

DEPARTMENT OF INFORMATICS

THESIS REPORT

---

# Application Agnostic SES Modeling Environment and Interactive Pruning Tool

---

Bikash CHANDRA KARMOKAR

A thesis submitted to the  
Institute of Computer Science  
Technical University of Clausthal  
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Internet Technologies and Information Systems

**Supervisors:**

First Examiner: Prof. Dr. Sven Hartmann

Second Examiner: PD Dr. Umut Durak

# Declaration of Authorship

I, Bikash CHANDRA KARMOKAR, declare that this thesis titled, 'Application Agnostic SES Modeling Environment and Interactive Pruning Tool' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

---

Date:

---

*“Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do.”*

Pele

# *Abstract*

System Entity Structure (SES) is a high-level ontology which was introduced for knowledge representation of decomposition, taxonomy and coupling of systems. It has its roots from the systems theory-based approaches to modeling and simulation. SES has been applied for various purposes by modeling and simulation community such as creating suites of models for global warming, modeling the elements of a scenario in a research flight simulator or for variant modeling in model-based design. Despite its diverse use cases, there still exists a lack of standardized computational representation. This hinders the shareability of SES artifacts and interoperability of SES tools. Adapting and enhancing the XML-based computational representation proposed by Zeigler and Hammond, we are proposing standardization in SES artifacts. Furthermore, we introduce an application agnostic tool suite: SESEditor and PESEditor. We exploit the traditional graphical representation in SESEditor and further propose an interactive pruning approach with PESEditor. Finally, we exploit the proposed standardization, the SESEditor and the PESEditor with example application use cases.

# *Acknowledgements*

First and foremost, praises and thanks to the God, the Almighty, for his showers of blessings throughout my research work to complete the research successfully.

I would like to express my deep and sincere gratitude to my thesis advisors Prof. Dr. Sven Hartmann and Priv.-Doz. Dr. Umut Durak for the continuous support of my study and research, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisors, I would like to thank to Asst. Prof. Dr. Shafagh Jafer, Prof. Dr.-Ing. Thorsten Pawletta, Hendrik Folkerts, Bharvi Chhaya for their insightful comments and encouragement.

My sincere thanks also goes to Institute of Flight Systems of German Aerospace Center (DLR) for providing me an opportunity to join their team as a student assistant, and giving me the access to the laboratory and research facilities.

Finally, I must express my very profound gratitude to my parents for their love, prayers, caring and sacrifices for educating and preparing me for my future. This accomplishment would not have been possible without them. Thank you.

# Contents

<b>Declaration of Authorship</b>	<b>i</b>
<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>Contents</b>	<b>v</b>
<b>List of Figures</b>	<b>ix</b>
<b>List of Tables</b>	<b>xi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Existing Problems . . . . .	2
1.1.1 Computational Representation . . . . .	2
1.1.2 Common Tool Suite . . . . .	2
1.2 The Contribution of This Thesis . . . . .	3
1.2.1 Computational Representation Enhancement . . . . .	3
1.2.2 Application Agnostic Tool Suite . . . . .	4
1.2.3 Research Questions . . . . .	4
1.3 Thesis Outline . . . . .	5
<b>2 Background</b>	<b>7</b>
2.1 System Entity Structure . . . . .	7
2.1.1 SES Elements . . . . .	7
2.1.2 SES Axioms . . . . .	8
2.2 Modeling Using SES . . . . .	9
2.2.1 Modeling Entity . . . . .	10
2.2.2 Modeling Association . . . . .	10
2.2.3 Modeling Attributes . . . . .	12
2.2.4 Modeling Semantic Constraint . . . . .	12
2.3 Pruning an SES Model . . . . .	13
2.3.1 Pruning . . . . .	13
2.3.2 Pruning Procedure . . . . .	14
2.4 Computational Representation . . . . .	15

2.4.1	SES as an XML . . . . .	15
2.4.2	SES as an XSD . . . . .	15
2.4.3	SES Ontology as XML Schema . . . . .	17
2.4.4	XPath . . . . .	23
2.4.4.1	XPath Examples . . . . .	23
<b>3</b>	<b>Modeling Environment: SESEditor</b>	<b>25</b>
3.1	Introduction . . . . .	25
3.2	Graphical User Interface . . . . .	26
3.2.1	Block 1: Toolbox . . . . .	26
3.2.2	Block 2: Project Window . . . . .	26
3.2.3	Block 3: Drawing Panel . . . . .	26
3.2.4	Block 4: Synchronized Tree Window . . . . .	26
3.2.5	Block 5: Console Window . . . . .	28
3.2.6	Block 6: Variable Window . . . . .	29
3.2.7	Block 7: Constraint Window . . . . .	29
3.2.8	Block 8: Display Window . . . . .	30
3.2.9	SESEditor Menus . . . . .	31
3.3	Functionality . . . . .	34
3.3.1	Element Addition . . . . .	34
3.3.2	Element Deletion . . . . .	35
3.3.3	Element Renaming . . . . .	35
3.3.4	Variable Addition . . . . .	36
3.3.5	Variable Deletion . . . . .	37
3.3.6	Constraint Addition . . . . .	37
3.3.7	Constraint Deletion . . . . .	37
3.3.8	Module Saving . . . . .	38
3.3.9	Module Addition . . . . .	39
3.3.10	Validation . . . . .	40
3.3.11	Generating SES XML and SES Schema . . . . .	41
3.4	Package Design and Class Static Structure . . . . .	41
3.4.1	Common Packages . . . . .	41
3.4.2	SESEditor Packages . . . . .	42
3.4.3	Package Design . . . . .	42
3.4.4	Class Diagram: core . . . . .	43
3.4.5	Class Diagram: seseditor . . . . .	43
3.5	Used Technologies . . . . .	43
3.5.1	Java . . . . .	43
3.5.2	Xerces . . . . .	46
3.5.3	JGraphX . . . . .	47
3.5.4	RSyntaxTextArea . . . . .	49
3.5.5	Goava . . . . .	49
<b>4</b>	<b>Interactive Pruning Tool: PESEditor</b>	<b>50</b>
4.1	Introduction . . . . .	50
4.2	Graphical User Interface . . . . .	50
4.2.1	Block 1: Toolbox . . . . .	52

4.2.2	Block 2: Project Window . . . . .	52
4.2.3	Block 3: Drawing Panel . . . . .	53
4.2.4	Block 4: Synchronized Tree Window . . . . .	54
4.2.5	Block 5: Console Window . . . . .	54
4.2.6	Block 6: Variable Window . . . . .	54
4.2.7	Block 7: Constraint Window . . . . .	54
4.2.8	Block 8: Display Window . . . . .	54
4.2.9	PESEditor Menus . . . . .	55
4.3	Functionality . . . . .	57
4.3.1	Interactive Pruning . . . . .	57
4.3.2	Validation . . . . .	57
4.3.3	Generating PES XML and PES Schema . . . . .	57
4.3.4	XSLT Transformation . . . . .	57
4.4	Pruning Procedures Implementation in PESEditor . . . . .	58
4.4.1	Aspect Node Pruning . . . . .	58
4.4.2	Multi-Aspect Node Pruning . . . . .	58
4.4.3	Specialization Node Pruning . . . . .	59
4.4.4	Aspect Siblings Pruning . . . . .	59
4.4.5	Specialization Siblings Pruning . . . . .	59
4.4.6	Multi-Aspect Siblings Pruning . . . . .	60
4.4.7	Aspect and Multi-Aspect Siblings Pruning . . . . .	61
4.4.8	Two Specialization Nodes in One Path Pruning . . . . .	61
4.4.9	Specialization with Succeeding Aspect Pruning . . . . .	62
4.4.10	Specialization and Aspect Siblings Pruning . . . . .	62
4.5	Package Design and Class Static Structure . . . . .	65
4.5.1	Common Packages . . . . .	65
4.5.2	PESEditor Packages . . . . .	65
4.5.3	Package Design . . . . .	66
4.5.4	Class Diagram: core . . . . .	66
4.5.5	Class Diagram: peseditor . . . . .	66
4.6	Used Technologies . . . . .	68
4.6.1	Java . . . . .	68
4.6.2	Xerces . . . . .	68
4.6.3	JGraphX . . . . .	68
4.6.4	RSyntaxTextArea . . . . .	68
4.6.5	Goava . . . . .	68
<b>5</b>	<b>Use Cases</b> . . . . .	<b>69</b>
5.1	Music Performances . . . . .	69
5.2	Flight Simulation Scenario Development . . . . .	71
5.2.1	Scenario Related Work . . . . .	73
5.2.2	Scenario Development Process . . . . .	73
5.2.3	Scenario Examples . . . . .	74
5.2.3.1	Aviation Scenario Core . . . . .	75
5.2.3.2	Traffic Server . . . . .	75
5.2.3.3	Air Traffic Control . . . . .	79
5.3	Geofence . . . . .	83



<b>6 Conclusion</b>	<b>87</b>
<b>A Class Descriptions of Editors</b>	<b>89</b>
A.1 Classes of core package . . . . .	89
A.2 Classes of seseditor package . . . . .	90
A.3 Classes of schema package . . . . .	91
A.4 Classes of peseditor package . . . . .	92
A.5 Classes of xslt package . . . . .	92
<b>B Menus of Editors</b>	<b>93</b>
B.1 Menus of SESEditor . . . . .	93
B.1.1 Menus . . . . .	93
B.1.1.1 File: New . . . . .	93
B.1.1.2 File: Open . . . . .	94
B.1.1.3 File: Save . . . . .	94
B.1.1.4 File: Save As... . . . .	94
B.1.1.5 File: Import . . . . .	95
B.1.1.6 File: Export . . . . .	95
B.1.1.7 File: Exit . . . . .	95
B.1.1.8 Help: Manual . . . . .	95
B.1.1.9 Help: About . . . . .	95
B.2 Menus of PESEditor . . . . .	95
B.2.1 Menus . . . . .	95
B.2.1.1 File: Open . . . . .	96
B.2.1.2 File: Save . . . . .	96
B.2.1.3 File: Save As... . . . .	96
B.2.1.4 File: Export . . . . .	96
B.2.1.5 File: Exit . . . . .	97
B.2.1.6 Help: Manual . . . . .	97
B.2.1.7 Help: About . . . . .	97
<b>Bibliography</b>	<b>98</b>

# List of Figures

1.1	Player Selection Metamodel. . . . .	3
2.1	System Entity Structure: Elements. . . . .	8
2.2	System Entity Structure Example. . . . .	9
2.3	Car Selection Metamodel. . . . .	13
2.4	Aircraft Metamodel Using SES. . . . .	14
2.5	Pruned Model of Aircraft . . . . .	15
3.1	SESEditor User Interface. . . . .	27
3.2	Project Window. . . . .	28
3.3	Project Window With Added Module. . . . .	28
3.4	Drawing Panel With Example. . . . .	28
3.5	JTree Representation of Figure 3.4. . . . .	29
3.6	Console Window. . . . .	29
3.7	Variable Table. . . . .	29
3.8	Constraint Table. . . . .	29
3.9	SES Ontology, XML and Schema Display Window. . . . .	30
3.10	SES Ontology Tab. . . . .	30
3.11	SES XML Tab. . . . .	31
3.12	SES Schema Tab. . . . .	32
3.13	Element Addition from Drawing Panel. . . . .	34
3.14	Element Addition From Left Tree Panel. . . . .	35
3.15	Element Deletion From Drawing Panel. . . . .	35
3.16	Element Renaming From Drawing Panel. . . . .	36
3.17	Variable Addition. . . . .	36
3.18	Variable Addition. . . . .	37
3.19	Constraint Addition. . . . .	38
3.20	Constraint Modification. . . . .	38
3.21	Constraint Deletion. . . . .	39
3.22	Module Saving. . . . .	39
3.23	Module Addition. . . . .	39
3.24	Common Package Structure. . . . .	42
3.25	SESEditor Package Structure. . . . .	42
3.26	SESEditor Package Design. . . . .	43
3.27	SESEditor Class Diagram: core package. . . . .	44
3.28	SESEditor Class Diagram: seseditor package. . . . .	45
4.1	PESEditor User Interface. . . . .	51

4.2	Project Window.	53
4.3	Project Window With Added Module.	53
4.4	Drawing Panel Example Before Pruning.	53
4.5	JTree Representation of Figure 4.4.	54
4.6	PES XML and SES Schema Display Window	55
4.7	PES XML Tab.	55
4.8	PES Schema Tab.	56
4.9	Pruning of Aspect Node.	58
4.10	Pruning of Multi-Aspect Node.	58
4.11	Pruning of Specialization Node.	59
4.12	Pruning of Aspect Siblings Node.	60
4.13	Pruning of Specialization Siblings Node.	60
4.14	Pruning of Multi-Aspect Siblings Node.	61
4.15	Pruning of Aspect and Multi-Aspect Siblings Node.	61
4.16	Pruning of Two Specialization Nodes in One Path.	62
4.17	Pruning of Specialization with Succeeding Aspect Node.	62
4.18	Pruning of Specialization and Aspect Siblings Node.	64
4.19	Common Package Structure.	65
4.20	PESEditor Package Structure.	66
4.21	PESEditor Package Design.	66
4.22	PESEditor Class Diagram: core package.	67
4.23	PESEditor Class Diagram: peseditor package.	68
5.1	SES Metamodel for Musical Performances	70
5.2	Symphonic Musical Performance.	71
5.3	Folk Musical Performance.	71
5.4	Jazz Musical Performances.	72
5.5	SES for Scenario Development (Durak et al. [2018b]).	75
5.6	Aviation Scenario Core.	76
5.7	Traffic Server.	77
5.8	Pruned Traffic Server.	78
5.9	Air Traffic Control.	80
5.10	Pruned Air Traffic Control.	81
5.11	Geofence Metamodel Using SES.	84
5.12	Pruned Model of Geofence	85
B.1	File Menus.	93
B.2	Help Menus.	93
B.3	New Project Window.	94
B.4	File Menus.	96
B.5	Help Menus.	96

# List of XML Codes

1	Constraint Example Using XPath and Assert Statement. . . . .	4
2	Constraint for Ensuring BMW and Germany have been Selected Together. . . . .	13
3	Constraint for Ensuring Ford and USA have been Selected Together. . . . .	13
4	XML instance of the SES ontology XML Schema . . . . .	16
5	XPath Query for Strict Hierarchy Axiom of the SES. . . . .	18
6	XPath Query for Valid Brothers Axiom of the SES. . . . .	18
7	XPath Query for Attached Variables Axiom of the SES. . . . .	18
8	XML Schema of SES Ontology: Part 1 . . . . .	19
9	XML Schema of SES Ontology: Part 2 . . . . .	20
10	XML instance of the SES Metamodel presented in figure 2.2. . . . .	24
11	Constraint Example for Symphonic Musical Performance Combinations. . . . .	70
12	Constraint Example for Folk Musical Performance Combinations. . . . .	70
13	Constraint Example for Jazz Musical Performance Combinations. . . . .	70
14	Excerpt from the Executable Scenario of Traffic Server . . . . .	82

# List of Tables

2.1	Representation of an SES as a DTD (Zeigler and Hammonds [2007]). . . . .	17
2.2	Representation of an SES as an XML Schema. . . . .	21
2.3	Representation of an SES as an XML Schema (Reduced). . . . .	22
2.4	XPath Examples . . . . .	24
3.1	ToolBox Description of SESEditor . . . . .	33
4.1	ToolBox Description of PESEditor . . . . .	52
5.1	Possible Music Performances . . . . .	69

*Dedicated to my parents who sacrificed their comfort for  
supporting me to reach this phase of my life. . . .*

# Chapter 1

## Introduction

The System Entity Structure (SES) is a high level ontology framework targeted to modeling, simulation, systems design and data engineering ([Zeigler and Hammonds \[2007\]](#)). An SES is a formal structure governed by a set of elements and a small number of axioms that provide clarity and consistency to its models. The elements of SES are entity, aspect, specialization and multiple-aspect ([Zeigler and Hammonds \[2007\]](#)). Using entity nodes real or artificial system components can be represented ([Zeigler et al. \[2000\]](#)). An entity is an object of interest, and can also have variables attached to it. The decomposition relationship and the taxonomy of an entity node is denoted by aspect and specialization nodes respectively. A multi-aspect node specifies the parent entity as a composition of multiple entities of the same type. There are six axioms of SES: uniformity, strict hierarchy, alternating mode, valid brothers, attached variables and inheritance ([Zeigler \[1984\]](#)). Uniformity axiom confirms any two nodes with the same labels with isomorphic subtrees. Strict hierarchy prevents a label from appearing more than once down any path of the tree. To satisfy alternating mode, if a node is an entity, then the successor has to be either aspect or specialization and vice versa. Valid brothers axiom disallow two brothers from having the same label. An attached variable specifies variable types attached to the same item with distinct names. Specialization inherits all variables and aspects during inheritance.

SES supports hierarchical and modular compositions allowing large complex structures to be built in step wise fashion from smaller, simpler ones. Various tools have been developed to transform SESs back and forth to XML allowing many operations to be specified in either SES directly or in its XML guise. The axioms and functionally based semantics of the SES promote practical design and are easily understandable by data modelers ([Salas \[2008\]](#)). SES is used in many application domains for structural knowledge representation. It is particularly suitable for describing system configurations

for various application domains (Deatcu et al. [2018]). For efficient use of resources in modeling and simulation based data engineering SES has become popular also. So far it has been applied for various purposes by modeling and simulation community.

Since the SES theory has published, many researchers around the globe have extended the classic SES theory (Santucci et al. [2016], Pawletta et al. [2016], Schmidt et al. [2016], Durak et al. [2018b], Deatcu et al. [2018]). The Aeronautical Informatics which is a collaborative research group of TU Clausthal Department of Informatics and German Aerospace Center (DLR) Institute of Flight Systems at TU Clausthal is also working on SES theory extension and its tooling. As a research outcome we have extended computational representation of SES by adding constraint and developed application agnostic SES modeling environment and interactive pruning tool.

## 1.1 Existing Problems

### 1.1.1 Computational Representation

For promoting wide acceptance and standardization, computational representation of SES, which was proposed by Zeigler and Hammonds [2007] is still a key challenge to handle. By establishing widely accepted standardized computational representation this challenge can be solved. SES ontology vows an applicable formal means of domain modeling in various fields. Despite of SES's diverse use cases still there is no common computational representation for this. Lack of standardized computational representation hinders shareability of SES artifacts among researchers. Capability of adding condition is also very important in computational representation. Deatcu et al. [2018] used semantic condition to limit the value range of the variable but didn't explain anything about its computational representation. So far, there is no way of adding condition in the computational representation. That's why, extension of computational representation is necessary to add condition.

### 1.1.2 Common Tool Suite

SES has been applied for various purposes by modeling and simulation community such as creating suites of models for global warming, modeling the elements of a scenario in a research flight simulator or for variant modeling in model-based design. Various tools have been used for modeling in these cases because there are no common tools for SES modeling environment and interactive pruning. It eventually leads to lowered shareability of simulation scenarios across researchers around the world.



## 1.2 The Contribution of This Thesis

### 1.2.1 Computational Representation Enhancement

Computational representation denotes particular SES in XML Schema format. For completeness and consistency condition is very important in XML Schema. As there is no way of adding condition on current computational representation, we have extended computational representation by introducing constraint as a condition in the representation using XPath 2.0 and XML Schema assert (xs:assert). Figure 1.1 shows a player selection metamodel. Here, the Name node contains two players, Messi and Ronaldo and the Country node contains two countries, Portugal and Argentinian. During selecting player, if Messi is selected then his country should be Argentina in Country node and if Ronaldo is selected then his country should be Portugal in Country node. To add condition like these, we have enhanced computational representation by introducing constraint. Listing 1 shows our proposed constraint example of restricting player selection according to his country using XPath in assert statement. Being already familiar to the modeling and simulation community, XPath will bring easy adaption and the enhancement of the computational representation will be widely accepted.

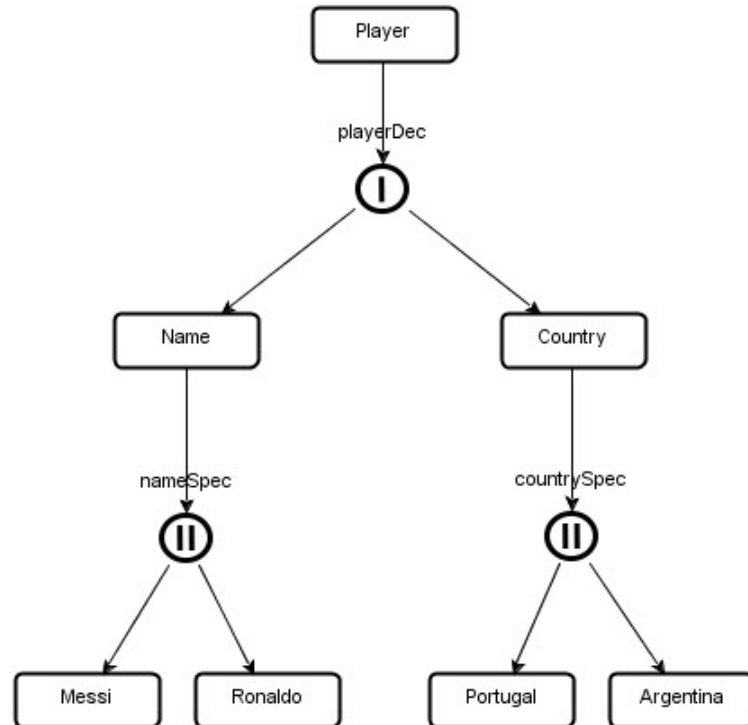


FIGURE 1.1: Player Selection Metamodel.

```
<xs:assert test=
"if(Name/*[@name='Messi']) then (Country/*[@name='Argentina']) else true()"
/>
```

LISTING 1: Constraint Example Using XPath and Assert Statement.

### 1.2.2 Application Agnostic Tool Suite

In modeling and simulation community SES has been using in various fields. SES is used to create suites of models for global warming example (Zeigler et al. [2013]). For XML meta data modeling in US climate normals application SES is used (Cheon et al. [2008]). In case of variant modeling SES is also used (Pawletta et al. [2015]). Not only these but also in aviation sector now use of SES is spreading. Recently, to model the elements of a scenario in a research flight simulator SES is used (Durak et al. [2017]). The use of SES is increasing day by day. Though many people in various fields are using SES for different kinds of modeling till now there is no common editor for all fields. Keeping this inconsistency in mind we have developed these application agnostic SES modeling environment and interactive pruning tools. These tools will remove modeling and pruning barrier among researchers by bringing consistency. Developed tool suite has the following features:

- (i) Graphical User Interface(GUI)
- (ii) Tree and Graph based SES modeling environment
- (iii) Variable and Constraint adding facility
- (iv) Interactive SES pruning facility
- (v) XML Schema and XML Instance generation facility
- (vi) Module adding from existing project and saving for later use
- (vii) XSLT transformation facility and
- (viii) XML Schema and XML Instance generation facility

### 1.2.3 Research Questions

SES is applied for various purposes by modeling and simulation community. Despite its diverse use cases, there still exists a lack of standardized computational representation, common tools, formats and practices. This eventually impairs shareability and interoperability.

Some researches have been conducted for standard computational representation and also developed application specific tools for modeling using SES. Following these advancement we will try to enhance the computational representation by adding constraint and develop an application agnostic tool suite for modeling and pruning through this study.

Thus, the assumption of this thesis is:

*To find a way of adding constraint in computational representation and develop an application agnostic tool suite for modeling using SES.*

Through out this thesis we will try to answer the following questions to reach the aforementioned assumption:

**Question 1:** Is it possible to add constraints for limiting choices?

**Question 2:** How can we implement SES axioms in XML Schema?

**Question 3:** How can we make the tool application agnostic?

**Question 4:** How can we create reusable and shareable model?

**Question 5:** Which technology will be used for implementation?

## 1.3 Thesis Outline

### Chapter 2

This chapter provides background information about System Entity Structure and its elements and axioms, modeling using SES, pruning an SES model, computational representation and use of XPath during modeling for adding constraint.

### Chapter 3

This chapter describes the SES modeling environment which is named as SESEditor, its graphical user interface, functionality, package structure, class diagram and used technologies.

### Chapter 4

This chapter describes the interactive pruning tool which is named as PESEditor, its graphical user interface, functionality, package structure, class diagram and used technologies.

### Chapter 5

This chapter describes uses of SESEditor and PESEditor tools providing three different use cases.

**Chapter 6**

To conclude this research this chapter provides summary of the whole enhancement and implementation as a contribution to the modeling and simulation community.

## Chapter 2

# Background

### 2.1 System Entity Structure

System Entity Structure (SES) is a high-level ontology which was introduced for knowledge representation of decomposition, taxonomy and coupling of systems ([Kim et al. \[1990\]](#)). SES has a clear limited set of elements and axioms. Element represents objects and its relationship and axioms controls these relationships by imposing various rules.

#### 2.1.1 SES Elements

According to [Zeigler and Hammonds \[2007\]](#) there are four elements of SES which are Entity, Aspect, Specialization and Multiple-Aspect.

- (i) Entity: It represents real or artificial system components. It is an object of interest and also variable can be attached to the Entity. To describe parent and child entities other node types are used.
- (ii) Aspect: It denotes the decomposition relationship of an Entity node. It represents ways of decomposing things into more fine-grained ones.
- (iii) Specialization: It represents the taxonomy of an Entity. Categories or families of specific forms that a thing can assume are represented by Specialization.
- (iv) Multi-aspect: It is a special kind of aspect represents a multiplicity relationship that specifies the parent entity as a composition of multiple entities of the same type.

Among these four elements there are two groups: Entity nodes and 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, Specialization and Multi-Aspect nodes.

An SES is represented by a directed graph whose nodes are Entities, Aspects, multi-Aspects and Specializations. Edges determine the relations between two nodes using one of the descriptive nodes. Entity node is represented by a rectangle and Aspects, Specializations and Multi-Aspect descriptive nodes are represented by single line, double lines and triple lines vertically drawn inside a circle respectively. Figure 2.1 shows the Entity and Descriptive elements of SES and Figure 2.2 shows an example of using SES.

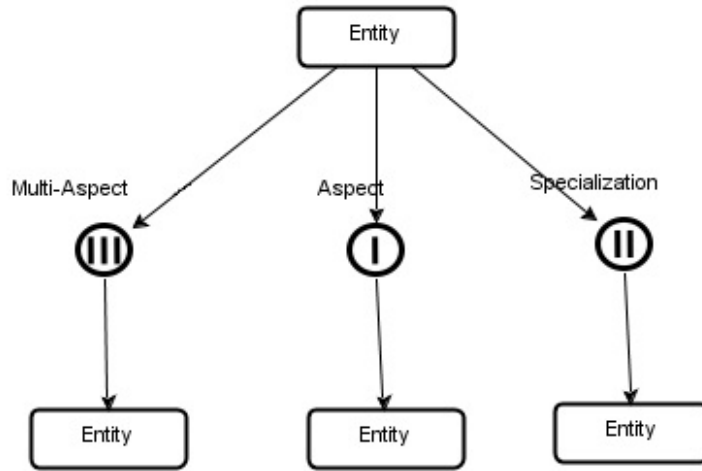


FIGURE 2.1: System Entity Structure: Elements.

### 2.1.2 SES Axioms

According to Zeigler [1984] there are six axioms of SES which are uniformity, strict hierarchy, alternating mode, valid brothers, attached variables and inheritance.

- (i) Uniformity: According to uniformity axiom any two nodes with the same label have isomorphic subtrees.
- (ii) Strict hierarchy: It defines a restriction that prevents a label from appearing more than once down any path of the tree.

