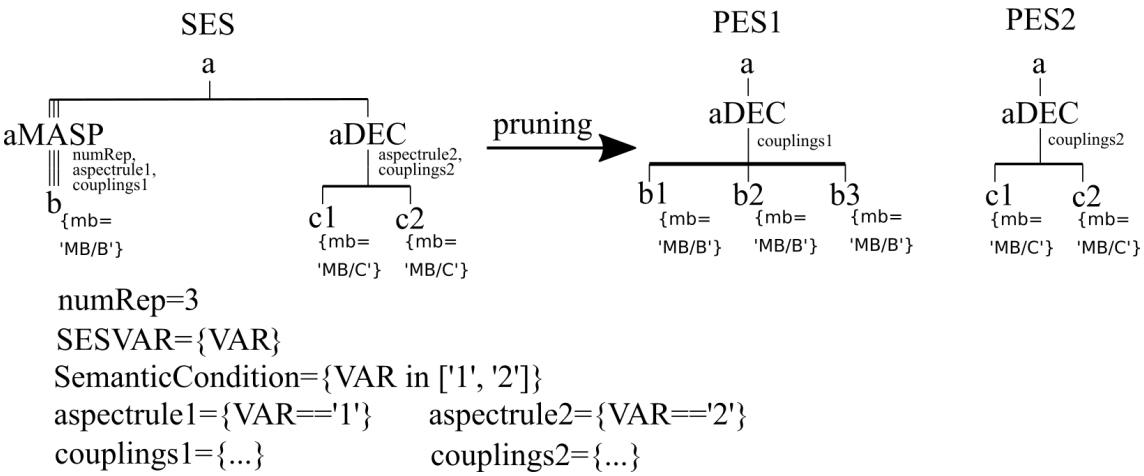


Figure 4.6: Example 6 - Aspect and multi-aspect siblings

selected and, at specialization *aSPEC2*, the right child *e* is selected. During pruning, it depends on the pruning algorithm which of the two specializations is evaluated first. In SESToPy, the left nodes are evaluated first. Therefore in this example, we assume that the left specialization node *aSPEC1* 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.

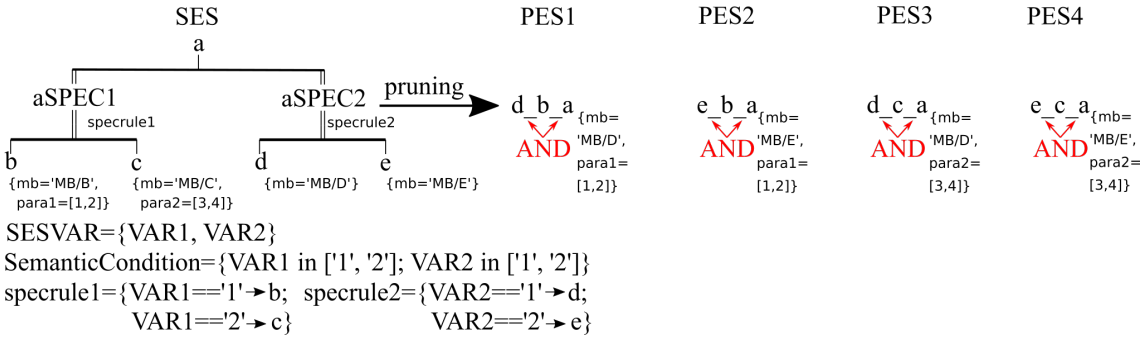


Figure 4.7: Example 07 - Specialization siblings

4.8 Example #08: The NONE Element

The NONE element is an extension of the SES. Nodes can have the name NONE. A NONE element for a leaf entity node means that, if the NONE branch is selected during pruning, the entity is not included at all. In the example shown in Figure 4.8, 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.

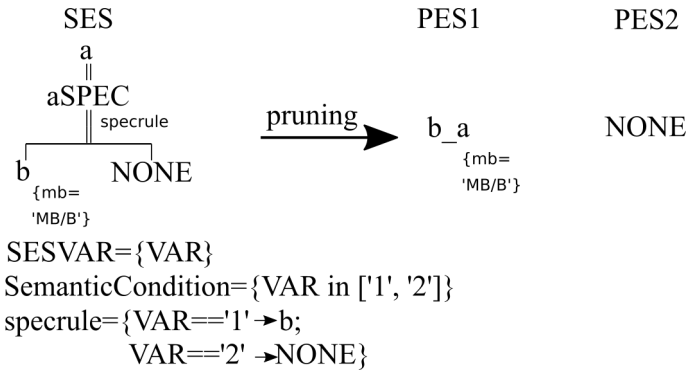


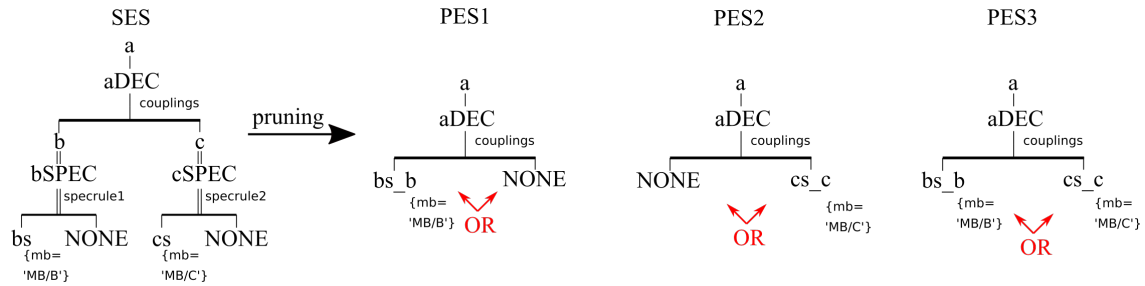
Figure 4.8: Example 08 - The NONE element

4.9 Example #09: Express OR in the SES

The example in Figure 4.9 shows an aspect node whose children are followed by specializations. Each specialization contains a NONE element (see Section 4.8) as one child. For expressing a logical OR, at least one specialization has to evaluate to a node not being NONE, as coded in the semantic condition. By defining the specialization rules reasonably, the user has to ensure, that this is guaranteed and the semantic conditions are met. This example is composed by the example in Section 4.1 in combination with the example in Section 4.8. 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 justified 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 by SES Functions. Examples for variable couplings are shown in Sections 4.13 and 4.14.

4.10 Example #10: Two Specialization Nodes in One Path

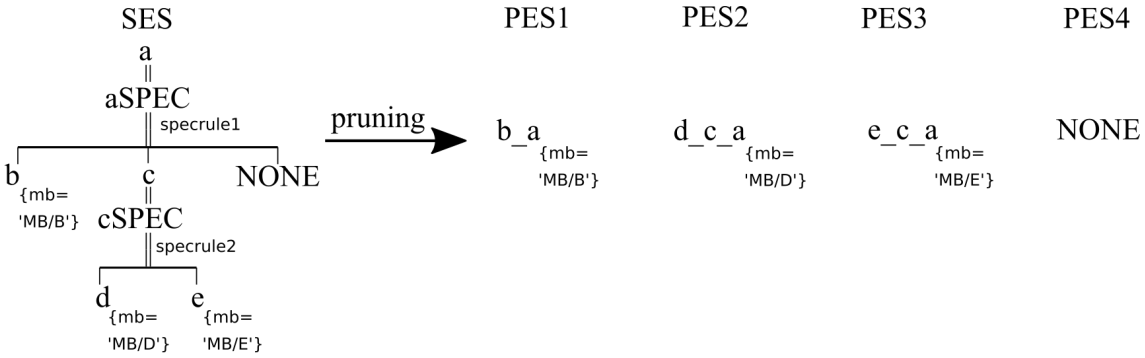
During pruning, specialization nodes inherit the attributes of the selected child and append them to the father's attributes, as previously described. In Figure 4.10, there are two specialization nodes in one path. Additionally, the NONE element is used. This shall clarify the axiom for attribute inheritance. During pruning, firstly



SESVAR={VAR1, VAR2}
SemanticCondition={VAR1 in ['1', '2']; VAR2 in ['1', '2']; not (VAR1 == '2' and VAR2 == '2')}
specrule1={VAR1=='1'➝bs;
VAR1=='2'➝NONE}
specrule2={VAR2=='1'➝cs;
VAR2=='2'➝NONE}
couplings={...}

Figure 4.9: Example 09 - Express OR in the SES

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.



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}

Figure 4.10: Example 10 - Two specialization nodes in one path

4.11 Example #11: Specialization with Succeeding Aspect

Figure 4.11 depicts a specialization node with a succeeding aspect node. This example is a combination of a specialization node followed by a single aspect node. The system *a* can be of type *b* or *c*, the *b* can be decomposed in *d* and *e*.

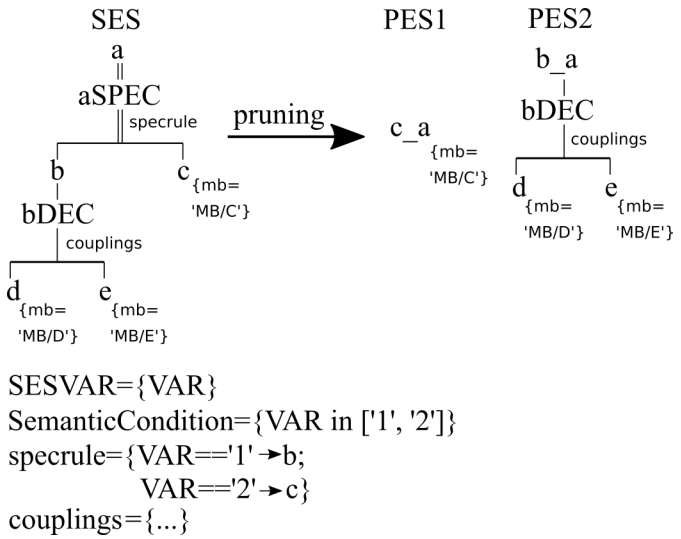


Figure 4.11: Example 11 - Specialization with succeeding aspect

4.12 Example #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 nodes was introduced. Throughout the whole SES, aspect nodes get a unique number. If, during pruning, aspect nodes become brothers, a decision as to which to choose can be found. Nodes with higher priority numbers are priorized over those with lower numbers. In Figure 4.12, 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 f and g. In case that the system is of type e, a decision as to which decomposition to take is made by the priority attribute.

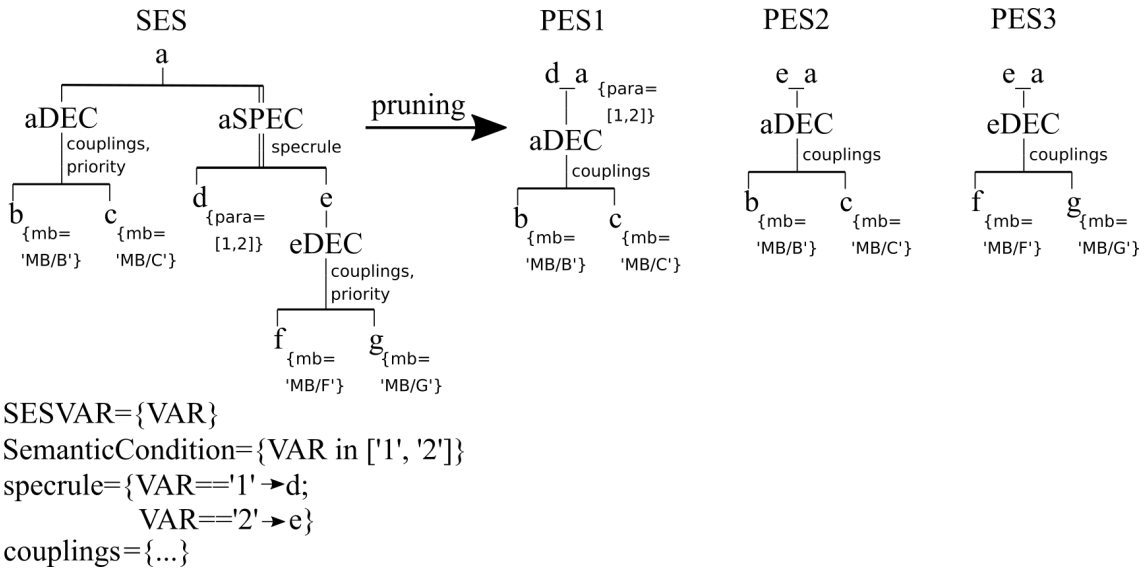


Figure 4.12: Example 12 - Specialization and aspect siblings

4.13 Complex Example Describing a Watershed

A complex example describing a watershed including the complete MB is given in this section. The watershed was described by J.-F. Santucci and L. Capocci at the University of Corse and the SES was developed by T. Pawletta at the University of Applied Sciences Wismar. In Chapter 5 a paper with background information on the watershed is linked.

In this example, models are shown for clearance. Please be aware, that model-building is not included in SESToPy. It is covered in the modelbuilder software SESMoPy. *This example is not discussed in the documentation of SESMoPy.*

The entire SES is presented in Figure 4.13. The node *EWS* is actually a wildcard for an SES being merged with this tree and describes a coupled model therefore, but is not specified here and can be seen as basic model for the purpose of this example.

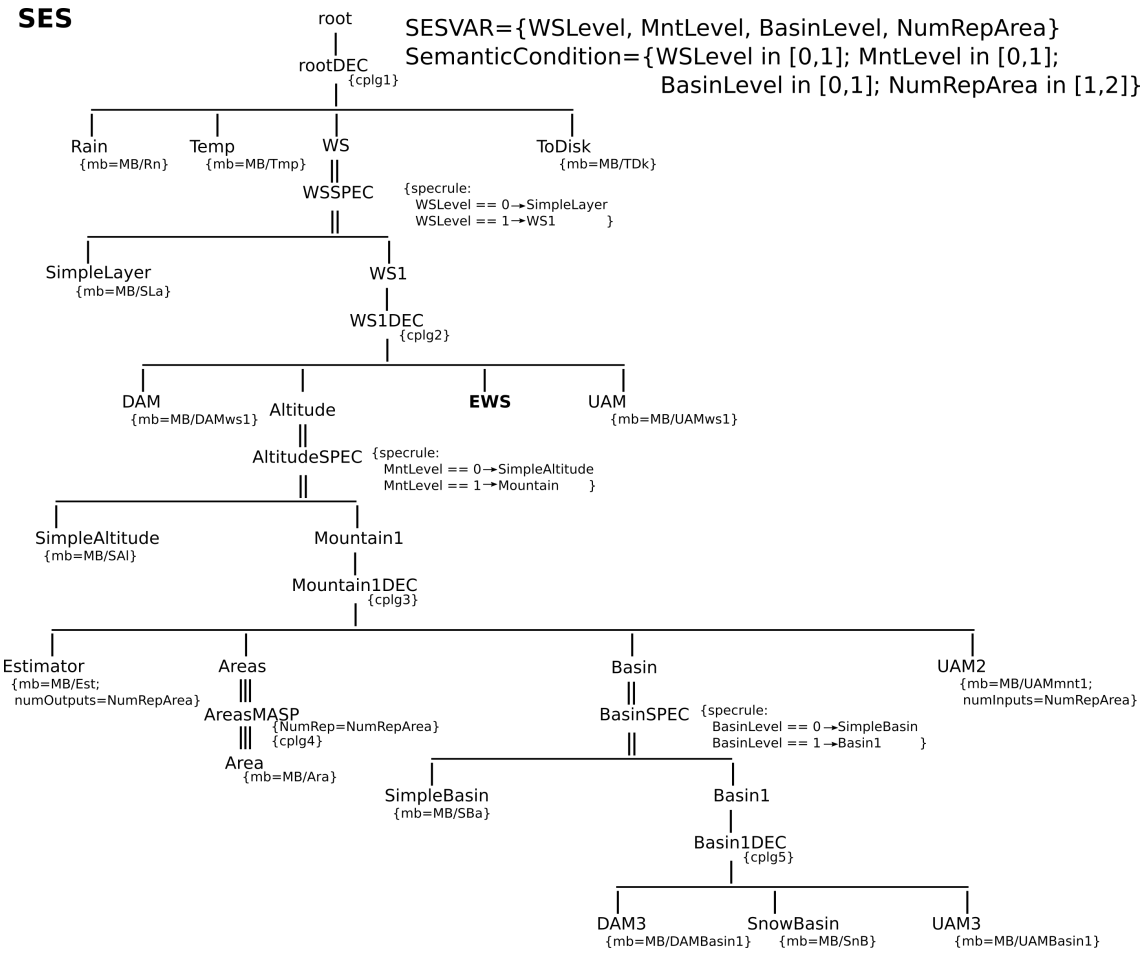


Figure 4.13: Watershed example - entire SES

All couplings are listed in Figure 4.14. Since cplg3 and cplg4 depend on the number of areas to create, it is necessary to set them dynamically by an SESfcn. With the reserved parameters “PARENT“, “CHILDREN“, and “NUMREP“ it is possible to access a node’s parent, its children as a list, and the value of the number of replications of a multi-aspect node. Note, that the types of the ports of basic models, which the couplings connect, are left away in all parts of this example to keep matters simple. Furthermore the couplings have a comment field, which is not depicted in Figure 4.14 as well. The full SES is given as example and the Python 3 code for the SESfcns is presented in the appendix (with the types of the ports and the comment field). An SESfcn for couplings has to output couplings with porttypes and a comment field to be recognized as correct couplingfunction.

```
cplg1={(Rain, out, WS, in1),
       (Temp, out, WS, in2),
       (WS, out, ToDisk, in)}

cplg2={(WS1, in1, DAM, in1),          # EIC
       (WS1, in2, DAM, in2),          # EIC
       (DAM, out1, Altitude, in1),    # IC
       (DAM, out2, Altitude, in2),    # IC
       (DAM, out3, Altitude, in3),    # IC
       (DAM, out4, Altitude, in4),    # IC
       (Altitude, out, EWS, in1),     # IC
       (EWS, out1, UAM, in1),         # IC
       (EWS, out2, UAM, in2),         # IC
       (EWS, out3, UAM, in3),         # IC
       (UAM, out, WS1, out)           # EOC }

cplg5={(Basin1, in1, DAM3, in1),          # EIC
       (Basin1, in2, DAM3, in2),          # EIC
       (DAM3, out1, SnowBasin, in1), # IC
       (DAM3, out2, SnowBasin, in2), # IC
       (SnowBasin, out, UAM3, in),    # IC
       (UAM3, out, Basin1, out1)      # EOC }
```

Couplings that are set via coupling functions:

variable NUMREP refers to the number of replications of the node calling the function -> refers to SESvar NumRepArea

```
cplg3=cplfcn1(CHILDREN,PARENT,NumRepArea)

cplfcn1(children,parent,NUM)

    # parent is Mountain1,
    # children(1) is Estimator,
    # children(2) is Areas,
    # children(3) is Basin,
    # children(4) is UAM2.

    # fixed couplings
    cplg(1)=(parent,in1,children(1),in1)          # EIC
    cplg(2)=(parent,in2,children(1),in2)          # EIC
    cplg(3)=(parent,in3,children(1),in3)          # EIC
    cplg(4)=(parent,in4,children(3),in1)          # EIC
    cplg(5)=(children(1),out1,children(2),in1)    # IC
    cplg(6)=(children(2),out1,children(3),in2)    # IC
    cplg(7)=(children(3),out1,children(4),in1)    # IC
    cplg(8)=(children(4),out,parent,out)          # EOC

    # variable couplings
    if NUM==2
      cplg(9)=(children(1),out2,children(2),in2)   # IC
      cplg(10)=(children(2),out2,children(4),in2) # IC
    end

return(cplg)
```

```
cplg4=cplfcn2(CHILDREN,PARENT,NUMREP)

cplfcn2(children,parent,NUM)

    # parent is Areas,
    # children(1) is Area_1,
    # children(2) is Area_2.

    k=1
    for i in range (1,NUM)
      cplg(k)=(parent,in(i),children(i),in)          # EIC
      cplg(k+1)=(children(i),out,parent,out(i))   # EOC
      k=k+2
    end

return(cplg)
```

Figure 4.14: Watershed example - entire SES couplings

One derived PES is given in Figure 4.15. In the Figure the settings for the SESvars are printed. If the SESvars are set to other values without violating the semantic conditions, other structure variants can be derived for the PES. All possible structure variants, which can be achieved by setting the SESvars, are listed in the appendix. The resulting system variants are presented as well. Stepwise pruning is done.

