
Development of an Overture/VDM++ Tool Set for Eclipse

Jacob Porsborg Nielsen, s001722
Jens Kielsgaard Hansen, s001842

Kgs. Lyngby, August 15, 2005
M.Sc. Project
IMM-THESIS-2005-58

IMM

Informatics and Mathematical Modelling
Technical University of Denmark

Abstract

In this project a kernel for an Overture Tool Set supporting OML (Overture Modelling Language) has been developed. OML is very similar to the formal specification language VDM++. The Overture Tool Set is based on the Eclipse framework, which means that the tools integrate with an Eclipse based editor. The kernel provides functionality for parsing an OML specification and storing the information in an AST (Abstract Syntax Tree), reconstructing source code from the AST, and importing and exporting this AST representation to XML. The kernel is extensible so that further functionality can be added to the Overture Tool Set without changing the kernel implementation. This feature is implemented using the plug-in structure of Eclipse and Visitor Design Patterns. Furthermore, three 'proof of concept' plug-ins have been developed – one for exporting a simple OML specification to an UML class diagram, one for importing a simple UML class diagram to OML, and one to show that the kernel can handle refactoring of an AST. The report documents analysis, design, implementation, test, and how the kernel can be extended.

Keywords Overture, OML, VDM++, Eclipse, tool set, kernel, parser, AST, XML.

Resumé

I dette projekt er kernen til et værktøjssæt, Overture Tool Set, til sproget OML (Overture Modelling Language) blevet udviklet. OML ligner meget det formelle specifikationssprog VDM++. Værktøjssættet er bygget til Eclipse platformen, så værktøjerne er integreret med en Eclipse baseret editor. Kerne- neren tilbyder funktionalitet til at parse en OML specifikation og opbygge et AST (Abstrakt Syntaks Træ), gendanne kildekoden fra et AST, samt mulighed for at eksportere og importere et AST til/fra XML. Kerne- neren er opbygget, så den er let at udvide, idet værktøjssættet kan udbygges med yderligere funktionalitet uden at ændre implementeringen af kernen. Dette er muligt gennem anvendelse af Eclipses plug-in koncept og Visitor Design Patterns. Derudover er tre 'proof of concept' plug-ins blevet udviklet – et til at exportere en simpel OML specifikation til et UML klassediagram, et til at importere et simpelt UML klassediagram til OML, og et til at vise at ker- nen kan håndtere 'refactoring' af et AST. Rapporten dokumenterer analyse, design, implementering, test, samt hvordan kernen kan udbygges.

Nøgleord Overture, OML, VDM++, Eclipse, værktøjssæt, kerne, parser, AST, XML.

Preface

This report documents the M.Sc. thesis project of Jacob Porsborg Nielsen and Jens Kielsgaard Hansen. The project has been carried out in the period from January 25th 2005 to August 15th 2005, at the Technical University of Denmark, Department of Informatics and Mathematical Modelling, the Computer Science and Engineering Division.

The project has been supervised by Associate Professor, Ph.D. Anne E. Haxthausen and Associate Professor Hans Bruun. The external supervisor has been Ph.D. Peter Gorm Larsen, now Associate Professor at the University College of Aarhus.

We would like to thank our supervisors for their interest and enthusiasm in our project, and for their constructive suggestions through the entire process. Also great thanks to the Overture Core Group – you have been most helpful and brought many constructive ideas to us at the monthly net-meetings. Finally we want to thank the association Formal Methods Europe (FME) for funding support, that enabled us to present our project at the Overture Workshop in Newcastle, July 18th 2005.

Kgs. Lyngby, August 12, 2005

Jacob Porsborg Nielsen
s001842

Jens Kielsgaard Hansen
s001722

Contents

1	Introduction	1
1.1	Overview of the Report Structure	1
1.2	Reading Guidelines	2
2	Background and Motivation	3
2.1	The OML Language	3
2.2	Tool Support for OML	5
2.3	Participation in Overture Open Source Project	5
3	Clarifying the Problem	7
3.1	Defining the Purpose of the Kernel	7
3.2	Scope of the Project	8
4	Eclipse	11
4.1	General about Eclipse	11
4.2	Plug-ins	11
4.2.1	Defining a Plug-in	12
4.2.2	Extension Points and Extensions	12
4.2.3	Dependencies	13
4.2.4	Plug-in Development	13
4.3	Update Sites – Distribution of Plug-ins	13
4.4	Provided Facilities by the Eclipse Framework	14
4.4.1	Extending the Eclipse Framework	14
4.4.2	Editors	15
4.4.3	Perspectives and Views	15
4.4.4	Dialogs and Wizards	16
4.4.5	Preferences	16
4.4.6	Markers	16
4.4.7	Resources	18
4.4.8	Natures and Builders	18
4.4.9	Concurrency / Jobs	18
4.4.10	Help	18

CONTENTS

5	Theory	19
5.1	Defining the Syntax of a Language	19
5.2	Trees representing Languages	20
5.3	Theory on AST	21
5.4	Theory on Visitor Design Pattern	21
5.5	Parser Theory	23
5.5.1	Bottom Up vs. Top Down Parsing	23
5.5.2	Top Down Parsing Algorithms	24
5.5.3	Fixed Lookahead (Recursive Decent Parsing)	25
5.5.4	Dynamic Lookahead (Recursive Decent Parsing)	25
6	Analysis	27
6.1	AST Structure	27
6.2	Construction of AST Classes	28
6.2.1	Construction of AST Classes using a Parser Grammar File	29
6.2.2	Construction of AST Classes using an XML Schema	29
6.2.3	Construction of AST Classes using an UML Tool	29
6.2.4	Construction of AST Classes by Hand	30
6.2.5	Summary	30
6.3	Construction of the Parser	30
6.3.1	Bottom Up and Top Down Parser	30
6.3.2	ANTLR or JavaCC	31
6.3.3	Summary	31
6.4	XML Facilities	31
6.4.1	Exporting an AST to XML	31
6.4.2	Importing an AST from XML	32
6.4.3	Summary	32
6.5	Applicable Eclipse Facilities	32
6.6	Summary	33
7	Design of the Overture Kernel	35
7.1	General Design Considerations	35
7.2	Dividing the Functionality into Plug-ins and Packages	35
7.3	Design of the AST	37
7.4	Design of the Parser	37
7.4.1	Design of Comments Handling	39
7.4.2	Precedence Principles	40
7.5	Design of the XML Schema	40
7.6	Designing the General Visitors	42
7.7	Designing the Pretty Print Visitor	42
7.8	Designing the AST to XML and Outline Visitors	43
7.9	Design of the XML to AST Converter	43
7.10	Design of the Editor	43

7.10.1	Editor – Wizards	44
7.11	Design of Extension Possibilities	46
7.12	Summary	47
8	Implementation	49
8.1	General Implementation Issues	49
8.2	Implementation of the AST Classes	49
8.2.1	Structure in General	50
8.2.2	Setting the Positions	50
8.2.3	Handling Comments	51
8.2.4	Handling Semicolons	52
8.2.5	Stability using Java Generics	52
8.2.6	AST Classes Special Case – Multiple Inheritance	53
8.3	Implementation of the Parser	53
8.3.1	Specifying the Grammar	54
8.3.2	Error Handling	54
8.3.3	Building an AST	54
8.3.4	Handling Unicode Characters	54
8.3.5	Handling Comments	55
8.3.6	Handling Precedence	56
8.3.7	Grouping of Binary Expressions	57
8.4	Operating on the AST	58
8.4.1	Implementation of the AST to XML Visitor	59
8.4.2	Implementation of the AST to Outline Visitor	62
8.4.3	Implementation of the AST to OML Visitor	62
8.5	Implementation of the XML to AST Converter	63
8.6	Implementing Installation Facilities	64
8.7	Implementation of the Integration with the Eclipse Framework	65
8.8	Implementation of the Overture Editor	66
8.9	Extending the Kernel Plug-in	68
8.10	Summary	69
9	Test	71
9.1	Functional Test of the Parser	71
9.1.1	How the Tests are Performed	72
9.1.2	Test of Part1.oml	73
9.1.3	Test of Part2.oml	73
9.1.4	Test of Part3.oml	75
9.1.5	Test of Part4.oml	76
9.2	Functional Test of Import and Export Facilities	78
9.2.1	How the Tests are Performed	78
9.2.2	Test of the Eclipse Functionality	79
9.3	Performance Issues	80
9.4	Summary	80

CONTENTS

10 Additional Created Plug-ins	81
10.1 Refactoring	81
10.1.1 Analysis	81
10.1.2 Design and Implementation	82
10.1.3 Test	82
10.2 Import and Export to UML	82
10.2.1 Analysis	83
10.2.2 Design and Implementation	83
10.2.3 Test	85
10.3 Conclusion	86
11 Future Work	87
11.1 Guide to Extending the Kernel	87
11.1.1 New operator	87
11.2 Ideas for Future Plug-ins and Eclipse Features	87
11.3 Approaches to use Formal Specification for Future Development	89
12 Conclusion	91
12.1 Achieved Results	91
12.2 Discussion of Results	92
12.3 Concluding Remarks	92
Bibliography	96
Appendices	97
A Term List and Abbreviations	97
B CD Contents Guide	101
C Installation Guide	105
C.1 Installation of Required Tools	105
C.2 Installation of the Overture Kernel	105
C.3 Getting Started	106
C.4 Installation of Source Code from a Kernel Release	107
C.5 Installation of the Full Source Code	107
C.6 How to Access the Provided Test Cases	108
D Project Description	109
D.1 Danish	109
D.2 English	109

E	Overture Workshop in Newcastle Upon Tyne	111
E.1	Attendance in Workshop	111
E.2	Contribution to Technical Report	111
F	Overview of Precedence and Grouping Conventions	119
F.1	Precedence Overview – Expressions	120
F.2	Precedence Overview – Type Operators	122
G	Structured Decision Making	123
G.1	Choosing Parser Generation Method	123
H	Modifying the OML language	129
H.1	Adding a operator to the OML language	129
I	Source Code - Selected Samples	131
I.1	ANTLR Grammar File	133
I.2	Selected AST Classes and Interfaces	219
I.2.1	ASTNode	219
I.2.2	InternalASTNode	219
I.2.3	ASTNodeWithComments	221
I.2.4	InternalASTNodeWithComments	221
I.2.5	ASTKeyword	222
I.2.6	InternalASTKeyword	222
I.2.7	ASTIdentifier	223
I.2.8	InternalASTIdentifier	223
I.2.9	ASTDocument	224
I.2.10	InternalASTDocument	224
I.2.11	ASTClass	225
I.2.12	InternalASTClass	225
I.2.13	ASTExpression	226
I.2.14	InternalASTExpression	226
I.2.15	ASTBinaryExpression	227
I.2.16	InternalASTBinaryExpression	227
I.2.17	ASTBinaryOperator	228
I.2.18	InternalASTBinaryOperator	228
I.2.19	ASTBinaryOperatorArithmeticPlus	228
I.2.20	InternalASTBinaryOperatorArithmeticPlus	229
I.3	Visitor Interface	231
I.4	Outline Visitor	237
I.5	Ast2Oml Pretty Print Visitor	241
I.6	XML Schema – Sample of the XML Schema	243
I.7	Selected Editor Files	247
I.7.1	Ast2XmlAction.java	247
I.7.2	Xml2AstAction.java	249

CONTENTS

I.7.3	OvertureContentProvider.java	251
I.7.4	OvertureContentOutlinePage.java	255
I.8	Export/Import to/from UML	257
I.8.1	XML to XMI/UML Conversion	257
I.8.2	XMI/UML to XML Conversion	263
J	Tests	265
J.1	Test – Part2.oml	265

List of Figures

3.1	Overview of what the kernel should include	7
4.1	Overview of the the Eclipse GUI	15
4.2	Example of a customized wizard	17
5.1	Sequence diagram: Example of visitor interaction	22
5.2	Illustration of top down parsing	24
7.1	Overview of dependencies	36
7.2	Example of selected AST classes based on inheritance and interfaces	38
7.3	Example of an AST with inheritance	39
7.4	Filtering comments - Filter between lexer and parser	39
7.5	Filtering comments - Comments stored in hidden linked list	40
7.6	XML Schema example (XML-Spy diagram notation)	41
7.7	XML Schema example (How comments are stored)	41
7.8	Eclipse wizard overview - Class diagram	44
7.9	Eclipse editor overview - The most important use of classes	45
7.10	Eclipse action overview - Class diagram	46
7.11	Eclipse editor overview - Loading classes for extensions	47
9.1	Outline view showing the structure of selected parts of an AST.	78
9.2	Outline view showing the structure of selected parts of an AST.	79
10.1	Example of an UML class diagram	83
10.2	Overview of how to convert between OML and UML.	84
B.1	Cd-rom contents	102
E.1	Overture Workshop Report, Page 1	112
E.2	Overture Workshop Report, Page 2	113
E.3	Overture Workshop Report, Page 3	114
E.4	Overture Workshop Report, Page 4	115
E.5	Overture Workshop Report, Page 5	116
E.6	Overture Workshop Report, Page 6	117

LIST OF FIGURES

E.7 Overture Workshop Report, Page 7 118

Chapter 1

Introduction

This M.Sc. project is part of a larger open source project called Overture[13]. The Overture project aims at developing an industrial strength tool for precise abstract models in software development. The idea is to make it easy to add and alter the functionality of the tool. The tool should support the OML language (Overture Modelling Language). OML is similar to the formal specification language VDM++ (Vienna Development Method) as defined by CSK[12]. The overture project has though intensions of future modifications of the language, therefore the term OML is used as the name of the supported language. The goal for this project is to develop a well-designed kernel for the Overture Tool Set. The kernel should implement the basic functionalities and be easy to extend.

This report analyzes available tools and technologies suitable for developing a kernel for the Eclipse based Overture Tool Set. Eclipse is described in Chapter 4. The report then documents the choices we have made and how the kernel is designed, implemented, and tested. The official project description of the Thesis Project can be found in Appendix D. The implementation is done using Java 5.0, and the produced kernel is integrated with the Eclipse development environment.

1.1 Overview of the Report Structure

First the background and motivation for this project is given in Chapter 2. Chapter 3 gives an overview of the task to be solved and specifies the requirements. A short introduction to Eclipse is given in Chapter 4. Then in Chapter 5 there are explanations of theories relevant for the project. Analysis of solution strategies and applicable tools are given in Chapter 6. Design issues and principles are discussed in Chapter 7, and important aspects of the implementation is described in Chapter 8. Chapter 9 explains how the kernel is tested. Chapter 10 presents some additional plug-ins we have designed and implemented for the kernel, whereas Chapter 11 outlines how

the kernel can be improved and extended with new functionalities. Finally, Chapter 12 concludes what has been achieved in this project.

A set of appendices provides additional information. Appendix A defines terms and abbreviations used in this report. Appendix B gives an overview of the content of the cd-rom handed in with the report. In Appendix C a guide is given on how to install the kernel and how to obtain the source code. The official project descriptions for the M.Sc. thesis project are shown in Appendix D. The contribution to the technical report at the Overture workshop can be found in Appendix E. How the kernel implements precedence and grouping is listed in Appendix F. The choice of parser generation tool is documented in Appendix G. An overview of how to extend the OML language is given in Appendix H.

Selected parts of the source code can be found in Appendix I, whereas some of the test examples are given in Appendix J. Please note that only selected parts of the source code and tests are in appendix – the entire source code and test case suite is available on cd-rom, as described in Appendix B.

In the report we sometimes provide a few lines of code to illustrate the implementation. Some of these source code samples have been simplified to ease readability.

1.2 Reading Guidelines

Different readers of this report will be interested in different aspects of the project. This is an outline of different ways to read this report.

We recommend all readers to read Chapter 2 and Chapter 3, as they gives an overview of this project. Technical terms and abbreviations are defined in Appendix A. If you are unfamiliar with Eclipse, Chapter 4 will give you a basic introduction.

If your interest is in extending or developing this solution, Chapter 7, Chapter 8, Chapter 10 and Chapter 11 are especially important.

If your main focus is to examine the kernel, the methods and techniques applied throughout the project, and the achievements of the project, it will be beneficial to read Chapter 5, Chapter 6, Chapter 7 and Chapter 9. To get an idea of the possibilities of extending the solution, we refer to Chapter 10 and Chapter 11.

We encourage all readers to read the conclusions presented in Chapter 12.

A cd-rom has been made containing the source code for the kernel, the language manual for the VDM++ language, this report, tests, installation guide, and an update site that can be used for installation. A more detailed description of the content of the cd-rom can be found in Appendix B

Chapter 2

Background and Motivation

This chapter describes the background and motivation that led to the creation of this project. The background for having an OML language is defined in Section 2.1. Then the need for tool support for OML is described in Section 2.2. Finally, an explanation is given in Section 2.3 of how this M.Sc. project relates to the overture open source project.

2.1 The OML Language

VDM_SL[5] is a formal specification language used to specify software in an abstract and accurate way, and VDM++[6] is an object oriented extension to this language. After defining the requirements, developers can specify the requirements using VDM++. If the specification is well written, it is unambiguous and makes it easy to implement and test the system afterwards. With the current tool supporting VDM++ (VDMTools), it is possible to auto generate Java code from a VDM++ specification and run test cases directly on the model. Investigations have shown that using methods like this will significantly shorten the development time and the time for testing for big and complex projects. It should, of course, always be considered in which part of a project it is beneficial to use VDM++, but using it in the right way can be very beneficial when writing quality software.

The OML language is intended to be a further development of VDM++. An example of an OML specification defining two OML classes, can be seen in Listing 2.1. More examples can be found in Appendix J.

Listing 2.1: Simple OML example

```
1 class Alarm
2
3 types
4   public String = seq of char;
5
6 instance variables
7
8   descr    : String;
9   reqQuali : Expert'Qualification
10
11 operations
12
13   public Alarm: Expert'Qualification * String ==> Alarm
14   Alarm(quali , str) ==
15   ( descr := str;
16     reqQuali := quali
17   );
18
19   public GetReqQuali: () ==> Expert'Qualification
20   GetReqQuali() ==
21   return reqQuali;
22
23 end Alarm
24
25 class Expert
26
27 instance variables
28
29   quali : set of Qualification
30
31 types
32
33   public Qualification = <Mech> | <Chem> | <Bio> | <Elec>;
34
35 operations
36
37   public Expert: set of Qualification ==> Expert
38   Expert(qs) ==
39   quali := qs;
40
41   public GetQuali: () ==> set of Qualification
42   GetQuali() ==
43   return quali;
44
45 end Expert
```

2.2 Tool Support for OML

Currently there is a commercial tool (called VDMTools) that supports VDM++, but it does not make use of the technologies available today. In a previous project connected to the Overture project, a proof of concept kernel was build[9], but it does not meet the requirements the Overture core group is now requesting. New technologies and principles of constructing tools are now available. By using an IDE (Integrated Development Environment) framework as e.g. Eclipse and by designing modules as plug-ins, it will be possible to integrate the tool with other tools and easy to extend the tool with new facilities. The goal for this project is therefore to produce a well designed kernel supporting the OML language using modern tools and techniques.

2.3 Participation in Overture Open Source Project

This M.Sc. thesis project is a contribution to the Overture open source project. The intension of the project (and the intension of this thesis) is both to serve as a master thesis and to serve as a kernel that the overture project can use as basis for future development.

The overture project is open source and the development is led by a core group that discusses, plans, and co-ordinates development of the overture tool set. Throughout the project, we have discussed design issues with the overture core group in order to ensure that the developed kernel will meet the needs of the Overture project. The cooperation with the overture core group has primarily been through monthly instant messaging net meetings. In July 2005, a workshop was held in Newcastle to discuss and plan the future of the overture project. FME¹ sponsored us, so that this M.Sc. project could be presented at the workshop. It is now the intension of the overture workshop to try to use the developed kernel as base for further development.

Throughout the entire project, it has been important that the project actually would solve the expectations. Therefore it was chosen to develop the kernel in iterations. A small subset of OML was therefore selected, and a kernel was created to support this. We believe that this iterative development approach has helped us to develop a better kernel.

¹Formal Methods Europe, <http://www.fmeurope.org>

Chapter 3

Clarifying the Problem

This chapter clarifies the wishes for and scope of the project. It summarizes what the project should include, and defines the problems the project is intended to solve.

The official project description that was used for registration of this M.Sc. project can be found in Appendix D.

3.1 Defining the Purpose of the Kernel

Part of the project is defining what functionalities the kernel should offer. Figure 3.1 shows an overview of what the kernel should include, based on discussions with the supervisors.

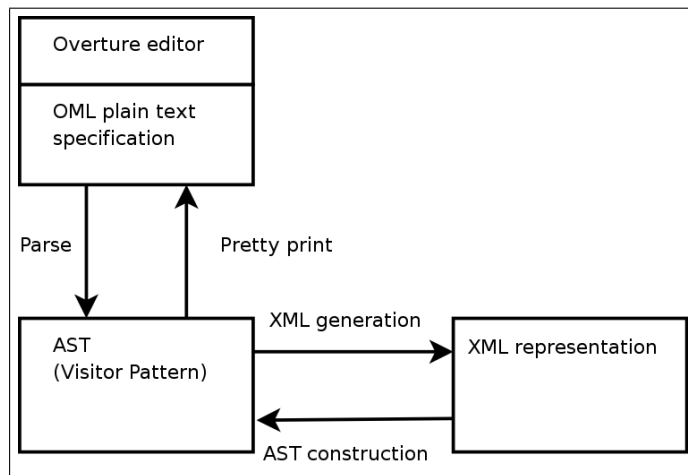


Figure 3.1: Overview of what the kernel should include

The main issue to address is to create a kernel capable of parsing OML specifications. When parsing a specification, an AST (Abstract Syntax Tree)

should be built. The kernel should be designed in a manner, which is easily extendible for the additional tools that need to operate on the AST. The kernel should also provide export and import facilities to/from XML. This is to enable interaction with tools that uses XML for exchange of information. Finally, there should be a facility to create a plain text OML specification from an AST, in case other tools have modified the AST.

Eclipse has been chosen as a suitable framework for the Overture Tool Set. The kernel should be developed for the Eclipse framework, such that both the kernel and future plug-ins can make use of the facilities provided by the Eclipse framework. Eclipse offers a range of facilities that can help to make the kernel flexible and extendible.

To give an overview of the primary needed facilities which the kernel must offer, a list of these facilities is given:

- An Eclipse based editor with editing facilities for OML.
- Parsing an OML plain text specification¹ to an Abstract Syntax Tree (AST). The AST should be implemented to support use of Visitor Design Pattern in order to make it easy for plug-ins to operate on the AST.
- Converting the AST to XML
- Converting XML to AST
- Pretty print from an AST to an OML plain text specification. Other tools/plug-ins may have modified the AST, forcing the kernel to create a fresh OML plain text specification to present to the user.
- The implementation should result in a plug-in for Eclipse.
- It should provide extension points so that new functionality can be added as additional Eclipse plug-ins that extend the kernel.

The listed issues illustrates the initial requirements for the project. In the analysis, design, and implementation of the kernel, these requirements has been used as a base for the development.

3.2 Scope of the Project

It is important to agree on a common understanding of what a project is to solve. Through the initial discussions with the supervisors and the overture core group, a list of statements has been made, that sets the expected scope of the project. These statements are listed below.

¹The phrase 'OML plain text specification' is defined in the term list in Appendix A

- The parser should do syntax checking and find more than one error, but not do contextual analysis.
- The AST classes should be created using inheritance and interfaces in order to make it easy for the plug-in developers to do operations on the tree. The structure of the AST is explained in Section 6.1
- Parsing an OML specification and exporting it to XML should preserve comments made by the user.
- A plug-in that extends the kernel and operates on the AST must be made in order to show the extendability of the kernel.

Chapter 4

Eclipse

This chapter introduces Eclipse. The purpose of the chapter is to give readers who are unfamiliar with Eclipse, some basic knowledge of what Eclipse is. Afterwards the chapter gives a more technical descriptions of the facilities offered by the Eclipse framework, and how these facilities can be used when creating Eclipse based solutions.

4.1 General about Eclipse

Eclipse is a framework for tools. The framework is extendable and can be used as a base for all kinds of tools. The overall intension of Eclipse is to serve as a platform that let tools integrate seamlessly on any platform. A wide range of software companies support Eclipse – including IBM, who were among the initiators of the project. The license used for Eclipse allows it to be used in commercial applications. IBM has developed a commercial development environment, Websphere Studio, which is based on Eclipse.

4.2 Plug-ins

A very central concept of Eclipse is Plug-ins. Basically, Eclipse is nothing but a framework intended to be extended by plug-ins. Though Eclipse is distributed with advanced support for programming in Java, the main purpose of Eclipse is to serve as a basic framework for tool plug-ins. In fact, all Java supporting tools in Eclipse, are ordinary plug-ins themselves. Plug-ins interact with each other and with the Eclipse framework through extension points.

Eclipse ships with plug-ins that provides a Java development environment. The Java Editor is well integrated with the Eclipse framework. It has error handling, debugger, continuous automatic builds, on-the-fly marking of errors while writing code, generation of Java doc and much more. The

architecture and sources of both the Eclipse framework and the Java development environment are publicly available, so that the Java development environment can serve as inspiration for other tool developers.

Eclipse has a built in plug-in development environment. In principle plug-ins can be developed in any tool, but it is recommendable to use the Eclipse plug-in development tool.

4.2.1 Defining a Plug-in

Traditionally, all information about a plug-in is stored in a file called `plugin.xml`. This xml file defines etc. name, version, provider, runtime requirements, dependencies, extensions, and extension points of the plug-in. Using this approach all plug-in formalities are specified in a single file. On installation of a plug-in, Eclipse will have to be re-started.

Recently Eclipse has launched OSGi[22] support, which is a different way to specify the plug-in. With OSGi, the above mentioned information is stored in multiple files. OSGi is a general open standard for distribution and management of services and applications that uses networks. As an Eclipse plug-in developer, the main advantage of using OSGi, is that OSGi based plug-ins can be hot-plugged. Installing an OSGi plug-in will not require the user to reboot Eclipse. It should though be noted that documentation of the OSGi support in Eclipse has been fairly weak, but this is improving since Eclipse 3.1 has now been officially released.

4.2.2 Extension Points and Extensions

If a plug-in provides an extension point, other plug-ins can interact with it by extending this extension point. The information about extension points and extensions is placed in the XML (eXtensible Markup Language) file described in Section 4.2.1. Listing 4.1 is an example of an extension point, and Listing 4.2 is an example of an extension extending this extension point.

Listing 4.1: XML code example of an extension point

```
1 <extension-point id="extensionParser" name="ExtPoint.extensionParser"/>
```

Listing 4.2: XML code example of an extension

```
1 <extension point="org.overturetool.eclipse.editor.extensionParser">
2   <parser
3     name="ParserExtension"
4     class="org.overturetool.eclipse.parser.OvertureParser"
5     id="org.overturetool.eclipse.parser.OvertureParser">
6   </parser>
7 </extension>
```

Listing 4.2 shows that the extension provides a reference to a class called `OvertureParser`. It should be loaded when the plug-in defining the extension point invokes the parser. To give a better performance Eclipse analyzes the XML files and waits to load the extensions until just before they are to be used[3].

If a plug-in provides an extension point, a number of plug-ins can extend it, and this adds flexibility to the solution. This plug and play idea is e.g. useful when several development teams are contributing to the same tool set or if the user should be able to choose between different implementations of a parser.

4.2.3 Dependencies

If a plug-in needs classes from another plug-in in order to work properly, a dependency can be specified. If a dependency is specified for a plug-in A, that it depends on a plug-in B, it tells Eclipse that A cannot operate properly if B is not available. Upon installation, the user will get warnings that dependencies are not fulfilled, if he/she tries to install A without having or installing B.

If different plug-ins have to operate on the same classes, it is a common solution to create a plug-in to host the shared classes. Each of the two plug-ins will then have a dependency to the plug-in with the shared classes.

4.2.4 Plug-in Development

Eclipse provides a development environment for development of plug-ins. This environment provides facilities to help developers create all parts of a plug-in – both Java code and XML files defining the plug-in.

Eclipse offers a GUI for modifying all common parts of a plug-in. This includes facilities to modify run time requirements, version numbers, names, dependencies, extensions, and extension points. In addition, one can make use of many of the general Eclipse facilities. For a plug-in project, it will in be beneficial to use the built in CVS system .

4.3 Update Sites – Distribution of Plug-ins

Eclipse based plug-ins are most commonly distributed though the internet. Eclipse has a built in mechanism for installation and updating of plug-ins, where plug-ins are fetched from update sites on the internet. The plug-ins are automatically exported to jar files and a related feature project is exported to a jar file as well. An update site consists of a set of jar files and some XML documents. In addition, there is a HTML file in case someone tries to access the update site URL through a web browser. In this case, the user

will be shown a web page with the available plug-ins. The files created by the update site need to be exported to a server in order to publish a release.

A feature defines a collection of plug-ins. A feature is represented as an XML file containing information about licensing, the related plug-ins and their versions. Furthermore, it is possible to specify dependencies between plug-ins. This feature information is both used by Eclipse during the installation as well as managing the plug-ins after installation.

When an Eclipse user wants to install a plug-in for Eclipse, the update site must first be added to the list of update sites. Afterwards Eclipse will be able to search the update site location for available plug-ins and the user can then choose which of the offered plug-ins to install. Eclipse then handles the downloading, installation, and possibly rebooting of Eclipse.

4.4 Provided Facilities by the Eclipse Framework

The Eclipse framework provides many facilities and extension points that can be used when creating plug-ins. In the following sections, we will present some central facilities of Eclipse. Most, but not all, of the mentioned features are applied in the implementation of the kernel. For more insight into Eclipse, we recommend [3] as well as the built in documentation and help files.

4.4.1 Extending the Eclipse Framework

Each plug-in has as described in Section 4.2.1 an XML file containing plug-in specific information. This file also defines how to extend the Eclipse Framework and by doing this contribute to the GUI. Listing 4.3 is an example of the information Eclipse needs in order to be aware of a customized perspective. It has a reference to the class that should be executed when opening the perspective. Perspectives are presented in Section 4.4.3.

Listing 4.3: XML code example of perspectives

```
1 <extension
2   point="org.eclipse.ui.perspectives">
3   <perspective
4     name="Overture Perspective "
5     icon="icons/sample.gif "
6     class="org.overturetool.eclipse.editor.OverturePerspective "
7     id="org.overturetool.eclipse.editor.OverturePerspective">
8   </perspective>
9 </extension>
```

After the XML file has been analyzed by Eclipse, the Overture Perspective can be found in the perspective menu. Using a similar principle plug-ins can contribute to e.g. a menu, an editor or a wizard.

4.4. PROVIDED FACILITIES BY THE ECLIPSE FRAMEWORK

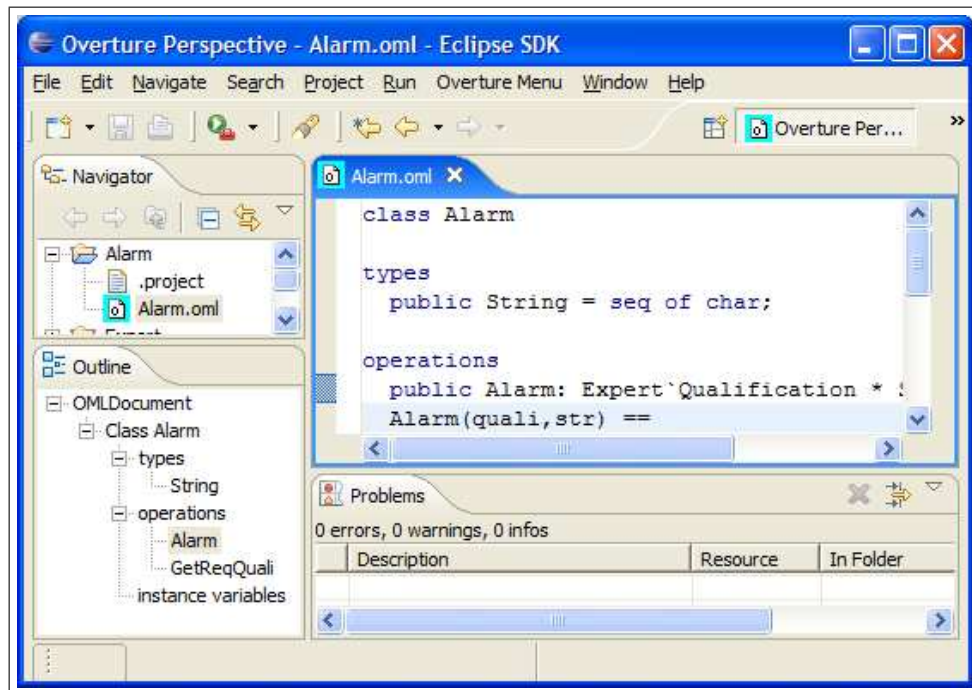


Figure 4.1: Overview of the the Eclipse GUI

Figure 4.1 shows the Overture editor as an example of how an Eclipse based editor can look. The figure shows the navigator view in the upper left corner, the outline view in the lower left corner, the editor area in the middle, the problem view in the bottom, the current perspective in the upper right corner, and finally, the Overture Menu.

4.4.2 Editors

Eclipse offers a standard text editor that can be extended and configured to work as a customized editor for any language. In addition, an editor can be associated with a file extension, such that all files of a specific file format opened in Eclipse will automatically launch the appropriate editor. Typical customizations and extensions to an Eclipse based editor are e.g coloring of keywords, wizards for creation of new files, and generation of different views. In general many views are related to an editor – views will be explained in Section 4.4.3.

4.4.3 Perspectives and Views

Two very central concepts of Eclipse are perspectives and views. When using Eclipse, one perspective will always be open. When choosing to edit e.g. a

Java file, the entire user interface will change so that only Java development relevant facilities are presented to the user. Had the user chosen to explore a CVS repository instead, the tools presented to the user would only be those relevant for this. Such a set of tools for some purpose is called a perspective. Some standard perspectives of Eclipse are the Java development, plug-in development, CVS Repository, and the team synchronizing perspectives.

A perspective consists of views. Views can be really different, such as tree, outline, error log, or properties views. A view typically has a quite specific purpose, e.g. to show identified errors and warnings and provide facilities to go directly to the error. Both views and perspectives are adjustable by the user, as views can be rearranged or hidden.

4.4.4 Dialogs and Wizards

Dialog and wizard facilities are provided by Eclipse to ease interaction with the user. There are several dialogs and wizards available suited for different kinds of interaction with the user. The dialogs are typically used to show some error, warning, or information message, possibly giving the user several answer options. Wizards are easy to customize and can be configured to ask for e.g. some file names and paths. It is possible to add code of own choice to a wizard, which is handy if the wizard is e.g. to make some validation of the entered input. An example of the customized Overture wizard is shown in Figure 4.2.

4.4.5 Preferences

Many tools can be highly customized. Eclipse offers a set of facilities especially targeted to create views for preference settings. There are also facilities for storing preferences, so that they are still set next time Eclipse is used. Preferences can be used for all kinds of settings, typical examples are default paths, default names, checkboxes indicating if some action is to be performed, etc.

4.4.6 Markers

Markers are an Eclipse based concept covering selection and highlighting of text in an editor. Markers can in addition link objects, such that e.g. an error message is linked to highlighting of the related text. A situation where markers are used, is when a problem is shown in the problem view. It is possible to double-click on a problem, which will typically make the editor jump to and highlight the related code. Markers are objects that enables this kind of linking.

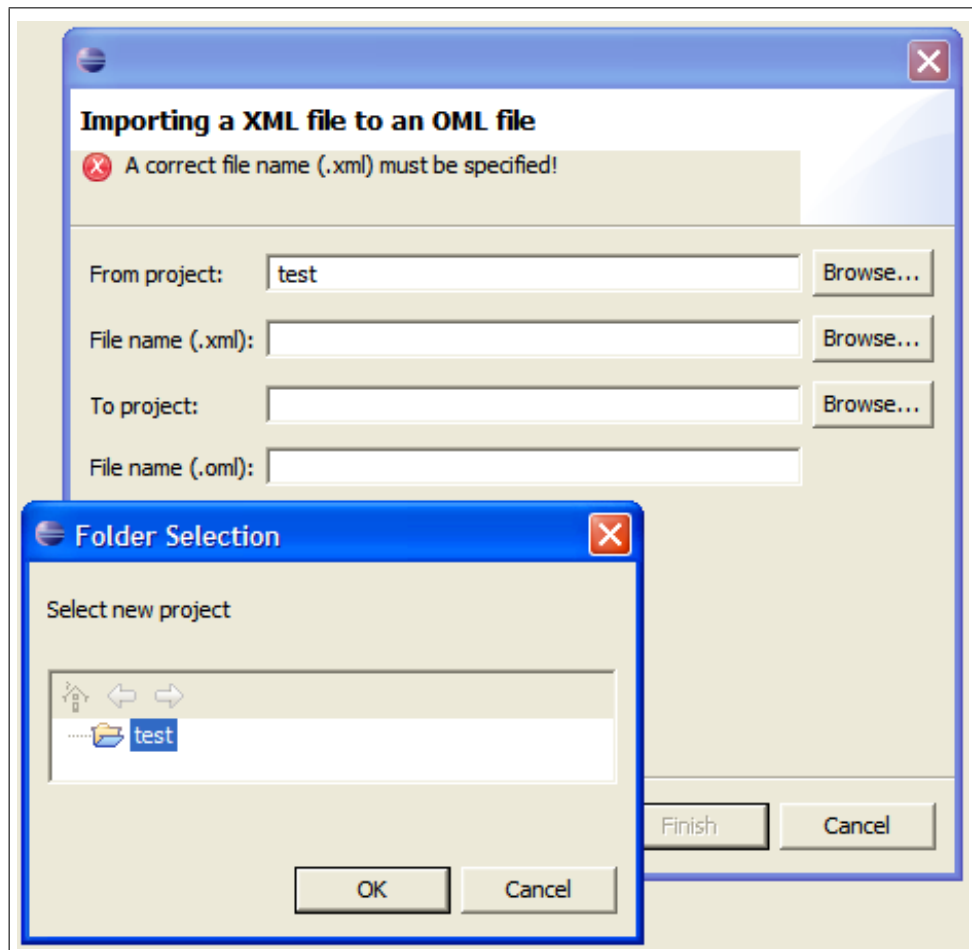


Figure 4.2: Example of a customized wizard

4.4.7 Resources

In Eclipse terminology, a resource is a file or a container, where a container is an Eclipse term for a folder/directory. Eclipse offers a range of resource related features. They can be used when extracting file extension from a file name, when there is need for the path for the workbench directory, or similar.

4.4.8 Natures and Builders

Most Eclipse based tools use parser technology in some form. Eclipse offers advanced mechanisms that can monitor changes of files (and resources in general). If the parser supports it, it is possible to create incremental builders, that only parses the files that has been changed.

4.4.9 Concurrency / Jobs

Jobs is a concept of Eclipse, that represent tasks such as parsing a specifications or storing a file. If a process is defined as a job, Eclipse offers some concurrency related features. Jobs can be placed as background jobs, which will allow the user to continue working while the actions of the job are performed. It is obvious to use jobs e.g. for parsers, converters, or similar time consuming activities.

4.4.10 Help

Finally, Eclipse offers a wide framework for creation of help facilities. Using these, help information of plug-ins can integrate with all other help topics of the Eclipse help catalogue.

Chapter 5

Theory

This chapter describes different theory that can be used in the project. The intension with the chapter is to outline the applied theory – both to present theories to the reader, as well as to define the terminology used in the report.

The chapter uses many abbreviations, that can be found in Appendix A.

5.1 Defining the Syntax of a Language

The syntax of a language like OML can be defined using a context-free grammar[11]. This grammar can be written in different Backus-Naur Form (BNF) dialects. The purpose of a BNF specification is to specify the valid syntax of a language. A BNF specification consists of:

- A finite set of terminal symbols representing the keywords, identifiers, numbers, etc. of the language.
- A finite set of non-terminal symbols each of which representing a phrase in the language.
- A start symbol being one of the non-terminal symbols.
- A finite set of production rules defining how phrases in the language can be composed. This is done by having a choice operator between the different nonterminal and terminal symbols. Each non-terminal symbol will at some stage be represented by a series of terminal symbol.

Though the syntax of a language can be defined in BNF, the productions can be written in a shorter and easy to read format using an Extended Backus-Naur Form (EBNF) notation. EBNF can express the same languages as BNF, but it has some additional convenient capabilities. There are e.g. notions representing optional and repeated occurrences of a symbol.

An extract from the full OML/VDM++ language specification[12] is presented in Listing 5.1.

Listing 5.1: Example of a specification[12] in EBNF notation

```
1 type definition = identifier , '=' , type ;
2
3 type = bracketed type
4     | basic type ;
5
6 bracketed type = '(' , type , ')' ;
7
8 basic type = 'bool' | 'int' ;
```

In this dialect of the EBNF, commas represent concatenation of phrases. Names of terminal and nonterminal symbols can therefore contain white spaces. The equal sign defines how each production behaves and the vertical bar represents an alternative. The nonterminal symbol `identifier` is defined elsewhere and represents a string that follows a specific pattern.

A sentence is a phrase starting with the start symbol. A language can therefore be defined as all sentences satisfying the related grammar. Listing 5.2 shows a sentences satisfying the EBNF from Listing 5.1

Listing 5.2: Example of sentence satisfying EBNF specification in Listing 5.1

```
1 var = ((int))
```

5.2 Trees representing Languages

It is important to notice that BNF notations are primarily intended to define the valid sentences of a language. A BNF specification can be ambiguous, such that two different sequences of chosen productions reflect the same sentence. Different trees can in such a case represent the same sentence. The syntax of the language is defined if a BNF specification is ambiguous, but the meaning of a sentence is ambiguous. When building parsers to build trees, one can re-write the grammar to avoid this ambiguity, taking grouping rules and precedence levels of operators and productions into account.

If building an Abstract Syntax Tree (AST), the nodes are structured so that they contain the necessary information when later defining the semantics of the structure. There are different types of nodes for different language structures and the trees thereby represents the meaning of the language. When building AST's, it is especially important to consider ambiguities in the grammar, as the trees has to be build with respect to the precedence conventions.

Precedence information for the different operators specifies which parts should be evaluated first when parsing a language specification.

A binary operator can either have left or right grouping. This influences if the left or right most side should be evaluated first when a particular binary operator is used a number of time.

A precise description of the precedence and grouping conventions used in this project can be found in Appendix F[12].

5.3 Theory on AST

A popular approach¹ for building language tools is to represent the language specification in an AST after parsing it. Each language structure is represented by a AST class. The intension of building AST's is to build trees that reflect the semantic structure of a specification. This representation is built by creating instances of AST classes and relating these to one another in a hierarchical manner. This tree structure follows the structure of the language. When a production rule in the EBNF grammar is defined as a choice between two or more non-terminals, the theory suggests letting the class representing the production rule be made abstract so that no instances of this class can be made. The AST classes representing the different possibilities can then extend this abstract class. The approach can be illustrated by an example. If a `type` can be either a `bracket type` or a `basic type` then the two latter should extend `type`. Using this approach the AST can be build as described in Section 5.5. Furthermore, the AST can contain additional information used for e.g. type checking, pretty printing etc. Using this technique the AST is able to represent any program fulfilling the syntax for the language.

5.4 Theory on Visitor Design Pattern

The Visitor Design Pattern, is a widely used design pattern that allows action code and data structures to be separated. Literature defines the Visitor Design Pattern in different variations, the notion used in this project is based on [4].

The main focus of the Visitor Design Pattern is to make the data structure as independent from the action code as possible. Variants exists if additional arguments on methods are required, but here the simplest visitor approach is illustrated. An example of how visitors works in practice is shown in Figure 5.1.

- A visitor interface is defined. The visitor interface must define a set of visit methods taking the data structure classes as arguments. If implementing in Java, function overloading can be used to distinguish the different methods. This means that a method called `visit` should be implemented for for each data structure, in order to make it possible for it to operate on. Each visit method takes an object of the data structure type as argument.

¹Used in [11]

- Accept methods are added to all non-abstract data structure classes. An accept method takes a visitor implementing a visitor interface as an argument, and simply calls the visit method in the visitor with the class itself as argument.
- To implement the action code that operates on the data structures, a concrete visitor implementing the visitor interface must be created. In each of the visit methods, action code can be written to specify what should be performed on the specific data structure in question. A popular approach is to let these visit methods also decide which additional data structures to visit. If a child of the current data structure should be visited, the child's accept method should be called with the active visitor as argument.

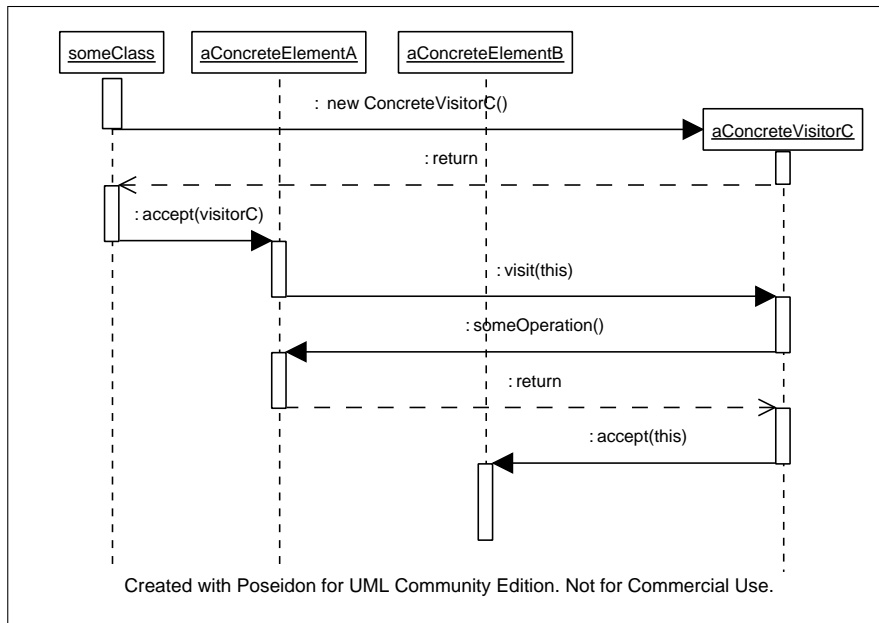


Figure 5.1: Sequence diagram: Example of visitor interaction

In Figure 5.1 a visitor scenario is shown. Briefly it shows a situation, where there initially are three classes – two data structure classes and a class that has the active executing thread. This class creates an instance of `aConcreteVisitor`. Then the `accept` method of `aConcreteElementA` is called with the visitor as argument. The `accept` method immediately calls the `visit` method of `aConcreteVisitor` with itself as argument. The visitor has now access to this data structure and can perform operations of its own choice. It chooses to call `someOperation` on the element. Afterwards the visitor visits `aConcreteElementB`, so it calls the `accept` method

of `aConcreteElementB` with itself as argument. In other words – the visitor sends a reference to itself when it calls other visit methods.

The visitor technique can be used by any tool operating on the data structure. The main advantage is that new functionality can easily be added without changing the data structure implementation, as the action code is completely separated from the data structure. If writing a new visitor, one will only have to make it extend a specified visitor interface. It is in the visitor it is decided which nodes to visit – it is therefore possible for the visitors to visit the nodes it finds relevant.

5.5 Parser Theory

There are two main principles of parser construction, namely bottom up- or top down parsers. These two principles are very different and is a study of itself. In the following, we will give a brief introduction to each principle, and then continue to investigate top down parsers. To get a deeper understanding of parsing mechanisms, many books are available. We recommend [11], as the terminology used in this book is the same as in this report.

Most parsers use a lexers to divide the input stream into a stream of tokens. A parser then reads the stream of tokens, to parse it. Some parsers will read through the tokens once and during recognition produce trees (or whatever output the parser produces), while other parsers will use multiple parses. The most central role of a parser is naturally to parse the text and possibly to create an appropriate tree as result. Parsers are though often more complicated, since they often implement some error handling mechanisms. If the text the parser parses does not follow the syntax, the parser generates error messages. To be able to find several errors in a specification, the parser will also need to have some sort of error recovery mechanism, such that it can continue parsing after finding the first error.

5.5.1 Button Up vs. Top Down Parsing

The syntax of a language that a parser is to recognize, is typically defined in EBNF or similar notations, see Section 5.1. The overall approach for respectively top down and button up parsing are very different, and are indicated here:

- A top down parser starts on the start production rule of the language, tries to apply the rule, tries to follow the sub productions, and continues till the text is either recognized or rejected (no possibility for matching it). The parser will look at the tokens and process these in some manner when choosing which productions to follow. There exists different top down parsing algorithms.

- A bottom up parser starts by looking at the tokens, tries to match these with some of the productions, then tries to find productions using these, and continues till it reaches a start production rule. Different algorithms can be used for identifying which rules to follow.

In general, bottom up parsers are known for being able to recognize almost any language, whereas certain top down parsers can have problems recognizing specific complicated language structures. The discussions leading to choice of parsing technology in our project can be found in Chapter 6. As the analysis ends concluding that the project should use top down parsing principles, the following sections will focus on additional theoretic aspects of top down parsers only.

5.5.2 Top Down Parsing Algorithms

A recursive decent parser is a widely used parser variant. Parsers are often built using parse generation tools, and many parser generation tools create recursive decent based parsers. A recursive decent parser is constructed from the grammar (which must fulfill certain conditions).

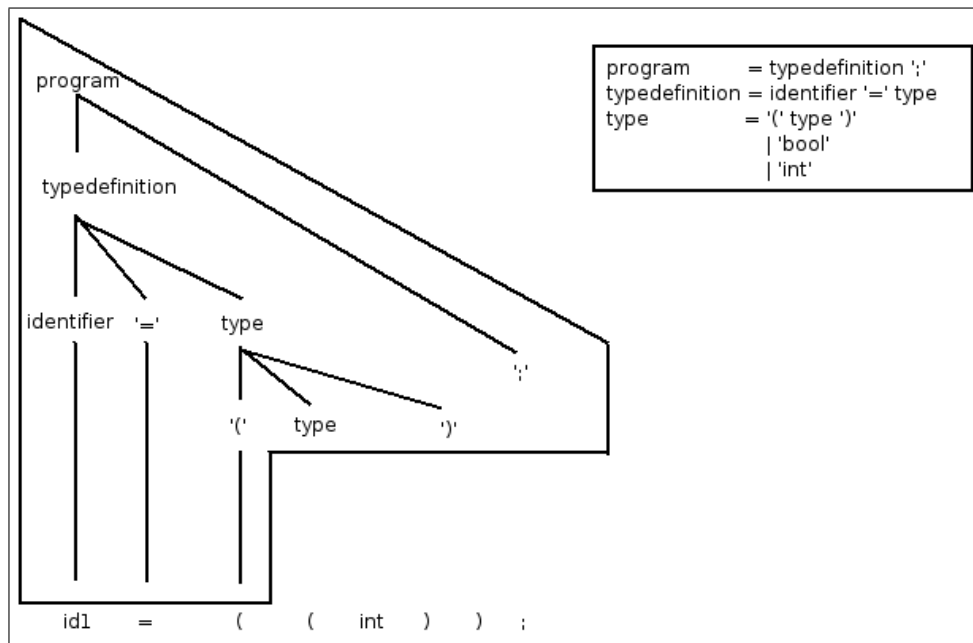


Figure 5.2: Illustration of top down parsing

In Figure 5.2, the top down principle is indicated using a simple language. The illustration is inspired by [2]. At the bottom of the figure, the incoming tokens from the lexer are shown. The parser starts at the start

production, `program`. It follows the productions, and starts by looking for a `typedefinition`. Note, that the parser already knows that the parsing will have to end with a `;`, but this is not tested yet. There is no choice in `typedefinition`, so the parser follows the production, and now expects an `identifier`, an `=`, and then a `type`. The `identifier` is matched to the identifier `id1`, and the `=` is then matched. To apply the `type` rule, the following token, `(`, is used when determining which of the available rules to follow. The parser will therefore be able to match the `(`. The illustration shows this situation, where the parser has not yet chosen which rule to follow next. The parsing has succeeded, if the last token expected by the parser matches the last token from the token stream.

When implementing the parser, each production rule of the grammar will cause the creation of parsing method/function. An example of a method could be the method parsing a value expression. From a parsing method, other parsing methods are called. The parsing methods reads in the tokens from a stream. Parsing methods can consume tokens from the stream as the parsing progresses. The following tokens in the token stream is used to decide which of the possible rules to follow. If a situation occurs where there is no viable alternative to use, there is an error handling mechanism.

How powerful the parser is and whether a recursive decent parser can be created to recognize a specific language, depends on the available algorithms for deciding which productions to follow. Some popular principles for this are described in Section 5.5.3 and 5.5.4.

5.5.3 Fixed Lookahead (Recursive Decent Parsing)

When a recursive decent parser has to decide which of several production rules to apply, it has to base the decision on the future coming tokens. A widely used principle is to make a parser, that has a fixed lookahead, `k`. Such a parser will always base its decisions on the following `k` tokens. This implies that the grammar of the recognized language must be an `LL(k)` grammar, such that a fixed lookahead is always sufficient to avoid non-determinism. Many languages can be recognized by a parser with `k`-lookahead. It should be noted that the `k` is fixed upon creation of the parser – it can not be adjusted at the time the parsing is performed. During the implementation of the parser, it is advisable to keep `k` as small as possible, as larger values will slow down the parsing. Parser generation tools exists that can create good `k`-lookahead recursive decent parsers.