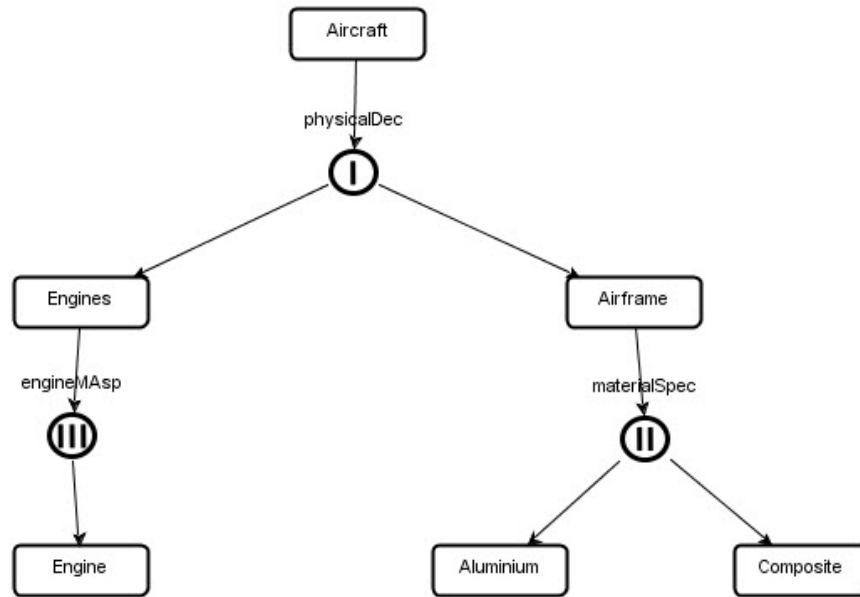


FIGURE 2.2: System Entity Structure Example.

- (iii) Alternating mode: It recommends that, if a node is an Entity, then the successor is either Aspect or Specialization and if the node is Aspect or Specialization then the successor must be an Entity node.
- (iv) Valid brothers: It disallow two brothers from having the same label. Each added element should be unique if they have same parent node.
- (v) Attached variables: It specifies a constraint that variable types attached to the same item shall have distinct names.
- (vi) Inheritance: According to inheritance, it is indicated that Specialization inherits all variables and Aspects.

## 2.2 Modeling Using SES

SES is one of the significant enhancements from the system theory based approach to modeling and simulation. To define data engineering ontologies SES is a very useful ontological framework. SES represents various system classifications which are logically possible set of entities.

### 2.2.1 Modeling Entity

According to Zeigler [1984], data engineering refers to activities regarding data in design, development, management and utilization of information systems. Zeigler and Hammonds [2007] proposed SES as an ontological framework for simulation data modeling. Durak et al. [2017] proposed an SES based approach for modeling scenario elements. During modeling, filtering entity elements from data is very important activity. For each system there should be an object of interest. The main object should be root entity element. Consider the description of the Aircraft shown in figure 2.2 : **From physical perspective Aircraft is made of Engines and Airframe. From multiple perspective Engines are made of one or more Engine. Airframe can be Aluminium or Composite in material.** From the description it is clear that whole paragraph is circulating based on Aircraft. So during modeling, this Aircraft would be root entity element. Other objects are Engines, Airframe, Engine, Aluminium and Composite. During modeling using SES these objects will be used as entity elements.

### 2.2.2 Modeling Association

Association connects the entities. From the model description have to find relation between nodes. SES elements aspect, specialization and multi-aspect are used to build this association. From above paragraph, Aircraft can be decomposed into Engines and Airframe. So SES element Aspect can be used to make association between these three entities. Again, there are more than one engine so SES element Multi-Aspect can be used to make association between these two nodes. Finally, there is a choice for choosing material for building Airframe from Aluminium and Composite. So, SES element Specialization can be used to make association between these entities to represent taxonomy of Airframe entity.

A set-theoretic characterization of the SES is provided by Zeigler and Hammonds [2007]. It shows how the axioms are satisfied, particularly, the axioms of uniformity and alternation. According to Park et al. [1997] this provides an alternative formulation of the SES as a relational structure that offers insight into the roles of the elements and provides the basis equality tests and comparison of SESs. Zeigler and Hammonds [2007] employ this set-theoretic structure to prove various properties needed to support manipulations of SES structures for application to data engineering and modeling and simulation. The relational formulation is as follows:

An SES is specified by a structure:



$SES = <$   
 Entities,  
 Aspects,  
 Specializations,  
 rootEntity,  
 entityHasAspect,  
 entityHasMultiAspect,  
 entityHasSpecialization,  
 aspectHasEntity,  
 multiAspectHasEntity  
 multiAspectHasVariable  
 specializationHasEntity,  
 entityHasVariable,  
 variableHasRange,  
 $>$

where

- Entities, Aspects, Specializations, Variables, multiAspects are finite mutually disjoint sets
- rootEntity  $\in$  Entities

and the following are relations:

$entityHasAspect \subseteq Entities \times Aspects$   
 $entityHasMultiAspect \subseteq Entities \times multiAspects$   
 $entityHasSpecialization \subseteq Entities \times Specializations$   
 $aspectHasEntity \subseteq Aspects \times Entities$   
 $multiAspectHasEntity \subseteq multiAspects \times Entities$   
 $multiAspectHasVariable \subseteq multiAspects \times Variables$   
 $specializationHasEntity \subseteq Specializations \times Entities$   
 $entityHasVariable \subseteq Entities \times Variables$   
 $variableHasRange \subseteq Variables \times RangeSpec$

The SES is represented by a directed graph whose nodes are Entities, Aspects, multi-Aspects, Specializations and edges determined with the relations above. The Aircraft description written in section 2.2.1 describes the following SES.

$SesForAircraft = <$   
 Entities = Engines, Engine, Airframe, Aliminium, Composite

```

Aspects = physicalDec
multiAspects = engineMultiAsp
Specializations = materialSpec
rootEntity = Aircraft
entityHasAspect = (Aircraft, physicalDec)
entityHasMultiAspect = (Engines, engineMultiAsp)
entityHasSpecialization = (Airframe, materialSpec)
aspectHasEntity = ( physicalDec, Engines), ( physicalDec, Airframe)
multiAspectHasEntity = ( engineMultiAsp, Engine)
multiAspectHasVariable = ( engineMultiAsp, numberofengines)
specializationHasEntity = ( materialSpec, Aliminium), (materialSpec, Composite)

>

```

So based on the SES *SesForAircraft* the model can be visualized like figure 2.2.

### 2.2.3 Modeling Attributes

During modeling, use of attributes is also very important. By changing its values during pruning of SES model, multiple scenarios can be designed for testing. Consider the Aircraft description written in section 2.2.1, for Engine there might be attributes like engineType or engineName and for Aluminium there might be attributes like sheetType or strength. Each attributes attached to the element gives extra information about that entity.

### 2.2.4 Modeling Semantic Constraint

During modeling constraint is very important. Sometimes it is needed to provide fixed choices. Also combination of choices from two nodes is needed in few cases. For setting these kinds of constraints use of XPath query in Aspect node has been introduced. Figure 2.3 shows an car selection metamodel. For selecting car two choices BMW and Ford have been added in Name node and two choices Germany and USA have been added as the origin of those cars in Country node. Here, Car selection could be restricted in two ways. First, if BMW is the selected car in Name node then country should be Germany in Country node. Second, if Ford is the selected car in Name node then country should be USA in Country node. If BMW has been selected in Name node then it would be wrong to select USA in Country node. For setting these constraints the Aspect node carDec where Name and Country nodes are connected is used. For ensuring first and

second restrictions described above XPath queries shown in listing 2 and listing 3 have been added in carDec Aspect node.

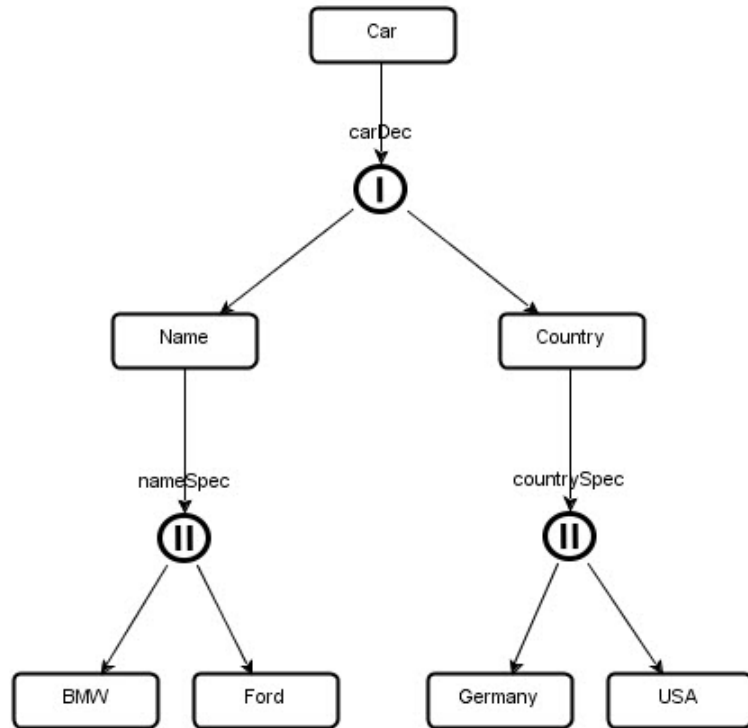


FIGURE 2.3: Car Selection Metamodel.

```

<xs:assert test=
  "if(Name/*[@name='BMW']) then (Country/*[@name='Germany']) else true()"
/>

```

LISTING 2: Constraint for Ensuring BMW and Germany have been Selected Together.

```

<xs:assert test=
  "if(Name/*[@name='Ford']) then (Country/*[@name='USA']) else true()"
/>

```

LISTING 3: Constraint for Ensuring Ford and USA have been Selected Together.

## 2.3 Pruning an SES Model

### 2.3.1 Pruning

Pruning is the operation in which a unique system structure is derived from an SES and the result is called Pruned Entity Structure (PES). SES represents a model of a given application domain in terms of decompositions, component taxonomies and coupling

specifications. During modeling using SES, all the available options of a system are considered. As an SES describes a number of system configurations, the SES tree needs to be pruned to get a particular configuration. Pruning cut off unnecessary structure from an SES tree based on the specification of a realistic frame to bring this particular configuration or PES which is a selection-free tree. The pruning process normally reduces SES by removing choices of entity which has multiple aspects and specialization consists of multiple entities.

### 2.3.2 Pruning Procedure

An SES tree can be pruned by performing following tasks:

- (i) Assigning values to the variables
- (ii) Choosing particular subject from several aspect nodes for several decomposition of the system on same hierarchical level
- (iii) Selecting one entity from various options of specialization node.
- (iv) Specifying cardinality in Multi-Aspect node

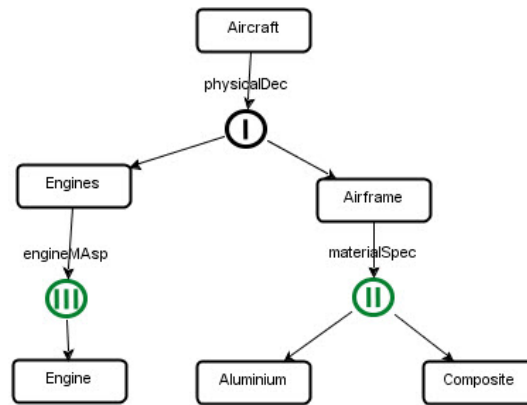


FIGURE 2.4: Aircraft Metamodel Using SES.

Figure 2.4 shows an Aircraft metamodel using SES. Figure 2.5 shows the pruned model of the Aircraft metamodel. During pruning the cardinality of multi-aspect node is set to 2 here and as a result Engine\_1 and Engine\_2 are generated in pruned entity structure. Also Composite entity is selected from available options of specialization node and Composite\_Airframe node is created in the final pruned entity structure. In this example tasks (iii) and (iv) of pruning procedure are implemented.

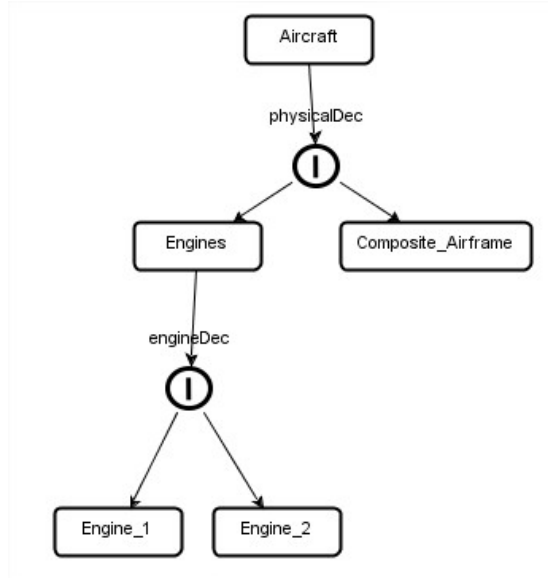


FIGURE 2.5: Pruned Model of Aircraft

## 2.4 Computational Representation

Computational representation is the process of representing SES in XML formats and it is proposed by [Zeigler and Hammonds \[2007\]](#). While the SES ontology promises an effective formal means of domain modeling, computational representation is still a key challenge to handle for promoting wide-acceptance and standardization. [Zeigler and Hammonds \[2007\]](#) have proposed various ways to construct and represent SES in computational form.

### 2.4.1 SES as an XML

The car selection metamodel using SES presented in Figure 2.3 can be represented using XML like listing 4. This XML must obey the structure imposed by a Data Type Definition (DTD) file, a document type definition that captures the alternating mode axiom of the SES specification discussed in 2.1.2. For example, as illustrated in Table 2.1, an entity can not be placed directly within the scope of another one — only aspects, multiAspects, specializations, and variables are allowed under an entity. Likewise, only entities can be placed within the scopes of aspects, multiAspects, and specializations.

### 2.4.2 SES as an XSD

The XML representation of the particular SES can be written into an XML Schema Definition (XSD). XSD specifies how to formally describe the elements in an Extensible

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entity xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="Car">
  <aspect name="carDec">
    <entity name="Name">
      <specialization name="nameSpec">
        <entity name="BMW">
        </entity>
        <entity name="Ford">
        </entity>
      </specialization>
    </entity>
    <entity name="Country">
      <specialization name="countrySpec">
        <entity name="Germany">
        </entity>
        <entity name="USA">
        </entity>
      </specialization>
    </entity>
  </aspect>
</entity>

```

LISTING 4: XML instance of the SES ontology XML Schema .

Markup Language (XML) document. XSD can be used to express a set of rules to which an XML document must conform in order to be considered "valid" according to that schema. To validate an SES XML document based on SES axioms, an XSD can be created using all the rules or conditions which are presented as SES axioms by [Zeigler \[1984\]](#). XSD is being used because it has several advantages over earlier XML schema languages, such as Document Type Definition (DTD). XSD is written in XML, which means that it doesn't require intermediary processing by a parser. Other benefits include self-documentation, automatic schema creation and the ability to be queried through XML Transformations (XSLT). For developing an XSD from an SES XML, such as the XML instance for car selection metamodel presented in Listing 4, the proposed structure of [Zeigler and Hammonds \[2007\]](#) as shown in Table 2.2 has adapted. SES Entity can be modeled as XML Schema element (xs:element) with a name attribute that designates the SES Entity name. It is by definition an XML Schema complex type (xs:complexType) which can contain elements and/or attributes. Aspect can also be represented by an XML Schema element (xs:element) with a name attribute that designates the SES Aspect name. It will also be a complex type which has an XML Schema sequence (xs:sequence) composed of XML Schema elements (xs:element). MultiAspect can be represented pretty much the same way as the Aspect, but with only having one element in its sequence. The MultiAspect element possesses minOccurs and maxOccurs properties to specify the

SES Item	Maps to XML specification
<b>Entity</b>	<code>&lt;!ELEMENT entity ((aspect   specialization   multiAspect)*,var*)&gt; &lt;!ATTLIST entity name CDATA #REQUIRED&gt;</code>
<b>Aspect</b>	<code>&lt;!ELEMENT aspect (entity*)&gt; &lt;!ATTLIST aspect name CDATA #REQUIRED&gt;</code>
<b>Multi-Aspect</b>	<code>&lt;!ELEMENT multiAspect (numberOfComponents,entity*)&gt; &lt;!ATTLIST multiAspect name CDATA #REQUIRED&gt;</code>
<b>Specialization</b>	<code>&lt;!ELEMENT specialization (entity*)&gt; &lt;!ATTLIST specialization name CDATA #REQUIRED&gt;</code>
<b>Variable</b>	<code>&lt;!ELEMENT var (#PCDATA) &lt;!ATTLIST var name CDATA #REQUIRED&gt;</code>

TABLE 2.1: Representation of an SES as a DTD (Zeigler and Hammonds [2007]).

cardinality. Specialization is accordingly an element with a complex type. XML Schema choice (`xs:choice`) is then used to specify the specialization relation to child entities. Variable can be defined by attributes where attribute name is used to specify the name of a variable and type for its type.

There is an overhead associated with representing an SES following the structure presented in Table 2.2. A general restructuring is recommended by Zeigler and Hammonds [2007] for eliminating the elements including aspect, multiAspect and specialization. Eliminating them in a schema for an SES will reduce that overhead for representing the domain structure. The semantics of aspect and multiAspect can be represented by an `xs:sequence` and multiAspects with `xs:sequence` and `minOccurs` and `maxOccurs` attributes. `xs:choice` can then represent the semantic for the specialization. Following recommendation Durak et al. [2018b] showed the reduced XML schema for an SES which is presented in Table 2.3.

### 2.4.3 SES Ontology as XML Schema

The Data Type Definition (DTD) can impose alternating mode axiom of the SES definition but other important axioms such as uniformity, valid brothers, strict hierarchy and attached variables cannot be enforced by the DTD. Durak et al. [2018b] stated that the XML Schema assert (`xs:assert`) can be utilized to formalize the axioms of the SES

ontology that are introduced in 2.1.2 section. According to Gao et al. [2009] assertions provide means to constrain the existence and values of elements and attributes in XML Schemas. Assertions are specified using XPath 2.0 which is a functional language that is used to navigate through elements and attributes.

DTD representation of the SES ontology that is proposed by Zeigler and Hammonds [2007] presented in table 2.1 is adapted to prepare the SES ontology as XML Schema which is presented in Listing 8 and 9. Following Durak et al. [2018b] to formalize SES axioms named strict hierarchy, valid brothers and attached variables the XML Schema assert (xs:assert) with XPath query has been utilized here. XPath Query for Strict Hierarchy, Valid Brothers and Attached Variables axioms of the SES are presented in Listing 5, 6 and 7 respectively.

```
<xs:assert test="every $x in ./entity satisfies empty($x//*[ @name =
$x/@name])"
/>
```

LISTING 5: XPath Query for Strict Hierarchy Axiom of the SES.

```
<xs:assert test="every $x in ./entity satisfies count(*[$x/@name =
$x/following-sibling::*/@name]) = 0"
/>
```

LISTING 6: XPath Query for Valid Brothers Axiom of the SES.

```
<xs:assert test="every $x in ./var satisfies count(*[@name =
following-sibling::*/@name]) = 0"
/>
```

LISTING 7: XPath Query for Attached Variables Axiom of the SES.



```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified"
  xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  vc:minVersion="1.1">

  <xs:complexType name="aspectType">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="entity" />
    </xs:sequence>
    <xs:attribute name="name" use="required" />
  </xs:complexType>

  <xs:complexType name="multiAspectType">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="entity" />
    </xs:sequence>
    <xs:attribute name="name" use="required" />
    <xs:attribute name="constraint" use="optional" />
  </xs:complexType>

  <xs:complexType name="specializationType">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="entity" />
    </xs:sequence>
    <xs:attribute name="name" use="required" />
  </xs:complexType>

  <xs:complexType name="varType">
    <xs:sequence>
      <xs:element minOccurs="0" maxOccurs="unbounded" ref="entity" />
    </xs:sequence>
    <xs:attribute name="name" use="required" />
    <xs:attribute name="type" use="optional" />
    <xs:attribute name="default" use="optional" />
    <xs:attribute name="lower" use="optional" />
    <xs:attribute name="upper" use="optional" />
  </xs:complexType>

```

LISTING 8: XML Schema of SES Ontology: Part 1

```
<xs:element name="entity">
  <xs:complexType>
    <xs:sequence>

      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="aspect" />
        <xs:element ref="specialization" />
        <xs:element ref="multiAspect" />
        <xs:element ref="var" />
      </xs:choice>
    </xs:sequence>

    <xs:attribute name="name" use="required" />
    <xs:attribute name="ref" use="optional" />
    <xs:assert test="every $x in ../entity satisfies empty($x/*[@name =
      $x/@name])" />
    <xs:assert test="every $x in ../entity satisfies count(*[$x/@name =
      $x/following-sibling::*/@name]) = 0" />
    <xs:assert test="every $x in ../var satisfies count(*[@name =
      following-sibling::*/@name]) = 0" />
  </xs:complexType>
</xs:element>

<xs:element name="aspect" type="aspectType" />
<xs:element name="multiAspect" type="multiAspectType" />
<xs:element name="specialization" type="specializationType" />
<xs:element name="var" type="varType" />
</xs:schema>
```

LISTING 9: XML Schema of SES Ontology: Part 2

SES Elements	XSD Structure
Entity	<pre> &lt;xs:element name="[entity.name]"&gt;   &lt;xs:complexType&gt;     . . .   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Aspect	<pre> &lt;xs:element name="[aspect.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:sequence&gt;       &lt;xs:element name="[child.entity.name]" /&gt;       . . .     &lt;/xs:sequence&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Multi-Aspect	<pre> &lt;xs:element name="[multiAspect.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:sequence&gt;       &lt;xs:element name="[entity.name]"         minOccurs="[numberComponentsVar.min]"         maxOccurs="[numberComponentsVar.max]"&gt;         . . .       &lt;/xs:element&gt;     &lt;/xs:sequence&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Specialization	<pre> &lt;xs:element name="[spec.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:choice&gt;       &lt;xs:element name="[child.entity.name]" /&gt;       . . .     &lt;/xs:choice&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Variable	<pre> &lt;xs:complexType mixed="true"&gt;   &lt;xs:attribute name="[var.name]" type="xs:[var.rangeSpec]" /&gt; &lt;/xs:complexType&gt; </pre>

TABLE 2.2: Representation of an SES as an XML Schema.

SES Elements	XSD Structure
Entity	<pre> &lt;xs:element name="[entity.name]"&gt;   &lt;xs:complexType&gt;     . . .   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Aspect	<pre> &lt;xs:element name="[entity.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:sequence&gt;       &lt;xs:element name="[child.entity.name]"/&gt;       . . .     &lt;/xs:sequence&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Multi-Aspect	<pre> &lt;xs:element name="[element.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:sequence&gt;       &lt;xs:element name="[entity.name]"/&gt;     &lt;/xs:sequence&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Specialization	<pre> &lt;xs:element name="[entity.name]"&gt;   &lt;xs:complexType&gt;     &lt;xs:choice&gt;       &lt;xs:element name="[child.entity.name]"/&gt;       . . .     &lt;/xs:choice&gt;   &lt;/xs:complexType&gt; &lt;/xs:element&gt; </pre>
Variable	<pre> &lt;xs:complexType mixed="true"&gt;   &lt;xs:attribute name="[var.name]" type="xs:[var.rangeSpec]"/&gt; &lt;/xs:complexType&gt; </pre>

TABLE 2.3: Representation of an SES as an XML Schema (Reduced).

#### 2.4.4 XPath

XPath is a query language for specifying navigation within an XML document ([W3C \[2010\]](#)). It is a necessary stepping stone towards querying and transforming XML documents. It also provides basic facilities for manipulating strings, numbers, and booleans. XPath gets its name from its use of path notation as in URLs for navigating through the hierarchical structure of an XML document.

##### 2.4.4.1 XPath Examples

Listing 10 represents the XML instance of the SES Metamodel presented in figure 2.2. During validation XPath query is used to validate XML document like listing 10 for checking an SES model whether it is conforming to the SES axioms or not. Tables 2.4 shows some example XPath query and its output based on listing 10. Row 1 of the table shows the query which retrieving all the entity elements presents in the document. Row 2 shows the query for retrieving all the aspects of Aircraft entity presents in the document and result shows Engines and Airframe which are aspects of Aircraft. For retrieving available options of Airframe specialization XPath query presented in row 3 is used and Aluminium and Composite is given as output. To find the other available options instead of Aluminium XPath query presented in row 4 is used and the Composite is presented as the result of the query. To check unique child entity name the XPath query presented in row 5 is used and it shows true as output. To find the immediate child of the entity Engines the query presented in row 6 is used and it shows engineMAsp as an output. For listing all the childs of the Engines node the XPath query presented in row 7 is used and it shows engineMAsp and Engine as outputs. Row 8 shows the XPath query to check unique name of the entities of the same level by counting following sibling's similar names. As the output says true its means that there is no two entities of the same name in the same level.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<entity xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  name="Aircraft">
  <aspect name="physicalDec">
    <entity name="Engines">
      <multiAspect name="engineMAsp">
        <entity name="Engine">
        </entity>
      </multiAspect>
    </entity>
    <entity name="Airframe">
      <specialization name="materialSpec">
        <entity name="Aluminium">
        </entity>
        <entity name="Composite">
        </entity>
      </specialization>
    </entity>
  </aspect>
</entity>

```

LISTING 10: XML instance of the SES Metamodel presented in figure 2.2.

Row	XPath Query	Output
1	<code>//entity/@name</code>	Aircraft Engines Engine Airframe Aluminium Composite
2	<code>//aspect/entity/@name</code>	Engines Airframe
3	<code>//specialization/entity/@name</code>	Aluminium Composite
4	<code>//entity[@name="Aluminium"]/ following-sibling::entity/@name</code>	Composite
5	<code>every \$x in .//entity satisfies empty(\$x//*[ @name = \$x/@name])</code>	true
6	<code>//entity[@name="Engines"]/*/@name</code>	engineMAsp
7	<code>//entity[@name="Engines"]/*//@name</code>	engineMAsp Engine
8	<code>every \$x in .//entity satisfies count (*[\$x/@name = \$x/following-sibling:: */@name]) = 0</code>	true

TABLE 2.4: XPath Examples

## Chapter 3

# Modeling Environment: SESEditor

### 3.1 Introduction

SESEditor has been developed as an SES modeling environment. The graphical user interface is designed in such a way that a user can draw graphs on the screen almost as one would on paper. The vertices or elements and edges can be drawn by clicking and dragging the mouse. Elements icons are added in the toolbox for easy access. Also nodes and edges can be easily moved to any position. The drawing panel is synchronized with the bottom left side tree. Element addition in one place either in the white drawing panel or in the left tree, it will be added in both the sections automatically. Variable can be attached with the nodes. And attached variables are shown in the variable table on the right top corner during any node selection from either of the trees. Also constraint can be added to the aspect node to restrict the choices of entities. Constraint is written using XPath query language. Like variable, constraint is also visible on the constraint table in the right side of the editor during aspect node selection. SESEditor allows to save part of the designed model as a module for future use. That saved module can be reused in any project later. Editor also has the facility to validate the created model against SES axioms using predefined XML Schema. Validation result is displayed in the console window and for valid model SES XML and SES Schema is displayed in the bottom right display window. The editor's export and import options increased the share ability of the designed models among researchers. Newly created or opened project name is displayed in the top left corner of the editor.

## 3.2 Graphical User Interface

Figure 3.1 shows the Graphical User Interface (GUI) of SESEditor. GUI is divided into 8 blocks which are indicated in the figure using number boxes. Each block is described below.

### 3.2.1 Block 1: Toolbox

To create an SES model this toolbox is very important. Summary of each tool is described in the table 3.1. There are four tools named Entity, Aspect, Specialization and Multi-Aspect for adding SES elements in the drawing panel and other tools are for purposes like delete, save, undo, redo, zoom in, zoom out and validation of SES model. Also a cursor tool is added to make the selection free from SES elements tool. For example if Entity tool is selected from the toolbox and without using have to free the selection then Cursor tool is used to free the selection.