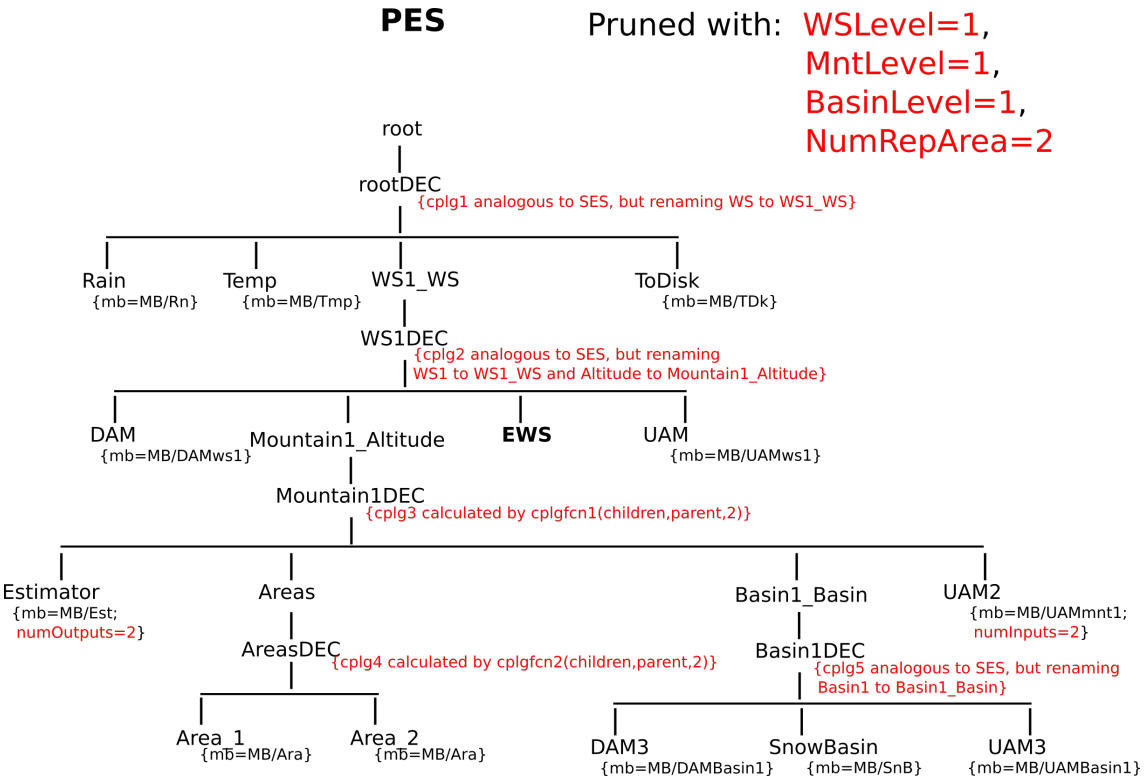


Figure 4.15: Watershed example - entire PES

The couplings of the PES are evaluated like in Figure 4.16.

```
cplg1={(Rain, out, WS1_WS, in1),
       (Temp, out, WS1_WS, in2),
       (WS1_WS, out, ToDisk, in)}

cplg2={(WS1_WS, in1, DAM, in1),                  # EIC
       (WS1_WS, in2, DAM, in2),                  # EIC
       (DAM, out1, Mountain1_Altitude, in1),     # IC
       (DAM, out2, Mountain1_Altitude, in2),     # IC
       (DAM, out3, Mountain1_Altitude, in3),     # IC
       (DAM, out4, Mountain1_Altitude, in4),     # IC
       (Mountain1_Altitude, out, EWS, in1),      # IC
       (EWS, out1, UAM, in1),                    # IC
       (EWS, out2, UAM, in2),                    # IC
       (EWS, out3, UAM, in3),                    # IC
       (UAM, out, WS1_WS, out)                   # EOC }

cplg5={(Basin1_Basin, in1, DAM3, in1),    # EIC
       (Basin1_Basin, in2, DAM3, in2),    # EIC
       (DAM3, out1, SnowBasin, in1),      # IC
       (DAM3, out2, SnowBasin, in2),      # IC
       (SnowBasin, out, UAM3, in),        # IC
       (UAM3, out, Basin1_Basin, out1)  # EOC }
```

Couplings that are found via evaluating coupling functions:

```
cplg3={(Mountain1_Altitude, in1, Estimator, in1),
       (Mountain1_Altitude, in2, Estimator, in2),
       (Mountain1_Altitude, in3, Estimator, in3),
       (Mountain1_Altitude, in4, Basin1_Basin, in1),
       (Estimator, out1, Areas, in1),
       (Areas, out1, Basin1_Basin, in2),
       (Basin1_Basin, out1, UAM2, in1),
       (UAM2, out, Mountain1_Altitude, out),
       (Estimator, out2, Areas, in2),
       (Areas, out2, UAM, in2)      }

cplg4={(Areas, in1, Area_1, in),
       (Area_1, out, Areas, out1),
       (Areas, in2, Area_2, in),
       (Area_2, out, Areas, out2)}
```

Figure 4.16: Watershed example - entire PES couplings

The basic models are stored in an MB, which looks like Figure 4.17.

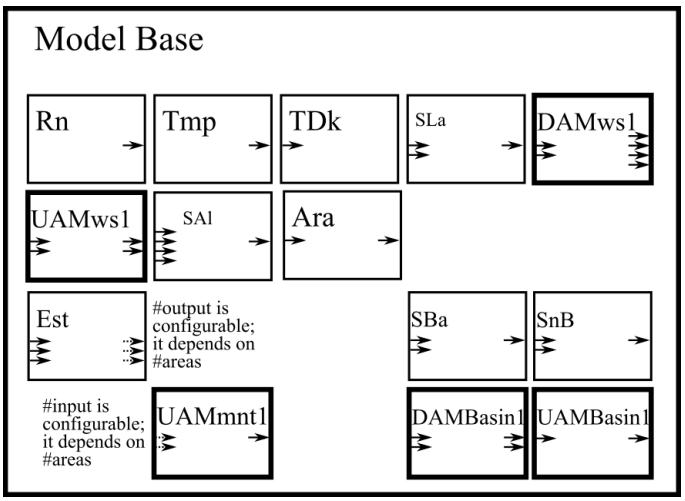


Figure 4.17: Watershed example - MB

The full model, built of the PES in Figure 4.15 and of the basic models in 4.17, looks like Figure 4.18.

Pruned with: WSLevel=1,
MntLevel=1,
BasinLevel=1,
NumRepArea=2

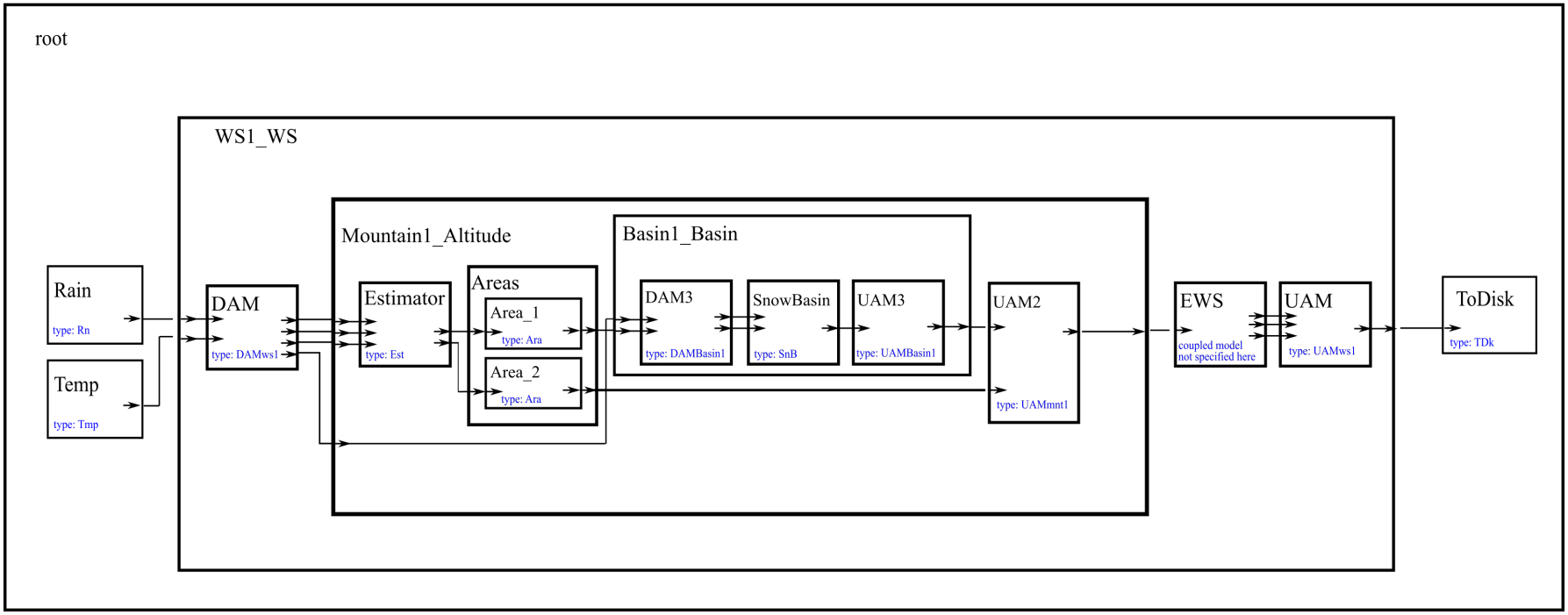


Figure 4.18: Watershed example - the model constructed from the PES

However, the model in Figure 4.18 is a coupled model, in which there are some "containers" describing the structure with no influence on the model. Therefore the PES in Figure 4.15 is flatted and the FPES in Figure 4.19 is created before building the model.

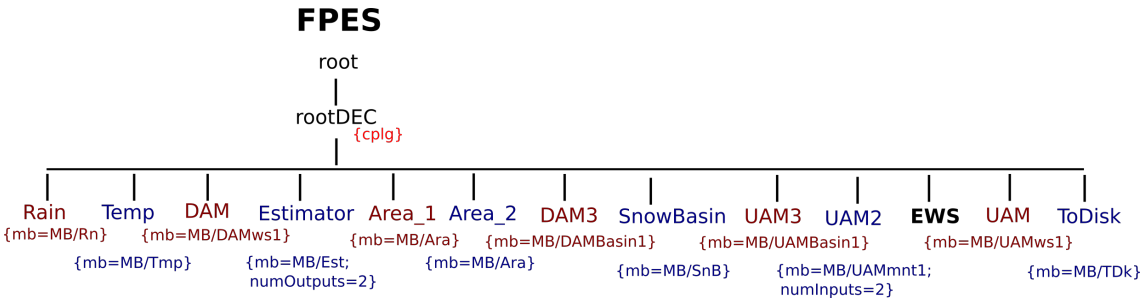


Figure 4.19: Watershed example - entire FPES

This FPES has the couplings as given in Figure 4.20. The complete process of deriving the couplings for the FPES is shown in the appendix.

```
cplg={(Temp, out, DAM, in2),
      (UAM, out, ToDisk, in),
      (Rain, out, DAM, in1),
      (DAM, out2, Estimator, in2),
      (DAM, out4, DAM3, in1),
      (UAM2, out, EWS, in1),
      (EWS, out1, UAM, in1),
      (EWS, out2, UAM, in2),
      (EWS, out3, UAM, in3),
      (DAM, out1, Estimator, in1),
      (DAM, out3, Estimator, in3),
      (Area_1, out, DAM3, in2),
      (UAM3, out, UAM2, in1),
      (Estimator, out2, Area_2, in),
      (Area_2, out, UAM2, in2),
      (Estimator, out1, Area_1, in),
      (DAM3, out1, SnowBasin, in1),
      (DAM3, out2, SnowBasin, in2),
      (SnowBasin, out, UAM3, in)     }
```

Figure 4.20: Watershed example - entire FPES couplings

Creating a model of the FPES presented in Figure 4.19 using the MB in Figure 4.17 and the couplings shown in Figure 4.20, a model as shown in Figure 4.21 is received.

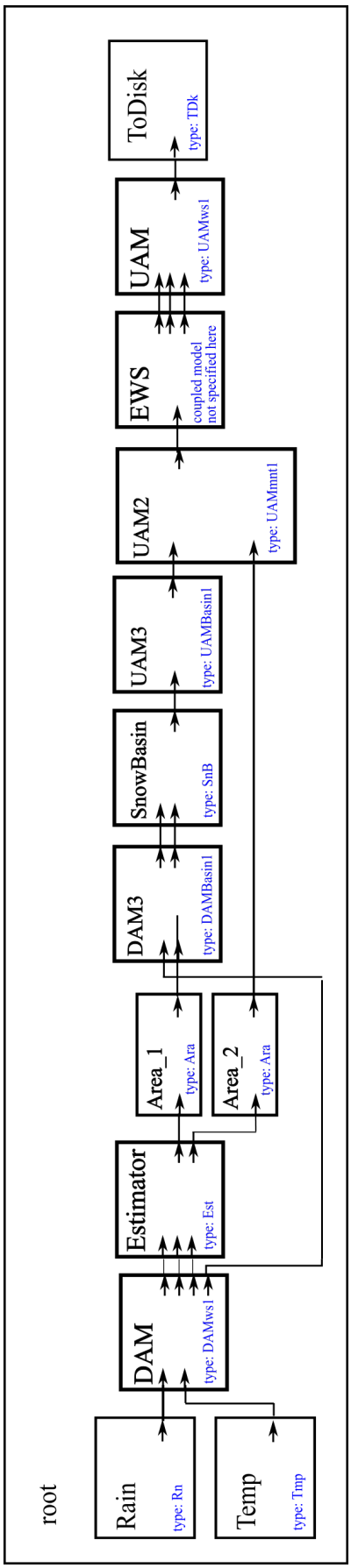


Figure 4.21: Watershed example - the model constructed from the FPES

4.14 Example Describing a Feedback Control System with Optional Feedforward Control

An example describing a feedback control system with optional feedforward control is given in this section. *This example and an SES representing the system are introduced here and modelbuilding is discussed in the documentation of the modelbuilder software SESMoPy with focus on different simulators. Therefore no models are shown here.*
A feedback control system can be modeled using transfer functions describing the behavior of the components in frequency domain. Controlled variables in a feedback control system are usually influenced by disturbances. A common approach for minimizing the influence of predictable disturbances is adding a feedforward control. How such a system can be designed and tested using the eSES/MB infrastructure and the introduced tools is described in the following paragraphs.

4.14.1 Problem Description

A process unit with a *PT1* behavior shall be controlled using a PID controller. A disturbance with a *PT1* behavior affects the output of the process unit. There are two structure variants, either with a feedforward control or without a feedforward control. Figure 4.22 depicts a schematic representation of the application.

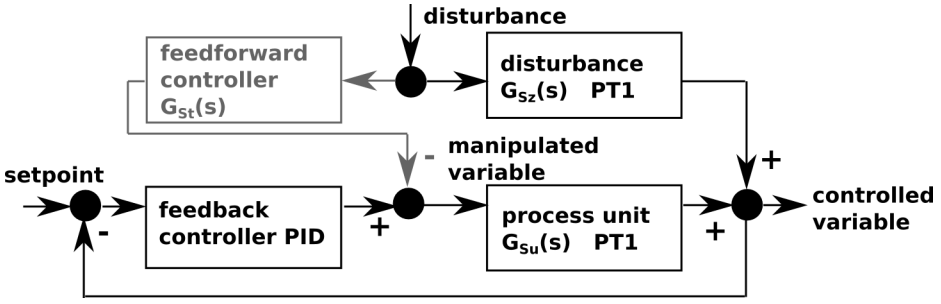


Figure 4.22: Structure of the feedback control system with optional feedforward control.

The system's behavior follows the *PT1* transfer function in equation 4.1 and the step-shaped disturbance affects the output of the process unit with a *PT1* behavior according to equation 4.2. The optional feedforward control is realized by subtracting the disturbing signal calculated by equation 4.3 from the manipulated variable.

$$G_{Su}(s) = \frac{1}{20 \cdot s + 1} \tag{4.1}$$

$$G_{Sz}(s) = \frac{1}{10 \cdot s + 1} \tag{4.2}$$

$$G_{St}(s) = \frac{G_{Sz}(s)}{G_{Su}(s)} = \frac{20 \cdot s + 1}{10 \cdot s + 1} \tag{4.3}$$

4.14.2 Variant Modeling

Figure 4.23 depicts the SES.

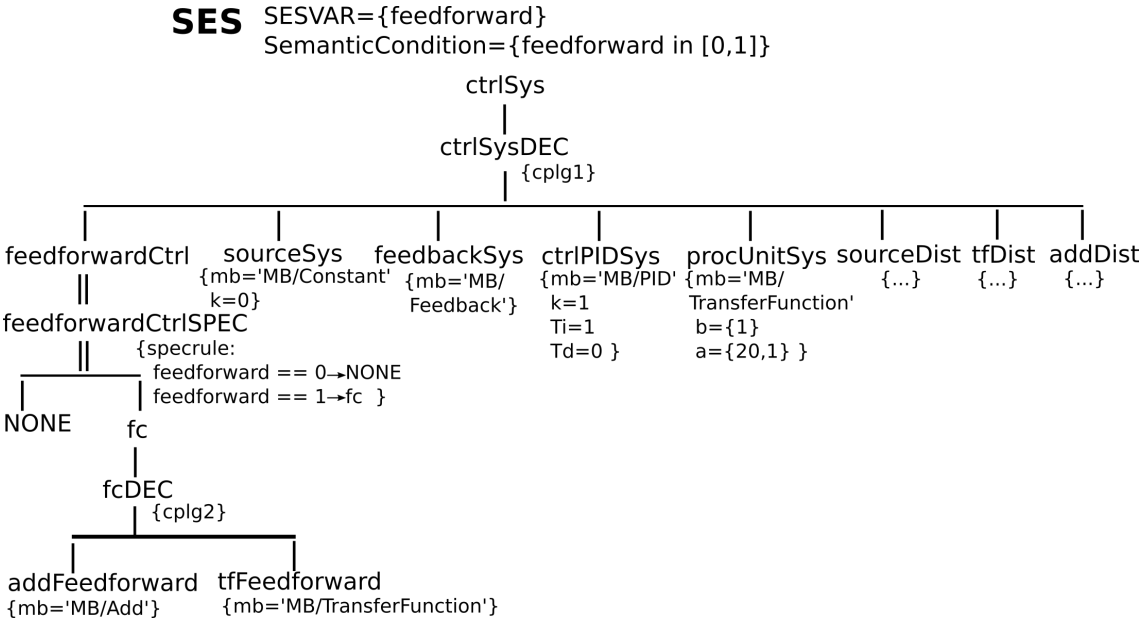


Figure 4.23: SES specifying the feedback control system.

The subtree of the root entity node *ctrlSys* specifies the two system structures, i.e. a variant with and a variant without feedforward controller. The aspect *ctrlSysDEC* describes that each system variant consists of the entities: *sourceSys*, *feedbackSys*, *ctrlPIDSys*, *procUnitSys*, *sourceDist*, *tfDist* and *addDist*. They are mandatory system elements. The optional feedforward control is specified by the subtree of entity *feedforwardCtrl*.

Optional parts in an SES are expressed by a specialization node where one of its children is a NONE element (see example before). A NONE element means that the entity is not included at all. The selection at a specialization is defined by the specrule. The specrule of the specialization *feedforwardCtrlSpec* defines that

either the entity *fc* or *NONE* is selected during pruning. The result of evaluating the specrule at node *feedforwardCtrlSPEC* depends on the value of the SESvar *feedforward*. The SESvar codes the two possible structure variants as values 1 or 0. Therefore, the semantic condition $feedforward \in [0, 1]$ applies to the SESvar. The entity *fc* and its subsequent aspect *fcDEC* specifies the feedforward control structure as a composition of the two entities *tfFeedforward* and *addFeedforward*.

The coupling relations are abbreviated with *cplg* in Figure 4.23. Due to the varying system structures specified in the SES, the couplings in attribute *cplg1* of aspect node *ctrlSysDEC* are defined using an SESfcn. The coupling definitions in *cplg2* at node *fcDEC* are invariable and can therefore be defined without using anSESfcn. The coupling definitions are not listed separately since the SES is given as example.

Each leaf node defines an mb-attribute referring to a basic model in an MB. The other attributes of the leaf nodes define properties to configure the linked basic models.