5.5.4 Dynamic Lookahead (Recursive Decent Parsing)

If the language is more complicated, parsers can be created with arbitrary dynamic lookahead. Such a solution is more complicated, but can recognize sophisticated languages. Creation of dynamic lookahead parsers is offered

by ANTLR[18]. In general this tool generate k-lookahead parsers, but it can be instructed to create more advanced parsers. In ANTLR terminology, the technique is to specify syntactic predicates. If a rule is non-deterministic, since the alternatives cannot be chosen by k-lookahead, a syntactic predicate can be added. In the syntactic predicate, production rules can be written. When evaluating a syntactic predicate, ANTLR will try to execute the syntactic predicate without consuming any tokens from the lexer token stream. If the execution of the syntactic predicate succeeds, the following rule will be applied. Effectively ANTLR can generate k-lookahead parsers with automatic arbitrary dynamic lookahead on selected productions.

Chapter 6

Analysis

This chapter describes analysis and research of tools, techniques, and principles useful for development of the kernel. The different parts that should be implemented are analyzed, to choose appropriate tools and techniques.

With respect to analysis of suitable tools, we have tested selected tools on a small subset of the OML language. It is essential for the project that we look at a number of selected tools and techniques and how they can work together before finally deciding which one to use.

6.1 AST Structure

The classes to use for the AST's can be structured in different ways. As requirements on the AST structure influences the way to build these and the choice of parser generation tools, we will here discuss the demands for the AST and the AST classes. General principles of AST's are described in Section 5.3.

When writing plug-ins working on the AST structure, it would be nice to be able to handle e.g. all expressions in one way but functions in another way. Inheritance can be used to enable this, if we use different classes for different types of entities. One could imagine having e.g. a class representing a unary expression, a class representing a binary expression, and letting both of these classes extend a class representing an expression in general. All the classes to use in the AST could extend a simple node class providing basic facilities for all nodes. This inheritance will be more clearly illustrated in Chapter 7.

Another concern is how to protect data in the nodes. One could choose a protective approach providing only read access to the data when accessed from outside the package. This would however make it impossible to add plug-ins that needs to alter the AST structure, e.g. a refactoring plug-in. It has been suggested by the overture core group to structure the AST as described in the blueprint [8]. It suggests to create interfaces for all AST

classes. By having interfaces, one can let the interfaces specify only read access to the data structures. Ordinary plug-ins that only needs to read data, can then access the data using only the facilities provided in the interfaces. Advanced plug-ins that need writing capabilities can do their job by operating on the classes directly.

The requirements for the AST structure is summarized below.

- Different AST classes must be created to represent the different language structures. If the AST classes are built with inheritance and the concept of abstract classes, common properties for e.g. all expressions only needs to be defined once.
- If there is an interface for each AST class, we can protect the tree from being modified by visitors. If a visitor needs to modify an AST, this is still possible, if it references the classes directly.
- The children of an AST class should be strongly typed. By this we mean, that that each language structure is represented by its own type and that a child is an instance of one of these classes. As there are different classes to represent different language structures, it will be easy to specify a type for each child, - possibly the type of one of it's super classes. This prevents errors when building trees and gives additional possibilities for methods operating on AST classes.
- The non-abstract AST classes should have accept method for different visitors. By enabling use of Visitor Design Pattern, new functionality can be implemented without changing the AST structure.
- Visitor interfaces for some different type of visitors. Providing visitor interfaces handling arguments of generic types, can enable a very wide range of visitors to use the AST structure.
- The names used in the AST classes must be meaningful, as other developers should easily be able to use the AST classes in their plug-ins.

6.2 Construction of AST Classes

There will be a large number of AST classes (approximately 300-400) and an equal number of interfaces, and it is therefore significant to thorough test the possibilities for building these. We have therefore looked at four approaches for constructing Java classes for the AST. This section describes the pros and cons for using the different techniques.

6.2.1 Construction of AST Classes using a Parser Grammar File

We have tested the three tools JTB (Java Tree Builder)[20], JJTree[17] and ANTLR (ANother Tool for Language Recognition)[18] for the construction of AST classes. The technique is to build the AST classes from a parser specification file that is based on a BNF specification. JTB and JJTree both uses the parser specification file for the parser generating tool called JavaCC[17], described in section 6.3.2. ANTLR has some pre-defined classes for AST construction.

An advantage of using these tools is that the constructed AST classes can be used directly by the parser generated, as described in section 6.3.2. This means that the AST classes and the parser can be build automatically from one specification file. Therefore, by using these techniques, it is easy to build the many AST classes, the parser and to do future modifications on the language. Moreover, JTB and JJTree has good support for Visitor Design Pattern.

A disadvantage is that the tools cannot be configured to build AST classes with strongly typed children, with correct inheritance and with interfaces as described in Section 6.1. In addition, the generated code is very hard to read and not clearly structured. This makes the generated classes hard to use for developers.

6.2.2 Construction of AST Classes using an XML Schema

XML schemas defines the valid structure of XML instance documents. Tools exists, that can create tree classes from an XML schema.

The technique is to write a XML schema that describe the AST classes and then let JAXB generate these tree classes based on the XML schema. JAXB[19] also creates a parser that automatically can import and export to XML.

A positive point is that the issue of importing and exporting to XML is solved, and that it can take care of error handling in case the XML document is not well-formed.

A negative point is that there are only limited possibilities for customizing the generated AST classes. JAXB can not build AST classes with strongly typed children, with correct inheritance and with interfaces as described in Section 6.1. Furthermore, the generated code is very long and difficult to read.

6.2.3 Construction of AST Classes using an UML Tool

The idea is to draw the AST classes in a UML tool. Many commercial tools are available, but we tested the free version of the tool Poseidon UML[23].

This tool can generate Java files from an UML class diagram and it is possible to write Java code to supply the auto generated code.

The advantage of this method is that it gives a good overview of the structure and content of the AST classes, and it can fulfill the requirements specified in Section 6.1.

The disadvantage of Poseidon UML, is that it does not support Java generics. This is a problem, as we would like to use generics when e.g. a list of some specific type is needed. Thereby, it is not possible for us to create UML diagrams for our AST structure and generate the code for the AST classes using the code generation features of Poseidon UML. We have not been able to find other free UML tools capable of this. Furthermore, Poseidon UML seems to have some scalability problems. Using the tool, we experienced the program to freeze on class diagrams with as few as thirty classes.

6.2.4 Construction of AST Classes by Hand

Constructing the AST classes by hand makes it possible to fulfill all the requirements described in section 3. In addition to this, it is possible to use the generic type concept of Java 1.5. This should help us and future plug-in developers to write better code, as more errors now are found at compile time rather than as run time exceptions.

6.2.5 Summary

We have evaluated the above mentioned solutions and found that constructing the AST classes by hand is the only solution satisfying the requirements described in Section 6.1.

6.3 Construction of the Parser

This section describes the approaches for constructing a parser. From section 6.2, it is clear that the AST classes should be written by hand and the parser tools are therefore investigated with that in mind. We have chosen only to look at the most popular parsers generation tools even though there are many available.

6.3.1 Bottom Up and Top Down Parser

Section 5.5 discussed the possibility of using a bottom up or top down parsing approach. Some parser generation tools, e.g. YACC can be used to generate bottom up parsers while ANTLR and JavaCC can be used to generate top down parsers. After examining the EBNF specification for the OML language[12], we have concluded that a top down parser can be configured

to recognize the whole language. If the language is recognizable by a top down parser, we prefer top down parsing over bottom up parsing.

6.3.2 ANTLR or JavaCC

JavaCC and ANTLR work using the same principle. They take specification files as input and create a top down parser with a constant look ahead. The specification defines the production rules, the parser can follow. As discussed in 5.5.4, ANTLR has the ability to generate parsers with dynamic lookahead on selected productions.

We have examined the possibilities for the two tools and built a parser on a small subset of the OML language.

As the principles of the two tools and their generated parsers are very similar, the evaluation of which tool to use has to be based on the overall impression. The evaluation is documented using structured decision making, see Appendix G. Structured decision making is a principle where structured tables is used to document the decision process.

6.3.3 Summary

We have used structured decision making techniques to evaluate parser generation tool choice. The result is presented in the structured decision making tables in Appendix G.1. We have found ANTLR to be the best tool for the project.

By choosing ANTLR we have a powerful parser generation tool, that is highly customizable so that it creates the parser and builds AST-trees exactly as we intent. An valuable feature of ANTLR is that it besides generation of k-lookahead recursive decent parsers is capable of generating parsers with dynamic lookahead on productions of our choice, 5.5.4. The more advanced lookahead features, the more complicated languages can be supported.

ANTLR fulfil all requirements we have identified for the parser generation tool. These are also presented in Appendix G.1

6.4 XML Facilities

This section analyzes the tools and techniques that can be used to export the AST representation to and from XML.

6.4.1 Exporting an AST to XML

The most straightforward way of exporting an AST representation to XML is to use a tool called JDOM[21]. This tool can read an XML document, validate it according to an XML schema and build it own AST representation of the XML document. It is also possible to build an AST and export

this to a formatted XML document. Using JDOM it is possible to write a visitor traversing our AST representation and continuously building a JDOM tree. This can afterwards be exported to an XML document. It is also possible to write a visitor and manually create the necessary XML tags. This solution would give better performance but also be more difficult to program. However, we find use of JDOM to be the best solution because it makes the visitor easier to program and understand.

6.4.2 Importing an AST from XML

The import can be done by letting JDOM build a JDOM tree from the XML document and converting the JDOM tree to our AST representation. This can be done by traversing the JDOM tree and transferring all the information when creating our AST representation. This can also be done using a program called SAX[16] (Simple API for XML). SAX reads the content of the XML document and continuously provides the content for the user. It gives better performance than JDOM, but it is also more difficult to program because it works at a lower level.

6.4.3 Summary

It is very unlikely that the task of converting one tree structure to another will result in any performance problems. We will therefore not consider this as an significant issue when choosing the method for converting an AST to an from XML. The main issue is that it is simple to program produces a good result. We have therefore chosen to use JDOM for importing and exporting to XML.

6.5 Applicable Eclipse Facilities

In Chapter 4, a number of Eclipse specific facilities was presented. Some of them, like the use of extension points and the creation of an editor, were specified in Chapter 3. This section will focus on which of the other facilities that should be included in the kernel.

It would be beneficial to create a perspective that tailer make the user interface in the Overture editor. Some relevant features would be an outline view to give an overview of an OML specification, a dialog for feedback if an imported XML document is not valid according to an XML schema. Moreover, a problem view in combination with a marker can be used to give feedback to the user in case of syntax errors during parse. Finally, wizards can be used to specify resources.

Natures, builders and jobs can also be used in the kernel, but for the functionality described in Chapter 3 it would not make any significant difference. However, they would be useful in relation to performance issues.

6.6 Summary

As argued for in the sub-conclusions in the preceding sections, we have decided to write the AST classes by hand and use ANTLR for parser generation. JDOM will be used in a visitor to export the AST representation to XML, as well as for importing the AST from XML.

Chapter 7

Design of the Overture Kernel

In this chapter of the report, the considerations and choices made during the design process is described. The chapter uses diagrams to show the overall design principles, and the related text describes special design considerations.

Detailed and technical aspects of how to implement the design using the power of the Eclipse environment and Java concepts are discussed in Chapter 8.

7.1 General Design Considerations

It is a requirement and desire for the project, that the kernel is built as flexible as possible, as we expect the community to develop additional plug-in's that extend the system with additional functionalities. When designing our solution, there are primarily two scenarios to consider. First of all, our solution should be structured to provide clear useful interfaces for future plug-ins. However we also need to consider how our solution in a reasonable manner can support cases where parts of the core needs to be upgraded; e.g. to fix a bug or on modifications of the OML language.

7.2 Dividing the Functionality into Plug-ins and Packages

There is a preference towards designing the kernel in a way which is as flexible as possible. Therefore, it is obvious to divide the functionality of the kernel into separate plug-ins and packages. A plug-in can later easily be replaced by a new version.

We have chosen to divide the Overture kernel into 11 modules. 9 of these modules provides the kernel functionalities, and two modules defines the packaging needed to distribute the kernel. The main modules are listed below.

- AST
- Parser
- XML to AST parser
- AST to OML visitor
- AST to XML visitor
- Outline visitor (outline view for OML developers)
- AST outline visitor (AST outline view used for test purposes)
- Util
- Editor

The purpose for dividing the kernel into modules is so that developers can work on different parts of the kernel at the same time. Furthermore, we demonstrate that the kernel is flexible and extendable. Finally, it gives a better overview of the code.

Figure 7.1 shows how the different modules can depend on each other. An arrow from A to B indicates that A depends on B. In the following, the role of the different modules will be indicated.

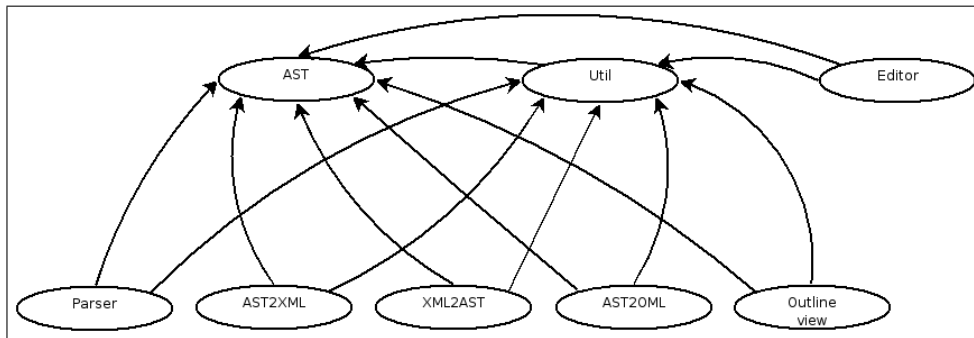


Figure 7.1: Overview of dependencies

The AST module described in Section 7.3 should define all AST nodes, their interfaces, the visitor interfaces, and general visitor implementations for traversing an AST.

The Parser modules described in Section 7.4 and Section 7.9 should construct an AST from an OML specification. The XML2AST module described in Section 7.9 should convert an XML document to an AST.

The Visitor modules described in Section 7.6 should traverse the AST and build other trees or perform tasks on the AST. In Figure 7.1 the plug-ins

ast2xml, ast2oml, and outline are capable of respectively exporting an AST to XML, printing an AST to a plain OML specification, and creating an outline view.

The Editor module described in Section 7.10 should be responsible defining the Overture editor and for loading the different parsers and visitors.

The Util module should contain interfaces and classes that are used by more than one module. Util will in a sense be the module, that combines the other modules.

Furthermore, the design of the XML schema is described in Section 7.5 and how it should be possible to extend the kernel is described in Section 7.11.

7.3 Design of the AST

This section gives an overview of the design principles for AST construction. Figure 7.2 is an example of how the AST should be implemented.

The Figure shows that an AST class can either inherit from one of the two abstract classes `InternalASTNode` or `InternalASTNodeWithComments`, depending on if it can have associated comments.

It has been decided to store position information for each AST node. This information should e.g. be used for pretty printing or for making it easier to find the right line in the Overture editor using an outline view when working with large OML specification.

To handle keywords, it has been decided to create an AST class representing keywords. This class should store the keyword, its positions, and possible following comments, so that this information can later be used e.g. for pretty printing.

The AST classes should have set and get methods for each of their children and accept methods for visitors and they should implement corresponding interfaces showing only the get methods to the user. By having these interfaces, the user can operate on the interfaces without being aware of the implemented AST classes.

It can also be the case that classes inherit from other abstract classes which some how inherit from `InternalASTNode` or `InternalASTNodeWithComments`. This is the case whenever there is an option in the corresponding BNF specification. This is shown in Figure 7.3.

7.4 Design of the Parser

ANTLR has been chosen as the parser generation tool. Therefore the overall design of the parser is limited to be able to use the generated parser.

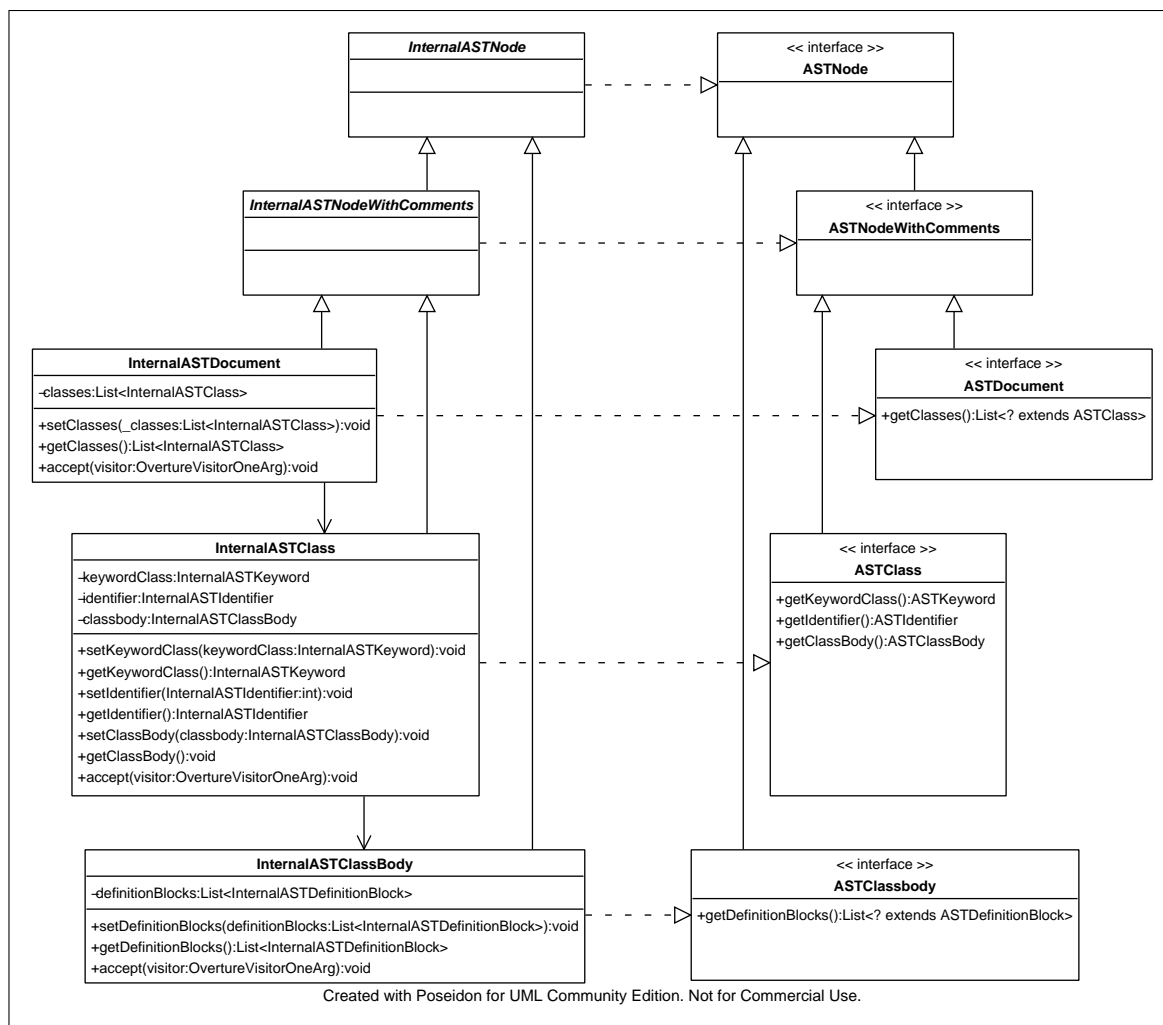


Figure 7.2: Example of selected AST classes based on inheritance and interfaces

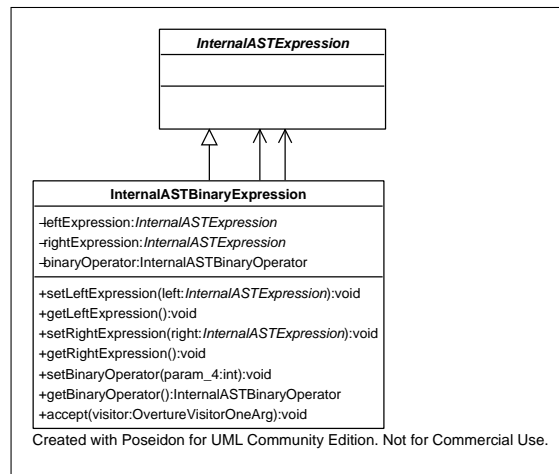


Figure 7.3: Example of an AST with inheritance

7.4.1 Design of Comments Handling

A built in mechanism of ANTLR is available for handling comments. If building a traditional parser, that ignores comments, ANTLR will generate a lexer and a parser. Usually the lexer will ignore all comment-tokens. As the parser of this project must preserve and store comments, we have to let the lexer pass comment-tokens on to the parser.

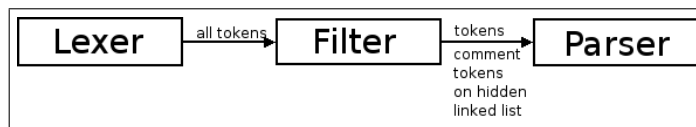


Figure 7.4: Filtering comments - Filter between lexer and parser

The solution is to add an intermediate filtering layer between the lexer and the parser. This filter can hide the comment-tokens, so that the parser will not see them. They are however stored in linked lists. The filter is placed as indicated in Figure 7.4.

How the comments are stored are shown in Figure 7.5. Each square represent a token. The parser cannot see the comment-tokens, so it will read the tokens a, b, c, d, and e. In the original text, there were however two comments between tokens c and d. There are methods on all tokens to get preceding or following comment-tokens. This means that during e.g. tree construction, it will be possible to retrieve the comments that e.g. follows c, if the a method is called on c.

A special case is if comments occur in the very beginning of an OML specification. As indicated in Figure 7.5, comments are both accessible from

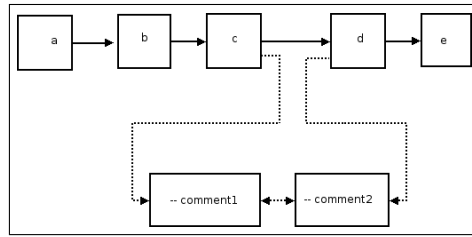


Figure 7.5: Filtering comments - Comments stored in hidden linked list

the preceding and from the following token. Therefore the trailing comments of an OML file can be retrieved by calling a method on the token of the first word `class` in the file. When building the AST, the trailing comments in a file is stored in the root node, `InternalASTDocument`.

7.4.2 Precedence Principles

With respect to precedence, there is a need to re-write the production rules to make the parser respect precedence as intended. ANTLR has no obvious way of specifying precedence easier. The technique is well known in parser construction and works by dividing a production rule (e.g. the rule looking for all expressions) into new rules. Each rule is to recognize the expressions of a specific precedence level. From a rule, there will only be references to rules of similar or higher precedence, as expressions of lower precedence cannot be the child of a given expression. When the parser is to look for an expression, it must start by trying to match expression of lowest precedence and then try with higher and higher precedence until it finds a matching rule. The principle used to handle precedence is further discussed in Section 8.3.

7.5 Design of the XML Schema

The XML schema is used to verify the correctness of an XML document. In the overture kernel, a validation is performed every time an XML document is imported. Figure 7.6 is an extract of how the XML schema is structured.

The schema is created using a tool called XML Spy¹[24] and Figure 7.6 shows an image representation of the XML schema. An extract of the XML schema can be found in Appendix I.6. The complete textual version of the XML schema can be found on the cd-rom in the plug-in called `org.overturetool.eclipse.editors`. In Figure 7.6 each box represents an XML element. The box named `OMLType` contains four elements, namely:

¹Here the structure of the XML schema can be created using a graphical editor environment. This gives a nice overview instead of the textual version that is very difficult to read and edit.

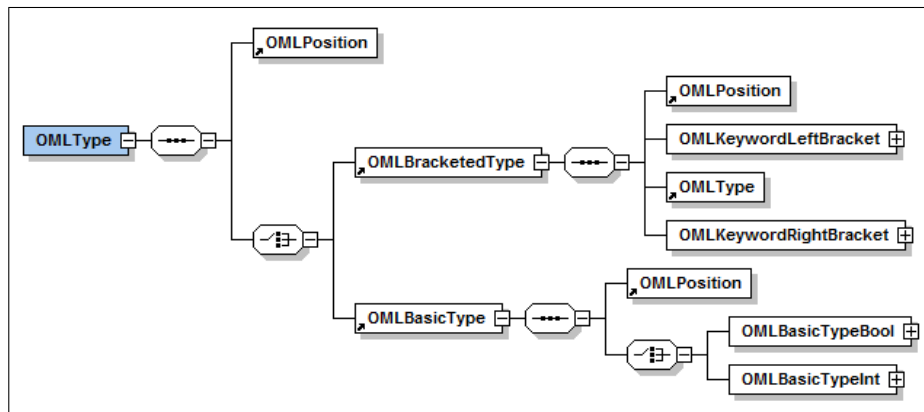


Figure 7.6: XML Schema example (XML-Spy diagram notation)

`OMLPosition`, `OMLIdentifier`, `OMLKeywordEqualsign` and a choice between `OMLBracketedType` and `OMLBasicType`. An `OMLComments` element is optional and therefore has a dotted frame.

The schema is structured so that each production rule in the OML language specification[12] is specified as was it a root element. Notice that the structure of the XML example closely follows the structure of the BNF given in Listing 5.1. The other elements can then reference it. The element `OMLBracketedType` has e.g. a reference to `OMLType`. This structure makes it able to handle if the elements in the XML document is nested in any way. Comments are stored in an attribute in an element called `OMLComment`. The element called `OMLComment` can contain several elements of type `OMLComment`. Figure 7.7 shows how `OMLComments` elements is related to other elements.

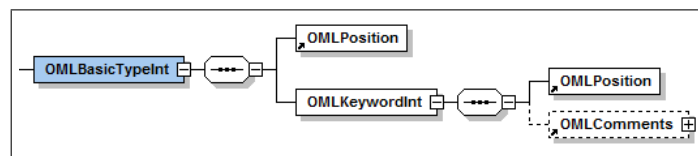


Figure 7.7: XML Schema example (How comments are stored)

An element can have corresponding attributes. For keywords, identifiers, comments and symbolic literals it is necessary store the corresponding string value. This information is used when pretty printing an AST. For keywords the corresponding string value is fixed, meaning that a particular keyword element has to contain the given keyword text. For identifiers, comments and symbolic literals the string value is required. Furthermore, the schema is designed so that no information is placed between the element tags. All

these features are made in order to make the XML documents as ambiguous as possible.

In order to make it easier to operate on the XML instance document, a convention has been made stating that a position placed above another position element should always have an equal or smaller start line value. If the start line values are similar the start column value should be smaller.

7.6 Designing the General Visitors

The different visitors should work using the same principle and the design of some general visitors will therefore be explained here.

There should be two general visitor interfaces for traversing an AST. One of these called `OvertureVisitorOneArg` should not provide a possibility for passing any information as arguments when examining the different nodes and the other one called `OvertureVisitorTwoArg` should provide this. These two general interfaces will be sufficient when writing the majority of the visitor implementations including all the visitors for the kernel. The purpose of having these interfaces is that they are used as arguments for the accept methods in the AST classes and by implementing these the visitor implementations are able to traverse the AST. A general explanation of visitors can be found in Section 5.4.

Furthermore, there should be two general visitor implementations called `OvertureVisitorOneArgImpl` and `OvertureVisitorTwoArgImpl` implementing the two general visitor interfaces. By extending these the specific visitors should only have visit method for the nodes where there should be some action. The general visitor will here provide the functionality for traversing the tree.

7.7 Designing the Pretty Print Visitor

The pretty print visitor called `OvertureAst2OmlVisitor` should write the AST to a text representation. This means that the pretty print visitor can be used to regenerate an OML specification from an AST. The formatting can be done by using the position information in each node.

The algorithm for producing a formatted string is first to insert new lines and spaces until the correct place is found and afterwards insert the string. This should be done for all keywords, identifiers, comments and symbolic literals in the AST.

It is here significant that the nodes containing the keywords and other strings which have been parsed first, is also visited first when traversing an AST in a deep-first order. If an external plug-in does not preserve the order of e.g. comments, it is possible to write a refactoring visitor that sorts the comments according to their positions information.

The AST to OML visitor should extend the general visitor class called `OvertureVisitorOneArgImpl`. This means it should only have visit methods for the nodes where a keyword, or another string should be pretty printed. This will reduce the number of visit methods significantly.

7.8 Designing the AST to XML and Outline Visitors

The AST to XML visitor called `OvertureAst2XmlVisitor` should produce a formatted XML instance document that stores all information in the AST so a similar AST can be created using the information in the XML document. The algorithm for doing this is to create an element for each node in the AST and nest these in a way that reflect the AST. The additional information like position and comments can then be stored as attributes. It should traverse all the nodes in the AST and therefore implement the `OvertureVisitorTwoArg` interface.

The Outline visitor should similarly traverse the AST structure and collect a selected part of the information for the outline view. By extending the `OvertureVisitorTwoArgImpl`, it will only be relevant to specify visit methods for the nodes that the visitor is to operate on.

7.9 Design of the XML to AST Converter

The XML to AST converter called `OvertureXml2AstConverter` should operate on a XML instance document and produce an AST. As the XML-schema closely reflects the AST structure, the conversion can be done by traversing the XML instance document once and continuously build a corresponding AST. This also means that it is not necessary to handle grouping or precedence. The XML instance document should be well-formed according to the schema described in Section 7.5 and it is therefore not necessary to do any error handling.

7.10 Design of the Editor

This section will give an overview of how the most central parts of the Overture editor is structured and how the editor integrates with the Eclipse framework. The UML class diagrams are intended to give an overview of how the editor can interact with the Eclipse framework. In the class diagrams, some classes and interfaces are marked `«Eclipse»`. This indicates that the class is part of the Eclipse framework. It is up to us, the developers, to create and implement the remaining classes. In the first diagram we will present

how the wizards should be integrated with Eclipse. Then an overview of the editors and menu items integration will be given.

7.10.1 Editor – Wizards

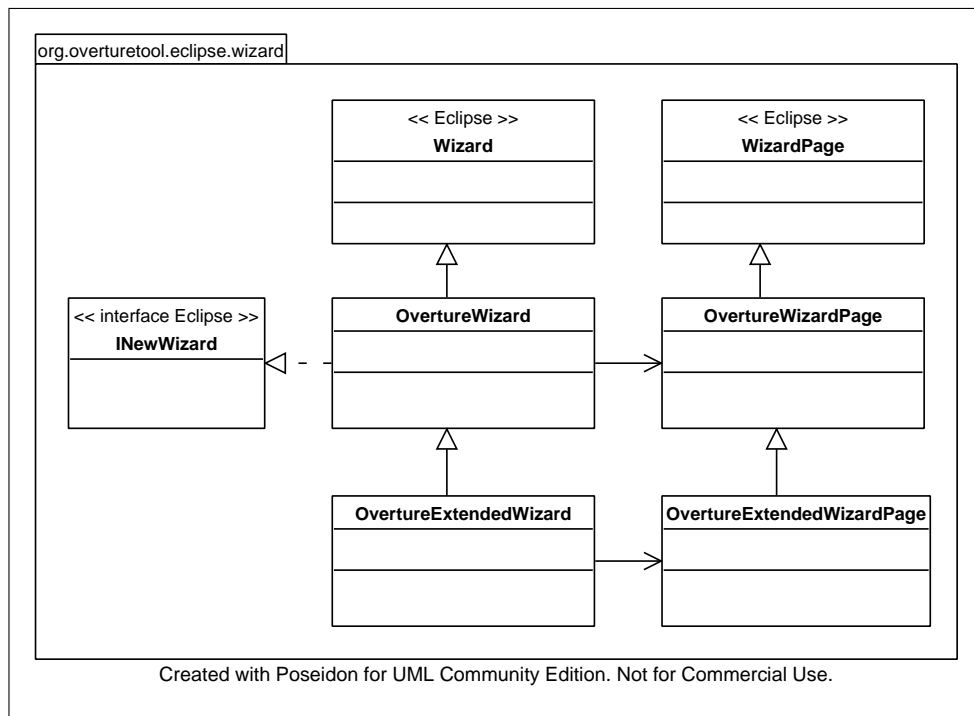


Figure 7.8: Eclipse wizard overview - Class diagram

Eclipse provides as indicated some classes and interfaces for wizards. Classes to represent an overture wizard and a class for representing an overture wizard page should be implemented, so that they extend the related Eclipse framework provided classes.

In Figure 7.8, it is shown how overture wizards can extend the Eclipse based wizards. The intension of having both the class `OvertureWizard` and the class `ExtendedOvertureWizard` is that `OvertureWizard` can be a simple wizard asking for one file location and file name, whereas the additional `ExtendedOvertureWizard` can be a wizard asking for multiple file locations and file names. Using a design like this for e.g. wizards enables both to use the Eclipse facilities as much as possible, but also to reuse own components for more advanced wizards.

Editor interaction with Eclipse

Eclipse offers many facilities for editors. It is up to the developer how many of these to explore and use, but the following design in Figure 7.9 can be used to create a powerful editor.

In the center of the diagram there is a class that extends `TextEditor`. `TextEditor` is a basic editor provided by the Eclipse framework. The editor has a reference to a configuration class, which has then a reference to a scanner. In Eclipse based editor terminology, a scanner is the module responsible for finding keywords, strings, and similar structures, that the editor should highlight graphically with some color.

The editor also has a reference to the class called `OvertureOutlinePage` being responsible for displaying a tree structure in the outline view. The class `Overture Content Provider` is used to provide this tree structure. A method in `OvertureOutlinePage` will automatically be invoked by Eclipse when a user opens an OML file in the editor. It is therefore possible to open the customized Overture perspective from here.

Please note that almost all classes extend some eclipse provided class. This shows that the programmer will only have to specify actions and special behavior, while the most basic functionality is offered by Eclipse.

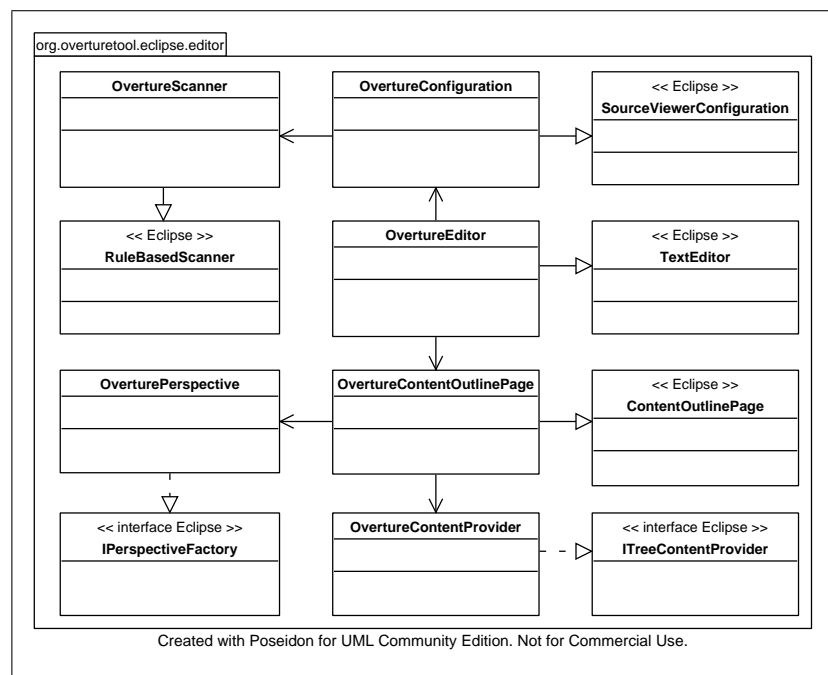


Figure 7.9: Eclipse editor overview - The most important use of classes

Figure 7.10 shows how a menu item to an editors menu can start execu-

tion of the relevant actions. In the package in the middle of the figure, the actions representing each menu item is shown. There are actions for AST to XML conversions and reverse. The actions implement an action delegate interface offered by eclipse. The action will then be activated when the related menu item is clicked in the menu, and the action can perform its execution using the classes it needs.

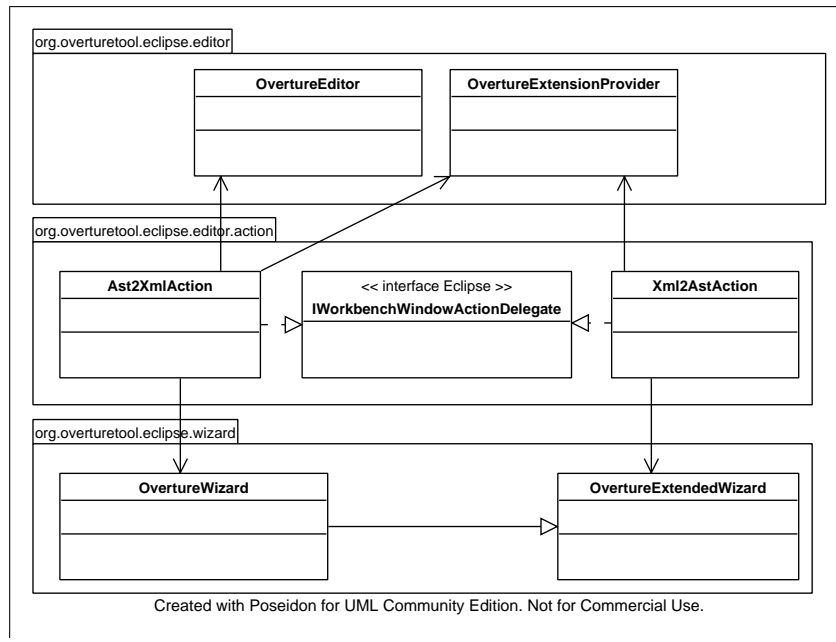


Figure 7.10: Eclipse action overview - Class diagram

7.11 Design of Extension Possibilities

A central point is how to design the extension possibilities. We have created a design, shown in figure:editor3, which makes use of the extension concept of eclipse.

As shown in Figure 7.11 the `OvertureExtensionProvider` class is responsible for loading the parser, converter and visitors. This functionality works so that the Overture editor is unaware of any parser or visitor implementation. It relies fully on the fact that the `OvertureExtensionProvider` can provide classes that implements the four interfaces shown on the figure.

Using this design, other developers can replace our implementation of the parsers and visitors without modifying on any of the plug-ins. This approach also makes it possible to prepare the Overture kernel for e.g. type checking without it being implemented. By implementing the type checker

interface anyone can extend the kernel with this facility. However, when adding completely new functionality some alterations has to be done.

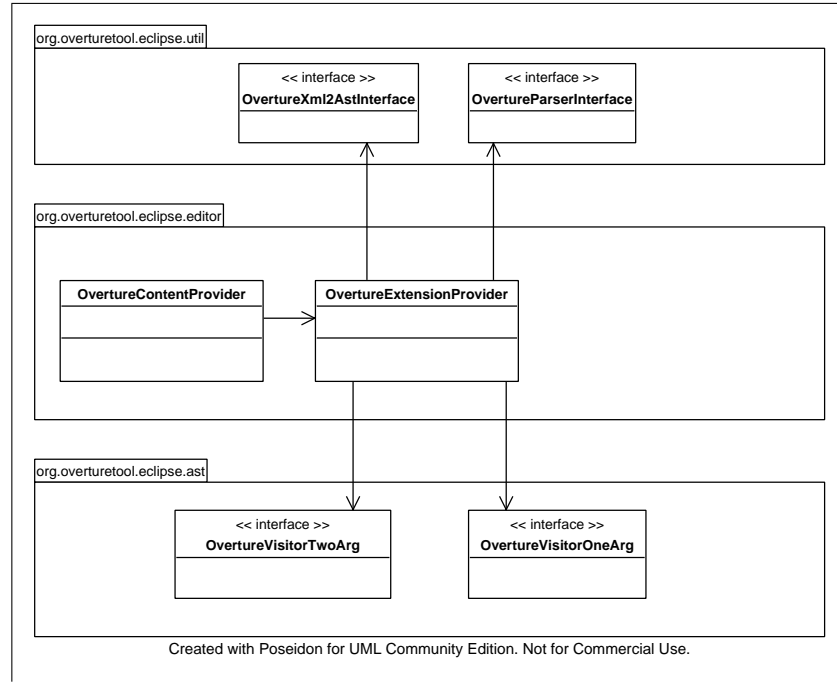


Figure 7.11: Eclipse editor overview - Loading classes for extensions

7.12 Summary

The design has been created with main focus on extendability and flexibility. Where possible, the system is designed to use Eclipse facilities. The intension is to make the kernel integrate best possible with Eclipse.

The illustrated design has been used for the implementation, which is described in Chapter 8.

Chapter 8

Implementation

This chapter describes implementation related aspects of the project. The general design decisions has been described in Chapter 6 and in Chapter 7. The entire kernel has first been implemented for a small subset of the OML language. This was done in order to ensure that the full functionality described in Chapter 3 could be supported. Afterwards the kernel has been implemented for the entire OML language. Here the main focus has been on building the parser with respect to the correct precedence and grouping conventions. The design of the implementation is the same for both iterations – the design presented in Chapter 7 is closely followed.

8.1 General Implementation Issues

The implementation of the kernel is written as plug-ins for Eclipse using the plug-in development environment in Eclipse. The plug-in structure is described in Section 7.2. As discussed in Section 7.11 the solution will be very flexible, if the different parts of the kernel are implemented as individual plug-ins. The following sections will describe the implementation issues of the different plug-ins. Developing the different kernel parts as plug-ins has some advantages for the implementation in general. The advantages are described in Section 7.2.

In Eclipse there is the concept of projects. Projects can have dependencies, if a project depend on another project. Technically each plug-in created will have it's own project in Eclipse. Thereby one kernel plug-in can be modified without changing or rebuilding the other plug-ins.

8.2 Implementation of the AST Classes

In the following sections, the implementation of the AST structures is implemented. A set of AST classes and their interfaces can be found in Appendix I.2. These have been selected, so that they represent the applied

techniques and principles used in the implementation of the AST structures.

8.2.1 Structure in General

The classes representing the AST is located in a plug-in¹. This plug-in contains two packages, **A**² and **B**³. **A** contains interfaces for all AST classes, whereas **B** contains an implementation that implements the interfaces. The intension by the two packages is as follows; whenever a plug-in needs to read from the AST without changing it, it should use the interface classes from **A**, as this provides only read access to the tree structure. If a plug-in needs to change, add, or delete content of a tree, the classes from **B** can be used.

The trees are built using different classes for the different structures that can occur in an OML specification. In other words there is a class representing an OML class, another class representing an access type definition, etc. The classes are organized in a hierarchical structure, built using inheritance. The power of using inheritance in this manner is that plug-ins that should operate on the tree, can easily visit the different nodes. In addition to that, visitors can choose to treat similar structures in the same way. A visitor could e.g. choose to handle all kinds of expressions in the same way without having to provide visit methods for each particular expression.

8.2.2 Setting the Positions

As discussed in the design of the AST classes Section 7.3, position information is significant when pretty printing an AST. When creating instances of the different AST nodes, the position of the object is set. Position is in the solution defined as start line number, start column number, and end line number of the code in the parsed specification. All nodes have a position. We have defined the position of a node to start at the first token in the expression and ending at the last token of the expression. Listing 8.1 shows an example of a simple OML access type definition.

Listing 8.1: Example of OML code

```
1  ...
2  public var1    — Comment1
3      = int;    — Comment2
4      — Comment3
5  ...
```

The parser will identify that this is an access type definition. There will be created an instance of `InternalASTAccessTypeDefinition`, to represent the expression. The start position set for this node will be the position of the

¹org.overturetool.eclipse.ast

²org.overturetool.eclipse.ast

³org.overturetool.eclipse.internal.ast

'p' in 'public', whereas the end line position will be set from the last token 'int'.

How are these positions actually found? When the parser parses e.g. the word `public`, it will identify it as the keyword `public`, and create an instance of `InternalASTKeyword`. This node representing the keyword should have the same positions as the token representing the keyword. Therefore the constructor of `InternalASTKeyword` is called with the token as an argument. This is the case when setting positions for all nodes created from one token, e.g. the keywords, identifiers, and most basic types. The constructor code that takes a token and stores its position information in the node is only defined once, namely in `InternalASTNode`. This constructor code can be called when creating all kinds of nodes, as all nodes inherit this constructor from it. The source code of `InternalASTNode` can be found in Appendix I.2.

When a node represents a language construct, e.g. an access type definition, the node will have children representing the individual keywords and identifiers it consists of. The access type definition node can therefore easily set its position information from its first and last children. Below are extracts from the source code showing how this is done for an access type definition.

Listing 8.2: Example of code that sets position information

```

1 ...
2 internalASTAccessTypeDefinition . setStartPositionFromNode (
   internalASTAccess );
3 ...
4 internalASTAccessTypeDefinition . setEndPositionFromNode (
   internalASTTypeDefinition );
5 ...

```

In the code above, methods `setStartPositionFromNode` and `setEndPositionFromNode` are called with a node as argument. These methods are only defined in `InternalASTNode`, as all other node classes inherit from that class.

8.2.3 Handling Comments

End line comments may occur anywhere in an OML specification. It is a request that the comments are stored in the AST. Comments will always be attached to the simplest node type preceding it. This is illustrated by an example – the following explanation explains comments handling in the example shown in Listing 8.1.

The comment 'Comment1' will be attached to the identifier node representing 'var1', whereas the comments 'Comment2' and 'Comment3' will be attached to the entire access type definition example. The reason they seem to be treated differently is, that 'Comment1' is preceded by an identifier. Therefore the comment is attached to this node. The comments appearing

after the semicolon will need to be attached to a keyword representing the semicolon. This keyword is attached to the access type definition node. The handling of semicolons is treated in Section 8.2.4.

It is only certain kinds of nodes that should have the ability to store comments. All node classes which instances may have to store comments extends `InternalASTNodeWithComments` directly or indirectly. This class inherits all methods and properties of `InternalASTNode`, but adds the concept of handling comments.

When the parser needs to store a comment for a token, the method `addCommentsFromToken` is invoked, either directly from the parser or for simple structures by the parser. An ANTLR token is passed as argument. The task for the method is to investigate whether there are hidden tokens/comments after this token. If comments exists, they should be added to a comments object, that should be stored in the node to which the comments belong.

8.2.4 Handling Semicolons

There are several places in the grammar for OML, where there are optional semi-colons. Semicolons are e.g. optional after the last access type definition of a series of type definitions, after the last function definition of a series of function definitions, etc. For future plug-ins generating code, these semicolons are needless, if the AST reflects the structure of the specification. For plug-ins that needs to display the code, the positions of semi-colons may though be of importance.

In order for the parser to be able to identify these optional semicolons correctly, the ANTLR grammar specification that is used to generate the parser specifies that the parser should be created with a lookahead of $k=2$.

When the parser finds a semicolon in a specification, it creates an instance of the class `InternalASTKeyword`. If comments occur right after the semicolon, they are collected and stored in this node.

The semicolon node is attached to the structure it closes. This means that a reference to the (optional) semicolon node after an access type definition is stored in the access type definition node. Future plug-ins may decide if they need to visit the semicolon nodes.

8.2.5 Stability using Java Generics

From Java 1.5.0, Java generics has been available[1]. Traditionally there has been no type on the content of a list or collection – the content had type `Object`. With Java generics it is now possible to create lists of a specific type. It is also possible to create a list of any type that extends a given type. When operating on e.g. typed lists, it is at compile-time checked that type requirements are fulfilled. This is valuable as many trivial programming errors can now be found by the compiler.

In the kernel implementation, we have used the new possibilities. Every time a node can have several children of the same kind, we use Java generic lists. For example, a document can contain a number of classes. Therefore a document will have a list containing classes. A plug-in developer will get a compile time error or warning, if he treats the children of a document as anything but classes. An example of such a class that uses typed lists is given in Listing 8.3

Listing 8.3: The interface `ASTDocument.java` has a method with a generic typed list as return type

```

1 package org.overturetool.eclipse.ast;
2
3 import java.util.List;
4
5 public interface ASTDocument extends ASTNodeWithComments {
6     List<? extends ASTClass> getClasses ();
7 }

```

Generic types can also be used when defining methods. Generic types has therefore also been used when specifying the visitor interface and visitor implementations that takes arguments. The main benefit of this is to allow the creation of visitors taking any type of argument, but still check that the types of the arguments are used consistently in a specific visitor.

8.2.6 AST Classes Special Case – Multiple Inheritance

The design works in general well and gives a clean implementation. There has however been cases that needed special care. An example of this is `name` which can be both an `expression`, `state designator`, and an `object designator`. A solution would be to let the AST-class representing `name` extend all three abstract classes. Multiple inheritance is however not allowed in Java.

The chosen solution has therefore been to create a few additional classes with the wanted properties. For the given example, we chose to let the class `InternalASTName` extend `InternalASTExpression` and then create the additional classes `InternalASTObjectDesignatorName` to represent names being design designators, and `InternalASTStateDesignatorName` to represent names being state designators. These two classes are simple classes that extends the appropriate abstract class and has a reference to `InternalASTName` in them.

8.3 Implementation of the Parser

We use ANTLR to create the parser. The parser is specified in an ANTLR grammar file, which can be found in its entirety in Appendix I.1.

8.3.1 Specifying the Grammar

ANTLR operates by reading a ANTLR specification (from a .g file) and generating parser, lexer, tokens, etc. The specification file is written in ANTLR's own specification language.

The specification is closely related to the EBNF specification of the OML language the parser should process. The names of the BNF-productions are reused for the ANTLR productions. As an example, the BNF production, `class`, is represented by the ANTLR production, `overtureClass`. For each production, the corresponding EBNF production from the CSK VDM++ specification is written as a comment. This is to make it easy to verify that the grammar specification reflects the CSK language specification.

8.3.2 Error Handling

It is important how the parser handles errors. The parser is responsible for identifying errors in a specification, and for providing a reasonable error message. If possible, the parser should try to recover from the error, in order to proceed to see if there are more errors in the specification. Upon errors, the parser provides a list of identified errors. It is however not the responsibility of the parser to present those errors to the user; this will be done by the editor.

If ANTLR is asked to create a parser from a well specified, unambiguous grammar specification, it is capable of providing quite good error handling. This includes mechanisms to recover from errors to find additional errors in a specification. Through tests, we have decided that the built in error support and error recovery support of ANTLR is sufficient for the project. Therefore we have focused on writing stable grammar specifications for ANTLR and for handling the provided error messages from ANTLR well in the editor.

8.3.3 Building an AST

ANTLR offers a built in way to create parsers that build AST's. As found in Section 6.2.5, this method is however not applicable for our project, as we want the AST classes structured in a hierarchical manner. Therefore we have disabled ANTLR's automatic AST construction. Instead we have added action code creating instances of our AST classes when ANTLR matches a production in the parser. In this way the AST will be built while the parser parses a specification. It is made in a way that still enables ANTLR's error recovery mechanism to operate properly.

8.3.4 Handling Unicode Characters

It is a wish that the kernel should be able to handle unicode characters in specifications. Following [12], unicode characters are not allowed in specifi-

cations. In the current kernel, special characters are not allowed, but it is important to notice that the parser generation tool, ANTLR, has unicode character support. Therefore it is easy to add unicode support once there is agreement on exactly where in specifications to allow which special characters. Java should support any unicode characters, and should therefore not rise problems regarding this. To implement unicode character support e.g. in identifier names, a single line in the ANTLR grammar specification must be adjusted to specify the relevant characters or character range.

8.3.5 Handling Comments

Handling comments is of importance for the project. Often when writing parsers, comments are recognized by the lexer, but discarded and never presented to the parser. This makes sense when writing a compiler, as comments have no semantic influence on the generated code. We are however writing a general kernel, that should support all kinds of tools operating on OML specifications. We should not discard comments, as plug-ins might want to be able to find them in the AST structure, e.g. when writing a pretty printer.

We have to analyze how it is possible to handle comments, given that we use ANTLR for parser construction. Naturally we want the lexer to recognize comments as tokens, – we will need them when building the AST. On the other hand, it is not a plausible solution to treat the comments as ordinary tokens in the parser, as this would force us to specify an optional comment-token everywhere in all productions where comments could occur. However, it is possible to create a solution satisfying all needs.

As described in Section 7.4.1, ANTLR has the concept of token filters. A token filter is a filter applied on the token stream generated by the lexer before the tokens are read by the parser. This means that the parser actually reads it's tokens from the filter, which reads it's tokens from the lexer. A token filter is capable of splitting the token stream into two separate token streams. We split on comment tokens, meaning that all comment tokens goes into one stream and all other tokens go into a different stream. The parser is then asked to read from the stream without comment tokens. Thereby the parser sees the tokens as if there were no comments at all in the specification that is parsed.