### 3.2.2 Block 2: Project Window

Figure 3.2 shows the project window of the SESEditor. Project has two subfolders: MainModule and AddedModule. MainModule contains the created or opened project file. Default name is **Main.xml**. AddedModule contains the files which are added from previously saved module file. Figure 3.3 shows that an Events.xml is listed in the AddedModule folder. Events module has been added here from saved module file. This feature is promoting reusing of existing modules. users can save frequently used modules and can add that in any projects.

### 3.2.3 Block 3: Drawing Panel

The graphical user interface is designed in such a way that a user can draw graphs on the screen almost as one would on paper. The vertices or elements and edges are drawn by clicking and dragging the mouse. Element nodes and edges can be easily moved to any position. Figure 3.4 shows a drawing panel with an example metamodel.

### 3.2.4 Block 4: Synchronized Tree Window

Drawn tree in the drawing panel represents in block 4 as JTree format. Figure 3.5 shows an example of JTree which is synchronized with figure 3.4. Any addition or deletion in any place is synchronized.



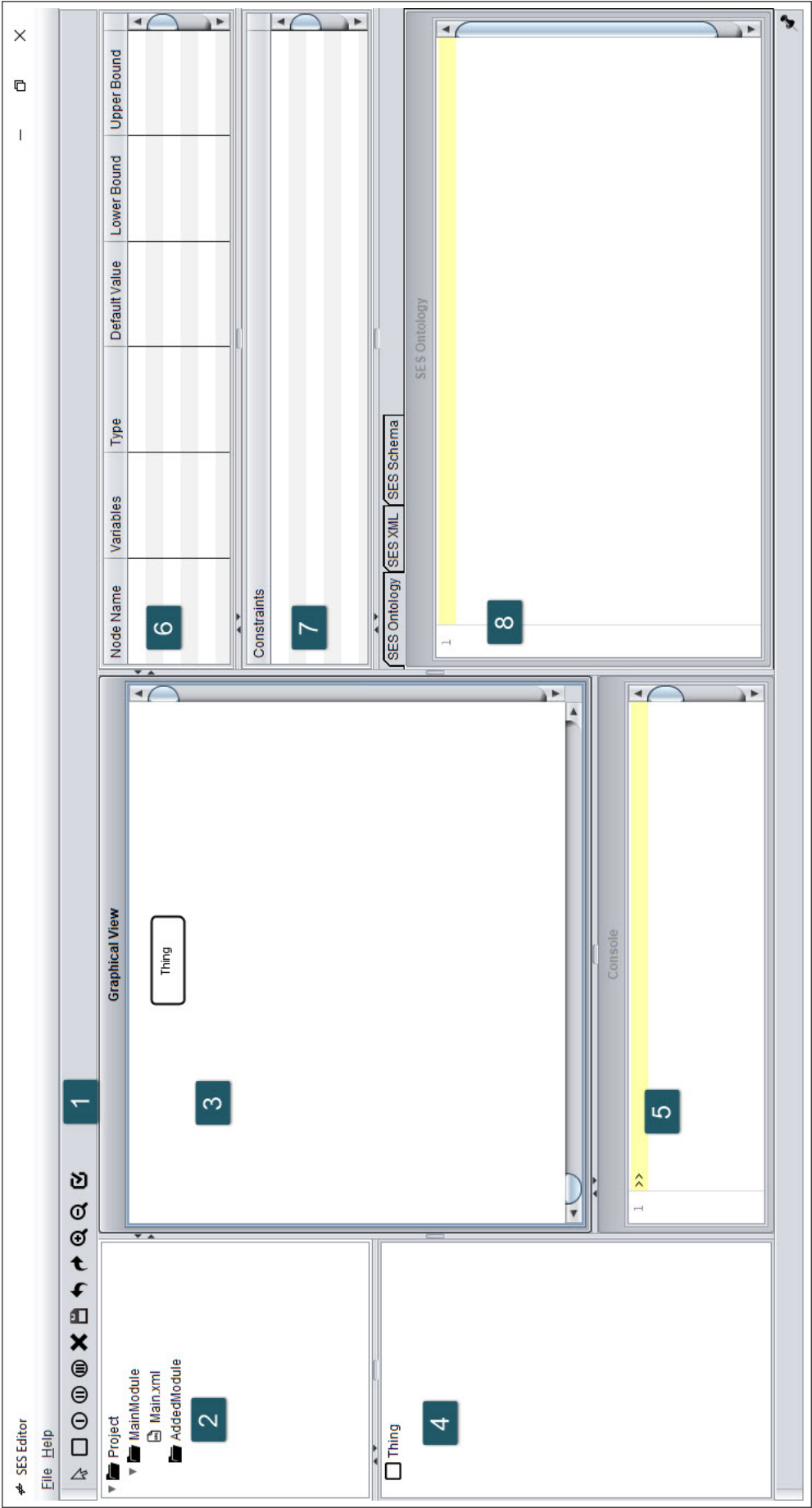


FIGURE 3.1: SESEditor User Interface.

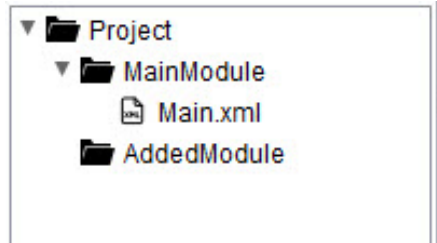


FIGURE 3.2: Project Window.

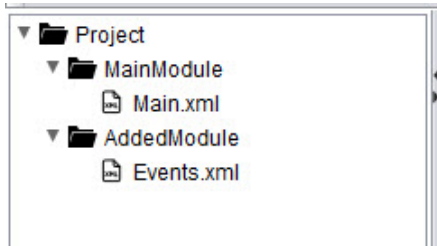


FIGURE 3.3: Project Window With Added Module.

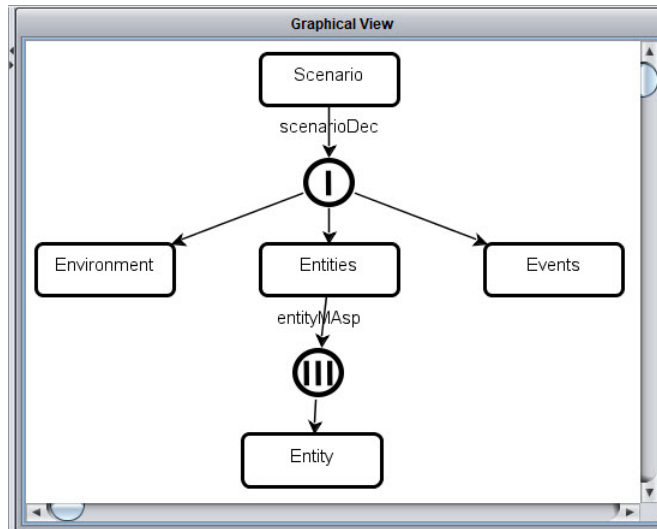


FIGURE 3.4: Drawing Panel With Example.

### 3.2.5 Block 5: Console Window

Console shows the validation result of the SES model. If the developed model doesn't satisfy the SES axioms then in console window that particular axioms information will be displayed. Then user can understand the problem seeing the information. Also new elements can be added from console using some predefined command. An example console window is shown in figure 3.6. The error message is "*cvc-complex-type.2.4.a: Invalid content was found starting with element 'aspect'. One of '{entity}' is expected*". It indicates that SES model contains an aspect node as a child of non-entity (aspect, specialization, multiAspect) element. Aspect node must be a child of an entity node.

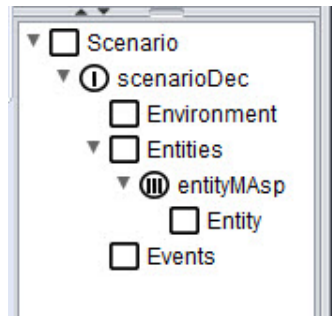


FIGURE 3.5: JTree Representation of Figure 3.4.

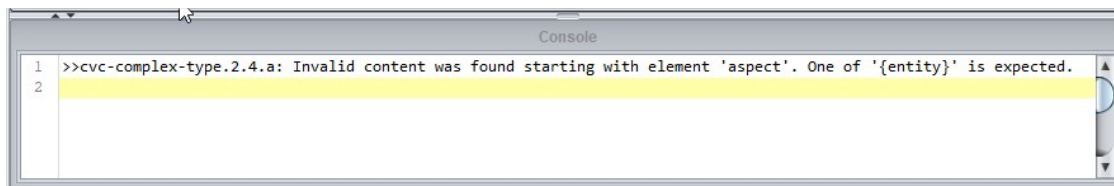


FIGURE 3.6: Console Window.

### 3.2.6 Block 6: Variable Window

Block 6 displays the attached variable of the selected node in the tree. Values of any variable can be changed from this variable table. New variable can be attached to the selected node using this table. An example variable table is shown in figure 3.7.

Node Name	Variables	Type	Default Value	Lower Bound	Upper Bound
Engine	type	string	turbojets		

FIGURE 3.7: Variable Table.

### 3.2.7 Block 7: Constraint Window

Block 7 displays the constraint attached with the selected Aspect node. Constraint query can be edited from this table. Also new constraint can be added from this table to the selected Aspect node. An example constraint table is shown in figure 3.8.

Constraints
if(Name/*[@name='Ford']) then (Country/*[@name='USA']) else true()

FIGURE 3.8: Constraint Table.

### 3.2.8 Block 8: Display Window

Block 8 shown in figure 3.9 consists of three tabs: SES Ontology, SES XML and SES Schema. SES Ontology tab shows the SES ontology schema which is used to validate the created model. SES ontology is presented in Listing 8 and 9 in previous chapter. Here, a portion of the SES Ontology is presented in figure 3.10. SES XML represents the model into XML instance and SES Schema creates the SES schema of the SES model. Figure 3.11 shows an SES XML and figure 3.12 shows an SES schema of the meta model presented in figure 2.2

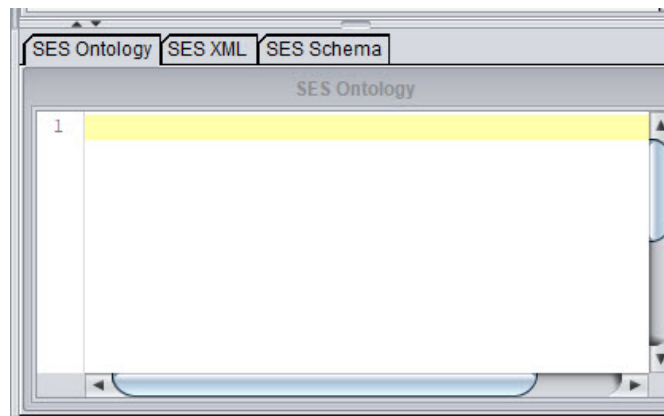


FIGURE 3.9: SES Ontology, XML and Schema Display Window.

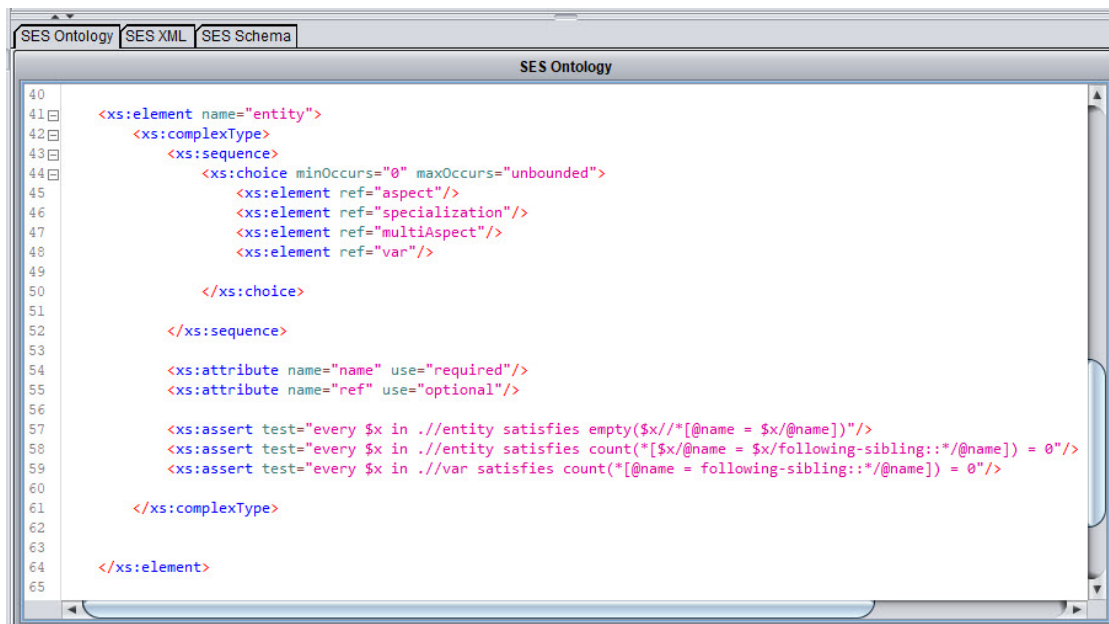


FIGURE 3.10: SES Ontology Tab.

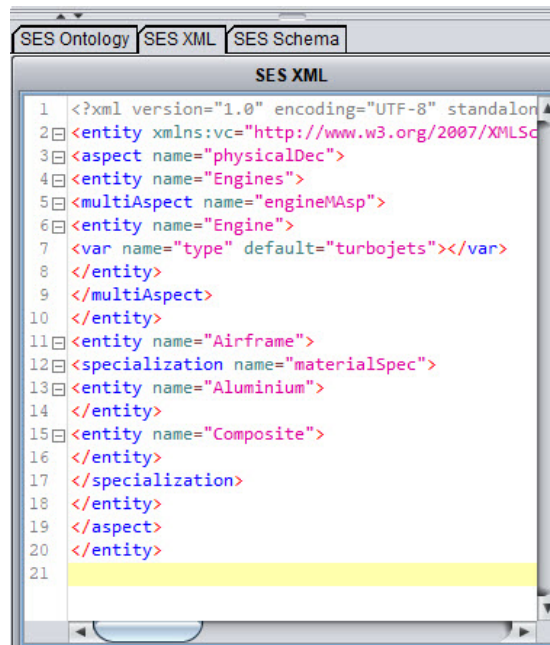


FIGURE 3.11: SES XML Tab.

### 3.2.9 SESEditor Menus

For handling editor's different functionalities various menus are added in SESEditor. From the menu one can create new project, can save created project into disk and also can open existing project from disk. Needed menus are added also for export or import operations. For quick reference of the editor help option is also added. Description of all the available menus are written in Appendix B.1.1.



FIGURE 3.12: SES Schema Tab.













Symbols	Representation	Description
	Cursor	Free the selection of SES entity elements and delete tool
	Entity	Add Entity node in the diagram
	Aspect	Add Aspect node in the diagram
	Specialization	Add Specialization node in the diagram
	Multi-Aspect	Add Multi-Aspect node in the diagram
	Delete	Delete selected node from the diagram
	Save	Save the diagram and variables
	Undo	Cancel or reverse the last command
	Redo	Bring back changes done by undo
	Zoom In	Increases the scale of the diagram
	Zoom Out	Decreases the scale of the diagram
	Validate	Validates created diagram according to SES axioms

TABLE 3.1: ToolBox Description of SESEditor

## 3.3 Functionality

### 3.3.1 Element Addition

There are four types of elements. For each type there is a tool in the toolbox. After selecting any element tool from toolbox have to click on the drawing panel. On the clicked position an element of the selected type will be added. Normally in this way element are named with node prefix with number as suffix. For example node1 for the first Entity addition on the drawing panel. Aspect, Specialization and Multi-Aspect elements ends with Dec, Spec and MAsp suffix respectively. For example node2Dec, node3Spec and node4MAsp will be added in the panel if Aspect, Specialization and Multi-Aspect tools are selected respectively.

Also by right click on any position on the drawing panel elements can be added. Figure 3.13 shows the steps. At first have to right click and then select from the available options. Options are Add Entity, Add Specialization, Add Aspect and Add MultiAspect. Then a New Node pop box will come and writing the name have to select OK to add the element. Based on selection Dec, Spec or MAsp will be added as suffix automatically if not added during writing the node name.

Another way of adding element nodes in the drawing panel is selecting already added node from left tree panel and right click on it. Figure 3.14 shows the steps. At first have to right click and then select Add Node option. Then a New Node pop window will come and writing the name of the element name have to select OK. Then the node will be automatically added as a child of the selected node in the drawing panel. In this way Aspect, Specialization and MultiAspect node must ends with suffix Dec, Spec and MAsp respectively. This option is kept intentionally in this way to make user familiar with the naming convention of the elements.

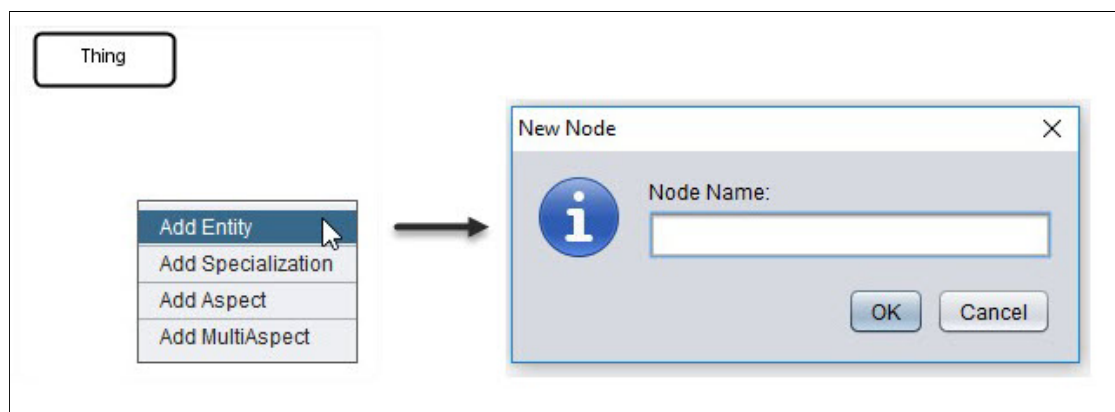


FIGURE 3.13: Element Addition from Drawing Panel.



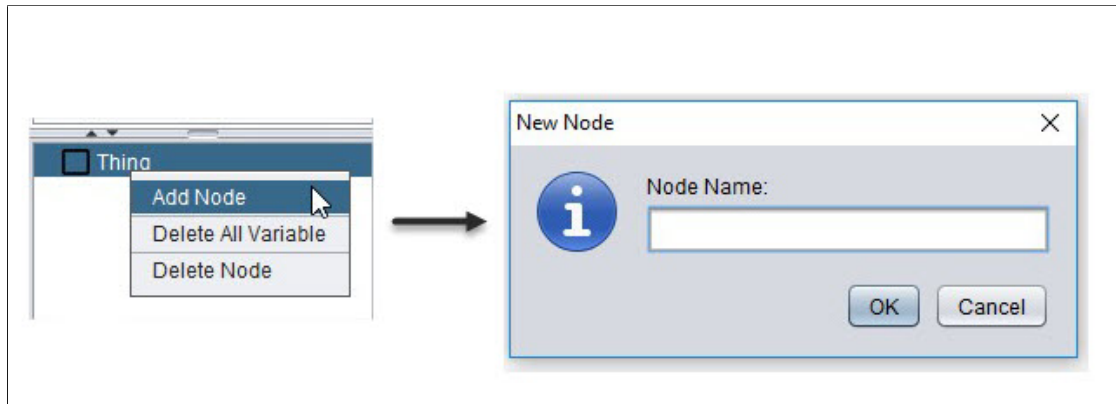


FIGURE 3.14: Element Addition From Left Tree Panel.

### 3.3.2 Element Deletion

To delete an element from the drawing panel have to select that node and right click on it. Then by selecting Delete option like figure 3.15 that node can be deleted. If that node has child element then all the child element will also be deleted. Also by selecting Delete tool from the toolbox this deletion operation can be done in the same way.

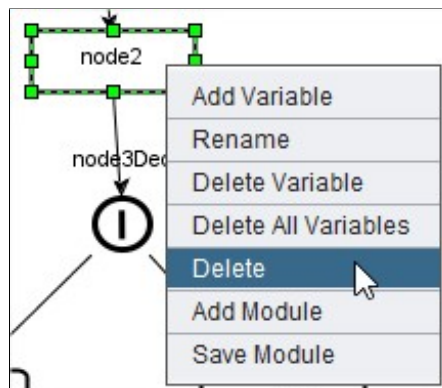


FIGURE 3.15: Element Deletion From Drawing Panel.

### 3.3.3 Element Renaming

To rename an element from the drawing panel have to select that node and right click on it. After that have to select the Rename option like figure 3.16. Then a Rename Node window will pop up and have to write the new name there. After pressing OK the selected node name can be changed to the newly written name.

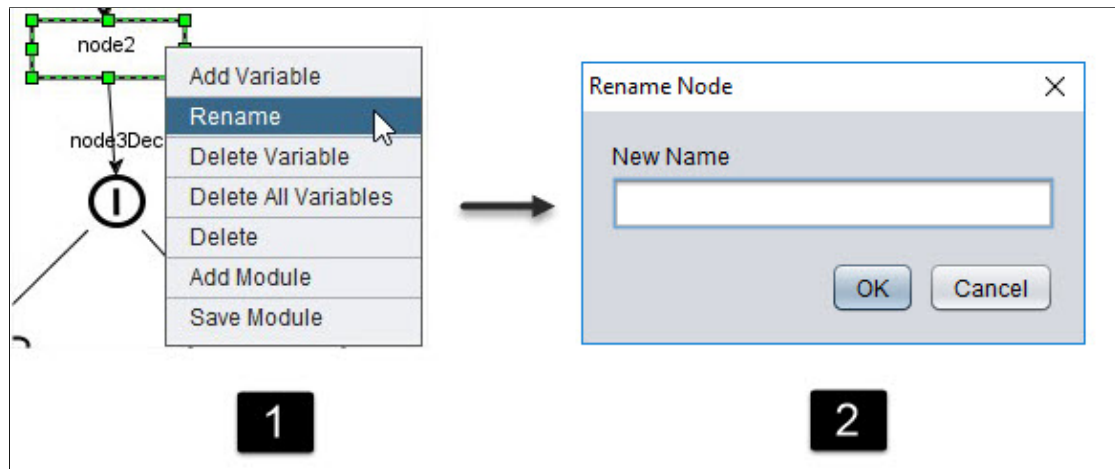


FIGURE 3.16: Element Renaming From Drawing Panel.

### 3.3.4 Variable Addition

To add variable to the selected node have to right click on it and select Add Variable then a window will pop up like figure 3.17. By writing variable name and selecting its type and optional values have to press OK to finish variable addition.

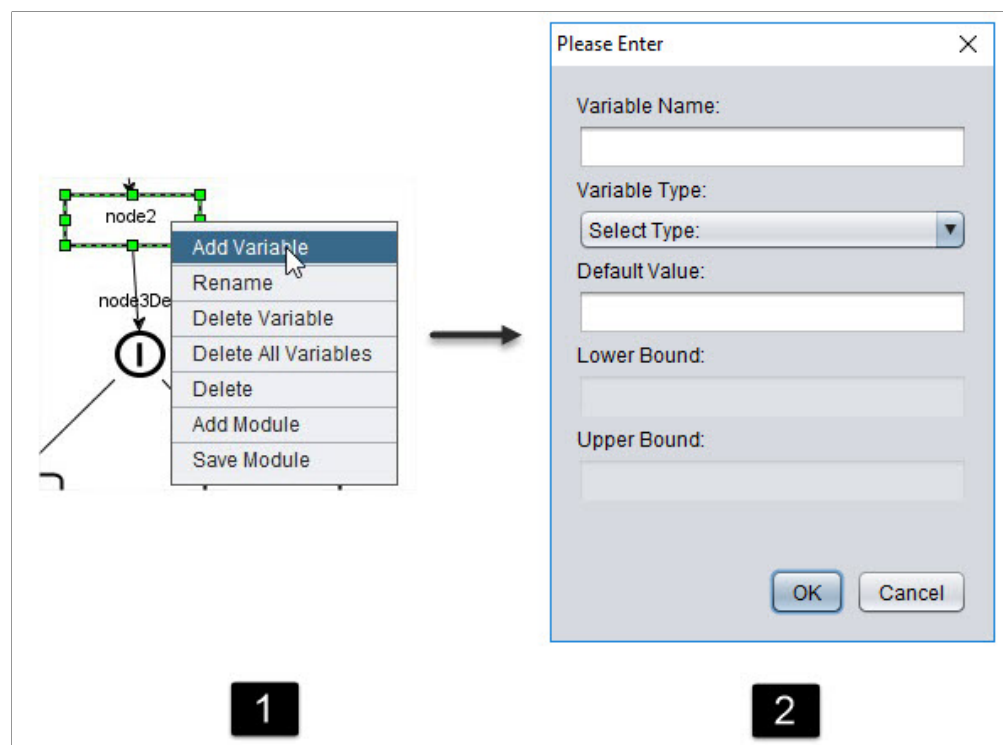


FIGURE 3.17: Variable Addition.

### 3.3.5 Variable Deletion

To delete variable from the selected node have to right click on it and select Delete Variable. Then a window will pop up like figure 3.18 with available variables and have to select the variable for deletion. To delete all the variables after right clicking on the selected node have to select Delete All Variables.

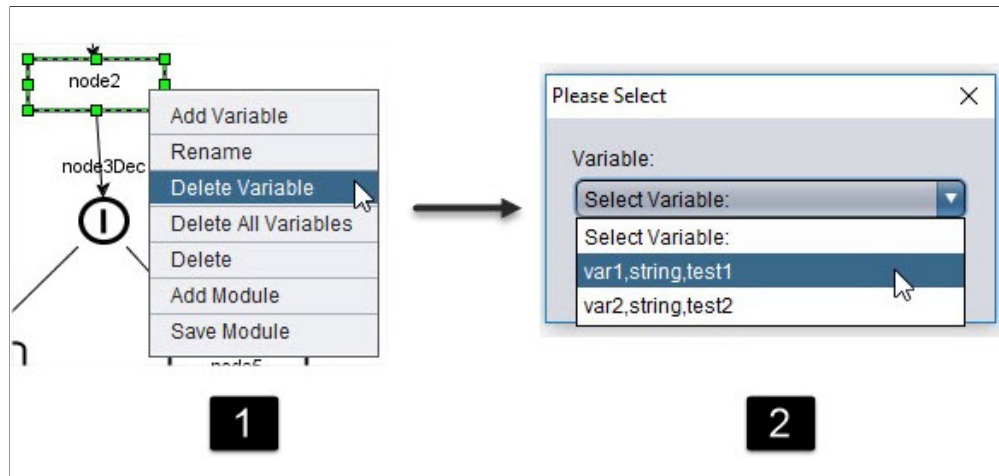


FIGURE 3.18: Variable Addition.

### 3.3.6 Constraint Addition

To add constraint to the selected aspect node have to right click on it and select Add Constraint. A window will pop up like figure 3.19. By writing query using XPath, a query language for selecting nodes from an XML document which is described in section 2.4.4, have to press OK to finish constraint addition. This query will be used to check the validity of the added nodes in the SES model. To modify added constraint have to double click on the row of the constraint shown in the constraint table then a pop up window will come like figure 3.20. Then after rewriting the constraint have to press OK to finish the modification of the constraint.

### 3.3.7 Constraint Deletion

To delete all the constraint added to the selected node just have to right click on it and select Delete All Constraint like shown in figure 3.21.