4.14.3 PES and FPES of the Feedback Control System

As described before, two different structure variants can be derived as PES. These are presented in Figures 4.24 and 4.25.

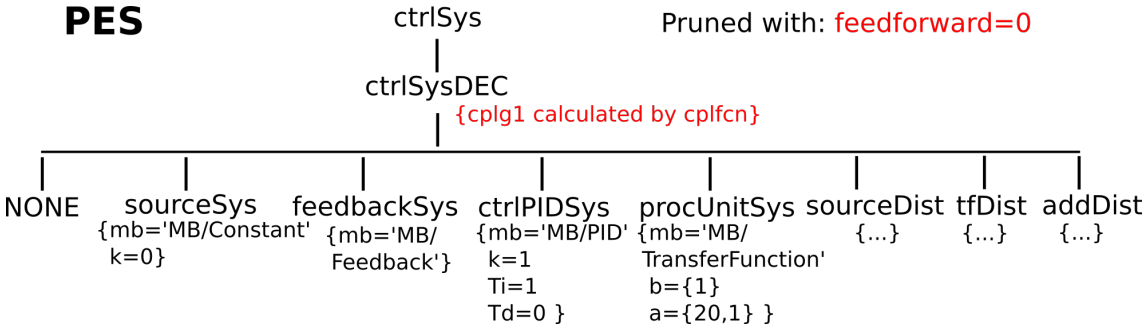


Figure 4.24: PES of the feedback control system (without feedforward control).

In Section 4.13 an example on the advantage of flattening is given. Furthermore it is demonstrated, how the couplings need to be adapted. Figures 4.26 and 4.27 show the structure variants of both FPES.

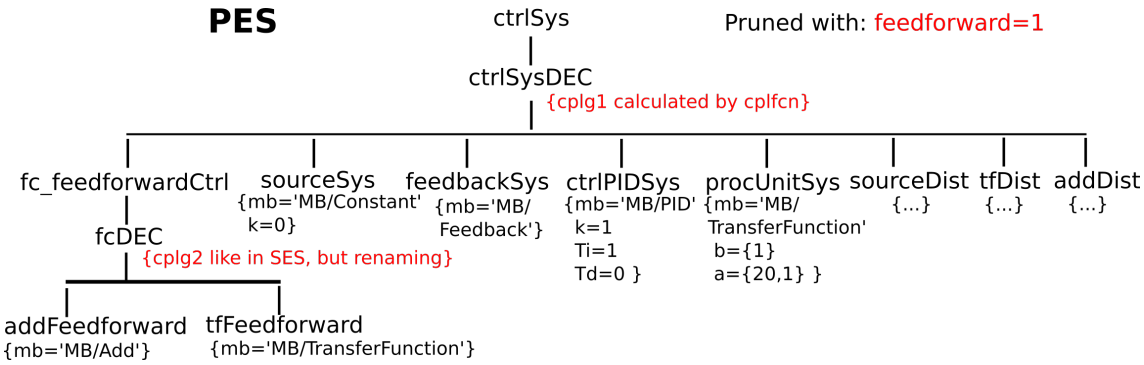


Figure 4.25: PES of the feedback control system (with feedforward control).

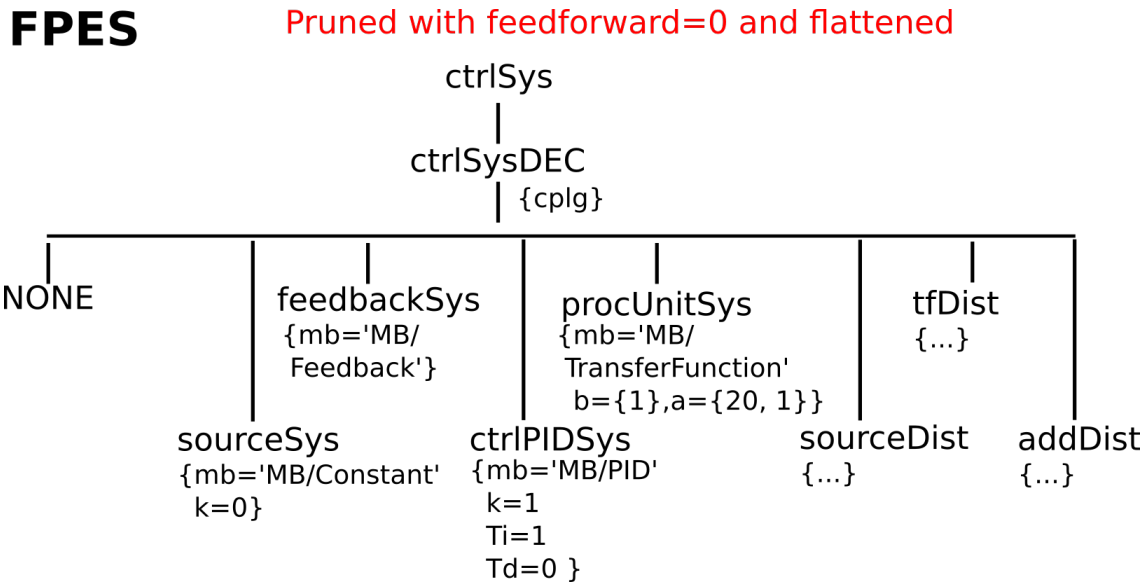


Figure 4.26: FPES of the feedback control system (without feedforward control).

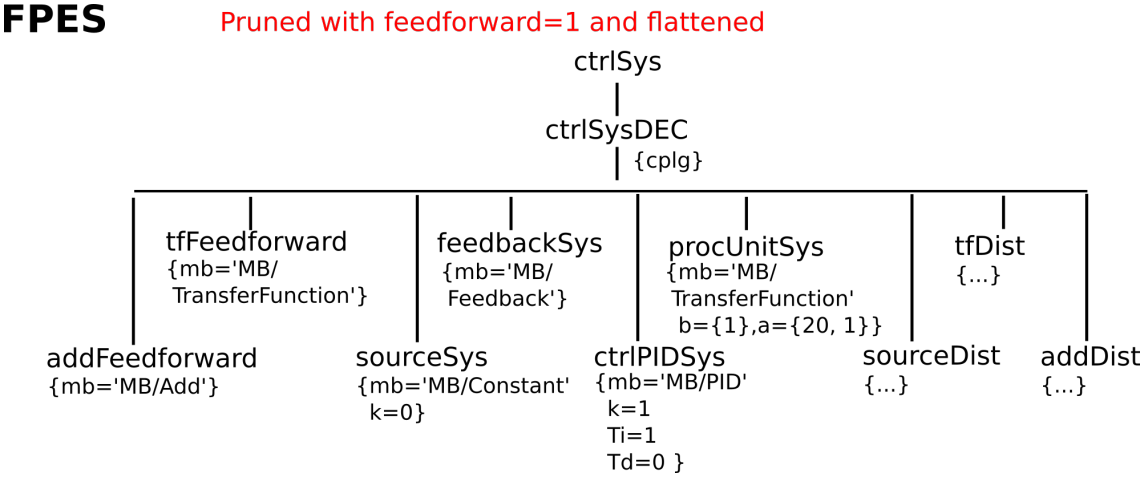


Figure 4.27: FPES of the feedback control system (with feedforward control).

4.15 Example for Variability in Software: A Clock in HTML / JavaScript

This is an example for a non-simulation-specific component-based system. It shows variants of a HTML / JavaScript clock. A clock has a mechanism, can show the date optionally, and the display can be in dark or light style. Furthermore the display has an icon. The mb-attributes refer to software components. There are no couplings between the components. Figure 4.28 shows the SES of the variants of the clock.

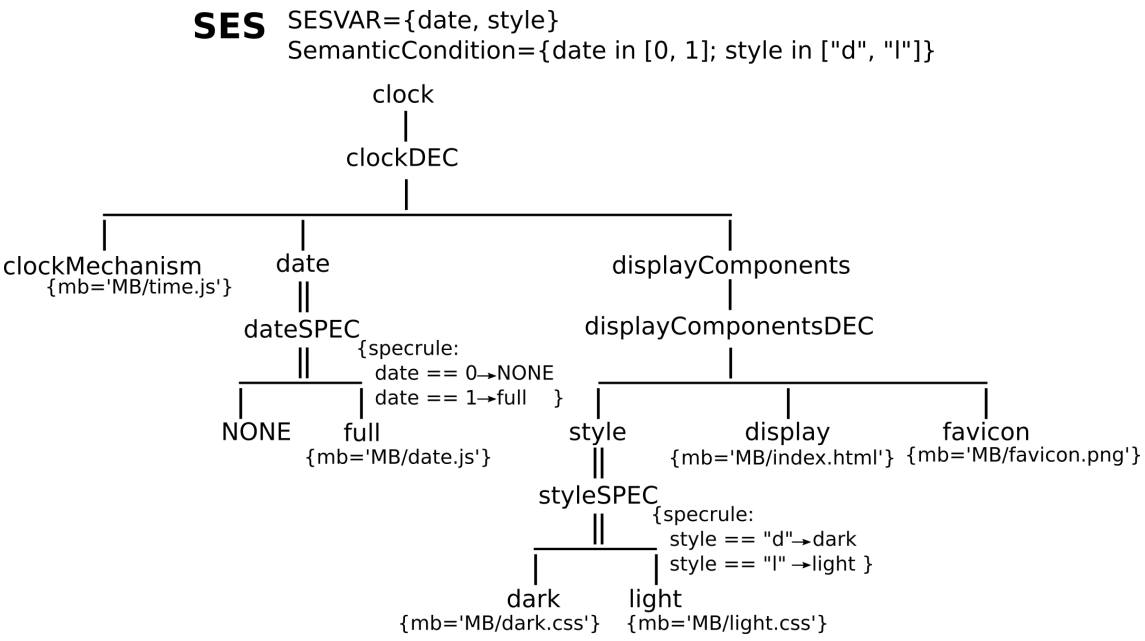


Figure 4.28: SES of the clock.

Possible PES or FPES shall not be given here, please look at the other examples. This example is delivered with the software as well.

4.16 Multi-aspect Nodes with PATH

Examples of hierarchies of multi-aspect nodes in combination with specialization nodes are given in this section. Pruning specializations or aspect brothers leads to reduction in the size of the tree. In contrast, pruning a multi-aspect node leads to expanding the tree by generating a number of children of the multi-aspect node. To illustrate how hierarchies of multi-aspect nodes and specialization nodes can be used for modeling, an artificial example is shown. Furthermore a way to assign different attribute values for the children of multi-aspect nodes and the usage of hierarchy dependent couplings is presented.

For automatic pruning of hierarchies of multi-aspect nodes it may be necessary to access knowledge of the path a node is in. Therefore children of multi-aspect nodes get an underscore attribute, which has the name of the childnode and is initialized with “0“ in the SES. During pruning this attribute gets the number of the generated child of a multi-aspect node as string value. These underscore attributes can only be accessed with the parameter “PATH“. PATH is a dictionary containing the names of underscore attributes in the path from the current node to the root as keys and the number of the during pruning created child of a multi-aspect node as value. The number of the created child in this dictionary is a string value. To be used as index, this value needs to be converted to an integer value. Conversion to integer values is left away in the functions in the figures in this section. Remember that the values of the underscore variables start with 1 (the first during pruning generated child of an multi-aspect node gets number “1“), while indices in Python lists start with the index 0. Therefore mapping the number which is read from an underscore variable to a Python list requires the subtraction of 1.

The use of underscore attributes and PATH is clarified in the following examples. In the first examples the SES is pruned stepwise and one possible PES is depicted in each figure. In each pruning step it is shown how to assign the SESvars in the pruning process to gather the depicted PES. The term pool in the examples is subsequently used for a related set of resources, such as a set of computers in a laboratory. Consequently, the term pools refers to a set of pools. The example in Section 4.16.9 shows variants of a chemical plant and does not show pruning steps, but the SES and one possible PES.

4.16.1 Multi-aspect

Figure 4.29 shows the simplest case using a multi-aspect node. A system of *pools* consisting of a variable number of *pool* is described. In the multi-aspect node *poolsMASP* the number of replications attribute numRep is defined by the SESvar *NumPools*. It must be an element of $\mathbb{N}$ as stated in the Semantic Condition. Depending on *NumPools* a number of *pool* is generated during pruning. In Figure 4.29 a constant value two is assigned to *NumPools*. Thus two pool entities are generated during pruning, *pool_1* and *pool_2*. The multi-aspect node *poolsMASP* is changed to the aspect node *poolsDEC*. The number of the created pool is assigned to the attribute *_pool*.

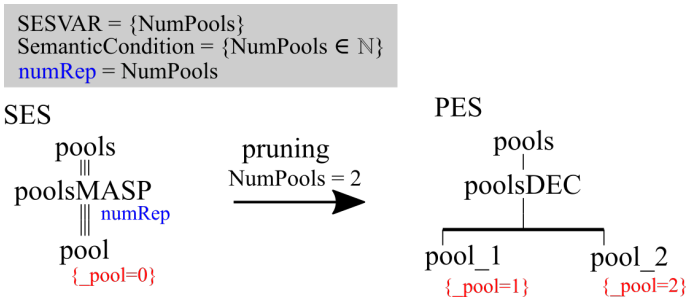


Figure 4.29: SES with a multi-aspect node and a possible PES.

4.16.2 Multi-aspect with Different Attribute Values for Each Child

This example demonstrates how to assign attributes with different values using underscore attributes and PATH. The SES in Figure 4.30 is the same as the example in Section 4.16.1, only the leaf entitiy has an attribute "mb" which has the SESfcn *fun()* as value. Next to the SESvar *NumPools* as known from Section 4.16.1 another SESvar *Types* is defined. *Types* consists of a list with 'nb' or 'dt' elements and has as many elements as the value in *NumPools*. It describes the type of each pool being generated during pruning. In the leaf node *pool* the SESfcn *fun()* is called with the parameters PATH and *Types*. During pruning *pool_1* and *pool_2* are generated, the multi-aspect node is changed to aspect node, and the underscore attributes are assigned as described in Section 4.16.1. The SESfcn *fun(PATH, Types)* reads the underscore attribute of *pool_1* respective *pool_2* and finds the corresponding element in *Types*. A string depending on the path and thus the type of the pool is returned. The string is then assigned to the mb-attribute.

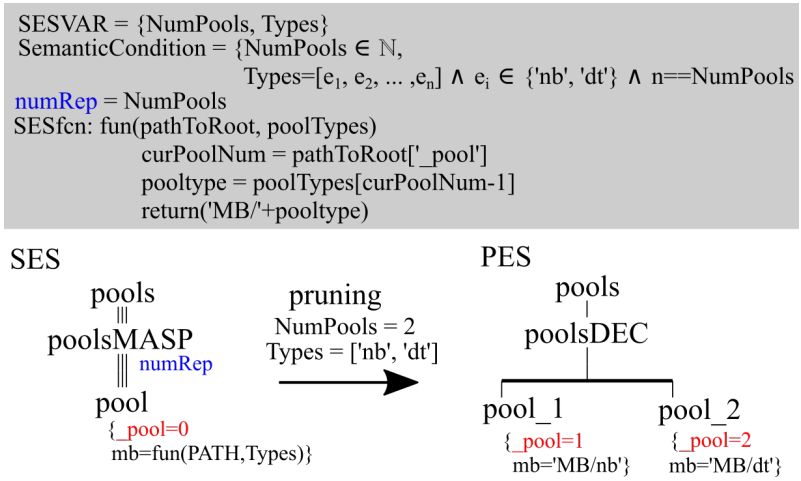


Figure 4.30: SES with a multi-aspect node and a possible PES with different attribute values for each child.

4.16.3 Multi-aspect Followed by a Specialization Node

The example from Section 4.16.1 is expanded by a further layer. A *pool* shall specialize in *notebooks* or *desktops*. Therefore the SES is extended with a specialization node *poolSPEC* as depicted in Figure 4.31. The SESvars *NumPools* and *Types* as discussed in Sections 4.16.1 and 4.16.2 are defined. The node *poolSPEC* defines a specrule with the SESfcn *fun()*. The selection of one child depends on the path in which the specrule in *poolSPEC* is called. While the first pruning step is the same as in Section 4.16.1 leading to *pool_1* and *pool_2* with the underscore attributes *_pool*, in the second pruning step the specrule in *poolSPEC* is evaluated. The SESfcn *fun(PATH, Types)* reads the underscore attribute *_pool* in the *pool_i* node in the current path and returns the corresponding element in *Types*. Thus the returned value depends on the pool number. In case the returned value is 'nb' the node *notebooks* is selected, if the returned value is 'dt' the node *desktops* is selected. Other children of the specialization *poolSPEC* are cut away. Finally *poolSPEC* is resolved and the selected child is united with the respective parent.

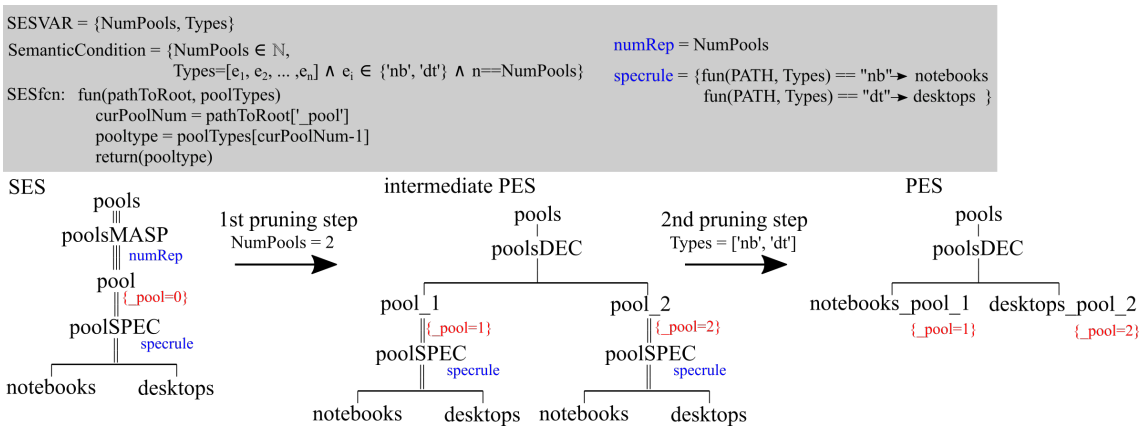


Figure 4.31: SES with a multi-aspect node, a succeeding specialization node and a possible PES.

4.16.4 Multi-aspect Followed by Aspect Brothers

The example in Section 4.16.3 can be expressed with aspect node brothers as depicted in Figure 4.32. The SES in Section 4.16.1 is expanded with aspect node brothers, *notebooksPoolDEC* and *desktopsPoolDEC*. The SESvars *NumPools* and *Types* as discussed in Sections 4.16.1 and 4.16.2 are defined. For each aspect node an aspectrule is defined. Both aspectrules call the SESfcn *fun()*. The selection of one aspect node depends on the path in which the aspectrule in *notebooksPoolDEC*

respective *desktopsPoolDEC* is called. While the first pruning step is the same as in Section 4.16.1 leading to *pool_1* and *pool_2* with the underscore attributes *_pool*, in the second step the aspectrules in *notebooksPoolDEC* and *desktopsPoolDEC* are evaluated. The SESfcn *fun(PATH, Types)* reads the underscore attribute *_pool* in the *pool_i* node in the current path and returns the corresponding element in *Types*. Thus the returned value depends on the pool number. In case the returned value is 'nb' the aspectrule in the node *notebooksPoolDEC* evaluates to true, if the returned value is 'dt' the aspectrule in the node *desktopsPoolDEC* evaluates to true. If the rule evaluates to true, the node is selected, the other brother is neglected.