In the action code building the AST, one can for each token ask for preceding or trailing comment-tokens. Thereby it is possible to get the hidden comment tokens, that followed or preceded the current token.

As OML only allows end line comments, we have chosen to store comments in the AST node representing the preceding ordinary token. By having this simple but consistent convention on how to store comments, there is no doubt of how to interpret comments in the AST. There is though one exception, – what to do about comments appearing in the very beginning of a OML specification? We have chosen to add these comments to the class

'ASTDocument' representing an entire specification. Comments at the beginning of a file will typically be copyright comments or comments about the specification in general.

8.3.6 Handling Precedence

The OML language has different precedence priorities for different expressions. The parser has to take the implicit given precedence rules into account when building AST tree instances. The implementation is based on Section 7.4.2.

Handling precedence correctly in a recursive descent parser demands some considerations. Wherever the BNF specification is ambiguous, the productions should be chosen, such that nodes with high precedence are placed as close to the leaves as possible. A recursive decent parser will however start at the root of the AST tree it is building, so it must try to match the expressions with lowest priority before matching expressions of higher priority.

For handling precedence of expressions correctly, we have assigned a precedence value to all kinds of expressions. In [12], there are tables specifying precedence of e.g. all evaluators or all connectives. However, it is also stated that all evaluators must have higher precedence than all connectives. We have therefore assigned new precedence values that respects the above mentioned requirements and gives an unambiguous precedence level. The precedence levels are shown in Appendix F.

To handle precedence as intended, we specify a rule for each precedence level. From a rule representing a precedence level, only rules of higher precedence levels are referenced. When the parser is to look for an expression, it starts by trying to recognize an expression of lowest precedence. If it fails to recognize this, it will try to recognize an expression of higher precedence.

With the chosen design and development approach, the parser is entirely defined in an ANTLR grammar file, from which ANTLR can generate the parser. When ANTLR works on a grammar file, it will produce nondeterminism warnings for all situations where the parser may parse the given text in two different ways. All of these situations have been solved by adjusting lookahead, setting a 'greedy' option, or by using syntactic predicates. These terms are explained in Appendix A. Greedy is an ANTLR command telling ANTLR to make as many matches as possible, ignoring non-determinism warnings.

Listing 8.4: Example of ANTLR specification: a production for `type` at precedence level 2

```

1  overtureTypePre2 returns [InternalASTType internalASTType]
2  {
3      //Type operators with precedence 2 (low, but not lowest precedence)
4      internalASTType = null;
5  }
6      :
7      (overtureUnionType)=> internalASTType = overtureUnionType
8      | internalASTType = overtureTypePre3
9
10 ;

```

Listing 8.4 shows a few lines from the ANTLR specification. They specify how the parser should recognize `type` with precedence two or higher. The specification should be read as follows – the production returns an `InternalASTType`, which is an abstract class. If possible, it matches the following tokens with an union type, otherwise it will look for a type of precedence three or higher. The production for union type will use both the productions for precedence levels two and three, but will never use a lower precedence (precedence one). Had it done this, the productions would not respect the precedence conventions.

The "`(...)=>`" in line 7 of Listing 8.4 is the use of a syntactic predicate. Syntactic predicates are defined in Appendix A, whereas the theoretical aspects of dynamic arbitrary lookahead is presented in Section 5.5.4.

8.3.7 Grouping of Binary Expressions

For binary expressions, grouping is of importance. Grouping refers to how the trees are constructed. In mathematical terms, the tree must be built with left grouping when an operator is left associative and with right grouping when an operator is right associative. The rationale for this is, that we want the structure of the tree to reflect the grouping rules.

For some operators the associative law holds. In these cases a left grouped representation is equivalent to a right grouped representation and therefore the parser builds either left or right grouped trees depending on what was most convenient to implement for the given operator. Groupings are shown in Appendix F.

In the parser specification file, rules are specified differently depending on whether the tree is to be built left or right. The parser reads the tokens of a given specification left to right. It is however still possible to recognize and build trees for left associative expressions, though rules for right associative expressions have a clearer structure.

When recognizing and building the AST for a right associative expression, one will have a simple looking rule. In Listing 8.5 the essence of a simple

ANTLR specification is indicated. It recognizes binary plus expressions and builds a right grouped AST. Note that the the production can call itself right recursively.

Listing 8.5: ANTLR pseudo example: Plus Expression, right grouping

```
1 expressionPlusRule returns [ExpressionPlus expressionPlus] :  
2 expression1 = expressionHigherPrecedence  
3 PLUS  
4 expression2 = expressionPlusRule  
5 { expressionPlus = new ExpressionPlus(expression1 , expression2);  
6 }  
7 ;
```

When recognizing and building the AST for a left associative expression, the grammar rule is slightly more complicated. In Listing 8.6 the essence of a simple ANTLR specification is indicated. It recognizes binary plus expressions and now builds a left grouped AST. Note that there is no recursive call of the method itself. Instead an ANTLR facility, `*` is used, which has the same effect as the same symbol of a EBNF notation – it means zero or more occurrences of the content in the brackets. The rule will mach a series of expressions and plus operators. As seen in line 10 of Listing 8.6, the result variable can be overwritten such that the tree is built left grouped.

Listing 8.6: ANTLR pseudo example: Plus Expression, left grouping

```
1 expressionPlusRule returns [ExpressionPlus expressionPlus] :  
2 expression1 = expressionHigherPrecedence  
3 PLUS  
4 expression2 = expressionHigherPrecedence  
5 { expressionPlus = new ExpressionPlus(expression1 , expression2);  
6 }  
7 (  
8     expression2 = expressionHigherPrecedence  
9     {  
10        expressionPlus = new ExpressionPlus(expressionPlus , expression2);  
11    }  
12 )*  
13 ;
```

8.4 Operating on the AST

As described in Section 5.3 and in Section 7.6, a visitor is used when operating on an AST. We have defined two general visitor interfaces describing how to visit all the nodes in an AST. Listing 8.7 shows an example of how the children of `ASTDocument` are visited.

Listing 8.7: Example of a visit method in the general visit implementation

```

1 public void visit (ASTDocument astDocument) {
2     //The action for the astDocument will be performed here
3     List<? extends ASTClass> astClasses = astDocument.getClasses ();
4     for (ASTClass astClass : astClasses) {
5         astClass.accept (this);
6     }
7 }

```

Listing 8.8 shows an example of an accept method in an AST class.

Listing 8.8: Example of an accept method in InternalASTClass

```

1 public void accept (OvertureVisitorOneArg visitor) {
2     visitor.visit (this);
3 }

```

When visiting a node of type ASTDocument, this visit method will visit all its children of type ASTClass.

8.4.1 Implementation of the AST to XML Visitor

A visitor called `OvertureAst2XmlVisitor` located in `org.overturetool.eclipse.ast2xml` is used to convert an AST to an XML document. Listing 8.9 shows an extract of this visitor. It implements the generic visitor `OvertureVisitorTwoArg` and uses an JDOM element as the generic parameter. It therefore follows the design described in Section 7.8.

Listing 8.9: Visitor example for exporting an AST to XML

```

1 public class OvertureAst2XmlVisitor implements OvertureVisitorTwoArg<
2     Element> {
3     public void visit (ASTDocument astDocument, Element parent) {
4
5         storePosition ( astDocument, parent);
6         for (ASTClass astClass : astDocument.getClasses ()) {
7             astClass.accept (this, parent);
8         }
9     }
10
11     public void visit (ASTClass astClass, Element parent) {
12
13         Element omlClass = new Element ("OMLClass");
14         storePosition ( astClass, omlClass);
15         astClass.getKeywordClass ().accept (this, omlClass);
16         astClass.getIdentifier ().accept (this, omlClass);
17         ASTInheritanceClause inheritanceClause = astClass.
18             getInheritanceClause ();
19         astClass.getKeywordEnd ().accept (this, omlClass);

```

```
19     Element omlIdentifierEnd = new Element("OMLIdentifier2");
20     astClass.getIdentifierEnd().accept(this, omlIdentifierEnd);
21     omlClass.addContent(omlIdentifierEnd);
22     parent.addContent(omlClass);
23 }
24
25 public void visit(ASTIdentifier astIdentifier, Element parent) {
26
27     Element omlIdentifier = new Element("OMLIdentifier");
28     omlIdentifier.setAttribute("name", astIdentifier.getIdentifierName
29         ());
30     storePosition(astIdentifier, omlIdentifier);
31     astIdentifier.getComments().accept(this, omlIdentifier);
32     parent.addContent(omlIdentifier);
33 }
34
35 public void visit(ASTKeyword astKeyword, Element parent){
36
37     Element omlKeyword = new Element("OMLKeyword" + astKeyword.getID
38         ());
39     omlKeyword.setAttribute("value", astKeyword.getValue());
40     storePosition(astKeyword, omlKeyword);
41     astKeyword.getComments().accept(this, omlKeyword);
42     parent.addContent(omlKeyword);
43 }
44
45 public void storePosition(ASTNode astNode, Element parent) {
46
47     Element omlPosition = new Element("OMLPosition");
48     omlPosition.setAttribute("startLine", astNode.getStartLine()+"");
49     omlPosition.setAttribute("startColumn", astNode.getStartColumn()+"
50         ");
51     omlPosition.setAttribute("endLine", astNode.getEndLine()+"");
52     parent.addContent(omlPosition);
53 }
```

The visitor works by traversing an AST and continuously building the corresponding XML document. The technique is to create a new XML element every time a node is visited. This happens in line 13, 27 and 36. A reference to this created element is then parsed on when visiting the children in the AST. After all the children have been visited the element is added to the parent XML element. In this way the structure of the XML document will follow the structure of the related AST. The position information is transferred from the AST to the XML element using the `storePosition` method.

In an XML element it is not possible to distinguish two elements with the same name. We have chosen that the constructor for an `InternalASTKeyword`

class should store an ID and the string value of the keyword. By creating different element names for the different keyword it is possible to do this distinction. Moreover, handling the creation of keyword elements in a visit method instead of the body of the different visit methods saves a lot of code lines.

Listing 8.10 shows a very simple OML specification. The corresponding AST can be visited by the visitor shown in Listing 8.9 and it will produce the XML instance document shown in Listing 8.9

Listing 8.10: Very simple OML specification

```

1 class simple
2 end simple

```

Listing 8.11: XML document produced by exporting the OML specification in Listing 8.10 to XML using the visitor shown in Listing 8.9

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <OMLDocument>
3   <OMLPosition startLine="1" startColumn="1" endLine="2" />
4   <OMLClass>
5     <OMLPosition startLine="1" startColumn="1" endLine="2" />
6     <OMLKeywordClass value="class">
7       <OMLPosition startLine="1" startColumn="1" endLine="1" />
8     </OMLKeywordClass>
9     <OMLIdentifier name="simple">
10      <OMLPosition startLine="1" startColumn="7" endLine="1" />
11    </OMLIdentifier>
12    <OMLKeywordEnd value="end">
13      <OMLPosition startLine="2" startColumn="1" endLine="2" />
14    </OMLKeywordEnd>
15    <OMLIdentifier2>
16      <OMLIdentifier name="simple">
17        <OMLPosition startLine="2" startColumn="5" endLine="2" />
18      </OMLIdentifier>
19    </OMLIdentifier2>
20  </OMLClass>
21 </OMLDocument>

```

The structure of the XML instance document follows the structure of the AST tree as described in Section 7.5. An exception is that an abstract AST class will result in an element in the XML document. Writing the abstract AST class to an XML element will result in a nice XML to AST converter implementation. The visitor implementation follows a convention mentioned in Section 7.5, stating that small position values will appear before higher position values.

8.4.2 Implementation of the AST to Outline Visitor

The visitor for creating the outline and the AST outline have exactly the same structure as the visitor described in Section 8.4.1. The difference is that it is working on a simple tree structure suitable for display in Eclipse. These tree classes are called `TreeParent` and `TreeObject` and can be found in the util plug-in ⁴. While the AST to XML visitor should mirror the entire AST, the outline visitor should only show a selected part of the AST in order to give a good overview. The outline visitor is called `OvertureOutlineVisitor`, and it is shown in Appendix I.4. It extends the general `OvertureVisitorTwoArgImpl`, in order to implement it as clear as possible, as described in Section 7.8

8.4.3 Implementation of the AST to OML Visitor

The AST to OML visitor, `OvertureAst2OmlVisitor`, is used to pretty print the AST to the editor. It works using the same principle as described in Section 8.4.1. The main difference is that the built document is of type string. Listing 8.12 is an example of how the action code in a visit method looks.

Listing 8.12: Java code example: A visit method in `ast2omlVisitor`

```
1 public String visit(ASTIdentifier astIdentifier){
2     setPosition(astIdentifier);
3     doc = doc + astIdentifier.getIdentifierName();
4     columnCount = columnCount + astIdentifier.getIdentifierName().
      length();
5     setComments(astIdentifier);
6 }
```

Small position values will appear before higher position values in a XML document. When parsing from XML to AST this means that the order of the nodes are preserved and the string can therefore be stored as a local variable. It is therefore built by first setting correct position, then appending the identifier name, then update the global column value. Finally, the same is done for the comments associated the this identifier. Listing 8.13 gives an example of how positions is set.

Listing 8.13: Java code example: Using position information when pretty printing from AST to OML

```
1 public String visit(ASTNode astNode, String parent){
2     int startLine = astNode.getStartLine();
3     int startColumn = astNode.getStartColumn();
4     while (lineCount < startLine){
5         parent = parent + "/n";
6         lineCount++;
    }
```

⁴org.overturetool.eclipse.util

```

7     columnCount = 1;
8     }
9     while (columnCount < startColumn){
10        parent = parent + " ";
11        columnCount++;
12    }
13    return parent;
14 }

```

This method is called before appending a contribution to the document string. First it makes a new line until the correct line is reached and then it makes the spacing correct before appending a keyword or an identifier. The result is that an OML document that is exported to XML and imported again will look identical to the user.

8.5 Implementation of the XML to AST Converter

An XML to AST converter, called `OvertureXml2AstConverter`, been implemented as designed in Section 7.9. The conversion is done in two steps. First the XML document is loaded into a JDOM representation and validated against the XML schema. Then the XML to AST converter creates an AST from the JDOM tree. The error handling has already been done while validating the XML instance document. This makes the implementation clean and easy to read, as the converter simply traverses the JDOM tree while instantiating AST classes to build the corresponding AST. Listing 8.14 shows an example of how the XML document example used in Listing 8.11 can be imported to an AST.

Listing 8.14: Java code example: Converting XML to AST

```

1 public class OvertureXml2AstConverter implements
2     OvertureXml2AstConverterInterface{
3     public final InternalASTDocument overtureXMLDocument(Element
4         xmlDocument){
5         InternalASTDocument internalASTDocument = null;
6
7         internalASTDocument = new InternalASTDocument(xmlDocument);
8         List<InternalASTClass> internalASTClassList =
9             internalASTDocument.getClasses();
10        List<Element> classlist = xmlDocument.getChildren("OMLClass");
11        for(Element xmlClass : classlist){
12            InternalASTClass internalASTClass = overtureXMLClass(
13                xmlClass);
14            internalASTClassList.add(internalASTClass);
15        }
16        return internalASTDocument;
17    }
18 }

```

```
15
16     public final InternalASTClass overtureXMLClass(Element xmlClass){
17         InternalASTClass internalASTClass = null;
18
19         internalASTClass = new InternalASTClass(xmlClass);
20
21         Element xmlKeywordClass = xmlClass.getChild("OMLKeywordClass");
22         InternalASTKeyword internalASTKeywordClass = new
23             InternalASTKeyword(xmlKeywordClass);
24         internalASTClass.setKeywordClass(internalASTKeywordClass);
25
26         Element xmlIdentifier1 = xmlClass.getChild("OMLIdentifier");
27         InternalASTIdentifier internalASTIdentifier1 = new
28             InternalASTIdentifier(xmlIdentifier1);
29         internalASTClass.setIdentifier(internalASTIdentifier1);
30
31         Element xmlKeywordEnd = xmlClass.getChild("OMLKeywordEnd");
32         InternalASTKeyword internalASTKeywordEnd = new
33             InternalASTKeyword(xmlKeywordEnd);
34         internalASTClass.setKeywordEnd(internalASTKeywordEnd);
35
36         Element xmlIdentifier2 = xmlClass.getChild("OMLIdentifier2");
37         Element xmlIdentifierEnd = xmlIdentifier2.getChild("
38             OMLIdentifier");
39         InternalASTIdentifier internalASTIdentifier2 = new
40             InternalASTIdentifier(xmlIdentifierEnd);
41         internalASTClass.setIdentifierEnd(internalASTIdentifier2);
42
43         return internalASTClass;
44     }
45 }
```

First a new `InternalASTDocument` node is created. The position and string information in the XML element is transferred in the constructor as described in Section 8.2.2. If there are any class elements in the JDOM tree, instances of `InternalASTClasses` are created and added to the instance of the `InternalASTDocument`.

8.6 Implementing Installation Facilities

There are two important concepts when creating local or online installing facilities for plug-in on the Eclipse platform, namely update sites and features. The update site contains a file called `site.xml`, specifying a reference to the different releases of the Overture kernel. This file is used by Eclipse during the installation. In order to provide the information in a readable form for the user, the file is translated into a html file using an XSL script. The different releases is represented by a file called `feature.xml`.

It also contains all the information Eclipse needs for loading a plug-in. It specifies license issues etc., which other plug-ins are required, the versions of the different plug-ins and the version of the release associated to this feature. Based on this information Eclipse can load the different plug-ins from associated jar files. The implementation is done so that the user should accept the GNU General Public License before the Overture kernel can be installed.

8.7 Implementation of the Integration with the Eclipse Framework

The Eclipse framework focuses on extensibility and scalability. Due to the large number of plug-ins, it is essential that classes are loaded to the memory just before they are executed the first time. Each plug-in therefore contains a file called `plugin.xml` informing Eclipse which contributions the plug-in has. These contributions can both be additional functionality as well as contributions to GUI. Here is an extract from the `plugin.xml` file for the Overture editor plug-in:

Listing 8.15: XML code example: Integration with the Eclipse framework

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <?eclipse version="3.0"?>
3 <plugin
4   id="org.overturetool.eclipse.editor"
5   name="Overture Editor Plug-in"
6   version="0.0.1"
7   provider-name="Overture"
8   class="org.overturetool.eclipse.editor.OvertureEditorPlugin">
9   <runtime>
10    <library name="antlr.jar"><export name="*" /></library>
11  </runtime>
12  <requires>
13    <import plugin="org.eclipse.ui" />
14  </requires>
15  <extension-point id="extensionParser" name="ExtPoint.extensionParser" />
16  <extension point="org.eclipse.ui.editors">
17    <editor
18      name="Overture Editor"
19      extensions="oml"
20      icon="icons/sample.gif"
21      class="org.overturetool.eclipse.editor.OvertureEditor"
22      id="org.overturetool.eclipse.editor.OvertureEditor">
23    </editor>
24  </extension>
25  <extension point="org.eclipse.ui.perspectives">

```

```
26     <perspective
27         name="Overture Perspective "
28         icon="icons/sample.gif "
29         class="org.overturetool.eclipse.editor.OverturePerspective "
30         id="org.overturetool.eclipse.editor.OverturePerspective ">
31     </perspective>
32 </extension>
33 </plugin>
```

The first element specifies details for this plug-in, and it gives a reference to the `OvertureEditorPlugin` class. Afterwards, the files that this plug-in provides for others and the files it requires are specified. Then an extension point for a parser is defined, and finally, the extensions that this plug-in uses are defined. By extending the Eclipse's `org.eclipse.ui.editors` and `org.eclipse.ui.perspectives` extension points, the framework knows that some editor and perspective will be available for the user. The attribute called `name` specifies that they are called `Overture Editor` and `Overture Perspective`. The attribute called `id` is used for further reference to the editor and perspective respectively.

8.8 Implementation of the Overture Editor

The classes for the editor are located in the plug-in⁵. Diagrams presenting a design overview is given in Section 7.10. One of the packages⁶ contains the classes for creating the Overture editor with syntax highlight etc. The Overture editor is a customized version of Eclipse's standard `TextEditor`. The implementation strictly follows the design as described in Section 7.10. This customization is done by letting it use the Overture specific class `OvertureScanner` that extends Eclipse's `RuleBasedScanner`. This class sets the rules for the keyword highlight.

The focus has been on providing an easy to understand implementation of the editor. It has therefore been chosen not to use fancy features like partitions (the editor should behave differently for different part of the text), code assist (it makes a suggestion for e.g. keyword to use) and folding regions (the user can define different parts of the text that can collapse and expand). An example that these choices has made the implementation easier, is that we can use Eclipse's `FileDocumentProvider` class as document provider instead of writing our own. This class is sufficient in order to translate the editor input into a textual representation.

The Overture editor uses the class `OvertureContentOutlinePage` extending Eclipse's `ContentOutlinePage` to create the outline view. This is done by creating an instance of `TreeViewer` and associating it with the class

⁵org.overturetool.eclipse.editor

⁶org.overturetool.eclipse.editor

`OvertureContentProvider` extending the `ITreeContentProvider`.

The `OvertureContentProvider` invokes the `inputChanged` method when the user saves an OML file. It then uses the `OvertureParser` to parse the content of the editor and create an AST. Furthermore, it uses the `OvertureOutlineVisitor` to create another tree representation suitable for display in the outline viewer. Finally, the `getElements` method returning the outline tree is invoked by the system and the `OvertureContentOutlinePage` displays the outline view. When the `OvertureContentOutlinePage` is instantiated it sets the perspective to be the `OverturePerspective`. This means that only the Outline and Problem view are used when opening an OML file.

A functionality for presenting syntax errors to the user has also been implemented. In order to show the strength of Eclipse, we will here show how errors are reported to the user.

Listing 8.16: Java code example: Reporting errors to the user using markers

```

1 private ASTDocument parseOML(IDocument document) {
2     OvertureParserInterface overtureParser = null;
3     try{
4         overtureParser = overtureExtensionProvider.loadParser();
5         astDocument = (InternalASTDocument) overtureParser.parse(
6             new ByteArrayInputStream(document.get().getBytes()));
7         errors = overtureParser.getErrors();
8         if (!errors.isEmpty()) {
9             for (OvertureParseException error: errors) {
10                addMarker(input.getFile(), error.getError(),
11                    new Integer(error.getLine()),
12                    new Integer(IMarker.SEVERITY_ERROR));
13            }
14        } catch (Exception e) {
15            e.printStackTrace();
16        }
17        return astDocument;
18    }
19
20    private void addMarker(IFile file, String message, int lineNumber,
21        int severity) {
22        try {
23            IMarker marker = file.createMarker(IMarker.PROBLEM);
24            marker.setAttribute(IMarker.MESSAGE, message);
25            marker.setAttribute(IMarker.SEVERITY, severity);
26            if (lineNumber == -1) { lineNumber = 1; }
27            marker.setAttribute(IMarker.LINE_NUMBER, lineNumber);
28        } catch (CoreException e) {
29        }}

```

Listing 8.16 shows the method that is invoked to start a parsing. First

the parser is loaded, the parsing is performed and the result is stored, and finally errors are retrieved if present. The method `AddMarker` is then capable of adding the errors to Eclipse's problem view. It has been decided that the outline view should only be updated when the user saves an OML document, and not as it is the case in the Java editor when the user alters a Java document. We think it from a performance perspective is unnecessary to parse a specification continuously while the user might still be typing in the editor.

We have created two wizards for finding file resources in the project in the workspace. The first one extends Eclipse's Wizard and implements Eclipse's `INewWizard`. This `OvertureWizard` is also in charge of writing a file to a project in the workspace. The `OvertureExtendedWizard` extends `OvertureWizard` and it can handle two file resources in stead of one. The simplest of them is used for asking the user where to export a file to, whereas the extended wizard is used when asking the user which file to import and under what name to store it. As one can imagine the size of the code for the extended wizard is written much more compact, as most of the functionality is inherited.

8.9 Extending the Kernel Plug-in

As discussed in Section 7.1 the kernel is build using extension points and extensions. The editor plug-in ⁷ is a central part of the kernel, as it provides extension points for the other plug-ins, such as the parser and the converters. The requirement for defining an extension is that extensions should implement an interface specified by the editor.

The editor uses the class `OvertureExtensionProvider` to load the extensions. In Listing 8.17, there is an example of how a parser is loaded. The method called `loadParser` uses the method called `loadExtensionRegistry` to get an interface for a configuration element used to load the parser class for the Overture parser. The `loadExtensionRegistry` works by calling Eclipse's plug-in registry in order to find the parser plug-in that extends the parser extension point. Eclipse knows the available extension points and extensions from XML files called `plugin.xml` provided by the installed plug-ins.

Listing 8.17: Loading a Parser in the `OvertureExtensionProvider`

```
1 public OvertureParserInterface loadParser() {
2     IConfigurationElement config = loadExtensionRegistry("
      extensionParser");
3     if (config.getName().equals("parser")) {
4         parser = (OvertureParserInterface) config.
          createExecutableExtension("class");
5     }
```

⁷org.overturetool.eclipse.editor

```
6     return (OvertureParserInterface) parser ;
7     }
8 }
9
10 public IConfigurationElement loadExtensionRegistry(String extensionName
11     ){
12     IExtensionRegistry pluginRegistry = Platform.getExtensionRegistry();
13     IExtensionPoint point = pluginRegistry.getExtensionPoint(
14         OvertureConstants.PLUGIN_ID, extensionName);
15     IExtension[] extensions = point.getExtensions();
16     IConfigurationElement[] configElements = currentExtension.
17         getConfigurationsElements();
18     return configElements[0];
19 }
```

Notice that the `loadParser` method type casts the parser implementation to the interface `OvertureParserInterface`. This interface works as a proxy for the editor, meaning it can compile the code without knowing the existence of any parser implementations.

8.10 Summary

The kernel has been implemented as intended. For tests of the implementation, we refer to Chapter 9. The implemented kernel offers a flexible AST structure with support for different visitors. The parser respects precedence and grouping conventions and builds the AST's accordingly. With respect to XML support, the intension were to be able to export to XML and afterwards import from XML without loss of any kind of information. This has also succeeded. The created XML schema, which exports AST's naturally satisfies, should make it possible for other tools to exchange XML based information. The solution uses many Eclipse facilities and supports additional Eclipse based plug-ins and extension points and as intended, the kernel itself is also composed of separate plug-ins. It is therefore really simple to add additional functionality, as new plug-ins can be added without recompilation of the existing code.

The implementation was done in two iterations, so that the kernel was first implemented to support a minor subset of the OML language. Through this iterative process, we had a chance to verify that the design would work in practice before implementing support for all of OML.

Chapter 9

Test

This chapter describes how the kernel has been tested. The principles for testing is described, and there are given examples of test cases. It is described how the test results are evaluated, and this is also exemplified. Finally it is discussed which additional test principles, that could be used for the kernel.

Numerous tests has been performed on the kernel. A number of test cases and their results has been selected to show how the tests has been performed. A sample of test cases is shown in Appendix J. The entire set of test cases are available on the cd-rom, described in Appendix B.

Because of the size of the kernel, we have decided not to do a structural test but instead focus on a comprehensive functional test of the entire kernel.

9.1 Functional Test of the Parser

The functional test has been divided into four different parts, testing different aspects of the parser. Here is an overview of how the test is structured:

- Part one contains 167 test cases for each of the different language structures. All branches of the syntax found in the language manual[12] is tested to verify that the parser can recognize the entire OML language and build the corresponding AST correctly.
- Part two contains 37 test cases from the VDM++ book[6]. The purpose of these tests is to show that the parser works on larger examples.
- Part three contains 4 larger test cases from the VDM++ book[6]. In order to test how the parser reacts to different combinations of errors, 13 errors is made.
- Part four contains 4 small test cases. The purpose is to show that the parser is able to handle precedence as well as grouping correctly. An overview of how the parser handles this can be found Appendix F.

The four test files containing the four parts can be found on the cd-rom in the test folder. An overview of the content of the CD can be found in Appendix B.

9.1.1 How the Tests are Performed

This section describes the procedure for how the kernel is tested. In addition to this, it serves as a guide for how to run the tests.

In order to test this the full source code for the kernel should be installed and the test should be performed from the Eclipse development environment. Appendix C.5 describes how to install the full source code. When running the tests the outline view `org.overturetool.eclipse.outline` should be deselected. This plug-in is used to give an overview of the code when writing OML specifications.

The plug-in called `org.overturetool.eclipse.astoutline` is used to show the structure of the AST that is build after a OML specification has been parsed. In order to verify that the AST is build correctly according to the precedence and grouping conventions, it is important to be able to view this tree structure. The reason why the two outline views should not be installed at the same time is that they are using the same extension point.

In order to test that the parser can build an AST and import and export to XML, it is sufficient to install the Overture kernel as described in Appendix C.2.

The four parts are tested following these steps:

1. A new project is made and the four test files are imported from the CD into the project.
2. Each test file is opened. Every time it is saved, all the test cases it contains are parsed. If there are no syntax errors, an AST is build and an outline view shows the AST structure.
3. After a successful parse, we have manually examined the outline view of the AST structure in order to verify that the parser recognizes the OML specifications correctly.
4. If there are syntax errors, the error messages are shown to user in Eclipse's problem view. These messages has manually been examined as well.
5. For each test case the name, the expected output and the result is written immediately before the test case.

9.1.2 Test of Part1.oml

The test file called Part1.oml contains test cases for the different language structures. Listing 9.1 shows an example of a test case and the format for specifying the name of the test case, the expected output and the result.

Listing 9.1: Test Part 1, Example of test of a production rule

```

1 --- case13           : bracket type
2 --- tested using    : -
3 --- expected output : correctly built tree
4 --- result          : as expected
5
6 var1 = (bool);
```

The strategy in this test is to find the simplest way to specify each language structure. All branches from the language manual have been tested in this way. If a language structure contains other language structures, the simplest ones have been chosen.

Whenever there is an optional language structure, tests has been written both with and without these. For lists, both the empty list and a list with one element has been tested.

Results

The examination of the test results for the 167 test cases shows that the parser is able to recognize the entire OML language and build a correctly structured AST. This means that the parser is able to operate on any OML specification that satisfies that syntax and follow the precedence and grouping rules described in Appendix F.

9.1.3 Test of Part2.oml

The test file called Part2.oml contains larger examples of OML specifications. Listing 9.2 shows an example of a test case.

Listing 9.2: Test Part 2, Example testing with larger OML example

```

1 --- case219 um       : chapter9/Full/Alphabet
2 --- tested using    : -
3 --- expected output : correctly built tree
4 --- result          : as expected
5 class Alphabet
6
7 instance variables
8   alph : seq of char := [];
9
10 inv AlphabetInv(alph)
11
```

```

12 functions
13   AlphabetInv: seq of char -> bool
14   AlphabetInv (palph) ==
15     len palph mod 2 = 0 and
16     card elems palph = len palph
17
18 operations
19   public Alphabet: seq of char ==> Alphabet
20   Alphabet (pa) == alph := pa
21   pre AlphabetInv(pa);
22
23   public GetChar: nat ==> char
24   GetChar (pidx) == return alph(pidx)
25   pre pidx in set inds alph;
26
27   public GetIndex: char ==> nat
28   GetIndex (pch) ==
29     let pidx in set {i | i in set inds alph
30                       & alph(i) = pch} in
31     return pidx
32   pre pch in set elems alph;
33
34   public GetIndices: () ==> set of nat
35   GetIndices () == return inds alph;
36
37   public GetSize: () ==> nat
38   GetSize () == return len alph;
39
40   public Shift: nat * nat ==> nat
41   Shift (pidx, poffset) ==
42     if pidx + poffset > len alph
43     then return pidx + poffset - len alph
44     else return pidx + poffset
45   pre pidx in set inds alph and
46     poffset <= len alph;
47
48   public Shift: nat ==> nat
49   Shift (pidx) == Shift(pidx, 1)
50 end Alphabet

```

In this test it is verified that larger OML examples can be handled as intended by the kernel.

Results

The examination of the test results for the 37 test cases shows that the parser is able to parse real life OML specification that satisfies that syntax and follows the precedence and grouping rules described in Appendix F

9.1.4 Test of Part3.oml

The test file called Part3.oml contains four examples of OML specifications. Listing 9.3 shows an example of one of the test cases containing some errors. It contains errors in lines 30, 40, 54.

Listing 9.3: Test Part 3, Example of testing that the parser can find several errors

```

1  — case304 um          : chapter9/Full/TestCase
2  — tested using      : —
3  — expected output   : is able to locate the syntax errors
4  — result            :
5  — line 156:34: expecting "then", found 'hen'
6  — line 169:29: expecting RBRACKET, found 's'
7  — line 170:7: unexpected token: in
8  — line 172:15: unexpected token: ;
9  — line 183:15: expecting DOUBLEEQUAL, found '='
10
11 — comment          : some times the parser recorgnise the error in another
12   way as
13   expected. However it finds the line where the error
14   occurs.
15   The reason is of cause that it is unaware of that the
16   user
17   wants to specify.
18
19 class TestCase
20   is subclass of Test
21
22 instance variables
23   name : seq of char
24
25 operations
26   public TestCase: seq of char ==> TestCase
27   TestCase(nm) == name := nm;
28
29   public GetName: () ==> seq of char
30   GetName () == return name;
31
32   protected AssertTrue: bool ==> ()
33   —AssertTrue (pb) == if not pb then exit <FAILURE>;
34   AssertTrue (pb) == if not pb hen exit <FAILURE>;
35
36   protected AssertFalse: bool ==> ()
37   AssertFalse (pb) == if pb then exit <FAILURE>;
38
39   public Run: TestResult ==> ()
40   Run (ptr) ==
41   trap <FAILURE>

```

```

39     with
40         —ptr.AddFailure(self)
41         ptr.AddFailure(self.s)
42     in
43         (SetUp());
44         RunTest();
45         TearDown();
46
47     protected SetUp: () ==> ()
48     SetUp () = is subclass responsibility;
49
50     protected RunTest: () ==> ()
51     RunTest () = is subclass responsibility;
52
53     protected TearDown: () ==> ()
54     —TearDown () = is subclass responsibility
55     TearDown () = is subclass responsibility
56
57 end TestCase

```

The purpose of this test is to verify that the parser can find several errors. We have found it efficient with these four examples containing 13 errors. The error messages provided by the parser are shown as comments in lines 5 to 9.

Results

The examination of the test results for the 4 test cases shows that the parser is able to find all 13 syntax errors. Each error message contains line and column for where the error is located, which token it expects and which token it has found. In most cases the parser gives a precise description of the error, whereas it in a few cases is only able to tell that there is an error and present the first unexpected token. An error in a specification may be identified as multiple errors by the parser, if the first error leads the parser to follow a different rule than intended.

Error handling is a study in itself. There are several methods and theories for identifying errors, but we have chosen to let ANTLR do the error handling. We have found the standard error handling mechanism of ANTLR suitable for the parser, as it (at least in our tests) gives accurate error messages and is good on recovering from errors to find potential additional errors.

9.1.5 Test of Part4.oml

The test file called Part4.oml contains two test cases showing that the parser can handle precedence and grouping. Listing 9.4 shows two test cases for handling of precedence and grouping.

Listing 9.4: Test Part 4, Test of precedence and grouping

```

1  --- case400           : precedence
2  --- tested using    : -
3  --- expected output : correctly built tree
4  --- result          : as expected
5  class precedence
6  values
7  var1 = true or false and (true);
8  var2 = true and false or (true);
9  end precedence
10
11 --- case401           : grouping
12 --- tested using    : -
13 --- expected output : correctly built tree
14 --- result          : as expected
15 class grouping
16 types
17 var3 = int | bool | nat | char;
18 var4 = int -> bool -> nat -> char ;
19 end grouping

```

Precedence is tested by constructing an expression containing different combinations of the two binary operators **and** and **or** and verifying that the AST is correctly build. The two type constructs **union type** and **partial function type** has left and right grouping respectively and they have therefore been used to test that the parser can handle grouping. All other language structures which are influenced by precedence or grouping rules are implemented using a similar techniques as these four language constructs. They have all to some extend been tested during development and they are working correctly. However, a full test of precedence would be very time consuming because of the number of combinations.

Results

The examination of the test results for the 2 test cases shows that the parser is able to handle precedence as well as grouping correctly. This can be seen by inspecting the views of the threes presented in Figure 9.1 and 9.2.

In Figure 9.1, the 'and' operator has higher precedence than the 'or' operator in both a and b. The node representing 'or' is therefore nearer to the root than the node representing 'and'. This is, as expected, the case in both the a and b branches of the test.

In Figure 9.2, the a and b parts of the test gives different structured trees, as 'union' has left grouping whereas 'partial function type' has right grouping. In part a, where there is an expression with multiple union operators, the tree shows that this is recognized respecting that union is (in this implementation) left associative. In the b part of the test, the expression

with multiple partial function type operators, the tree is built with respecting that partial function type is right associative. The trees are built as expected.

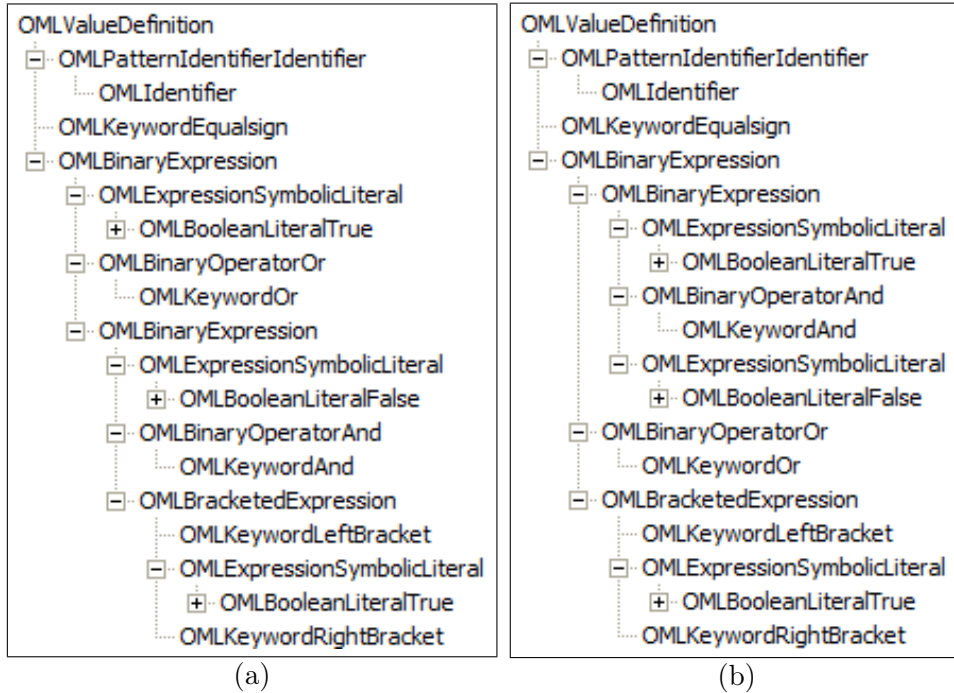


Figure 9.1: Outline view showing the structure of selected parts of an AST.

9.2 Functional Test of Import and Export Facilities

The functional test for exporting to XML, importing from XML and pretty printing is performed for all the test cases in Part1.oml, Part2.oml and Part4.oml.

9.2.1 How the Tests are Performed

First, each of the three test files is opened in the Overture editor and parsed in order to build an AST. Then the AST is exported to XML, and thereafter imported to a new OML file using the wizards in the Overture menu. Finally, the test file and the newly generated OML file are compared using the file compare program in Eclipse.

Results

Using the file compare program, one can conclude that each of the test files is equivalent to the corresponding generated OML file. Furthermore, all

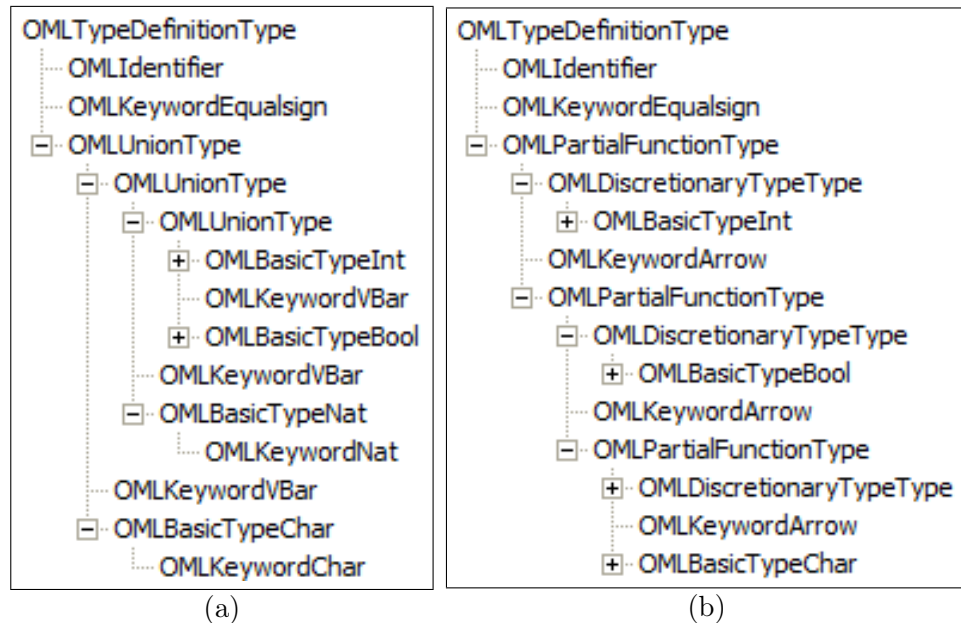


Figure 9.2: Outline view showing the structure of selected parts of an AST.

the corresponding XML instance documents are valid according to the XML schema. Therefore, the results of these tests shows that export, import and the different visitors are working correctly.

9.2.2 Test of the Eclipse Functionality

The kernel uses several extension points from Eclipse and provides several extension points for the parser and visitor plug-ins. A selected part of the Eclipse functionality has therefore been tested:

- The outline view has been test on an OML specification. The expected behavior is that the content of the outline view reflects the structure of the corresponding OML specification. This has been tested using the file called Part5.oml in the test folder on the cd-rom. The program behaves as expected.
- The wizard for creating a new OML file using the file menu and using the "Export to XML" menu item in the Overture Menu. Both has been tested even though it uses the same code. The expected behavior is that the name of the OML file is validated and the file is created. The program behaves as expected.
- The wizard for importing from an XML file to an OML file. The expected behavior is that the names are validated and a file is imported. The program behaves as expected.

- Opening of the Overture Perspective when opening an OML file. The expected behavior is that the Overture Perspective opens and only shows the two relevant views. The program behaves as expected.
- The editor is independent of the extensions. The expected behavior is that two plug-ins can extend the same extension point. The program behaves as expected. This has only been tested for the two kinds of outline visitors.
- Importing an invalid XML document. The expected behavior is that the kernel informs the user where the error is and what it is. The program behaves as expected.

9.3 Performance Issues

Though the parser can parse all the language structures, four of the test cases takes between one and two minutes to parse. The language constructs `let statement`, `let be statement`, `def statement` and `if statement` works perfectly, whereas `let expression`, `let be expression`, `def expression` and `if expression` take longer to parse even they are nearly identical. Performance has not been the main focus in this project. It is probably small modifications that need to be done to improve performance, but resources have to be allocated to identify the causes.

All test have been performed on a laptop with 1.70GHz Intel Pentium processor and 512 RAM. Parsing the four time consuming language constructs results in full use of CPU as well as RAM. It would therefore make a significant difference to execute the Overture Tool Set on a very fast computer with considerable more RAM. Moreover, Eclipse has facilities to make use of multiple processors. This is further described in Section 11.2.

9.4 Summary

The testing of the kernel was divided into four parts. The result of the first part showed that the parser can recognize the entire OML language and build a correctly structured AST according to the OML language specification[12]. The second part showed that the parser works on real life OML examples. The third part showed that the parser is able to find several and the fourth part shows that the parser is able to handle precedence and grouping correctly. Additional tests show that all the test cases that can parse can also be exported and imported correctly from XML.

Chapter 10

Additional Created Plug-ins

To show that the developed kernel is indeed easy extendible, we have developed, implemented, and tested some additional plug-ins for the kernel. As these additional plug-ins are not core-components of the kernel, we have chosen to describe analysis, design, implementation, and test of them separately in this chapter. The additional plug-ins should be seen as 'proof of concept' – they are intended to show that the kernel is extendible and provide examples of how to extend the kernel.

Section 10.1 describes some principles for using refactoring of the AST to do different tasks. It also outlines how a 'proof of concept' implementation of a refactoring visitor has been implemented and tested.

Section 10.2 examines how an OML specification can be exported and imported to and from UML diagrams. It then outlines how two 'proof of concept' programs for doing this have been implemented and test.

10.1 Refactoring

Refactoring is an important concept for a development tool. This section will outline how refactoring facilities can be added to the kernel. A simple refactoring plug-in has been implemented to illustrate the possibilities of using refactoring.

10.1.1 Analysis

Refactoring facilities for the kernel can add important advanced facilities to the editor. In the following we give an analysis of a few refactoring possibilities.

The Overture Editor has an outline view showing e.g. identifier and operation names. If one wants to alter e.g. an identifier name, the name should be changed all the places it occurs within scope for this identifier. In the Eclipse Java editor it is possible to rename an identifier by right clicking on

it in the outline view. A similar functionality could be added to the Overture editor. Refactoring could also be used for a number of other tasks like auto-formatting an OML file, or generation of get methods.

The tree structure in the Overture outline view is created using a tree node called `TreeParent`. In the current implementation this tree node contains position information for the corresponding AST class. When left clicking on a node represented by e.g. an identifier name, the line where the identifier is placed in the editor is highlighted. If the tree node also contains a reference to the corresponding AST class it could be possible to right click on a node and type a new name for the identifier. A visitor can then be used to refactor the AST in order to e.g. re-name all occurrences of a particular identifier.

10.1.2 Design and Implementation

A 'proof of concept' plug-in called `org.overturetool.eclipse.refactoring` has been made to show how refactoring can be done using a visitor. It can rename identifiers for a class.

In order to make it easier to test this plug-in, it extends the outline extension point provided by the Overture editor plug-in.

The implementation of the editor makes it relative simple to add e.g. renaming functionality as described in Section 10.1.1. For further development a good place to start is to take a look at `OvertureContentOutlinePage` in the in the editor plug-in.

10.1.3 Test

Before this test can be performed the full source code for the kernel and the additional plug-ins should be installed. Appendix C.5 describes how to do this. When running the plug-ins from the Eclipse development environment, the plug-in called `org.overturetool.eclipse.refactoring` should be selected instead of the outline visitors. Every time an OML file is saved, it is parsed and the refactoring visitor alters the class name to new `NewClassName` before pretty printing the AST to the console. By manually inspecting the output one can see that the name has changed as expected.

10.2 Import and Export to UML

UML is a powerful graphical notation that provides different kinds of diagrams to illustrate many aspects of software design. The diagrams give a structural overview. Having facilities that ease co-existence of UML diagrams and formal specifications, will make it easier to combine software development methods, as it will be possible e.g. to start by illustrating the

overall structure in UML and afterwards specifying the system more detailed in OML.

10.2.1 Analysis

UML is a widely used diagram notation, especially in the Java developer community. The different aspects of UML diagrams can of course also be used to illustrate development in other languages, including OML. A class diagram is the type of diagram having the strongest relation to an OML specification and we will therefore focus on this type in the implementation. The class diagram on Figure 10.1 represents a class with the name Info containing an attribute with the name age, the type int and the visibility private. It should be possible to convert the OML specification shown in Listing 10.1 to and from UML.

Listing 10.1: Small OML example suitable for export and import to UML

```
1 class Info
2
3 types
4 private age = int
5
6 end Info
```

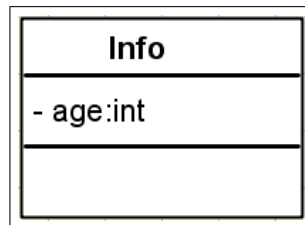


Figure 10.1: Example of an UML class diagram

10.2.2 Design and Implementation

This section will give an overview of how an OML specification can be exported and imported to UML, as well as explain some of the techniques and implementation details used to do this conversion. UML diagrams can be represented as an XMI file following the same structure as an XML file[14]. Figure 10.2 shows an overview of how to convert between OML and UML. The interesting conversions are between XML and XMI as the others conversions are part of the kernel. The most efficient way to do this transformation is to use an XSL (Extensible Stylesheet Language) document.

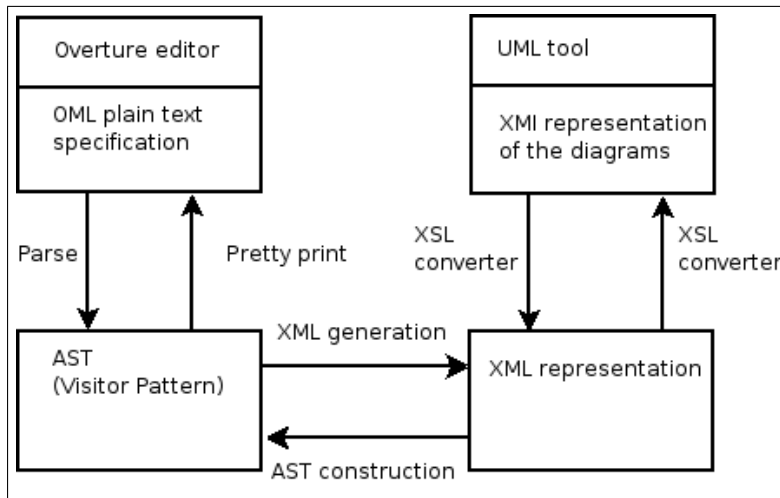


Figure 10.2: Overview of how to convert between OML and UML.

An XSL stylesheet is basically a set of templates. Each template matches a set of elements in the input XML document and describe how each of these should be transformed. In a template it is possible to reach any information in the XML input document using an XPath expression. Furthermore, templates can be applied for the corresponding children elements. The transformation starts when the root element is matched[15].

Using the technique described above, we have implemented an XSL document for converting between an XML representation of a small OML program and a XMI representation of the corresponding UML class diagram. The program for importing is placed in `org.overturetool.eclipse.uml2xml` and the program for exporting is called `org.overturetool.eclipse.xml2uml`. Listing 10.2 show the BNF grammar for this subset. It is not clear[14] to what extent the different UML tool follows the standard for importing and exporting UML to XMI. We have therefore chosen to use Posidon [23] as the UML tool, and use their way of representing positions of the graphical objects.

Listing 10.2: BNF of the OML language subset that can be imported and exported to UML

```

1 Document = class
2
3 class = 'class ', identifier ,
4
5         [ class body ]
6
7         'end', identifier ;
8

```

```
9 class body = definition block
10
11 definition block = type definitions
12
13 type definitions = 'types', access type definition;
14
15 access type definition = [access], type definition;
16
17 access = 'private',
18         | 'public',
19         | 'protected',
20
21
22
23 type definition = identifier = type;
24
25 type = int;
```

10.2.3 Test

The programs for importing and exporting have been tested using the following procedure. Before starting this test the Overture kernel should be installed as described in Appendix C.2.

First the OML specification shown in Listing 10.1 was exported to an XML document called `PersonInformation.xml`. This XML document was then imported to the Java project `org.overturetool.eclipse.xml2uml`. Then the XML document was converted to the corresponding XMI representation in a file called `PersonInformation.xmi` using the program called `OvertureXml2Uml`. Finally, this file was opened in the UML tool called Posidon. The result was the UML diagram shown in Figure 10.2. By manually inspection one can see that it produces the correct result.

After saving the UML diagram in a file called `PersonInformation.xmi`, it was imported to the project called `org.overturetool.eclipse.uml2xml` and converted to a file called `PersonInformation.xml` using the program called `OvertureUml2Xml`. This file was then imported to a project in the Overture editor. After manually formatting the file by changing the position information the file was imported from XML using the Overture editor. The imported XML file looks like the one in Figure 10.2. The two programs therefore work as intended. The program has been tested on different examples and it can therefore handle any examples following the BNF described in Listing 10.2.

10.3 Conclusion

This chapter gave examples of how the Overture kernel can be further developed. A 'prove of concept' program for doing refracturing of the AST was implemented, and the prospects of using refactoring to do different tasks was discussed. The test showed that it is possible to write refactoring plug-ins. Two 'prof of concept' programs for importing and exporting OML to UML have been implemented and tested. The test showed that it is possible to do this conversion.

Chapter 11

Future Work

The intension of this chapter is to serve as inspiration for future development of tools based on the kernel developed in this project. The chapter will illustrate how the kernel has been designed to be prepared for extensions, suggest potential new features, and illustrate how existing formal method tools might be used in development of new tools.

11.1 Guide to Extending the Kernel

This chapter will describe how to extend this kernel. We will both illustrate what will need to be changed if the language is changed, as well as present some ideas of how the implementation can be extended by additional plug-ins extending the provided extension points. In addition, we have listed some ideas for future plug-ins.

11.1.1 New operator

With the chosen design, the parser has been implemented, such that the parser can be modified by changing one file – the parser specification file. If e.g. a new operator is added to the language, the following adjustments will be necessary.