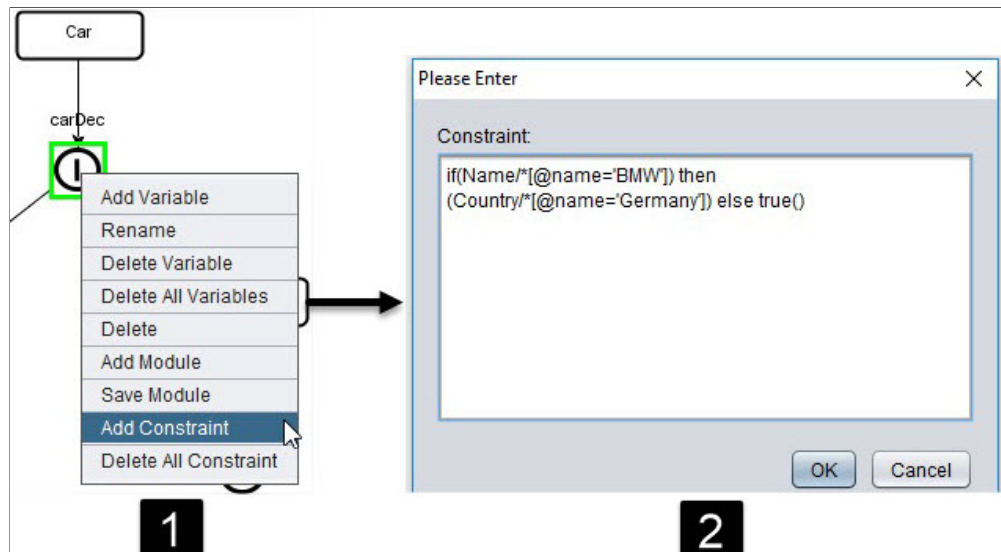


FIGURE 3.19: Constraint Addition.

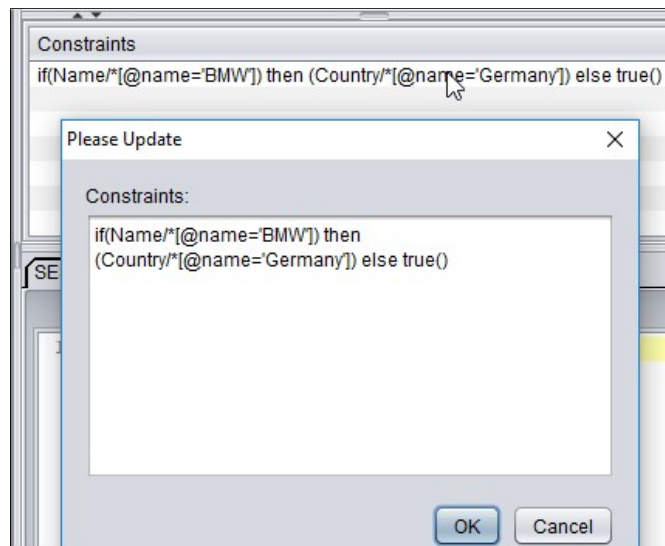


FIGURE 3.20: Constraint Modification.

### 3.3.8 Module Saving

Module is a small part of a whole model. Some parts are very common and used frequently during designing. In that case a small part can be saved as a module for future use. The module is saved as an XML file. To save a small part as a module have to select the starting node of that module like the figure shown in 3.22 and have to right click on it. Then selecting Save Module it can be saved.

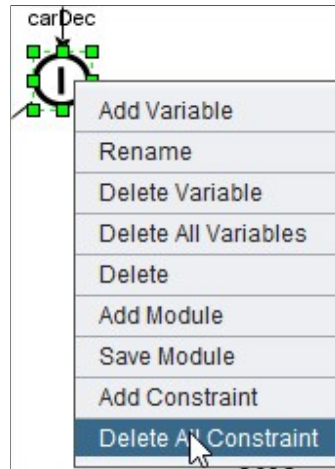


FIGURE 3.21: Constraint Deletion.

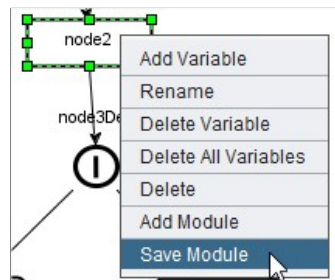


FIGURE 3.22: Module Saving.

### 3.3.9 Module Addition

During designing its not a good idea to draw frequently used small parts or module of a whole model again and again from scratch. In that case the previously saved module are useful. To add a saved module have to select the desired node like the figure shown in 3.23 and have to right click on it. Then have to select Add Module to browse the saved module in file system. Then selecting the saved module have to open it. Then the saved module will be automatically added to the selected node.

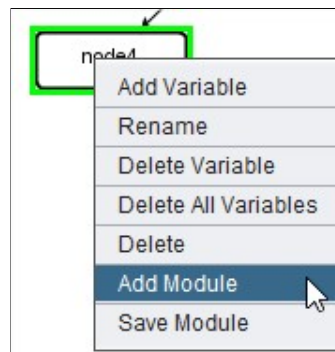


FIGURE 3.23: Module Addition.

### 3.3.10 Validation

Validation is checked by pressing the validation tool from the toolbox present in the block 1 of the SESEditor GUI. If there is any error then it can be seen from the console window. Possible errors and its explanations are given below:

(i) **Error 1:**

cvc-complex-type.2.4.a: Invalid content was found starting with element 'entity'. One of '{ aspect, specialization, multiAspect, var}' is expected.

**Explanation:**

Error 1 indicates SES model contains entity elements as a child of another entity element. It is not allowed. Child of any entity element must be aspect, specialization or multiAspect. 'Var' indicates entity element may contain variables.

(ii) **Error 2:**

cvc-complex-type.2.4.a: Invalid content was found starting with element 'aspect'. One of '{entity}' is expected.

**Explanation:**

Error 2 indicates SES model contains an aspect node as a child of non-entity (aspect, specialization, multiAspect) element. Aspect node must be a child of an entity node.

(iii) **Error 3:**

cvc-complex-type.2.4.a: Invalid content was found starting with element 'specialization'. One of '{entity}' is expected.

**Explanation:**

Error 3 indicates SES model contains an specialization node as a child of non-entity (aspect, specialization, multiAspect) element. Specialization node must be a child of an entity node.

(iv) **Error 4:**

cvc-complex-type.2.4.a: Invalid content was found starting with element 'multiAspect'. One of '{entity}' is expected.

**Explanation:**

Error 4 indicates SES model contains an multiAspect node as a child of non-entity (aspect, specialization, multiAspect) element. MultiAspect node must be a child of an entity node.

(v) **Error 5:**

```
cvc-assertion.3.13.4.1: Assertion evaluation ('every $x in .//entity
satisfies count(*[$x/@name = $x/following-sibling::*/@name]) = 0')
for element 'entity' with type '#anonymous' did not succeed.
```

**Explanation:**

Error 5 occurs if two nodes of same label are added in the same level. It violates the valid brother SES axiom. In same level node names should be unique.

**(vi) Error 6:**

```
cvc-assertion.3.13.4.1: Assertion evaluation ('every $x in .//entity
satisfies empty($x/*[@name = $x/@name])') for element 'entity' with
type '#anonymous' did not succeed.
```

**Explanation:**

Error 6 occurs if SES model violates strict hierarchy SES axiom that is if a label appear more than once down any path of the tree. The node names should be unique in any path of the tree.

**(vii) Error 7:**

```
cvc-assertion.3.13.4.1: Assertion evaluation ('every $x in .//var
satisfies count(*[@name = following-sibling::*/@name]) = 0') for
element 'entity' with type '#anonymous' did not succeed.
```

**Explanation:**

Error 7 occurs if SES model violates attached variables SES axiom. If there are more than one variable attached to any node then the variable names should be unique.

### 3.3.11 Generating SES XML and SES Schema

After validation, generating SES XML and SES schema is just a click of a button. Generated XML or schema will be presented in the display window of the SESEditor.

## 3.4 Package Design and Class Static Structure

### 3.4.1 Common Packages

Figure 3.24 shows how packages are nested for the common use. Packages common for both the editors are:

- dlr.xml.schema
- dlr.ses.core
- dlr.resources.images
- dlr.resources.docs

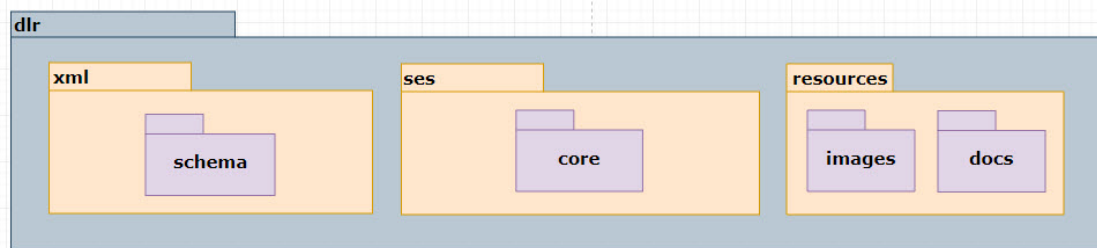


FIGURE 3.24: Common Package Structure.

### 3.4.2 SESEditor Packages

Figure 3.25 shows how packages are nested in the SESEditor. All the classes which are only related to SESEditor are placed in the **seseditor** package. SESEditor packages are:

- dlr.xml.schema
- dlr.ses.core
- dlr.ses.seseditor
- dlr.resources.images
- dlr.resources.docs

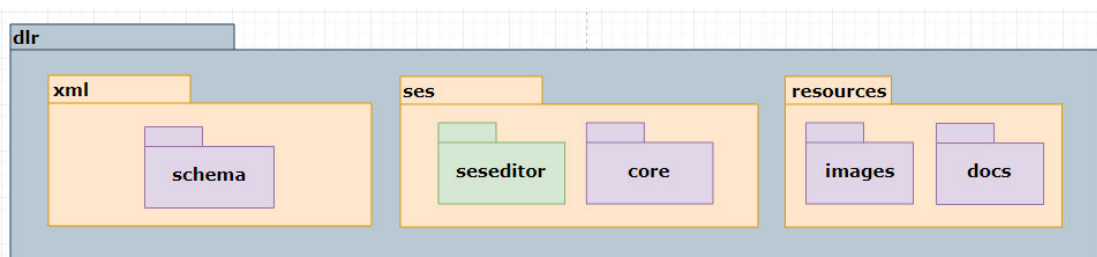


FIGURE 3.25: SESEditor Package Structure.

### 3.4.3 Package Design

Figure 3.26 shows the relation between all the packages used in the SESEditor.



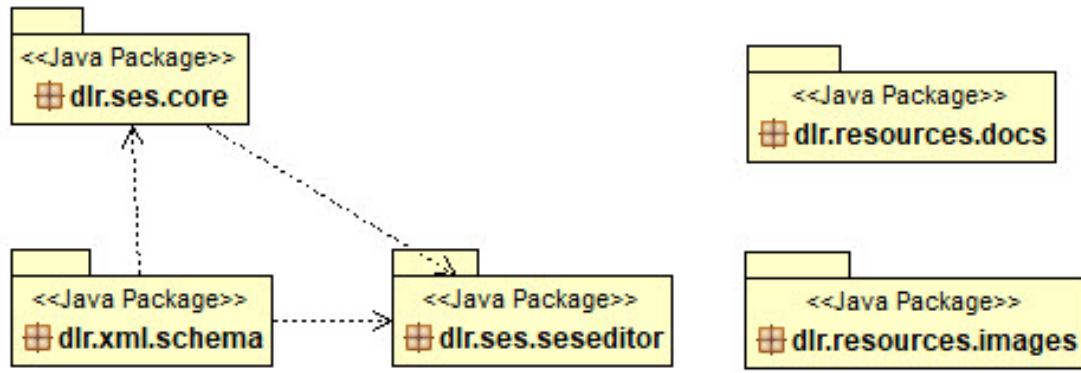


FIGURE 3.26: SESEditor Package Design.

#### 3.4.4 Class Diagram: core

Figure 3.27 shows the class diagram of SESEditor core classes. Description of all the classes are added in Appendix A.

#### 3.4.5 Class Diagram: seseditor

Figure 3.28 shows the class diagram of SESEditor seseditor classes. Description of all the classes are added in Appendix A.

### 3.5 Used Technologies

#### 3.5.1 Java

For developing SESEditor as a desktop application Java SE has been used. Before choosing Java SE several other programming languages came into consideration for the implementation. The following languages came up as candidates: Java, C# and C++.

**Java** is a predominantly static typed programming language and developed by Gosling [1995]. It also supports dynamic typing for some Object-oriented programming concepts like polymorphism. Though it is influenced by C and C++, by removing pointer it is unique and easy to use programming language. It dropped the headache of managing memory as it brought in automatic memory management with the introduction of Garbage Collection. Another feature that made Java to stand out of its peers was that it favoured WORA philosophy (i.e. Write Once Run Anywhere) with the introduction of Java Virtual Machine. Java has document parsing libraries with XSD 1.1 support. For creating desktop application Java has AWT, Swing, SWT, JavaFX and

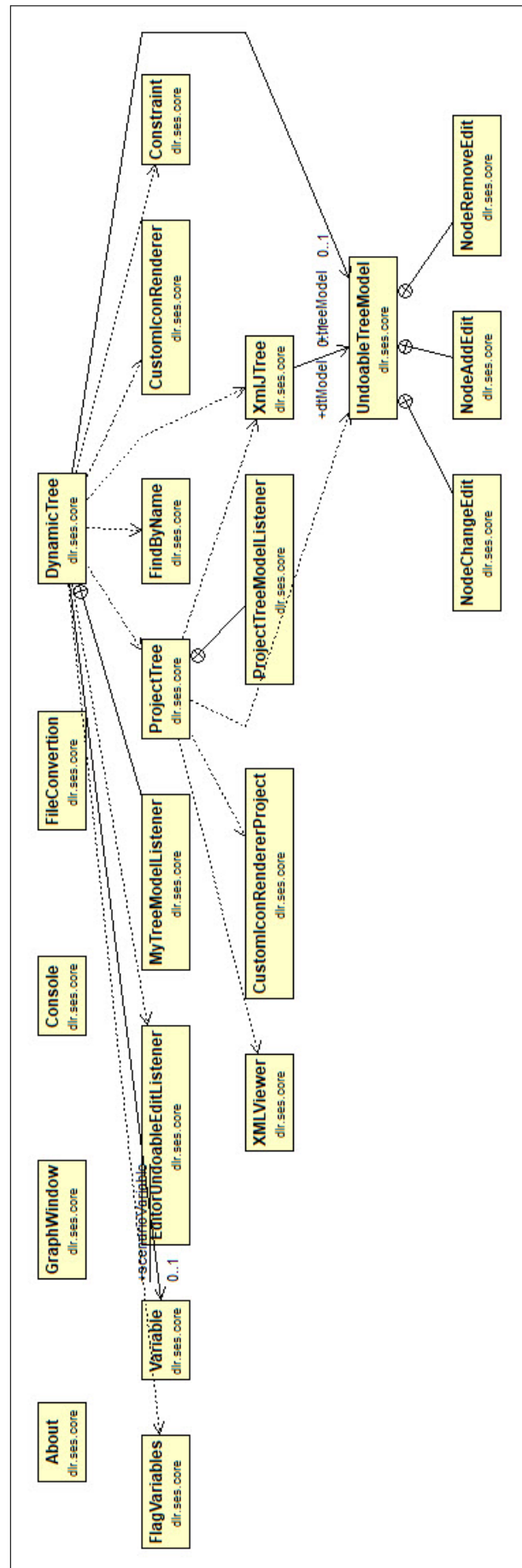


FIGURE 3.27: SESEditor Class Diagram: core package.

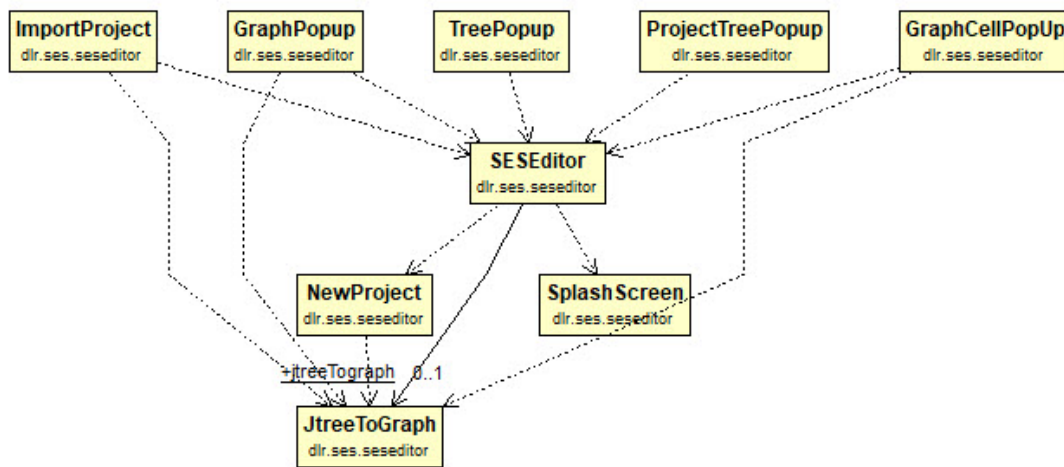


FIGURE 3.28: SESEditor Class Diagram: seseditor package.

many more frameworks. Based on the following points Swing seems more perfect for the development of our editor:

- More dynamic components - through the use of delegation objects
- More complex components - JTree, JTable, JFileChooser
- New containment model for root containers - JFrame, JDialog, and JApplet use a containment
- Model to maintain their component hierarchy
- A new BoxLayout manager
- Pluggable look and feel

For creating desktop application there was also **C#** which was introduced to the world by [Microsoft](#) [2000]. C# has UWP, WPF, WinForms etc frameworks for desktop application. But C# doesn't have any library support for XSD 1.1 and it mostly works on windows operating system.

**C++** is developed by [Stroustrup](#) [1985]. It is designed following the object oriented modular approach by defining classes, structs, objects, methods and interfaces. C++ also has free libraries for document parsing but doesn't support XSD 1.1. For creating desktop application C++ has QT, GTK+, WxWidgets etc frameworks.

The decision of which one of these programming language to use is based on the following factors:

- (i) open source technology
- (ii) platform independent
- (iii) available libraries for various purposes
- (iv) well documented manual and community support
- (v) nice graphical user interface component support

Since C++ doesn't have any free library which support XSD 1.1, it is left from the consideration list. C# was really powerful option. But unfortunately C# is not platform independent. C# is good only for windows operating system. Finally Java with Swing framework has been selected as our programming language of implementation because it has our desired required library which has support of XML Schema 1.0 and 1.1 processor.

### 3.5.2 Xerces

For validating XML document against XML Schema Xerces2 Java Parser has been used. Before choosing Xerces2 Java Parser several options were taken into consideration. The following toolkits came up as candidates: Xerces2 Java Parser, Xerces-C++ and Saxon.

**Xerces** is Apache's collection of software libraries for parsing, validating, serializing and manipulating XML. It is a set of parsers compatible with Extensible Markup Language (XML). A parser is a program that analyzes and organizes formal language statements into a usable form for a given purpose. Xerces parsers are available for Java and C++, implementing World Wide Web Consortium (W3C) XML, Document Object Model (DOM), and Simple API for XML (SAX) standards.

**Xerces2** is the next generation of high performance, fully compliant XML parsers in the [Apache Software Foundation \[2010\]](#) Xerces family. This new version of Xerces introduces the Xerces Native Interface (XNI), a complete framework for building parser components and configurations that is extremely modular and easy to program. Xerces2 is a fully conforming XML Schema 1.0 and 1.1 processor. Xerces2 is able to parse documents written according to the XML 1.1 Recommendation, except that it does not yet provide an option to enable normalization checking. It also handles namespaces according to the XML Namespaces 1.1 Recommendation, and will correctly serialize XML 1.1 documents if the DOM level 3 load/save APIs are in use.

**Xerces-C++** is a validating XML parser written in a portable subset of C++ and maintain by [Apache Software Foundation \[2008\]](#). Xerces-C++ makes it easy to give your application the ability to read and write XML data. A shared library is provided

for parsing, generating, manipulating, and validating XML documents using the DOM, SAX, and SAX2 APIs. For an introduction to programming with Xerces-C++ refer to the Programming Guide.

**Saxon 9.9** includes highly conformant implementations of the current W3C Recommendations: XSLT 3.0, XQuery 3.1, XPath 3.1, and XSD 1.1. It includes a complete implementation of XML Schema 1.1. This provides the ability to process schema documents that use the new features of XSD 1.1, and use them to validate instance documents. It is developed by [Kay \[2008\]](#).

The decision of which one of these library to use is based on the following factors:

- (i) XSD 1.1 support
- (ii) easy integration of the library in order to work with Java Swing
- (iii) free of cost
- (iv) well documented manual and community support

Since Xerces-C++ doesn't support XSD 1.1, it is left from the consideration list. Saxon was really powerful library. But unfortunately saxon HE doesn't support XSD 1.1. Saxon EE supports XSD 1.1 but it is not free. Finally Xerces2 Java Parser has been selected as the library is free and it is a fully conforming XML Schema 1.0 and 1.1 processor.

### 3.5.3 JGraphX

For developing SESEditor drawign panel for model visualization JGraphX has been used. Before choosing JGraphX several graphical toolkits came into consideration for the visualizing of the process. For licensing reasons only open-source toolkits were considered. The following toolkits came up as candidates: JGraph, JUNG, Prefuse and Piccolo.

**JGraphX** is known as the most powerful, easy-to-use, feature-rich and standards-compliant open source graph component available for Java and it is developed by [Alder \[2007\]](#). This toolkit qualified itself by an easy installation, its small size and a good documentation. JGraphX contains a lot of features, like a mathematical grid to draw on, predefined edge routing algorithms, predefined adjustable edge labelling algorithms, and a good collaboration with Java Swing. Programming with JGraphX requires an advanced level of knowledge in Java, but knowledge in Swing is sufficient to use JGraphX

without problems. Unfortunately, a bigger minus is the look-and-feel towards the objects which are created with JGraphX, since they give away a very static look. Also, keyboard integration is not at expected level. These points what made it hard to decide whether JGraphX would be fit for this development.

**JUNG** (short for 'Java Universal Network/Graph Framework') is an open source software library that provides a common and extendable language for the modelling, analysis, and visualization of data that can be represented as a graph or network and it is maintaining by [Joshua O'Madadhain \[2010\]](#). Providing enough demonstrations JUNG offers many variations in tree layouts as well as demos for zoom, panning, drag, drop, and even tree node collapsing and expanding. This toolkit also has the possibility of adding images, labelling edges, and using matrix operations on embedded objects. But the installation of JUNG and its necessary components were problematic and time consuming. Programming with JUNG requires a more advanced level of architectural knowledge than JGraphX.

**Prefuse** is a set of software tools for creating interactive data visualizations and supports a rich set of features for data modelling, visualization, and interaction and it is developed by [Heer \[2011\]](#). The most remarkable features of Prefuse are its great animation support as well as the pre-embedded input for data types like .txt, .xml and other. Tree structures, even more complex layouts for trees like radial, organic and balloon trees, are available, as well as a theoretically usable grid/gridlayout. Unfortunately, a lot of parts in the documentation of Prefuse are missing. Programming with Prefuse is on about the same level as with JUNG, but Prefuse contains very few demos which are too few to help understanding the complex features of Prefuse. Another drawback is that Prefuse community is not strong and it will be difficult to come out from any bug during development.

**Piccolo** is a toolkit that supports the development of 2D structured graphics programs and it is developed by [Bederson et al. \[2004\]](#). Since Piccolo was originally created to have zooming and panning embedded in the framework from the start, the Java source code for those are no more than none to two lines of code. Drawing itself is not problematic as well, because the introduction into the material of Piccolo does not take much time to learn for implementation. A grid is available, as well as translation and rotation for objects. Piccolo does not work well with Swing (Bugs appeared while testing). Furthermore, since Piccolo is missing the possibility of drawing shapes other than a line, rectangles and circles, the programming of other shapes only takes extra time.

The decision of which one of these toolkits to use is based on the following factors:

- (i) simplicity of creating a tree-like structure

- (ii) easy compatibility of the toolkit in order to work with Java Swing
- (iii) avoidance of additional efforts to be put in when programming zoom, collapsing/-expanding, insertion of images, but also trivial things like writing labels on edges or drawing arrows.
- (iv) well documented manual and community support

Since the possibility of drawing tree with arrows and getting a built in functions to connect the nodes is easy in JGraphX, finally JGraphX has been selected as the library for the development of the editor.

#### 3.5.4 RSyntaxTextArea

RSyntaxTextArea is a fast and efficient syntax highlighting and code folding text component for Java Swing and it is developed by [Futrell \[2008\]](#). It extends JTextComponent and that's why it integrates completely with the standard javax.swing.text package. RSyntaxTextArea has been used in SESEditor and PESEditor for displaying generated XML and XML Schema with syntax highlighting and code folding options.

#### 3.5.5 Goava

Guava is an open-source set of common libraries for Java, mainly developed by [Google \[2011\]](#) engineers. Google Guava can be roughly divided into three components: basic utilities to reduce menial labors to implement common methods and behaviors; an extension to the Java collections framework (JCF) formerly called the Google Collections Library; and other utilities which provide convenient and productive features such as functional programming, graphs, caching, range objects, and hashing. The creation and architecture of the collection component were partly motivated by generics introduced in JDK 1.5. Although generics improve the productivity of programmers, the standard JCF does not provide sufficient functionality, and its complement Apache Commons Collections had not adopted generics in order to maintain backward compatibility. This fact led two engineers Kevin Bourrillion and Jared Levy to develop an extension to JCF, which provides additional generic classes such as multisets, multimaps, bimap, and immutable collections.

## Chapter 4

# Interactive Pruning Tool: PESEditor

### 4.1 Introduction

PESEditor has been developed as an interactive pruning tool. The GUI of the PESEditor is almost same as SESEditor. PESEditor also has variable table for displaying variables or editing values of variables. Also constraint table and console window work exactly same way. Here also left side tree is synchronized with the white drawing panel. Nodes are also movable here. But like SESEditor, here new project can't be created. Only models created in SESEditor can be opened here as a project. Also new elements can't be added on the model in PESEditor. Then the main functionality of PESEditor is pruning SES model to create pruned entity structure. After completing pruning of created SES metamodel, executable scenario can be exported for target simulation environment by using project specific XSLT file.

### 4.2 Graphical User Interface

Figure 4.1 shows the Graphical User Interface (GUI) of PESEditor. GUI is divided into 8 blocks which are indicated in the figure using number boxes. Each block is described below.