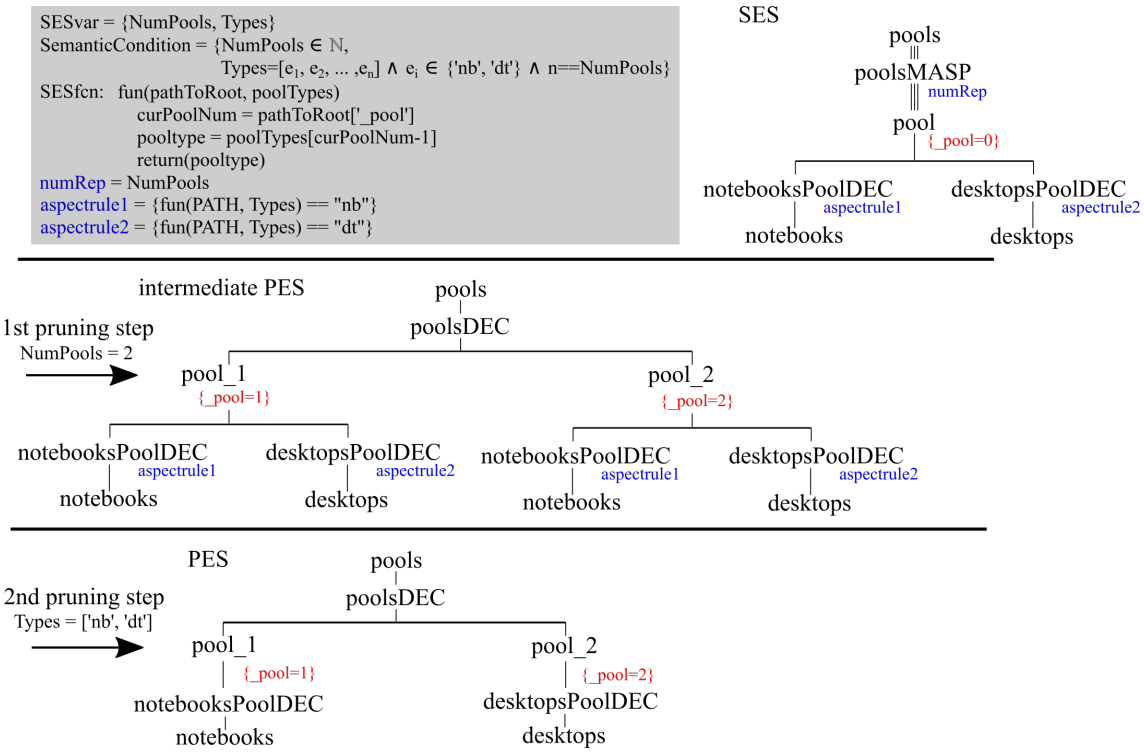


Figure 4.32: SES with a multi-aspect node, succeeding aspect brothers, and a possible PES.

4.16.5 Multi-aspect Followed by Specialization and Multi-aspect Nodes

The example in Section 4.16.3 is extended by one more layer. A *notebooks* pool and a *desktops* pool shall have a number of *notebook* respective *desktop* computers. Therefore the SES is extended with multi-aspect nodes *notebooksMASP* and *desktopsMASP* as depicted in Figure 4.33. The SESvars *NumPools* and *Types* as discussed in Sections 4.16.1 and 4.16.2 are defined. Furthermore an SESvar *NumRepInPool*

defining the number of computers in each pool needs to be set. *NumRepInPool* is a list with elements of type $\mathbb{N}$ and has as many elements as the value in *NumPools*. The value numRep for the number of replications in *notebooksMASP* and *desktopsMASP* are set by the SESfcn *fun2()*. The value for numRep depends on the path in which *fun2()* in the numRep value of *notebooksMASP* and *desktopsMASP* is called. While the first and the second pruning step are the same as in Sections 4.16.1 and 4.16.3 leading to *notebooks_pool_1* and *desktops_pool_2* with the underscore attributes *_pool*, in the third pruning step the numRep values in *notebooksMASP* and *desktopsMASP* are calculated. The SESfcn *fun2(PATH, NumRepInPool)* reads the underscore attribute *_pool* in the *pool_i* node in the current path and returns the corresponding element in *NumRepInPool*. Thus the returned value depends on the pool number. The number of *notebook* or *desktop* entities are created below the multi-aspect nodes. The multi-aspect nodes *notebooksMASP* and *desktopsMASP* are changed to aspect nodes and the underscore attributes in *notebook* and *desktop* are assigned with the number of the created computer.

4.16.6 Multi-aspect Followed by Specialization and Sequenced Multi-aspect Nodes

The SES from Section 4.16.5 is extended again with a new layer. A *notebook* or a *desktop* can be detailed with a number of different processor *cores*. Therefore the SES is extended with multi-aspect node *coresMASP* in both branches as depicted in Figure 4.34. Due to the axiom of uniformity, nodes with the same name need to have the same variables and isomorphic subtrees. The SESvars *NumPools*, *Types*, and *NumRepInPool* as discussed in Sections 4.16.1, 4.16.2, and 4.16.5 are defined. Furthermore an SESvar *NumCores* defining the number of processor cores needs to be set. *NumCores* is a list of vectors. The number of list elements is equal to the number of pools, defined by *NumPools*. The number of elements in each vector is equal to the number of entities in the corresponding pool, defined by *NumRepInPool(i)*. Each element in the vector is of type $\mathbb{N}$. The value numRep for the number of replications in *coresMASP* in each path is set by the SESfcn *fun3()*. The value for numRep depends on the path in which *fun3()* in the numRep value of *coresMASP* is called. While the first, the second, and the third pruning step are the same as in Sections 4.16.1, 4.16.3, and 4.16.5 leading to a number of *notebook* and *desktop* entities with the underscore attributes *_pool*, *_notebook*, and *_desktop*, in the fourth pruning step the numRep values for *coresMASP* in each path are calculated. The SESfcn *fun3(PATH, NumCores)* reads the underscore

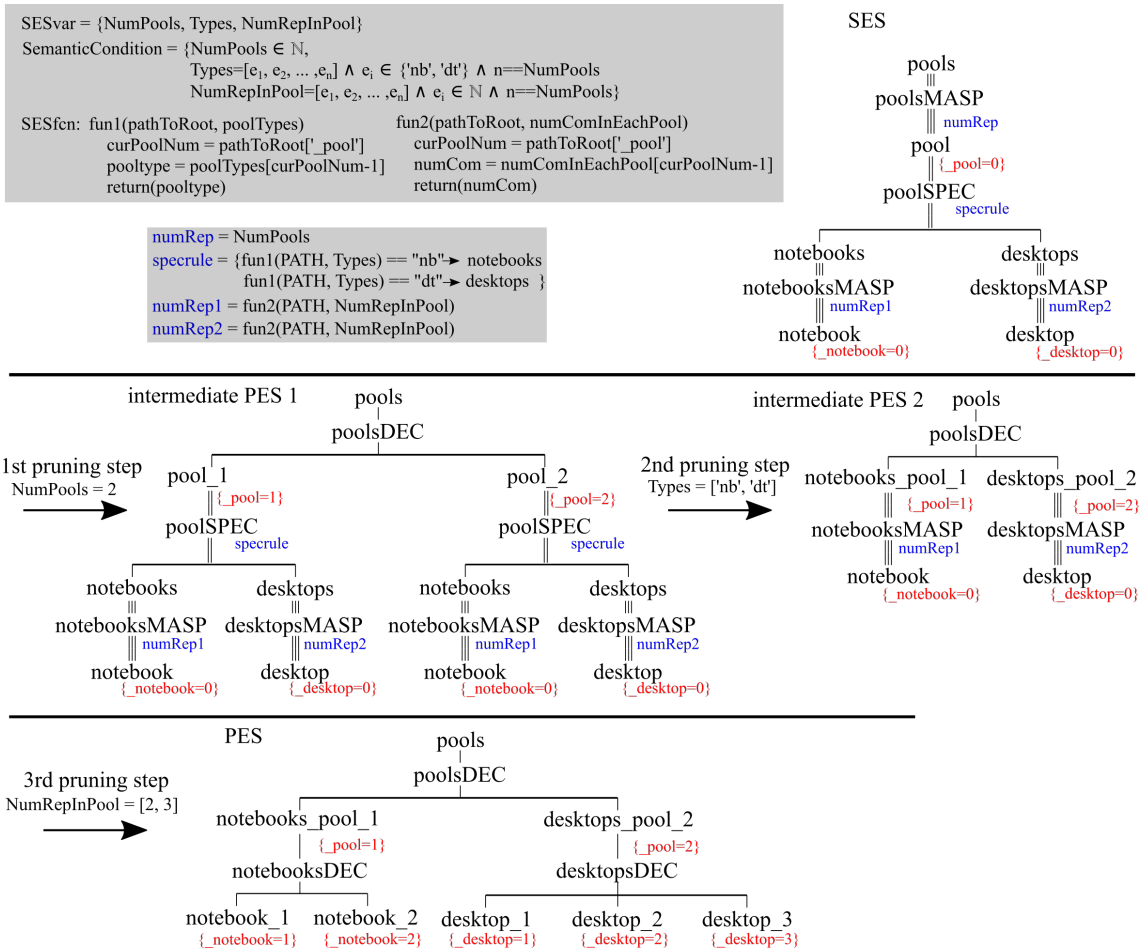


Figure 4.33: SES with a multi-aspect node and a succeeding specialization node followed by multi-aspect nodes and a possible PES.

attribute *_pool* in the *pool_i* node in the current path. If the current path has the entity *notebook*, PATH has the *_notebook* element, if the current path has the entity *desktop*, PATH has the *_desktop* element. The SESfcn *fun3(PATH, NumCores)* tries to access either the *_notebook* or the *_desktop* element and reads the computer number. Then the element in *NumCores* at the position of the pool number and the position of the computer number is returned. Thus the returned value depends on the pool number and the computer number. The number of *core* entities are created below the multi-aspect nodes *coreMASP* in each path. The multi-aspect node *coreMASP* is changed to aspect nodes in each path and the underscore attribute in *core* is assigned with the number of the created core.

In this example it becomes clear that in the PES the axiom of uniformity needs to be relaxed. Nodes with the same name have no isomorphic subtrees in this example.

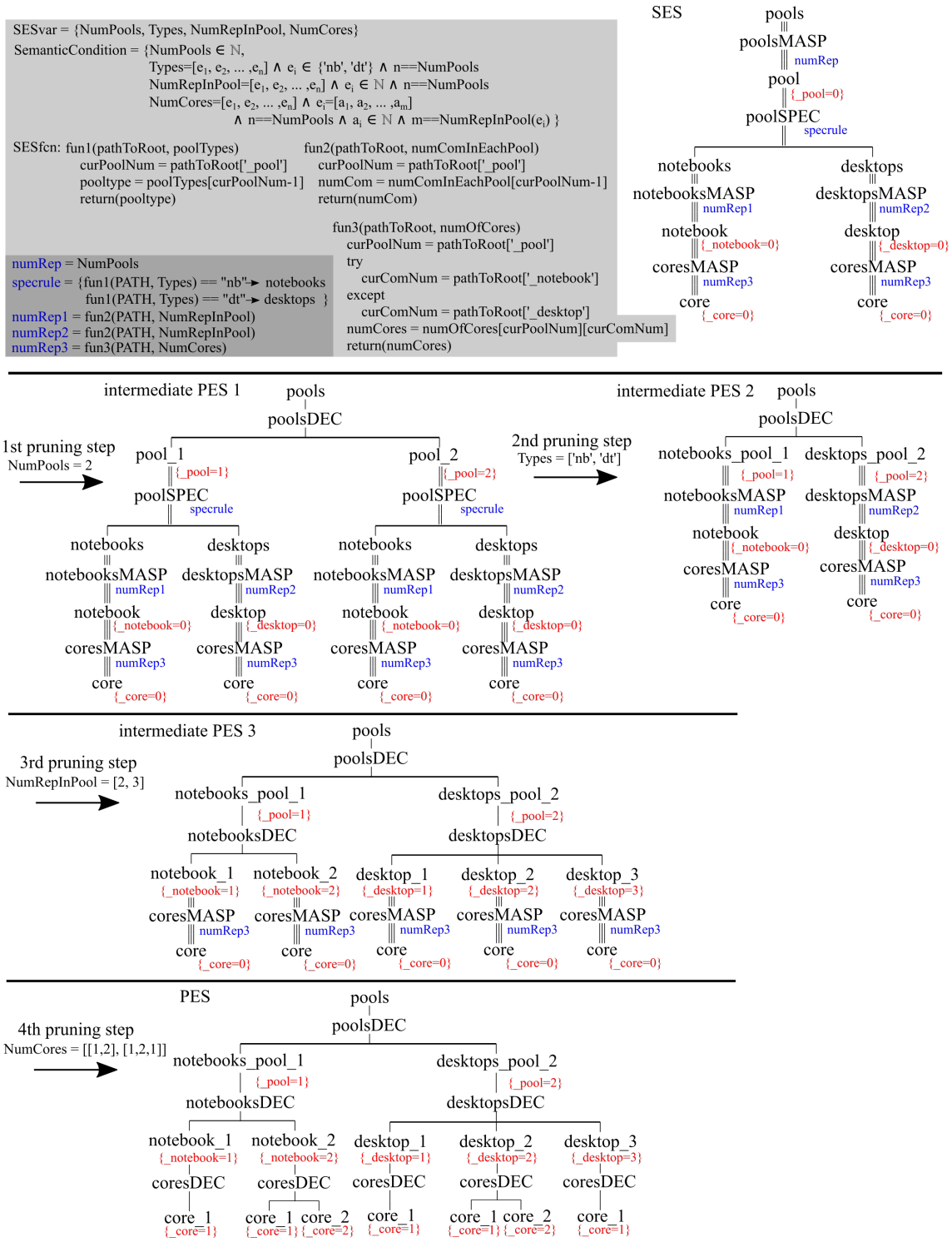


Figure 4.34: SES with a multi-aspect node and a specialization node followed by sequenced multi-aspect nodes and a possible PES.

4.16.7 Using PATH in Priority

In Figure 4.35 an example for setting priority depending on the selection in *poolSPEC* is given. This example is based on the example for priority in Section 4.12. It is extended with the *root* entity node and the *rootMASP* multi-aspect node. The number of replications in the *rootMASP* are defined by the SESvar *NumA*, the specrule evaluates the value in the SESvar *VAR*. *NumA* is hardcoded with the value two. In the first pruning step the entity nodes *a_1* and *a_2* are created with their

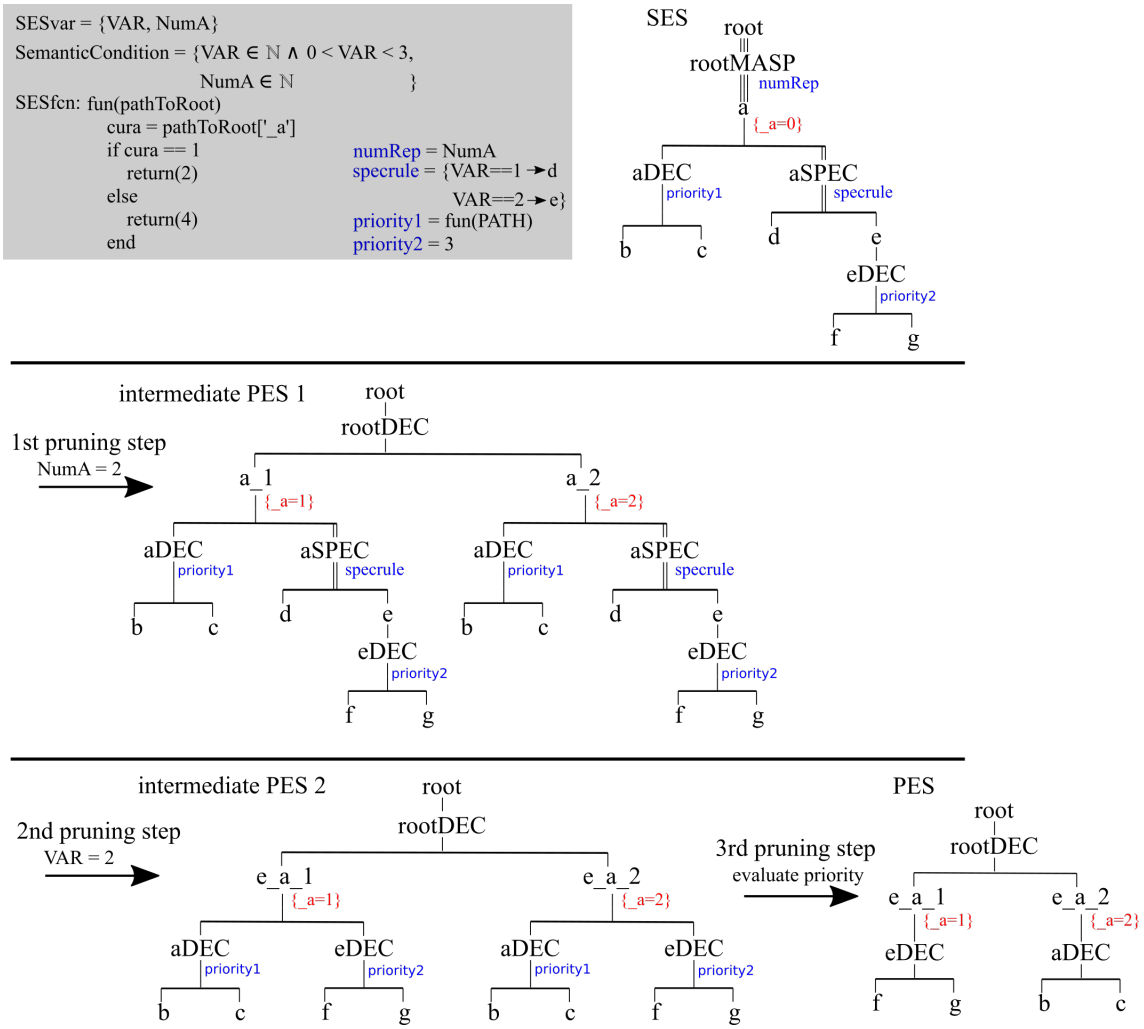


Figure 4.35: Using PATH in priority.

subtrees. In the nodes *a_1* and *a_2* the underscore attribute *_a* is assigned with the number of the created node in each paths. When aspect node and specialization nodes are brothers, the specialization nodes have to be pruned first. Thus in the second pruning step the specialization node *aSPEC* is resolved as described in Section 4.12. If *VAR* is two like in Figure 4.35, the aspect nodes *aDEC* and *eDEC*

become brothers. Since they were no brothers in the SES, no aspectrule is defined. Therefore they have a priority, which is set by the SESfcn *fun()* in the node *aDEC*. The function *fun(PATH)* gets the value of *_a* and returns the priority value two or four. Thus for the node *aDEC* the priority value depends on the number of the node *a*. The node *eDEC* has a hardcoded three for the priority. In the third pruning step the priority is evaluated. A higher value means a higher priority. In the path with the node *e_a_1* the node *aDEC* gets a priority of two, which is a lower priority than the hardcoded three in *eDEC*. In the path with the node *e_a_2* the node *aDEC* gets a priority of four, which is a higher priority than the hardcoded three in *eDEC*.

4.16.8 Using PATH in Couplings

In Figure 4.36 an example for couplings depending on the selection in *poolSPEC* is given. It is demonstrated how a varying number of *notebook / desktop* is coupled to the *notebooks / desktops* entity using only one function defining the couplings. Because of the variability of the number of *notebook / desktop* the coupling needs to be set dynamically by an SESfcn. With the reserved parameters "PARENT", "CHILDREN", "NUMREP", and "PATH" it is possible to access a node's parent, its children as a list, the value of the number of replications of a multi-aspect node, and the dictionary with the underscore attributes from the current node to the root. Like in the example in Section 4.13 the types of ports of basic models, which the couplings connect, and the comment field of couplings are left away to keep matters simple. An SESfcn for couplings has to output couplings with porttypes and a comment field to be recognized as correct couplingfunction in SESToPy.