The necessary modifications are listed in H.

11.2 Ideas for Future Plug-ins and Eclipse Features

This section suggests future plug-ins or features for the Overture project. Some of the topics were discussed at the overture workshop.

Functionality	Description
Plug-in Suggestions	
Static semantics checker	Plug-in checking static semantics of an OML model.
Dynamic semantics	Interpreter/debugger with dynamic semantics capabilities.
Code generator	Generation of e.g. Java, C++ or C# code.
Support for test cases	Test case facilities, test coverage analysis. Possibly test case generation or generation of fault injection test cases. How can Eclipse illustrate e.g. test coverage? Automatic tests?
Proof obligation generator	Tool support for automatic generation of proof obligations.
Proof support	Proof support in general.
Real time support	Support for specification of real time systems. Adding e.g. VICE to the supported language. Facilities for monitoring execution of real time systems.
Extended static semantics checker	Static semantic checks of e.g. concurrency aspects.
API support	Extended API support.
Reverse engineering	Generate OML specification from e.g. Java, C++ or C# source code.
Useful Eclipse Features	
Use natures and builders	Builders can register if files in a folder has been changed. By using builders, one can let Eclipse manage parsing, static semantics checking, exports, etc. automatically on only those files that has changed.
Use monitors and jobs	Running an operation as a job provides a facility of executing the job in the background while the user can continues working. Using this feature can enable highlighting errors on-the-fly while the user types the specification.

Plug-ins developed for the kernel should use facilities from the Eclipse framework. An example of a well-integrated editor is the build in open source Java editor. By inspecting the facilities this editor provides, developers can get valuable inspiration of how to make use of Eclipse's features. It will be a challenge for plug-in developers to consider how the different tools can integrate with Eclipse. Most of the requested tools are inspired by VDM-

tools. However Eclipse may offer facilities for some plug-ins, that can give the tool a different, but much better user interface.

An example could be a presentation of test coverage analysis. With VDM-tools in mind, one might think of this as a pretty printing facility with highlights indicating the tested parts of the code. However, for Eclipse, it may be beneficial to create a test perspective, that could present at set of views with all test related facilities. This could include an overview of all test cases, details for each test cases with comments, and a view showing source code with highlighting of tested sections of the code.

11.3 Approaches to use Formal Specification for Future Development

How to structure the development of each plug-in is up to the developers. Plug-in developers may though want to specify selected plug-ins using formal specification techniques in e.g. OML. The kernel has been developed using common, non-formal development methods. Therefore, exists no formal specification of either the kernel nor the AST classes.

During the project, it has come to our knowledge that there exists VDM++ specifications defining both the static semantics and the dynamic semantics of VDM++. It is obvious to investigate how these specifications can be used in development of static and dynamic semantic plug-ins. It is though a problem that these specifications are currently owned by CSK, but CSK may later choose to release them.

Assuming the developer has VDM++ specifications available for the static and dynamic semantics, it will be obvious to use these in the development. The commercial version of VDM-Tools can be used on these specifications to auto-generate Java code. If all code for static and dynamic semantics support is to be generated with VDM-Tools, it will be useful to have a VDM++ representation of the AST classes of the kernel. The commercial version of VDM-Tools has a module for reverse engineering from Java code to VDM++ specifications. This will be a fast way to create a VDM++ specification that is consistent with the implementation of the AST classes.

Following the indicated strategy, it will not be hard to use formal development methods for development of selected plug-ins. There may be minor problems, as the reverse engineering module of VDM-Tools does not support generic types and the for-iterator of Java 1.5. Fixing errors due to this is though manageable.

Chapter 12

Conclusion

This chapter summarizes the results of the project and compare these with the initial intentions of the project. A discussion of the results is given, and the projects ability to serve as a kernel for future development of the overture project evaluated. Finally some concluding remarks is presented.

12.1 Achieved Results

The overall objective of this project was to provide an extendible kernel for an Overture Tool Set using the Eclipse framework. The functionalities identified as basic were parsing an OML specification to an AST, exporting an AST to XML, importing from XML to an AST, and printing an AST out as a plain OML specification.

We have investigated available tools and techniques and analyzed how these could be used in the kernel development. In order to meet all requirements for the kernel, we have chosen to implement the parser with the parser generation tool ANTLR, and to create the AST classes and their interfaces by hand. This was evaluated to give the most flexible solution. We identified that by implementing the AST classes with support for Visitor Design Pattern, we could provide a data structure completely independent from action code implementations. JDOM was chosen as an appropriate tool for importing and exporting to XML.

By investigating the facilities provided by Eclipse, we identified possible Eclipse Framework features that was suitable for integrating the kernel with Eclipse.

When designing the system, we focused on developing the overall design to be as flexible as possible. With this design, the kernel itself is flexible with its modular structure, and the system is prepared for extensions and further development.

The kernel is designed so that modules are connected using the Eclipse plug-in structures. Extension points are provided – both internally between

the kernel components and towards the external plug-ins.

The kernel has been implemented and tested, and provides all the requested facilities. In the implementation, a range of Eclipse facilities has been extended to provide some useful views. The implementation has also been packaged such that it is ready for distribution on an Eclipse update site on the internet.

The report discusses the possibilities of how the kernel can be extended. We have chosen to implement some non-kernel functionalities – a refactoring plug-in, and conversion facilities to/from UML. The intension of creating these additional modules has been to show in practice that the kernel is extendible as intended. The refactoring plug-in extends an extension point offered by the kernel, whereas the UML facilities operates on AST's exported to XML. In addition, a range of ideas for future development of the Overture Tool Set is presented.

12.2 Discussion of Results

The kernel supports the entire OML language and builds AST's respecting the precedence and grouping conventions of OML. The structure of the AST classes fully fulfills the identified requirements and is therefore a flexible base for any kind of tool that should operate on the AST like the kernel plug-ins. The additional created plug-in also shows that the kernel is prepared for extensions that alters the AST. The implemented UML facilities shows that the XML representation can be used to exchange information with other tools or formats.

The modular plug-in structure makes it easy for someone to interchange parts of the kernel with new modules. If an improved outline view is for instance developed, it can easily replace the existing without modifying any other part of the kernel. Thereby the kernel is also prepared for situations where different development teams experiments creating new additional tools. Tools can be implemented independently for the kernel. Furthermore, it is easy to manage different versions – a situation that might be relevant if different development groups develops different versions of the same plug-in.

12.3 Concluding Remarks

This master thesis project has been a valuable experience giving us 'hands on' experience on developing a system that is part of a larger project. The project has been full of technical challenges, where we have been able to apply theory from many different software development fields. In addition, the collaboration and co-ordination with the overture core group throughout the project has developed our skills in communication. Discussing ideas and

designs with people across the world on instant messaging net meetings is a challenge, but proved to be very effective.

It is our hope that the developed kernel can work as a base for the future open source development of the Overture Tool Set. It is our belief that the developed kernel meets the expectations and needs that were present when we chose to develop this kernel as our M.Sc. thesis project. The kernel is well tested and in addition we have created examples of additional plug-ins.

Hopefully the Overture project and the Overture Tool Set will, over time, develop as intended. Reaching the goals of having a truly open source tool set of industrial strength with advanced capabilities such as code generation, would make OML a powerful formal specification language with potential for wide use. If the overture project is open to new research and innovative development, the overture tools may be a step forward in making formal methods more usable in real life software development.

Bibliography

- [1] G. Bracha. *Generics in the Java Programming Language*. SUN, July 2004. <http://java.sun.com/j2se/1.5/pdf/generics-tutorial.pdf>.
- [2] H. Bruun. Parser notes, 02120. Lecture notes on parsing, written for course 02120 at Technucal University of Denmark, fall 2004 and spring 2005.
- [3] J. D’Anjou, S. Fairbrother, D. Kehn, J. Kelleman, and P. McCarthy. *The Java Developer’s Guide to Eclipse*. Addison-Wesley, second edition, 2005.
- [4] E. G. et al. *Design Patterns*. Addison-Wesley, 1995.
- [5] J. Fitzgerald and P. G. Larsen. *Modelling Systems*. University Press, Cambridge, 1998.
- [6] J. Fitzgerald, P. G. Larsen, P. Mukherjee, N. Plat, and M. Verhoef. *Validated Designs for Object-oriented Systems*. Springer, New York, 2005.
- [7] U. Hjarnaa. Translation of a subset of RSL into java. Master’s thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2004. Supervised by Associate Professor, Ph.D. Anne E. Haxthausen and Associate Professor Hans Bruun.
- [8] P. Mukherjee. A blueprint for development of language tools. Memo produced as input to Overture, 2004.
- [9] P. van der Spek. The overture project, designing an open source tool set. Master’s thesis, Delft University of Technology, 2004.
- [10] P. van der Spek. *The Overture Project, Towards an open source toolset*. Research report, Delft University of Technology, January 2004. <http://www.overturetool.org/downloads/researchassignment.pdf>.
- [11] D. A. Watt and D. F. Brown. *Programming Language Processors in Java*. Pearson Education Limited, 2000.

BIBLIOGRAPHY

- [12] *VDMTools, The VDM++ language*. a language specification document defining the VDM++ language supported by VDMTools, http://www.vdmbook.com/langmanpp_a4_001.pdf.
- [13] *Homepage of the Overture Project*. <http://www.overturetool.org>.
- [14] *UML, XMI and code generation, Part 1-4*. <http://www-128.ibm.com/developerworks/views/xml/libraryview.jsp>.
- [15] *XSL – XML Stylesheets*. <http://www.informatics.sussex.ac.uk/courses/it/it04S09.html>.
- [16] *Homepage of SAX*. <http://www.saxproject.org/>.
- [17] *Homepage of JavaCC*. <https://javacc.dev.java.net>.
- [18] *Homepage of ANTLR*. <http://www.ANTLR.org>.
- [19] *Homepage of JAXB*. <https://jaxb.dev.java.net>.
- [20] *Homepage of JTB*. <http://compilers.cs.ucla.edu/jtb/>.
- [21] *Homepage of JDOM*. <http://www.jdom.org/>.
- [22] *Homepage of The OSGi Alliance*. OSGi, an open service platform for delivery and management of applications and services as networked applications, <http://www.osgi.org/>.
- [23] *Homepage of Poseidon*. <http://www.gentleware.com/>.
- [24] *Homepage of XML spy*. <http://www.altova.com/>.

Appendix A

Term List and Abbreviations

This appendix defines abbreviations and terms used in the report.

ANTLR ANOther Tool for Language Recognition. Parser generation tool that in general generates *k-lookahead* recursive decent parsers, but is capable of using dynamic lookahead on selected production rules. Explained in Section 5.5.

AST Abstract Syntax Tree. A data structure that representing a language specification. Explained in Section 5.3.

BNF Backus-Naur Form. Notation to specify a language.

EBNF Extended Backus-Naur Form. A notation similar to *BNF*, but with additional notation possibilities.

Eclipse An extendable tool framework that can serve as a base for different kinds of tools. Eclipse is discussed in Chapter 4.

Eclipse Framework see *Eclipse*.

IDE Integrated Desktop Environment. An example is *Eclipse*.

Greedy option (ANTLR) An ANTLR concept. Tells ANTLR to ignore non-determinism warnings in a production rule and to apply a specific production instead.

JAXB Java Architecture for XML Binding. Provides a convenient way to bind an XML schema to a representation in Java code. It can automatically generate import and export facilities between an XML document satisfying an XML schema and the corresponding Java representation.

Java Generics Java generics is a notation of generic types that was introduced from Java 1.5.0. Using generics it is possible to write type safe

APPENDIX A. TERM LIST AND ABBREVIATIONS

generic classes and functions - thereby finding more errors at compile-time. Many of Java's own class libraries has been rewritten to make use of generics.

JavaCC Java Compiler Compiler. Parser generator capable of generating k-lookahead recursive decent parsers.

JDOM Java representation of an *XML document*. JDOM provides a way to represent XML documents for easy and efficient reading, manipulation, and writing.

JJTree A preprocessor for *JavaCC* that can insert parse tree building actions at various places in a JavaCC source. The output of JJTree is run through JavaCC to create a parser.

JTB Java tree builder. A tool similar to *JJTree*.

Kernel Is the basic functionality of the *Overture Tool Set*. The kernel is discussed in Chapter 3.

lookahead The number of following tokens a parser is able to investigate when choosing between alternatives in a production rule.

OML Overture Modelling language. A development of *VDM++*.

OML plain text specification A specification written in OML – simply an OML file.

Overture Name of the project developing the *Overture Tool Set*

Overture Tool Set An open source tool set for OML. This M.Sc. thesis project has created the kernel for the Overture Tool Set.

Refactoring of AST The concept of restructuring the structure and the content of an AST automatically.

SAX Simple API for XML. An event-based API, on the other hand, reports parsing events (such as the start and end of elements) directly to the application through callbacks, and does not usually build an internal tree. The application implements handlers to deal with the different events, much like handling events in a graphical user interface. SAX is the best known example of such an API[16].

Syntactic Predicate (ANTLR) An ANTLR notion telling ANTLR to try to apply a production rule, and only actually apply it, if it can succeed.

UML Unified Modeling Language. An object oriented diagram notation used in software development.

VDM Vienna Development Method is a program development method based on formal specification using the VDM specification language *VDM-SL*.

VDM++ An object oriented version of *VDM-SL*.

VDM-SL The formal specification language of *VDM*, used in formal software specification.

XMI XML Metadata Interchange. A *XML* data format used to store UML diagrams.

XML eXtensible Markup Language. A general-purpose markup language for creating special-purpose markup languages. It is used to store and exchange data.

XML document A document in *XML* format.

XML Schema Can be used to specify the structure of *XML documents*. These can be automatically validated against the schema.

XPath XPath is a language for addressing parts of an XML document. It is designed to be used with XSL.

XSL, XSLT Extensible Stylesheet Language. A stylesheet for transforming an XML document to another XML document.

Appendix B

CD Contents Guide

This appendix is a guide to the cd handed in with this report. The cd contains all source code and programs needed for the project.

If you are reading this report without having the cd available, all source code will also be available at www.sourceforge.net/projects/overture.

Figure B.1 shows the contents of the cd-rom.

APPENDIX B. CD CONTENTS GUIDE

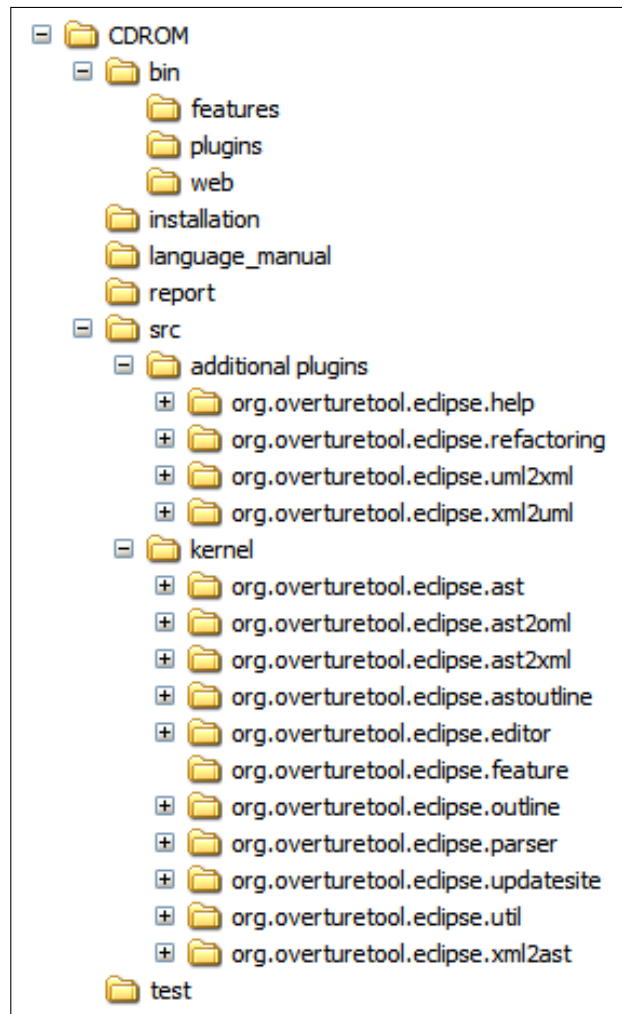


Figure B.1: Cd-rom contents

bin contains the update site from where the Overture Tool Set can be installed.

installation contains the installation guide. Furthermore it explains how to get started and how to get the entire source code.

language_manual contains the VDM++ language specification (BNF), [12].

report contains a pdf version of this report.

src contains the entire source code. It is organized in folders named as the different plug-in projects. The XML schema can be found in a subfolder of the `org.overturetool.eclipse.editor` plug-in.

test contains all tests files used to test the implementation.

Appendix C

Installation Guide

This Appendix provides an installation guide showing how to install the developed kernel for the Overture Tool Set. It will also describe how to install the full source code and how to import the provided test cases used to test the kernel.

C.1 Installation of Required Tools

Java and Eclipse should be installed before installing the Overture kernel or the Overture kernel source code.

1. Download 'JRE 5.0' (J2SE Runtime Environment) from Sun:
<http://java.sun.com/j2se/1.5.0/download.jsp> and install. In the case the user would like to look at the code, JDK 5.0 should be installed.
2. Download 'Eclipse SDK 3.1' from '<http://www.eclipse.org>' and extract the files in a folder called `eclipse_install`. Open Eclipse (run '`eclipse_install/eclipse/exlipse.exe`') and create a folder called 'workspace' which should be located in '`eclipse_install/workspace`'.

C.2 Installation of the Overture Kernel

Install instructions for the Overture kernel (Windows). Please read Appendix C.1 before installing.

1. Go to 'Help' > 'Software Updates' > 'Find and Install'; Select 'Search for new features to install' and press 'Next';

After 1. September 2005, it will be possible to install the newest version of the kernel and additional plug-ins from the internet. The web-site <http://www.overturetool.org> will provide an url from where it is possible to do an online installation.

2. If installing from the internet go to 5, and if installing from a cd-rom goto 6.
3. To install the newest version of the kernel from the internet, Press 'New Remote Site'; and type in an url of the Overture update site (see 3 for further instructions).
4. It is also possible to do a local installation using files from the cd-rom. To do this, create a folder called 'CD' which should be located in 'eclipse_install/CD', and copy the content of the cd-rom to this location. Press 'New Local Site' and select the path 'eclipse_install/CD/bin'.
5. After specifying a local or remote site, click the 'OK' button. Check the box in front of 'Overture' and uncheck 'Ignore features ...', then press 'Next';
6. Check the box in front of 'Overture' and 'Show the latest...' and uncheck 'Filter features...' and press 'Next';
7. Accept the license, press 'Next', then press 'Finish' and finally press 'Install All'.
8. Press 'Yes' to restart Eclipse and select the workspace located at 'eclipse_install/workspace'.

The Overture kernel is now installed.

C.3 Getting Started

We will now show you how to use the Overture kernel. In this example we will create an OML file, export it to XML and import it again.

1. Go to 'File' > 'New' > 'Project'.
2. Choose 'Simple' > 'Project' and give it a name.
3. Right click on the project and select 'New' > 'Other' > 'Overture Wizard' > 'Overture Editor file' and then press 'Next'.

4. Choose a project and give the file a name. Here is an example of an OML file:

```
class file1
types
public var1 = int;
var2 = (bool);
end file1
```

5. Export the file to XML by choosing 'Overture Menu' > 'Export to XML' and press 'Finish'.
6. Import an XML file to an OML file by choosing 'Overture Menu' > 'Import from XML'.
7. The imported XML file is validated against a XML schema, and errors will be reported to the user.

C.4 Installation of Source Code from a Kernel Release

After installing the Overture kernel, the user can have a look at the code used in this release. Please read Appendix C.1 before installing.

1. Go to 'File' > 'Import' > 'External plug-ins and Fragments'.
2. Uncheck the box in front of 'The target platform...'.
3. Press 'Browse' and select eclipse_install/eclipse/plugins and click 'Ok'.
4. Choose 'Select from all plugins...' and choose 'Projects with source folders' and press 'Next'.
5. Now highlight (choose) all plug-ins starting with the name 'org.overturetool.eclipse...' and press 'Add' and press 'Finish'.
6. In order to use JDK 5.0 go to: 'window' -> 'preferences' -> 'java' -> 'compiler' and chose compiler compliance level 5.0.

The Overture kernel source code for the actual release is now installed.

C.5 Installation of the Full Source Code

It is also possible to install the full source code from the cd-rom.

After 1. September 2005, the entire sources will be available for download from a SourceForge CVS server. This is the easiest way to work with the

source code. The web-site <http://www.overturetool.org> will contain a description for how to do this. Please read Appendix C.1 before installing.

1. If the content of the cd-rom is not placed in a folder called 'CD' located at 'eclipse_install/CD' do the following: create a folder called 'CD' which should be located in 'eclipse_install/CD', and copy the content of the cd-rom to this location.
2. Go to 'File' > 'Import' > 'Existing Projects into Workspace'.
3. Select the path 'eclipse_install/CD/src/kernel' and click 'Finish'.
4. Do the same using the path 'eclipse_install/CD/src/additional plug-ins'.
5. Now all the source code is installed. It is possible to run the Overture kernel in the developer mode and here choose which plug-in to include when running the program. This feature is used when testing different parts for the kernel.
6. In order to use JDK 5.0 go to: 'window' -> 'preferences' -> 'java' -> 'compiler' and choose compiler compliance level 5.0.

The source code for the kernel and additional plug-ins is now installed.

C.6 How to Access the Provided Test Cases

We will now show you how to access the test cases used in this project. It is only possible to do a full test of the kernel and the additional plug-ins from the Eclipse development environment. Furthermore, the full source code should be installed in order to do this.

1. In the Overture editor create a project called 'testproject'.
2. Go to 'File' > 'Import' > 'File system'.
3. Find the test cases at 'eclipse_install/CD/test' and choose 'testproject' as the input folder.

The test cases are now ready for testing.

Appendix D

Project Description

This chapter contains the official title and the project description text of the M.Sc. Thesis Project. The original text is in Danish. We bring an English translation as well, as the thesis is written in English.

D.1 Danish

Titel: Udvikling af et Overture/VDM++ værktøjssæt til Eclipse

Beskrivelse: Overture er et projekt, der stiler mod udvikling af et omfattende værktøjssæt til VDM++ under Eclipse platformen. Projektet skal bygges op om en kerne, der kan opbygge et abstrakt syntax træ ud fra en VDM++ specifikation, kan omforme træet til XML, kan gendanne træet fra XML og som tilbyder en Eclipse baseret editor til VDM++ specifikationer. Kernen skal opbygges, så forskellige værktøjer/plugins kan arbejde direkte på dette syntax træ. Det er hensigten at kommende plugins skal kunne udvikles uafhængigt af hinanden.

I dette projekt undersøges og fastlægges hvad der skal indgå i kernen. Det skal undersøges hvilke teknologier, der hensigtsmæssigt kan understøtte udviklingen af kernen. Kernen skal herefter implementeres som plugin til Eclipse under hensyn til ovenstående udviklingsstrategi.

D.2 English

Title: Development of an Overture/VDM++ Tool Set for Eclipse

Description: Overture is a project attending to develop a comprehensive tool set for VDM++ under the Eclipse platform. The project should be based on a kernel able to construct an abstract syntax tree from a VDM++ specification, to convert this tree to XML, to reconstruct the tree from XML, and to provide an Eclipse based editor for VDM++ specifications. The kernel

APPENDIX D. PROJECT DESCRIPTION

should be designed so that different tools/plugins can operate directly on the abstract syntax tree. It is the intention that future plugins can be developed independently.

This project investigates and determines what the kernel should consist of. Furthermore, it explores which technologies that are suitable for supporting the development of the kernel. Afterwards, the kernel is implemented as a plugin for Eclipse taking the above mentioned development strategies into account.

Appendix E

Overture Workshop in Newcastle Upon Tyne

E.1 Attendance in Workshop

The Overture project held a workshop in at the University of Newcastle Upon Tyne at July 18.th 2005. The Overture core group had invited us to participate and give a presentation of this M.Sc. thesis project. FME¹ agreed to sponsor one of us to participate in the workshop.

E.2 Contribution to Technical Report

We have written a contribution to a technical report, that will soon be published. Our contribution is shown below.

¹Formal Methods Europe, www.fmeurope.org

Designing a Flexible Kernel Providing VDM++ Support for Eclipse

Jacob Porsborg Nielsen and Jens Kielsgaard Hansen

Informatics and Mathematical Modelling, Technical University of Denmark

Abstract. This paper describes the development of an Eclipse [2] based tool set supporting the VDM++ language. It outlines how the Eclipse framework is used to provide a flexible and easily extendible kernel. The basic functionality of the kernel is described as well as how the design is prepared for further development. It is a central point that new functionality can be added without modifying the kernel implementation.

1 Introduction

The project described in this paper is part of the Overture Project[12]. Overture is an open source project aiming at developing tools for VDM++.

Our project aims to provide the kernel for the Overture Tool Set. The project is carried out as a M.Sc. Thesis Project [1] at Technical University of Denmark.

Other tools exist that support VDM++ development [11]. The motivation for the Overture Project is, however, to create a new open source tool, which can both be used as a development tool and for research purposes.

This paper will first introduce the functionality of the kernel. Then it summarizes our analysis of suitable tools and techniques for the kernel. The advantages of using Eclipse as a framework is presented followed by an overview of the chosen design. Some ideas for future development based on our project are then described. Finally we conclude the achievements of the project.

2 Functionality

This section will outline the functionality of the kernel.

As shown in Figure 1 the kernel provides the following facilities:

- Providing an Eclipse based editor.
- Parsing a VDM++ plain text specification to an Abstract Syntax Tree (AST). The AST supports use of Visitor Design Pattern [3] in order to make it easy for plug-ins to operate on the AST.
- Converting an AST to XML
- Converting XML to AST
- Pretty print from AST to VDM++ specification in plain text in the Editor.
- The kernel is implemented as plug-ins for Eclipse.
- It provides extension points so that new functionality can be added as Eclipse plug-ins.

Figure E.1: Overture Workshop Report, Page 1

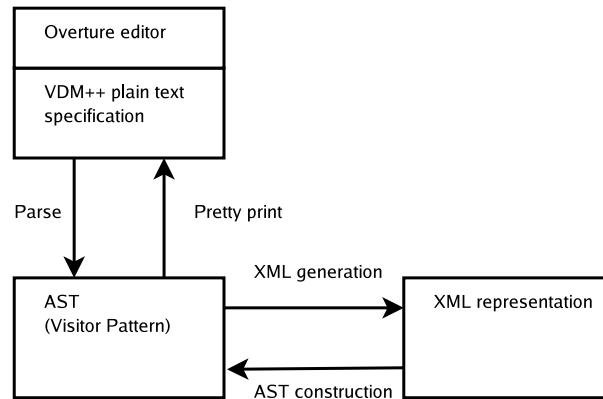


Fig. 1. Insert caption

3 Analysis

This chapter describes the choice of tools, techniques, and principles used in the kernel.

Requirements for the AST Classes

- Different AST classes must be created to represent the various language structures. If the AST classes are built with inheritance and the concept of abstract classes, common properties only need to be defined once.
- If there is an interface for each AST class, we can protect the tree from being modified by visitors. If a visitor needs to modify an AST, then this is still possible if it references the classes directly.
- The children of an AST class should be strongly typed. As there are different classes to represent different language structures, it will be easy to specify a type for each child, - possibly the type of one of its super classes. This prevents errors when building trees and gives additional possibilities for methods operating on AST classes.
- The non abstract AST classes should have an accept method for different visitors. By enabling use of visitor design pattern, new functionality can be implemented without changing the AST structure.
- Visitor interfaces must be provided for some different types of visitors. Providing visitor interfaces handling arguments of generic types enables a very wide range of visitors to use the AST structure.
- The names used in the AST classes must be meaningful, as other developers should easily be able to use the AST classes in their plug-ins.

Figure E.2: Overture Workshop Report, Page 2

Creation of AST Classes Because of the large number of AST classes and corresponding interfaces, we have tested a selected tools for auto generation of these [1]. As none of them could fulfill all the requirements described in Section 3 we chose to write the AST classes by hand. This means writing about 350 classes and 350 interfaces.

Creation of the Parser After examining the EBNF specification for the VDM++ language[6] it was decided that a top down parser is sufficient for parsing the language. The approach for AST and parser construction is described in [4]. The parser generation tool called ANTLR [8] was chosen because it has good error handling, good documentation and a wide community.

Creation of the Pretty Print Visitor Pretty printing can be done by traversing an AST and continuously building a string. Positions and comments are stored in the AST and in the XML instance documents.

Working with XML The tool called JDOM [9] was chosen when working with XML. For importing an XML document JDOM can read an XML document, validate it according to an XML schema and build its own tree representation of the XML document. This can afterwards be converted to the AST representation. For exporting to XML, a visitor is used to traverse the AST and continuously build a corresponding JDOM tree. This can afterwards be exported to a formatted XML instance document.

4 Eclipse

Basically, Eclipse [2] is nothing but a framework intended to be extended by plug-ins. Though Eclipse is distributed with advanced support for programming in Java, the main purpose of Eclipse is to serve as a basic framework for tool plug-ins. In fact, all Java supporting tools in Eclipse, are ordinary plug-ins themselves. Plug-ins are based on extension points and extending those. If a plug-in provides an extension point, other plug-ins can interact with it by extending this extension point. Similarly, plug-ins can interact with the Eclipse framework by extending extension points provided by Eclipse. If a plug-in needs classes from another plug-in in order to work properly, one can specify a dependency.

Eclipse was chosen as the platform for the project, as it is a stable framework with extensive possibilities for adding additional plug-ins, but other environments exists. Some central features making Eclipse a good choice for the project are:

- Eclipse is a wide general framework, that can serve as a platform for many kinds of tools
- Eclipse is well documented and up-to-date supporting e.g. development with Java Generics
- The plug-in concept of Eclipse is powerful for the Overture project
- The framework provides many advanced features to Eclipse based editors

Figure E.3: Overture Workshop Report, Page 3

5 Design

This section describes the design of the different parts of the kernel. Because of the size of the kernel the focus is on showing principles in order to give an overview.

5.1 Design of the AST

This section gives an overview of the design principles for AST construction. Here is an example of how the AST is implemented:

Figure 2 shows that an AST class can inherit from one of the two abstract classes `InternalASTNode` or `InternalASTNodeWithComments`, depending on whether it can have associated comments or not. It was decided to store position information for each AST node. This information should e.g. be used for pretty printing or for making it easier to find the right line in the Overture editor using an outline view when working with large VDM++ specifications. The AST classes have set and get methods for each of their children and accept methods for visitors. They implement corresponding interfaces [5] showing only the get methods to the user. By having these interfaces the user can operate on the interfaces without being aware of the implemented AST classes. All AST classes inherits somehow from `InternalASTNode` or `InternalASTNodeWithComments`, either by inheriting from them directly or by inheriting from an abstract class that inherits from them. The abstract classes represents concepts of VDM++, like expressions, functions, operations, etc.

5.2 Design of the Parser

ANTLR generates a lexer and a parser unit. Furthermore, a filtering layer is placed between these two layers in order to handle comments. ANTLR has support for this as well. Precedence rules can be specified in the ANTLR grammar and grouping conventions are implemented when building the AST in the action code. With this solution it is possible to regenerate the parser without having to modify any files manually afterwards.

5.3 Design of the XML Schema

An XML schema is used to verify the correctness of an XML document. In the overture kernel a validation is performed every time an XML document is imported or exported. The schema is created using a tool called XML spy [10]. An XML instance document represents an AST and we have therefore chosen to follow the same structure when creating the XML schema.

5.4 Designing the Visitors

The different visitors should all implement an interface defining which nodes the visitor should visit. Furthermore, there should be a general visitor visiting all

Figure E.4. Overture Workshop Report, Page 4

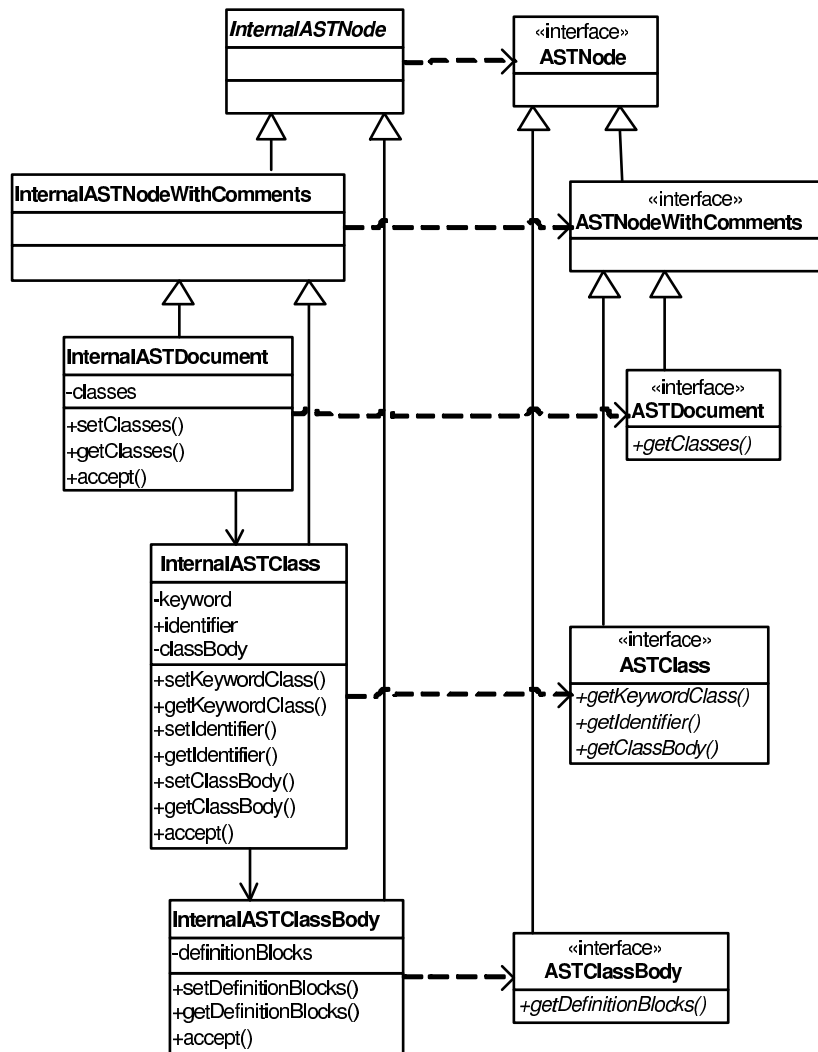


Fig. 2. Design of the AST classes

Figure E.5: Overture Workshop Report, Page 5

nodes. All visitors can then extend this general visitor. Section 3 briefly explained how JDOM can be used to export to XML using a visitor. The technique for pretty printing an AST to a VDM++ plain text specification is to use the position information stored in the AST classes for the identifier and keyword nodes to build a correctly formatted string.

5.5 Design of the XML to AST Parser

The XML to AST parser works similar to the parser described in Section 5.2. The difference is that it builds the AST from a JDOM document instead of getting the tokens from the lexer. No error handling is needed because the XML instance document is by definition well formed.

5.6 Design of the Editor

The Overture editor uses a large amount of Eclipse's functionality by extending the extension points defined by Eclipse. The implementation is therefore well integrated with the eclipse framework. It provides coloring of keywords as well as an outline view for giving overviews of large VDM++ specifications. Using the Overture menu provided it is possible to import and export XML.

5.7 Design of Extension Possibilities

The Overture editor is implemented in a plug-in providing a number of extension points in order to make the kernel as flexible as possible. The rest of the functionality of the kernel is implemented in other plug-ins extending the provided extension points. This functionality works so that the Overture editor is unaware of any parser or visitor implementation. Using this design other developers can replace our implementation of the parsers and visitors without having to modify any of the plug-ins. By implementing an interface for a type checker, anyone can extend the kernel with this facility.

6 Future work

There is a wide range of additional functionality that can be added to the kernel. This includes e.g. a type checker, a Java code generator, and import and export facilities to UML. These can all be added using the extension principles provided by the kernel.

For future development based on our kernel, we refer to [1]. A chapter of the master thesis report is dedicated to present how to extend the language or how to add additional plug-ins with new functionality. The report also documents the analysis, design, implementation, and test issues of the kernel development.

Figure E.6: Overture Workshop Report, Page 6

7 Conclusion

This paper has presented an overview of the functionality and the design of the kernel for the Overture tool set. The AST classes, parsers and the visitors have been implemented by following the theory [4] and selected guidelines [5]. The integration with Eclipse is done using plug-ins and the kernel has been constructed to be as flexible as possible.

8 Acknowledgements

We would like to thank our supervisors Anne E. Haxthausen, Hans Bruun and Peter Gorm Larsen for their enthusiasm for our project. We also thank the Overture Core Group for their input and good discussions.

References

1. M.Sc. Thesis Project, Development of an Overture/VDM++ Tool Set for Eclipse, by Jacob Porsborg Nielsen and Jens Kielsgaard Hansen, Informatics and Mathematical Modelling, Technical University of Denmark. Supervisors: Hans Bruun and Anne E. Haxthausen, External Supervisor: Peter Gorm Larsen. The project will be available ultimo August 2005 at http://www.imm.dtu.dk/English/Research/Search_publications.aspx
2. DAnjou et.al., Eclipse, The Java Developer's Guide to Eclipse, Addison-Wesley, second edition, 2005.
3. Erich Gamma et.al., Design Patterns, Addison-Wesley 1995.
4. David A. Watt et.al., Programming Language Processors in Java, Pearson Education Limited, 2000.
5. P. Mukherjee. A blueprint for development of language tools. Memo produced as input to Overture, 2004.
6. VDMTools language definitions, The VDM++ language, EBNF grammar and description of the language, <http://www.vdmbook.com/langmanpp.a4.001.pdf>
7. Eclipse, The Eclipse framework home page <http://www.eclipse.org>
8. ANTLR, ANTLR parser generator, <http://www.antlr.org>
9. JDOM, An XML framework for Java, <http://www.jdom.org>
10. Altova xmlspy, Graphical tool for generation of XML schemas, www.xmlspy.com
11. VDMTools, Tool set supporting VDM++ development. Was prior called IFAD VDMTools, now called CSK VDMTools. <http://www.vdmbook.com/tools.php>
12. Overture, The Overture project, <http://www.overturetool.org>

Figure E.7: Overture Workshop Report, Page 7

Appendix F

Overview of Precedence and Grouping Conventions

Based on the descriptions of precedence in [12], we have assigned values of precedence to different expressions. These are shown below.

F.1 Precedence Overview – Expressions

Name of production	Precedence	Grouping
Expressions		
bracketed expression	70	
let expression	70	
def expression	70	
if expression	70	
cases expression	70	
undefined expression	70	
set enumeration	70	
name	70	
old name	70	
symbolic literal	70	
self expression	70	
threadid expression	70	
subsequence	50	left
apply	50	left
field select	50	left
function type instantiation	50	left
Unary Operators		
unary plus	46u	
unary minus	46u	
arithmetic abs	46u	
floor	46u	
not	25	
set cardinality	46u	
finite power set	46u	
distributed set union	41	
distributed set intersection	41	
sequence head	46u	
sequence tail	46u	
sequence length	46u	
sequence elements	46u	
sequence indices	46u	
distributed sequence concatenation	46u	
map domain	46u	
map range	46u	
distributed map merge	46u	

F.1. PRECEDENCE OVERVIEW – EXPRESSIONS

Name of production	Precedence	Grouping
Binary Operators		
arithmetic plus	41	left
arithmetic minus	41	left
arithmetic multiplication	42	left
arithmetic divide	42	left
arithmetic integer division	42	left
arithmetic rem	42	left
arithmetic mod	42	left
less than	31	no grouping
less than or equal	31	no grouping
greater than	31	no grouping
greater than or equal	31	no grouping
equal	31	no grouping
not equal	31	no grouping
or	23	right
and	24	right
imply	22	right
logical equivalence	21	right
in set	31	no grouping
not in set	31	no grouping
subset	31	no grouping
proper subset	31	no grouping
set union	41	left
set difference	41	left
set intersection	42	left
sequence concatenate	41	left
map or sequence modify	41	left
map merge	41	left
map domain restrict to	44	right
map domain restrict by	44	right
map range restrict to	45	left
map range restrict by	45	left
composition	61	right
iterate	62	right

F.2 Precedence Overview – Type Operators

Name of production	Precedence	Grouping
Type operators		
partial function types	1	right
total function types	1	right
union type	2	left
product type	3	no grouping
general map type	4	right
injective map type	4	right
set type	5	
seq type	5	
seq1 type	5	

Appendix G

Structured Decision Making

Structured decision making is a method of software engineering, that helps documenting the analysis of important choices made in a project. An objective of doing structured decision making, is that it helps documenting the background of important decisions. We have found it most relevant to use structured decision making in the process of choosing parser generation tool.

G.1 Choosing Parser Generation Method

Problem

Problem description	Which tool is best to generate the parser
Solution requirements and objectives	Which parser generation tool is best, given that we want it to build an AST based on inheritance. In addition, the parser needs to support or be extendible with error handling. The built parser should be able to handle comments, as we need to store these in the AST. If a top-down parser is chosen, it needs to support at least fixed n lookahead, in order for the parser to support the needed language. The generated code must be Java code.

Evaluation Criteria

Selected Criteria	Description	Rank*
Structure	General structure of the specification and of the generated parser. The parser must build AST where the AST classes are based on inheritance. This is a requirement for the project.	5
Error handling	The parser must be extendible to find several syntax error in the parse text. It is preferable that the parser generation tool offers error handling capabilities in the generated parser.	4
Comment support	The parser needs to store comments in the AST when building it. It is preferable, that the parser generation tool offers a proper comment handling mechanism.	3
*Rank description: 1-5, where 5 is very important and 1 is optional		

Rejected Criteria	Reason for Rejection
Auto generation of AST classes	It is flexible if the parser generation tool offers to generate AST classes. Writing AST classes by hand will however give easier readable code, making it easier to operate on when writing plug-ins that operates on the AST.

Possible Solutions

Possible Solution ¹	Description
ANTLR	Using ANTLR ² for parser generation
JavaCC	Using JavaCC ³ for parser generation

¹SableCC is not considered as a possible solution, as earlier work with SableCC in the Overture project has shown severe scalability problems. [9]

²ANTLR version 2.7.5 [18]

³JavaCC version 3.2 [17]. A new version of JavaCC (4.0 beta) was made available later in the project.

Evaluation Methods

Evaluation Method	Description
Tests on examples	Test in practice on examples and small subsets of the OML language. Test tool when used with possible additional tools.
Inspection of generated code	Investigate generated code to learn more about the structure of the generated code
Research of documentation	What documentation is available?

Evaluation Results

Solution:	ANTLR		
Methods vs. Criteria	Structure	Error handling	Comment support
Tests on examples	Clear structure of specifications. AST building action code can be written clear. ANTLR can't generate AST classes automatically in requested manner. ANTLR can construct parser's with n-lookahead.	Built in error handling mechanism well functioning.	Filter token streams into two separate streams. Well structured solution.
Inspection of generated code	Generated code is clear and easy readable.	Generated error handling code seams reasonable.	Elegant code generated for support of comments.
Research of documentation	Features of ANTLR is well documented.	Error handling well described.	There is an article of how to handle comments as needed in the project.
Uncertainty:	none		

APPENDIX G. STRUCTURED DECISION MAKING

Solution:	JavaCC		
Methods vs. Criteria	Structure	Error handling	Comment support
Tests on examples	Using own AST classes is possible as with ANTLR. Generation of AST classes is not possible. JavaCC can construct parser's with n-lookahead.	Error support exists.	Special comment structure suited for the purpose.
Inspection of generated code	Generated code readable.	Reasonable error handling code.	Nice implementation.
Research of documentation	Limited documentation.	Some documentation.	Documentation is fairly weak, but exists.
Uncertainty:	none		

G.1. CHOOSING PARSER GENERATION METHOD

Additional Solution, Criteria, or Method	Description	Reason for addition
Criteria: Available documentation	Checking that there is sufficient documentation of the tools suggested as solutions: ANTLR seems to have a large amount of good and accurate documentation with many examples. The official documentation for JavaCC was not up-to-date when decision of parser tool was taken. JavaCC has only few examples.	Criteria added, as it was found, that documentation of the parser generation tool is important both when developing the parser, but also if other people are to maintain the parser afterwards.
Criteria: Active development of the tool?	Considering if there is ongoing development of the tools: ANTLR is still developed and has regular releases of new versions with additional features. The new features are well documented and seems to be backward compatible with specifications written for older versions of ANTLR. There are development of JavaCC, but not as active as for ANTLR.	Criteria added, as active development and new releases of the parser generation tool may enable improving the performance or functionality of the parser in the future.
Criteria: Licence issues?	Investigate if the licence of the tool limits or restricts the use of the generated parser: ANTLR has a very open licence, where both the source code of ANTLR and the generated code are available without license restrictions. JavaCC has a licence by Sun, that should allow use of the generated parser in the overture project, if certain copyright notices are shown to the end user.	Criteria added, as it is important that we are actually allowed to distribute the generated parser as part of the kernel.
Criteria: Dynamic lookahead capabilities	Investigate if the parser generation tool has facilities to use dynamic lookahead on selected productions.	Criteria added, as parser generators that generates fixed lookahead parsers may not be able to generate a parser that supports all of the OML language.

APPENDIX G. STRUCTURED DECISION MAKING

Recommended Solution	ANTLR used for parser generation. Parser constructing AST based on manually written AST classes.
Reason for Recommendation	The selected criteria in mind, ANTLR fulfills all criteria. JavaCC supports most of the criteria, but has weaker error handling than ANTLR. ANTLR has no problems, taking the additional criteria in mind. JavaCC might have problems as the documentation seems to be limited. ANTLR has syntactic predicates, which effectively enables use of dynamic lookahead on selected productions. We have found no similar facility for JavaCC

Selected Solution

Selected Solution	ANTLR
Selection Rationale	ANTLR fulfills all criteria, has especially good documentation, and is used by a wide community. Writing the AST classes manually in combination with an ANTLR generated parser will be a good base for the kernel. As ANTLR provides good error handling mechanisms as well as comment handling facilities, there is no need to modify the generated code. This means that future modifications of the parser can be done, altering the ANTLR grammar file and making ANTLR rebuild the parser. The syntactic predicates of ANTLR might be useful to make the parser support the entire OML language.
Associated Risks	none

Rejected Solution	Reason for Rejection
JavaCC	There are no criteria where JavaCC has advantages over ANTLR. The available documentation seems to be insufficient guidance if JavaCC is chosen. We are not sure that a fixed lookahead parser generated by JavaCC will be able to recognize the entire OML language.

Appendix H

Modifying the OML language

In this project the entire OML language has been implemented. However, one of the goals for the Overture project was that it should be possible to experiment with different modifications of the language. This chapter will therefore describe how to do such a modification.

H.1 Adding a operator to the OML language

This section will illustrate what parts of the kernel needs to be altered when a new operator is added. The kernel is build in a modular manner so that the different parts of the kernel are independent.

A new binary operator can be added by following these steps:

- Create the AST class and the corresponding interface for the new operator. These files should be placed in the `org.overturetool.eclipse.ast` plug-in.
- Modify the ANTLR grammar file to include the new operator and regenerate the parser. This file is placed in the `org.overturetool.eclipse.parser` plug-in.
- Add a visit method for the new operator in the different visitors. These files are placed in the plug-ins containing the visitors.
- Modify the XML schema and the XML to AST converter to take the new operator into account. The XML schema is placed in the `org.overturetool.eclipse.editor` plug-in and the XML to AST parser is placed in the `org.overturetool.eclipse.xml2ast` plug-in.

Different dialects of the VDM language (or related languages like RSL) can all be supported by the kernel. This can be done by creating new parser and visitor plug-ins for each of the dialects. At the current implementation it is up to the user to install the correct plug-ins. The plug-in loader class is

APPENDIX H. MODIFYING THE OML LANGUAGE

not designed to handle if there is several parsers installed at the same time, but it is possible to add support for this.

Appendix I

Source Code - Selected Samples

This appendix shows selected code examples. In the project, more than 50.000 lines of code has been written. We have selected some files that represents the applied theory and techniques, and they are shown in this appendix. The entire code is available on the cd-rom as explained in Appendix B.

I.1 ANTLR Grammar File

Listing I.1 shows the entire ANTLR grammar specification.