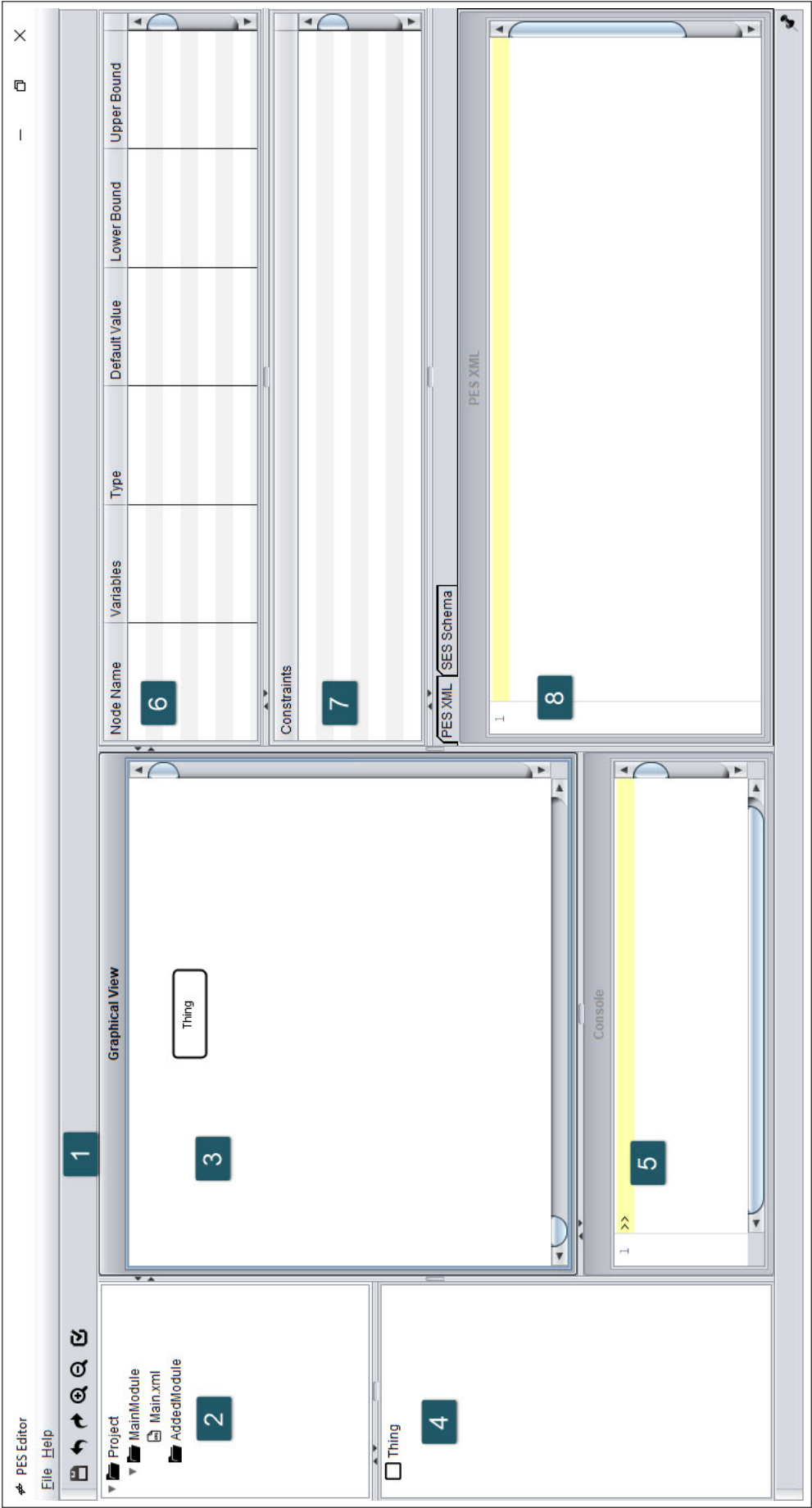


FIGURE 4.1: PESEditor User Interface.

### 4.2.1 Block 1: Toolbox

There are six tools named save, undo, redo, zoom in, zoom out and validate. Summary of each tool is described in the table 4.1. Validation errors and explanations are discussed in section 3.3.10.







Symbols	Representation	Description
	Save	Save the diagram and variables
	Undo	Cancel or reverse the last command
	Redo	Bring back changes done by undo
	Zoom In	Increases the scale of the diagram
	Zoom Out	Decreases the scale of the diagram
	Validate	Validates created diagram according to SES axioms

TABLE 4.1: ToolBox Description of PESEditor

### 4.2.2 Block 2: Project Window

Figure 4.2 shows the project window of the PESEditor. Project has two subfolders: MainModule and AddedModule. MainModule contains the opened project file. Default name is **Main.xml**. AddedModule contains the files which are added from saved module file. Figure 4.3 shows that an Events.xml is listed in the AddedModule folder. Events module has been added here from saved module file. In PESEditor there is no option of creating new project and that's why adding of new module is not possible. It will just show the module name in the project window if exist in the opened project.

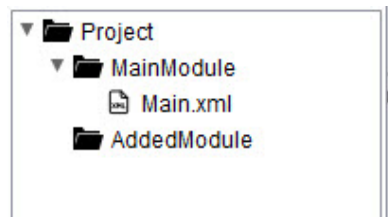


FIGURE 4.2: Project Window.

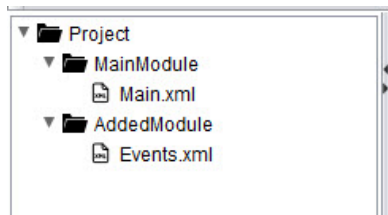


FIGURE 4.3: Project Window With Added Module.

### 4.2.3 Block 3: Drawing Panel

The graphical user interface is designed in such a way that a user can drag the elements on the panel using mouse. Also nodes and edges can be easily moved to any position. Figure 4.4 shows a panel with an example metamodel. Green icon means that node needs to be pruned.

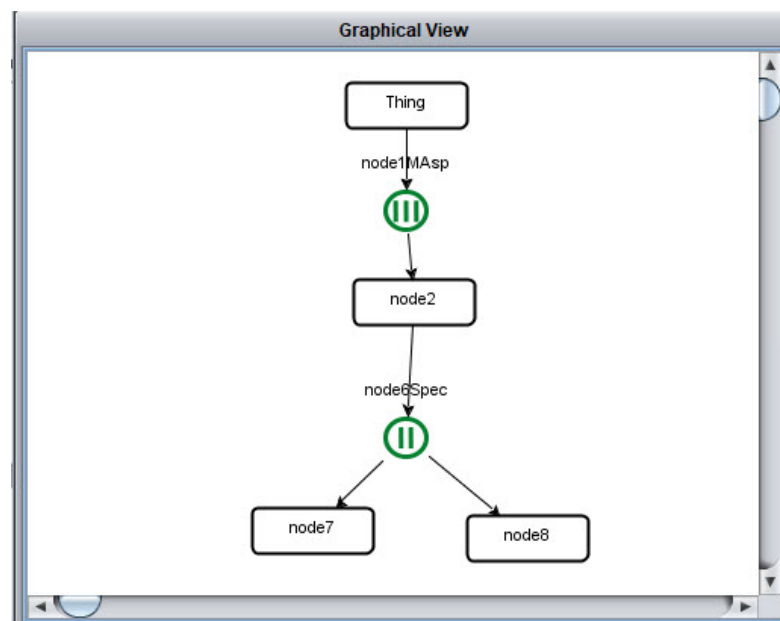


FIGURE 4.4: Drawing Panel Example Before Pruning.

#### 4.2.4 Block 4: Synchronized Tree Window

Opened tree in the drawing panel represents in block 4 as JTree format. Figure 4.5 shows an example of JTree which is synchronized with figure 4.4. During pruning this left tree will also be synchronized.

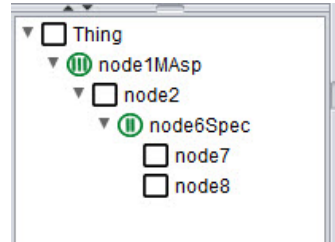


FIGURE 4.5: JTree Representation of Figure 4.4.

#### 4.2.5 Block 5: Console Window

Console window of PESEditor is almost same as the console window of SESEditor which is described in section 3.2.5. It also shows the validation results of the created model.

#### 4.2.6 Block 6: Variable Window

Variable window also provides functionalities same as the variable window of SESEditor which is described in section 3.2.6. Here it is very important because user assign values here for the node variables.

#### 4.2.7 Block 7: Constraint Window

Here constraint window is just to display the added constraint in the model node. It is also designed in the way in which the constraint window of SESEditor described in section 3.2.7 is designed.

#### 4.2.8 Block 8: Display Window

Block 8 shown in figure 4.6 consists of two tabs: PES XML and SES Schema. PES XML represents the pruned model into XML instance and SES Schema creates the SES schema of the PES model. Figure 4.7 shows an PES XML and figure 4.8 shows an SES schema of the meta model presented in figure 2.2

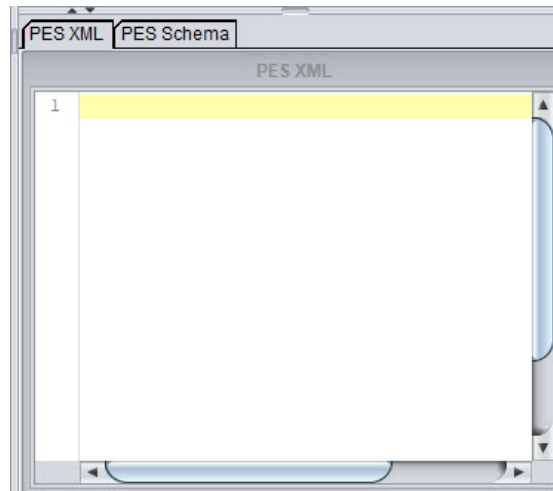


FIGURE 4.6: PES XML and SES Schema Display Window



FIGURE 4.7: PES XML Tab.

#### 4.2.9 PESEditor Menus

For handling editor's different functionalities various menus are added in PESditor. From the menu one can open existing project from disk and can save opened project into disk. For XSLT transformation Export sub menu is also added under the file menu. For quick reference of the editor help option is also added. Description of all the available menus are written in Appendix [B.2.1](#).



```
1 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
3   xmlns:vc="http://www.w3.org/2007/XMLSchema-versioning"
4   elementFormDefault="qualified" vc:minVersion="1.1" >
5
6 <xs:element name="Aircraft">
7   <xs:complexType>
8     <xs:sequence id="physicalDec">
9       <xs:element name="Engines">
10        <xs:complexType>
11          <xs:sequence id="engineMAp">
12            <xs:element name="Engine" minOccurs="0" maxOccurs="unbounded">
13              <xs:complexType>
14                <xs:attribute name="type" default="turbojets">
15                </xs:attribute>
16                <xs:attribute name="name" use="optional"/>
17              </xs:complexType>
18            </xs:element>
19          </xs:sequence>
20          <xs:attribute name="name" use="optional"/>
21        </xs:complexType>
22      </xs:element>
23    <xs:element name="Airframe">
24      <xs:complexType>
25        <xs:choice id="materialSpec">
26          <xs:element name="Aluminium">
27            <xs:complexType>
28              <xs:attribute name="name" use="optional"/>
29            </xs:complexType>
30          </xs:element>
31          <xs:element name="Composite">
32            <xs:complexType>
33              <xs:attribute name="name" use="optional"/>
34            </xs:complexType>
35          </xs:element>
36        </xs:choice>
37        <xs:attribute name="name" use="optional"/>
38      </xs:complexType>
39    </xs:element>
40  </xs:sequence>
41  <xs:attribute name="name" use="optional"/>
42 </xs:complexType>
43 </xs:element>
44 </xs:schema>
45
```

FIGURE 4.8: PES Schema Tab.

## 4.3 Functionality

### 4.3.1 Interactive Pruning

The main functionality of the PESEditor is the interactive pruning system. Using mouse and graphical user interface of the editor it is very easy to choose particular subject from several aspect nodes for several decomposition of the system. From available options of specialization node user can choose specific entity node by clicking on the panel. Also by setting cardinality number user can generate specific number of aspects of the same type from the multi-aspect node easily using mouse click. User can assign the value of node variables by selecting the node variable from the panel and putting in the value field with ease. Section 4.4 describes various pruning procedures in detail.

### 4.3.2 Validation

User can validate their model, as the same way like SESEditor which is described in section 3.3.10, here in PESEditor also.

### 4.3.3 Generating PES XML and PES Schema

After pruning and validation, generating PES XML and PES schema is just a click of a button. Generated XML or schema will be presented in the display window of the PESEditor.

### 4.3.4 XSLT Transformation

The another important functionality of the PESEditor is XSLT (Extensible Stylesheet Language Transformations) transformation capability. XSLT is a language for transforming XML documents into other XML documents, or other formats. Section 5.2.3.2 listing 14 shows an example of XSLT transformation. On that example, from the pruned entity structure executable scenario of the traffic server is generated using XSLT.

## 4.4 Pruning Procedures Implementation in PESEditor

### 4.4.1 Aspect Node Pruning

Figure 4.9 shows the pruning of Aspect node. Aircraft consist of an entity Engines and an entity Airframe. Here, the derived Pruned Entity Structure (PES) is identical to SES. Aspect node don't need pruning.

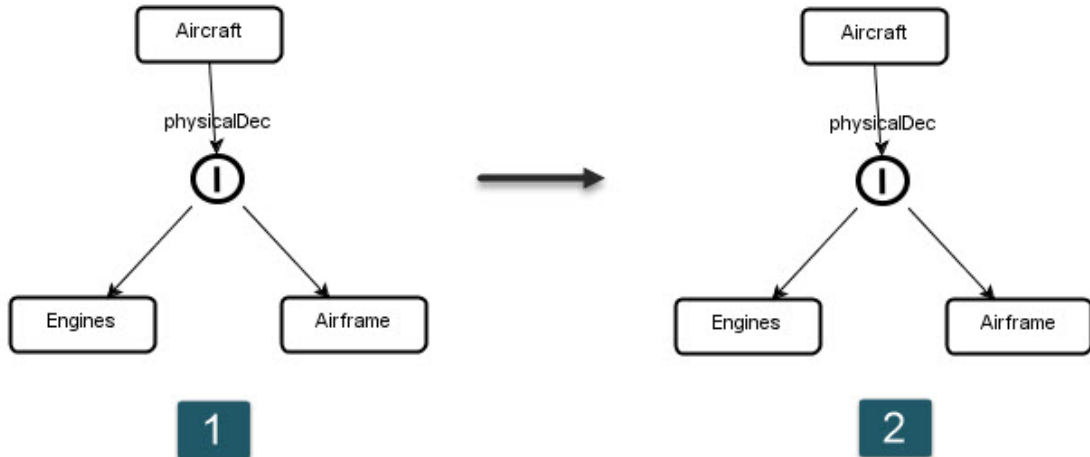


FIGURE 4.9: Pruning of Aspect Node.

### 4.4.2 Multi-Aspect Node Pruning

Figure 4.10 shows the pruning process of Multi-Aspect node. Multi-Aspect node need to define its cardinality or number of aspect before pruning. During pruning a multi-aspect node, cardinality is evaluated and chosen number of aspects of the same type are created. In the figure it can be seen that based on the cardinality number, after pruning three engines are generated.

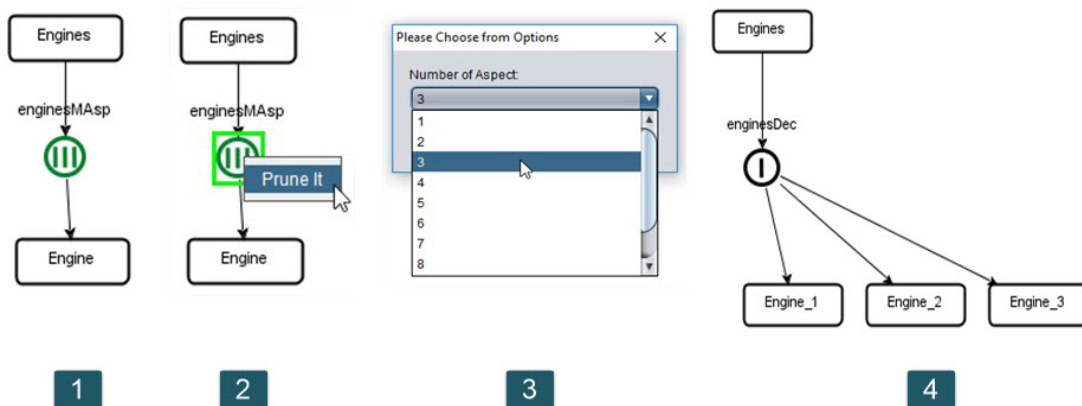


FIGURE 4.10: Pruning of Multi-Aspect Node.



### 4.4.3 Specialization Node Pruning

Figure 4.11 shows the pruning of Specialization node. Based on the specialization rule, exactly one child needs to be selected to construct a valid variant. The pruning figure shows that, entity Airframe has two options Aluminium and Composite. Thus, after pruning, the resulting entity can either be of type Aluminium or type Composite. In the figure in step 4 the completed pruned entity Aluminium\_Airframe is generated.

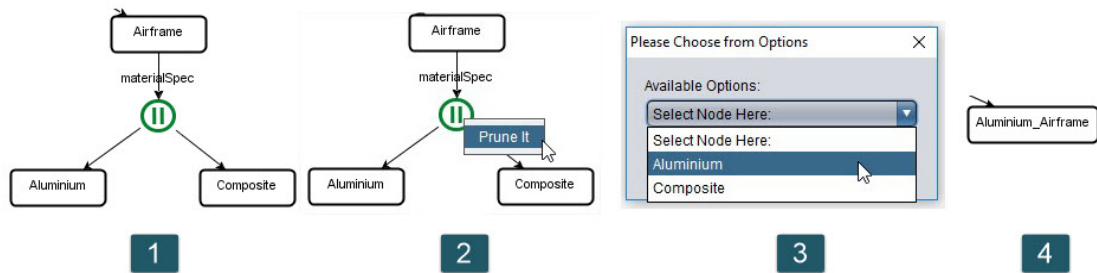


FIGURE 4.11: Pruning of Specialization Node.

### 4.4.4 Aspect Siblings Pruning

Figure 4.12 shows the pruning of Aspect Siblings node. If there are more than one aspect node in same hierarchy level then exactly one of them has to be selected by evaluating aspect rules of aspect brothers. From the figure it is clear that either node1Dec or node3Dec has to be selected during pruning. In step 3 of the aspect siblings pruning process node1Dec has been selected and finally in step 4 entity node9 has generated with one aspect node.

### 4.4.5 Specialization Siblings Pruning

Figure 4.13 shows the pruning of Specialization Siblings node. If there are more than one specialization node on the same hierarchy level then have to evaluate all. That's why in the figure in step 4, A is specialized with B from B and C and changed to B\_A. Again in final step 7, B\_A specialize with D from D and E and changed to D\_B\_A as final pruned entity. During pruning, it depends on the pruning algorithm which of the two specializations is evaluated first. For this example, left specialization node a1Spec is evaluated first. If the right specialization node a2Spec evaluated first with the same selection then the resultant pruned entity would be B\_D\_A. Other possible combinations are D\_C\_A, E\_B\_A, E\_C\_A, B\_E\_A, C\_D\_A, C\_E\_A.

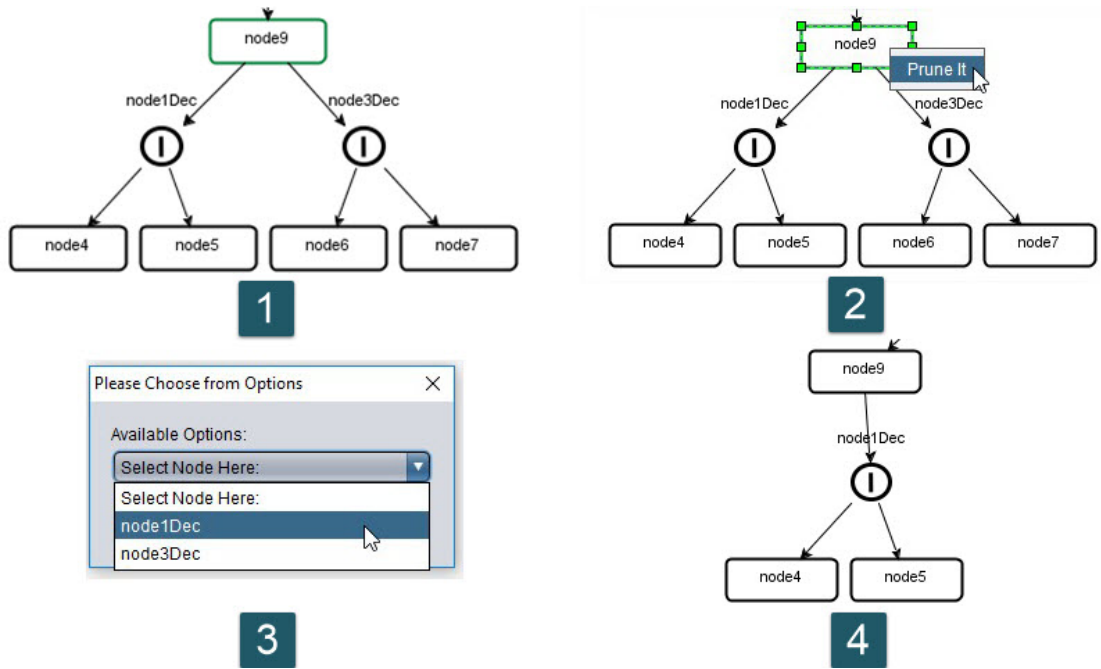


FIGURE 4.12: Pruning of Aspect Siblings Node.

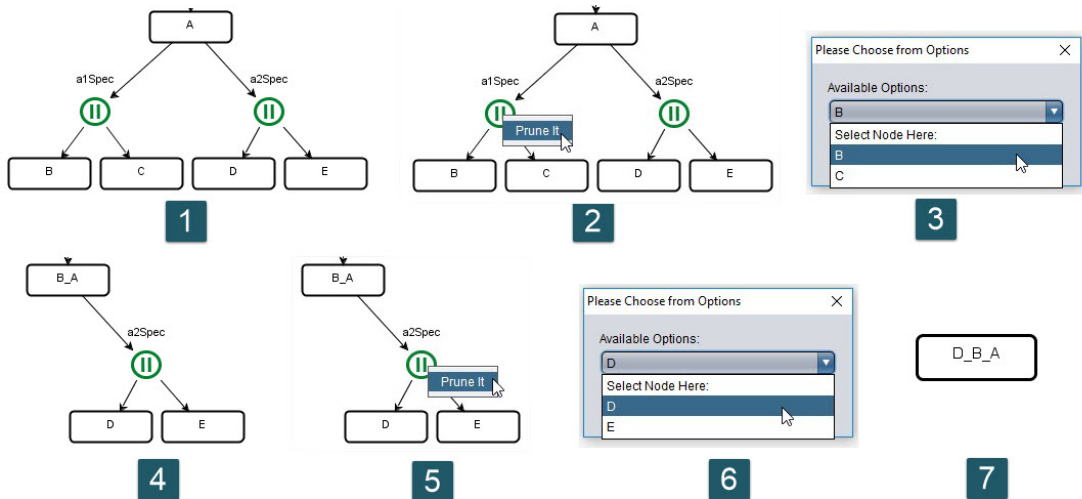


FIGURE 4.13: Pruning of Specialization Siblings Node.

#### 4.4.6 Multi-Aspect Siblings Pruning

Figure 4.14 shows the pruning of Multi-Aspect Siblings node. If there are more than one multi-aspect node in same hierarchy level then following Multi-Aspect pruning shown in 4.4.2 the multi-aspect nodes has pruned first. In the figure in position 2 it is shown that, after pruning two aspects a1Dec with its two child entities B.1, B.2 and a2Dec with its two child entities E.1, E.2 are generated. Then following Aspect Siblings pruning shown in 4.4.4 the aspect siblings node has pruned. Finally the pruned entity node A is generated in the figure in position 3 with a2Dec aspect node and its child entities.

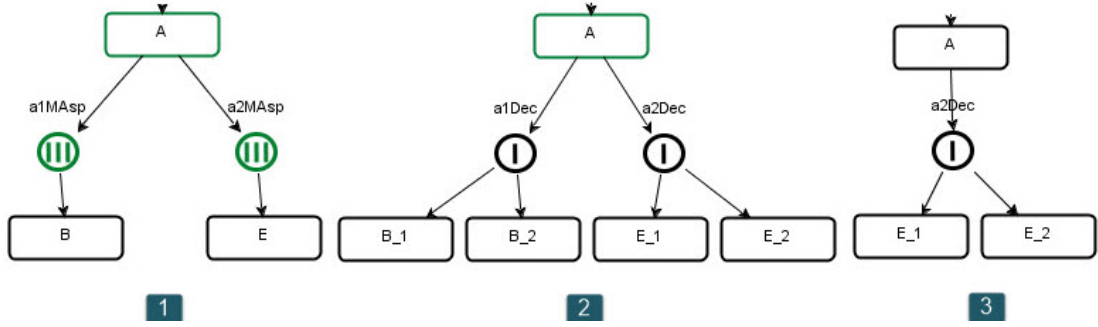


FIGURE 4.14: Pruning of Multi-Aspect Siblings Node.

#### 4.4.7 Aspect and Multi-Aspect Siblings Pruning

Figure 4.15 shows the pruning of Aspect and Multi-Aspect Siblings node. If there are aspect and multi-aspect node in same hierarchy level then following Multi-Aspect pruning shown in 4.4.2 have to prune the multi-aspect node first. In the figure 4.15 in position 2 it is shown that, aspect node a1Dec is generated with its two child entities B\_1 and B\_2 after pruning. Then following Aspect Siblings pruning shown in 4.4.4 the aspect siblings node has pruned. Finally the pruned entity node A is generated in the same figure in position 3 with a2Dec aspect node and its child entity.

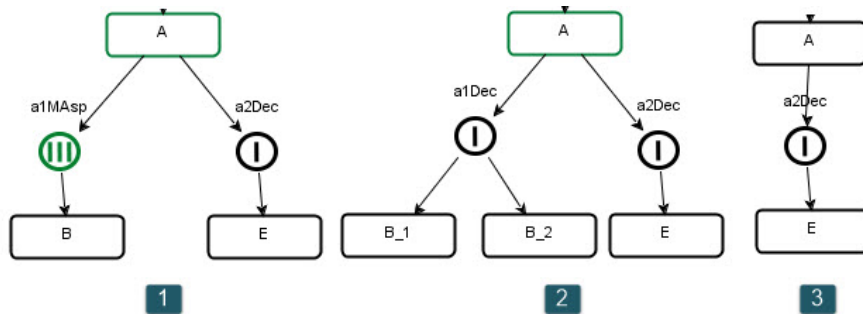


FIGURE 4.15: Pruning of Aspect and Multi-Aspect Siblings Node.

#### 4.4.8 Two Specialization Nodes in One Path Pruning

Figure 4.16 shows the pruning of Two Specialization Nodes in One Path. If there are two nodes to prune in one path then have to prune the bottom node first because PESEditor uses Botom-Up approach during pruning process. In figure 4.16, following Specialization pruning shown in 4.4.3, node B is pruned in position 2 by choosing E from the available options D and E. Again, following the same way node A is pruned by choosing E.B from the available options E.B and C. Finally, node E.B.A is generated in position 3 on the same figure.

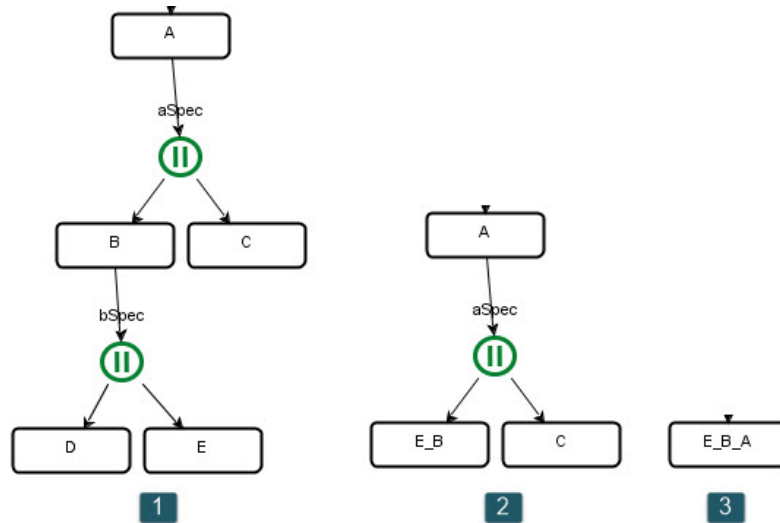


FIGURE 4.16: Pruning of Two Specialization Nodes in One Path.

#### 4.4.9 Specialization with Succeeding Aspect Pruning

Figure 4.17 shows the pruning of Specialization with Succeeding Aspect node. This pattern is a combination of a specialization node followed by a single aspect node. So, there is one alternative option and one mandatory option. In the figure 4.17 in position 2 the node A is pruned choosing B from the available options B and C and B\_A is generated with bDec as its child aspect node which is inherited from B during the specialization pruning.