In the SES in Figure 4.36 the calls of *cplfun* are depicted as *couplings1* and *couplings2*. They link to the couplings of the nodes *notebooksMASP* and *desktopsMASP*. The couplings of both nodes are defined by only one couplingfunction. Thus the function needs to depend on the path of the node. For this the parameter PATH is passed next to other necessary parameters. The functions sets the couplings depending on the path and thus the type and the number of the computers in the pool.
In Figure 4.37 the resulting model is depicted. A model of a *notebook* has two *in* ports and one *out* port, while a model of a *desktop* has one *in* port and one *out* port. Thus the number of *in* ports of *notebooks* or *desktops* depends on the type and the number of *out* ports of *notebooks* or *desktops* depends on the number of *notebook*

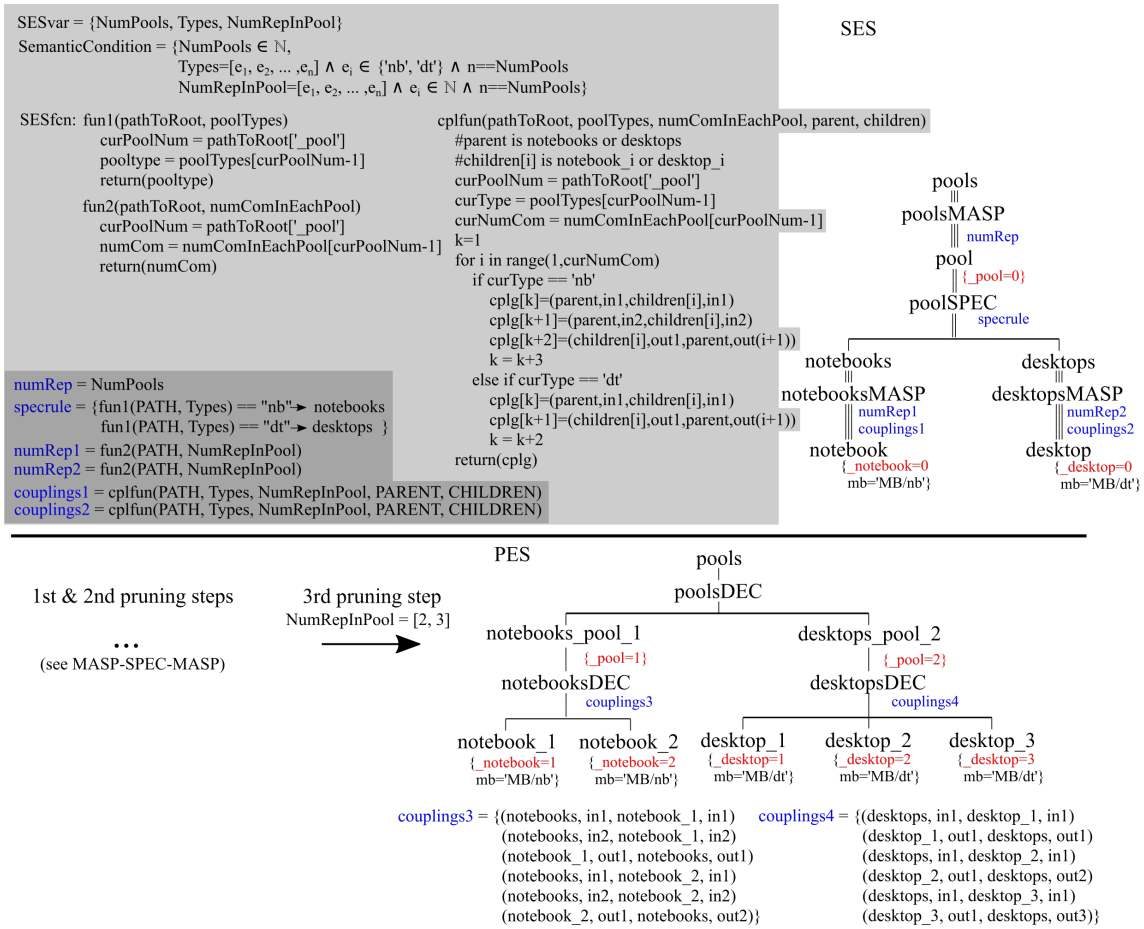


Figure 4.36: Using PATH in couplings.

and *desktop*. The *cplfun* needs to evaluate the couplings like that.

The model is shown here for clearance. Please be aware, that modelbuilding is not included in SESToPy. It is covered in the modelbuilder software SESMoPy. During pruning the function is evaluated and a list with couplings is generated. The resulting couplings are shown as *couplings3* and *couplings4* in the PES in Figure 4.36.

4.16.9 Modeling a Chemical Plant

This example demonstrates a use case of an SES with a hierarchy of multi-aspect nodes as discussed before. It specifies variants of a chemical plant. The full SES and one possible PES are demonstrated in Figures 4.38 and 4.39.

A *plant* can have several *partialPlants*. This is specified by the multi-aspect node *plantMASP*. The number of partial plants is set in the SESvar *NumPartialPlants*.

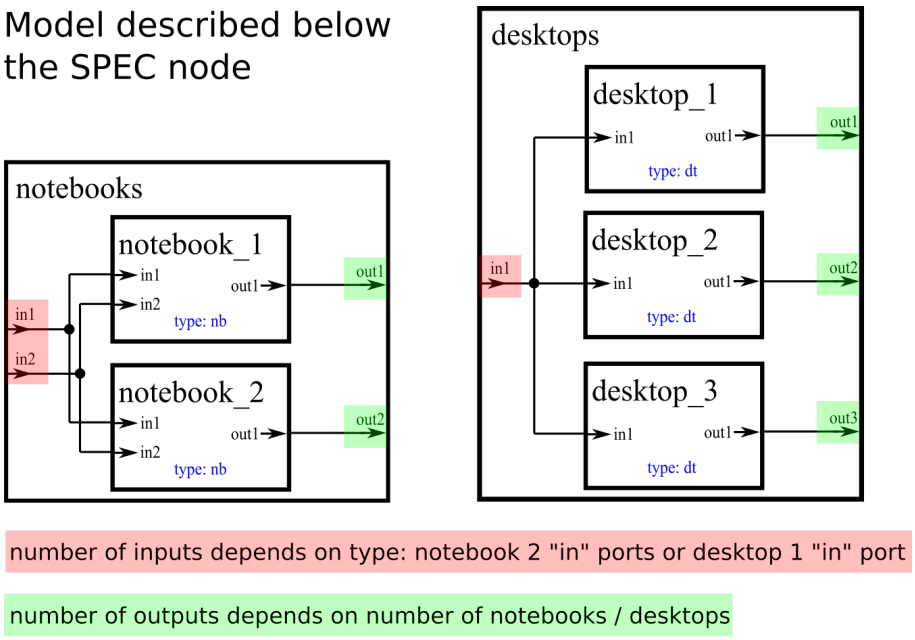


Figure 4.37: Using PATH in couplings - resulting model.

This number must be part of $\mathbb{N}$ as stated in the semantic conditions. In the first pruning step the nodes *partialPlant_i* are generated, the multi-aspect node is changed to aspect node, and the underscore attributes are assigned with the number of the created child.

Each *partialPlant* has one type out of 'power station', 'chemical productions', and 'waste treatment'. The type of a *partialPlant* is set in the specialization node *partialPlantSPEC*. This specialization uses the SESfcn *ppTypesFun()* to select the child nodes of the specialization *powerStation*, *chemicalProductions* or *wasteTreatment*. The SESfcn *ppTypesFun()* is called with the parameters PATH and *PartialPlantTypes*. The SESvar *PartialPlantTypes* is a list of 'ps', 'cp', and 'wt' elements and it needs to have as many elements as the value in *NumPartialPlants*. In the second pruning step the SESfcn *ppTypesFun(PATH, PartialPlantTypes)* reads the underscore attribute *_partialPlant* in the *partialPlant_i* node in the current path and returns the corresponding element in *PartialPlantTypes*. Thus the selection of one child depends on the path in which the specrule in *partialPlantSPEC* is called. In case the returned variable is 'ps' the *powerStation* node is selected, if the returned variable is 'cp' the *chemicalProductions* node is selected, whereas in case the returned variable is 'wt' the *wasteTreatment* node is selected. Not selected children of the specialization *partialPlantSPEC* are cut away. Finally *partialPlantSPEC* is resolved and the selected child is united with the respective parent.

The *powerStation* as well as the *wasteTreatment* nodes are decomposed in pipes,

SESvar = {NumPartialPlants, PartialPlantTypes, NumChemicalProductions,
ChemicalProductionTypes, NumCtrlSysChemicalProductions}

SemanticCondition = {NumPartialPlants ∈ ℕ,
PartialPlantTypes=[e_1, e_2, ... ,e_n] ∧ e_i ∈ {'ps', 'cp', 'wt'} ∧ n==NumPartialPlants
NumChemicalProductions=[e_1, e_2, ... ,e_n] ∧ e_i ∈ ℕ ∧ n==|{x | x ∈ (PartialPlantTypes=='cp')}|
ChemicalProductionTypes=[e_1, e_2, ... ,e_n] ∧ e_i=[a_1, a_2, ... ,a_m] ∧ n==|NumChemicalProductions|
∧ a_i ∈ {'ac', 'ba'} ∧ m==NumChemicalProductions(e_i)
NumCtrlSysChemicalProductions=[e_1, e_2, ... ,e_n] ∧ e_i=[a_1, a_2, ... ,a_m] ∧ n==|NumChemicalProductions|
∧ a_i ∈ ℕ ∧ m==NumChemicalProductions(e_i) }

SESfcn: ppTypesFun(pathToRoot, ppTypes)
#find partial plant from pathToRoot
#map to ppTypes
#return typePartialPlant

cpNumFun(pathToRoot, ppTypes, cpNum)
#find partial plant from pathToRoot
#map to ppTypes
#if ppType is 'cp'
get number of chemical productions
return numberChemicalProductions

cpTypesFun(pathToRoot, ppTypes, cpTypes)
#find partial plant from pathToRoot
#map to ppTypes
#if ppType is 'cp'
get the number of the chemical production from pathToRoot
map to cpTypes
return chemicalProductionType

cpCtrlFun(pathToRoot, ppTypes, cpCtrlsNum)
#find partial plant from pathToRoot
#map to ppTypes
#if ppType is 'cp'
get the number of the chemical production from pathToRoot
map to cpCtrlsNum
return numberControls

numRep1 = NumPartialPlants
specrule1 = {ppTypesFun(PATH, PartialPlantTypes) == "ps"➔powerStation
ppTypesFun(PATH, PartialPlantTypes) == "cp"➔chemicalProductions
ppTypesFun(PATH, PartialPlantTypes) == "wt"➔wasteTreatment }
numRep2 = cpNumFun(PATH, PartialPlantTypes, NumChemicalProductions)
specrule2 = {cpTypesFun(PATH, PartialPlantTypes, ChemicalProductionTypes) == "ac"➔acid
cpTypesFun(PATH, PartialPlantTypes, ChemicalProductionTypes) == "ba"➔base}
numRep3 = cpCtrlFun(PATH, PartialPlantTypes, NumCtrlSysChemicalProductions)

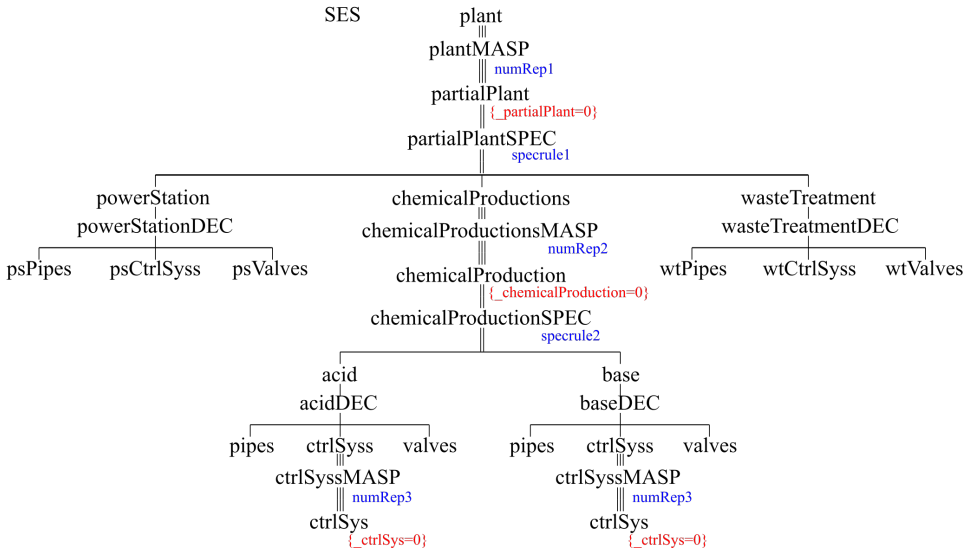


Figure 4.38: The SES of a chemical plant.

controlled systems, and valves. This is not specified further.

One partial plant of type *chemicalProductions* can have several *chemicalProductions*. This is specified in the multi-aspect node *chemicalProductionsMASP*. The number of *chemicalProductions* to generate is specified with an SESfcn *cpNumFun()*. The SESfcn *cpNumFun()* is called with the parameters PATH, *PartialPlantTypes*, and *NumChemicalProductions*. The SESvar *NumChemicalProductions* is a list with the number of *chemicalProductions* to generate in each *partialPlant*. The numbers in this list need to be of type ℕ and the list needs to have as many elements as there are *chemicalProductions* – so 'cp' elements in the SESvar *PartialPlantTypes*. In the third pruning step the SESfcn *cpNumFun(PATH, PartialPlantTypes, NumChemi-*

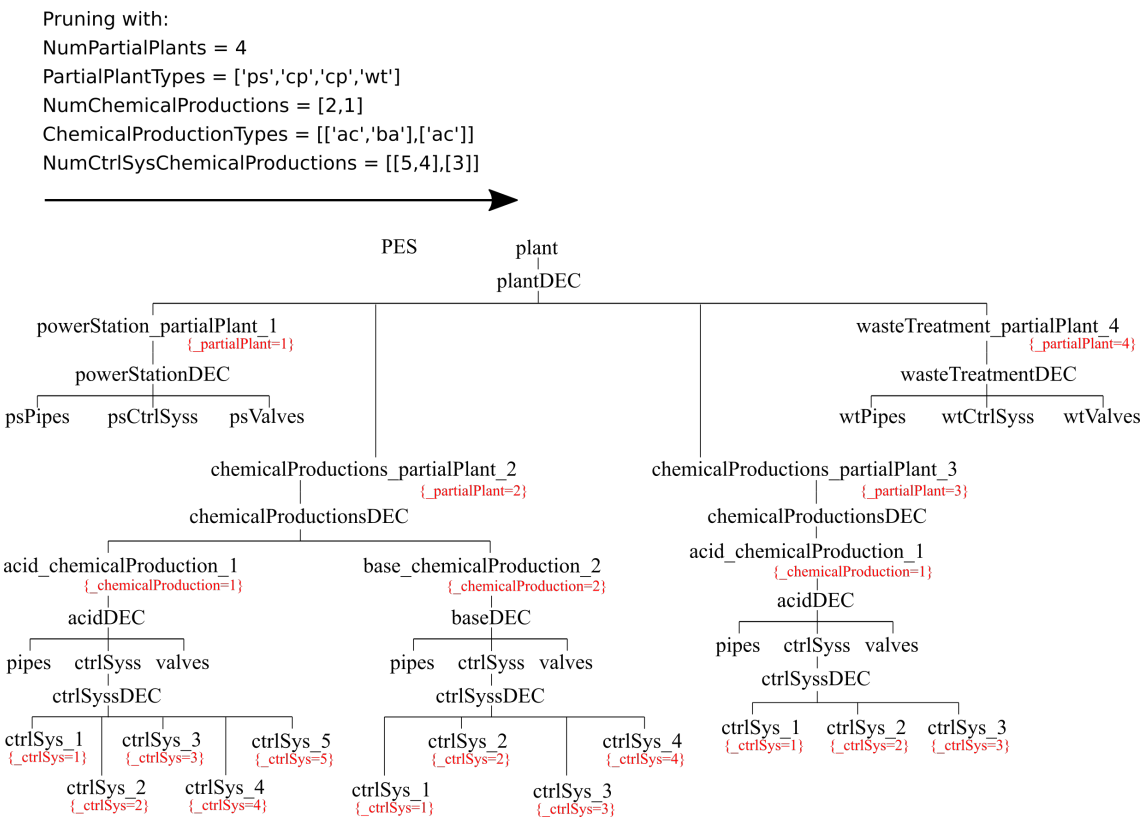


Figure 4.39: The PES of a chemical plant.

calProductions) reads the underscore attribute *_partialPlant* in the *partialPlant_i* node in the current path, determines the number of the partial plant of type *chemicalProductions* using *PartialPlantTypes*, and maps to the number in *NumChemicalProductions* for the current path. Since in *PartialPlantTypes* the value 'cp' can be on any position, the SESfcn needs to be more complex to map the index of each 'cp' in *PartialPlantTypes* to the correct index in *NumChemicalProductions*. For example the second 'cp' in *PartialPlantTypes* is on index 2. It needs to be mapped to index 1 in *NumChemicalProductions*, since indices start at 0. However, the second 'cp' in *PartialPlantTypes* can be on any index >1, so the mapping needs to be dynamical. The number in *NumChemicalProductions* for the current path is returned. Thus the number of *chemicalProductions* depends on the path in which the multi-aspect node *chemicalProductionsMASP* is called and thus on the *partialPlant*. The nodes *chemicalProduction_i* are generated, the multi-aspect node is changed to aspect node, and the underscore attributes are assigned with the number of the created child.

Each *chemicalProduction* has one type out of 'ac' and 'ba'. The type of a *chemicalProduction* is set in the specialization node *chemicalProductionSPEC*. This specialization uses the SESfcn *cpTypesFun()* to select the child nodes of the specializa-

tion *acid* or *base*. The SESfcn *cpTypesFun()* is called with the parameters PATH, *PartialPlantTypes*, and *ChemicalProductionTypes*. The SESvar *ChemicalProductionTypes* is a list of vectors. The number of list elements is equal to the number of *chemicalProductions* – so 'cp' elements in the SESvar *PartialPlantTypes*. The number of elements in each vector is equal to the number of entities in the corresponding number of chemical productions, defined by *NumChemicalProductions(i)*. Each element in the vector is of type 'ac' or 'ba'. In the fourth pruning step the SESfcn *cpTypesFun(PATH, PartialPlantTypes, ChemicalProductionTypes)* reads the underscore attribute *_partialPlant* in the *partialPlant_i* node in the current path, determines the number of the partial plant of type *chemicalProductions* using *PartialPlantTypes*, reads the underscore attribute *_chemicalProduction* in the *chemicalProduction_i* node in the current path, and maps with these information to the type in *ChemicalProductionTypes*. The challenges for mapping as discussed for pruning step three occur as well. The type in *ChemicalProductionTypes* for the current path is returned. Thus the selection of one child depends on the path in which the specrule in *chemicalProductionSPEC* is called. In case the returned variable is 'ac' the *acid* node is selected, if the returned variable is 'ba' the *base* node is selected. Not selected children of the specialization *chemicalProductionSPEC* are cut away. Finally *chemicalProductionSPEC* is resolved and the selected child is united with the respective parent.

The *acid* as well as the *base* nodes are decomposed in *pipes*, controlled systems *ctrlSyss*, and *valves*. While *pipes* and *valves*, are not specified further, the controlled systems are specified in detail. A *chemicalProduction* of type *acid* or *base* can have several *ctrlSyss*. This is specified in the multi-aspect node *ctrlSyssMASP*. The number of *ctrlSys* to generate is specified with an SESfcn *cpCtrlFun()*. The SESfcn *cpNumFun()* is called with the parameters PATH, *PartialPlantTypes*, and *NumCtrlSysChemicalProductions*. The SESvar *NumCtrlSysChemicalProductions* is a list of vectors. The number of list elements is equal to the number of *chemicalProductions* – so 'cp' elements in the SESvar *PartialPlantTypes*. The number of elements in each vector is equal to the number of entities in the corresponding number of chemical productions, defined by *NumCtrlSysChemicalProductions(i)*. Each element in the vector is of type $\mathbb{N}$. In the fifth pruning step the SESfcn *cpCtrlFun(PATH, PartialPlantTypes, NumCtrlSysChemicalProductions)* reads the underscore attribute *_partialPlant* in the *partialPlant_i* node in the current path, determines the number of the partial plant of type *chemicalProductions* using *PartialPlantTypes*, reads the underscore attribute *_chemicalProduction* in the *chemicalProduction_i* node in the current path, and maps with these information to the number in *NumCtrlSysChem-*

icalProductions. The challenges for mapping as discussed for pruning step three occur as well. The number in *CtrlSysChemicalProductions* for the current path is returned. Thus the number of *ctrlSys* depends on the path in which the multi-aspect node *ctrlSyssMASP* is called. The nodes *ctrlSys_i* are generated, the multi-aspect node is changed to aspect node, and the underscore attributes are assigned with the number of the created child.
The node *ctrlSyss* with its child *ctrlSyssMASP* are in the paths of *acid* as well as *base*. According to the axiom of uniformity nodes with the same name need to have the same variables and isomorphic subtrees. In the SES this axiom is fulfilled. Like in Section 4.16.6 this axiom is relaxed in the PES.

In example 4.14 is discussed how different control structures can be examined. This example SES can be merged to *ctrlSys*.