Listing I.1: `overture.g` located in `org.overturetool.eclipse.parser`

```

1  header {
2  package org.overturetool.eclipse.parser;
3
4  import org.overturetool.eclipse.internal.ast.*;
5  import java.util.List;
6  import java.util.ArrayList;
7  }
8
9
10 class OvertureInternalParser extends Parser;
11 options {
12   k = 2; // one token lookahead
13   exportVocab=Overture; // Call its vocabulary "Overture"
14   codeGenMakeSwitchThreshold = 2; // Some optimizations
15   codeGenBitsetTestThreshold = 3;
16   defaultErrorHandler = true; // Generate parser error handlers
17   buildAST = false;
18 }
19
20 // KEYWORDS
21 tokens {
22   A = "a" ;
23   ABS = "abs" ;
24   ALL = "all" ;
25   ALLSUPER = "allsuper" ;
26   ALWAYS = "always" ;
27   AND = "and" ;
28   ANSWER = "answer" ;
29   ASSUMPTION = "assumption" ;
30   ATOMIC = "atomic" ;
31   BE = "be" ;
32   BOOL = "bool" ;
33   BY = "by" ;
34   CARD = "card" ;
35   CASES = "cases" ;
36   CHAR = "char" ;
37   CLASS = "class" ;
38   COMP = "comp" ;
39   COMPOSE = "compose" ;
40   CONC = "conc" ;
41   DCL = "dcl" ;
42   DEF = "def" ;
43   DEL = "del" ;
44   DINTER = "dinter" ;
45   DIV = "div" ;
46   DO = "do" ;
47   DOM = "dom" ;
48   DUNION = "dunion" ;
49   EFFECT = "effect" ;
50   ELEMS = "elems" ;
51   ELSE = "else" ;
52   ELSEIF = "elseif" ;
53   END = "end" ;
54   ERROR = "error" ;
55   ERRS = "errs" ;
56   EXISTS = "exists" ;
57   EXISTS1 = "exists1" ;
58   EXIT = "exit" ;
59   EXT = "ext" ;
60   FALSE = "false" ;
61   FLOOR = "floor" ;
62   FOR = "for" ;
63   FORALL = "forall" ;
64   FROM = "from" ;
65   FUNCTIONS = "functions" ;
66   GENERAL = "general" ;
67   HD = "hd" ;
68   IF = "if" ;
69   IN = "in" ;
70   INDS = "inds" ;
71   INIT = "init" ;
72   INMAP = "inmap" ;
73   INPUT = "input" ;
74   INSTANCE = "instance" ;
75   INT = "int" ;
76   INTER = "inter" ;

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

77 INV = "inv" ;
78 INVERSE = "inverse" ;
79 IOTA = "iota" ;
80 IS = "is" ;
81 ISU = "is_" ;
82 ISUBOOL = "is_bool" ;
83 ISUCHAR = "is_char" ;
84 ISUINT = "is_int" ;
85 ISUNAT = "is_nat" ;
86 ISUNAT1 = "is_nat1" ;
87 ISURAT = "is_rat" ;
88 ISUREAL = "is_real" ;
89 ISUTOKEN = "is_token" ;
90 ISOFBASECLASS="isofbaseclass";
91 ISOFCLASS = "isofclass";
92 LAMBDA = "lambda" ;
93 LEN = "len" ;
94 LET = "let" ;
95 MAKE = "mk" ;
96 MAP = "map" ;
97 MERGE = "merge" ;
98 MOD = "mod" ;
99 MU = "mu" ;
100 MUNION = "munion" ;
101 MUTEX = "mutex" ;
102 NAT = "nat" ;
103 NAT1 = "nat1" ;
104 NEW = "new" ;
105 NIL = "nil" ;
106 NOT = "not" ;
107 OF = "of" ;
108 OPERATIONS = "operations";
109 OR = "or" ;
110 OTHERS = "others" ;
111 PER = "per" ;
112 PERIODIC = "periodic";
113 POST = "post" ;
114 POWER = "power" ;
115 PRE = "pre" ;
116 PREF = "pref" ;
117 PRIVATE = "private" ;
118 PROTECTED = "protected" ;
119 PSUBSET = "psubset" ;
120 PUBLIC = "public" ;
121 QSYNC = "qsync" ;
122 RAT = "rat" ;
123 RD = "rd" ;
124 REAL = "real" ;
125 REM = "rem" ;
126 RESPONSIBILITY="responsibility";
127 RETURN = "return" ;
128 REVERSE = "reverse" ;
129 RNG = "rng" ;
130 SAMEBASECLASS="samebaseclass";
131 SAMECLASS = "sameclass";
132 SEL = "sel" ;
133 SELF = "self" ;
134 SEQ = "seq" ;
135 SEQ1 = "seq1" ;
136 SET = "set" ;
137 SKIP = "skip" ;
138 SPECIFIED = "specified";
139 ST = "st" ;
140 START = "start" ;
141 STARTLIST = "startlist";
142 SUBCLASS = "subclass";
143 SUBSET = "subset" ;
144 SUBTRACE = "subtrace";
145 SYNC = "sync" ;
146 SYNONYM = "synonym" ;
147 T = "t" ;
148 THEN = "then" ;
149 THREAD = "thread" ;
150 THREADID = "threadid";
151 TIXE = "tixe" ;
152 TL = "tl" ;
153 TO = "to" ;
154 TOKEN = "token" ;
155 TRAP = "trap" ;
156 TRUE = "true" ;
157 TYPES = "types" ;
158 UNDEFINED = "undefined";
159 UNION = "union" ;
160 USING = "using" ;

```

I.1. ANTLR GRAMMAR FILE

```

161 VALUES = "values" ;
162 VARIABLES = "variables";
163 W = "w" ;
164 WHILE = "while" ;
165 WITH = "with" ;
166 WR = "wr" ;
167 YET = "yet" ;
168 RESULT = "RESULT" ;
169 }
170
171
172 // overtureDocument: This is the ASTERIX
173 // rule for this parser
174 // document = class, { class };
175 overtureDocument returns [InternalASTDocument internalASTDocument]
176 {
177     internalASTDocument = null;
178     List<InternalASTClass> internalASTClassList = new ArrayList<InternalASTClass>();
179     InternalASTClass internalASTClass = null;
180 }
181 :
182
183     internalASTClass = overtureClass
184     {
185         internalASTClassList.add(internalASTClass);
186     }
187
188     (
189         internalASTClass = overtureClass
190         {
191             internalASTClassList.add(internalASTClass);
192         }
193     )*
194 EOF!
195 {
196     internalASTDocument = new InternalASTDocument(internalASTClassList);
197 }
198 ;
199
200 // class = 'class', identifier, [inheritance clause],
201 //     [class body],
202 //     'end', identifier;
203 overtureClass returns [InternalASTClass internalASTClass]
204 {
205     internalASTClass = null;
206     InternalASTClassBody internalASTClassBody = null;
207     InternalASTInheritanceClause internalASTInheritanceClause = null;
208 }
209 : cl:CLASS id:IDENTIFIER
210 (
211     internalASTInheritanceClause = overtureInheritanceClause
212 )?
213 (
214     internalASTClassBody = overtureClassBody
215 )?
216
217 en:END id2:IDENTIFIER
218 {
219     InternalASTKeyword keywordClass = new InternalASTKeyword(cl,"Class");
220     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
221     InternalASTKeyword keywordEnd = new InternalASTKeyword(en,"End");
222     InternalASTIdentifier internalASTIdentifierEnd = new InternalASTIdentifier(id2);
223     internalASTClass = new InternalASTClass(keywordClass, internalASTIdentifier,
224         internalASTInheritanceClause, internalASTClassBody, keywordEnd, internalASTIdentifierEnd
225     );
226 }
227 ;
228
229 // inheritance clause = 'is subclass of', identifier, {identifier};
230 overtureInheritanceClause returns [InternalASTInheritanceClause internalASTInheritanceClause]
231 {
232     InternalASTKeyword keywordIs = null;
233     InternalASTKeyword keywordSubclass = null;
234     InternalASTKeyword keywordOf = null;
235     InternalASTIdentifier internalASTIdentifier1 = null;
236     internalASTInheritanceClause = null;
237     List<InternalASTIdentifier> inheritanceIdentifierList = new ArrayList<InternalASTIdentifier>();
238 }
239 :
240     is:IS
241     su:SUBCLASS
242     of:OF
243     id1:IDENTIFIER

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

243 {
244     keywordIs = new InternalASTKeyword(is, "Is");
245     keywordSubclass = new InternalASTKeyword(su, "Subclass");
246     keywordOf = new InternalASTKeyword(of, "Of");
247     internalASTIdentifier1 = new InternalASTIdentifier(id1);
248     inheritanceIdentifierList.add(internalASTIdentifier1);
249 }
250 (
251     id:IDENTIFIER
252     {
253         InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
254         inheritanceIdentifierList.add(internalASTIdentifier);
255     }
256 )*
257 {
258     internalASTInheritanceClause = new InternalASTInheritanceClause(keywordIs, keywordSubclass,
259         keywordOf, inheritanceIdentifierList);
260 }
261 ;
262
263 // class body = definition block, {definition block} ;
264 overtureClassBody returns [InternalASTClassBody internalASTClassBody]
265 {
266     internalASTClassBody = null;
267     List<InternalASTDefinitionBlock> internalASTDefinitionBlockList = new ArrayList<
268         InternalASTDefinitionBlock>();
269     InternalASTDefinitionBlock internalASTDefinitionBlock = null;
270 }
271 :
272     internalASTDefinitionBlock = overtureDefinitionBlock
273     {
274         internalASTDefinitionBlockList.add(internalASTDefinitionBlock);
275     }
276     (
277         internalASTDefinitionBlock = overtureDefinitionBlock
278         {
279             internalASTDefinitionBlockList.add(internalASTDefinitionBlock);
280         }
281     )*
282     {
283         internalASTClassBody = new InternalASTClassBody(internalASTDefinitionBlockList);
284     }
285 ;
286
287 // definition block = type definitions
288 //     | value definitions
289 //     | function definitions
290 //     | operation definitions;
291 //     | instance variable definitions;
292 //     | synchronization definitions;
293 //     | thread definitions;
294 overtureDefinitionBlock returns [InternalASTDefinitionBlock internalASTDefinitionBlock]
295 {
296     //ASTDefinitionBlock is abstract...
297     internalASTDefinitionBlock = null;
298 }
299 :
300     (
301         internalASTDefinitionBlock = overtureTypeDefinitions
302         | internalASTDefinitionBlock = overtureValueDefinitions
303         | internalASTDefinitionBlock = overtureFunctionDefinitions
304         | internalASTDefinitionBlock = overtureOperationDefinitions
305         | internalASTDefinitionBlock = overtureInstanceVariableDefinitions
306         | internalASTDefinitionBlock = overtureSynchronizationDefinitions
307         | internalASTDefinitionBlock = overtureThreadDefinitions
308     )
309 ;
310
311
312 // type definitions = 'types',[access type definition,
313 //     {';', access type definition},[';']]
314 overtureTypeDefinitions returns [InternalASTTypeDefinitions internalASTTypeDefinitions]
315 {
316     internalASTTypeDefinitions = null;
317     InternalASTKeyword keywordTypes = null;
318     List<InternalASTTypeDefinitionsElement> typeDefinitionsElements = new ArrayList<
319         InternalASTTypeDefinitionsElement>();
320     InternalASTTypeDefinitionsElement currentElement = null;
321     InternalASTAccessTypeDefinition internalASTAccessTypeDefinition = null;
322 }
323 :
324     ty:TYPES

```


I.1. ANTLR GRAMMAR FILE

```

324     {
325     keywordTypes = new InternalASTKeyword(ty, "Types");
326     }
327
328     (
329     internalASTAccessTypeDefinition = overtureAccessTypeDefinition
330     {
331     currentElement = new InternalASTTypeDefinitionsElement(internalASTAccessTypeDefinition);
332     typeDefinitionsElements.add(currentElement);
333     }
334
335     (options {greedy=true};)
336     seNoN:SEMICOLON
337     {
338     InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN, "Semicolon");
339     currentElement.setKeywordSemicolon(semicolonNoN);
340     currentElement.setEndPositionFromNode(semicolonNoN);
341     }
342     internalASTAccessTypeDefinition = overtureAccessTypeDefinition
343     {
344     currentElement = new InternalASTTypeDefinitionsElement(internalASTAccessTypeDefinition);
345     typeDefinitionsElements.add(currentElement);
346     }
347     )*
348
349     (seLast:SEMICOLON
350     {
351     InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast, "Semicolon");
352     currentElement.setKeywordSemicolon(semicolonLast);
353     currentElement.setEndPositionFromNode(semicolonLast);
354     }
355     )?
356
357     )?
358     {
359     internalASTTypeDefinitions = new InternalASTTypeDefinitions(keywordTypes,
360     typeDefinitionsElements);
361     }
362     ;
363
364     // access type definition = [access], type definition;
365     overtureAccessTypeDefinition returns [InternalASTAccessTypeDefinition
366     internalASTAccessTypeDefinition]
367     {
368     internalASTAccessTypeDefinition = null;
369     InternalASTAccess internalASTAccess = null;
370     InternalASTTypeDefinition internalASTTypeDefinition = null;
371     }
372     :
373     (
374     internalASTAccess = overtureAccess
375     )?
376     internalASTTypeDefinition = overtureTypeDefinition
377     {
378     internalASTAccessTypeDefinition = new InternalASTAccessTypeDefinition(internalASTAccess,
379     internalASTTypeDefinition);
380     }
381     ;
382
383     // access = 'public',
384     //           | 'private',
385     //           | 'protected';
386     overtureAccess returns [InternalASTAccess internalASTAccess]
387     {
388     //ASTAccess is abstract...
389     internalASTAccess = null;
390     }
391     :
392     (
393     pub:PUBLIC
394     {
395     internalASTAccess = new InternalASTAccessPublic(new InternalASTKeyword(pub, "Public"));
396     }
397     |
398     pri:PRIVATE
399     {
400     internalASTAccess = new InternalASTAccessPrivate(new InternalASTKeyword(pri, "Private"));
401     }
402     |
403     pro:PROTECTED
404     {
405     internalASTAccess = new InternalASTAccessProtected(new InternalASTKeyword(pro, "Protected"));
406     }
407     )
408     ;
409
410     // type definition = identifier, '=', type, [invariant]

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

405 //      | identifier, '::', field list, [invariant];
406 overtureTypeDefinition returns [InternalASTTypeDefinition internalASTTypeDefinition]
407 {
408     internalASTTypeDefinition = null;
409 }
410 :
411 (
412     internalASTTypeDefinition = overtureTypeDefinitionType
413 |
414     internalASTTypeDefinition = overtureTypeDefinitionFieldList
415 )
416 ;
417
418 // Matches identifier, '=', type, [invariant]
419 overtureTypeDefinitionType returns [InternalASTTypeDefinitionType internalASTTypeDefinitionType]
420 {
421     internalASTTypeDefinitionType = null;
422     InternalASTType internalASTType = null;
423     InternalASTInvariant internalASTInvariant = null;
424 }
425 :
426     id:IDENTIFIER
427     eq:EQUALSIGN
428     internalASTType = overtureType
429     (
430         internalASTInvariant = overtureInvariant
431     )?
432     {
433         InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
434         InternalASTKeyword keywordEqualsign = new InternalASTKeyword(eq,"Equalsign");
435         internalASTTypeDefinitionType = new InternalASTTypeDefinitionType(internalASTIdentifier ,
436             keywordEqualsign, internalASTType, internalASTInvariant);
437     }
438 ;
439
440 // Matches identifier, '::', field list, [invariant]
441 overtureTypeDefinitionFieldList returns [InternalASTTypeDefinitionFieldList
442     internalASTTypeDefinitionFieldList]
443 {
444     internalASTTypeDefinitionFieldList = null;
445     InternalASTFieldList internalASTFieldList = null;
446     InternalASTInvariant internalASTInvariant = null;
447 }
448 :
449     id:IDENTIFIER
450     dc:DOUBLECOLON
451     internalASTFieldList = overtureFieldList
452     (
453         internalASTInvariant = overtureInvariant
454     )?
455     {
456         InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
457         InternalASTKeyword keywordDoubleColon = new InternalASTKeyword(dc,"DoubleColon");
458         internalASTTypeDefinitionFieldList = new InternalASTTypeDefinitionFieldList(
459             internalASTIdentifier, keywordDoubleColon, internalASTFieldList, internalASTInvariant);
460     }
461 ;
462
463 // type = bracketed type
464 // | basic type
465 // | quote type
466 // | composite type
467 // | union type
468 // | product type
469 // | optional type
470 // | set type
471 // | seq type
472 // | map type
473 // | partial function type
474 // | type name
475 // | type variable;
476 overtureType returns [InternalASTType internalASTType]
477 {
478     //ASTAccess is abstract...
479     internalASTType = null;
480 }
481 :
482     internalASTType = overtureTypePre1
483 ;
484
485 overtureTypePre6 returns [InternalASTType internalASTType]
486 {
487     internalASTType = null;
488 }

```

```

486 :
487 (
488 //Type type expressions - base cases
489   internalASTType = overtureBracketedType
490 | internalASTType = overtureBasicType
491 | internalASTType = overtureQuoteType
492 | internalASTType = overtureCompositeType
493 | internalASTType = overtureOptionalType
494 | internalASTType = overtureTypeName
495 | internalASTType = overtureTypeVariable
496 )
497 ;
498
499 overtureTypePre5 returns [InternalASTType internalASTType]
500 {
501 //Type operators with precedence 5 (highest precedence)
502   internalASTType = null;
503 }
504 :
505     internalASTType = overtureSetType
506 | internalASTType = overtureSeqType
507 | internalASTType = overtureTypePre6
508 ;
509
510 overtureTypePre4 returns [InternalASTType internalASTType]
511 {
512 //Type operators with precedence 4 (high, but not highest precedence)
513   internalASTType = null;
514 }
515 :
516     internalASTType = overtureMapType
517 | internalASTType = overtureTypePre5
518 ;
519
520 overtureTypePre3 returns [InternalASTType internalASTType]
521 {
522 //Type operators with precedence 3 (middle precedence)
523   internalASTType = null;
524 }
525 :
526     (overtureProductType)=> internalASTType = overtureProductType
527 | internalASTType = overtureTypePre4
528 ;
529
530 overtureTypePre2 returns [InternalASTType internalASTType]
531 {
532 //Type operators with precedence 2 (low, but not lowest precedence)
533   internalASTType = null;
534 }
535 :
536     (overtureUnionType)=> internalASTType = overtureUnionType
537 | internalASTType = overtureTypePre3
538 ;
539
540 overtureTypePre1 returns [InternalASTType internalASTType]
541 {
542 //Type operators with precedence 1 (lowest precedence)
543   internalASTType = null;
544 }
545 :
546     (overturePartialFunctionType)=> internalASTType = overturePartialFunctionType
547 | internalASTType = overtureTypePre2
548 ;
549
550 // bracketed type = '( , type, )';
551 overtureBracketedType returns [InternalASTBracketedType internalASTBracketedType]
552 {
553   internalASTBracketedType = null;
554   InternalASTType internalASTType = null;
555 }
556 :
557     lb:LBRACKET
558     internalASTType = overtureType
559     rb:RBRACKET
560     {
561       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
562       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
563       internalASTBracketedType = new InternalASTBracketedType(keywordLeftBracket, internalASTType,
564         keywordRightBracket);
565     }

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

569     }
570     ;
571
572 // basic type = 'bool' / 'int';
573 overtureBasicType returns [InternalASTBasicType internalASTBasicType]
574 {
575     //ASTAccess is abstract...
576     internalASTBasicType = null;
577 }
578 :
579 ( bo:BOOL
580   {
581     internalASTBasicType = new InternalASTBasicTypeBool(new InternalASTKeyword(bo,"Bool"));
582   }
583 | na:NAT
584   {
585     internalASTBasicType = new InternalASTBasicTypeNat(new InternalASTKeyword(na,"Nat"));
586   }
587 | nal:NAT1
588   {
589     internalASTBasicType = new InternalASTBasicTypeNat1(new InternalASTKeyword(nal,"Nat1"));
590   }
591 | in:INT
592   {
593     internalASTBasicType = new InternalASTBasicTypeInt(new InternalASTKeyword(in,"Int"));
594   }
595 | ra:RAT
596   {
597     internalASTBasicType = new InternalASTBasicTypeRat(new InternalASTKeyword(ra,"Rat"));
598   }
599 | re:REAL
600   {
601     internalASTBasicType = new InternalASTBasicTypeReal(new InternalASTKeyword(re,"Real"));
602   }
603 | ch:CHAR
604   {
605     internalASTBasicType = new InternalASTBasicTypeChar(new InternalASTKeyword(ch,"Char"));
606   }
607 | to:TOKEN
608   {
609     internalASTBasicType = new InternalASTBasicTypeToken(new InternalASTKeyword(to,"Token"));
610   }
611 )
612 ;
613
614 // quote type =
615 overtureQuoteType returns [InternalASTQuoteType internalASTQuoteType]
616 {
617     internalASTQuoteType = null;
618     InternalASTQuoteLiteral qliteral = null;
619 }
620 :
621 qliteral = overtureQuoteLiteral
622 {
623     internalASTQuoteType = new InternalASTQuoteType(qliteral);
624 }
625 ;
626
627 // composite type =
628 overtureCompositeType returns [InternalASTCompositeType internalASTCompositeType]
629 {
630     internalASTCompositeType = null;
631     InternalASTFieldList fieldlist = null;
632 }
633 :
634 cp:COMPOSE
635 id:IDENTIFIER
636 of:OF
637 fieldlist = overtureFieldList
638 en:END
639 {
640     InternalASTKeyword keywordCompose = new InternalASTKeyword(cp,"Compose");
641     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
642     InternalASTKeyword keywordOf = new InternalASTKeyword(of,"Of");
643     InternalASTKeyword keywordEnd = new InternalASTKeyword(en,"End");
644     internalASTCompositeType = new InternalASTCompositeType(keywordCompose, identifier, keywordOf
645         , fieldlist, keywordEnd);
646 }
647 ;
648
649 // field list = { field }
650 overtureFieldList returns [InternalASTFieldList internalASTFieldList]
651 {

```

I.1. ANTLR GRAMMAR FILE

```
652 internalASTFieldList = null;
653 List<InternalASTField> fields = new ArrayList<InternalASTField>();
654 InternalASTField internalASTField = null;
655 }
656 :
657 (
658     internalASTField = overtureField
659     {
660         fields.add(internalASTField);
661     }
662 )*
663 {
664     internalASTFieldList = new InternalASTFieldList(fields);
665 }
666 ;
667
668 // field = [ identifier, ':' ], type
669 overtureField returns [InternalASTField internalASTField]
670 {
671     internalASTField = null;
672     InternalASTType type = null;
673 }
674 :
675 (
676     (
677         (
678             id1:IDENTIFIER
679             co:COLON
680             type = overtureType
681             {
682                 InternalASTIdentifier identifier = new InternalASTIdentifier(id1);
683                 InternalASTKeyword operator = new InternalASTKeyword(co,"Colon");
684                 internalASTField = new InternalASTFieldColon(identifier, operator, type);
685             }
686         )
687         |
688         (
689             id2:IDENTIFIER
690             coli:COLONLINE
691             type = overtureType
692             {
693                 InternalASTIdentifier identifier = new InternalASTIdentifier(id2);
694                 InternalASTKeyword operator = new InternalASTKeyword(coli,"ColonLine");
695                 internalASTField = new InternalASTFieldColonLine(identifier, operator, type);
696             }
697         )
698         |
699         (
700             type = overtureType
701             {
702                 internalASTField = new InternalASTFieldBasic(type);
703             }
704         )
705     )
706 )
707 ;
708
709 // union type =
710 overtureUnionType returns [InternalASTUnionType internalASTUnionType]
711 {
712     internalASTUnionType = null;
713     InternalASTType type1 = null;
714     InternalASTKeyword keywordVBar = null;
715     InternalASTType type2 = null;
716 }
717 :
718 (
719     type1 = overtureTypePre3
720 )
721 vb:VBAR
722 {
723     keywordVBar = new InternalASTKeyword(vb,"VBar");
724 }
725 (
726     type2 = overtureTypePre3
727     {
728         internalASTUnionType = new InternalASTUnionType(type1,keywordVBar,type2);
729     }
730 )
731
732 (options {greedy=true};):
733 vbn:VBAR
734 {
735
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

736     keywordVBar = new InternalASTKeyword(vbn, "VBar");
737   }
738   (
739     type2 = overtureTypePre3
740     {
741       internalASTUnionType = new InternalASTUnionType(internalASTUnionType, keywordVBar, type2);
742     }
743   )
744   )*)
745   ;
746
747 // product type =
748 overtureProductType returns [InternalASTProductType internalASTProductType]
749 {
750   internalASTProductType = null;
751   InternalASTType type1 = null;
752   InternalASTKeyword keywordAsterix = null;
753   InternalASTType type2 = null;
754 }
755 :
756 (
757   type1 = overtureTypePre4
758 )
759 as:ASTERIX
760 {
761   keywordAsterix = new InternalASTKeyword(as, "Asterix");
762 }
763 (
764   type2 = overtureTypePre3
765   {
766     internalASTProductType = new InternalASTProductType(type1, keywordAsterix, type2);
767   }
768 )
769 ;
770
771 // optional type =
772 overtureOptionalType returns [InternalASTOptionalType internalASTOptionalType]
773 {
774   internalASTOptionalType = null;
775   InternalASTType type = null;
776 }
777 :
778 lbc:LBRACK
779 type = overtureType
780 rbc:RBRACK
781 {
782   InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lbc, "LeftBrack");
783   InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rbc, "RightBrack");
784   internalASTOptionalType = new InternalASTOptionalType(keywordLeftBrack, type,
785     keywordRightBrack);
786 }
787 ;
788
789 // set type =
790 overtureSetType returns [InternalASTSetType internalASTSetType]
791 {
792   internalASTSetType = null;
793   InternalASTType type = null;
794 }
795 :
796 se:SET
797 of:OF
798 type = overtureTypePre5
799 {
800   InternalASTKeyword keywordSet = new InternalASTKeyword(se, "Set");
801   InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
802   internalASTSetType = new InternalASTSetType(keywordSet, keywordOf, type);
803 }
804 ;
805
806 // seq type =
807 overtureSeqType returns [InternalASTSeqType internalASTSeqType]
808 {
809   internalASTSeqType = null;
810 }
811 :
812 internalASTSeqType = overtureSeq0Type
813 |
814 internalASTSeqType = overtureSeq1Type
815 ;
816
817 // seq0 type = 'seq of', type;
818 overtureSeq0Type returns [InternalASTSeq0Type internalASTSeq0Type]
819 {

```

I.1. ANTLR GRAMMAR FILE

```
819     internalASTSeq0Type = null;
820     InternalASTType type;
821 }
822 :
823     se0:SEQ
824     of:OF
825     type = overtureTypePre5
826     {
827         InternalASTKeyword keywordSeq = new InternalASTKeyword(se0, "Seq");
828         InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
829         internalASTSeq0Type = new InternalASTSeq0Type(keywordSeq, keywordOf, type);
830     }
831 ;
832
833 // seq1 type = 'seq1 of', type;
834 overtureSeq1Type returns [InternalASTSeq1Type internalASTSeq1Type]
835 {
836     internalASTSeq1Type = null;
837     InternalASTType type;
838 }
839 :
840     se1:SEQ1
841     of:OF
842     type = overtureTypePre5
843     {
844         InternalASTKeyword keywordSeq1 = new InternalASTKeyword(se1, "Seq1");
845         InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
846         internalASTSeq1Type = new InternalASTSeq1Type(keywordSeq1, keywordOf, type);
847     }
848 ;
849
850 // map type =
851 overtureMapType returns [InternalASTMapType internalASTMapType]
852 {
853     internalASTMapType = null;
854 }
855 :
856     internalASTMapType = overtureGeneralMapType
857     |
858     internalASTMapType = overtureInjectiveMapType
859 ;
860
861 // general map type = 'map', type, 'to', type;
862 overtureGeneralMapType returns [InternalASTGeneralMapType internalASTGeneralMapType]
863 {
864     internalASTGeneralMapType = null;
865     InternalASTType type1;
866     InternalASTType type2;
867 }
868 :
869     ma:MAP
870     type1 = overtureTypePre4
871     to:TO
872     type2 = overtureTypePre4
873     {
874         InternalASTKeyword keywordMap = new InternalASTKeyword(ma, "Map");
875         InternalASTKeyword keywordTo = new InternalASTKeyword(to, "To");
876         internalASTGeneralMapType = new InternalASTGeneralMapType(keywordMap, type1, keywordTo, type2
877             );
878     }
879 ;
880
881 // injective map type = 'inmap', type, 'to', type;
882 overtureInjectiveMapType returns [InternalASTInjectiveMapType internalASTInjectiveMapType]
883 {
884     internalASTInjectiveMapType = null;
885     InternalASTType type1;
886     InternalASTType type2;
887 }
888 :
889     im:INMAP
890     type1 = overtureTypePre4
891     to:TO
892     type2 = overtureTypePre4
893     {
894         InternalASTKeyword keywordInmap = new InternalASTKeyword(im, "Inmap");
895         InternalASTKeyword keywordTo = new InternalASTKeyword(to, "To");
896         internalASTInjectiveMapType = new InternalASTInjectiveMapType(keywordInmap, type1, keywordTo,
897             type2);
898     }
899 ;
900 // function type = partial function type
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

901 //      | total function type;
902 overtureFunctionType returns [InternalASTFunctionType internalASTFunctionType]
903 {
904     internalASTFunctionType = null;
905 }
906 :
907     (
908         (overturePartialFunctionType)=> internalASTFunctionType = overturePartialFunctionType
909         | internalASTFunctionType = overtureTotalFunctionType
910     )
911 ;
912
913 // partial function type = discretionary type '->' type
914 overturePartialFunctionType returns [InternalASTPartialFunctionType
915     internalASTPartialFunctionType]
916 {
917     internalASTPartialFunctionType = null;
918     InternalASTDiscretionaryType disctype = null;
919     InternalASTType type = null;
920 }
921 :
922     disctype = overtureDiscretionaryType
923     arw : LINEARROW
924     type = overtureType
925     {
926         InternalASTKeyword keywordLineArrow = new InternalASTKeyword(arw, "Arrow");
927         internalASTPartialFunctionType = new InternalASTPartialFunctionType(disctype,
928             keywordLineArrow, type);
929     }
930 ;
931
932 // Total total type = discretionary type '+>' type
933 overtureTotalFunctionType returns [InternalASTTotalFunctionType internalASTTotalFunctionType]
934 {
935     internalASTTotalFunctionType = null;
936     InternalASTDiscretionaryType disctype = null;
937     InternalASTType type = null;
938 }
939 :
940     disctype = overtureDiscretionaryType
941     kwpa : PLUSARROW
942     type = overtureType
943     {
944         InternalASTKeyword keywordPlusArrow = new InternalASTKeyword(kwpa, "PlusArrow");
945         internalASTTotalFunctionType = new InternalASTTotalFunctionType(disctype, keywordPlusArrow,
946             type);
947     }
948 ;
949
950 // discretionary type = type | '(,)' ;
951 overtureDiscretionaryType returns [InternalASTDiscretionaryType internalASTDiscretionaryType]
952 {
953     internalASTDiscretionaryType = null;
954     InternalASTType type2 = null;
955 }
956 :
957     (
958         type2 = overtureTypePre2
959         {
960             internalASTDiscretionaryType = new InternalASTDiscretionaryTypeType(type2);
961         }
962     )
963 |
964     (
965         lb:LBRACKET
966         rb:RBRACKET
967         {
968             InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
969             InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
970             internalASTDiscretionaryType = new InternalASTDiscretionaryTypeBrackets(keywordLeftBracket,
971                 keywordRightBracket);
972         }
973     )
974 ;
975
976 // type name =
977 overtureTypeName returns [InternalASTTypeName internalASTTypeName]
978 {
979     internalASTTypeName = null;
980     InternalASTName internalASTName = null;
981 }
982 :
983     internalASTName = overtureName
984     {

```


I.1. ANTLR GRAMMAR FILE

```
981     internalASTTypeName = new InternalASTTypeName(internalASTName);
982   }
983   ;
984
985
986
987
988 // type variable =
989 overtуреTypeVariable returns [InternalASTTypeVariable internalASTTypeVariable]
990 {
991   internalASTTypeVariable = null;
992   InternalASTTypeVariableIdentifier typeVariableIdentifier = null;
993 }
994 :
995   typeVariableIdentifier = overtуреTypeVariableIdentifier
996   {
997     internalASTTypeVariable = new InternalASTTypeVariable(typeVariableIdentifier);
998   }
999 ;
1000
1001 // invariant = 'inv', invariant initial function;
1002 overtуреInvariant returns [InternalASTInvariant internalASTInvariant]
1003 {
1004   internalASTInvariant = null;
1005   InternalASTKeyword keywordInv = null;
1006   InternalASTInvariantInitialFunction invariantInitialFunction = null;
1007 }
1008 :
1009   in:INV
1010   invariantInitialFunction = overtуреInvariantInitialFunction
1011   {
1012     keywordInv = new InternalASTKeyword(in, "Inv");
1013     internalASTInvariant = new InternalASTInvariant(keywordInv, invariantInitialFunction);
1014   }
1015 ;
1016
1017 // invariant initial function = pattern, '==', expression;
1018 overtуреInvariantInitialFunction returns [InternalASTInvariantInitialFunction
1019   internalASTInvariantInitialFunction]
1020 {
1021   internalASTInvariantInitialFunction = null;
1022   InternalASTKeyword keywordDoubleEqual = null;
1023   InternalASTPattern pattern = null;
1024   InternalASTExpression expression = null;
1025 }
1026 :
1027   pattern = overtуреPattern
1028   de:DOUBLEEQUAL
1029   expression = overtуреExpression
1030   {
1031     keywordDoubleEqual = new InternalASTKeyword(de, "DoubleEqual");
1032     internalASTInvariantInitialFunction = new InternalASTInvariantInitialFunction(pattern,
1033       keywordDoubleEqual, expression);
1034   }
1035 ;
1036
1037 // VALUE DEFINITIONS
1038 // value definitions = 'values', [access value definition,
1039 //   {';', access value definition },
1040 //   ['; ']];
1041 overtуреValueDefinitions returns [InternalASTValueDefinitions internalASTValueDefinitions]
1042 {
1043   internalASTValueDefinitions = null;
1044   InternalASTKeyword keywordValues = null;
1045   List<InternalASTValueDefinitionsElement> valueDefinitionsElements = new ArrayList<
1046     InternalASTValueDefinitionsElement>();
1047   InternalASTValueDefinitionsElement currentElement = null;
1048   InternalASTAccessValueDefinition internalASTAccessValueDefinition = null;
1049 }
1050 :
1051   va:VALUES
1052   {
1053     keywordValues = new InternalASTKeyword(va, "Values");
1054   }
1055 (
1056   internalASTAccessValueDefinition = overtуреAccessValueDefinition
1057   {
1058     currentElement = new InternalASTValueDefinitionsElement(internalASTAccessValueDefinition);
1059     valueDefinitionsElements.add(currentElement);
1060   }
1061 )
1062 (options {greedy=true};):
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1062     seNoN:SEMICOLON
1063     {
1064         InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN, "Semicolon");
1065         currentElement.setKeywordSemicolon(semicolonNoN);
1066         currentElement.setEndPositionFromNode(semicolonNoN);
1067     }
1068     internalASTAccessValueDefinition = overtureAccessValueDefinition
1069     {
1070         currentElement = new InternalASTValueDefinitionsElement(internalASTAccessValueDefinition);
1071         valueDefinitionsElements.add(currentElement);
1072     }
1073     )*
1074     (seLast:SEMICOLON
1075     {
1076         InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast, "Semicolon");
1077         currentElement.setKeywordSemicolon(semicolonLast);
1078         currentElement.setEndPositionFromNode(semicolonLast);
1079     }
1080     )?
1081     )?
1082     )?
1083     {
1084         internalASTValueDefinitions = new InternalASTValueDefinitions(keywordValues,
1085             valueDefinitionsElements);
1086     }
1087     ;
1088     ;
1089     ;
1090     // access value definition = [access] value definition;
1091     overtureAccessValueDefinition returns [InternalASTAccessValueDefinition
1092         internalASTAccessValueDefinition]
1093     {
1094         internalASTAccessValueDefinition = null;
1095         InternalASTAccess internalASTAccess = null;
1096         InternalASTValueDefinition internalASTValueDefinition = null;
1097     }
1098     :
1099     (
1100         internalASTAccess = overtureAccess
1101         )?
1102         internalASTValueDefinition = overtureValueDefinition
1103         {
1104             internalASTAccessValueDefinition = new InternalASTAccessValueDefinition(internalASTAccess,
1105                 internalASTValueDefinition);
1106         }
1107     ;
1108     // value definition = pattern [':', type], '=' expression
1109     overtureValueDefinition returns [InternalASTValueDefinition internalASTValueDefinition]
1110     {
1111         internalASTValueDefinition = null;
1112         InternalASTPattern pattern = null;
1113         InternalASTKeyword keywordColon = null;
1114         InternalASTType internalASTType = null;
1115         InternalASTExpression expression = null;
1116     }
1117     :
1118     pattern = overturePattern
1119     (
1120         co:COLON
1121         internalASTType = overtureType
1122         {
1123             keywordColon = new InternalASTKeyword(co, "Colon");
1124         }
1125     )?
1126     kweq:EQUALSIGN
1127     expression = overtureExpression
1128     {
1129         if(expression == null) {System.err.println("expression i valuedefinition er NULL"); }
1130         InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq, "Equalsign");
1131         internalASTValueDefinition = new InternalASTValueDefinition(pattern, keywordColon,
1132             internalASTType, keywordEqualsign, expression);
1133     }
1134     ;
1135     // FUNCTION DEFINITIONS
1136     // function definitions = 'function', [access function definition,
1137     //     {':', access function definition },
1138     //     [':'];
1139     overtureFunctionDefinitions returns [InternalASTFunctionDefinitions
1140         internalASTFunctionDefinitions]

```

I.1. ANTLR GRAMMAR FILE

```

1141 {
1142     internalASTFunctionDefinitions = null;
1143     InternalASTKeyword keywordFunctions = null;
1144     List<InternalASTFunctionDefinitionsElement> FunctionDefinitionsElements = new ArrayList<
        InternalASTFunctionDefinitionsElement>();
1145     InternalASTFunctionDefinitionsElement currentElement = null;
1146     InternalASTAccessFunctionDefinition internalASTAccessFunctionDefinition = null;
1147 }
1148 :
1149 fu:FUNCTIONS
1150 {
1151     keywordFunctions = new InternalASTKeyword(fu, "Functions");
1152 }
1153 (
1154     internalASTAccessFunctionDefinition = overtureAccessFunctionDefinition
1155     {
1156         currentElement = new InternalASTFunctionDefinitionsElement(
            internalASTAccessFunctionDefinition);
1157         FunctionDefinitionsElements.add(currentElement);
1158     }
1159     (options {greedy=true};
1160     seNoN:SEMICOLON
1161     {
1162         InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN, "Semicolon");
1163         currentElement.setKeywordSemicolon(semicolonNoN);
1164         currentElement.setEndPositionFromNode(semicolonNoN);
1165     }
1166     internalASTAccessFunctionDefinition = overtureAccessFunctionDefinition
1167     {
1168         currentElement = new InternalASTFunctionDefinitionsElement(
            internalASTAccessFunctionDefinition);
1169         FunctionDefinitionsElements.add(currentElement);
1170     }
1171     )*
1172 )?
1173 (seLast:SEMICOLON
1174 {
1175     InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast, "Semicolon");
1176     currentElement.setKeywordSemicolon(semicolonLast);
1177     currentElement.setEndPositionFromNode(semicolonLast);
1178 }
1179 )?
1180 )?
1181 {
1182     internalASTFunctionDefinitions = new InternalASTFunctionDefinitions(keywordFunctions,
        FunctionDefinitionsElements);
1183 }
1184 ;
1185
1186 // access function definition = [access], function definition;
1187 overtureAccessFunctionDefinition returns [InternalASTAccessFunctionDefinition
        internalASTAccessFunctionDefinition]
1188 {
1189     internalASTAccessFunctionDefinition = null;
1190     InternalASTAccess internalASTAccess = null;
1191     InternalASTFunctionDefinition internalASTFunctionDefinition = null;
1192 }
1193 :
1194 (
1195     internalASTAccess = overtureAccess
1196 )?
1197 internalASTFunctionDefinition = overtureFunctionDefinition
1198 {
1199     internalASTAccessFunctionDefinition = new InternalASTAccessFunctionDefinition(
        internalASTAccess, internalASTFunctionDefinition);
1200 }
1201 ;
1202
1203 // function definition = explicit function definition
1204 // | implicit function definition;
1205 // | extended explicit function definition
1206 overtureFunctionDefinition returns [InternalASTFunctionDefinition internalASTFunctionDefinition]
1207 {
1208     internalASTFunctionDefinition = null;
1209 }
1210 :
1211 (
1212     (overtureExplicitFunctionDefinition=> internalASTFunctionDefinition =
        overtureExplicitFunctionDefinition
1213     | (overtureImplicitFunctionDefinition=> internalASTFunctionDefinition =
        overtureImplicitFunctionDefinition

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1217 | internalASTFunctionDefinition = overtureExtendedExplicitFunctionDefinition
1218 | )
1219 ;
1220
1221
1222
1223 //explicit function definition = identifier, [ type variable list ], ':',
1224 //      function type,
1225 //      identifier, parameters list,
1226 //      '=', function body,
1227 //      [ 'pre', expression ],
1228 //      [ 'post', expression ] ;
1229
1230 overtureExplicitFunctionDefinition returns [InternalASTExplicitFunctionDefinition
1231         internalASTExplicitFunctionDefinition]
1232 {
1233     internalASTExplicitFunctionDefinition = null;
1234     InternalASTTypeVariableList internalASTTypeVariableList = null;
1235     InternalASTFunctionType internalASTFunctionType = null;
1236     InternalASTParametersList internalASTParametersList = null;
1237     InternalASTFunctionBody internalASTFunctionBody = null;
1238     InternalASTKeyword keywordPre = null;
1239     InternalASTExpression internalASTExpressionPre = null;
1240     InternalASTKeyword keywordPost = null;
1241     InternalASTExpression internalASTExpressionPost = null;
1242 }
1243 : id1:IDENTIFIER
1244 (
1245     internalASTTypeVariableList = overtureTypeVariableList
1246 )?
1247 kwc:COLON
1248     internalASTFunctionType = overtureFunctionType
1249     id2:IDENTIFIER
1250     internalASTParametersList = overtureParametersList
1251     kwde:DOUBLEEQUAL
1252     internalASTFunctionBody = overtureFunctionBody
1253     (
1254         kwpre:PRE
1255         internalASTExpressionPre = overtureExpression
1256         {
1257             keywordPre = new InternalASTKeyword(kwpre, "Pre");
1258         }
1259     )?
1260     (
1261         kwpost:POST
1262         internalASTExpressionPost = overtureExpression
1263         {
1264             keywordPost = new InternalASTKeyword(kwpost, "Post");
1265         }
1266     )?
1267     {
1268         InternalASTIdentifier internalASTIdentifier1 = new InternalASTIdentifier(id1);
1269         InternalASTKeyword keywordColon = new InternalASTKeyword(kwc, "Colon");
1270         InternalASTIdentifier internalASTIdentifier2 = new InternalASTIdentifier(id2);
1271         InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde, "DoubleEqual");
1272         internalASTExplicitFunctionDefinition = new InternalASTExplicitFunctionDefinition(
1273             internalASTIdentifier1, internalASTTypeVariableList, keywordColon,
1274             internalASTFunctionType, internalASTIdentifier2, internalASTParametersList,
1275             keywordDoubleEqual, internalASTFunctionBody, keywordPre, internalASTExpressionPre,
1276             keywordPost, internalASTExpressionPost);
1277     }
1278 }
1279 ;
1280
1281
1282 //implicit function definition = identifier, [ type variable list ],
1283 //      parameter types,
1284 //      identifier type pair list,
1285 //      [ 'pre', expression ],
1286 //      [ 'post', expression ] ;
1287
1288 overtureImplicitFunctionDefinition returns [InternalASTImplicitFunctionDefinition
1289         internalASTImplicitFunctionDefinition]
1290 {
1291     internalASTImplicitFunctionDefinition = null;
1292     InternalASTTypeVariableList internalASTTypeVariableList = null;
1293     InternalASTParameterTypes internalASTParameterTypes = null;
1294     InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
1295     InternalASTKeyword keywordPre = null;
1296     InternalASTExpression internalASTExpressionPre = null;
1297     InternalASTKeyword keywordPost = null;
1298     InternalASTExpression internalASTExpressionPost = null;
1299 }
1300 : id:IDENTIFIER
1301 (

```

I.1. ANTLR GRAMMAR FILE

```

1295         internalASTTypeVariableList = overtureTypeVariableList
1296     )?
1297     internalASTParameterTypes = overtureParameterTypes
1298     internalASTIdentifierTypePairList = overtureIdentifierTypePairList
1299     (
1300         kwpre:PRE
1301         internalASTExpressionPre = overtureExpression
1302         {
1303             keywordPre = new InternalASTKeyword(kwpre, "Pre");
1304         }
1305     )?
1306     kwpost:POST
1307     internalASTExpressionPost = overtureExpression
1308     {
1309         keywordPost = new InternalASTKeyword(kwpost, "Post");
1310         InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
1311         internalASTImplicitFunctionDefinition = new InternalASTImplicitFunctionDefinition(
1312             internalASTIdentifier, internalASTTypeVariableList, internalASTParameterTypes,
1313             internalASTIdentifierTypePairList, keywordPre, internalASTExpressionPre, keywordPost,
1314             internalASTExpressionPost);
1315     }
1316 ;
1317
1318 //extended explicit function definition = identifier, [ type variable list ],
1319 // parameter types,
1320 // identifier type pair list,
1321 // '=', function body,
1322 // [ 'pre', expression ],
1323 // [ 'post', expression ] ;
1324
1325 overtureExtendedExplicitFunctionDefinition returns [InternalASTExtendedExplicitFunctionDefinition
1326     internalASTExtendedExplicitFunctionDefinition]
1327 {
1328     internalASTExtendedExplicitFunctionDefinition = null;
1329     internalASTTypeVariableList internalASTTypeVariableList = null;
1330     internalASTParameterTypes internalASTParameterTypes = null;
1331     internalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
1332     internalASTFunctionBody internalASTFunctionBody = null;
1333     InternalASTKeyword keywordPre = null;
1334     InternalASTExpression internalASTExpressionPre = null;
1335     InternalASTKeyword keywordPost = null;
1336     InternalASTExpression internalASTExpressionPost = null;
1337 }
1338 : id:IDENTIFIER
1339 (
1340     internalASTTypeVariableList = overtureTypeVariableList
1341 )?
1342 internalASTParameterTypes = overtureParameterTypes
1343 internalASTIdentifierTypePairList = overtureIdentifierTypePairList
1344 de:DOUBLEEQUAL
1345 internalASTFunctionBody = overtureFunctionBody
1346 (
1347     kwpre:PRE
1348     internalASTExpressionPre = overtureExpression
1349     {
1350         keywordPre = new InternalASTKeyword(kwpre, "Pre");
1351     }
1352 )?
1353 (
1354     kwpost:POST
1355     internalASTExpressionPost = overtureExpression
1356     {
1357         keywordPost = new InternalASTKeyword(kwpost, "Post");
1358     }
1359 )?
1360 {
1361     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
1362     InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(de, "DoubleEqual");
1363     internalASTExtendedExplicitFunctionDefinition = new
1364         InternalASTExtendedExplicitFunctionDefinition(internalASTIdentifier,
1365             internalASTTypeVariableList, internalASTParameterTypes,
1366             internalASTIdentifierTypePairList, keywordDoubleEqual, internalASTFunctionBody,
1367             keywordPre, internalASTExpressionPre, keywordPost, internalASTExpressionPost);
1368 }
1369 ;
1370
1371 // type variable list = '|', type variable identifier,
1372 // { '|', type variable identifier }, '|';
1373 overtureTypeVariableList returns [InternalASTTypeVariableList internalASTTypeVariableList]
1374 {
1375     internalASTTypeVariableList = null;
1376     List<InternalASTTypeVariableListElement> typeVariableListElements = new ArrayList<
1377         InternalASTTypeVariableListElement>();

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1370 | InternalASTTypeVariableListElement currentElement = null;
1371 | InternalASTTypeVariableIdentifier typeVariableIdentifier = null;
1372 | }
1373 | :
1374 |   kwlb:LBRACK
1375 |   typeVariableIdentifier = overtureTypeVariableIdentifier
1376 |   {
1377 |     currentElement = new InternalASTTypeVariableListElement(typeVariableIdentifier);
1378 |     typeVariableListElements.add(currentElement);
1379 |   }
1380 |   (
1381 |     kwc:COMMA
1382 |     typeVariableIdentifier = overtureTypeVariableIdentifier
1383 |     {
1384 |       InternalASTKeyword keywordComma = new InternalASTKeyword(kwc, "Comma");
1385 |       currentElement = new InternalASTTypeVariableListElement(keywordComma,
1386 |         typeVariableIdentifier);
1387 |       typeVariableListElements.add(currentElement);
1388 |     }
1389 |   )*
1390 |   kwrb:RBRACK
1391 |   {
1392 |     InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb, "LeftBrack");
1393 |     InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb, "RightBrack");
1394 |     internalASTTypeVariableList = new InternalASTTypeVariableList(keywordLeftBrack,
1395 |       typeVariableListElements, keywordRightBrack);
1396 |   }
1397 | ;
1398 |
1399 | //identifier type pair = identifier, ':' type;
1400 | overtureIdentifierTypePair returns [InternalASTIdentifierTypePair internalASTIdentifierTypePair]
1401 | {
1402 |   internalASTIdentifierTypePair = null;
1403 |   internalASTType internalASTType = null;
1404 | }
1405 | : id:IDENTIFIER
1406 |   kwc:COLON
1407 |   internalASTType = overtureType
1408 |   {
1409 |     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
1410 |     InternalASTKeyword keywordColon = new InternalASTKeyword(kwc, "Colon");
1411 |     internalASTIdentifierTypePair = new InternalASTIdentifierTypePair(internalASTIdentifier,
1412 |       keywordColon, internalASTType);
1413 |   }
1414 | ;
1415 |
1416 | //pattern list type pair = pattern list, ':' type;
1417 | overturePatternListTypePair returns [InternalASTPatternListTypePair
1418 |   internalASTPatternListTypePair]
1419 | {
1420 |   internalASTPatternListTypePair = null;
1421 |   internalASTType internalASTType = null;
1422 |   InternalASTPatternList patternList = null;
1423 | }
1424 | :
1425 |   patternList = overturePatternList
1426 |   kwc:COLON
1427 |   internalASTType = overtureType
1428 |   {
1429 |     InternalASTKeyword keywordColon = new InternalASTKeyword(kwc, "Colon");
1430 |     internalASTPatternListTypePair = new InternalASTPatternListTypePair(patternList, keywordColon
1431 |       , internalASTType);
1432 |   }
1433 | ;
1434 |
1435 | //parameter types = '(', [ pattern type pair list ], ')';
1436 | overtureParameterTypes returns [InternalASTParameterTypes internalASTParameterTypes]
1437 | {
1438 |   internalASTParameterTypes = null;
1439 |   InternalASTPatternTypePairList patternTypePairList = null;
1440 | }
1441 | : kwlb:LBRACKET
1442 |   (
1443 |     patternTypePairList = overturePatternTypePairList
1444 |   )?
1445 |   kwrb:RBRACKET
1446 |   {
1447 |     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
1448 |     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
1449 |     internalASTParameterTypes = new InternalASTParameterTypes(keywordLeftBracket,
1450 |       patternTypePairList, keywordRightBracket);

```

I.1. ANTLR GRAMMAR FILE

```

1448     }
1449     ;
1450
1451     //identifier type pair list = identifier type pair,
1452     //      { ',', identifier type pair } ;
1453     overtуреIdentifierTypePairList returns [InternalASTIdentifierTypePairList
1454         internalASTIdentifierTypePairList]
1455     {
1456         internalASTIdentifierTypePairList = null;
1457         InternalASTIdentifierTypePairListElement internalASTIdentifierTypePairListElement = null;
1458         List<InternalASTIdentifierTypePairListElement> identifierTypePairListElements = new ArrayList<
1459             InternalASTIdentifierTypePairListElement>();
1460         InternalASTIdentifierTypePair internalASTIdentifierTypePair = null;
1461     }
1462     :
1463     internalASTIdentifierTypePair = overtуреIdentifierTypePair
1464     {
1465         internalASTIdentifierTypePairListElement = new InternalASTIdentifierTypePairListElement (
1466             internalASTIdentifierTypePair);
1467         identifierTypePairListElements.add(internalASTIdentifierTypePairListElement);
1468     }
1469     (options {greedy=true};):
1470     kwcomma:COMMA
1471     {
1472         internalASTIdentifierTypePair = overtуреIdentifierTypePair
1473         {
1474             InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
1475             internalASTIdentifierTypePairListElement = new InternalASTIdentifierTypePairListElement (
1476                 keywordComma, internalASTIdentifierTypePair);
1477             identifierTypePairListElements.add(internalASTIdentifierTypePairListElement);
1478         }
1479     }
1480     )*
1481     {
1482         internalASTIdentifierTypePairList = new InternalASTIdentifierTypePairList (
1483             identifierTypePairListElements);
1484     }
1485     ;
1486
1487     //pattern type pair list = pattern list type pair,
1488     //      { ',', pattern list type pair } ;
1489     overtуреPatternTypePairList returns [InternalASTPatternTypePairList
1490         internalASTPatternTypePairList]
1491     {
1492         internalASTPatternTypePairList = null;
1493         InternalASTPatternListTypePair internalASTPatternListTypePair = null;
1494         InternalASTPatternTypePairListElement internalASTPatternTypePairListElement = null;
1495         List<InternalASTPatternTypePairListElement> patternTypePairListElements = new ArrayList<
1496             InternalASTPatternTypePairListElement>();
1497     }
1498     :
1499     internalASTPatternListTypePair = overtуреPatternListTypePair
1500     {
1501         internalASTPatternTypePairListElement = new InternalASTPatternTypePairListElement (
1502             internalASTPatternListTypePair);
1503         patternTypePairListElements.add(internalASTPatternTypePairListElement);
1504     }
1505     (
1506     kwcomma:COMMA
1507     {
1508         internalASTPatternListTypePair = overtуреPatternListTypePair
1509         {
1510             InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
1511             internalASTPatternTypePairListElement = new InternalASTPatternTypePairListElement (keywordComma
1512                 , internalASTPatternListTypePair);
1513             patternTypePairListElements.add(internalASTPatternTypePairListElement);
1514         }
1515     }
1516     )*
1517     {
1518         internalASTPatternTypePairList = new InternalASTPatternTypePairList (
1519             patternTypePairListElements);
1520     }
1521     ;
1522
1523     //parameters list = parameters, { parameters } ;
1524     overtуреParametersList returns [InternalASTParametersList internalASTParametersList]
1525     {
1526         internalASTParametersList = null;
1527         InternalASTParameters internalASTParameters = null;
1528         List<InternalASTParameters> parameters = new ArrayList<InternalASTParameters>();
1529     }
1530     :
1531     internalASTParameters = overtуреParameters
1532     {
1533         parameters.add(internalASTParameters);
1534     }

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1522     (
1523         internalASTParameters = overtureParameters
1524     {
1525         parameters.add(internalASTParameters);
1526     }
1527     )*
1528     {
1529         internalASTParametersList = new InternalASTParametersList(parameters);
1530     }
1531     ;
1532
1533     // parameters = '(', [ pattern list ], ')';
1534     overtureParameters returns [InternalASTParameters internalASTParameters]
1535     {
1536         internalASTParameters = null;
1537         InternalASTPatternList internalASTPatternList = null;
1538     }
1539     :
1540     kwlb:LBACKET
1541     (
1542         internalASTPatternList = overturePatternList
1543     )?
1544     kwrb:RBACKET
1545     {
1546         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
1547         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
1548         internalASTParameters = new InternalASTParameters(keywordLeftBracket, internalASTPatternList,
1549             keywordRightBracket);
1550     }
1551     ;
1552     // function body = expression
1553     //           | 'Is not yet specified';
1554     //           | 'is subclass responsibility'
1555     overtureFunctionBody returns [InternalASTFunctionBody internalASTFunctionBody]
1556     {
1557         internalASTFunctionBody = null;
1558     }
1559     :
1560     (
1561         (overtureExpression)=> internalASTFunctionBody = overtureExpression
1562     |
1563     (
1564         is:IS
1565         no:NOT
1566         ye:YET
1567         sp:SPECIFIED
1568         {
1569             InternalASTKeyword keywordIs = new InternalASTKeyword(is,"Is");
1570             InternalASTKeyword keywordNot = new InternalASTKeyword(no,"Not");
1571             InternalASTKeyword keywordYet = new InternalASTKeyword(ye,"Yet");
1572             InternalASTKeyword keywordSpecified = new InternalASTKeyword(sp,"Specified");
1573             internalASTFunctionBody = new InternalASTFunctionBodyIsNotYetSpecified(keywordIs,
1574                 keywordNot, keywordYet, keywordSpecified);
1575         }
1576     )
1577     |
1578     (
1579         is2:IS
1580         su:SUBCLASS
1581         re:RESPONSIBILITY
1582         {
1583             InternalASTKeyword keywordIs = new InternalASTKeyword(is2,"Is");
1584             InternalASTKeyword keywordSubclass = new InternalASTKeyword(su,"Subclass");
1585             InternalASTKeyword keywordResponsibility = new InternalASTKeyword(re,"Responsibility");
1586             internalASTFunctionBody = new InternalASTFunctionBodyIsSubclassResponsibility(keywordIs,
1587                 keywordSubclass, keywordResponsibility);
1588         }
1589     )
1590     )
1591     ;
1592     // OPERATION DEFINITIONS
1593
1594     // operation definitions = 'values', [access operation definition,
1595     //           {';', access operation definition },
1596     //           [';'];
1597     overtureOperationDefinitions returns [InternalASTOperationDefinitions
1598         internalASTOperationDefinitions]
1599     {
1600         internalASTOperationDefinitions = null;
1601         InternalASTKeyword keywordOperations = null;
1602         List<InternalASTOperationDefinitionsElement> operationDefinitionsElements = new ArrayList<
1603             InternalASTOperationDefinitionsElement>();