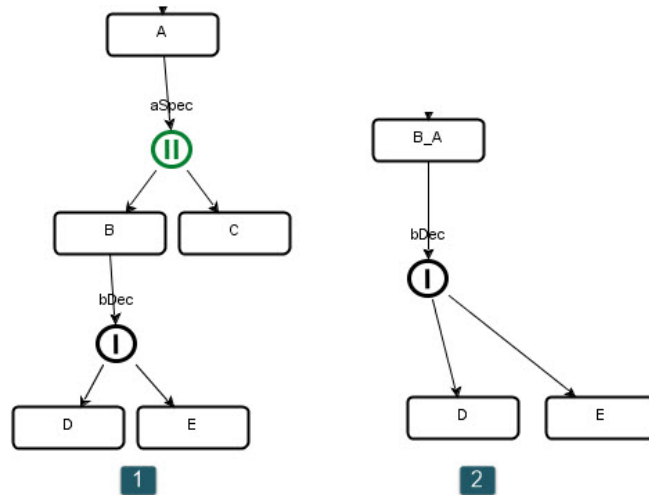


FIGURE 4.17: Pruning of Specialization with Succeeding Aspect Node.

#### 4.4.10 Specialization and Aspect Siblings Pruning

Figure 4.18 shows the pruning of Specialization and Aspect Siblings node. When aspect nodes and specialization nodes are brothers, the specialization node has to be resolved

first during pruning. In addition to that, if there is any aspect node as a child of that specialization node then two aspect node will be siblings. In figure 4.18 position 2 after pruning two aspect node is generated because from the available options D and E, E is selected as a type of A. Then following Aspect Siblings pruning shown in 4.4.4 the aspect siblings node has pruned. Finally the pruned entity node E\_A is generated in the same figure in position 3 with eDec aspect node and its child entities F and G.

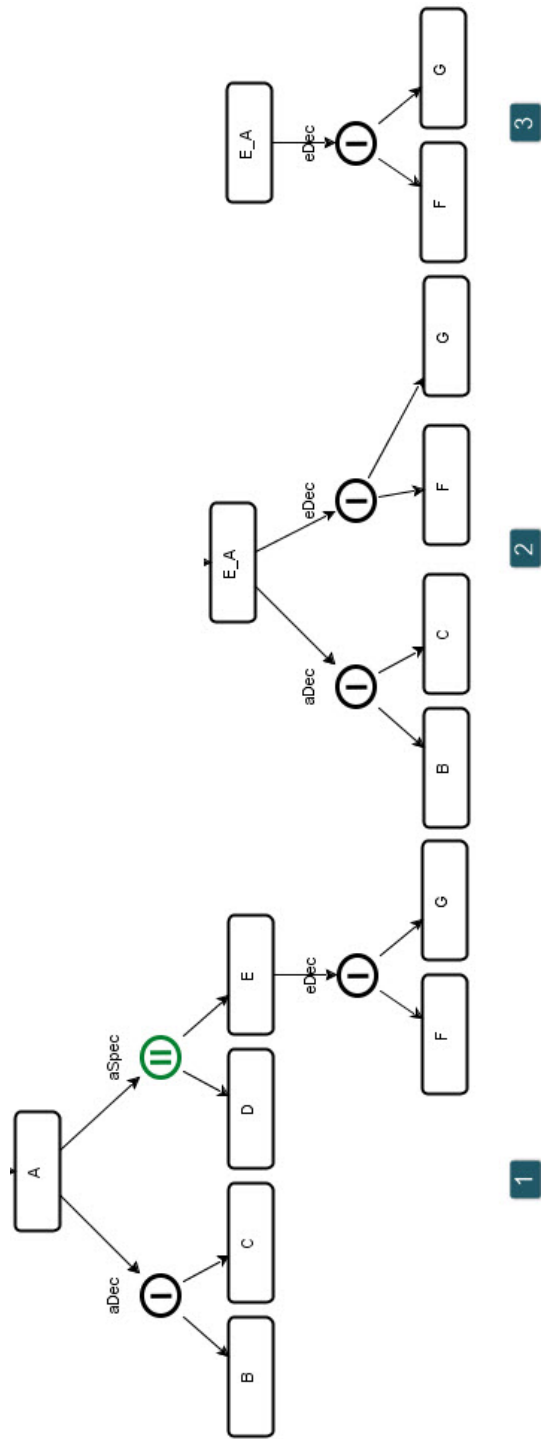


FIGURE 4.18: Pruning of Specialization and Aspect Siblings Node.

## 4.5 Package Design and Class Static Structure

### 4.5.1 Common Packages

Figure 4.19 shows how packages are nested for the common use. Packages common for both the editors are:

- dlr.xml.schema
- dlr.ses.core
- dlr.resources.images
- dlr.resources.docs

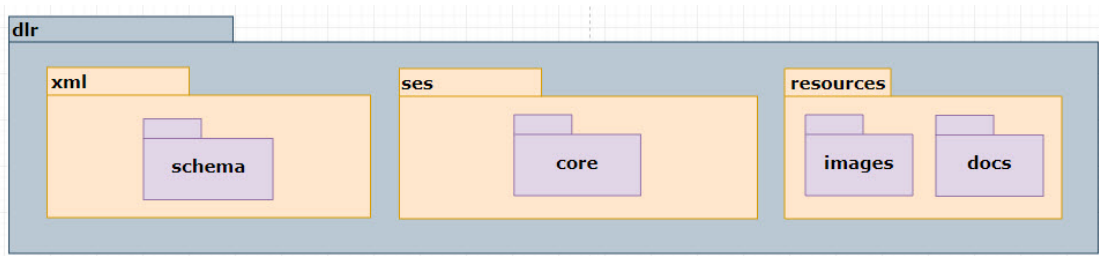


FIGURE 4.19: Common Package Structure.

### 4.5.2 PESEditor Packages

Figure 4.20 shows how packages are nested in the SESEditor. All the classes which are only related to PESEditor are placed in the **peseditor** package. Packages for PESEditor are:

- dlr.xml.schema
- dlr.xml.xslt
- dlr.ses.core
- dlr.ses.peseditor
- dlr.resources.images
- dlr.resources.docs

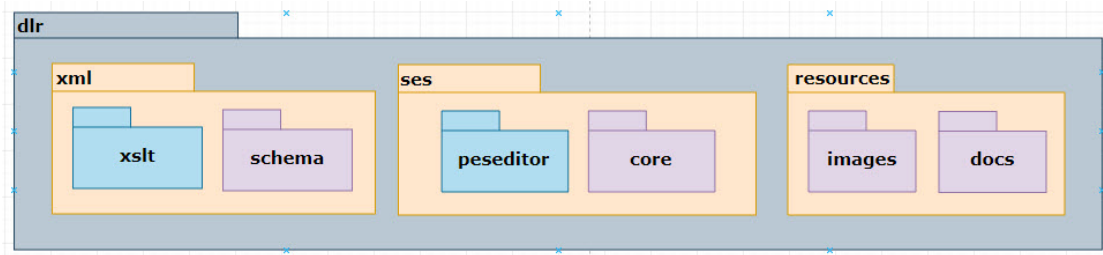


FIGURE 4.20: PESEditor Package Structure.

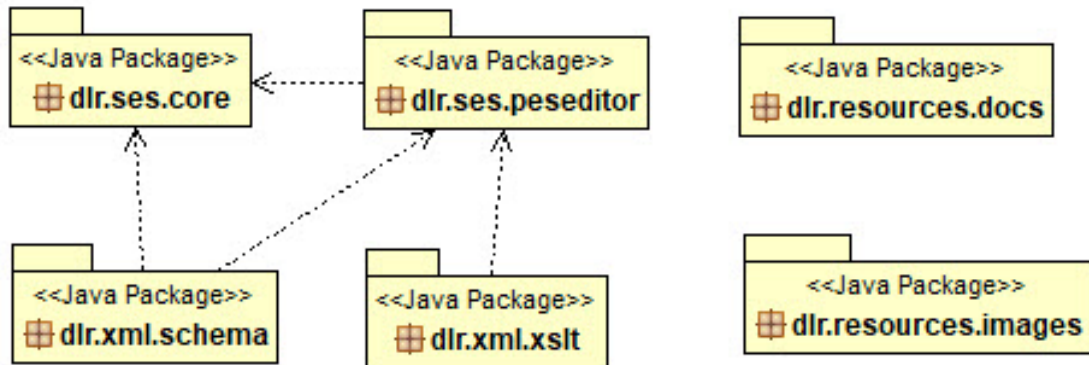


FIGURE 4.21: PESEditor Package Design.

### 4.5.3 Package Design

Figure 4.21 shows the relation between all the packages used in the PESEditor.

### 4.5.4 Class Diagram: core

Figure 4.22 shows the class diagram of PESEditor core classes. Description of all the classes are added in Appendix A.

### 4.5.5 Class Diagram: peseditor

Figure 4.23 shows the class diagram of PESEditor peseditor classes. Description of all the classes are added in Appendix A.



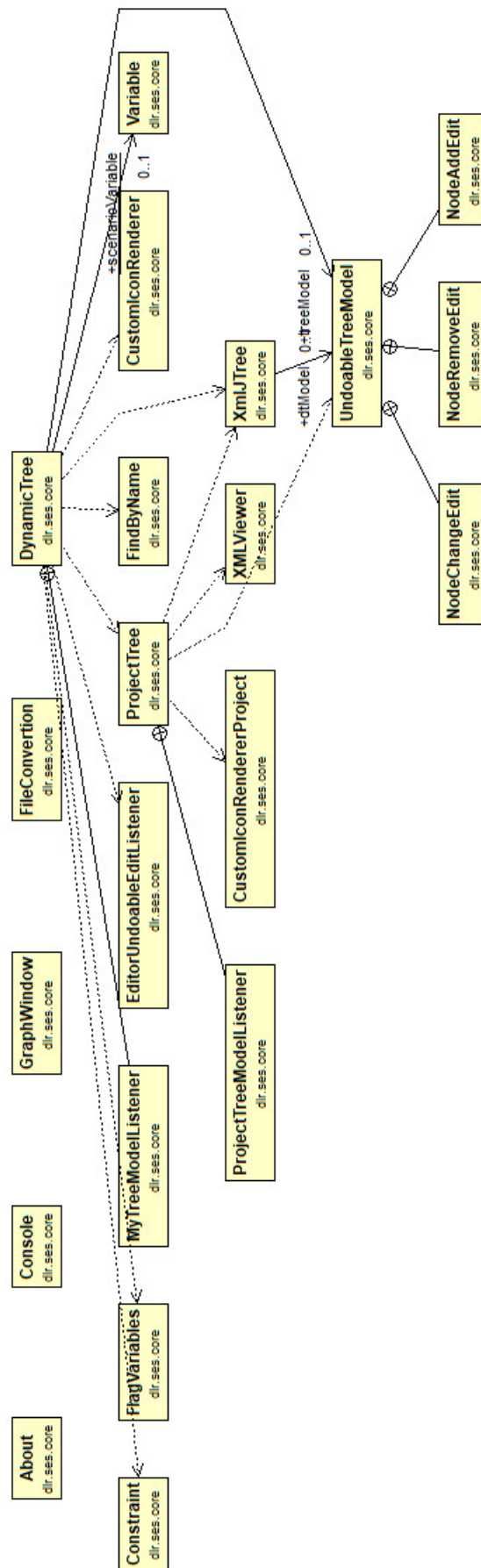


FIGURE 4.22: PESEditor Class Diagram: core package.

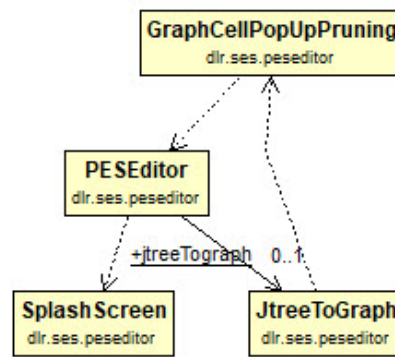


FIGURE 4.23: PESEditor Class Diagram: peseditor package.

## 4.6 Used Technologies

### 4.6.1 Java

PESEditor is the extension of SESEditor code base. Like SESEditor described in section 3.5.1, Java is also used as programming language here.

### 4.6.2 Xerces

Following SESEditor validation technology described in section 3.5.2 Xerces is used in PESEditor also for checking the validity of the pruned model.

### 4.6.3 JGraphX

Drawing panel of PESEditor works almost similar to SESEditor and thats why it is desinged using the same technology. Here also JGraphX is used like SESEditor which is described in section 3.5.3.

### 4.6.4 RSyntaxTextArea

For displaying PES XML and PES Schema PESEditor used RSyntaxTextArea exactly same way like SESEditor described in section 3.5.4.

### 4.6.5 Goava

For long string processing Goava library is used in PESEditor exactly same way as it is used in SESEditor described in section 3.5.5.

## Chapter 5

# Use Cases

### 5.1 Music Performances

System Entity Structure can be used for modeling music performances. A metamodel created in SESEditor is presented in figure 5.1. It shows three types of music specialization namely Symphonic, Folk and Jazz and three types of ensemble specialization namely Orchestra, SmallGroup and Soloist.

As not all combinations are likely, table 5.1 shows the possible music performances. Indeed, some might be so unlikely as to be considered impossible. For example, Symphonic music can only be played by an Orchestra; Folk music is usually played by a small group and sometimes by a soloist; and Jazz is usually played by a Small Group and sometimes by an Orchestra

styleSpec/ensembleSpec	Orchestra	SmallGroup	Soloist
Symphonic	x		
Folk		x	x
Jazz	x	x	

TABLE 5.1: Possible Music Performances

To restrict choices only to the possible music performances constraints are added in the MusicPerformanceDec aspect node of the SES metamodel presented in figure 5.1. Listing 11 shows the the constraint for implementing Symphonic music performance. If Symphonic is selected from the Music node then the only option from performer is Orchestra can be selected. Listing 12 shows the the constraint for implementing Folk music performance. Orchestra can not be selected if the Music selection is Folk. Listing 13 shows the the constraint for implementing Jazz music performance. In case of Jazz

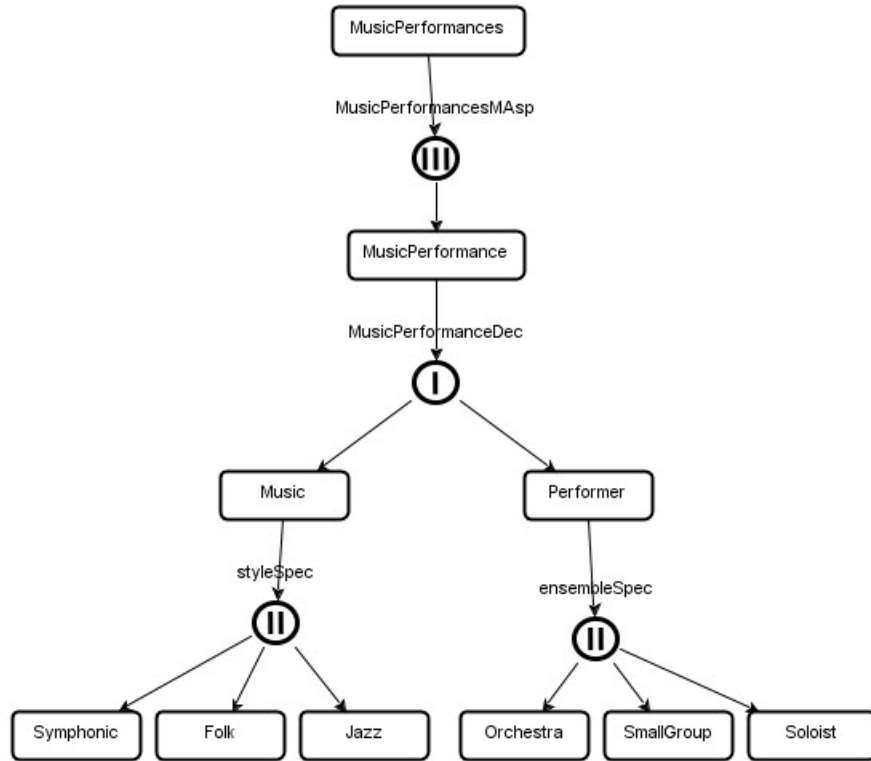


FIGURE 5.1: SES Metamodel for Musical Performances

music performance Soloist can not be selected. Orchestra and SmallGroup are possible performer for Jazz music.

```

<xs:assert test="if(Music/*[@name='Symphonic']) then (Performer/*[@name=
'Orchestra']) else true()"
/>

```

LISTING 11: Constraint Example for Symphonic Musical Performance Combinations.

```

<xs:assert test="if(Music/*[@name='Folk']) then (Performer/*[@name=
'SmallGroup']) or (Performer/*[@name='Soloist']) else true()"
/>

```

LISTING 12: Constraint Example for Folk Musical Performance Combinations.

```

<xs:assert test="if(Music/*[@name='Jazz']) then (Performer/*[@name=
'Orchestra']) or (Performer/*[@name='SmallGroup']) else true()"
/>

```

LISTING 13: Constraint Example for Jazz Musical Performance Combinations.

After modeling, musical performance SES metamodel is pruned in PESEditor for specific music performance. Figure 5.2, 5.3 and 5.4 shows Symphonic, Folk and Jazz musical performance pruned model respectively. Symphonic and Folk musical performances

represents one music performance. Jazz musical performance shows two music performances.

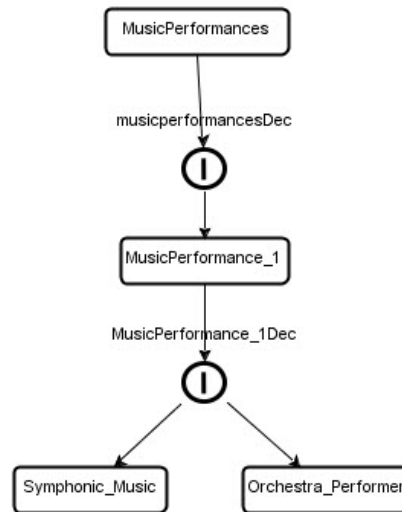


FIGURE 5.2: Symphonic Musical Performance.

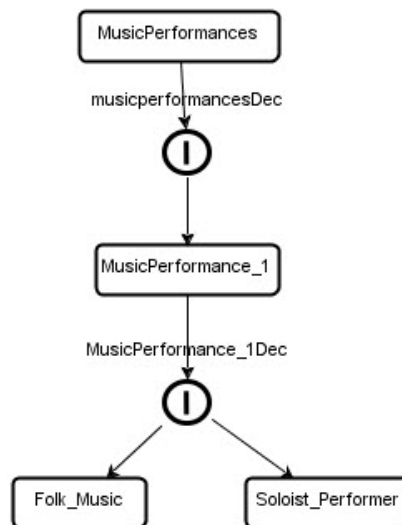


FIGURE 5.3: Folk Musical Performance.

## 5.2 Flight Simulation Scenario Development

Simulation scenarios are the subjects of interest throughout the whole simulation study. Siegfried et al. [2012, 2013] identified three types of scenarios: operational scenario, conceptual scenario and executable scenario in their paper. A simulation study typically starts with the description of the simulation scenario and ends with a successful simulation execution. Scenarios are used as executable specifications of a series of events and conditions being simulated. They are also used for identifying simulator capability

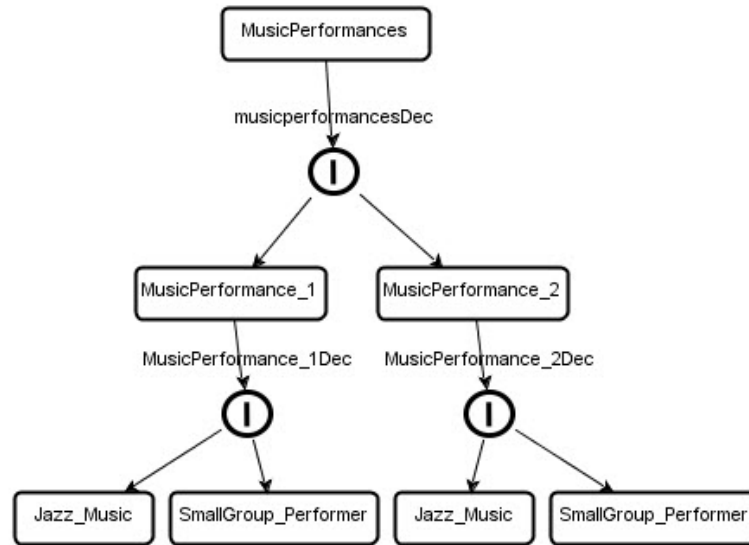


FIGURE 5.4: Jazz Musical Performances.

gaps and identifying minimum simulator requirements. Taking scenario development as one of the core simulation engineering activities, we need shared understanding, common practices among stakeholders, operational subject matter experts and technical personnel and engineers, and we further need standards to boost interoperability and sharability.

[Durak et al. \[2018a\]](#) summarized the panel discussion on standardization for simulation scenario development in aviation in their paper. User-friendliness of the scenario development process is very important. Also, the process for reusing of existing scenarios or a part of an existing meta-model is important. Consistency and completeness in scenario development is essential for this reason. If the elements of the scenario are changing over time as well as the consistency rules, then it is an important and challenging task to ensure the consistency of the scenario definition. Although automation is used heavily, maintaining a huge number of scenarios that needs to be executed in a relatively short simulator campaign requires checking and fixing the errors in a large number of scenarios is a challenge. Furthermore, current commercial flight simulator infrastructures have a set of certified legacy scenarios and the authorities are totally happy with them. He also added that there may be some resistance in changing to new scenario development approaches unless there exists a clear motivation.

Despite the known importance of scenarios, there is still a lack of common understanding and standardized practices in simulation scenario development which motivates a recent effort for standardization of scenario languages in aviation. System Entity Structure (SES) was proposed as a formal approach for a standard simulation scenario definition

language. According to [Durak et al. \[2017\]](#) SES provides formal means to capture concepts and their relations in a domain. It enables specifying family of concept structures and properties which can then be pruned to a Pruned Entity Structure (PES). Then, an XML-based computational representation of SES and PES was also presented by [Durak et al. \[2018b\]](#). Based on [Durak et al. \[2018b, 2017\]](#), our developed tools SESEditor and PESEditor are suitable for generating simulation scenario.

### 5.2.1 Scenario Related Work

According to [Brambilla et al. \[2012\]](#) Domain Specific Languages (DSLs) are custom-tailored computer languages for a particular application domain that specifically target problems in that specific domain, and stresses upon the main idea, features, constraints and characteristics of that domain. In early 2000s industry adapted the idea for developing a standard scenario definition language for military simulations. The Military Scenario Definition Language (MSDL) was published ([Group \[2008\]](#)) as a standard by Simulation Interoperability Standards Organization (SISO) in 2008.

The research on extending the idea to other simulation domains and eventually with better formal basis is still active. A scenario specification based on a DSL is presented by [Schütte \[2011\]](#) in his paper which can be used to formally describe simulations and scenarios. [Siegfried et al. \[2013\]](#) proposed using Base Object Model (BOM) for modeling simulation scenarios in military domain.

There is a recent effort coordinated by the American Institute of Aeronautics and Astronautics (AIAA) Modeling and Simulation Technical Committee (MSTC) towards development of a standard scenario definition language for aviation. There are currently two modeling approaches: [Jafer et al. \[2017, 2016\]](#) proposes a metamodeling using Eclipse Modeling Framework extending the idea presented in [Siegfried et al. \[2013\]](#), [Durak et al. \[2018b, 2017\]](#) utilize SES for metamodeling in order to specify the language. According to [Jafer et al. \[2018\]](#) both of the approaches yield to a unified computational representation based on XML Schema.

### 5.2.2 Scenario Development Process

[Siegfried et al. \[2012\]](#) distinguished three types of scenarios which are produced during scenario development process and these are used from the operator, sponsor or user statement of scenario to the executable specifications for machine processing.

1. Operational scenarios are described in early stages by the user or the sponsor. The intended situation and its dynamics are described in operational scenarios.

2. Conceptual scenario can be understood as a modeling and simulation expert's specification of the operational scenario. It is a structured scenario specification that identifies and clarifies all points required for consistency and completeness.
3. Executable scenario is defined as the specification of the conceptual scenario to be processed by the simulation applications for preparation, initialization and execution.

Scenario development is viewed as the transformation of operational scenarios to conceptual scenarios and eventually to executable scenarios. According to [Durak et al. \[2014\]](#) as the development process occurs, the scenario models are refined and transformed. The scenario development process starts with operational scenarios, mostly in text form constructed in natural language by domain experts. They usually present key information from the user's perspective and are often not complete and consistent. Operational scenarios are then used to develop conceptual scenarios, essentially to be based on a metamodel, either an explicit or an implicit one. The result is then a complete and a consistent scenario specification, often compiled in a table. Finally conceptual scenarios are transformed into executable scenarios for target simulation environments using a set of rules specific to the target simulation environment. Fig. 5.5 shows the transformation details of operational scenarios to executable scenarios during scenario development using SES. At the beginning domain experts provides text of the scenario in operational space. From that text possible entities and their classifications are filtered. Entities are classified using SES elements. In conceptual space, Core Scenario SES is used with the filtered entities from the text to create the metamodel. This metamodel is designed in the developed SES modeling environment SESEditor. The editor also check the validity of the created model according to the predefined XML Schema. To create a particular scenario from that metamodel another developed tool PESEditor is used. By using various combination from the metamodel a number of specific scenario then derived. This derived scenario is called pruned entity structure. When pruned entity structure is ready then in computational space a computational representation of the scenario is prepared using XML Schema. And finally from the pruned entity structure executable scenarios are generated using XSLT.

### 5.2.3 Scenario Examples

In order to model aviation scenarios, an SES metamodel has been created using our developed SESEditor. The idea behind the metamodels presented here is to have two sets of metamodels that contain information for all types of scenarios to be modeled: (1) an aviation scenario core which contains the elements that are present in all scenarios, and



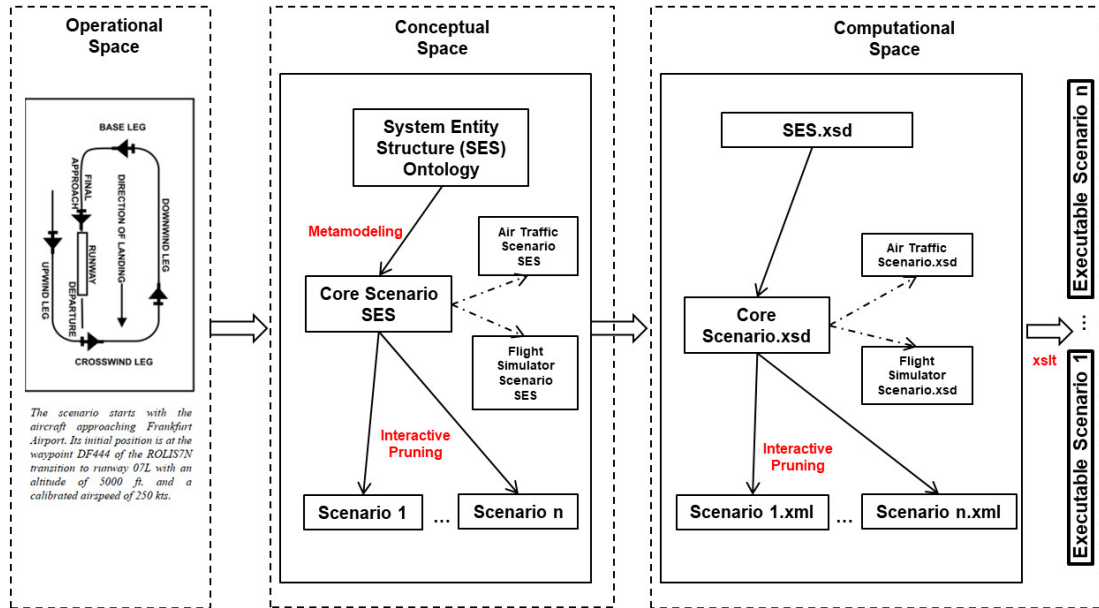


FIGURE 5.5: SES for Scenario Development (Durak et al. [2018b]).