5 Related Work

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 SES by Zeigler [Zei84]. This approach was connected with models defined in an MB. A first abstract approach for dynamic variability modeling using an SES was published in [ZP89]. Several abstract transformation methods for deriving a particular system configuration and for generating an executable simulation model were added [ZKP00]. The entire approach was continuously further developed by Zeigler and many other researchers, such as in [RZ93], [ZH07] and [ZSDS12]. Based on this first idea, a prototype of a full modeling and simulation infrastructure has been developed and implemented within MATLAB/Simulink [PSZD16]. In this context some extensions of the SES were developed, e.g. in [PSZD16] and [SDP16] the SES theory was extended with a procedural knowledge representation. For development of the pruning method, a set of design patterns for the SES were defined in [DFPD18]. A comprehensive example on how the SES can be used, is demonstrated in [FPDD17]. Next to building executable models for several popular simulators, there is an interest to support DEVSimPy for building DEVS systems as well. The Watershed Example was created following a discussion started on the paper [SCZ16]. In [FPD+19] it is presented in the context of the introduced infrastructure. The feedback control system example is discussed in [FPDH19b], [FPDH19a], and [FPD20]. In [FPDZ20] automatic pruning of hierarchies of multi-aspect nodes is presented.

Bibliography

[DFPD18] Christina Deatcu, Hendrik Folkerts, Thorsten Pawletta, and Umut Durak. Design patterns for variability modeling using ses ontology. In *Proceedings of the Model-driven Approaches for Simulation Engineering Symposium*, Mod4Sim '18, pages 3:1–3:12, San Diego, CA, USA, 2018. Society for Computer Simulation International. URL: http://dl.acm.org/citation.cfm?id=3213214.3213217.

[FPD+19] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, Jean-Francois Santucci, and Laurent Capocchi. An integrated modeling, simulation and experimentation environment in python based on ses/mb and devs. In *Proceedings of the 2019 Summer Simulation Conference*, SummerSim '19, San Diego, CA, USA, 2019. Society for Computer Simulation International.

[FPD20] Hendrik Folkerts, Thorsten Pawletta, and Christina Deatcu. Model generation for multiple simulators using ses/mb and fmi. In *Proceedings ASIM SST 2020*. ARGESIM Pub. Vienna/Austria, 2020. doi:10.11128/arep.59.a59003.

[FPDD17] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, and Umut Durak. Variability modeling for engineering applications. *SNE Simulation Notes Europe*, 27(4):167–176, December 2017. URL: https://elib.dlr.de/119222/.

[FPDH19a] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, and Sven Hartmann. Python-based eses/mb framework: Model specification and automatic model generation for multiple simulators. *SNE Simulation Notes Europe*, 29:207–215, 12 2019. doi:10.11128/sne.29.tn.10497.

[FPDH19b] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, and Sven Hartmann. A python framework for model specification and automatic

model generation for multiple simulators. In *Proceedings ASIM Workshop STS/GMMS 2019*, pages 69–75. ARGESIM Pub. Vienna/Austria, 2019.

[FPDZ20] Hendrik Folkerts, Thorsten Pawletta, Christina Deatcu, and Bernard P Zeigler. Automated, reactive pruning of system entity structures for simulation engineering. SCS SpringSim'20, Fairfax, VA, USA, 2020. Society for Computer Simulation International.

[PM90] Franz Pichler and Roberto Moreno-Díaz, editors. *Computer Aided Systems Theory - EUROCAST'89, A Selection of Papers from the International Workshop EUROCAST'89, Las Palmas, Spain, February 26 - March 4, 1989, Proceedings*, volume 410 of *Lecture Notes in Computer Science*. Springer, 1990. URL: https://doi.org/10.1007/3-540-52215-8, doi:10.1007/3-540-52215-8.

[PSZD16] Thorsten Pawletta, Artur Schmidt, Bernard P. Zeigler, and Umut Durak. Extended variability modeling using system entity structure ontology within matlab/simulink. In *Proceedings of the 49th Annual Simulation Symposium*, ANSS '16, pages 22:1–22:8, San Diego, CA, USA, 2016. Society for Computer Simulation International. URL: http://dl.acm.org/citation.cfm?id=2962374.2962396.

[RZ93] Jerzy W. Rozenblit and Bernard P. Zeigler. Representing and constructing system specifications using the system entity structure concepts. In *Proceedings of the 25th Conference on Winter Simulation*, WSC '93, pages 604–611, New York, NY, USA, 1993. ACM. URL: http://doi.acm.org/10.1145/256563.256742, doi:10.1145/256563.256742.

[SCZ16] Jean-François Santucci, Laurent Capocchi, and Bernard P Zeigler. System entity structure extension to integrate abstraction hierarchies and time granularity into devs modeling and simulation. *Simulation*, 92(8):747–769, August 2016. URL: https://doi.org/10.1177/0037549716657168, doi:10.1177/0037549716657168.

[SDP16] Artur Schmidt, Umut Durak, and Thorsten Pawletta. Model-based testing methodology using system entity structures for matlab/simulink models. *SIMULATION*, 92(8):729–746, 2016. URL: https://doi.org/10.1177/0037549716656791, arXiv:https://doi.org/10.1177/0037549716656791, doi:10.1177/0037549716656791.

[Zei84] Bernard P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press Professional, Inc., San Diego, CA, USA, 1984.

[ZH07] Bernard P. Zeigler and Phillip E. Hammonds. *Modeling & Simulation-Based Data Engineering: Introducing Pragmatics into Ontologies for Net-Centric Information Exchange*. Academic Press, Inc., Orlando, FL, USA, 2007.

[ZKP00] B.P. Zeigler, T.G. Kim, and H. Praehofer. *Theory of Modeling and Simulation*. Elsevier Science, 2000. URL: https://books.google.de/books?id=REzmYOQmHuQC.

[ZP89] Bernard P. Zeigler and Herbert Praehofer. Systems theory challenges in the simulation of variable structure and intelligent systems. In Pichler and Moreno-Díaz [PM90], pages 41–51. URL: https://doi.org/10.1007/3-540-52215-8_4, doi:10.1007/3-540-52215-8_4.

[ZSDS12] Bernard P. Zeigler, Hessam S. Sarjoughian, Raphal Duboz, and Jean-Christophe Souli. *Guide to Modeling and Simulation of Systems of Systems*. Springer Publishing Company, Incorporated, 2012.

List of Figures

List of Tables

List of Listings

Acronyms

CEA Computational Engineering und Automation.

EC experiment control.

eSES/MB extended SES/MB.

EU execution unit.

FMI Functional Mock-up Interface.

FMU Functional Mock-up Unit.

FPES Flattened Pruned Entity Structure.

MB Model Base.

numRep number of replications.

PES Pruned Entity Structure.

SES System Entity Structure.

SESfcn SES function.

SESvar SES variable.

SM simulation model.

SME Simulation Model Executable.

SMR Simulation Model Representation.

uid unique identifier.

A Example for the JSON Structure

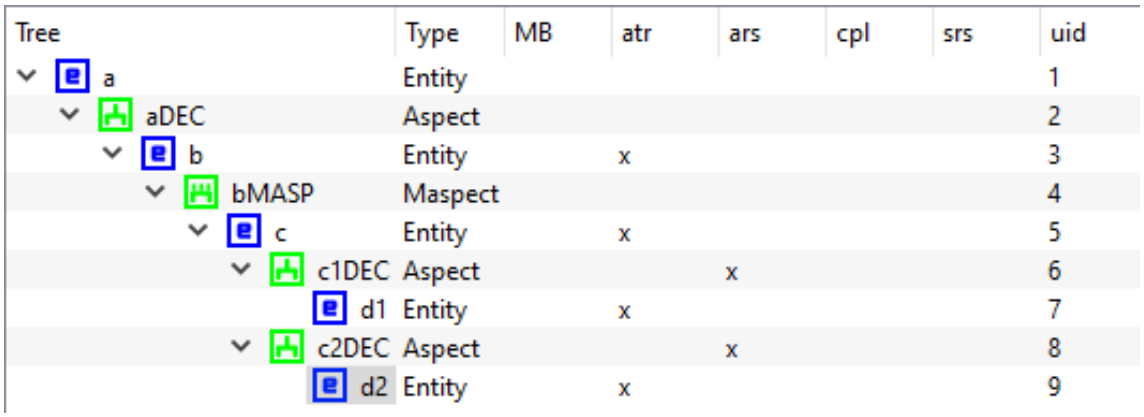
An example of a JSON structure saved by SESToPy is shown in Listing A.1. Its representation in SESToPy shows Figure A.1.

```
1  SES JSON Version 1 - DO NOT EDIT THIS FILE MANUALLY!
2  Generated by SESToPy (University of Applied Sciences Wismar, Research Group
       Computational Engineering and Automation); Contact: Prof. Dr.-Ing.
       Thorsten Pawletta, thorsten.pawletta@hs-wismar.de; developed by Hendrik
       Martin Folkerts originally using Python 3.4.1 and PyQt 5.5
3  System Entity Structure tree with settings generated by SESToPy (University
       of Applied Sciences Wismar, Research Group Computational Engineering and
        Automation)
4  Created: 2018-08-24 10:19:28
5  [["1", "Entity Node", "a", "0", "#000000", "False", [], [], [], "1", [], "1",
        "0"], ["2", "Aspect Node", "aDEC", "1", "#000000", "False", [], [], [],
        "1", [], "1", "1"], ["3", "Entity Node", "b", "2", "#000000", "False",
       [["attr1", "1", "", ""]], [], [], "1", [], "1", "2"], ["4", "Maspect
       Node", "bMASP", "3", "#000000", "False", [], [], [], "A", [], "1", "3"],
        ["5", "Entity Node", "c", "4", "#000000", "False", [["attr2", "2", "",
       ""]], [], [], "1", [], "1", "4"], ["6", "Aspect Node", "c1DEC", "5", "
       #000000", "False", [], [["c1DEC", "6", "A==\"abc\"", "F", ""]], [], "1",
        [], "1", "5"], ["7", "Entity Node", "d1", "6", "#000000", "False", [["
       attr3", "3", "", ""]], [], [], "1", [], "1", "6"], ["8", "Aspect Node",
       "c2DEC", "5", "#000000", "False", [], [["c2DEC", "8", "A==2", "F", ""]],
        [], "1", [], "1", "5"], ["9", "Entity Node", "d2", "8", "#000000", "
       False", [["attr4", "4", "", ""]], [], [], "1", [], "1", "6"]]
6  [["ses", "This is a test comment."]]
7  [["A", "1", ""]]
8  [["A in [1,2,3]", "T"]]
9  []
10 [["cplExample", "def cplExample(var):\n    if var == 1:\n        retwert =
       [[\"c\", \"soport\", \"d_1\", \"siport\", \"\"]]\n    if var == 2:\n
              retwert = [[\"c\", \"soport\", \"d_1\", \"siport\", \"\"], [\"c\"
       , \"soport\", \"d_2\", \"siport\", \"\"]]\n    return retwert"]]
```

Listing A.1: JSON structure for saving an SES.

- Line 5: The nodes with their SESToPy internal and node specific properties are encoded.
- Line 6: The type (SES / incomplete pruned PES / PES and FPES) is saved together with a comment on the SES.
- Line 7: SESvars are saved.
- Line 8: Semantic conditions are saved here.

- Line 9: This line is for saving selection constraints. There are no selection constraints in this example.
- Line 10: An SESfcn (Python-Code) is encoded in this line.



Tree	Type	MB	atr	ars	cpl	srs	uid
a	Entity						1
aDEC	Aspect						2
b	Entity		x				3
bMASP	Maspect						4
c	Entity		x				5
c1DEC	Aspect			x			6
d1	Entity		x				7
c2DEC	Aspect			x			8
d2	Entity		x				9

Figure A.1: The representation of the JSON code in Listing A.1 in SESToPy.

B Example for the XML Structure

An example of an XML structure exported by SESToPy is shown in Listing B.1. Its representation in SESToPy shows Figure B.1.

```
1  <?xml version="1.0" ?>
2  <SESwithSettings name="SES" xmlns:vc="http://www.w3.org/2007/XMLSchema-
       versioning" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
3  <globals>
4  <sespes comment="" value="ses"/>
5  <sesvars/>
6  <semcons/>
7  <selcons/>
8  <sesfcns/>
9  </globals>
10 <tree>
11 <node comment="" name="a" type="entity">
12 <node comment="" name="aDEC" type="aspect">
13 <prio value="1"/>
14 <node comment="" name="b" type="entity">
15 <node comment="" name="bMASP" type="multiaspect">
16 <aspr comment="" condition="" result=""/>
17 <prio value="1"/>
18 <numr value="1"/>
19 </node>
20 </node>
21 </node>
22 </node>
23 </tree>
24 </SESwithSettings>
```

Listing B.1: XML structure for exporting an SES.

- Line 2: Tag for the structure and further xml information.
- Line 24: Closing the structure.
- Line 3: All global settings are placed in a tag globals.
- Line 5: Each sesvar gets an own tag under the sesvars tag.
- Line 6-8: The same applies for other global attributes.
- Line 9: Closing the globals tag.
- Line 10: Tag indicating the start of the tree.
- Line 23: Tag closing the tree.
- Line 11: First node.

- Line 22: Close first node.
- Line 12: Second node.
- Line 13: Attribute of the second node.
- ...

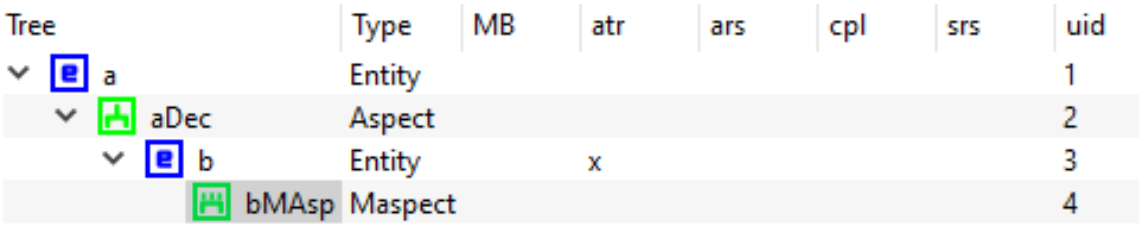


Figure B.1: The representation of the XML code in Listing B.1 in SESToPy.

C Pruning Activity Diagram

The pruning method is shown in an Activity Diagram (see Unified Modeling Language for details on the Activity Diagram). It is best practice to print out the following six pages and lay the pages as shown in the overview Figure C.1.

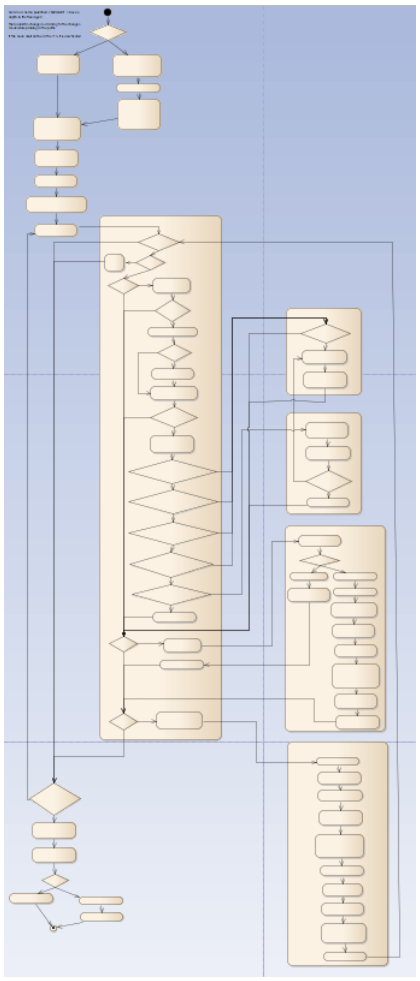


Figure C.1: The complete pruning algorithm as Activity Diagram.

On the following six pages the pruning algorithm is given.

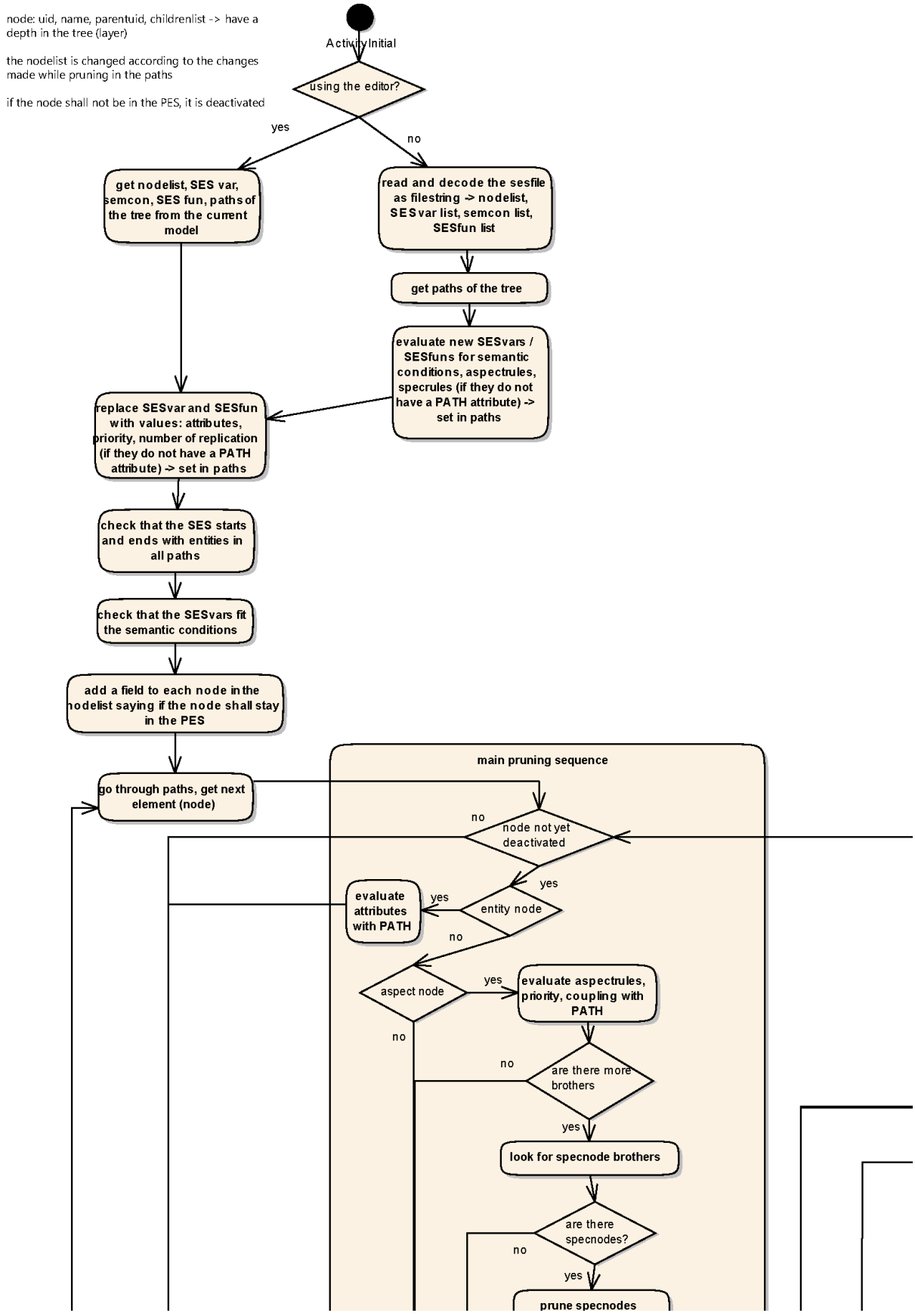


Figure C.2: The complete pruning algorithm as Activity Diagram - page 1.

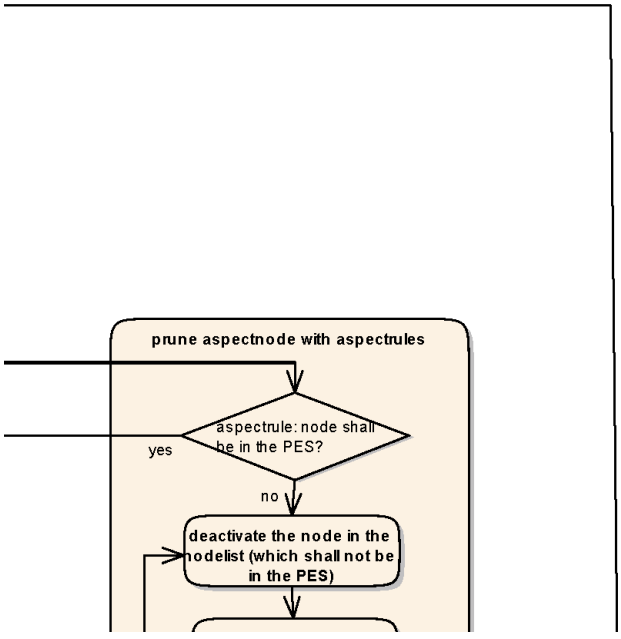


Figure C.3: The complete pruning algorithm as Activity Diagram - page 2.