```


I.1. ANTLR GRAMMAR FILE

```

1601 InternalASTOperationDefinitionsElement currentElement = null;
1602 InternalASTAccessOperationDefinition internalASTAccessOperationDefinition = null;
1603 }
1604 :
1605 op:OPERATIONS
1606 {
1607     keywordOperations = new InternalASTKeyword(op,"Operations");
1608 }
1609 (
1610     internalASTAccessOperationDefinition = overtureAccessOperationDefinition
1611     {
1612         currentElement = new InternalASTOperationDefinitionsElement (
1613             internalASTAccessOperationDefinition);
1614         operationDefinitionsElements.add(currentElement);
1615     }
1616     (options {greedy=true};)
1617     seNoN:SEMICOLON
1618     {
1619         InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
1620         currentElement.setKeywordSemicolon(semicolonNoN);
1621         currentElement.setEndPositionFromNode(semicolonNoN);
1622     }
1623     internalASTAccessOperationDefinition = overtureAccessOperationDefinition
1624     {
1625         currentElement = new InternalASTOperationDefinitionsElement (
1626             internalASTAccessOperationDefinition);
1627         operationDefinitionsElements.add(currentElement);
1628     }
1629     )*
1630     (seLast:SEMICOLON
1631     {
1632         InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
1633         currentElement.setKeywordSemicolon(semicolonLast);
1634         currentElement.setEndPositionFromNode(semicolonLast);
1635     }
1636     )?
1637     )?
1638     {
1639         internalASTOperationDefinitions = new InternalASTOperationDefinitions(keywordOperations,
1640             operationDefinitionsElements);
1641     }
1642     ;
1643
1644
1645 // access operation definition = [access], operation definition;
1646 overtureAccessOperationDefinition returns [InternalASTAccessOperationDefinition
1647     internalASTAccessOperationDefinition]
1648 {
1649     internalASTAccessOperationDefinition = null;
1650     InternalASTAccess internalASTAccess = null;
1651     InternalASTOperationDefinition internalASTOperationDefinition = null;
1652 }
1653 :
1654 (
1655     internalASTAccess = overtureAccess
1656 )?
1657 internalASTOperationDefinition = overtureOperationDefinition
1658 {
1659     internalASTAccessOperationDefinition = new InternalASTAccessOperationDefinition(
1660         internalASTAccess, internalASTOperationDefinition);
1661 }
1662 ;
1663
1664 // operation definition = explicit operation definition
1665 // | implicit operation definition;
1666 // | extended explicit operation definition
1667 overtureOperationDefinition returns [InternalASTOperationDefinition
1668     internalASTOperationDefinition]
1669 {
1670     internalASTOperationDefinition = null;
1671 }
1672 :
1673 (
1674     internalASTOperationDefinition = overtureExplicitOperationDefinition
1675     | (overtureImplicitOperationDefinition => internalASTOperationDefinition =
1676         overtureImplicitOperationDefinition
1677     | internalASTOperationDefinition = overtureExtendedExplicitOperationDefinition
1678     )
1679 )
1680 ;
1681 //explicit operation definition = identifier , ':', operation type,

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1678 //      identifier, parameters,
1679 //      '=',' operation body,
1680 //      [ 'pre', expression ],
1681 //      [ 'post', expression ] ;
1682 overtureExplicitOperationDefinition returns [InternalASTExplicitOperationDefinition
      internalASTExplicitOperationDefinition]
1683 {
1684   internalASTExplicitOperationDefinition = null;
1685   internalASTOperationType internalASTOperationType = null;
1686   internalASTParameters internalASTParameters = null;
1687   internalASTOperationBody internalASTOperationBody = null;
1688   internalASTKeyword keywordPre = null;
1689   internalASTExpression internalASTExpressionPre = null;
1690   internalASTKeyword keywordPost = null;
1691   internalASTExpression internalASTExpressionPost = null;
1692 }
1693 : id1:IDENTIFIER
1694 kwc:COLON
1695   internalASTOperationType = overtureOperationType
1696 id2:IDENTIFIER
1697   internalASTParameters = overtureParameters
1698 kwde:DOUBLEEQUAL
1699   internalASTOperationBody = overtureOperationBody
1700 (
1701   kwpre:PRE
1702     internalASTExpressionPre = overtureExpression
1703     {
1704       keywordPre = new InternalASTKeyword(kwpre, "Pre");
1705     }
1706   )?
1707   (
1708     kwpost:POST
1709       internalASTExpressionPost = overtureExpression
1710       {
1711         keywordPost = new InternalASTKeyword(kwpost, "Post");
1712       }
1713   )?
1714   {
1715     InternalASTIdentifier internalASTIdentifier1 = new InternalASTIdentifier(id1);
1716     InternalASTKeyword keywordColon = new InternalASTKeyword(kwc, "Colon");
1717     InternalASTIdentifier internalASTIdentifier2 = new InternalASTIdentifier(id2);
1718     InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde, "DoubleEqual");
1719     internalASTExplicitOperationDefinition = new InternalASTExplicitOperationDefinition(
      internalASTIdentifier1, keywordColon, internalASTOperationType, internalASTIdentifier2,
      internalASTParameters, keywordDoubleEqual, internalASTOperationBody, keywordPre,
      internalASTExpressionPre, keywordPost, internalASTExpressionPost);
1720   }
1721 ;
1722
1723 //implicit operation definition = identifier, parameter types,
1724 // [ identifier type pair list ],
1725 // implicit operation body;
1726 overtureImplicitOperationDefinition returns [InternalASTImplicitOperationDefinition
      internalASTImplicitOperationDefinition]
1727 {
1728   internalASTImplicitOperationDefinition = null;
1729   internalASTParameterTypes internalASTParameterTypes = null;
1730   internalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
1731   internalASTImplicitOperationBody internalASTImplicitOperationBody = null;
1732 }
1733 : id:IDENTIFIER
1734 internalASTParameterTypes = overtureParameterTypes
1735 (
1736   internalASTIdentifierTypePairList = overtureIdentifierTypePairList
1737 )?
1738 internalASTImplicitOperationBody = overtureImplicitOperationBody
1739 {
1740   InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
1741   internalASTImplicitOperationDefinition = new InternalASTImplicitOperationDefinition(
      internalASTIdentifier, internalASTParameterTypes, internalASTIdentifierTypePairList,
      internalASTImplicitOperationBody);
1742 }
1743 ;
1744
1745 //implicit operation body = [ externals ],
1746 // [ 'pre', expression ],
1747 // 'post', expression,
1748 // [ exceptions ] ;
1749 overtureImplicitOperationBody returns [InternalASTImplicitOperationBody
      internalASTImplicitOperationBody]
1750 {
1751   internalASTImplicitOperationBody = null;
1752   InternalASTExternals internalASTExternals = null;
1753

```

I.1. ANTLR GRAMMAR FILE

```

1754 InternalASTKeyword keywordPre = null;
1755 InternalASTExpression internalASTExpressionPre = null;
1756 InternalASTKeyword keywordPost = null;
1757 InternalASTExpression internalASTExpressionPost = null;
1758 InternalASTExceptions internalASTExceptions = null;
1759 }
1760 :
1761 (
1762     internalASTExternals = overtureExternals
1763 )?
1764 (
1765     kwpre:PRE
1766     internalASTExpressionPre = overtureExpression
1767     {
1768         keywordPre = new InternalASTKeyword(kwpre, "Pre");
1769     }
1770 )?
1771 kwpost:POST
1772     internalASTExpressionPost = overtureExpression
1773     (
1774         internalASTExceptions = overtureExceptions
1775     )?
1776     {
1777         keywordPost = new InternalASTKeyword(kwpost, "Post");
1778         internalASTImplicitOperationBody = new InternalASTImplicitOperationBody(
1779             internalASTExternals, keywordPre, internalASTExpressionPre, keywordPost,
1780             internalASTExpressionPost, internalASTExceptions);
1781     }
1782 ;
1783 //extended explicit operation definition = identifier , parameter types ,
1784 //    [ identifier type pair list ],
1785 //    '=' , operation body ,
1786 //    [ externals ],
1787 //    ['pre ' , expression ],
1788 //    ['post ' , expression ],
1789 //    [ exceptions ] ;
1790 overtureExtendedExplicitOperationDefinition returns [
1791     InternalASTExtendedExplicitOperationDefinition
1792     internalASTExtendedExplicitOperationDefinition ]
1793 {
1794     internalASTExtendedExplicitOperationDefinition = null;
1795     InternalASTParameterTypes internalASTParameterTypes = null;
1796     InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
1797     InternalASTOperationBody internalASTOperationBody = null;
1798     InternalASTExternals externals = null;
1799     InternalASTKeyword keywordPre = null;
1800     InternalASTExpression preExpression = null;
1801     InternalASTKeyword keywordPost = null;
1802     InternalASTExpression postExpression = null;
1803     InternalASTExceptions exceptions = null;
1804 }
1805 : id:IDENTIFIER
1806 internalASTParameterTypes = overtureParameterTypes
1807 (
1808     internalASTIdentifierTypePairList = overtureIdentifierTypePairList
1809 )?
1810 kwde:DOUBLEEQUAL
1811 internalASTOperationBody = overtureOperationBody
1812 (
1813     externals = overtureExternals
1814 )?
1815 (
1816     pr:PRE
1817     preExpression = overtureExpression
1818     {
1819         keywordPre = new InternalASTKeyword(pr, "Pre");
1820     }
1821 )?
1822 (
1823     po:POST
1824     postExpression = overtureExpression
1825     {
1826         keywordPost = new InternalASTKeyword(po, "Post");
1827     }
1828 )?
1829 (
1830     exceptions = overtureExceptions
1831 )?
1832 {
1833     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
1834     InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde, "DoubleEqual");
1835     internalASTExtendedExplicitOperationDefinition = new
1836         InternalASTExtendedExplicitOperationDefinition(internalASTIdentifier ,

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

        internalASTParameterTypes, internalASTIdentifierTypePairList, keywordDoubleEqual,
        internalASTOperationBody, externals, keywordPre, preExpression, keywordPost,
        postExpression, exceptions);
1833     }
1834 ;
1835
1836 //operation type = discretionary type, '==>', discretionary type:
1837 overtureOperationType returns [InternalASTOperationType internalASTOperationType]
1838 {
1839     internalASTOperationType = null;
1840     InternalASTDiscretionaryType internalASTDiscretionaryType1 = null;
1841     InternalASTDiscretionaryType internalASTDiscretionaryType2 = null;
1842 }
1843 :
1844     internalASTDiscretionaryType1 = overtureDiscretionaryType
1845     kwei:EQUALIMPLY
1846     internalASTDiscretionaryType2 = overtureDiscretionaryType
1847 {
1848     InternalASTKeyword keyword = new InternalASTKeyword(kwei, "EqualImPLY");
1849     internalASTOperationType = new InternalASTOperationType(internalASTDiscretionaryType1,
        keyword, internalASTDiscretionaryType2);
1850 }
1851 ;
1852
1853 // operation body = statement
1854 //      | 'Is not yet specified';
1855 //      | 'is subclass responsibility'
1856 overtureOperationBody returns [InternalASTOperationBody internalASTOperationBody]
1857 {
1858     internalASTOperationBody = null;
1859 }
1860 :
1861     (
1862     internalASTOperationBody = overtureStatement
1863     |
1864     (
1865         is: IS
1866         no: NOT
1867         ye: YET
1868         sp: SPECIFIED
1869         {
1870             InternalASTKeyword keywordIs = new InternalASTKeyword(is, "Is");
1871             InternalASTKeyword keywordNot = new InternalASTKeyword(no, "Not");
1872             InternalASTKeyword keywordYet = new InternalASTKeyword(ye, "Yet");
1873             InternalASTKeyword keywordSpecified = new InternalASTKeyword(sp, "Specified");
1874             internalASTOperationBody = new InternalASTOperationBodyIsNotYetSpecified(keywordIs,
        keywordNot, keywordYet, keywordSpecified);
1875         }
1876     )
1877     |
1878     (
1879         is2: IS
1880         su: SUBCLASS
1881         re: RESPONSIBILITY
1882         {
1883             InternalASTKeyword keywordIs = new InternalASTKeyword(is2, "Is");
1884             InternalASTKeyword keywordSubclass = new InternalASTKeyword(su, "Subclass");
1885             InternalASTKeyword keywordResponsibility = new InternalASTKeyword(re, "Responsibility");
1886             internalASTOperationBody = new InternalASTOperationBodyIsSubclassResponsibility(keywordIs,
        keywordSubclass, keywordResponsibility);
1887         }
1888     )
1889     )
1890 ;
1891
1892 //externals = 'ext', var information, {var information};
1893 overtureExternals returns [InternalASTExternals internalASTExternals]
1894 {
1895     internalASTExternals = null;
1896     List<InternalASTVarInformation> varInformations = new ArrayList<InternalASTVarInformation>();
1897
1898     InternalASTVarInformation internalASTVarInformation = null;
1899 }
1900 : kwext: EXT
1901     internalASTVarInformation = overtureVarInformation
1902     {
1903         varInformations.add(internalASTVarInformation);
1904     }
1905 (
1906     internalASTVarInformation = overtureVarInformation
1907     {
1908         varInformations.add(internalASTVarInformation);
1909     }
1910 )*)

```

I.1. ANTLR GRAMMAR FILE

```
1911     {
1912         InternalASTKeyword keywordExt = new InternalASTKeyword(kwext, "Ext");
1913         internalASTExternals = new InternalASTExternals(keywordExt, varInformations);
1914     }
1915 ;
1916
1917 //var information = mode, name list, [ ':', type ];
1918 overtureVarInformation returns [InternalASTVarInformation internalASTVarInformation]
1919 {
1920     internalASTVarInformation = null;
1921     InternalASTMode internalASTMode = null;
1922     InternalASTNameList internalASTNameList = null;
1923     InternalASTType internalASTType = null;
1924     InternalASTKeyword keywordColon = null;
1925 }
1926 :
1927     internalASTMode = overtureMode
1928     internalASTNameList = overtureNameList
1929
1930     (
1931         kwc:COLON
1932         internalASTType = overtureType
1933         {
1934             keywordColon = new InternalASTKeyword(kwc, "Colon");
1935         }
1936     )?
1937
1938     {
1939         internalASTVarInformation = new InternalASTVarInformation(internalASTMode,
1940             internalASTNameList, keywordColon, internalASTType);
1941     }
1942 ;
1943 //mode = 'rd' | 'wr';
1944 overtureMode returns [InternalASTMode internalASTMode]
1945 {
1946     internalASTMode = null;
1947     InternalASTModeRd internalASTModeRd = null;
1948     InternalASTModeWr internalASTModeWr = null;
1949 }
1950 :
1951     ( rd:RD
1952         {
1953             InternalASTKeyword keywordRd = new InternalASTKeyword(rd, "Rd");
1954             internalASTMode = new InternalASTModeRd(keywordRd);
1955         }
1956     | wr:WR
1957         {
1958             InternalASTKeyword keywordWr = new InternalASTKeyword(wr, "Wr");
1959             internalASTMode = new InternalASTModeWr(keywordWr);
1960         }
1961     )
1962 ;
1963
1964 //exceptions = 'errs', error list ;
1965 overtureExceptions returns [InternalASTExceptions internalASTExceptions]
1966 {
1967     internalASTExceptions = null;
1968     InternalASTErrorList errorList = null;
1969 }
1970 :
1971     errs:ERRS
1972     errorList = overtureErrorList
1973     {
1974         InternalASTKeyword keywordErrs = new InternalASTKeyword(errs, "Errs");
1975         internalASTExceptions = new InternalASTExceptions(keywordErrs, errorList);
1976     }
1977 ;
1978
1979 //error list = error, {error} ;
1980 overtureErrorList returns [InternalASTErrorList internalASTErrorList]
1981 {
1982     internalASTErrorList = null;
1983     List<InternalASTError> errors = new ArrayList<InternalASTError>();
1984     InternalASTError error = null;
1985 }
1986 :
1987     (
1988         error = overtureError
1989         {
1990             errors.add(error);
1991         }
1992     )
1993     error = overtureError
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

1994     {
1995         errors.add(error);
1996     }
1997     )*
1998     )
1999     {
2000     internalASTErrorList = new InternalASTErrorList(errors);
2001     }
2002     ;
2003
2004     //error = identifier , ':' , expression , '->' , expression ;
2005     overtуреError returns [InternalASTError internalASTError]
2006     {
2007     internalASTError = null;
2008     InternalASTExpression expression1 = null;
2009     InternalASTExpression expression2 = null;
2010     }
2011     :
2012     id:IDENTIFIER
2013     co:COLON
2014     expression1 = overtуреExpression
2015     la:LINEARROW
2016     expression2 = overtуреExpression
2017     {
2018     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
2019     InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
2020     InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
2021     internalASTError = new InternalASTError(identifier, keywordColon, expression1,
2022         keywordLineArrow, expression2);
2023     }
2024     ;
2025
2026     // Instance Variable Definitions
2027     //instance variable definitions = 'instance', 'variables',
2028     //                               [ instance variable definition ,
2029     //                               { ';' , instance variable definition } ];
2030     overtуреInstanceVariableDefinitions returns [InternalASTInstanceVariableDefinitions
2031         internalASTInstanceVariableDefinitions]
2032     {
2033     internalASTInstanceVariableDefinitions = null;
2034     List<InternalASTInstanceVariableDefinitionsElement> elementList = new ArrayList<
2035         InternalASTInstanceVariableDefinitionsElement>();
2036     InternalASTInstanceVariableDefinition iVDefinition = null;
2037     InternalASTInstanceVariableDefinitionsElement currentElement = null;
2038     }
2039     :
2040     ( in:INSTANCE
2041     va:VARIABLES
2042     (
2043     iVDefinition = overtуреInstanceVariableDefinition
2044     {
2045     currentElement = new InternalASTInstanceVariableDefinitionsElement(iVDefinition);
2046     elementList.add(currentElement);
2047     }
2048     )
2049     (
2050     se:SEMICOLON
2051     iVDefinition = overtуреInstanceVariableDefinition
2052     {
2053     InternalASTKeyword keywordSemicolon = new InternalASTKeyword(se, "Semicolon");
2054     currentElement = new InternalASTInstanceVariableDefinitionsElement(keywordSemicolon,
2055         iVDefinition);
2056     elementList.add(currentElement);
2057     }
2058     )? )
2059     {
2060     InternalASTKeyword keywordInstance = new InternalASTKeyword(in, "Instance");
2061     InternalASTKeyword keywordVariables = new InternalASTKeyword(va, "Variables");
2062     internalASTInstanceVariableDefinitions = new InternalASTInstanceVariableDefinitions(
2063         keywordInstance, keywordVariables, elementList);
2064     }
2065     ;
2066
2067     // instance variable definition = access assignment definition
2068     //                               | invariant definition
2069     //                               | init definition;
2070     overtуреInstanceVariableDefinition returns [InternalASTInstanceVariableDefinition
2071         internalASTInstanceVariableDefinition]
2072     {
2073     internalASTInstanceVariableDefinition = null;
2074     }
2075     :
2076     internalASTInstanceVariableDefinition = overtуреAccessAssignmentDefinition

```

I.1. ANTLR GRAMMAR FILE

```
2072 | internalASTInstanceVariableDefinition = overtureInvariantDefinition
2073 | internalASTInstanceVariableDefinition = overtureInitStatement
2074 ;
2075
2076 // access assignment definition = [ access ], assignment definition ;
2077 overtureAccessAssignmentDefinition returns [InternalASTAccessAssignmentDefinition
    internalASTAccessAssignmentDefinition]
2078 {
2079     internalASTAccessAssignmentDefinition = null;
2080     InternalASTAccess access = null;
2081     InternalASTAssignmentDefinition assignmentDefinition = null;
2082 }
2083 :
2084 (
2085     access = overtureAccess
2086 )?
2087 assignmentDefinition = overtureAssignmentDefinition
2088 {
2089     internalASTAccessAssignmentDefinition = new InternalASTAccessAssignmentDefinition(access ,
        assignmentDefinition);
2090 }
2091 ;
2092
2093 // invariant definition = 'inv', expression ;
2094 overtureInvariantDefinition returns [InternalASTInvariantDefinition
    internalASTInvariantDefinition]
2095 {
2096     internalASTInvariantDefinition = null;
2097     InternalASTExpression expression = null;
2098 }
2099 :
2100 in:INV
2101 expression = overtureExpression
2102 {
2103     InternalASTKeyword keywordInv = new InternalASTKeyword(in , "Inv");
2104     internalASTInvariantDefinition = new InternalASTInvariantDefinition(keywordInv , expression);
2105 }
2106 ;
2107
2108 // init statement = 'init', statement ;
2109 overtureInitStatement returns [InternalASTInitStatement internalASTInitStatement]
2110 {
2111     internalASTInitStatement = null;
2112     InternalASTStatement statement = null;
2113 }
2114 :
2115 in:INIT
2116 statement = overtureStatement
2117 {
2118     InternalASTKeyword keywordInit = new InternalASTKeyword(in , "Init");
2119     internalASTInitStatement = new InternalASTInitStatement(keywordInit , statement);
2120 }
2121 ;
2122
2123 // SYNCHRONIZATION DEFINITIONS
2124
2125 //synchronization definitions = 'sync', [ synchronization ];
2126 overtureSynchronizationDefinitions returns [InternalASTSynchronizationDefinitions
    internalASTSynchronizationDefinitions]
2127 {
2128     internalASTSynchronizationDefinitions = null;
2129     InternalASTSynchronization synchronization = null;
2130 }
2131 :
2132 (
2133     sy:SYNC
2134     (
2135         synchronization = overtureSynchronization
2136     )?
2137 )
2138 {
2139     InternalASTKeyword keywordSync = new InternalASTKeyword(sy , "Sync");
2140     internalASTSynchronizationDefinitions = new InternalASTSynchronizationDefinitions(keywordSync ,
        synchronization);
2141 }
2142 ;
2143
2144 //synchronization = permission predicates;
2145 overtureSynchronization returns [InternalASTSynchronization internalASTSynchronization]
2146 {
2147     internalASTSynchronization = null;
2148     InternalASTPermissionPredicates permPredicates = null;
2149 }
2150 :
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2151     permPredicates = overturePermissionPredicates
2152     {
2153         internalASTSynchronization = new InternalASTSynchronization(permPredicates);
2154     }
2155     ;
2156
2157     //permission predicates = permission predicate
2158     //      { ';', permission predicate };
2159     overturePermissionPredicates returns [InternalASTPermissionPredicates
        internalASTPermissionPredicates]
2160     {
2161         internalASTPermissionPredicates = null;
2162         List<InternalASTPermissionPredicatesElement> permissionPredicatesElements = new ArrayList<
            InternalASTPermissionPredicatesElement>();
2163         InternalASTPermissionPredicatesElement currentElement = null;
2164         InternalASTPermissionPredicate currentPredicate = null;
2165     }
2166     :
2167     currentPredicate = overturePermissionPredicate
2168     {
2169         currentElement = new InternalASTPermissionPredicatesElement(currentPredicate);
2170         permissionPredicatesElements.add(currentElement);
2171     }
2172     (
2173     se:SEMICOLON
2174     currentPredicate=overturePermissionPredicate
2175     {
2176         InternalASTKeyword keywordSemicolon = new InternalASTKeyword(se, "Semicolon");
2177         currentElement = new InternalASTPermissionPredicatesElement(keywordSemicolon,
            currentPredicate);
2178         permissionPredicatesElements.add(currentElement);
2179     }
2180     )*
2181     {
2182         internalASTPermissionPredicates = new InternalASTPermissionPredicates(
            permissionPredicatesElements);
2183     }
2184     ;
2185
2186     //permission predicate = 'per', name '>' expression
2187     //      | mutex predicate ;
2188     overturePermissionPredicate returns [InternalASTPermissionPredicate
        internalASTPermissionPredicate]
2189     {
2190         internalASTPermissionPredicate = null;
2191         InternalASTName name = null;
2192         InternalASTExpression expression = null;
2193         InternalASTMutexPredicate mutexpredicate = null;
2194     }
2195     :
2196     (
2197     (
2198         pe:PER
2199         name = overtureName
2200         im:IMPLY
2201         expression = overtureExpression
2202     )
2203     {
2204         InternalASTKeyword keywordPer = new InternalASTKeyword(pe, "Per");
2205         InternalASTKeyword keywordImPLY = new InternalASTKeyword(im, "ImPLY");
2206         internalASTPermissionPredicate = new InternalASTPermissionPredicateElement1(keywordPer, name
            , keywordImPLY, expression);
2207     }
2208     )
2209     |
2210     (
2211     (
2212         mutexpredicate = overtureMutexPredicate
2213     )
2214     {
2215         internalASTPermissionPredicate = new InternalASTPermissionPredicateElement2(mutexpredicate);
2216     }
2217     )
2218     ;
2219
2220     //mutex predicate = 'mutex', '(' , 'all', ')'
2221     //      | 'mutex', '(' , name list , ')'
2222     overtureMutexPredicate returns [InternalASTMutexPredicate internalASTMutexPredicate]
2223     {
2224         internalASTMutexPredicate = null;
2225         InternalASTNameList namelist = null;
2226     }
2227     :
2228     (MUTEX LBRACKET ALL RBRACKET)

```


I.1. ANTLR GRAMMAR FILE

```

2229 =>((
2230   mu:MUTEX
2231   lb:LBRACKET
2232   al:ALL
2233   rb:RBRACKET
2234 )
2235 {
2236   InternalASTKeyword keywordMutex = new InternalASTKeyword(mu, "Mutex");
2237   InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
2238   InternalASTKeyword keywordAll = new InternalASTKeyword(al, "All");
2239   InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
2240   internalASTMutexPredicate = new InternalASTMutexPredicateAll(keywordMutex,
2241     keywordLeftBracket, keywordAll, keywordRightBracket);
2242 }
2243 )
2244 (
2245 (
2246   mu2:MUTEX
2247   lb2:LBRACKET
2248   namelist = overtureNameList
2249   rb2:RBRACKET
2250 )
2251 {
2252   InternalASTKeyword keywordMutex = new InternalASTKeyword(mu2, "Mutex");
2253   InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
2254   InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
2255   internalASTMutexPredicate = new InternalASTMutexPredicateNameList(keywordMutex,
2256     keywordLeftBracket, namelist, keywordRightBracket);
2257 }
2258 );
2259
2260 // THREAD DEFINITIONS
2261
2262 //thread definitions = 'thread', [thread definition];
2263 overtureThreadDefinitions returns [InternalASTThreadDefinitions internalASTThreadDefinitions]
2264 {
2265   internalASTThreadDefinitions = null;
2266   InternalASTThreadDefinition threadDefinition = null;
2267 }
2268 :
2269 th:THREAD
2270 ( threadDefinition = overtureThreadDefinition )?
2271 {
2272   InternalASTKeyword keywordThread = new InternalASTKeyword(th, "Thread");
2273   internalASTThreadDefinitions = new InternalASTThreadDefinitions(keywordThread,
2274     threadDefinition);
2275 }
2276 ;
2277
2278 //thread definition = procedural thread definition
2279 overtureThreadDefinition returns [InternalASTThreadDefinition internalASTThreadDefinition]
2280 {
2281   internalASTThreadDefinition = null;
2282   InternalASTProceduralThreadDefinition proceduralThreadDefinition = null;
2283 }
2284 :
2285 proceduralThreadDefinition = overtureProceduralThreadDefinition
2286 {
2287   internalASTThreadDefinition = new InternalASTThreadDefinition(proceduralThreadDefinition);
2288 }
2289 ;
2290 //proceduralthread definition = statement
2291 overtureProceduralThreadDefinition returns [InternalASTProceduralThreadDefinition
2292   internalASTProceduralThreadDefinition]
2293 {
2294   internalASTProceduralThreadDefinition = null;
2295   InternalASTStatement statement = null;
2296 }
2297 :
2298 statement = overtureStatement
2299 {
2300   internalASTProceduralThreadDefinition = new InternalASTProceduralThreadDefinition(statement);
2301 }
2302 ;
2303 // EXPRESSIONS
2304
2305 // expression list = expression , { ',', expression };
2306 overtureExpressionList returns [InternalASTExpressionList internalASTExpressionList]
2307 {
2308   internalASTExpressionList = null;

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2309 | List<InternalASTExpressionListElement> expressionListElements = new ArrayList<
      |     InternalASTExpressionListElement>();
2310 | InternalASTExpressionListElement currentElement = null;
2311 | InternalASTExpression expression = null;
2312 | }
2313 | :
2314 | (
2315 |     expression = overtуреExpression
2316 |     {
2317 |         currentElement = new InternalASTExpressionListElement(expression);
2318 |         expressionListElements.add(currentElement);
2319 |     }
2320 | (options {greedy=true};):
2321 | co:COMMA
2322 |     expression = overtуреExpression
2323 |     {
2324 |         InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
2325 |         currentElement = new InternalASTExpressionListElement(keywordComma, expression);
2326 |         expressionListElements.add(currentElement);
2327 |     }
2328 | )*)
2329 | )
2330 | {
2331 |     internalASTExpressionList = new InternalASTExpressionList(expressionListElements);
2332 | }
2333 | ;
2334 |
2335 | // expression = ...
2336 | overtуреExpression returns [InternalASTExpression internalASTExpression]
2337 | {
2338 |     internalASTExpression = null;
2339 | }
2340 | :
2341 |     internalASTExpression = overtуреExpressionPre00
2342 | ;
2343 |
2344 | overtуреExpressionPre00 returns [InternalASTExpression internalASTExpression]
2345 | {
2346 |     internalASTExpression = null;
2347 | }
2348 | :
2349 |     (overtуреQuantifiedExpression)=> internalASTExpression = overtуреQuantifiedExpression
2350 |     | (overtуреIotaExpression)=> internalASTExpression = overtуреIotaExpression
2351 |     | (overtуреSetEnumeration)=> internalASTExpression = overtуреSetEnumeration
2352 |     | (overtуреSetComprehension)=> internalASTExpression = overtуреSetComprehension
2353 |     | (overtуреSetRangeExpression)=> internalASTExpression = overtуреSetRangeExpression
2354 |     | (overtуреSequenceEnumeration)=> internalASTExpression = overtуреSequenceEnumeration
2355 |     | (overtуреSequenceComprehension)=> internalASTExpression = overtуреSequenceComprehension
2356 |     | (overtуреMapEnumeration)=> internalASTExpression = overtуреMapEnumeration
2357 |     | (overtуреMapComprehension)=> internalASTExpression = overtуреMapComprehension
2358 |     | (overtуреTupleConstructor)=> internalASTExpression = overtуреTupleConstructor
2359 |     | (overtуреRecordConstructor)=> internalASTExpression = overtуреRecordConstructor
2360 |     | (overtуреRecordModifier)=> internalASTExpression = overtуреRecordModifier
2361 |     | (overtуреTupleSelect)=> internalASTExpression = overtуреTupleSelect
2362 |     | internalASTExpression = overtуреLambdaExpression
2363 |     | internalASTExpression = overtуреNewExpression
2364 |     | internalASTExpression = overtуреGeneralIsExpression
2365 |     | internalASTExpression = overtуреIsofbaseclassExpression
2366 |     | internalASTExpression = overtуреIsofclassExpression
2367 |     | internalASTExpression = overtуреSamebaseclassExpression
2368 |     | internalASTExpression = overtуреSameclassExpression
2369 |     | internalASTExpression = overtуреActExpression
2370 |     | internalASTExpression = overtуреFinExpression
2371 |     | internalASTExpression = overtуреActiveExpression
2372 |     | internalASTExpression = overtуреReqExpression
2373 |     | internalASTExpression = overtуреWaitingExpression
2374 |     | internalASTExpression = overtуреExpressionPre21
2375 | ;
2376 |
2377 | // Optional grouping... - right grouping chosen!
2378 | overtуреExpressionPre21 returns [InternalASTExpression internalASTExpression]
2379 | {
2380 |     internalASTExpression = null;
2381 |     InternalASTExpression expression1 = null;
2382 |     InternalASTBinaryOperator operator = null;
2383 |     InternalASTExpression expression2 = null;
2384 | }
2385 | :
2386 |     (overtуреExpressionPre22 overtуреBinaryOperatorLogicalEquivalence overtуреExpressionPre21) =>
2387 |     (
2388 |         expression1 = overtуреExpressionPre22
2389 |         operator = overtуреBinaryOperatorLogicalEquivalence
2390 |         expression2 = overtуреExpressionPre21
2391 |     {

```

I.1. ANTLR GRAMMAR FILE

```

2392         internalASTExpression = new InternalASTBinaryExpression(expression1 , operator , expression2)
2393     ;
2394     )
2395 |
2396     internalASTExpression = overtуреExpressionPre22
2397 ;
2398
2399 overtуреExpressionPre22 returns [InternalASTExpression internalASTExpression]
2400 {
2401     internalASTExpression = null;
2402     InternalASTExpression expression1 = null;
2403     InternalASTBinaryOperator operator = null;
2404     InternalASTExpression expression2 = null;
2405 }
2406 :
2407     (overtуреExpressionPre23 overtуреBinaryOperatorImPLY overtуреExpressionPre22) =>
2408     (
2409         expression1 = overtуреExpressionPre23
2410         operator = overtуреBinaryOperatorImPLY
2411         expression2 = overtуреExpressionPre22
2412         {
2413             internalASTExpression = new InternalASTBinaryExpression(expression1 , operator , expression2)
2414         ;
2415         }
2416     )
2417 |
2418     internalASTExpression = overtуреExpressionPre23
2419 ;
2420 overtуреExpressionPre23 returns [InternalASTExpression internalASTExpression]
2421 {
2422     internalASTExpression = null;
2423     InternalASTExpression expression1 = null;
2424     InternalASTBinaryOperator operator = null;
2425     InternalASTExpression expression2 = null;
2426 }
2427 :
2428     (overtуреExpressionPre24 overtуреBinaryOperatorOr overtуреExpressionPre23) =>
2429     (
2430         expression1 = overtуреExpressionPre24
2431         operator = overtуреBinaryOperatorOr
2432         expression2 = overtуреExpressionPre23
2433         {
2434             internalASTExpression = new InternalASTBinaryExpression(expression1 , operator , expression2)
2435         ;
2436         }
2437     )
2438 |
2439     internalASTExpression = overtуреExpressionPre24
2440 ;
2441 overtуреExpressionPre24 returns [InternalASTExpression internalASTExpression]
2442 {
2443     internalASTExpression = null;
2444     InternalASTExpression expression1 = null;
2445     InternalASTBinaryOperator operator = null;
2446     InternalASTExpression expression2 = null;
2447 }
2448 :
2449     (overtуреExpressionPre25 overtуреBinaryOperatorAnd overtуреExpressionPre24) =>
2450     (
2451         expression1 = overtуреExpressionPre25
2452         operator = overtуреBinaryOperatorAnd
2453         expression2 = overtуреExpressionPre24
2454         {
2455             internalASTExpression = new InternalASTBinaryExpression(expression1 , operator , expression2)
2456         ;
2457         }
2458     )
2459 |
2460     internalASTExpression = overtуреExpressionPre25
2461 ;
2462 overtуреExpressionPre25 returns [InternalASTExpression internalASTExpression]
2463 {
2464     internalASTExpression = null;
2465     InternalASTUnaryOperator operator = null;
2466     InternalASTExpression expression = null;
2467 }
2468 :
2469     (
2470         operator = overtуреUnaryOperatorNot
2471         expression = overtуреExpressionPre31

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2472     {
2473         internalASTExpression = new InternalASTPrefixExpression(operator, expression);
2474     }
2475     }
2476     |
2477     internalASTExpression = overtуреExpressionPre31
2478     ;
2479
2480 overtуреExpressionPre31 returns [InternalASTExpression internalASTExpression]
2481 {
2482     internalASTExpression = null;
2483     InternalASTExpression expression1 = null;
2484     InternalASTBinaryOperator operator = null;
2485     InternalASTExpression expression2 = null;
2486 }
2487 :
2488     (overtуреExpressionPre41
2489     (
2490         overtуреBinaryOperatorLessThan
2491         | overtуреBinaryOperatorLessThanOrEqual
2492         | overtуреBinaryOperatorGreaterThan
2493         | overtуреBinaryOperatorGreaterThanOrEqual
2494         | overtуреBinaryOperatorEqual
2495         | overtуреBinaryOperatorApprox
2496         | overtуреBinaryOperatorNotEqual
2497         | overtуреBinaryOperatorInSet
2498         | overtуреBinaryOperatorNotInSet
2499         | overtуреBinaryOperatorSubset
2500         | overtуреBinaryOperatorProperSubset
2501     )
2502     overtуреExpressionPre41
2503     ) =>
2504     (
2505         expression1 = overtуреExpressionPre41
2506         (
2507             operator = overtуреBinaryOperatorLessThan
2508             | operator = overtуреBinaryOperatorLessThanOrEqual
2509             | operator = overtуреBinaryOperatorGreaterThan
2510             | operator = overtуреBinaryOperatorGreaterThanOrEqual
2511             | operator = overtуреBinaryOperatorEqual
2512             | operator = overtуреBinaryOperatorApprox
2513             | operator = overtуреBinaryOperatorNotEqual
2514             | operator = overtуреBinaryOperatorInSet
2515             | operator = overtуреBinaryOperatorNotInSet
2516             | operator = overtуреBinaryOperatorSubset
2517             | operator = overtуреBinaryOperatorProperSubset
2518         )
2519         expression2 = overtуреExpressionPre41
2520         {
2521             internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2)
2522         }
2523     )
2524     |
2525     internalASTExpression = overtуреExpressionPre41
2526     ;
2527
2528 overtуреExpressionPre41 returns [InternalASTExpression internalASTExpression]
2529 {
2530     internalASTExpression = null;
2531     InternalASTExpression expression1 = null;
2532     InternalASTBinaryOperator operator = null;
2533     InternalASTExpression expression2 = null;
2534 }
2535 :
2536     (overtуреExpressionPre41u
2537     (
2538         overtуреBinaryOperatorArithmeticPlus
2539         | overtуреBinaryOperatorArithmeticMinus
2540         | overtуреBinaryOperatorSetUnion
2541         | overtуреBinaryOperatorSetDifference
2542         | overtуреBinaryOperatorSequenceConcatenate
2543         | overtуреBinaryOperatorMapOrSequenceModify
2544         | overtуреBinaryOperatorMapMerge
2545     )
2546     overtуреExpressionPre41u) =>
2547     (
2548         expression1 = overtуреExpressionPre41u
2549         (
2550             operator = overtуреBinaryOperatorArithmeticPlus
2551             | operator = overtуреBinaryOperatorArithmeticMinus
2552             | operator = overtуреBinaryOperatorSetUnion
2553             | operator = overtуреBinaryOperatorSetDifference
2554             | operator = overtуреBinaryOperatorSequenceConcatenate

```

I.1. ANTLR GRAMMAR FILE

```

2555     | operator = overtureBinaryOperatorMapOrSequenceModify
2556     | operator = overtureBinaryOperatorMapMerge
2557   )
2558   expression2 = overtureExpressionPre41u
2559   {
2560     expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
2561   }
2562
2563   (options {greedy=true};
2564   (
2565     operator = overtureBinaryOperatorArithmeticPlus
2566     | operator = overtureBinaryOperatorArithmeticMinus
2567     | operator = overtureBinaryOperatorSetUnion
2568     | operator = overtureBinaryOperatorSetDifference
2569     | operator = overtureBinaryOperatorSequenceConcatenate
2570     | operator = overtureBinaryOperatorMapOrSequenceModify
2571     | operator = overtureBinaryOperatorMapMerge
2572   )
2573   expression2 = overtureExpressionPre41u
2574   {
2575     expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
2576   }
2577   )*)
2578   {
2579     internalASTExpression = expression1;
2580   }
2581 )
2582
2583 |
2584 | internalASTExpression = overtureExpressionPre41u
2585 ;
2586
2587 overtureExpressionPre41u returns [InternalASTExpression internalASTExpression]
2588 {
2589   internalASTExpression = null;
2590   InternalASTUnaryOperator operator = null;
2591   InternalASTExpression expression = null;
2592 }
2593 :
2594 (
2595 (
2596   operator = overtureUnaryOperatorDistributedSetUnion
2597   | operator = overtureUnaryOperatorDistributedSetIntersection
2598 )
2599 expression = overtureExpressionPre41
2600 {
2601   internalASTExpression = new InternalASTPrefixExpression(operator, expression);
2602 }
2603 )
2604 |
2605 internalASTExpression = overtureExpressionPre42
2606 ;
2607
2608 overtureExpressionPre42 returns [InternalASTExpression internalASTExpression]
2609 {
2610   internalASTExpression = null;
2611   InternalASTExpression expression1 = null;
2612   InternalASTBinaryOperator operator = null;
2613   InternalASTExpression expression2 = null;
2614 }
2615 :
2616 (overtureExpressionPre44
2617 (
2618   overtureBinaryOperatorArithmeticMultiplication
2619   | overtureBinaryOperatorArithmeticDivide
2620   | overtureBinaryOperatorArithmeticIntegerDivision
2621   | overtureBinaryOperatorArithmeticRem
2622   | overtureBinaryOperatorArithmeticMod
2623   | overtureBinaryOperatorSetIntersection
2624 )
2625 overtureExpressionPre44) =>
2626 (
2627   expression1 = overtureExpressionPre44
2628   (
2629     operator = overtureBinaryOperatorArithmeticMultiplication
2630     | operator = overtureBinaryOperatorArithmeticDivide
2631     | operator = overtureBinaryOperatorArithmeticIntegerDivision
2632     | operator = overtureBinaryOperatorArithmeticRem
2633     | operator = overtureBinaryOperatorArithmeticMod
2634     | operator = overtureBinaryOperatorSetIntersection
2635   )
2636   expression2 = overtureExpressionPre44
2637   {
2638     expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2639     }
2640
2641     (options {greedy=true};)
2642     (
2643
2644         operator = overtureBinaryOperatorArithmeticMultiplication
2645         | operator = overtureBinaryOperatorArithmeticDivide
2646         | operator = overtureBinaryOperatorArithmeticIntegerDivision
2647         | operator = overtureBinaryOperatorArithmeticRem
2648         | operator = overtureBinaryOperatorArithmeticMod
2649         | operator = overtureBinaryOperatorSetIntersection
2650     )
2651     expression2 = overtureExpressionPre44
2652     {
2653         expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
2654     }
2655     )*
2656
2657     {
2658         internalASTExpression = expression1;
2659     }
2660     )
2661     |
2662     internalASTExpression = overtureExpressionPre44
2663     ;
2664
2665     overtureExpressionPre44 returns [InternalASTExpression internalASTExpression]
2666     {
2667         internalASTExpression = null;
2668         InternalASTExpression expression1 = null;
2669         InternalASTBinaryOperator operator = null;
2670         InternalASTExpression expression2 = null;
2671     }
2672     :
2673     (overtureExpressionPre45
2674     (
2675         overtureBinaryOperatorMapDomainRestrictTo
2676         | overtureBinaryOperatorMapDomainRestrictBy
2677     )
2678     overtureExpressionPre44) =>
2679     (
2680         expression1 = overtureExpressionPre45
2681         (
2682             operator = overtureBinaryOperatorMapDomainRestrictTo
2683             | operator = overtureBinaryOperatorMapDomainRestrictBy
2684         )
2685         expression2 = overtureExpressionPre44
2686         {
2687             internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2)
2688         }
2689         ;
2690     )
2691     |
2692     internalASTExpression = overtureExpressionPre45
2693     ;
2694
2695     overtureExpressionPre45 returns [InternalASTExpression internalASTExpression]
2696     {
2697         internalASTExpression = null;
2698         InternalASTExpression expression1 = null;
2699         InternalASTBinaryOperator operator = null;
2700         InternalASTExpression expression2 = null;
2701     }
2702     :
2703     (overtureExpressionPre46u
2704     (
2705         overtureBinaryOperatorMapRangeRestrictTo
2706         | overtureBinaryOperatorMapRangeRestrictBy
2707     )
2708     overtureExpressionPre46u) =>
2709     (
2710         expression1 = overtureExpressionPre46u
2711         (
2712             operator = overtureBinaryOperatorMapRangeRestrictTo
2713             | operator = overtureBinaryOperatorMapRangeRestrictBy
2714         )
2715         expression2 = overtureExpressionPre46u
2716         {
2717             expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
2718         }
2719     )
2720     (options {greedy=true};)
2721     (

```

I.1. ANTLR GRAMMAR FILE

```

2722         operator = overtureBinaryOperatorMapRangeRestrictTo
2723         | operator = overtureBinaryOperatorMapRangeRestrictBy
2724     )
2725     expression2 = overtureExpressionPre46u
2726     {
2727         expression1 = new InternalASTBinaryExpression(expression1 , operator , expression2);
2728     }
2729     )*
2730
2731     {
2732         internalASTExpression = expression1;
2733     }
2734     )
2735     |
2736     internalASTExpression = overtureExpressionPre46u
2737     ;
2738
2739     overtureExpressionPre46u returns [InternalASTExpression internalASTExpression]
2740     {
2741         internalASTExpression = null;
2742         InternalASTUnaryOperator operator = null;
2743         InternalASTExpression expression = null;
2744     }
2745     :
2746     (
2747     (
2748         operator = overtureUnaryOperatorUnaryPlus
2749         | operator = overtureUnaryOperatorUnaryMinus
2750         | operator = overtureUnaryOperatorArithmeticAbs
2751         | operator = overtureUnaryOperatorFloor
2752         | operator = overtureUnaryOperatorSetCardinality
2753         | operator = overtureUnaryOperatorFinitePowerSet
2754         | operator = overtureUnaryOperatorSequenceHead
2755         | operator = overtureUnaryOperatorSequenceTail
2756         | operator = overtureUnaryOperatorSequenceLength
2757         | operator = overtureUnaryOperatorSequenceElements
2758         | operator = overtureUnaryOperatorSequenceIndices
2759         | operator = overtureUnaryOperatorDistributedSequenceConcatenation
2760         | operator = overtureUnaryOperatorMapDomain
2761         | operator = overtureUnaryOperatorMapRange
2762         | operator = overtureUnaryOperatorDistributedMapMerge
2763     )
2764     expression = overtureExpressionPre46u
2765     {
2766         internalASTExpression = new InternalASTPrefixExpression(operator , expression);
2767     }
2768     )
2769     |
2770     internalASTExpression = overtureMapInverse
2771     |
2772     internalASTExpression = overtureExpressionPre50
2773     ;
2774
2775     overtureExpressionPre50 returns [InternalASTExpression internalASTExpression]
2776     {
2777         internalASTExpression = null;
2778     }
2779     :
2780     (overtureApply)=> internalASTExpression = overtureApply
2781     | (overtureSubsequence)=> internalASTExpression = overtureSubsequence
2782     | (overtureFieldSelect)=> internalASTExpression = overtureFieldSelect
2783     | (overtureFunctionTypeInstantiation)=> internalASTExpression =
2784         overtureFunctionTypeInstantiation
2785     | internalASTExpression = overtureExpressionPre61
2786     ;
2787
2788     overtureExpressionPre61 returns [InternalASTExpression internalASTExpression]
2789     {
2790         internalASTExpression = null;
2791         InternalASTExpression expression1 = null;
2792         InternalASTBinaryOperator operator = null;
2793         InternalASTExpression expression2 = null;
2794     }
2795     :
2796     (overtureExpressionPre62
2797     overtureBinaryOperatorComposition
2798     overtureExpressionPre61)=>
2799     (expression1 = overtureExpressionPre62
2800     operator = overtureBinaryOperatorComposition
2801     expression2 = overtureExpressionPre61
2802     {
2803         internalASTExpression = new InternalASTBinaryExpression(expression1 , operator , expression2);
2804     }

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2805     )
2806     | internalASTExpression = overtureExpressionPre62
2807     ;
2808
2809 overtureExpressionPre62 returns [InternalASTExpression internalASTExpression]
2810 {
2811     internalASTExpression = null;
2812     InternalASTExpression expression1 = null;
2813     InternalASTBinaryOperator operator = null;
2814     InternalASTExpression expression2 = null;
2815 }
2816 :
2817     (overtureExpressionPre70
2818     overtureBinaryOperatorIterate
2819     overtureExpressionPre62)=>
2820     (expression1 = overtureExpressionPre70
2821     operator = overtureBinaryOperatorIterate
2822     expression2 = overtureExpressionPre62
2823     {
2824         internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
2825     }
2826     )
2827     | internalASTExpression = overtureExpressionPre70
2828     ;
2829
2830 overtureExpressionPre70 returns [InternalASTExpression internalASTExpression]
2831 {
2832     internalASTExpression = null;
2833     InternalASTSymbolicLiteral symbolicLiteral = null;
2834 }
2835 :
2836     internalASTExpression = overtureBracketedExpression
2837     | (overtureLetBeExpression)=> internalASTExpression = overtureLetBeExpression
2838     | (overtureLetExpression)=> internalASTExpression = overtureLetExpression
2839     | internalASTExpression = overtureDefExpression
2840     | internalASTExpression = overtureIfExpression
2841     | internalASTExpression = overtureCasesExpression
2842     | internalASTExpression = overtureQuantifiedExpression
2843     //Names
2844     | (overtureOldName)=> internalASTExpression = overtureOldName
2845     | (overtureName)=> internalASTExpression = overtureName
2846     | symbolicLiteral = overtureSymbolicLiteral
2847     {
2848         internalASTExpression = new InternalASTExpressionSymbolicLiteral(symbolicLiteral);
2849     }
2850     | internalASTExpression = overtureSelfExpression
2851     | internalASTExpression = overtureThreadidExpression
2852     | internalASTExpression = overtureUndefinedExpression
2853     ;
2854
2855 overtureBracketedExpression returns [InternalASTBracketedExpression
2856     internalASTBracketedExpression]
2857 {
2858     internalASTBracketedExpression = null;
2859     InternalASTExpression expression = null;
2860 }
2861 :
2862     lb:LBRACKET
2863     expression = overtureExpression
2864     rb:RBRACKET
2865     {
2866         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
2867         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
2868         internalASTBracketedExpression = new InternalASTBracketedExpression(keywordLeftBracket,
2869             expression, keywordRightBracket);
2870     }
2871     ;
2872
2873 //let expression = 'let', local definition, { ',', local definition },
2874 //in', expression ;
2875 overtureLetExpression returns [InternalASTLetExpression internalASTLetExpression]
2876 {
2877     internalASTLetExpression = null;
2878     List<InternalASTLetExpressionElement> definitions = new ArrayList<
2879         InternalASTLetExpressionElement>();
2880     InternalASTExpression expression = null;
2881     InternalASTLocalDefinition localDefinition = null;
2882     InternalASTLetExpressionElement currentElement = null;
2883 }
2884 :
2885     let:LET
2886     localDefinition = overtureLocalDefinition
2887     {
2888         currentElement = new InternalASTLetExpressionElement(localDefinition);