(2) extension modules specific to certain types of aviation scenarios which are only used to model scenarios of that type. The Aviation Scenario Core and two modules—Traffic Server and Air Traffic Control—are discussed in this section.

### 5.2.3.1 Aviation Scenario Core

The aviation scenario core SES includes elements which are present in all scenarios, such as aircraft, airports and certain environmental entities. This metamodel can be seen in Figure 5.6.

Each scenario has three major components: (1) Environment, (2) Entities, and (3) Events. The major Entities that are a part of every aviation scenario are Airspace, Aircraft And Airport. The Events comprise of Actions and Guard conditions, which are defined by a change in State or Time. Specific events can also be modeled such as bird strikes or engine stall, for instance.

### 5.2.3.2 Traffic Server

The traffic server module deals with elements that are specific to Traffic Server scenarios. This metamodel can be seen in Figure 5.7.

Each Traffic Server (TS) scenario has four major components: (1) Simulation, (2) Environment, (3) Entities, and (4) Events. The elements specific to Simulation include

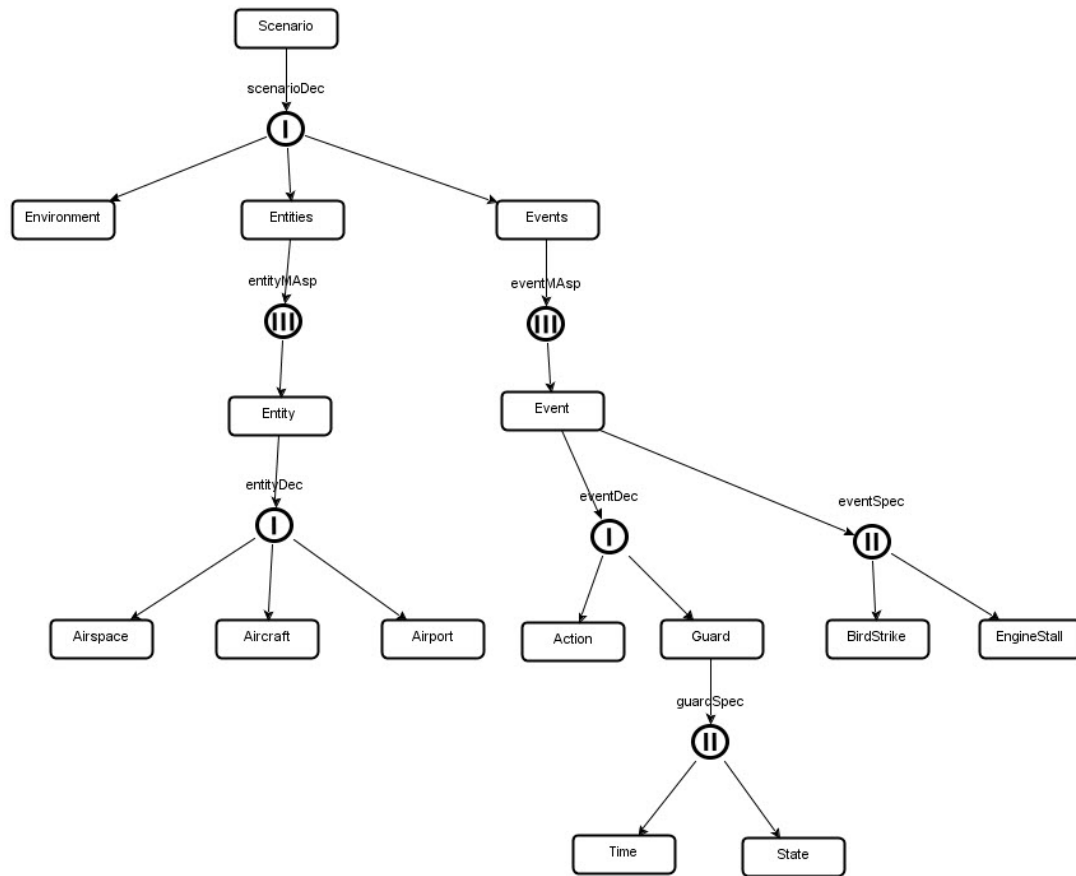


FIGURE 5.6: Aviation Scenario Core.

Settings, such as visual settings, motion settings and execution settings for the simulation. The Waypoints of the Aircraft are also added as entities for the traffic server scenarios.

In Figure 5.8 the pruned model of Traffic Server has been presented. During interactive pruning, at first, by setting the the cardinality of settingsMASp multiaspect node, two simulation aspects have been derived. Then, two aspects of Waypoints have been derived by setting cardinality of waypointMASp multiaspect node. Also, two aspects of Entities have been derived by setting cardinality of entityMASp node. Finally, from two available options BirdStrike has been selected to create specific event BirdStrike\_Event and later State has been selected to create specific State\_Guard condition aspect. By setting cardinality of eventMASp multiaspect node, two aspects of Events has been derived to complete the pruning of Traffic Server metamodel. Listing 14 represents the excerpt from the executable scenario of traffic server derived from the pruned entity structure presented in Figure 5.8.

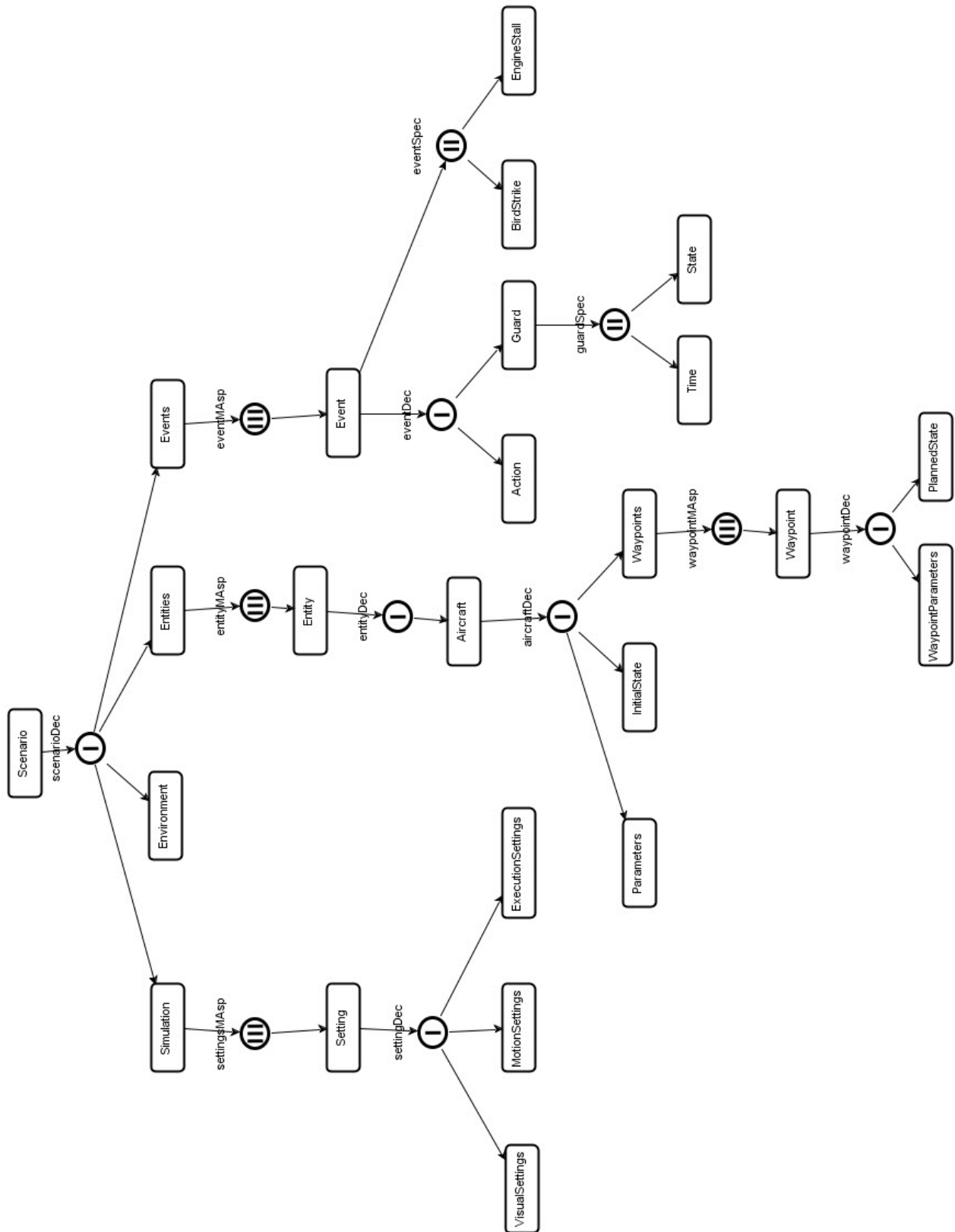


FIGURE 5.7: Traffic Server.

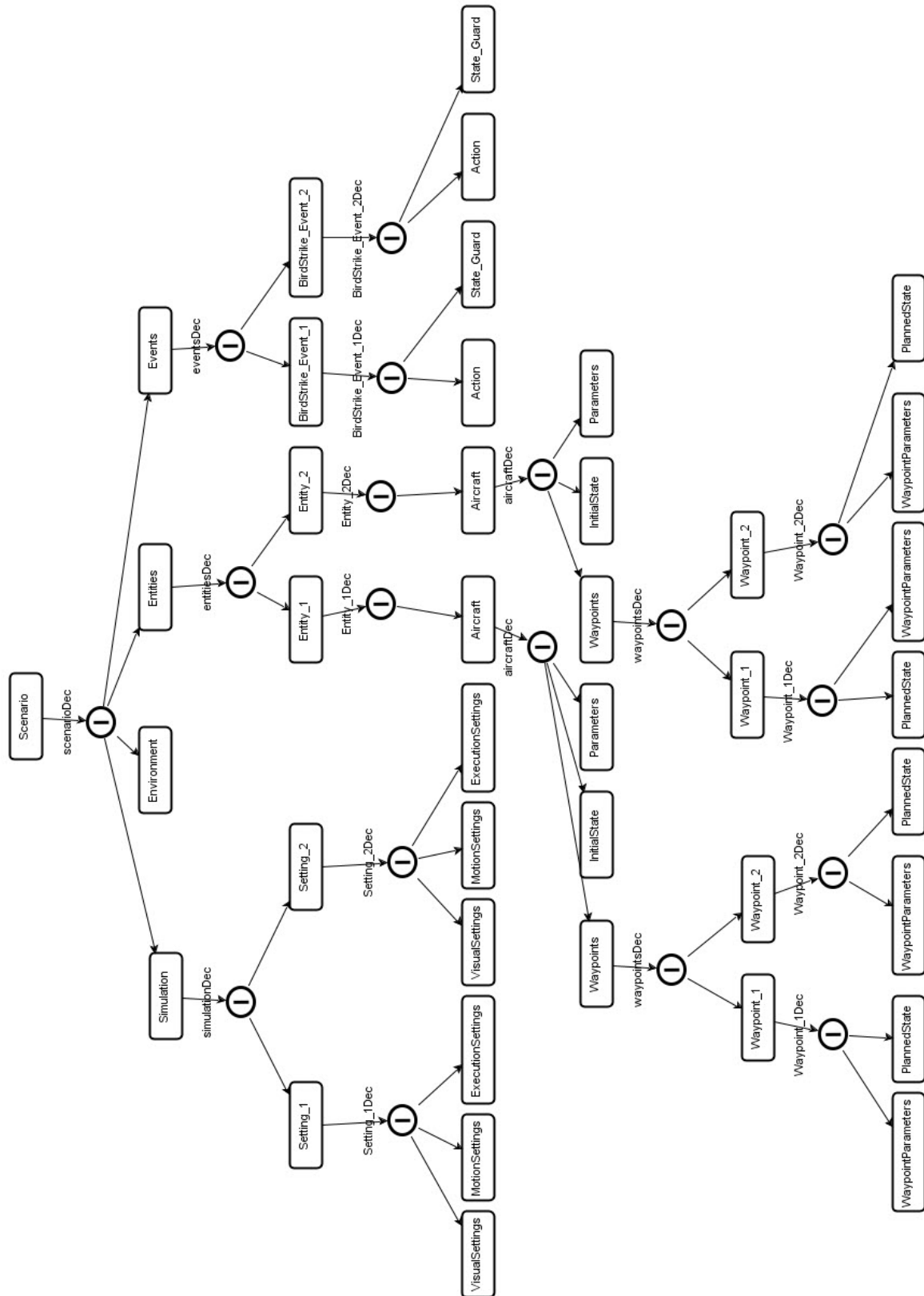


FIGURE 5.8: Pruned Traffic Server.

### 5.2.3.3 Air Traffic Control

The Air Traffic Control extension module deals with elements that are needed for simulation of ATC scenarios. This metamodel can be seen in Figure 5.9.

Each Air Traffic Control (ATC) scenario has three major components: (1) Environment, (2) Entities and (3) Events. A Scenario may have multiple Entities, but each Entity is either an Aircraft or Airspace. In any scenario, there will be one Airspace element only. But in most scenarios, there will be many instances of Aircraft. An Aircraft has two types of entities: (1) the physical systems onboard the aircraft, and (2) the current flight parameters. The Airspace contains Airports, airways and jet routes and fixes or waypoints used for navigation in ATC.

In Figure 5.10, the pruned model of Air Traffic Control has been presented. During interactive pruning, at first, from four available options Communication has been selected to create specific system `Communication_System` and later by setting the cardinality of `systemMAsp` multiaspect node two aspects of Systems has been derived. Then, from available Aircraft and Airspace entities, Aircraft has been selected to create specific entity `Aircraft_Entity`. And also two aspects of Entities have been derived by setting cardinality of `entityMAsp` multiaspect node. Finally, from two available options Bird-Strike has been selected to create specific event `BirdStrike_Event` and later Time has been selected to create specific `Time_Guard` condition aspect. By setting cardinality of `eventMAsp` multiaspect node, two aspects of Events has been derived to complete the pruning of Air Traffic Control metamodel.

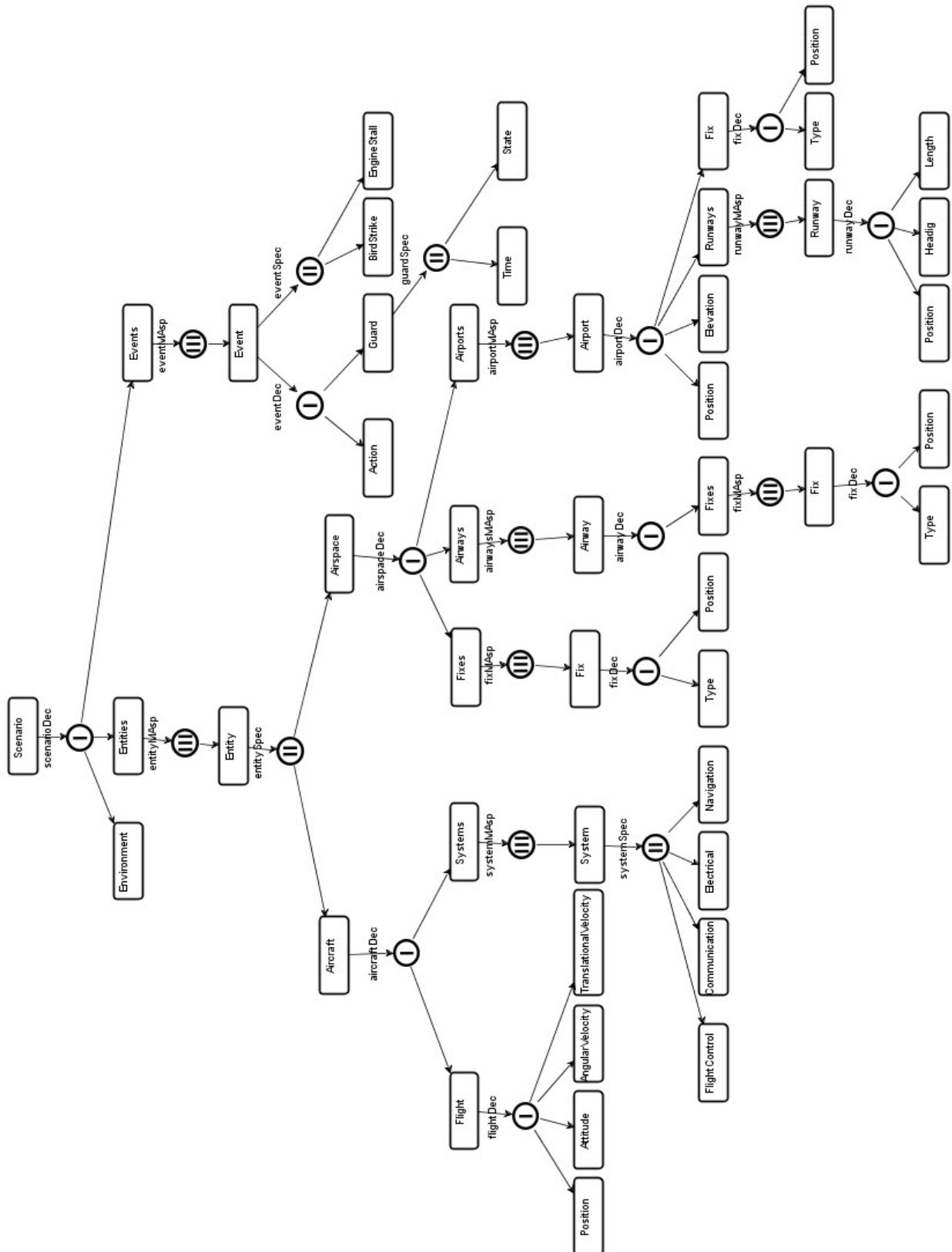


FIGURE 5.9: Air Traffic Control.

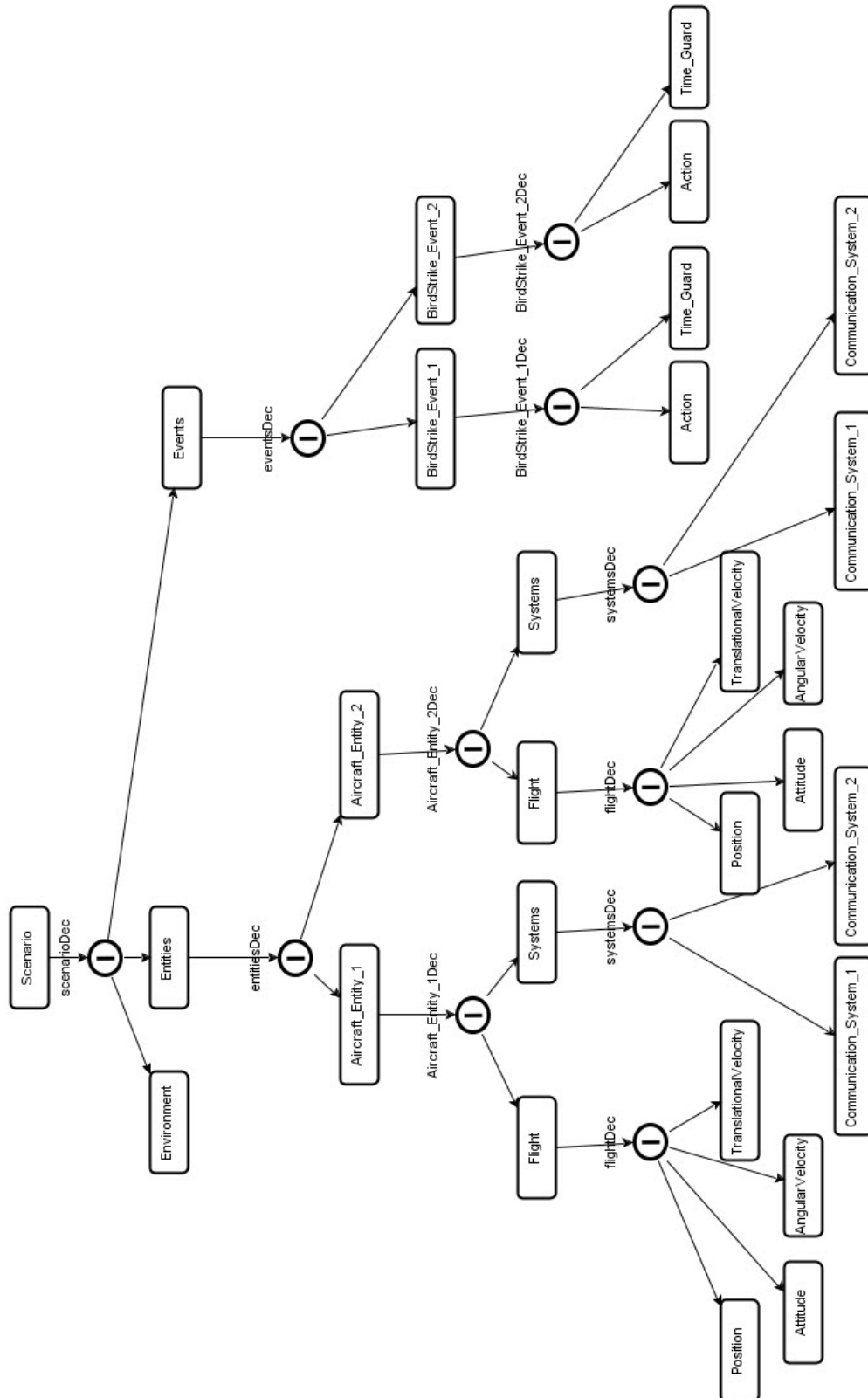


FIGURE 5.10: Pruned Air Traffic Control.

```

<objects name="" text="">
<object class="2">
<description text=""></description>
<properties>
  <property pid="1" type="1" size="1" unit="kg/m3">
    <value value="5.5"></value>
  </property>
</properties>
<waypoints>
  <waypoint>
    <position lat_rad="0.915563" lon_rad="0.169016" alt_m="56.5" />
    <next index="2" />
    <properties>
      <property pid="50" type="6" size="1" unit="rad">
        <value value=".915562"></value>
      </property>
      <property pid="51" type="6" size="1" unit="rad">
        <value value=".169071"></value>
      </property>
      <property pid="54" type="6" size="1" unit="kg/m3">
        <value value="0"></value>
      </property>
    </properties>
  </waypoint>
  <waypoint>
    <position lat_rad="0.913111" lon_rad="0.184149" alt_m="91" />
    <next index="0" />
    <properties>
      <property pid="50" type="6" size="1" unit="rad">
        <value value=".915562"></value>
      </property>
      <property pid="51" type="6" size="1" unit="rad">
        <value value=".169071"></value>
      </property>
      <property pid="54" type="6" size="1" unit="kg/m3">
        <value value="0"></value>
      </property>
    </properties>
  </waypoint>
</waypoints>
</object>
</objects>

```

LISTING 14: Excerpt from the Executable Scenario of Traffic Server



### 5.3 Geofence

Geofencing is simply the creation of virtual fences around areas or points of interest to keep drones away. The drone will need a reliable navigation system and autopilot software to create and interact with a fence. For example, if there is a five mile “no-fly-zone” around any airport then aircraft’s autopilot could store the position (latitude and longitude) of the airport. As the drone flies, the autopilot continually computes the distance between the drone and the airport. When it reaches five miles, the autopilot must respond with a maneuver and/or notify the pilot to do something. If the drone was already in the no-fly-zone, then the control system may prevent takeoff or flying above a specific altitude.

Geofencing is a virtual barrier created using a combination of the GPS (Global Positioning System) network and LRFID (Local Radio Frequency Identifier) connections such as Wi-Fi or Bluetooth beacons. This boundary is dictated by a combination of hardware and software which dictates the parameters of the geofence i.e. a drone app and an unmanned aircraft.

More modern uses of geofencing include ‘Smart Home’ functionality such as room heating turning itself off when phone pings at a certain distance from the house and back on when phone is heading back. Drones use geofencing on a much more focused level, usually to satisfy aviation agency regulations about the use of airspace.

Figure 5.11 shows a geofence metamodel created in SESEditor. Geofence is decomposed in Zones and Zones is consisting of multiple Zone. Zone is decomposed in Status, DataCaptureProhibit, PopulationDensity, Origin, OperatorCategory, LevelOfValidation and Coordinates. Status can be Forbidden, OnRequest or Authorized and it is decomposed into Height, HeightReference and TimeOfApplicationability. Height can be upHeightMax or downToHeightMin. HeightReference can be AGL or MSL. TimeOfApplicationability can be FromD1toD2, Always, FromT1toT2 or OnDays. DataCaptureProhibit can be Yes or No. PopulationDensity can be Assemblies, Populated, ScarceClose or ScarceFar. Origin can be AirTraffic, Privacy, WildLife, Population or SensitiveSite. OperatorCategory can be Military, FireFighting, Professional, Leisure, HealthAndRescue or Police. LevelOfValidation can be Low, Medium or High. Coordinates is consisting of multiple Coordinate.

Figure 5.12 shows the pruned entity structure of the geofence metamodel created in PESEditor. After pruning, Geofence is decomposed in Zones and Zones is consisting of Zone\_1 and Zone\_2. In each Zone, Status is specialized with Forbidden. Height is specialized with upHeightMax. HeightReference is specialized with AGL. TimeOfApplicationability is specialized with OnDays. DataCaptureProhibit is specialized with Yes.

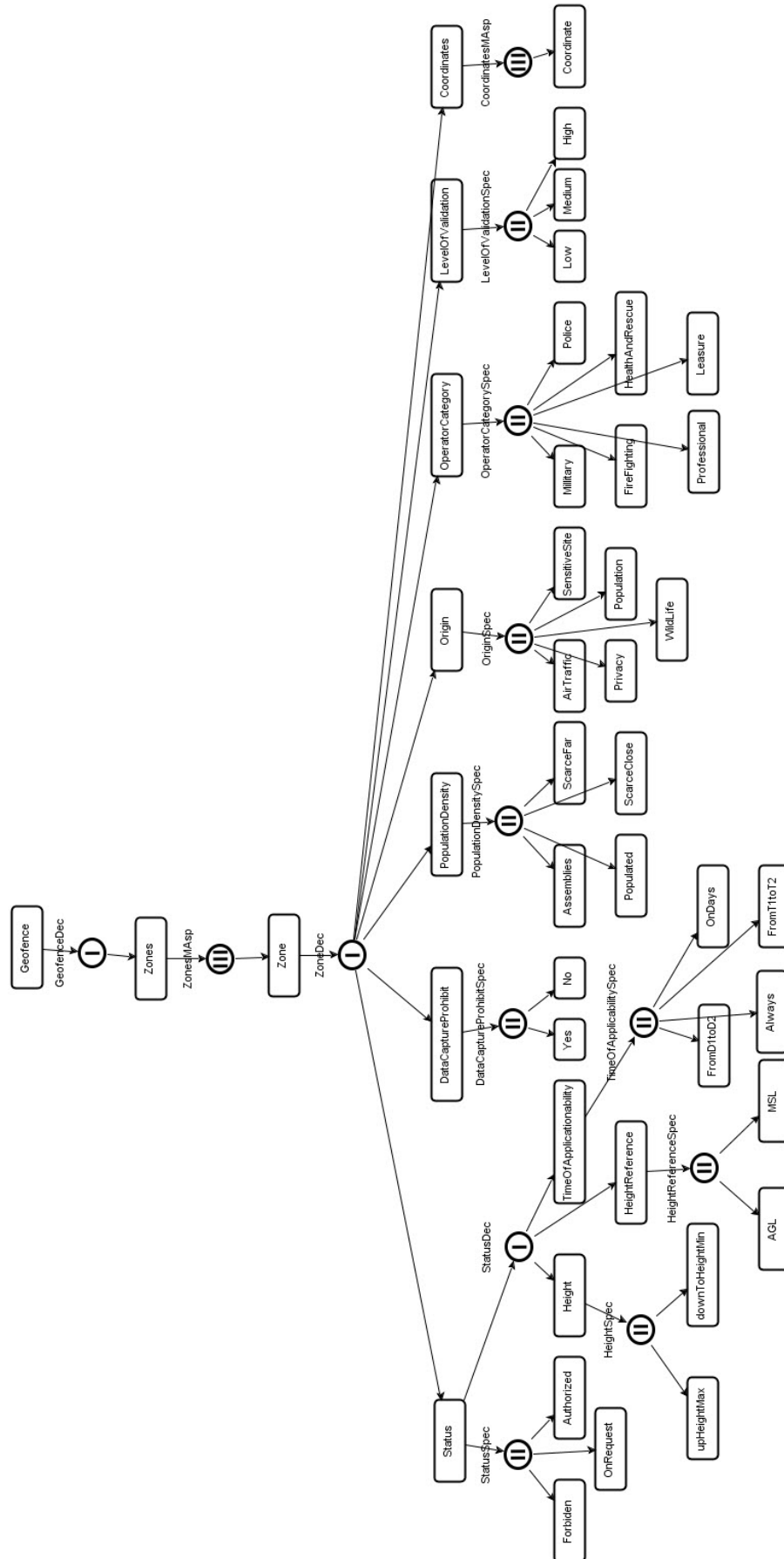


FIGURE 5.11: Geofence Metamodel Using SES.