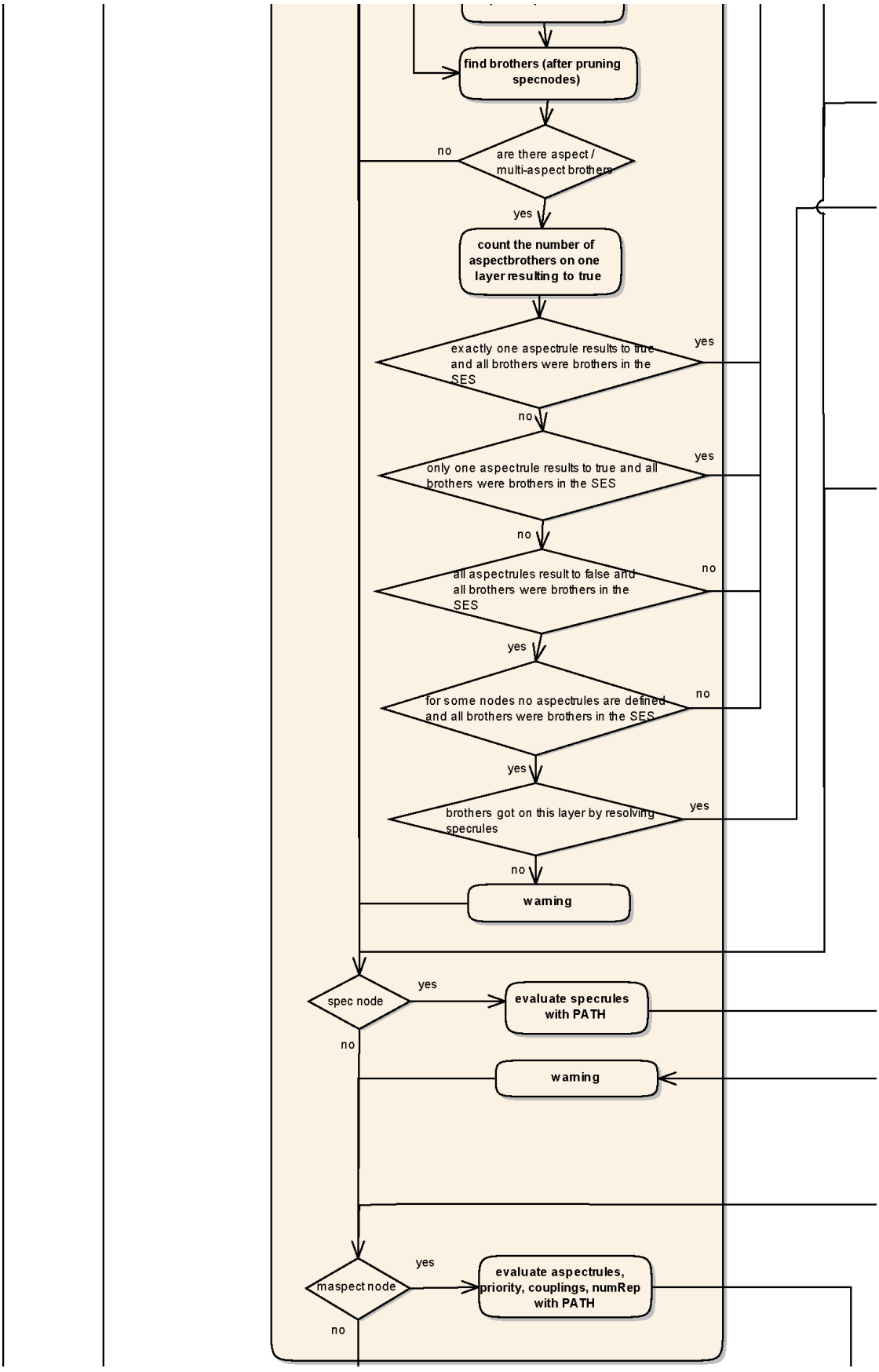


Figure C.4: The complete pruning algorithm as Activity Diagram - page 3.

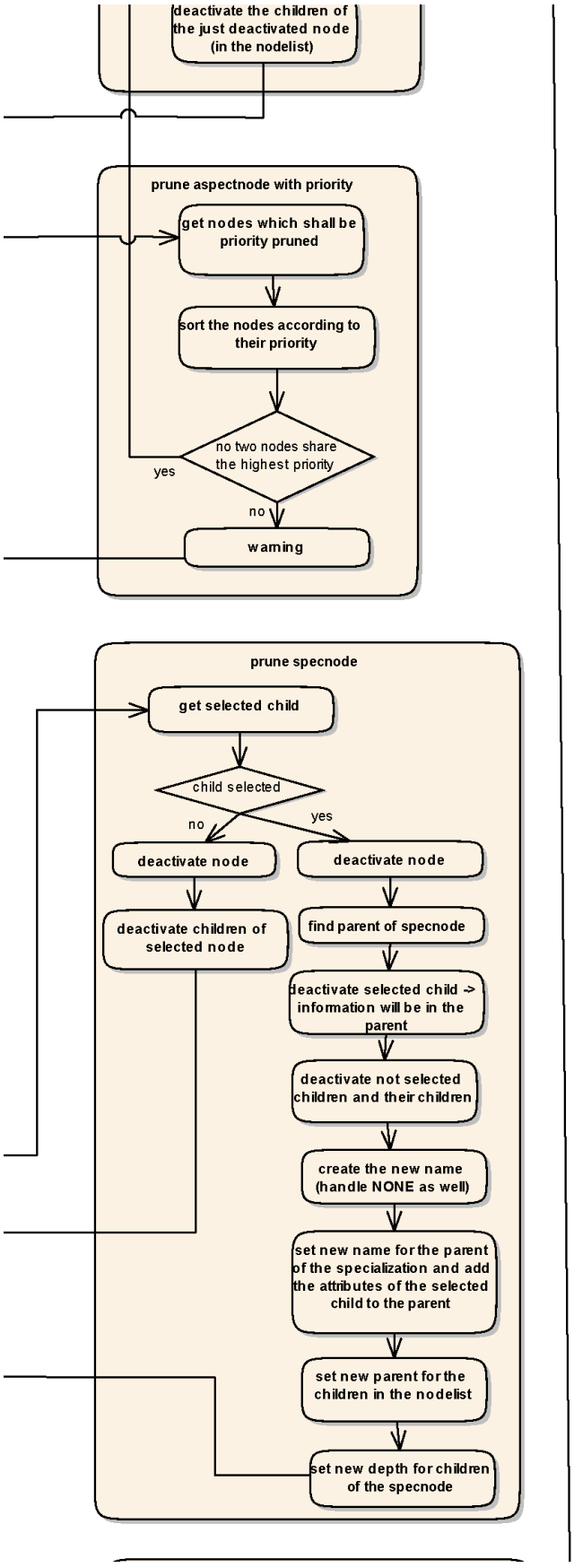


Figure C.5: The complete pruning algorithm as Activity Diagram - page 4.

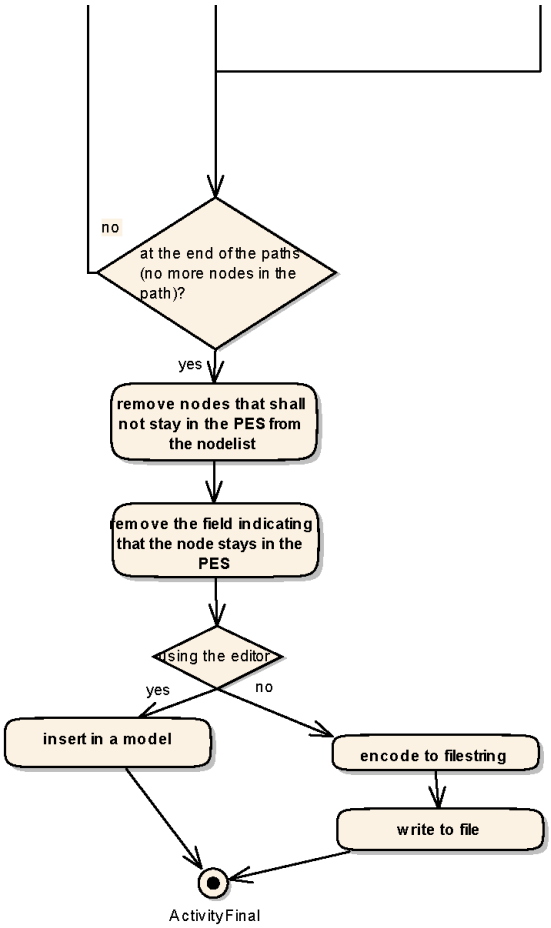


Figure C.6: The complete pruning algorithm as Activity Diagram - page 5.

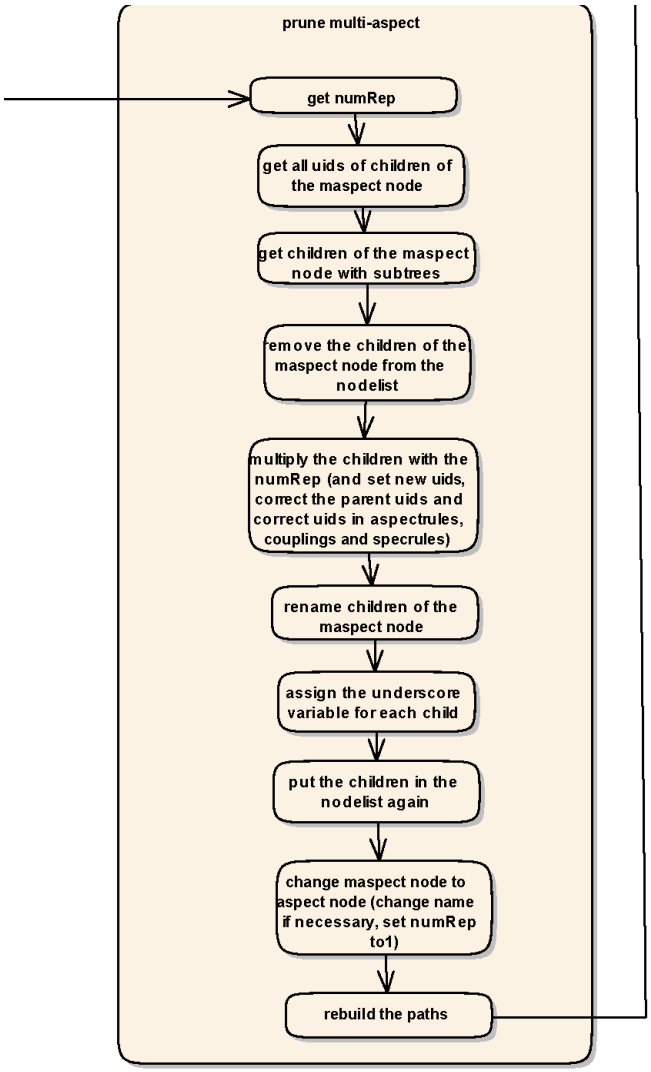


Figure C.7: The complete pruning algorithm as Activity Diagram - page 6.

D Python 3 Code of the cplfcn1 of the Watershed Example

This SESfcn describes the coupling relation in *Mountain1DEC*. This is referred to as cplg3.

```
def cplfcn1(children,parent,NUM):
    #parent is Mountain1
    #children[0] is Estimator
    #children[1] is Areas
    #children[2] is Basin
    #children[3] is UAM2

    cplg = []

    #fixed couplings
    cplg.append([parent,"in1 / SPR",children[0],"in1 / SPR",""])
    cplg.append([parent,"in2 / SPR",children[0],"in2 / SPR",""])
    cplg.append([parent,"in3 / SPR",children[0],"in3 / SPR",""])
    cplg.append([parent,"in4 / SPR",children[2],"in1 / SPR",""])
    cplg.append([children[0],"out1 / SPR",children[1],"in1 / SPR",""])
    cplg.append([children[1],"out1 / SPR",children[2],"in2 / SPR",""])
    cplg.append([children[2],"out1 / SPR",children[3],"in1 / SPR",""])
    cplg.append([children[3],"out / SPR",parent,"out / SPR",""])

    #variable couplings
    if NUM==2:
        cplg.append([children[0],"out2 / SPR",children[1],"in2 / SPR",""])
        cplg.append([children[1],"out2 / SPR",children[3],"in2 / SPR",""])

    #return
    return cplg
```

Listing D.1: Python 3 code of the cplfcn1 of the Watershed Example.

E Python 3 Code of the cplfcn2 of the Watershed Example

This SESfcn describes the coupling relation in *AreasMASP*. This is referred to as cplg4.

```
def cplfcn2(children, parent, NUM):
    #parent is Areas
    #children[0] is Area_1
    #children[1] is Area_2

    cplg = []

    #variable couplings
    for i in range(0,NUM):
        cplg.append([parent,"in"+str(i+1)+" / SPR",children[i],"in / SPR",""
            ])
        cplg.append([children[i],"out / SPR",parent,"out"+str(i+1)+" / SPR","
            "])

    #return
    return cplg
```

Listing E.1: Python 3 code of the cplfcn2 of the Watershed Example.

F Pruning the Watershed Example

In the Figures in section 4.13 one system variant, achieved by setting the SESvars as shown before pruning and flattening, was discussed for the Watershed SES. However, the SES describes more system variants, which can be derived with different settings for the SESvars. All system variants will be discussed in the following. Therefore layers of the SES are described and the models theoretically built of the PES are depicted. This is for clearance, no FPES are shown.
The *root* part of the SES describing the composition of the main structure is presented in Figure F.1.

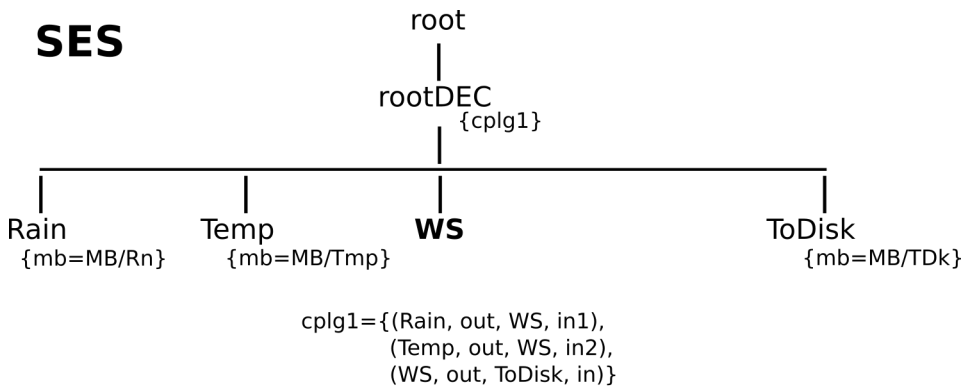


Figure F.1: Watershed example - the *root* part of the SES

When deriving the PES, the PES looks exactly like the SES. The PES and the model are shown in Figure F.2. Please keep in mind, that this step cannot be achieved during pruning, since the node *WS* is no leaf node.

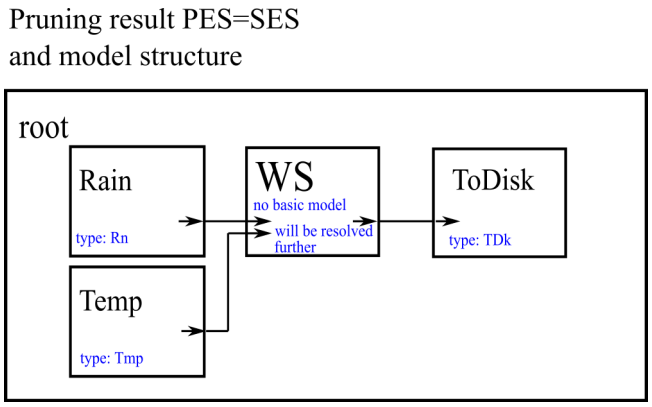


Figure F.2: Watershed example - root: the PES and the model

The next layers describe the type and, if the type is *WS1*, the composition of the node *WS*. It is depicted in Figure F.3. Remember, that the node *EWS* is seen as basic model in this context.

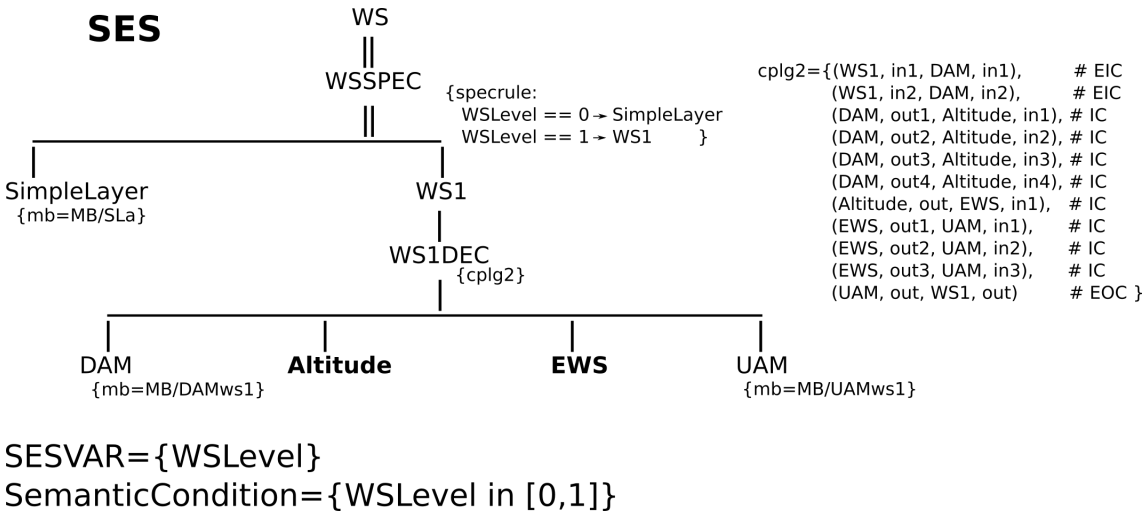


Figure F.3: Watershed example - the *WS* part of the SES

Deriving the PES, there are two structure variants for the PES leading to two possible system structures. They are presented in Figures F.4 and F.5. Please keep in mind, that this step cannot be achieved during pruning, since the node *Altitude* is no leaf node.

Depending on the value of the SESvar WSLevel, *WS* is either resolved to *SimpleLayer_WS* or to *WS1_WS*, which has a subtree. That means, *WS* is either of the type *SimpleLayer* or of the type *WS1*. *WS1* is composed of four entity nodes. The node *Altitude* is considered further.