```


I.1. ANTLR GRAMMAR FILE

```

2886     definitions.add(currentElement);
2887   }
2888   (
2889     co:COMMA
2890     localDefinition = overtureLocalDefinition
2891     {
2892       InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
2893       currentElement = new InternalASTLetExpressionElement(keywordComma, localDefinition);
2894       definitions.add(currentElement);
2895     }
2896   )*
2897   in:IN
2898   expression = overtureExpression
2899   {
2900     InternalASTKeyword keywordLet = new InternalASTKeyword(let, "Let");
2901     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
2902     internalASTLetExpression = new InternalASTLetExpression(keywordLet, definitions, keywordIn,
2903       expression);
2904   }
2905 ;
2906 overtureLetBeExpression returns [InternalASTLetBeExpression internalASTLetBeExpression]
2907 {
2908   internalASTLetBeExpression = null;
2909   List<InternalASTLetExpressionElement> definitions = new ArrayList<
2910     InternalASTLetExpressionElement>();
2911   InternalASTBind bind = null;
2912   InternalASTKeyword keywordBe = null;
2913   InternalASTKeyword keywordSt = null;
2914   InternalASTExpression expression1 = null;
2915   InternalASTExpression expression2 = null;
2916 }
2917 :
2918   let:LET
2919   bind = overtureBind
2920   (
2921     be:BE
2922     st:ST
2923     expression1 = overtureExpression
2924     {
2925       keywordBe = new InternalASTKeyword(be, "Be");
2926       keywordSt = new InternalASTKeyword(st, "St");
2927     }
2928   )?
2929   in:IN
2930   expression2 = overtureExpression
2931   {
2932     InternalASTKeyword keywordLet = new InternalASTKeyword(let, "Let");
2933     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
2934     internalASTLetBeExpression = new InternalASTLetBeExpression(keywordLet, bind, keywordBe,
2935       keywordSt, expression1, keywordIn, expression2);
2936   }
2937 ;
2938 overtureDefExpression returns [InternalASTDefExpression internalASTDefExpression]
2939 {
2940   internalASTDefExpression = null;
2941   List<InternalASTDefExpressionElement> elements = new ArrayList<InternalASTDefExpressionElement
2942     >();
2943   InternalASTPatternBind currentPatternBind = null;
2944   InternalASTExpression currentexpression = null;
2945   InternalASTDefExpressionElement currentelement = null;
2946   InternalASTExpression expression = null;
2947 }
2948 :
2949   def:DEF
2950   currentPatternBind = overturePatternBind
2951   eq1:EQUALSIGN
2952   currentexpression = overtureExpression
2953   {
2954     InternalASTKeyword keywordEqualSign = new InternalASTKeyword(eq1, "EqualSign");
2955     currentelement = new InternalASTDefExpressionElement(currentPatternBind, keywordEqualSign,
2956       currentexpression);
2957     elements.add(currentelement);
2958   }
2959 (options {greedy=true};):
2960   sc:SEMICOLON
2961   {
2962     InternalASTKeyword keywordSemicolon = new InternalASTKeyword(sc, "Semicolon");
2963     currentelement.setKeywordSemicolon(keywordSemicolon);
2964     currentelement.setEndPositionFromNode(keywordSemicolon);
2965   }

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

2965
2966
2967     currentPatternBind = overturePatternBind
2968     eq:EQUALSIGN
2969     currentExpression = overtureExpression
2970     {
2971         InternalASTKeyword keywordEqualsign = new InternalASTKeyword(eq, "Equalsign");
2972         currentElement = new InternalASTDefExpressionElement(currentPatternBind, keywordEqualsign,
                currentExpression);
2973         elements.add(currentElement);
2974     }
2975 )*
2976
2977 (
2978     selast:SEMICOLON
2979     {
2980         InternalASTKeyword keywordSemicolon = new InternalASTKeyword(selast, "Semicolon");
2981         currentElement.setKeywordSemicolon(keywordSemicolon);
2982         currentElement.setEndPositionFromNode(keywordSemicolon);
2983     }
2984 )?
2985
2986
2987 in:IN
2988 expression = overtureExpression
2989 {
2990     InternalASTKeyword keywordDef = new InternalASTKeyword(def, "Def");
2991     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
2992     internalASTDefExpression = new InternalASTDefExpression(keywordDef, elements, keywordIn,
                expression);
2993 }
2994 ;
2995
2996 overtureIfExpression returns [InternalASTIfExpression internalASTIfExpression]
2997 {
2998     internalASTIfExpression = null;
2999     InternalASTKeyword keywordIf = null;
3000     InternalASTExpression ifExpression = null;
3001     InternalASTKeyword keywordThen = null;
3002     InternalASTExpression thenExpression = null;
3003     List<InternalASTElseifExpression> elseifExpressions = new ArrayList<InternalASTElseifExpression
                >();
3004     InternalASTElseifExpression elseifExpression = null;
3005     InternalASTKeyword keywordElse = null;
3006     InternalASTExpression elseExpression = null;
3007 }
3008 :
3009     kwif:IF
3010     ifExpression = overtureExpression
3011
3012     kwthen:THEN
3013     thenExpression = overtureExpression
3014
3015     (
3016         elseifExpression = overtureElseifExpression
3017         {
3018             elseifExpressions.add(elseifExpression);
3019         }
3020     )*
3021
3022     kwelse:ELSE
3023     elseExpression = overtureExpression
3024     {
3025         keywordIf = new InternalASTKeyword(kwif, "If");
3026         keywordThen = new InternalASTKeyword(kwthen, "Then");
3027         keywordElse = new InternalASTKeyword(kwelse, "Else");
3028         internalASTIfExpression = new InternalASTIfExpression(keywordIf, ifExpression, keywordThen,
                thenExpression, elseifExpressions, keywordElse, elseExpression);
3029     }
3030 ;
3031
3032 overtureElseifExpression returns [InternalASTElseifExpression internalASTElseifExpression]
3033 {
3034     internalASTElseifExpression = null;
3035     InternalASTExpression expression1 = null;
3036     InternalASTExpression expression2 = null;
3037 }
3038 :
3039     elseif:ELSEIF
3040     expression1 = overtureExpression
3041     then:THEN
3042     expression2 = overtureExpression
3043     {
3044         InternalASTKeyword keywordElseif = new InternalASTKeyword(elseif, "Elseif");

```

I.1. ANTLR GRAMMAR FILE

```

3045     InternalASTKeyword keywordThen = new InternalASTKeyword(then, "Then");
3046     internalASTElseifExpression = new InternalASTElseifExpression(keywordElseif, expression1,
        keywordThen, expression2);
3047     }
3048 ;
3049
3050 overtureCasesExpression returns [InternalASTCasesExpression internalASTCasesExpression]
3051 {
3052     internalASTCasesExpression = null;
3053     InternalASTExpression expression = null;
3054     InternalASTCasesExpressionAlternatives casealternatives = null;
3055     InternalASTKeyword keywordComma = null;
3056     InternalASTOthersExpression othersExpression = null;
3057 }
3058 :
3059     ca:CASES
3060
3061     expression = overtureExpression
3062
3063     co:COLON
3064
3065     casealternatives = overtureCasesExpressionAlternatives
3066
3067     (
3068     com:COMMA
3069     othersExpression = overtureOthersExpression
3070     {
3071         keywordComma = new InternalASTKeyword(com, "Comma");
3072     }
3073     )?
3074
3075     en:END
3076     {
3077         InternalASTKeyword keywordCases = new InternalASTKeyword(ca, "Cases");
3078         InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
3079         InternalASTKeyword keywordEnd = new InternalASTKeyword(en, "End");
3080         internalASTCasesExpression = new InternalASTCasesExpression(keywordCases, expression,
            keywordColon, casealternatives, keywordComma, othersExpression, keywordEnd);
3081     }
3082 ;
3083
3084 overtureCasesExpressionAlternatives returns [InternalASTCasesExpressionAlternatives
    internalASTCasesExpressionAlternatives]
3085 {
3086     internalASTCasesExpressionAlternatives = null;
3087     List<InternalASTCasesExpressionAlternativesElement> alternatives = new ArrayList<
        InternalASTCasesExpressionAlternativesElement>();
3088     InternalASTCasesExpressionAlternativesElement element = null;
3089     InternalASTCasesExpressionAlternative alternative = null;
3090 }
3091 :
3092     alternative = overtureCasesExpressionAlternative
3093     {
3094         element = new InternalASTCasesExpressionAlternativesElement(alternative);
3095         alternatives.add(element);
3096     }
3097
3098     (options {greedy=true};
3099     co:COMMA
3100     alternative = overtureCasesExpressionAlternative
3101     {
3102         InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
3103         element = new InternalASTCasesExpressionAlternativesElement(keywordComma, alternative);
3104         alternatives.add(element);
3105     }
3106     )*
3107
3108     {
3109         internalASTCasesExpressionAlternatives = new InternalASTCasesExpressionAlternatives(
            alternatives);
3110     }
3111 ;
3112
3113 overtureCasesExpressionAlternative returns [InternalASTCasesExpressionAlternative
    internalASTCasesExpressionAlternative]
3114 {
3115     internalASTCasesExpressionAlternative = null;
3116     InternalASTPatternList patternlist = null;
3117     InternalASTExpression expression = null;
3118 }
3119 :
3120     patternlist = overturePatternList
3121     la:LINEARROW
3122     expression = overtureExpression

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

3123     {
3124         InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
3125         internalASTCasesExpressionAlternative = new InternalASTCasesExpressionAlternative(patternlist
           , keywordLineArrow, expression);
3126     }
3127 ;
3128
3129 overtureOthersExpression returns [InternalASTOthersExpression internalASTOthersExpression]
3130 {
3131     internalASTOthersExpression = null;
3132     InternalASTExpression expression = null;
3133 }
3134 :
3135 ot:OTHERS
3136 la:LINEARROW
3137 expression = overtureExpression
3138 {
3139     InternalASTKeyword keywordOthers = new InternalASTKeyword(ot, "Others");
3140     InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
3141     internalASTOthersExpression = new InternalASTOthersExpression(keywordOthers, keywordLineArrow
           , expression);
3142 }
3143 ;
3144
3145
3146 overtureUnaryOperatorUnaryPlus returns [InternalASTUnaryOperator internalASTUnaryOperator]
3147 {
3148     internalASTUnaryOperator = null;
3149 }
3150 :
3151 up:PLUS
3152 {
3153     internalASTUnaryOperator = new InternalASTUnaryOperatorUnaryPlus(
           new InternalASTKeyword(up, "Plus"));
3154 }
3155 ;
3156
3157
3158 overtureUnaryOperatorUnaryMinus returns [InternalASTUnaryOperator internalASTUnaryOperator]
3159 {
3160     internalASTUnaryOperator = null;
3161 }
3162 :
3163 mi:MINUS
3164 {
3165     internalASTUnaryOperator = new InternalASTUnaryOperatorUnaryMinus(
           new InternalASTKeyword(mi, "Minus"));
3166 }
3167 ;
3168
3169
3170 overtureUnaryOperatorArithmeticAbs returns [InternalASTUnaryOperator internalASTUnaryOperator]
3171 {
3172     internalASTUnaryOperator = null;
3173 }
3174 :
3175 abs:ABS
3176 {
3177     internalASTUnaryOperator = new InternalASTUnaryOperatorArithmeticAbs(
           new InternalASTKeyword(abs, "Abs"));
3178 }
3179 ;
3180
3181
3182 overtureUnaryOperatorFloor returns [InternalASTUnaryOperator internalASTUnaryOperator]
3183 {
3184     internalASTUnaryOperator = null;
3185 }
3186 :
3187 floor:FLOOR
3188 {
3189     internalASTUnaryOperator = new InternalASTUnaryOperatorFloor(
           new InternalASTKeyword(floor, "Floor"));
3190 }
3191 ;
3192
3193
3194 overtureUnaryOperatorNot returns [InternalASTUnaryOperator internalASTUnaryOperator]
3195 {
3196     internalASTUnaryOperator = null;
3197 }
3198 :
3199 not:NOT
3200 {
3201     internalASTUnaryOperator = new InternalASTUnaryOperatorNot(
           new InternalASTKeyword(not, "Not"));
3202 }
3203 ;
3204 ;

```

I.1. ANTLR GRAMMAR FILE

```
3205 | overtureUnaryOperatorSetCardinality returns [InternalASTUnaryOperator internalASTUnaryOperator]
3206 | {
3207 |   internalASTUnaryOperator = null;
3208 | }
3209 | :
3210 |   card: CARD
3211 |   {
3212 |     internalASTUnaryOperator = new InternalASTUnaryOperatorSetCardinality(
3213 |       new InternalASTKeyword(card, "Card"));
3214 |   }
3215 | ;
3216 |
3217 | overtureUnaryOperatorFinitePowerSet returns [InternalASTUnaryOperator internalASTUnaryOperator]
3218 | {
3219 |   internalASTUnaryOperator = null;
3220 | }
3221 | :
3222 |   pwr: POWER
3223 |   {
3224 |     internalASTUnaryOperator = new InternalASTUnaryOperatorFinitePowerSet(
3225 |       new InternalASTKeyword(pwr, "Power"));
3226 |   }
3227 | ;
3228 |
3229 | overtureUnaryOperatorDistributedSetUnion returns [InternalASTUnaryOperator
3230 |   internalASTUnaryOperator]
3231 | {
3232 |   internalASTUnaryOperator = null;
3233 | }
3234 | :
3235 |   du: DUNION
3236 |   {
3237 |     internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSetUnion(
3238 |       new InternalASTKeyword(du, "DUnion"));
3239 |   }
3240 | ;
3241 |
3242 | overtureUnaryOperatorDistributedSetIntersection returns [InternalASTUnaryOperator
3243 |   internalASTUnaryOperator]
3244 | {
3245 |   internalASTUnaryOperator = null;
3246 | }
3247 | :
3248 |   di: DINTER
3249 |   {
3250 |     internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSetIntersection(
3251 |       new InternalASTKeyword(di, "DInter"));
3252 |   }
3253 | ;
3254 |
3255 | overtureUnaryOperatorSequenceHead returns [InternalASTUnaryOperator internalASTUnaryOperator]
3256 | {
3257 |   internalASTUnaryOperator = null;
3258 | }
3259 | :
3260 |   hd: HD
3261 |   {
3262 |     internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceHead(
3263 |       new InternalASTKeyword(hd, "Hd"));
3264 |   }
3265 | ;
3266 |
3267 | overtureUnaryOperatorSequenceTail returns [InternalASTUnaryOperator internalASTUnaryOperator]
3268 | {
3269 |   internalASTUnaryOperator = null;
3270 | }
3271 | :
3272 |   t1: TL
3273 |   {
3274 |     internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceTail(
3275 |       new InternalASTKeyword(t1, "T1"));
3276 |   }
3277 | ;
3278 |
3279 | overtureUnaryOperatorSequenceLength returns [InternalASTUnaryOperator internalASTUnaryOperator]
3280 | {
3281 |   internalASTUnaryOperator = null;
3282 | }
3283 | :
3284 |   len: LEN
3285 |   {
3286 |     internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceLength(
3287 |       new InternalASTKeyword(len, "Len"));
3288 |   }
3289 | ;
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

3287     }
3288     ;
3289
3290     overtureUnaryOperatorSequenceElements returns [InternalASTUnaryOperator internalASTUnaryOperator]
3291     {
3292         internalASTUnaryOperator = null;
3293     }
3294     :
3295     el:ELEMS
3296     {
3297         internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceElements(
3298             new InternalASTKeyword(el, "Elems"));
3299     }
3300     ;
3301
3302     overtureUnaryOperatorSequenceIndices returns [InternalASTUnaryOperator internalASTUnaryOperator]
3303     {
3304         internalASTUnaryOperator = null;
3305     }
3306     :
3307     inds:INDS
3308     {
3309         internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceIndices(
3310             new InternalASTKeyword(inds, "Inds"));
3311     }
3312     ;
3313
3314     overtureUnaryOperatorDistributedSequenceConcatenation returns [InternalASTUnaryOperator
3315         internalASTUnaryOperator]
3316     {
3317         internalASTUnaryOperator = null;
3318     }
3319     :
3320     conc:CONC
3321     {
3322         internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSequenceConcatenation(
3323             new InternalASTKeyword(conc, "Conc"));
3324     }
3325     ;
3326
3327     overtureUnaryOperatorMapDomain returns [InternalASTUnaryOperator internalASTUnaryOperator]
3328     {
3329         internalASTUnaryOperator = null;
3330     }
3331     :
3332     dom:DOM
3333     {
3334         internalASTUnaryOperator = new InternalASTUnaryOperatorMapDomain(
3335             new InternalASTKeyword(dom, "Dom"));
3336     }
3337     ;
3338
3339     overtureUnaryOperatorMapRange returns [InternalASTUnaryOperator internalASTUnaryOperator]
3340     {
3341         internalASTUnaryOperator = null;
3342     }
3343     :
3344     rng:RNG
3345     {
3346         internalASTUnaryOperator = new InternalASTUnaryOperatorMapRange(
3347             new InternalASTKeyword(rng, "Rng"));
3348     }
3349     ;
3350
3351     overtureUnaryOperatorDistributedMapMerge returns [InternalASTUnaryOperator
3352         internalASTUnaryOperator]
3353     {
3354         internalASTUnaryOperator = null;
3355     }
3356     :
3357     mer:MERGE
3358     {
3359         internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedMapMerge(
3360             new InternalASTKeyword(mer, "Merge"));
3361     }
3362     ;
3363
3364     overtureMapInverse returns [InternalASTMapInverse internalASTMapInverse]
3365     {
3366         internalASTMapInverse = null;
3367         InternalASTExpression expression = null;
3368     }
3369     :
3370     in:INVERSE

```

I.1. ANTLR GRAMMAR FILE

```
3369     expression = overtureExpression
3370     {
3371         InternalASTKeyword keywordInverse = new InternalASTKeyword(in, "Inverse");
3372         internalASTMapInverse = new InternalASTMapInverse(keywordInverse, expression);
3373     }
3374 ;
3375
3376 overtureBinaryOperatorArithmeticPlus returns [InternalASTBinaryOperatorArithmeticPlus
3377     internalASTBinaryOperator]
3378 {
3379     internalASTBinaryOperator = null;
3380 }
3381 :
3382     op:PLUS
3383     {
3384         InternalASTKeyword operator = new InternalASTKeyword(op, "Plus");
3385         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticPlus(operator);
3386     }
3387 ;
3388 overtureBinaryOperatorArithmeticMinus returns [InternalASTBinaryOperatorArithmeticMinus
3389     internalASTBinaryOperator]
3390 {
3391     internalASTBinaryOperator = null;
3392 }
3393 :
3394     op:MINUS
3395     {
3396         InternalASTKeyword operator = new InternalASTKeyword(op, "Minus");
3397         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMinus(operator);
3398     }
3399 ;
3400 overtureBinaryOperatorArithmeticMultiplication returns [
3401     InternalASTBinaryOperatorArithmeticMultiplication internalASTBinaryOperator]
3402 {
3403     internalASTBinaryOperator = null;
3404 }
3405 :
3406     op:ASTERIX
3407     {
3408         InternalASTKeyword operator = new InternalASTKeyword(op, "Asterix");
3409         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMultiplication(operator);
3410     }
3411 ;
3412 overtureBinaryOperatorArithmeticDivide returns [InternalASTBinaryOperatorArithmeticDivide
3413     internalASTBinaryOperator]
3414 {
3415     internalASTBinaryOperator = null;
3416 }
3417 :
3418     op:SLASH
3419     {
3420         InternalASTKeyword operator = new InternalASTKeyword(op, "Slash");
3421         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticDivide(operator);
3422     }
3423 ;
3424 overtureBinaryOperatorArithmeticIntegerDivision returns [
3425     InternalASTBinaryOperatorArithmeticIntegerDivision internalASTBinaryOperator]
3426 {
3427     internalASTBinaryOperator = null;
3428 }
3429 :
3430     op:DIV
3431     {
3432         InternalASTKeyword operator = new InternalASTKeyword(op, "Div");
3433         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticIntegerDivision(operator);
3434     }
3435 ;
3436 overtureBinaryOperatorArithmeticRem returns [InternalASTBinaryOperatorArithmeticRem
3437     internalASTBinaryOperator]
3438 {
3439     internalASTBinaryOperator = null;
3440 }
3441 :
3442     op:REM
3443     {
3444         InternalASTKeyword operator = new InternalASTKeyword(op, "Rem");
3445         internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticRem(operator);
3446     }
3447 ;
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
3447 |
3448 | overtureBinaryOperatorArithmeticMod returns [InternalASTBinaryOperatorArithmeticMod
      |   internalASTBinaryOperator]
3449 | {
3450 |   internalASTBinaryOperator = null;
3451 | }
3452 | :
3453 |   op:MOD
3454 |   {
3455 |     InternalASTKeyword operator = new InternalASTKeyword(op, "Mod");
3456 |     internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMod(operator);
3457 |   }
3458 | ;
3459 |
3460 | overtureBinaryOperatorLessThan returns [InternalASTBinaryOperatorLessThan
      |   internalASTBinaryOperator]
3461 | {
3462 |   internalASTBinaryOperator = null;
3463 | }
3464 | :
3465 |   op:LESSTHAN
3466 |   {
3467 |     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThan");
3468 |     internalASTBinaryOperator = new InternalASTBinaryOperatorLessThan(operator);
3469 |   }
3470 | ;
3471 |
3472 | overtureBinaryOperatorLessThanOrEqual returns [InternalASTBinaryOperatorLessThanOrEqual
      |   internalASTBinaryOperator]
3473 | {
3474 |   internalASTBinaryOperator = null;
3475 | }
3476 | :
3477 |   op:LESSTHANEQUAL
3478 |   {
3479 |     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanEqual");
3480 |     internalASTBinaryOperator = new InternalASTBinaryOperatorLessThanOrEqual(operator);
3481 |   }
3482 | ;
3483 |
3484 | overtureBinaryOperatorGreaterThan returns [InternalASTBinaryOperatorGreaterThan
      |   internalASTBinaryOperator]
3485 | {
3486 |   internalASTBinaryOperator = null;
3487 | }
3488 | :
3489 |   op:GREATERTHAN
3490 |   {
3491 |     InternalASTKeyword operator = new InternalASTKeyword(op, "GreaterThan");
3492 |     internalASTBinaryOperator = new InternalASTBinaryOperatorGreaterThan(operator);
3493 |   }
3494 | ;
3495 |
3496 | overtureBinaryOperatorGreaterThanOrEqual returns [InternalASTBinaryOperatorGreaterThanOrEqual
      |   internalASTBinaryOperator]
3497 | {
3498 |   internalASTBinaryOperator = null;
3499 | }
3500 | :
3501 |   op:GREATERTHANEQUAL
3502 |   {
3503 |     InternalASTKeyword operator = new InternalASTKeyword(op, "GreaterThanEqual");
3504 |     internalASTBinaryOperator = new InternalASTBinaryOperatorGreaterThanOrEqual(operator);
3505 |   }
3506 | ;
3507 |
3508 | overtureBinaryOperatorEqual returns [InternalASTBinaryOperatorEqual internalASTBinaryOperator]
3509 | {
3510 |   internalASTBinaryOperator = null;
3511 | }
3512 | :
3513 |   op:EQUALSIGN
3514 |   {
3515 |     InternalASTKeyword operator = new InternalASTKeyword(op, "Equalsign");
3516 |     internalASTBinaryOperator = new InternalASTBinaryOperatorEqual(operator);
3517 |   }
3518 | ;
3519 |
3520 | overtureBinaryOperatorNotEqual returns [InternalASTBinaryOperatorNotEqual
      |   internalASTBinaryOperator]
3521 | {
3522 |   internalASTBinaryOperator = null;
3523 | }
3524 | :
```


I.1. ANTLR GRAMMAR FILE

```
3525     op:NOTEQUAL
3526     {
3527         InternalASTKeyword operator = new InternalASTKeyword(op, "NotEqual");
3528         internalASTBinaryOperator = new InternalASTBinaryOperatorNotEqual(operator);
3529     }
3530 ;
3531
3532 overtureBinaryOperatorApprox returns [InternalASTBinaryOperatorApprox internalASTBinaryOperator]
3533 {
3534     internalASTBinaryOperator = null;
3535 }
3536 :
3537     op:APPROX
3538     {
3539         InternalASTKeyword operator = new InternalASTKeyword(op, "Approx");
3540         internalASTBinaryOperator = new InternalASTBinaryOperatorApprox(operator);
3541     }
3542 ;
3543
3544 overtureBinaryOperatorOr returns [InternalASTBinaryOperatorOr internalASTBinaryOperator]
3545 {
3546     internalASTBinaryOperator = null;
3547 }
3548 :
3549     op:OR
3550     {
3551         InternalASTKeyword operator = new InternalASTKeyword(op, "Or");
3552         internalASTBinaryOperator = new InternalASTBinaryOperatorOr(operator);
3553     }
3554 ;
3555
3556 overtureBinaryOperatorAnd returns [InternalASTBinaryOperatorAnd internalASTBinaryOperator]
3557 {
3558     internalASTBinaryOperator = null;
3559 }
3560 :
3561     op:AND
3562     {
3563         InternalASTKeyword operator = new InternalASTKeyword(op, "And");
3564         internalASTBinaryOperator = new InternalASTBinaryOperatorAnd(operator);
3565     }
3566 ;
3567
3568 overtureBinaryOperatorImPLY returns [InternalASTBinaryOperatorImPLY internalASTBinaryOperator]
3569 {
3570     internalASTBinaryOperator = null;
3571 }
3572 :
3573     op:IMPLY
3574     {
3575         InternalASTKeyword operator = new InternalASTKeyword(op, "ImPLY");
3576         internalASTBinaryOperator = new InternalASTBinaryOperatorImPLY(operator);
3577     }
3578 ;
3579
3580 overtureBinaryOperatorLogicalEquivalence returns [InternalASTBinaryOperatorLogicalEquivalence
3581     internalASTBinaryOperator]
3582 {
3583     internalASTBinaryOperator = null;
3584 }
3585 :
3586     op:LOGICALEQUIVALENCE
3587     {
3588         InternalASTKeyword operator = new InternalASTKeyword(op, "LogicalEquivalence");
3589         internalASTBinaryOperator = new InternalASTBinaryOperatorLogicalEquivalence(operator);
3590     }
3591 ;
3592
3593 overtureBinaryOperatorInSet returns [InternalASTBinaryOperatorInSet internalASTBinaryOperator]
3594 {
3595     internalASTBinaryOperator = null;
3596 }
3597 :
3598     in:IN
3599     set:SET
3600     {
3601         InternalASTKeyword operatorIn = new InternalASTKeyword(in, "In");
3602         InternalASTKeyword operatorSet = new InternalASTKeyword(set, "Set");
3603         internalASTBinaryOperator = new InternalASTBinaryOperatorInSet(operatorIn, operatorSet);
3604     }
3605 ;
3606
3607 overtureBinaryOperatorNotInSet returns [InternalASTBinaryOperatorNotInSet
3608     internalASTBinaryOperator]
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

3607 {
3608     internalASTBinaryOperator = null;
3609 }
3610 :
3611     not:NOT
3612     in:IN
3613     set:SET
3614     {
3615         InternalASTKeyword operatorNot = new InternalASTKeyword(not, "Not");
3616         InternalASTKeyword operatorIn = new InternalASTKeyword(in, "In");
3617         InternalASTKeyword operatorSet = new InternalASTKeyword(set, "Set");
3618         internalASTBinaryOperator = new InternalASTBinaryOperatorNotInSet(operatorNot, operatorIn,
3619             operatorSet);
3620     };
3621
3622 overtureBinaryOperatorSubset returns [InternalASTBinaryOperatorSubset internalASTBinaryOperator]
3623 {
3624     internalASTBinaryOperator = null;
3625 }
3626 :
3627     op:SUBSET
3628     {
3629         InternalASTKeyword operator = new InternalASTKeyword(op, "Subset");
3630         internalASTBinaryOperator = new InternalASTBinaryOperatorSubset(operator);
3631     }
3632 ;
3633
3634 overtureBinaryOperatorProperSubset returns [InternalASTBinaryOperatorProperSubset
3635     internalASTBinaryOperator]
3636 {
3637     internalASTBinaryOperator = null;
3638 }
3639 :
3640     op:PSUBSET
3641     {
3642         InternalASTKeyword operator = new InternalASTKeyword(op, "PSubset");
3643         internalASTBinaryOperator = new InternalASTBinaryOperatorProperSubset(operator);
3644     }
3645 ;
3646
3647 overtureBinaryOperatorSetUnion returns [InternalASTBinaryOperatorSetUnion
3648     internalASTBinaryOperator]
3649 {
3650     internalASTBinaryOperator = null;
3651 }
3652 :
3653     op:UNION
3654     {
3655         InternalASTKeyword operator = new InternalASTKeyword(op, "Union");
3656         internalASTBinaryOperator = new InternalASTBinaryOperatorSetUnion(operator);
3657     }
3658 ;
3659
3660 overtureBinaryOperatorSetDifference returns [InternalASTBinaryOperatorSetDifference
3661     internalASTBinaryOperator]
3662 {
3663     internalASTBinaryOperator = null;
3664 }
3665 :
3666     op:BACKSLASH
3667     {
3668         InternalASTKeyword operator = new InternalASTKeyword(op, "Backslash");
3669         internalASTBinaryOperator = new InternalASTBinaryOperatorSetDifference(operator);
3670     }
3671 ;
3672
3673 overtureBinaryOperatorSetIntersection returns [InternalASTBinaryOperatorSetIntersection
3674     internalASTBinaryOperator]
3675 {
3676     internalASTBinaryOperator = null;
3677 }
3678 :
3679     op:INTER
3680     {
3681         InternalASTKeyword operator = new InternalASTKeyword(op, "Inter");
3682         internalASTBinaryOperator = new InternalASTBinaryOperatorSetIntersection(operator);
3683     }
3684 ;
3685
3686 overtureBinaryOperatorSequenceConcatenate returns [InternalASTBinaryOperatorSequenceConcatenate
3687     internalASTBinaryOperator]
3688 {
3689     internalASTBinaryOperator = null;

```

I.1. ANTLR GRAMMAR FILE

```
3685 }
3686 :
3687   op:HAT
3688   {
3689     InternalASTKeyword operator = new InternalASTKeyword(op, "Hat");
3690     internalASTBinaryOperator = new InternalASTBinaryOperatorSequenceConcatenate(operator);
3691   }
3692 ;
3693
3694 overtureBinaryOperatorMapOrSequenceModify returns [InternalASTBinaryOperatorMapOrSequenceModify
3695   internalASTBinaryOperator]
3696 {
3697   internalASTBinaryOperator = null;
3698 }
3699 :
3700   op:DOUBLEPLUS
3701   {
3702     InternalASTKeyword operator = new InternalASTKeyword(op, "DoublePlus");
3703     internalASTBinaryOperator = new InternalASTBinaryOperatorMapOrSequenceModify(operator);
3704   }
3705 ;
3706 overtureBinaryOperatorMapMerge returns [InternalASTBinaryOperatorMapMerge
3707   internalASTBinaryOperator]
3708 {
3709   internalASTBinaryOperator = null;
3710 }
3711 :
3712   op:MUNION
3713   {
3714     InternalASTKeyword operator = new InternalASTKeyword(op, "MUnion");
3715     internalASTBinaryOperator = new InternalASTBinaryOperatorMapMerge(operator);
3716   }
3717 ;
3718 overtureBinaryOperatorMapDomainRestrictTo returns [InternalASTBinaryOperatorMapDomainRestrictTo
3719   internalASTBinaryOperator]
3720 {
3721   internalASTBinaryOperator = null;
3722 }
3723 :
3724   op:LESSTHANCOLON
3725   {
3726     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanColon");
3727     internalASTBinaryOperator = new InternalASTBinaryOperatorMapDomainRestrictTo(operator);
3728   }
3729 ;
3730 overtureBinaryOperatorMapDomainRestrictBy returns [InternalASTBinaryOperatorMapDomainRestrictBy
3731   internalASTBinaryOperator]
3732 {
3733   internalASTBinaryOperator = null;
3734 }
3735 :
3736   op:LESSTHANLINECOLON
3737   {
3738     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanLineColon");
3739     internalASTBinaryOperator = new InternalASTBinaryOperatorMapDomainRestrictBy(operator);
3740   }
3741 ;
3742 overtureBinaryOperatorMapRangeRestrictTo returns [InternalASTBinaryOperatorMapRangeRestrictTo
3743   internalASTBinaryOperator]
3744 {
3745   internalASTBinaryOperator = null;
3746 }
3747 :
3748   op:COLONGREATERTHAN
3749   {
3750     InternalASTKeyword operator = new InternalASTKeyword(op, "ColonGreaterThan");
3751     internalASTBinaryOperator = new InternalASTBinaryOperatorMapRangeRestrictTo(operator);
3752   }
3753 ;
3754 overtureBinaryOperatorMapRangeRestrictBy returns [InternalASTBinaryOperatorMapRangeRestrictBy
3755   internalASTBinaryOperator]
3756 {
3757   internalASTBinaryOperator = null;
3758 }
3759 :
3760   op:COLONLINEGREATERTHAN
3761   {
3762     InternalASTKeyword operator = new InternalASTKeyword(op, "ColonLineGreaterThan");
3763     internalASTBinaryOperator = new InternalASTBinaryOperatorMapRangeRestrictBy(operator);
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

3763     }
3764     ;
3765
3766     overtureBinaryOperatorComposition returns [InternalASTBinaryOperatorComposition
        internalASTBinaryOperator]
3767     {
3768         internalASTBinaryOperator = null;
3769     }
3770     :
3771     op:COMP
3772     {
3773         InternalASTKeyword operator = new InternalASTKeyword(op, "Comp");
3774         internalASTBinaryOperator = new InternalASTBinaryOperatorComposition(operator);
3775     }
3776     ;
3777
3778     overtureBinaryOperatorIterate returns [InternalASTBinaryOperatorIterate internalASTBinaryOperator
        ]
3779     {
3780         internalASTBinaryOperator = null;
3781     }
3782     :
3783     op:ITERATE
3784     {
3785         InternalASTKeyword operator = new InternalASTKeyword(op, "Iterate");
3786         internalASTBinaryOperator = new InternalASTBinaryOperatorIterate(operator);
3787     }
3788     ;
3789
3790     // QUANTIFIED EXPRESSION
3791
3792     //quantified expression = all expression
3793     //      | exists expression
3794     //      | exists unique expression ;
3795     overtureQuantifiedExpression returns [InternalASTQuantifiedExpression
        internalASTQuantifiedExpression]
3796     {
3797         internalASTQuantifiedExpression = null;
3798     }
3799     :
3800     (
3801         internalASTQuantifiedExpression = overtureAllExpression
3802         | internalASTQuantifiedExpression = overtureExistsExpression
3803         | internalASTQuantifiedExpression = overtureExistsUniqueExpression
3804     )
3805     ;
3806
3807     //all expression = 'forall', bind list, 'E', expression ;
3808     overtureAllExpression returns [InternalASTAllExpression internalASTAllExpression]
3809     {
3810         internalASTAllExpression = null;
3811         InternalASTBindList bindList = null;
3812         InternalASTExpression expression = null;
3813     }
3814     :
3815     kwforall:FORALL
3816     bindList = overtureBindList
3817     kwand:ANDSIGN
3818     expression = overtureExpression
3819     {
3820         InternalASTKeyword keywordForall = new InternalASTKeyword(kwforall, "Forall");
3821         InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
3822         internalASTAllExpression = new InternalASTAllExpression(keywordForall, bindList, keywordAnd,
            expression);
3823     }
3824     ;
3825
3826     //exists expression = 'exists', bind list, 'E', expression ;
3827     overtureExistsExpression returns [InternalASTExistsExpression internalASTExistsExpression]
3828     {
3829         internalASTExistsExpression = null;
3830         InternalASTBindList bindList = null;
3831         InternalASTExpression expression = null;
3832     }
3833     :
3834     kwexists:EXISTS
3835     bindList = overtureBindList
3836     kwand:ANDSIGN
3837     expression = overtureExpression
3838     {
3839         InternalASTKeyword keywordExists = new InternalASTKeyword(kwexists, "Exists");
3840         InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");

```

I.1. ANTLR GRAMMAR FILE

```

3843     internalASTExistsExpression = new InternalASTExistsExpression(keywordExists, bindList,
3844     )
3845     }
3846     ;
3847
3848 //exists unique expression = 'exists1', bind, 'E', expression ;
3849 overtureExistsUniqueExpression returns [InternalASTExistsUniqueExpression
3850     internalASTExistsUniqueExpression]
3851 {
3852     internalASTExistsUniqueExpression = null;
3853     InternalASTBind bind = null;
3854     InternalASTExpression expression = null;
3855 }
3856 :
3857     kwexists1:EXISTS1
3858     bind = overtureBind
3859     kwand:ANDSIGN
3860     expression = overtureExpression
3861     {
3862         InternalASTKeyword keywordExists1 = new InternalASTKeyword(kwexists1, "Exists1");
3863         InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
3864         internalASTExistsUniqueExpression = new InternalASTExistsUniqueExpression(keywordExists1, bind
3865         , keywordAnd, expression);
3866     }
3867     ;
3868
3869 //iota expression = 'iota', bind, 'E', expression ;
3870 overtureIotaExpression returns [InternalASTIotaExpression internalASTIotaExpression]
3871 {
3872     internalASTIotaExpression = null;
3873     InternalASTBind bind = null;
3874     InternalASTExpression expression = null;
3875 }
3876 :
3877     kwiota:IOTA
3878     bind = overtureBind
3879     kwand:ANDSIGN
3880     expression = overtureExpression
3881     {
3882         InternalASTKeyword keywordIota = new InternalASTKeyword(kwiota, "Iota");
3883         InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
3884         internalASTIotaExpression = new InternalASTIotaExpression(keywordIota, bind, keywordAnd,
3885         expression);
3886     }
3887     ;
3888
3889 //set enumeration = '{', [ expression list ], '}' ;
3890 overtureSetEnumeration returns [InternalASTSetEnumeration internalASTSetEnumeration]
3891 {
3892     internalASTSetEnumeration = null;
3893     InternalASTExpressionList internalASTExpressionList = null;
3894 }
3895 :
3896     lb:LBRACE
3897     (
3898         internalASTExpressionList = overtureExpressionList
3899     )?
3900     rb:RBRACE
3901     {
3902         InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb, "LeftBrace");
3903         InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb, "RightBrace");
3904         internalASTSetEnumeration = new InternalASTSetEnumeration(keywordLeftBrace,
3905         internalASTExpressionList, keywordRightBrace);
3906     }
3907     ;
3908
3909 //set comprehension = '{', expression, '|', bind list,
3910 //     [ 'E', expression ], '}' ;
3911 overtureSetComprehension returns [InternalASTSetComprehension internalASTSetComprehension]
3912 {
3913     internalASTSetComprehension = null;
3914     InternalASTExpression expression1 = null;
3915     InternalASTBindList bindList = null;
3916     InternalASTKeyword keywordAndsign = null;
3917     InternalASTExpression expression2 = null;
3918 }
3919 :
3920     lb:LBRACE
3921     expression1 = overtureExpression
3922     vb:VBAR
3923     bindList = overtureBindList
3924     (
3925         an:ANDSIGN

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

3922     expression2 = overtureExpression
3923     {
3924         keywordAndsign = new InternalASTKeyword(an, "Andsign");
3925     }
3926 )?
3927 rb:RBRACE
3928 {
3929     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb,"LeftBrace");
3930     InternalASTKeyword keywordVBar = new InternalASTKeyword(vb, "VBar");
3931     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb,"RightBrace");
3932     internalASTSetComprehension = new InternalASTSetComprehension(keywordLeftBrace, expression1,
        keywordVBar, bindList, keywordAndsign, expression2, keywordRightBrace);
3933 }
3934 ;
3935
3936 //set range expression = '{', expression, ',', '...', ',',
3937 // expression, '}' ;
3938 overtureSetRangeExpression returns [InternalASTSetRangeExpression internalASTSetRangeExpression]
3939 {
3940     internalASTSetRangeExpression = null;
3941     InternalASTExpression expression1 = null;
3942     InternalASTExpression expression2 = null;
3943 }
3944 :
3945 lb:LBRACE
3946 expression1 = overtureExpression
3947 col:COMMA
3948 dots:DOTS
3949 co2:COMMA
3950 expression2 = overtureExpression
3951 rb:RBRACE
3952 {
3953     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb,"LeftBrace");
3954     InternalASTKeyword keywordCommal = new InternalASTKeyword(col, "Commal");
3955     InternalASTKeyword keywordDots = new InternalASTKeyword(dots, "Dots");
3956     InternalASTKeyword keywordComma2 = new InternalASTKeyword(co2, "Comma2");
3957     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb,"RightBrace");
3958     internalASTSetRangeExpression = new InternalASTSetRangeExpression(keywordLeftBrace,
        expression1, keywordCommal, keywordDots, keywordComma2, expression2, keywordRightBrace);
3959 }
3960 ;
3961
3962 //sequence enumeration = '[', [ expression list ], '[' ;
3963 overtureSequenceEnumeration returns [InternalASTSequenceEnumeration
    internalASTSequenceEnumeration]
3964 {
3965     internalASTSequenceEnumeration = null;
3966     InternalASTExpressionList internalASTExpressionList = null;
3967 }
3968 :
3969 lb:LBRACK
3970 (
3971     internalASTExpressionList = overtureExpressionList
3972 )?
3973 rb:RBRACK
3974 {
3975     InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lb,"LeftBrack");
3976     InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rb,"RightBrack");
3977     internalASTSequenceEnumeration = new InternalASTSequenceEnumeration(keywordLeftBrack,
        internalASTExpressionList, keywordRightBrack);
3978 }
3979 ;
3980
3981 //sequence comprehension = '[', expression, '[', set bind, [ 'E', expression ], '[' ;
3982 overtureSequenceComprehension returns [InternalASTSequenceComprehension
    internalASTSequenceComprehension]
3983 {
3984     internalASTSequenceComprehension = null;
3985     InternalASTExpression expression1 = null;
3986     InternalASTSetBind setBind = null;
3987     InternalASTKeyword keywordAndsign = null;
3988     InternalASTExpression expression2 = null;
3989 }
3990 :
3991 lb:LBRACK
3992 expression1 = overtureExpression
3993 vb:VBAR
3994 setBind = overtureSetBind
3995 (
3996     an:ANDSIGN
3997     expression2 = overtureExpression
3998     {
3999         keywordAndsign = new InternalASTKeyword(an, "Andsign");
4000     }

```

I.1. ANTLR GRAMMAR FILE

```

4001     )?
4002     rb:RBRACK
4003     {
4004         InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lb, "LeftBrack");
4005         InternalASTKeyword keywordVBar = new InternalASTKeyword(vb, "VBar");
4006         InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rb, "RightBrack");
4007         internalASTSequenceComprehension = new InternalASTSequenceComprehension(keywordLeftBrack,
4008             expression1, keywordVBar, setBind, keywordAndsign, expression2, keywordRightBrack);
4009     }
4010 ;
4011 //subsequence = expression, '(', expression, ',', '...', ',', expression, ')' ;
4012 overtureSubsequence returns [InternalASTSubsequence internalASTSubsequence]
4013 {
4014     internalASTSubsequence = null;
4015     InternalASTExpression expression1 = null;
4016     InternalASTExpression expression2 = null;
4017     InternalASTExpression expression3 = null;
4018 }
4019 :
4020     expression1 = overtureExpressionPre61
4021     lb:LBRACKET
4022     expression2 = overtureExpression
4023     co1:COMMA
4024     dots:DOTS
4025     co2:COMMA
4026     expression3 = overtureExpression
4027     rb:RBRACKET
4028     {
4029         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4030         InternalASTKeyword keywordCommal = new InternalASTKeyword(co1, "Commal");
4031         InternalASTKeyword keywordDots = new InternalASTKeyword(dots, "Dots");
4032         InternalASTKeyword keywordComma2 = new InternalASTKeyword(co2, "Comma2");
4033         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4034         internalASTSubsequence = new InternalASTSubsequence(expression1, keywordLeftBracket,
4035             expression2, keywordCommal, keywordDots, keywordComma2, expression3, keywordRightBracket)
4036     }
4037 ;
4038 //map enumeration = '{', maplet, { ',', maplet }, ','
4039 // | '{', '|->', ',' ;
4040 overtureMapEnumeration returns [InternalASTMapEnumeration internalASTMapEnumeration]
4041 {
4042     internalASTMapEnumeration = null;
4043     List<InternalASTMapEnumerationMapletElement> mapEnumerationMapletElements = new ArrayList<
4044         InternalASTMapEnumerationMapletElement>();
4045     InternalASTMapEnumerationMaplet mapEnumerationMaplet = null;
4046     InternalASTMapEnumerationMapletElement currentElement = null;
4047     InternalASTMapEnumerationArrow mapEnumerationArrow = null;
4048     InternalASTMaplet maplet = null;
4049 }
4050 :
4051     (
4052         lbr:LBRACE
4053         maplet = overtureMaplet
4054         {
4055             currentElement = new InternalASTMapEnumerationMapletElement(maplet);
4056             mapEnumerationMapletElements.add(currentElement);
4057         }
4058         co:COMMA
4059         maplet = overtureMaplet
4060         {
4061             InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
4062             currentElement = new InternalASTMapEnumerationMapletElement(keywordComma, maplet);
4063             mapEnumerationMapletElements.add(currentElement);
4064         }
4065     )*
4066     rbr:RBRACE
4067     {
4068         InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lbr, "LeftBrace");
4069         InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rbr, "RightBrace");
4070         internalASTMapEnumeration = new InternalASTMapEnumerationMaplet(keywordLeftBrace,
4071             mapEnumerationMapletElements, keywordRightBrace);
4072     }
4073 )
4074 |
4075     (
4076         lbr2:LBRACE
4077         vba:VBARARROW
4078         rbr2:RBRACE
4079         {

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4080     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lbr2, "LeftBrace");
4081     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rbr2, "RightBrace");
4082     InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(vba, "VBarArrow");
4083     internalASTMapEnumeration = new InternalASTMapEnumerationArrow(keywordLeftBrace,
        keywordVBarArrow, keywordRightBrace);
4084     }
4085     )
4086     )
4087     ;
4088
4089     //maplet = expression, '/->', expression ;
4090     overturmMaplet returns [InternalASTMaplet internalASTMaplet]
4091     {
4092     internalASTMaplet = null;
4093     InternalASTExpression internalASTExpression1 = null;
4094     InternalASTExpression internalASTExpression2 = null;
4095     }
4096     :
4097     internalASTExpression1 = overturmExpression
4098     kwvbararrow:VBARARROW
4099     internalASTExpression2 = overturmExpression
4100     {
4101     InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow, "VBarArrow");
4102     internalASTMaplet = new InternalASTMaplet(internalASTExpression1, keywordVBarArrow,
        internalASTExpression2);
4103     }
4104     ;
4105
4106     //map comprehension = '{', maplet, '|', bind list,
4107     // [ 'E', expression ], '}' ;
4108     overturmMapComprehension returns [InternalASTMapComprehension internalASTMapComprehension]
4109     {
4110     internalASTMapComprehension = null;
4111     InternalASTMaplet internalASTMaplet = null;
4112     InternalASTBindList internalASTBindList = null;
4113     InternalASTExpression internalASTExpression = null;
4114     InternalASTKeyword keywordAndsign = null;
4115     }
4116     :
4117     kwlb:LBACE
4118     internalASTMaplet = overturmMaplet
4119     kwvbar:VBAR
4120     internalASTBindList = overturmBindList
4121     (
4122     kwandsign:ANDSIGN
4123     internalASTExpression = overturmExpression
4124     {
4125     keywordAndsign = new InternalASTKeyword(kwandsign, "Andsign");
4126     }
4127     )?
4128     kwrb:RBACE
4129     {
4130     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(kwlb, "LeftBrace");
4131     InternalASTKeyword keywordVBar = new InternalASTKeyword(kwvbar, "VBar");
4132     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(kwrb, "RightBrace");
4133     internalASTMapComprehension = new InternalASTMapComprehension(keywordLeftBrace,
        internalASTMaplet, keywordVBar, internalASTBindList, keywordAndsign,
        internalASTExpression, keywordRightBrace);
4134     }
4135     ;
4136
4137     //tuple constructor = 'mk ', '(', expression, expression list, ')' ;
4138     overturmTupleConstructor returns [InternalASTTupleConstructor internalASTTupleConstructor]
4139     {
4140     internalASTTupleConstructor = null;
4141     InternalASTExpression internalASTExpression = null;
4142     InternalASTExpressionList internalASTExpressionList = null;
4143     }
4144     :
4145     kwmk:MAKE
4146     kwlb:LBACKET
4147     internalASTExpression = overturmExpression
4148     internalASTExpressionList = overturmExpressionList
4149     kwrb:RBACKET
4150     {
4151     InternalASTKeyword keywordMake = new InternalASTKeyword(kwmk, "Mk");
4152     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
4153     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
4154     internalASTTupleConstructor = new InternalASTTupleConstructor(keywordMake, keywordLeftBracket
        , internalASTExpression, internalASTExpressionList, keywordRightBracket);
4155     }
4156     ;
4157
4158     //record constructor = 'mk_', name, '(', [ expression list ], ')' ;

```


I.1. ANTLR GRAMMAR FILE

```

4159 | overtureRecordConstructor returns [InternalASTRecordConstructor internalASTRecordConstructor]
4160 | {
4161 |     internalASTRecordConstructor = null;
4162 |     InternalASTName internalASTName = null;
4163 |     InternalASTExpressionList internalASTExpressionList = null;
4164 | }
4165 | :
4166 |     kwmk:MAKE
4167 |     internalASTName = overtureName
4168 |     kwlb:LBRACKET
4169 |     (
4170 |         internalASTExpressionList = overtureExpressionList
4171 |     )?
4172 |     kwrb:RBRACKET
4173 |     {
4174 |         InternalASTKeyword keywordMake = new InternalASTKeyword(kwmk, "Mk");
4175 |         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
4176 |         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
4177 |         internalASTRecordConstructor = new InternalASTRecordConstructor(keywordMake, internalASTName,
4178 |             keywordLeftBracket, internalASTExpressionList, keywordRightBracket);
4179 |     }
4180 | ;
4181 | //record modifier = 'mu', '(', expression, ',',
4182 | //    record modification,
4183 | //    { ',', record modification }, ')' ;
4184 | overtureRecordModifier returns [InternalASTRecordModifier internalASTRecordModifier]
4185 | {
4186 |     internalASTRecordModifier = null;
4187 |     InternalASTRecordModifierElement internalASTRecordModifierElement = null;
4188 |     List<InternalASTRecordModifierElement> recordModifierElements = new ArrayList<
4189 |         InternalASTRecordModifierElement>();
4190 |     InternalASTExpression internalASTExpression = null;
4191 |     InternalASTRecordModification internalASTRecordModification = null;
4192 | }
4193 | : kwmu:MU
4194 |     kwlb:LBRACKET
4195 |         internalASTExpression = overtureExpression
4196 |         kwcomma:COMMA
4197 |         internalASTRecordModification = overtureRecordModification
4198 |         {
4199 |             InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
4200 |             internalASTRecordModifierElement = new InternalASTRecordModifierElement(keywordComma,
4201 |                 internalASTRecordModification);
4202 |             recordModifierElements.add(internalASTRecordModifierElement);
4203 |         }
4204 |         (
4205 |             kwcomma2:COMMA
4206 |             internalASTRecordModification = overtureRecordModification
4207 |             {
4208 |                 InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma2, "Comma");
4209 |                 internalASTRecordModifierElement = new InternalASTRecordModifierElement(keywordComma,
4210 |                     internalASTRecordModification);
4211 |                 recordModifierElements.add(internalASTRecordModifierElement);
4212 |             }
4213 |         )*
4214 |     kwrb:RBRACKET
4215 |     {
4216 |         InternalASTKeyword keywordMu = new InternalASTKeyword(kwmu, "Mu");
4217 |         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
4218 |         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
4219 |         internalASTRecordModifier = new InternalASTRecordModifier(keywordMu, keywordLeftBracket,
4220 |             internalASTExpression, recordModifierElements, keywordRightBracket);
4221 |     }
4222 | ;
4223 | //record modification = identifier, '|->', expression ;
4224 | overtureRecordModification returns [InternalASTRecordModification internalASTRecordModification]
4225 | {
4226 |     internalASTRecordModification = null;
4227 |     InternalASTExpression internalASTExpression = null;
4228 | }
4229 | :
4230 |     kwid:IDENTIFIER
4231 |     kwvbararrow:VBARARROW
4232 |     internalASTExpression = overtureExpression
4233 |     {
4234 |         InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(kwid);
4235 |         InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow, "VBarArrow");
4236 |         internalASTRecordModification = new InternalASTRecordModification(internalASTIdentifier,
4237 |             keywordVBarArrow, internalASTExpression);
4238 |     }
4239 | ;

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4237 //apply = expression, '(' [ expression list ], ')';
4238 overtureApply returns [InternalASTApply internalASTApply]
4239 {
4240     internalASTApply = null;
4241     InternalASTExpression internalASTExpression = null;
4242     InternalASTExpressionList internalASTExpressionList = null;
4243 }
4244 :
4245 (
4246     (overtureSubsequence)=> internalASTExpression = overtureSubsequence
4247     | (overtureFunctionTypeInstantiation) => internalASTExpression =
4248         overtureFunctionTypeInstantiation
4249     | internalASTExpression = overtureExpressionPre61
4250 )
4251 kwlb:LBRACKET
4252 (
4253     internalASTExpressionList = overtureExpressionList
4254 )?
4255 kwrb:RBRACKET
4256 {
4257     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
4258     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
4259     internalASTApply = new InternalASTApply(internalASTExpression, keywordLeftBracket,
4260         internalASTExpressionList, keywordRightBracket);
4261 }
4262 ;
4263 //field select = expression, '.', identifier ;
4264 overtureFieldSelect returns [InternalASTFieldSelect internalASTFieldSelect]
4265 {
4266     internalASTFieldSelect = null;
4267     InternalASTExpression internalASTExpression = null;
4268 }
4269 :
4270 (
4271     (overtureSubsequence)=> internalASTExpression = overtureSubsequence
4272     | (overtureApply)=> internalASTExpression = overtureApply
4273     | (overtureFunctionTypeInstantiation)=> internalASTExpression =
4274         overtureFunctionTypeInstantiation
4275     | internalASTExpression = overtureExpressionPre61
4276 )
4277 kwidot:DOT
4278 id:IDENTIFIER
4279 {
4280     InternalASTKeyword keywordDot = new InternalASTKeyword(kwidot,"Dot");
4281     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
4282     internalASTFieldSelect = new InternalASTFieldSelect(internalASTExpression, keywordDot,
4283         identifier);
4284 }
4285 ;
4286 //tuple select = expression, '#', numeral ;
4287 overtureTupleSelect returns [InternalASTTupleSelect internalASTTupleSelect]
4288 {
4289     internalASTTupleSelect = null;
4290     InternalASTExpression internalASTExpression = null;
4291     InternalASTNumeral internalASTNumeral = null;
4292     InternalASTSymbolicLiteral symbolicLiteral = null;
4293 }
4294 :
4295 (
4296     (overtureQuantifiedExpression)=> internalASTExpression = overtureQuantifiedExpression
4297     | internalASTExpression = overtureIotaExpression
4298     | (overtureSetEnumeration)=> internalASTExpression = overtureSetEnumeration
4299     | (overtureSetComprehension)=> internalASTExpression = overtureSetComprehension
4300     | (overtureSetRangeExpression)=> internalASTExpression = overtureSetRangeExpression
4301     | (overtureSequenceEnumeration)=> internalASTExpression = overtureSequenceEnumeration
4302     | (overtureSequenceComprehension)=> internalASTExpression = overtureSequenceComprehension
4303     | (overtureMapEnumeration)=> internalASTExpression = overtureMapEnumeration
4304     | (overtureMapComprehension)=> internalASTExpression = overtureMapComprehension
4305     | internalASTExpression = overtureTupleConstructor
4306     | internalASTExpression = overtureRecordConstructor
4307     | internalASTExpression = overtureRecordModifier
4308     | internalASTExpression = overtureLambdaExpression
4309     | internalASTExpression = overtureNewExpression
4310     | internalASTExpression = overtureGeneralIsExpression
4311     | internalASTExpression = overtureIsofbaseclassExpression
4312     | internalASTExpression = overtureIsofclassExpression
4313     | internalASTExpression = overtureSamebaseclassExpression
4314     | internalASTExpression = overtureSameclassExpression
4315     | internalASTExpression = overtureActExpression
4316     | internalASTExpression = overtureFinExpression
4317     | internalASTExpression = overtureActiveExpression
4318     | internalASTExpression = overtureReqExpression