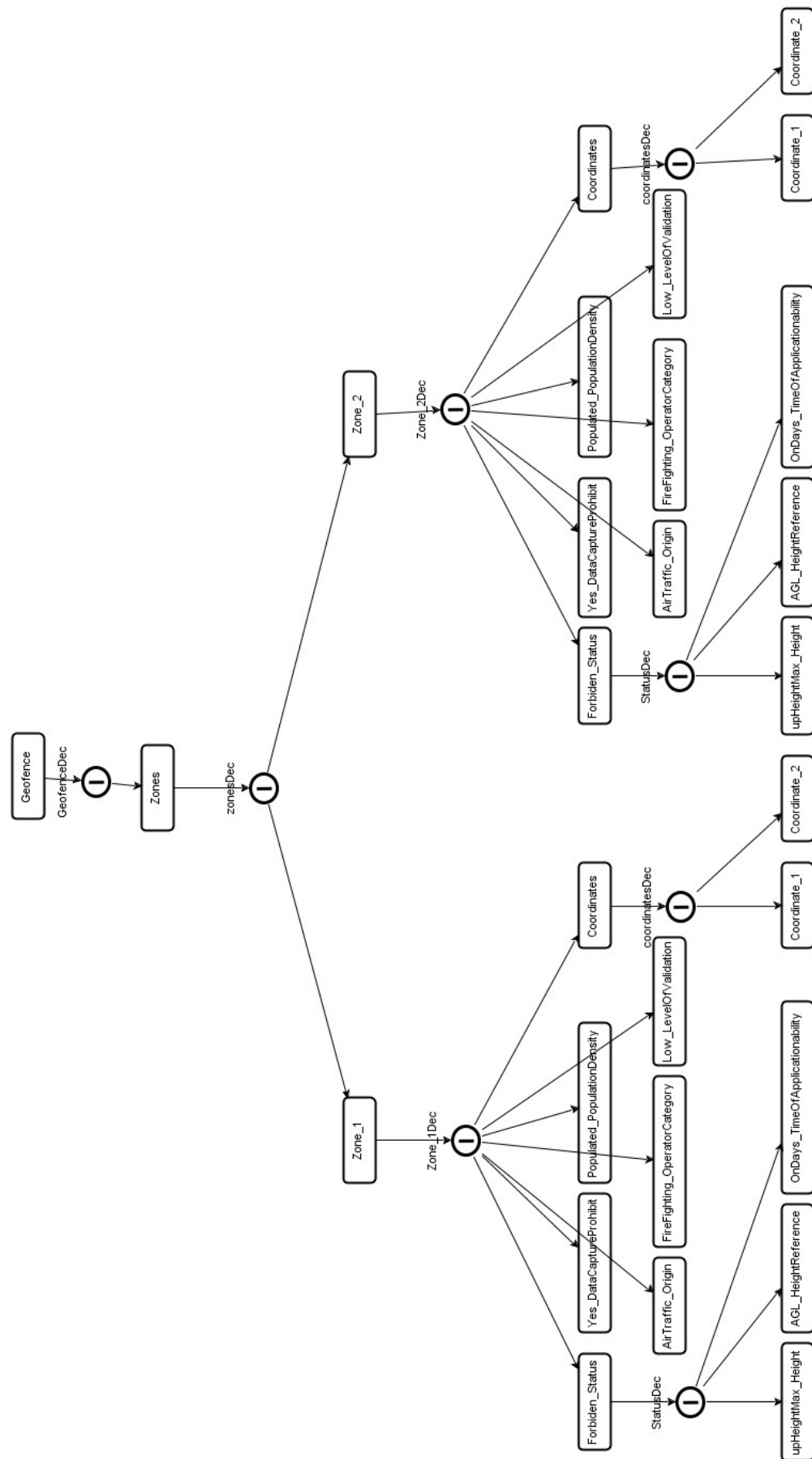


FIGURE 5.12: Pruned Model of Geofence

---

PopulationDensity is specialized with Populated. Origin is specialized with AirTraffic. OperatorCategory is specialized with FireFighting. LevelOfValidation is specialized with Low. Coordinates is consisting of Coordinate\_1 and Coordinate\_2.

## Chapter 6

# Conclusion

System Entity Structure is one of the most important topics in modeling and simulation related research and development. Despite the significant use of SES in modeling and simulation for various purposes, there is a lack of common tools, formats and practices. This eventually impairs shareability and interoperability. To provide a shareable and interoperable environment among the research community of modeling and simulation, in this thesis we have introduced application agnostic SES modeling environment, namely SESEditor and interactive pruning tool, namely PESEditor for developing model using System Entity Structure. Also there was no way of adding condition on current computational representation of SES. We have extended computational representation and introduced constraint as a condition using XPath and XML Schema in the computational representation of SES. We have also added functionality in our editors to handle constraint during generating computational representation of SES.

The developed tools are utilized in three different use cases in order to demonstrate an end-to-end SES modeling and pruning process. SES metamodel for possible music performances are modeled with constraint and later pruned to specific music performance. Also the simulation scenario definition that is specified using System Entity Structure formalism, is used to model conceptual scenarios and then pruning is exercised as a way of scenario modeling. Finally, geofence SES metamodel is created and later pruned to define a specific model.

There are currently two effort going on for formal modeling of simulation scenarios in aviation: using Eclipse Modeling Framework proposed by [Jafer et al. \[2017, 2016\]](#), and using SES proposed by [Durak et al. \[2018b, 2017\]](#). Both of the approaches yield to a unified computational representation based on XML Schema proposed by [Jafer et al. \[2018\]](#) which the standardization effort aims to reach. The tools proposed in this thesis enabled us to demonstrate the approach with use cases. These use cases will eventually

lead to a draft standard. After a successful standardization process, the industry can proceed with using general purpose XML tools or developing specific editors.

The tool suite presented in this thesis implements the methodology and technology presented in [Durak et al. \[2018b, 2017\]](#). It complements and even completes the effort on proposing a formal approach for developing a scenario definition language ([Chandra Karmokar et al. \[2019\]](#)). Though the tool suite added lots of benefits to the modeling and simulation community it is still not scalable. It can not maintain SES hierarchy right now. Our future plan is to make the developed editors scalable so that it will be able to handle large SES model. SES hierarchy maintaining capability will be added for better modeling environment. Also text to SES model transformation system will be implemented using natural language processing to automate the model creation.

# Appendix A

## Class Descriptions of Editors

### A.1 Classes of core package

- **About.java:** This class implements the about window of the editor. Information about the editor such as editor name, its version, developer name and the supervisor name are displayed on the window.
- **Console.java:** Creates an console like window and during validation shows all the outputs and errors here. Also it takes some input as a command and performs accordingly.
- **Constraint.java:** Creates an window to show added constraint in the SES model. The window contains a table and constraint of the selected aspect node is displayed in that table.
- **CustomIconRenderer:** To identify System Entity Structure elements uniquely different icons are used. CustomIconRenderer class is used to set the icon of the elements based on the name of the elements.
- **CustomIconRendererProject:** To change the icon of the project window tree specific icons are used. This class is used to set the folder icon and xml file icon for the project files.
- **DynamicTree:** This class handles all the activities related to JTree. It also helps to synchronized the Jtree with the tree from the graphical tree builder. The created tree in the graphical drawing paenl is displayed as a JTree format using this class.
- **EditorUndoableEditListener:** This class implements the undo and redo options of the JTree.

- **FileConversion:** This class is used to convert files to desired formats which are used in the project for various purposes.
- **FindByName:** This class used find the path of the selected node from the root node.
- **FlagVariables:** This class contains those variables which are used for counting. For storing these variables value as an object this FlagVariable class is used. As a result when the saved project will be opened then the count will start from the previously saved number.
- **GraphWindow:** This class is used as the drawing panel. SES Model is drawn in this window panel. This class inherits JInternalFrame to display drawing panel within another window.
- **ProjectTree:** This class is used to implements to show project file structure as JTree format. Current project name and its added modules are displayed with different icon in this tree.
- **UndoableTreeModel:** DefaultTreeModel implementation that supports undo and redo the javax.swing.undo package. In order to take advantage of the undo/redo support, all changes to the model should be performed using the insertNodeInto, removeNodeFromParent and valueForPathChanged methods.
- **Variable:** Creates an window to show added variables in the SES model. The window contains a table and variables of the selected node are displayed in that table.
- **XmlJTree:** This class is used to transform XML file into JTree model. During opening existing project an XML document is used and based on that document a JTree model is set to the current tree model using this class.
- **XMLViewer:** This class inherits JInternalFrame because it is using within another frame. XMLViewer is used to display XML instance of the created SES model. XML Schema of the model is also displayed in this viewer. Console window also use this viewer to display errors.

## A.2 Classes of seseditor package

- **GraphCellPopUp:** This class implements right click action of the mouse on the node in the graphical editor. It initiates all the mouse actions such as add variable, delete variable, rename, delete node etc which are used in the graphical editor for modifying SES nodes.



- **GraphPopup:** This class implements right click action of the mouse on the graphical editor panel. It initiates mouse actions related to various elements node addition on the panel.
- **ImportProject:** This class implements Import Project Window for importing SES project created in other SES IDE and saved as XML file following a certain format.
- **JtreeToGraph:** This class is the base of graphical drawing editor panel. JGraphX is used here to implement the edge-vertex connection. Functionalities of node addition, deletion, variable addition, deletion, constraint addition deletion etc all are implemented in this class. Also synchronization of the drawn tree in the graphic panel and the JTree are also done here in this class.
- **NewProject:** This class implements New Project Window for creating new SES project with configuration like project name, root name and project location.
- **ProjectTreePopup:** This class implements the JTree node pop up action of the project tree. Only delete action is implemented here. The main module can't be deleted. But added module file name can be deleted using this delete functionality.
- **SESEditor:** This is the main class of the SESEditor project and it is handling everything of the Editor. Graphical user interface of the editor is designed in this class. All the menu actions and toolbar actions are implemented in this class.
- **SplashScreen:** This class implements the splash screen of the SESEditor. During starting the SESEditor, few editor and development related information will be displayed for few seconds on a small window and then the main editor window will come.
- **TreePopup:** This class implements the JTree node pop up action of the SES JTree displayed in the left side of the editor. Only few actions are implemented here because all the actions are implemented in graphical panel right click option. Node addition and deletion and variable deletion options are added in the pop pup action of the SES JTree.

### A.3 Classes of schema package

- **TypeInfoWriter:** Provides a trace of the schema type information for elements and attributes in an XML document. XML instance of the created SES model is validated against an SES XML Schema using this class. Validation result is displayed in the console window of the editor as human readable format.

## A.4 Classes of peseditor package

- **GraphCellPopUpPruning:** This class initiates all the pruning operation of the editor. By right clicking on the selected node prune option can be found. Based on the node type automatically node specific prune operation will be executed.
- **JtreeToGraph:** This class is the base of graphical drawing editor panel. JGraphX is used here to implement the edge-vertex connection. Pruning operation which are initiated in GraphCellPopUpPruning class are implemented here. Only pruning is allowed here.
- **PESEditor:** This is the main class of the PESEditor project and it is handling everything of the Editor. Graphical user interface of the editor is designed in this class. All the menu actions and toolbar actions are implemented in this class.
- **SplashScreen:** This class implements the splash screen of the PESEditor. During starting the PESEditor, few editor and development related information will be displayed for few seconds on a small window and then the main editor window will come.

## A.5 Classes of xslt package

- **XsltTrasfromation:** This class implements the transformation of pruned scenario XML instance into into Executable Scenario. During conversion it uses manually coded XSLT file for input and transformation format.

## Appendix B

# Menus of Editors

### B.1 Menus of SESEditor

#### B.1.1 Menus

To provide a familiar and consistent user experience menus are added in SESEditor. It has File and Help menus. File menu contains all the common functionality and help menu contains editor information and manuals. Figure B.1 and B.2 shows the File and Help menus.

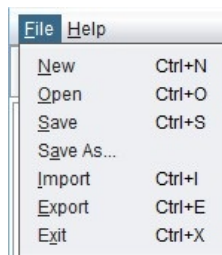


FIGURE B.1: File Menus.

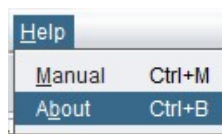


FIGURE B.2: Help Menus.

##### B.1.1.1 File: New

To create a new project has to select New from File menu. A window like figure B.3 will pop up for new SES project configuration. Also keyboard action Ctrl+N will bring

this window quickly. Project name should be unique in saved location otherwise it will overwrite the existing project. If there exist a project with the given name then a warning will show in red color in the blank space. Thing is the default name of the tree root. It can be changed to any name by deselecting the check box and writing new root name. Default location is where the jar file of the editor is placed. But project file location can be changed by deselecting check box and selecting new location by browsing desired location.

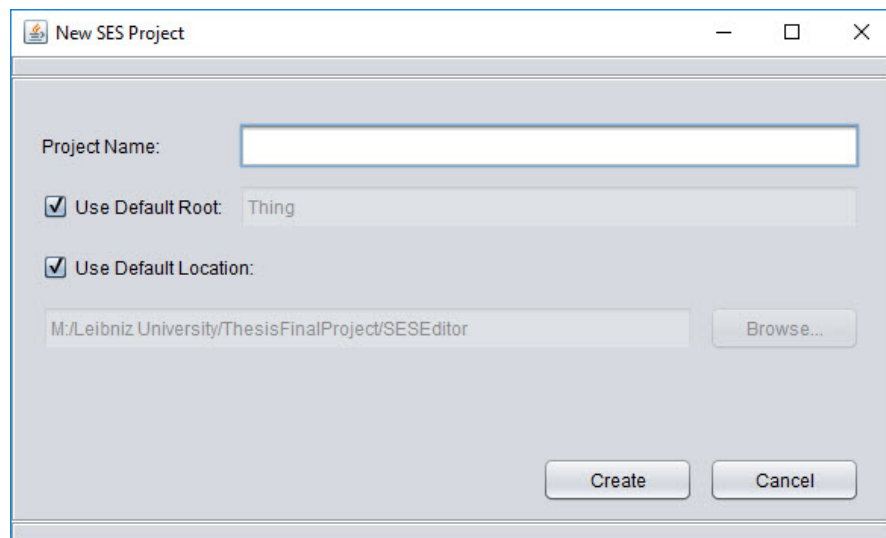


FIGURE B.3: New Project Window.

#### B.1.1.2 File: Open

For opening existing project have to select Open from File. Also keyboard action Ctrl+O will bring this window quickly. Default location is where the jar file of the editor is placed. Projects can be opened by browsing different locations also.

#### B.1.1.3 File: Save

To save the created project have to select Save from File. The easiest way to do this is pressing traditional Ctrl+S keyboard buttons. The project will be saved in the default location.

#### B.1.1.4 File: Save As...

To save the created project in different name have to select Save As... from File. Also project can be saved in different location by browsing the location. Once done then the editor current work space will be changed to newly saved project location.

#### **B.1.1.5 File: Import**

This is one of the cool feature of the editor. Project created in other editor can be imported here. Importing file structure is XML. Importing file should follow the structure of SES XML presented in figure 3.11. Ctrl+I is the shortest way to bring the import window. Selecting Import from File will also bring the same window.

#### **B.1.1.6 File: Export**

Created project can be exported also for other editor. Exported project saved as XML file. Here also file should follow the structure of SES XML presented in figure 3.11. To save the project for exporting have to select the Export from File and then by writing file name this export can be done. Ctrl+E is the shortest way to bring the export window.

#### **B.1.1.7 File: Exit**

To close the editor have to select Exit from File menu. Ctrl+X is the shortest way to close the editor.

#### **B.1.1.8 Help: Manual**

To read the editor manual have to select Manual from Help menu. Ctrl+M will open the manual quickly in the default PDF Reader of the operating system.

#### **B.1.1.9 Help: About**

To know editor information have to select About from Help menu. Ctrl+B will bring the About window quickly.

## **B.2 Menus of PESEditor**

### **B.2.1 Menus**

To provide a familiar and consistent user experience menus are added in PESEditor. It has File and Help menus. File menu contains all the common functionality and help menu contains editor information and manuals. Figure B.4 and B.2 shows the File and Help menus.

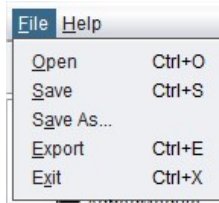


FIGURE B.4: File Menus.

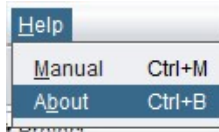


FIGURE B.5: Help Menus.

### B.2.1.1 File: Open

For opening existing project have to select Open from File. Also keyboard action Ctrl+O will bring this window quickly. Default location is where the jar file of the editor is placed. Projects can be opened by browsing different locations also.

### B.2.1.2 File: Save

To save the created project have to select Save from File. The easiest way to do this is pressing traditional Ctrl+S keyboard buttons. The project will be saved in the default location.

### B.2.1.3 File: Save As...

To save the created project in different name have to select Save As... from File. Also project can be saved in different location by browsing the location. Once done then the editor current work space will be changed to newly saved project location.

### B.2.1.4 File: Export

To produce executable scenario form the pruned entity structure have to export the file using target simulation environment specific XSLT file. To do this have to select Export from File menu and browse specific XSLT file and save it. Ctrl+E is the shortest way to bring that XSLT file selection pop up window.

**B.2.1.5 File: Exit**

To close the editor have to select Exit from File menu. Ctrl+X is the shortest way to close the editor.

**B.2.1.6 Help: Manual**

To read the editor manual have to select Manual from Help menu. Ctrl+M will open the manual quickly in the default PDF Reader of the operating system.

**B.2.1.7 Help: About**

To know editor information have to select About from Help menu. Ctrl+B will bring the About window quickly.

# Bibliography

- Alder, G. (2007). Jgraphx: a java swing diagramming (graph visualisation) library. Retrieved October 05, 2018, from [https://jgraph.github.io/mxgraph/docs/manual\\_javavis.html](https://jgraph.github.io/mxgraph/docs/manual_javavis.html).
- Apache Software Foundation (2008). Xerces-c++ XML parser. Retrieved October 01, 2018, from <https://xerces.apache.org/xerces-c>.
- Apache Software Foundation (2010). Xerces2 java XML parser. Retrieved October 01, 2018, from <http://xerces.apache.org/xerces2-j>.
- Bederson, B. B., Grosjean, J., and Meyer, J. (2004). Toolkit design for interactive structured graphics. *IEEE Transactions on software engineering*, 30(8):535–546.
- Brambilla, M., Cabot, J., and Wimmer, M. (2012). Model-driven software engineering in practice. *Synthesis Lectures on Software Engineering*, 1(1):1–182.
- Chandra Karmokar, B., Durak, U., Jafer, S., Chhaya, B. N., and Hartmann, S. (2019). Tools for scenario development using system entity structures. In *AIAA Scitech 2019 Forum*, page 1712.
- Cheon, S., Kim, D., and Zeigler, B. P. (2008). System entity structure for xml meta data modeling; application to the us climate normals. In *SEDE*, pages 216–221.
- Deatcu, C., Folkerts, H., Pawletta, T., and Durak, U. (2018). Design patterns for variability modeling using ses ontology. In *Proceedings of the Model-driven Approaches for Simulation Engineering Symposium*, page 3. Society for Computer Simulation International.
- Durak, U., Jafer, S., Beard, S. D., Reardon, S., Murphy, J. R., Crider, D. A., Gerretsen, A., Lenz, H., Macchiarella, N. D., Rigby, K. T., et al. (2018a). Towards a standardization for simulation scenario development in aviation-panel discussion. In *2018 AIAA Modeling and Simulation Technologies Conference*, page 1395.



- Durak, U., Jafer, S., Wittman, R., Mittal, S., Hartmann, S., and Zeigler, B. P. (2018b). Computational representation for a simulation scenario definition language. In *2018 AIAA Modeling and Simulation Technologies Conference*, page 1398.
- Durak, U., Pruter, I., Gerlach, T., Jafer, S., Pawletta, T., and Hartmann, S. (2017). Using system entity structures to model the elements of a scenario in a research flight simulator. In *AIAA Modeling and Simulation Technologies Conference*, page 1076.
- Durak, U., Topçu, O., Siegfried, R., and Oguztuzun, H. (2014). Scenario development: A model-driven engineering perspective. In *2014 International Conference on Simulation and Modeling Methodologies, Technologies and Applications (SIMULTECH)*, pages 117–124. IEEE.
- Futrell, R. (2008). Rsyntextextarea: A syntax highlighting text component. Retrieved October 10, 2018, from <http://bobbylight.github.io/rsyntextextarea>.
- Gao, S., Sperberg-McQueen, C. M., Thompson, H. S., Mendelsohn, N., Beech, D., and Maloney, M. (2009). W3c xml schema definition language (xsd) 1.1 part 1: Structures. *W3C candidate recommendation*, 30(7.2):16.
- Google (2011). Goava: an open source set of common libraries for java. Retrieved October 10, 2018, from <https://github.com/google/guava/wiki>.
- Gosling, J. (1995). Java: Multi-paradigm: object-oriented (class-based), structured, imperative, generic, reflective and concurrent programming language. Retrieved October 01, 2018, from [https://en.wikipedia.org/wiki/java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/java_(programming_language)).
- Group, M. S. D. L. P. D. (2008). Standard for: Military scenario definition language (msdl). Technical Report SISO-STD-007-2008, Simulation Interoperability Standards Organization (SISO), Orlando, FL.
- Heer, J. (2011). Prefuse: a set of software tools for creating rich interactive data visualizations. Retrieved October 05, 2018, from <http://prefuse.org>.
- Jafer, S., Chhaya, B., and Durak, U. (2017). Graphical specification of flight scenarios with aviation scenario definition language (asdl). In *AIAA Modeling and Simulation Technologies Conference*, Grapevine, TX.
- Jafer, S., Chhaya, B., Durak, U., and Gerlach, T. (2016). Formal scenario definition language for aviation: Aircraft landing case study. In *AIAA Modeling and Simulation Technologies Conference*, Washington, DC.
- Jafer, S., Chhaya, B., Updegrove, J., and Durak, U. (2018). Schema-based ontological representations of a domain-specific scenario modeling language. *Journal of Simulation Engineering*, 1.

- Joshua O'Madadhain, Danyel Fisher, T. N. (2010). Jung — java universal network/graph framework. Retrieved October 05, 2018, from <http://jung.sourceforge.net/index.html>.
- Kay, M. (2008). Saxon9.9: An xml schema processor with the support for both xsd 1.0 and xsd 1.1. Retrieved October 01, 2018, from <https://www.saxonica.com/welcome/welcome.xml>.
- Kim, T.-G., Lee, C., Christensen, E. R., and Zeigler, B. P. (1990). System entity structuring and model base management. *IEEE Transactions on Systems Man and Cybernetics*, 20(5):1013–1024.
- Microsoft (2000). C#: A structured, imperative, object-oriented, event-driven, task-driven, functional, generic, reflective and concurrent programming language. Retrieved October 01, 2018, from [https://en.wikipedia.org/wiki/c\\_sharp\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/c_sharp_(programming_language)).
- Park, H. C., Lee, W. B., and Kim, T. G. (1997). Rases: A database supported framework for structured model base management. *Simulation Practice and Theory*, 5(4):289–313.
- Pawletta, T., Pascheka, D., and Schmidt, A. (2015). System entity structure ontology toolbox for matlab/simulink: Used for variant modelling. In *F. Breiteneker, I. Troch eds., Proc. of MATHMOD 2015-8th Vienna Int. Conf. on Mathematical Modelling, Vienna, Austria, 2015, 2 pages (submitted for publishing)*.
- Pawletta, T., Schmidt, A., Zeigler, B. P., and Durak, U. (2016). Extended variability modeling using system entity structure ontology within matlab/simulink. In *Proceedings of the 49th Annual Simulation Symposium*, page 22. Society for Computer Simulation International.
- Salas, M. C. (2008). Autodevs: a methodology for automating systems development. *Electrical and Computer Engineering Dept., University of Arizona*.
- Santucci, J.-F., Capocchi, L., and Zeigler, B. P. (2016). System entity structure extension to integrate abstraction hierarchies and time granularity into devs modeling and simulation. *Simulation*, 92(8):747–769.
- Schmidt, A., Durak, U., and Pawletta, T. (2016). Model-based testing methodology using system entity structures for matlab/simulink models. *Simulation*, 92(8):729–746.
- Schütte, S. (2011). A domain-specific language for simulation composition. In *ECMS*, pages 146–152.

- Siegfried, R., Laux, A., Rother, M., Steinkamp, D., Herrmann, G., Lüthi, J., and Hahn, M. (2012). Scenarios in military (distributed) simulation environments. In *Spring Simulation Interoperability Workshop (S-SIW)*.
- Siegfried, R., Oguztüzün, H., Durak, U., Hatip, A., Herrmann, G., Gustavson, P., and Hahn, M. (2013). Specification and documentation of conceptual scenarios using base object models (boms). In *Spring Simulation Interoperability Workshop*. Simulation Interoperability Standards Organization (SISO) San Diego, CA.
- Stroustrup, B. (1985). C++: A multi-paradigm programming language: procedural, object oriented and generic. Retrieved October 01, 2018, from <https://en.wikipedia.org/wiki/C%2B%2B>.
- W3C (2010). Xml path language (xpath) 2.0 (second edition). Retrieved October 10, 2018, from <https://www.w3.org/tr/xpath20>.
- Zeigler, B. (1984). *Multifaceted modelling and discrete event simulation*. Academic Press.
- Zeigler, B. P. and Hammonds, P. E. (2007). *Modeling and simulation-based data engineering: introducing pragmatics into ontologies for net-centric information exchange*. Elsevier.
- Zeigler, B. P., Praehofer, H., and Kim, T. G. (2000). *Theory of modeling and simulation: integrating discrete event and continuous complex dynamic systems*. Academic Press.
- Zeigler, B. P., Seo, C., Coop, R., and Kim, D. (2013). Creating suites of models with system entity structure: global warming example. In *Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium*, page 32. Society for Computer Simulation International.