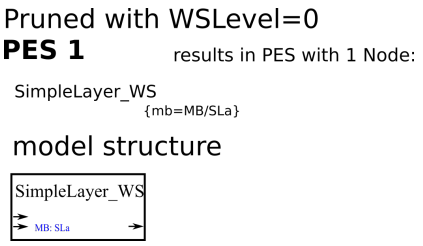


Figure F.4: Watershed example - WS: first possible PES and the model

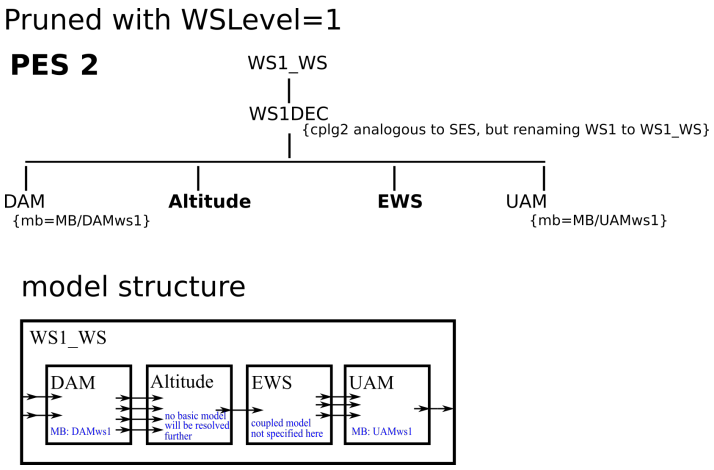
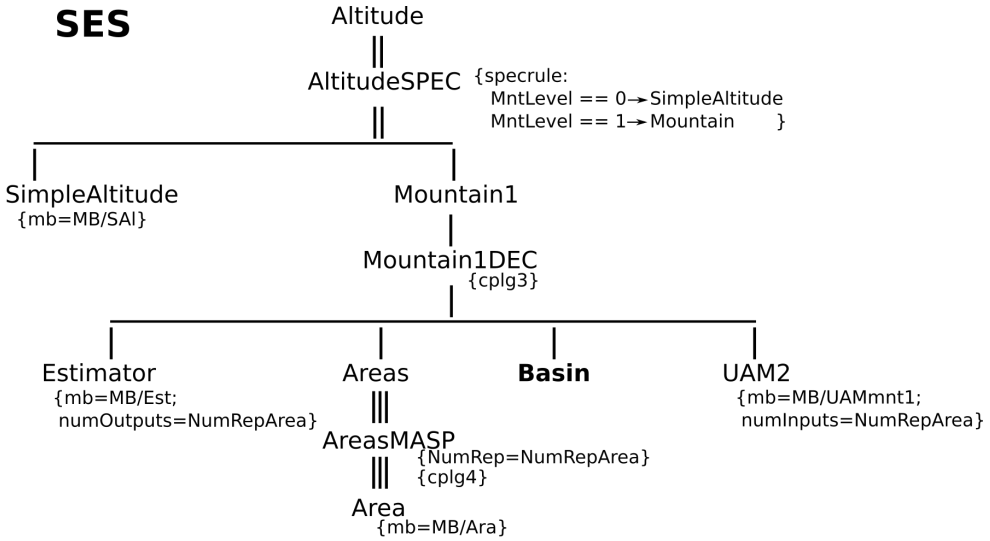


Figure F.5: Watershed example - WS: second possible PES and the model

The layer of the SES describing the altitude is given in Figure F.6. The type and, if the type is *Mountain1*, the composition of the node *Altitude* are described. The couplings are given as SESfcns, since the couplings depend on the number of areas created.



Couplings are set via coupling functions:

```
cplg3=cplfcn1(CHILDREN,PARENT,NumRepArea)

cplfcn1(children,parent,NUM)

    # parent is Mountain1,
    # children(1) is Estimator,
    # children(2) is Areas,
    # children(3) is Basin,
    # children(4) is UAM2.

    # fixed couplings
    cplg(1)=(parent,in1,children(1),in1)          # EIC
    cplg(2)=(parent,in2,children(1),in2)          # EIC
    cplg(3)=(parent,in3,children(1),in3)          # EIC
    cplg(4)=(parent,in4,children(3),in1)          # EIC
    cplg(5)=(children(1),out1,children(2),in1)  # IC
    cplg(6)=(children(2),out1,children(3),in2)  # IC
    cplg(7)=(children(3),out1,children(4),in1)  # IC
    cplg(8)=(children(4),out,parent,out)         # EOC

    # variable couplings
    if NUM==2
      cplg(9)=(children(1),out2,children(2),in2)  # IC
      cplg(10)=(children(2),out2,children(4),in2)# IC
    end

return(cplg)
```

variable NUMREP refers to the number of replications of the node calling the function -> refers to SESvar NumRepArea

```
cplg4=cplfcn2(CHILDREN,PARENT,NUMREP)

cplfcn2(children,parent,NUM)

    # parent is Areas,
    # children(1) is Area_1,
    # children(2) is Area_2.

    k=1
    for i in range (1,NUM)
      cplg(k)=(parent,in(i),children(i),in)          # EIC
      cplg(k+1)=(children(i),out,parent,out(i))  # EOC
      k=k+2
    end

return(cplg)
```

SESVAR={MntLevel, NumRepArea}
SemanticCondition={MntLevel in [0,1]; NumRepArea in [1,2]}

Figure F.6: Watershed example - the *Altitude* part of the SES

Deriving the PES, there are three structure variants for the PES leading to three possible system structures. They are presented in Figures F.7, F.8, and F.9. Please keep in mind, that this step cannot be achieved during pruning, since the node *Basin* is no leaf node.

Depending on the value of the SESvar MntLevel, *Altitude* is either resolved to *SimpleAltitude_Altitude* or to *Mountain1_Altitude*, which has a subtree. That means, *Altitude* is either of the type *SimpleAltitude* or of the type *Mountain1*. *Mountain1* is composed of four entity nodes. The node *Areas* can be composed of one or two nodes *Area* depending on the value of the SESvar NumRepArea (can have the values 1 or 2). Therefore there are three possible PES. Finally, the node *Basin* is considered further.

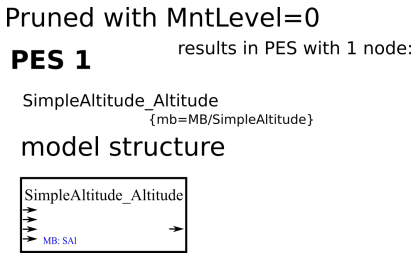


Figure F.7: Watershed example - Altitude: first possible PES and the model

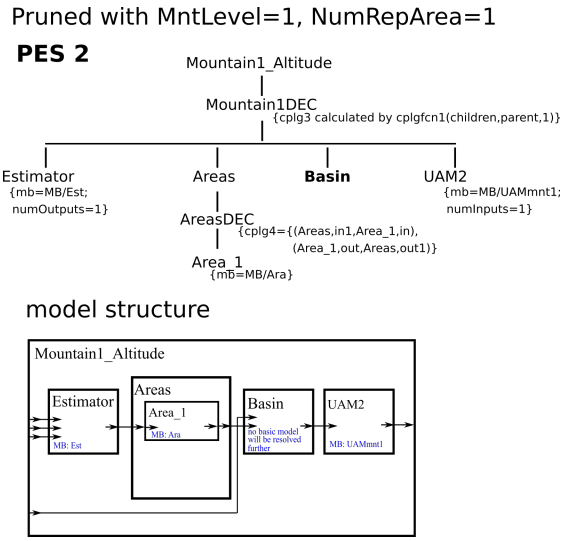


Figure F.8: Watershed example - Altitude: second possible PES and the model

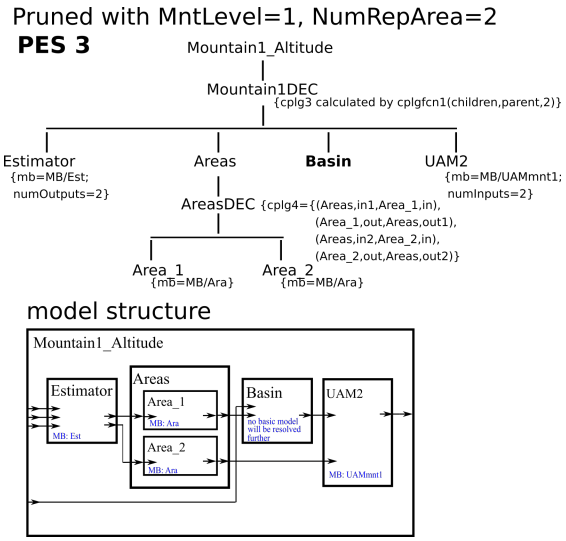


Figure F.9: Watershed example - Altitude: third possible PES and the model

The layer of the SES describing the basin is given in Figure F.10. The type and, if the type is *Basin1*, the composition of the node *Basin* are described.

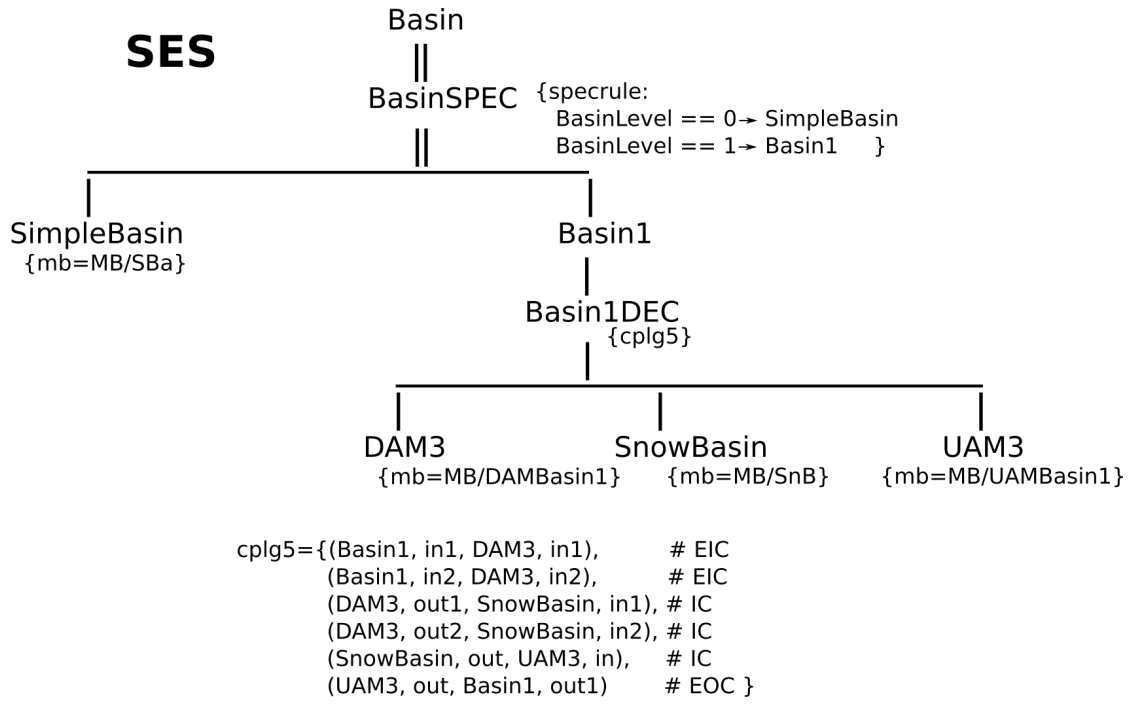


Figure F.10: Watershed example - the *Basin* part of the SES

Deriving the PES, there are two structure variants for the PES leading to two possible system structures. They are presented in Figures F.11 and F.12.
Depending on the value of the SESvar BasinLevel, *Basin* is either resolved to *SimpleBasin_Basin* or to *Basin1_Basin*, which has a subtree. That means, *Basin* is either of the type *SimpleBasin* or of the type *Basin1*. *Basin1* is composed of three entity nodes.

Pruned with BasinLevel=0

PES 1 results in PES with 1 node:

SimpleBasin_Basin
{mb=MB/SimpleBasin}

model structure

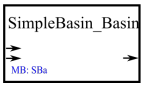
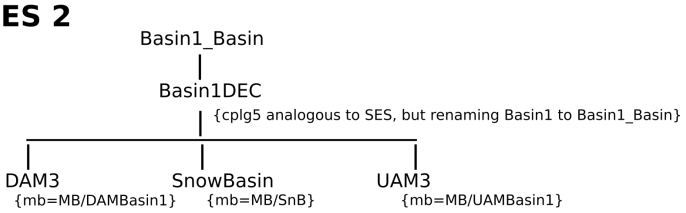


Figure F.11: Watershed example - Basin: first possible PES and the model

Pruned with BasinLevel=1

PES 2




```
cplg5={(Basin1_Basin, in1, DAM3, in1),    # EIC
       (Basin1_Basin, in2, DAM3, in2),    # EIC
       (DAM3, out1, SnowBasin, in1),      # IC
       (DAM3, out2, SnowBasin, in2),      # IC
       (SnowBasin, out, UAM3, in),        # IC
       (UAM3, out, Basin1_Basin, out1)  # EOC }
```

model structure

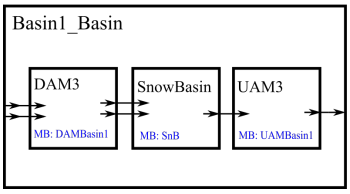


Figure F.12: Watershed example - Basin: second possible PES and the model

G Deriving the Couplings for the FPES of the Watershed Example

The couplings are adjusted for the FPES in Figure 4.19 as described in the following: Collect all couplings of the PES in Figure 4.15. Go through the collected couplings, until one is reached, that refers to a node not being in the FPES. The couplings have a source part and a sink part, which shall not be mixed. For more information on the algorithm, please look at section 3.7. All couplings of the PES are shown in the following Table G.1. The first column gives the coupling number, which is no part of the coupling. It is given to keep the overview.
Look at the source couplings. The colors show the steps taken next.

Table G.1: Couplings of the PES in Figure 4.15 given in Figure 4.16.

no.	source		sink	
1	Rain	out	WS1_WS	in1
2	Temp	out	WS1_WS	in2
3	WS1_WS	out	ToDisk	in
4	WS1_WS	in1	DAM	in1
5	WS1_WS	in2	DAM	in2
6	DAM	out1	Mountain1_Altitude	in1
7	DAM	out2	Mountain1_Altitude	in2
8	DAM	out3	Mountain1_Altitude	in3
9	DAM	out4	Mountain1_Altitude	in4
10	Mountain1_Altitude	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
14	UAM	out	WS1_WS	out
15	Mountain1_Altitude	in1	Estimator	in1
16	Mountain1_Altitude	in2	Estimator	in2
17	Mountain1_Altitude	in3	Estimator	in3
18	Mountain1_Altitude	in4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
22	UAM2	out	Mountain1_Altitude	out
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node Rain and Temp are in the FPES, skip them. Node WS1_WS is not in the FPES but the coupling WS1_WS with port “out“ (WS1_WS - out) can be found in coupling 14 on the sink side. Replace WS1_WS - out with UAM - out and delete coupling 14. The result is in Table G.2:

Table G.2: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the first step.

no.	source		sink	
1	Rain	out	WS1_WS	in1
2	Temp	out	WS1_WS	in2
3	UAM	out	ToDisk	in
4	WS1_WS	in1	DAM	in1
5	WS1_WS	in2	DAM	in2
6	DAM	out1	Mountain1_Altitude	in1
7	DAM	out2	Mountain1_Altitude	in2
8	DAM	out3	Mountain1_Altitude	in3
9	DAM	out4	Mountain1_Altitude	in4
10	Mountain1_Altitude	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	Mountain1_Altitude	in1	Estimator	in1
16	Mountain1_Altitude	in2	Estimator	in2
17	Mountain1_Altitude	in3	Estimator	in3
18	Mountain1_Altitude	in4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
22	UAM2	out	Mountain1_Altitude	out
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node WS1_WS is not in the FPES but the coupling WS1_WS - in1 is in coupling 1 on the sink side, replace with Rain - out and delete coupling 1. The result is in Table G.3:

Table G.3: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the second step.

no.	source		sink	
2	Temp	out	WS1_WS	in2
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	WS1_WS	in2	DAM	in2
6	DAM	out1	Mountain1_Altitude	in1
7	DAM	out2	Mountain1_Altitude	in2
8	DAM	out3	Mountain1_Altitude	in3
9	DAM	out4	Mountain1_Altitude	in4
10	Mountain1_Altitude	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	Mountain1_Altitude	in1	Estimator	in1
16	Mountain1_Altitude	in2	Estimator	in2
17	Mountain1_Altitude	in3	Estimator	in3
18	Mountain1_Altitude	in4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
22	UAM2	out	Mountain1_Altitude	out
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node WS1_WS is not in the FPES but the coupling WS1_WS - in2 is in coupling 2 on the sink side, replace with Temp - out and delete coupling 2. The result is in Table G.4:

Table G.4: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the third step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
6	DAM	out1	Mountain1_Altitude	in1
7	DAM	out2	Mountain1_Altitude	in2
8	DAM	out3	Mountain1_Altitude	in3
9	DAM	out4	Mountain1_Altitude	in4
10	Mountain1_Altitude	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	Mountain1_Altitude	in1	Estimator	in1
16	Mountain1_Altitude	in2	Estimator	in2
17	Mountain1_Altitude	in3	Estimator	in3
18	Mountain1_Altitude	in4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
22	UAM2	out	Mountain1_Altitude	out
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node DAM will be in the FPES, so do nothing there. Mountain1_Altitude is not in the FPES, but it is in coupling 22 on the sink side. Replace with UAM2 - out and delete coupling 22. The result is in Table G.5:

Table G.5: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the fourth step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
6	DAM	out1	Mountain1_Altitude	in1
7	DAM	out2	Mountain1_Altitude	in2
8	DAM	out3	Mountain1_Altitude	in3
9	DAM	out4	Mountain1_Altitude	in4
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	Mountain1_Altitude	in1	Estimator	in1
16	Mountain1_Altitude	in2	Estimator	in2
17	Mountain1_Altitude	in3	Estimator	in3
18	Mountain1_Altitude	in4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node EWS is in the FPES, so skip them. For Mountain1_Altitude with in ports 1 to 4 applies the same as before. They can be found on the sink side in coupling 6, 7, 8, and 9. Replace with DAM with out ports 1 to 4. Delete couplings 6, 7, 8, and 9. The result is in Table G.6:

Table G.6: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the fifth step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	DAM	out1	Estimator	in1
16	DAM	out2	Estimator	in2
17	DAM	out3	Estimator	in3
18	DAM	out4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Areas	out1	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
26	Area_1	out	Areas	out1
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node Estimator is in the FPES, so skip it. Node Areas is not in the FPES, but Areas - out1 can be found in coupling 26 on the sink side, replace and delete as before. The result is in Table G.7:

Table G.7: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the sixth step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	DAM	out1	Estimator	in1
16	DAM	out2	Estimator	in2
17	DAM	out3	Estimator	in3
18	DAM	out4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Area_1	out	Basin1_Basin	in2
21	Basin1_Basin	out1	UAM2	in1
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in
34	UAM3	out	Basin1_Basin	out1

Node Basin1_Basin is not in the FPES, but Basin1_Basin - out1 can be found in coupling 34 on the sink side, replace and delete as before. The result is in Table G.8:

Table G.8: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the seventh step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	DAM	out1	Estimator	in1
16	DAM	out2	Estimator	in2
17	DAM	out3	Estimator	in3
18	DAM	out4	Basin1_Basin	in1
19	Estimator	out1	Areas	in1
20	Area_1	out	Basin1_Basin	in2
21	UAM3	out	UAM2	in1
23	Estimator	out2	Areas	in2
24	Areas	out2	UAM2	in2
25	Areas	in1	Area_1	in
27	Areas	in2	Area_2	in
28	Area_2	out	Areas	out2
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in

Estimator is in the FPES, so skip it. Areas - out2 is found in coupling 28, replace with Area_2 - out. Areas - in1 is found in coupling 19, replace with Estimator - out1. Areas - in2 is found in coupling 23, replace with Estimator - out2. Delete couplings 28, 19, and 23. The result is in Table G.9:

Table G.9: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the eigth step.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	DAM	out1	Estimator	in1
16	DAM	out2	Estimator	in2
17	DAM	out3	Estimator	in3
18	DAM	out4	Basin1_Basin	in1
20	Area_1	out	Basin1_Basin	in2
21	UAM3	out	UAM2	in1
24	Area_2	out	UAM2	in2
25	Estimator	out1	Area_1	in
27	Estimator	out2	Area_2	in
29	Basin1_Basin	in1	DAM3	in1
30	Basin1_Basin	in2	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in

Basin1_Basin - in1 and in2 can be found on the sink side in couplings 18 and 20, replace with DAM - out4 and Area_1 - out. The other nodes DAM3 and SnowBasin will be part of the FPES, so they need not be adjusted. In Table G.10 the couplings according to the FPES in Figure 4.20 are derived.

Table G.10: Couplings of the PES in Figure 4.15 given in Figure 4.16 after the ninth step - like in the FPES.

no.	source		sink	
3	UAM	out	ToDisk	in
4	Rain	out	DAM	in1
5	Temp	out	DAM	in2
10	UAM2	out	EWS	in1
11	EWS	out1	UAM	in1
12	EWS	out2	UAM	in2
13	EWS	out3	UAM	in3
15	DAM	out1	Estimator	in1
16	DAM	out2	Estimator	in2
17	DAM	out3	Estimator	in3
21	UAM3	out	UAM2	in1
24	Area_2	out	UAM2	in2
25	Estimator	out1	Area_1	in
27	Estimator	out2	Area_2	in
29	DAM	out4	DAM3	in1
30	Area_1	out	DAM3	in2
31	DAM3	out1	SnowBasin	in1
32	DAM3	out2	SnowBasin	in2
33	SnowBasin	out	UAM3	in