```

I.1. ANTLR GRAMMAR FILE

```

4317 | internalASTExpression = overtureWaitingExpression
4318 | {
4319 |     internalASTExpression = new InternalASTExpressionSymbolicLiteral(symbolicLiteral);
4320 | }
4321 | internalASTExpression = overtureExpressionPre21
4322 )
4323 kwdotsharp:DOTSHARP
4324 internalASTNumeral = overtureNumeral
4325 {
4326     InternalASTKeyword keywordDotSharp = new InternalASTKeyword(kwdotsharp, "DotSharp");
4327     internalASTTupleSelect = new InternalASTTupleSelect(internalASTExpression, keywordDotSharp,
4328         internalASTNumeral);
4329 }
4329 ;
4330
4331 //function type instantiation = name, '[', type, { ',', type }, ']' ;
4332 overtureFunctionTypeInstantiation returns [InternalASTFunctionTypeInstantiation
4333     internalASTFunctionTypeInstantiation]
4334 {
4335     internalASTFunctionTypeInstantiation = null;
4336     InternalASTFunctionTypeInstantiationElement internalASTFunctionTypeInstantiationElement = null;
4337     List<InternalASTFunctionTypeInstantiationElement> functionTypeInstantiationElements = new
4338         ArrayList<InternalASTFunctionTypeInstantiationElement>();
4339     InternalASTName internalASTName = null;
4340     InternalASTType internalASTType = null;
4341 }
4342 :
4343     internalASTName = overtureName
4344     kwlb:LBRACK
4345     internalASTType = overtureType
4346     {
4347         internalASTFunctionTypeInstantiationElement = new
4348             InternalASTFunctionTypeInstantiationElement(internalASTType);
4349         functionTypeInstantiationElements.add(internalASTFunctionTypeInstantiationElement);
4350     }
4351     (
4352         kwcomma:COMMA
4353         internalASTType = overtureType
4354         {
4355             InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
4356             internalASTFunctionTypeInstantiationElement = new
4357                 InternalASTFunctionTypeInstantiationElement(keywordComma, internalASTType);
4358             functionTypeInstantiationElements.add(internalASTFunctionTypeInstantiationElement);
4359         }
4360     )*
4361     kwrb:RBRACK
4362     {
4363         InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb, "LeftBrack");
4364         InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb, "RightBrack");
4365         internalASTFunctionTypeInstantiation = new InternalASTFunctionTypeInstantiation(
4366             internalASTName, keywordLeftBrack, functionTypeInstantiationElements, keywordRightBrack)
4367         ;
4368     }
4369 }
4370 ;
4371
4372 //lambda expression = 'lambda', type bind list, 'E', expression ;
4373 overtureLambdaExpression returns [InternalASTLambdaExpression internalASTLambdaExpression]
4374 {
4375     internalASTLambdaExpression = null;
4376     InternalASTTypeBindList internalASTTypeBindList = null;
4377     InternalASTExpression internalASTExpression = null;
4378 }
4379 :
4380     kwlambda:LAMBDA
4381     internalASTTypeBindList = overtureTypeBindList
4382     kwandsign:ANDSIGN
4383     internalASTExpression = overtureExpression
4384     {
4385         InternalASTKeyword keywordLambda = new InternalASTKeyword(kwlambda, "Lambda");
4386         InternalASTKeyword keywordAndsign = new InternalASTKeyword(kwandsign, "Andsign");
4387         internalASTLambdaExpression = new InternalASTLambdaExpression(keywordLambda,
4388             internalASTTypeBindList, keywordAndsign, internalASTExpression);
4389     }
4390 }
4391 ;
4392
4393 //new expression = 'new', name, '(', [ instvarinit expression ], ')' ;
4394 overtureNewExpression returns [InternalASTNewExpression internalASTNewExpression]
4395 {
4396     internalASTNewExpression = null;
4397     InternalASTName internalASTName = null;
4398     InternalASTInstvarinitExpression internalASTInstvarinitExpression = null;
4399 }
4400 :
4401     kwnew:NEW

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4393     internalASTName = overtureName
4394     kwlb:LBRACKET
4395     (
4396     internalASTInstvarinitExpression = overtureInstvarinitExpression
4397     )?
4398     kwrb:RBRACKET
4399     {
4400     InternalASTKeyword keywordNew = new InternalASTKeyword(kwnew,"New");
4401     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
4402     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
4403     internalASTNewExpression = new InternalASTNewExpression(keywordNew, internalASTName,
         keywordLeftBracket, internalASTInstvarinitExpression, keywordRightBracket);
4404     }
4405     ;
4406
4407 //instvarinit expression = name, '|->', expression ;
4408 overtureInstvarinitExpression returns [InternalASTInstvarinitExpression
    internalASTInstvarinitExpression]
4409 {
4410     internalASTInstvarinitExpression = null;
4411     InternalASTName internalASTName = null;
4412     InternalASTExpression internalASTExpression = null;
4413 }
4414 :
4415     internalASTName = overtureName
4416     kwvbararrow:VBARARROW
4417     internalASTExpression = overtureExpression
4418     {
4419     InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
4420     internalASTInstvarinitExpression = new InternalASTInstvarinitExpression(internalASTName,
         keywordVBarArrow, internalASTExpression);
4421     }
4422     ;
4423
4424 //self expression = 'self' ;
4425 overtureSelfExpression returns [InternalASTSelfExpression internalASTSelfExpression]
4426 {
4427     internalASTSelfExpression = null;
4428 }
4429 :
4430     kwself:SELF
4431     {
4432     InternalASTKeyword keywordSelf = new InternalASTKeyword(kwself,"Self");
4433     internalASTSelfExpression = new InternalASTSelfExpression(keywordSelf);
4434     }
4435     ;
4436
4437 //threadid expression = 'threadid' ;
4438 overtureThreadidExpression returns [InternalASTThreadidExpression internalASTThreadidExpression]
4439 {
4440     internalASTThreadidExpression = null;
4441 }
4442 :
4443     kwthreadid:THREADID
4444     {
4445     InternalASTKeyword keywordThreadid = new InternalASTKeyword(kwthreadid,"Threadid");
4446     internalASTThreadidExpression = new InternalASTThreadidExpression(keywordThreadid);
4447     }
4448     ;
4449
4450 //general is expression = is expression
4451 //      | type judgement ;
4452 overtureGeneralIsExpression returns [InternalASTGeneralIsExpression
    internalASTGeneralIsExpression]
4453 {
4454     internalASTGeneralIsExpression = null;
4455 }
4456 :
4457     (
4458     (overtureIsExpression)=> internalASTGeneralIsExpression = overtureIsExpression
4459     | internalASTGeneralIsExpression = overtureTypeJudgement
4460     )
4461     ;
4462
4463 //is expression = 'is', name, '(', expression, ')'
4464 //      | is basic type, '(', expression, ')' ;
4465 overtureIsExpression returns [InternalASTIsExpression internalASTIsExpression]
4466 {
4467     internalASTIsExpression = null;
4468     InternalASTName internalASTName = null;
4469     InternalASTIsBasicType internalASTIsBasicType = null;
4470     InternalASTExpression internalASTExpression = null;
4471 }
4472 }

```

I.1. ANTLR GRAMMAR FILE

```

4473 :
4474 (
4475   kwis:ISU
4476   internalASTName = overtureName
4477   kwlb:LBRACKET
4478   internalASTExpression = overtureExpression
4479   kwrbr:RBRACKET
4480   {
4481     InternalASTKeyword keywordIs = new InternalASTKeyword(kwis, "Is");
4482     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
4483     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrbr, "RightBracket");
4484     internalASTIsExpression = new InternalASTIsExpressionIsName(keywordIs, internalASTName,
4485       keywordLeftBracket, internalASTExpression, keywordRightBracket);
4486   }
4487 )
4488 |
4489 (
4490   internalASTIsBasicType = overtureIsBasicType
4491   kwlb2:LBRACKET
4492   internalASTExpression = overtureExpression
4493   kwrbr2:RBRACKET
4494   {
4495     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb2, "LeftBracket");
4496     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrbr2, "RightBracket");
4497     internalASTIsExpression = new InternalASTIsExpressionIsBasicType(internalASTIsBasicType,
4498       keywordLeftBracket, internalASTExpression, keywordRightBracket);
4499   }
4500 )
4501 ;
4502 //type judgement = 'is_', '(' , expression , ',' , type , ')' ;
4503 overtureTypeJudgement returns [InternalASTTypeJudgement internalASTTypeJudgement]
4504 {
4505   internalASTTypeJudgement = null;
4506   InternalASTExpression internalASTExpression = null;
4507   InternalASTType internalASTType = null;
4508 }
4509 :
4510 kwis:ISU
4511 kwlb:LBRACKET
4512 internalASTExpression = overtureExpression
4513 kwcomma:COMMA
4514 internalASTType = overtureType
4515 kwrbr:RBRACKET
4516 {
4517   InternalASTKeyword keywordIs = new InternalASTKeyword(kwis, "Is");
4518   InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
4519   InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
4520   InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrbr, "RightBracket");
4521   internalASTTypeJudgement = new InternalASTTypeJudgement(keywordIs, keywordLeftBracket,
4522     internalASTExpression, keywordComma, internalASTType, keywordRightBracket);
4523 }
4524 ;
4525 //undefined expression = 'undefined' ;
4526 overtureUndefinedExpression returns [InternalASTUndefinedExpression
4527   internalASTUndefinedExpression]
4528 {
4529   internalASTUndefinedExpression = null;
4530 }
4531 :
4532   kwundefined:UNDEFINED
4533   {
4534     InternalASTKeyword keywordUndefined = new InternalASTKeyword(kwundefined, "Undefined");
4535     internalASTUndefinedExpression = new InternalASTUndefinedExpression(keywordUndefined);
4536   }
4537 ;
4538 //isofbaseclass expression = 'isofbaseclass', '(' , name , expression , ')' ;
4539 overtureIsofbaseclassExpression returns [InternalASTIsofbaseclassExpression
4540   internalASTIsofbaseclassExpression]
4541 {
4542   internalASTIsofbaseclassExpression = null;
4543   InternalASTName name = null;
4544   InternalASTExpression expression = null;
4545 }
4546 :
4547   isobc:ISOBASECLASS
4548   lb:LBRACKET
4549   name = overtureName
4550   expression = overtureExpression
4551   rb:RBRACKET
4552 {
4553   InternalASTKeyword keywordIsofbaseclass = new InternalASTKeyword(isobc, "Isofbaseclass");

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4552     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
4553     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
4554     internalASTIsOfClassExpression = new InternalASTIsOfClassExpression(
        keywordIsOfClass, keywordLeftBracket, name, expression, keywordRightBracket);
4555 }
4556 ;
4557
4558 //isOfClass expression = 'isOfClass', '(' , name, expression, ') ' ;
4559 overtureIsOfClassExpression returns [InternalASTIsOfClassExpression
    internalASTIsOfClassExpression]
4560 {
4561     internalASTIsOfClassExpression = null;
4562     InternalASTName name = null;
4563     InternalASTExpression expression = null;
4564 }
4565 :
4566     isoc:ISOFCCLASS
4567     lb:LBRACKET
4568     name = overtureName
4569     expression = overtureExpression
4570     rb:RBRACKET
4571     {
4572         InternalASTKeyword keywordIsOfClass = new InternalASTKeyword(isoc, "IsOfClass");
4573         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
4574         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
4575         internalASTIsOfClassExpression = new InternalASTIsOfClassExpression(keywordIsOfClass,
            keywordLeftBracket, name, expression, keywordRightBracket);
4576     }
4577 ;
4578
4579 //samebaseclass expression = 'samebaseclass', '(' , expression, expression, ') ' ;
4580 overtureSamebaseclassExpression returns [InternalASTSamebaseclassExpression
    internalASTSamebaseclassExpression]
4581 {
4582     internalASTSamebaseclassExpression = null;
4583     InternalASTName name = null;
4584     InternalASTExpression expression1 = null;
4585     InternalASTExpression expression2 = null;
4586 }
4587 :
4588     sa:SAMEBASECLASS
4589     lb:LBRACKET
4590     expression1 = overtureExpression
4591     expression2 = overtureExpression
4592     rb:RBRACKET
4593     {
4594         InternalASTKeyword keywordSamebaseclass = new InternalASTKeyword(sa, "Samebaseclass");
4595         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
4596         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
4597         internalASTSamebaseclassExpression = new InternalASTSamebaseclassExpression(
            keywordSamebaseclass, keywordLeftBracket, expression1, expression2, keywordRightBracket);
4598     }
4599 ;
4600
4601 //sameclass expression = 'sameclass', '(' , expression, expression, ') ' ;
4602 overtureSameclassExpression returns [InternalASTSameclassExpression
    internalASTSameclassExpression]
4603 {
4604     internalASTSameclassExpression = null;
4605     InternalASTName name = null;
4606     InternalASTExpression expression1 = null;
4607     InternalASTExpression expression2 = null;
4608 }
4609 :
4610     sa:SAMECLASS
4611     lb:LBRACKET
4612     expression1 = overtureExpression
4613     expression2 = overtureExpression
4614     rb:RBRACKET
4615     {
4616         InternalASTKeyword keywordSameclass = new InternalASTKeyword(sa, "Sameclass");
4617         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
4618         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
4619         internalASTSameclassExpression = new InternalASTSameclassExpression(keywordSameclass,
            keywordLeftBracket, expression1, expression2, keywordRightBracket);
4620     }
4621 ;
4622
4623 //act expression = '#act', '(' , name, ') '
4624 // | '#act', '(' , name list, ') ' ;
4625 overtureActExpression returns [InternalASTActExpression internalASTActExpression]
4626 {
4627     internalASTActExpression = null;
4628     InternalASTName name = null;

```

I.1. ANTLR GRAMMAR FILE

```

4629     InternalASTNameList namelist = null;
4630 }
4631 :
4632 (ACT LBRACKET overtureName RBRACKET)
4633 =>{
4634     sa:ACT
4635     lb:LBRACKET
4636     name = overtureName
4637     rb:RBRACKET
4638     {
4639         InternalASTKeyword keywordAct = new InternalASTKeyword(sa, "Act");
4640         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4641         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4642         internalASTActExpression = new InternalASTActExpressionName(keywordAct, keywordLeftBracket,
4643             name, keywordRightBracket);
4644     }
4645 )
4646 |
4647 (
4648     sa2:ACT
4649     lb2:LBRACKET
4650     namelist = overtureNameList
4651     rb2:RBRACKET
4652     {
4653         InternalASTKeyword keywordAct = new InternalASTKeyword(sa2, "Act");
4654         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
4655         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
4656         internalASTActExpression = new InternalASTActExpressionNameList(keywordAct,
4657             keywordLeftBracket, namelist, keywordRightBracket);
4658     }
4659 )
4660 ;
4661 //fin expression = '#fin', '(' , name, ') '
4662 // | '#fin', '(' , name list, ') ' ;
4663 overtureFinExpression returns [InternalASTFinExpression internalASTFinExpression]
4664 {
4665     internalASTFinExpression = null;
4666     InternalASTName name = null;
4667     InternalASTNameList namelist = null;
4668 }
4669 :
4670 (FIN LBRACKET overtureName RBRACKET)
4671 =>{
4672     sa:FIN
4673     lb:LBRACKET
4674     name = overtureName
4675     rb:RBRACKET
4676     {
4677         InternalASTKeyword keywordFin = new InternalASTKeyword(sa, "Fin");
4678         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4679         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4680         internalASTFinExpression = new InternalASTFinExpressionName(keywordFin, keywordLeftBracket,
4681             name, keywordRightBracket);
4682     }
4683 )
4684 |
4685 (
4686     sa2:FIN
4687     lb2:LBRACKET
4688     namelist = overtureNameList
4689     rb2:RBRACKET
4690     {
4691         InternalASTKeyword keywordFin = new InternalASTKeyword(sa2, "Fin");
4692         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
4693         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
4694         internalASTFinExpression = new InternalASTFinExpressionNameList(keywordFin,
4695             keywordLeftBracket, namelist, keywordRightBracket);
4696     }
4697 )
4698 ;
4699 //active expression = '#active', '(' , name, ') '
4700 // | '#active', '(' , name list, ') ' ;
4701 overtureActiveExpression returns [InternalASTActiveExpression internalASTActiveExpression]
4702 {
4703     internalASTActiveExpression = null;
4704     InternalASTName name = null;
4705     InternalASTNameList namelist = null;
4706 }
4707 :
4708 (ACTIVE LBRACKET overtureName RBRACKET)
4709 =>{
4710     sa:ACTIVE

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4709     lb:LBRACKET
4710     name = overtureName
4711     rb:RBRACKET
4712     {
4713         InternalASTKeyword keywordActive = new InternalASTKeyword(sa, "Active");
4714         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4715         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4716         internalASTActiveExpression = new InternalASTActiveExpressionName(keywordActive,
            keywordLeftBracket, name, keywordRightBracket);
4717     }
4718 )
4719 |
4720 (
4721     sa2:ACTIVE
4722     lb2:LBRACKET
4723     namelist = overtureNameList
4724     rb2:RBRACKET
4725     {
4726         InternalASTKeyword keywordActive = new InternalASTKeyword(sa2, "Active");
4727         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
4728         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
4729         internalASTActiveExpression = new InternalASTActiveExpressionNameList(keywordActive,
            keywordLeftBracket, namelist, keywordRightBracket);
4730     }
4731 )
4732 ;
4733
4734 //req expression = '#req', '(, name, )'
4735 // | '#req', '(, name list, )' ;
4736 overtureReqExpression returns [InternalASTReqExpression internalASTReqExpression]
4737 {
4738     internalASTReqExpression = null;
4739     InternalASTName name = null;
4740     InternalASTNameList namelist = null;
4741 }
4742 :
4743 (REQ LBRACKET overtureName RBRACKET)
4744 =>(
4745     sa:REQ
4746     lb:LBRACKET
4747     name = overtureName
4748     rb:RBRACKET
4749     {
4750         InternalASTKeyword keywordReq = new InternalASTKeyword(sa, "Req");
4751         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4752         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4753         internalASTReqExpression = new InternalASTReqExpressionName(keywordReq, keywordLeftBracket,
            name, keywordRightBracket);
4754     }
4755 )
4756 |
4757 (
4758     sa2:REQ
4759     lb2:LBRACKET
4760     namelist = overtureNameList
4761     rb2:RBRACKET
4762     {
4763         InternalASTKeyword keywordReq = new InternalASTKeyword(sa2, "Req");
4764         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
4765         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
4766         internalASTReqExpression = new InternalASTReqExpressionNameList(keywordReq,
            keywordLeftBracket, namelist, keywordRightBracket);
4767     }
4768 )
4769 ;
4770
4771 //waiting expression = '#waiting', '(, name, )'
4772 // | '#waiting', '(, name list, )' ;
4773 overtureWaitingExpression returns [InternalASTWaitingExpression internalASTWaitingExpression]
4774 {
4775     internalASTWaitingExpression = null;
4776     InternalASTName name = null;
4777     InternalASTNameList namelist = null;
4778 }
4779 :
4780 (WAITING LBRACKET overtureName RBRACKET)
4781 =>(
4782     sa:WAITING
4783     lb:LBRACKET
4784     name = overtureName
4785     rb:RBRACKET
4786     {
4787         InternalASTKeyword keywordWaiting = new InternalASTKeyword(sa, "Waiting");
4788         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");

```


I.1. ANTLR GRAMMAR FILE

```
4789     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4790     internalASTWaitingExpression = new InternalASTWaitingExpressionName(keywordWaiting,
        keywordLeftBracket, name, keywordRightBracket);
4791     }
4792   )
4793   |
4794   (
4795     sa2:WAITING
4796     lb2:LBRACKET
4797     namelist = overtureNameList
4798     rb2:RBRACKET
4799     {
4800       InternalASTKeyword keywordWaiting = new InternalASTKeyword(sa2, "Waiting");
4801       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
4802       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
4803       internalASTWaitingExpression = new InternalASTWaitingExpressionNameList(keywordWaiting,
        keywordLeftBracket, namelist, keywordRightBracket);
4804     }
4805   )
4806   ;
4807
4808 // name = identifier, [ ''', identifier ] ;
4809 overtureName returns [InternalASTName internalASTName]
4810 {
4811   internalASTName = null;
4812   InternalASTIdentifier identifier1 = null;
4813   InternalASTKeyword keywordMark = null;
4814   InternalASTIdentifier identifier2 = null;
4815 }
4816 :
4817 id1: IDENTIFIER
4818 {
4819   identifier1 = new InternalASTIdentifier(id1);
4820 }
4821 (
4822   ma: MARK
4823   {
4824     keywordMark = new InternalASTKeyword(ma, "Mark");
4825   }
4826   id2: IDENTIFIER
4827   {
4828     identifier2 = new InternalASTIdentifier(id2);
4829   }
4830 )?
4831 {
4832   internalASTName = new InternalASTName(identifier1, keywordMark, identifier2);
4833 }
4834 ;
4835
4836 // name list = name, { ',', name }
4837 overtureNameList returns [InternalASTNameList internalASTNameList]
4838 {
4839   internalASTNameList = null;
4840   InternalASTName name = null;
4841   List<InternalASTNameListElement> nameListElements = new ArrayList<InternalASTNameListElement>();
4842   InternalASTNameListElement currentElement = null;
4843 }
4844 :
4845 (
4846   name = overtureName
4847   {
4848     currentElement = new InternalASTNameListElement(name);
4849     nameListElements.add(currentElement);
4850   }
4851 )
4852 (
4853   co:COMMA
4854   name = overtureName
4855   {
4856     InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
4857     currentElement = new InternalASTNameListElement(keywordComma, name);
4858     nameListElements.add(currentElement);
4859   }
4860 )*
4861 )
4862 {
4863   internalASTNameList = new InternalASTNameList(nameListElements);
4864 }
4865 ;
4866
4867 // old name = identifier, '~' ;
4868 overtureOldName returns [InternalASTOldName internalASTOldName]
4869 {
4870 }
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

4871 | internalASTOldName = null;
4872 | }
4873 | :
4874 | (
4875 |   id:IDENTIFIER
4876 |   ti:TILDE
4877 | )
4878 | {
4879 |   InternalASTIdentifier identifier = new InternalASTIdentifier(id);
4880 |   InternalASTKeyword keywordTilde = new InternalASTKeyword(ti, "Tilde");
4881 |   internalASTOldName = new InternalASTOldName(identifier, keywordTilde);
4882 | }
4883 | ;
4884 |
4885 | // state designator = name | field reference | map or sequence reference ;
4886 | overtureStateDesignator returns [InternalASTStateDesignator internalASTStateDesignator]
4887 | {
4888 |   internalASTStateDesignator = null;
4889 |   InternalASTName name = null;
4890 | }
4891 | :
4892 | (overtureMapOrSequenceReference)=> internalASTStateDesignator = overtureMapOrSequenceReference
4893 | | (overtureFieldReference)=> internalASTStateDesignator = overtureFieldReference
4894 | | name = overtureName
4895 |   {
4896 |     internalASTStateDesignator = new InternalASTStateDesignatorName(name);
4897 |   }
4898 | ;
4899 |
4900 | // field reference = state designator, '.', identifier ;
4901 | overtureFieldReference returns [InternalASTFieldReference internalASTFieldReference]
4902 | {
4903 |   internalASTFieldReference = null;
4904 |   InternalASTStateDesignator statedesignator = null;
4905 |   InternalASTIdentifier identifier = null;
4906 |   InternalASTName name = null;
4907 | }
4908 | :
4909 | (
4910 |   (
4911 |     //(overtureMapOrSequenceReference)=> statedesignator = overtureMapOrSequenceReference
4912 |     name = overtureName
4913 |     {
4914 |       statedesignator = new InternalASTStateDesignatorName(name);
4915 |     }
4916 |   )
4917 |   dot:DOT
4918 |   id:IDENTIFIER
4919 | )
4920 | {
4921 |   identifier = new InternalASTIdentifier(id);
4922 |   InternalASTKeyword keywordDot = new InternalASTKeyword(dot, "Dot");
4923 |   internalASTFieldReference = new InternalASTFieldReference(statedesignator, keywordDot,
4924 |     identifier);
4925 | }
4926 | ;
4927 | // map or sequence reference = state designator, '(', expression, ')' ;
4928 | overtureMapOrSequenceReference returns [InternalASTMapOrSequenceReference
4929 |   internalASTMapOrSequenceReference]
4930 | {
4931 |   internalASTMapOrSequenceReference = null;
4932 |   InternalASTStateDesignator statedesignator = null;
4933 |   InternalASTExpression expression = null;
4934 |   InternalASTName name = null;
4935 | }
4936 | :
4937 | (
4938 |   (
4939 |     (overtureFieldReference)=> statedesignator = overtureFieldReference
4940 |     |
4941 |     name = overtureName
4942 |     {
4943 |       statedesignator = new InternalASTStateDesignatorName(name);
4944 |     }
4945 |   )
4946 |   lb:LBRACKET
4947 |   expression = overtureExpression
4948 |   rb:RBRACKET
4949 | )
4950 | {
4951 |   InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
4952 |   InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
4953 |   internalASTMapOrSequenceReference = new InternalASTMapOrSequenceReference(statedesignator,

```

I.1. ANTLR GRAMMAR FILE

```

keywordLeftBracket , expression , keywordRightBracket );
4953     }
4954     ;
4955
4956 // STATEMENT
4957 // statement = let statement
4958 // | let be statement
4959 // | def statement
4960 // | block statement
4961 // | general assign statement
4962 // | if statement
4963 // | cases statement
4964 // | sequence for loop
4965 // | set for loop
4966 // | index for loop
4967 // | while loop
4968 // | nondeterministic statement
4969 // | call statement
4970 // | specification statement
4971 // | start statement
4972 // | start list statement
4973 // | return statement
4974 // | always statement
4975 // | trap statement
4976 // | recursive trap statement
4977 // | exit statement
4978 // | error statement
4979 // | identity statement ;
4980
4981 overtureStatement returns [InternalASTStatement internalASTStatement]
4982 {
4983     internalASTStatement = null;
4984 }
4985 :
4986 (
4987     (overtureGeneralAssignStatement)=> internalASTStatement = overtureGeneralAssignStatement
4988     | (overtureLetBeStatement)=> internalASTStatement = overtureLetBeStatement
4989     | internalASTStatement = overtureLetStatement
4990     | internalASTStatement = overtureDefStatement
4991     | internalASTStatement = overtureBlockStatement
4992     | internalASTStatement = overtureIfStatement
4993     | internalASTStatement = overtureCasesStatement
4994     | (overtureSequenceForLoop)=> internalASTStatement = overtureSequenceForLoop
4995     | internalASTStatement = overtureSetForLoop
4996     | internalASTStatement = overtureIndexForLoop
4997     | internalASTStatement = overtureWhileLoop
4998     | internalASTStatement = overtureNondeterministicStatement
4999     | internalASTStatement = overtureCallStatement
5000     | internalASTStatement = overtureSpecificationStatement
5001     | internalASTStatement = overtureStartStatement
5002     | internalASTStatement = overtureStartListStatement
5003     | internalASTStatement = overtureReturnStatement
5004     | internalASTStatement = overtureAlwaysStatement
5005     | internalASTStatement = overtureTrapStatement
5006     | internalASTStatement = overtureRecursiveTrapStatement
5007     | internalASTStatement = overtureExitStatement
5008     | internalASTStatement = overtureErrorStatement
5009     | internalASTStatement = overtureIdentityStatement
5010 )
5011 ;
5012
5013 //let statement = 'let', local definition, { ',', local definition },
5014 // 'in', statement ;
5015 overtureLetStatement returns [InternalASTLetStatement internalASTLetStatement]
5016 {
5017     internalASTLetStatement = null;
5018     List<InternalASTLetStatementElement> letStatementElements = new ArrayList<
5019         InternalASTLetStatementElement>();
5020     InternalASTLetStatementElement currentElement = null;
5021     InternalASTKeyword keywordComma = null;
5022     InternalASTStatement internalASTStatement = null;
5023     InternalASTLocalDefinition internalASTLocalDefinition = null;
5024 }
5025 :
5026     klet:LET
5027     internalASTLocalDefinition = overtureLocalDefinition
5028     {
5029         currentElement = new InternalASTLetStatementElement(internalASTLocalDefinition);
5030         letStatementElements.add(currentElement);
5031     }
5032     (
5033         kwcomma:COMMA
5034         internalASTLocalDefinition = overtureLocalDefinition

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5035     {
5036         keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5037         currentElement = new InternalASTLetStatementElement(keywordComma, internalASTLocalDefinition
5038             );
5039         letStatementElements.add(currentElement);
5040     }
5041     )*
5042     kwin:IN
5043     internalASTStatement = overtureStatement
5044     {
5045         InternalASTKeyword keywordLet = new InternalASTKeyword(klet, "Let");
5046         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin, "In");
5047         internalASTLetStatement = new InternalASTLetStatement(keywordLet, letStatementElements,
5048             keywordIn, internalASTStatement);
5049     }
5050 ;
5051 //local definition = value definition
5052 //      | function definition ;
5053 overtureLocalDefinition returns [InternalASTLocalDefinition internalASTLocalDefinition]
5054 {
5055     internalASTLocalDefinition = null;
5056 }
5057 :
5058 (
5059     (overtureFunctionDefinition) => internalASTLocalDefinition = overtureFunctionDefinition
5060     | (overtureValueDefinition) => internalASTLocalDefinition = overtureValueDefinition
5061 )
5062 ;
5063 //let be statement = 'let', bind, [ 'be', 'st', expression ], 'in',
5064 //      statement ;
5065 overtureLetBeStatement returns [InternalASTLetBeStatement internalASTLetBeStatement]
5066 {
5067     internalASTLetBeStatement = null;
5068     InternalASTBind internalASTBind = null;
5069     InternalASTExpression internalASTExpression = null;
5070     InternalASTStatement internalASTStatement = null;
5071     InternalASTKeyword keywordBe = null;
5072     InternalASTKeyword keywordSt = null;
5073 }
5074 :
5075     kwlet:LET
5076     internalASTBind = overtureBind
5077     (
5078         kwbe:BE
5079         kwst:ST
5080         internalASTExpression = overtureExpression
5081         {
5082             keywordBe = new InternalASTKeyword(kwbe, "Be");
5083             keywordSt = new InternalASTKeyword(kwst, "St");
5084         }
5085     )?
5086     kwin:IN
5087     internalASTStatement = overtureStatement
5088     {
5089         InternalASTKeyword keywordLet = new InternalASTKeyword(kwlet, "Let");
5090         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin, "In");
5091         internalASTLetBeStatement = new InternalASTLetBeStatement(keywordLet, internalASTBind,
5092             keywordBe, keywordSt, internalASTExpression, keywordIn, internalASTStatement);
5093     }
5094 ;
5095 // def statement = 'def', equals definition,
5096 //      { ';', equals definition }, [ ';' ],
5097 //      'in', statement ;
5098 overtureDefStatement returns [InternalASTDefStatement internalASTDefStatement]
5099 {
5100     internalASTDefStatement = null;
5101     List<InternalASTDefStatementElement> defStatementElements = new ArrayList<
5102         InternalASTDefStatementElement>();
5103     InternalASTDefStatementElement currentElement = null;
5104     InternalASTEqualsDefinition internalASTEqualsDefinition = null;
5105     InternalASTAccessTypeDefinition internalASTAccessTypeDefinition = null;
5106     InternalASTKeyword keywordDef = null;
5107     InternalASTStatement internalASTStatement = null;
5108 }
5109 :
5110     kwdef:DEF
5111     {
5112         keywordDef = new InternalASTKeyword(kwdef, "Def");
5113     }
5114 (

```

I.1. ANTLR GRAMMAR FILE

```

5115     internalASTEqualsDefinition = overtureEqualsDefinition
5116     {
5117         currentElement = new InternalASTDefStatementElement(internalASTEqualsDefinition);
5118         defStatementElements.add(currentElement);
5119     }
5120
5121     (options {greedy=true};:
5122     seNoN:SEMICOLON
5123     {
5124         InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN, "Semicolon");
5125         currentElement.setKeywordSemicolon(semicolonNoN);
5126         currentElement.setEndPositionFromNode(semicolonNoN);
5127     }
5128     internalASTEqualsDefinition = overtureEqualsDefinition
5129     {
5130         currentElement = new InternalASTDefStatementElement(internalASTEqualsDefinition);
5131         defStatementElements.add(currentElement);
5132     }
5133     )*
5134
5135     (seLast:SEMICOLON
5136     {
5137         InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast, "Semicolon");
5138         currentElement.setKeywordSemicolon(semicolonLast);
5139         currentElement.setEndPositionFromNode(semicolonLast);
5140     }
5141     )?
5142
5143     )?
5144     kwin:IN
5145     internalASTStatement = overtureStatement
5146     {
5147         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin, "In");
5148         internalASTDefStatement = new InternalASTDefStatement(keywordDef, defStatementElements,
5149             keywordIn, internalASTStatement);
5150     }
5151     ;
5152     //equals definition = pattern bind, '=', expression ;
5153     overtureEqualsDefinition returns [InternalASTEqualsDefinition internalASTEqualsDefinition]
5154     {
5155         internalASTEqualsDefinition = null;
5156         InternalASTPatternBind internalASTPatternBind = null;
5157         InternalASTExpression internalASTExpression = null;
5158     }
5159     :
5160     internalASTPatternBind = overturePatternBind
5161     kweq:EQUALSIGN
5162     internalASTExpression = overtureExpression
5163     {
5164         InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq, "Equalsign");
5165         internalASTEqualsDefinition = new InternalASTEqualsDefinition(internalASTPatternBind,
5166             keywordEqualsign, internalASTExpression);
5167     }
5168     ;
5169     // block statement = '(' { dcl statement },
5170     // statement, { ';', statement }, [ ';' ], ')' ;
5171     overtureBlockStatement returns [InternalASTBlockStatement internalASTBlockStatement]
5172     {
5173         internalASTBlockStatement = null;
5174         InternalASTDclStatement internalASTDclStatement = null;
5175         List<InternalASTDclStatement> dclStatements = new ArrayList<InternalASTDclStatement>();
5176         InternalASTBlockStatementElement currentElement = null;
5177         InternalASTStatement internalASTStatement = null;
5178         List<InternalASTBlockStatementElement> blockStatementElements = new ArrayList<
5179             InternalASTBlockStatementElement>();
5180     }
5181     :
5182     kwlb:LBRACKET
5183     (
5184         internalASTDclStatement = overtureDclStatement
5185         {
5186             dclStatements.add(internalASTDclStatement);
5187         }
5188     )*
5189     (
5190         internalASTStatement = overtureStatement
5191         {
5192             currentElement = new InternalASTBlockStatementElement(internalASTStatement);
5193             blockStatementElements.add(currentElement);
5194         }
5195     )
5196     (options {greedy=true};:

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5196     seNoN:SEMICOLON
5197     {
5198         InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN, "Semicolon");
5199         currentElement.setKeywordSemicolon(semicolonNoN);
5200         currentElement.setEndPositionFromNode(semicolonNoN);
5201     }
5202     internalASTStatement = overtureStatement
5203     {
5204         currentElement = new InternalASTBlockStatementElement(internalASTStatement);
5205         blockStatementElements.add(currentElement);
5206     }
5207     )*
5208
5209     (seLast:SEMICOLON
5210     {
5211         InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast, "Semicolon");
5212         currentElement.setKeywordSemicolon(semicolonLast);
5213         currentElement.setEndPositionFromNode(semicolonLast);
5214     }
5215     )?
5216     )
5217     kwrb:RBRACKET
5218     {
5219         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
5220         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
5221         internalASTBlockStatement = new InternalASTBlockStatement(keywordLeftBracket, dclStatements,
5222             blockStatementElements, keywordRightBracket);
5223     }
5224     ;
5225     //dcl statement = 'dcl', assignment definition,
5226     // { ',', assignment definition }, ';' ;
5227     overtureDclStatement returns [InternalASTDclStatement internalASTDclStatement]
5228     {
5229         internalASTDclStatement = null;
5230         InternalASTAssignmentDefinition internalASTAssignmentDefinition = null;
5231         InternalASTDclStatementElement internalASTDclStatementElement = null;
5232         List<InternalASTDclStatementElement> dclStatementElements = new ArrayList<
5233             InternalASTDclStatementElement>();
5234     }
5235     :
5236     kwddl:DCL
5237     internalASTAssignmentDefinition = overtureAssignmentDefinition
5238     {
5239         internalASTDclStatementElement = new InternalASTDclStatementElement(
5240             internalASTAssignmentDefinition);
5241         dclStatementElements.add(internalASTDclStatementElement);
5242     }
5243     (
5244     kwcomma:COMMA
5245     internalASTAssignmentDefinition = overtureAssignmentDefinition
5246     {
5247         InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5248         internalASTDclStatementElement = new InternalASTDclStatementElement(keywordComma,
5249             internalASTAssignmentDefinition);
5250         dclStatementElements.add(internalASTDclStatementElement);
5251     }
5252     )*
5253     kwsc:SEMICOLON
5254     {
5255         InternalASTKeyword keywordDcl = new InternalASTKeyword(kwddl, "Dcl");
5256         InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc, "Semicolon");
5257         internalASTDclStatement = new InternalASTDclStatement(keywordDcl, dclStatementElements,
5258             keywordSemicolon);
5259     }
5260     ;
5261     //assignment definition = identifier, ':', type, [ ':=' , expression ] ;
5262     overtureAssignmentDefinition returns [InternalASTAssignmentDefinition
5263         internalASTAssignmentDefinition]
5264     {
5265         internalASTAssignmentDefinition = null;
5266         InternalASTType internalASTType = null;
5267         InternalASTKeyword keywordColonEqual = null;
5268         InternalASTExpression internalASTExpression = null;
5269     }
5270     :
5271     id:IDENTIFIER
5272     kwcolon:COLON
5273     internalASTType = overtureType
5274     (
5275     kwcolonequal:COLONEQUAL
5276     internalASTExpression = overtureExpression
5277     {

```

I.1. ANTLR GRAMMAR FILE

```

5274         keywordColonEqual = new InternalASTKeyword(kwcolonequal, "ColonEqual");
5275     }
5276 )?
5277 {
5278     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
5279     InternalASTKeyword keywordColon = new InternalASTKeyword(kwcolon, "Colon");
5280     InternalASTAssignmentDefinition = new InternalASTAssignmentDefinition(internalASTIdentifier,
        keywordColon, internalASTType, keywordColonEqual, internalASTExpression);
5281 }
5282 ;
5283
5284 //general assign statement = assign statement
5285 //      | multiple assign statement ;
5286 overtureGeneralAssignStatement returns [InternalASTGeneralAssignStatement
    internalASTGeneralAssignStatement]
5287 {
5288     internalASTGeneralAssignStatement = null;
5289 }
5290 :
5291 (
5292     internalASTGeneralAssignStatement = overtureAssignStatement
5293 | internalASTGeneralAssignStatement = overtureMultipleAssignStatement
5294 )
5295 ;
5296
5297 //assign statement = state designator, ':=', expression ;
5298 overtureAssignStatement returns [InternalASTAssignStatement internalASTAssignStatement]
5299 {
5300     internalASTAssignStatement = null;
5301     InternalASTStateDesignator internalASTStateDesignator = null;
5302     InternalASTExpression internalASTExpression;
5303 }
5304 :
5305     internalASTStateDesignator = overtureStateDesignator
5306     kwcolonequal:COLONEQUAL
5307     internalASTExpression = overtureExpression
5308     {
5309         InternalASTKeyword keywordColonEqual = new InternalASTKeyword(kwcolonequal, "ColonEqual");
5310         internalASTAssignStatement = new InternalASTAssignStatement(internalASTStateDesignator,
            keywordColonEqual, internalASTExpression);
5311     }
5312 ;
5313
5314 //multiple assign statement = 'atomic', '(' assign statement, ';'
5315 //      assign statement,
5316 //      [ {';', assign statement } ], ')' ;
5317 overtureMultipleAssignStatement returns [InternalASTMultipleAssignStatement
    internalASTMultipleAssignStatement]
5318 {
5319     internalASTMultipleAssignStatement = null;
5320     InternalASTAssignStatement internalASTAssignStatement = null;
5321     InternalASTMultipleAssignStatementElement internalASTMultipleAssignStatementElement = null;
5322     List<InternalASTMultipleAssignStatementElement> multipleAssignStatementElements = new ArrayList<
        InternalASTMultipleAssignStatementElement >();
5323     InternalASTExpression internalASTExpression;
5324 }
5325 :
5326     kwatomic:ATOMIC
5327     kwlb:LBACKET
5328     internalASTAssignStatement = overtureAssignStatement
5329     {
5330         internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement (
            internalASTAssignStatement);
5331         multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
5332     }
5333     kwsc:SEMICOLON
5334     internalASTAssignStatement = overtureAssignStatement
5335     {
5336         InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc, "Semicolon");
5337         internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement (
            keywordSemicolon, internalASTAssignStatement);
5338         multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
5339     }
5340 (
5341     kwsc2:SEMICOLON
5342     internalASTAssignStatement = overtureAssignStatement
5343     {
5344         InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc2, "Semicolon");
5345         internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement (
            keywordSemicolon, internalASTAssignStatement);
5346         multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
5347     }
5348 )*
5349     kwrb:RBRACKET

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5350     {
5351         InternalASTKeyword keywordAtomic = new InternalASTKeyword(kwatomic, "Atomic");;
5352         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");;
5353         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");;
5354         internalASTMultipleAssignStatement = new InternalASTMultipleAssignStatement(keywordAtomic,
           keywordLeftBracket, multipleAssignStatementElements, keywordRightBracket);
5355     }
5356 ;
5357
5358 //if statement = 'if', expression, 'then', statement,
5359 //      { elseif statement },
5360 //      [ 'else', statement ] ;
5361 overtureIfStatement returns [InternalASTIfStatement internalASTIfStatement]
5362 {
5363     internalASTIfStatement = null;
5364     InternalASTExpression internalASTExpression = null;
5365     InternalASTStatement internalASTStatement1 = null;
5366     InternalASTElseifStatement internalASTElseifStatement = null;
5367     List<InternalASTElseifStatement> elseifStatements = new ArrayList<InternalASTElseifStatement>();
5368     InternalASTStatement internalASTStatement2 = null;
5369     InternalASTKeyword keywordElse = null;
5370 }
5371 :
5372     kwif:IF
5373     internalASTExpression = overtureExpression
5374     kwthen:THEN
5375     internalASTStatement1 = overtureStatement
5376     (
5377     (options {greedy=true;}:
5378         internalASTElseifStatement = overtureElseifStatement
5379         {
5380             elseifStatements.add(internalASTElseifStatement);
5381         }
5382     )*)
5383     (options {greedy=true;}:
5384     kwelse:ELSE
5385     internalASTStatement2 = overtureStatement
5386     {
5387         keywordElse = new InternalASTKeyword(kwelse, "Else");
5388     }
5389     )?
5390 )
5391 {
5392     InternalASTKeyword keywordIf = new InternalASTKeyword(kwif, "If");;
5393     InternalASTKeyword keywordThen = new InternalASTKeyword(kwthen, "Then");;
5394     internalASTIfStatement = new InternalASTIfStatement(keywordIf, internalASTExpression,
           keywordThen, internalASTStatement1, elseifStatements, keywordElse, internalASTStatement2)
           ;
5395 }
5396 ;
5397
5398 //elseif statement = 'elseif', expression, 'then', statement ;
5399 overtureElseifStatement returns [InternalASTElseifStatement internalASTElseifStatement]
5400 {
5401     internalASTElseifStatement = null;
5402     InternalASTExpression internalASTExpression = null;
5403     InternalASTStatement internalASTStatement = null;
5404 }
5405 :
5406     kwelseif:ELSEIF
5407     internalASTExpression = overtureExpression
5408     kwthen:THEN
5409     internalASTStatement = overtureStatement
5410     {
5411         InternalASTKeyword keywordElseif = new InternalASTKeyword(kwelseif, "Elseif");;
5412         InternalASTKeyword keywordThen = new InternalASTKeyword(kwthen, "Then");;
5413         internalASTElseifStatement = new InternalASTElseifStatement(keywordElseif,
           internalASTExpression, keywordThen, internalASTStatement);
5414     }
5415 ;
5416
5417 //cases statement = 'cases', expression, ':',
5418 //      cases statement alternatives,
5419 //      [ ',', others statement ], 'end' ;
5420 overtureCasesStatement returns [InternalASTCasesStatement internalASTCasesStatement]
5421 {
5422     internalASTCasesStatement = null;
5423     InternalASTExpression internalASTExpression = null;
5424     InternalASTCasesStatementAlternatives internalASTCasesStatementAlternatives = null;
5425     InternalASTOthersStatement internalASTOthersStatement = null;
5426     InternalASTKeyword keywordComma = null;
5427 }
5428 :
5429     kwcases:CASES

```


I.1. ANTLR GRAMMAR FILE

```

5430     internalASTExpression = overtureExpression
5431     kwcolon:COLON
5432     internalASTCasesStatementAlternatives = overtureCasesStatementAlternatives
5433     (
5434         kwcomma:COMMA
5435         internalASTOthersStatement = overtureOthersStatement
5436         {
5437             keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5438         }
5439     )?
5440     kwend:END
5441     {
5442         InternalASTKeyword keywordCases = new InternalASTKeyword(kwcases, "Cases");
5443         InternalASTKeyword keywordColon = new InternalASTKeyword(kwcolon, "Colon");
5444         InternalASTKeyword keywordEnd = new InternalASTKeyword(kwend, "End");
5445         internalASTCasesStatement = new InternalASTCasesStatement(keywordCases,
5446             internalASTExpression, keywordColon, internalASTCasesStatementAlternatives,
5447             keywordComma, internalASTOthersStatement, keywordEnd);
5448     }
5449 ;
5450 //cases statement alternatives = cases statement alternative,
5451 //                                { ',', 'cases statement alternative' };
5452 overtureCasesStatementAlternatives returns [InternalASTCasesStatementAlternatives
5453     internalASTCasesStatementAlternatives]
5454 {
5455     internalASTCasesStatementAlternatives = null;
5456     InternalASTCasesStatementAlternative casesStatementAlternative = null;
5457     InternalASTCasesStatementAlternativesElement internalASTCasesStatementAlternativesElement = null
5458     ;
5459     InternalASTKeyword keywordComma = null;
5460     List<InternalASTCasesStatementAlternativesElement> casesStatementAlternatives = new ArrayList<
5461         InternalASTCasesStatementAlternativesElement>();
5462 }
5463 :
5464     casesStatementAlternative = overtureCasesStatementAlternative
5465     {
5466         internalASTCasesStatementAlternativesElement = new
5467             InternalASTCasesStatementAlternativesElement(casesStatementAlternative);
5468         casesStatementAlternatives.add(internalASTCasesStatementAlternativesElement);
5469     }
5470     (
5471         kwcomma:COMMA
5472         casesStatementAlternative = overtureCasesStatementAlternative
5473         {
5474             keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5475             internalASTCasesStatementAlternativesElement = new
5476                 InternalASTCasesStatementAlternativesElement(keywordComma, casesStatementAlternative)
5477             ;
5478             casesStatementAlternatives.add(internalASTCasesStatementAlternativesElement);
5479         }
5480     )*
5481 )
5482 {
5483     internalASTCasesStatementAlternatives = new InternalASTCasesStatementAlternatives(
5484         casesStatementAlternatives);
5485 }
5486 ;
5487 //cases statement alternative = pattern list, '->', statement ;
5488 overtureCasesStatementAlternative returns [InternalASTCasesStatementAlternative
5489     internalASTCasesStatementAlternative]
5490 {
5491     internalASTCasesStatementAlternative = null;
5492     InternalASTPatternList internalASTPatternList = null;
5493     InternalASTStatement internalASTStatement = null;
5494 }
5495 :
5496     internalASTPatternList = overturePatternList
5497     kwarrow:LINEARROW
5498     internalASTStatement = overtureStatement
5499     {
5500         InternalASTKeyword keywordArrow = new InternalASTKeyword(kwarrow, "Arrow");
5501         internalASTCasesStatementAlternative = new InternalASTCasesStatementAlternative(
5502             internalASTPatternList, keywordArrow, internalASTStatement);
5503     }
5504 ;
5505 //others statement = 'others', '->', statement ;
5506 overtureOthersStatement returns [InternalASTOthersStatement internalASTOthersStatement]
5507 {
5508     internalASTOthersStatement = null;
5509     InternalASTStatement internalASTStatement = null;

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5503 }
5504 :
5505     kwothers:OTHERS
5506     kwarrow:LINEARROW
5507     internalASTStatement = overtureStatement
5508     {
5509         InternalASTKeyword keywordOthers = new InternalASTKeyword(kwothers,"Others");
5510         InternalASTKeyword keywordArrow = new InternalASTKeyword(kwarrow,"Arrow");
5511         internalASTOthersStatement = new InternalASTOthersStatement(keywordOthers, keywordArrow,
5512             internalASTStatement);
5513     }
5514 ;
5515 //sequence for loop = 'for', pattern bind, 'in', [ 'reverse' ],
5516 //     expression, 'do', statement ;
5517 overtureSequenceForLoop returns [InternalASTSequenceForLoop internalASTSequenceForLoop]
5518 {
5519     internalASTSequenceForLoop = null;
5520     internalASTPatternBind internalASTPatternBind = null;
5521     InternalASTExpression internalASTExpression = null;
5522     InternalASTStatement internalASTStatement = null;
5523     InternalASTKeyword keywordReverse = null;
5524 }
5525 :
5526     kwfor:FOR
5527     internalASTPatternBind = overturePatternBind
5528     kwin:IN
5529     (
5530         kwreverse:REVERSE
5531         {
5532             keywordReverse = new InternalASTKeyword(kwreverse,"Reverse");
5533         }
5534     )?
5535     internalASTExpression = overtureExpression
5536     kwdo:DO
5537     internalASTStatement = overtureStatement
5538     {
5539         InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor,"For");
5540         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
5541         InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
5542         internalASTSequenceForLoop = new InternalASTSequenceForLoop(keywordFor,
5543             internalASTPatternBind, keywordIn, keywordReverse, internalASTExpression, keywordDo,
5544             internalASTStatement);
5545     }
5546 ;
5547 //set for loop = 'for', 'all', pattern, 'in set', expression,
5548 //     'do', statement ;
5549 overtureSetForLoop returns [InternalASTSetForLoop internalASTSetForLoop]
5550 {
5551     internalASTSetForLoop = null;
5552     InternalASTPattern internalASTPattern = null;
5553     InternalASTExpression internalASTExpression = null;
5554     InternalASTStatement internalASTStatement = null;
5555 }
5556 :
5557     kwfor:FOR
5558     kwall:ALL
5559     internalASTPattern = overturePattern
5560     kwin:IN
5561     kwset:SET
5562     internalASTExpression = overtureExpression
5563     kwdo:DO
5564     internalASTStatement = overtureStatement
5565     {
5566         InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor,"For");
5567         InternalASTKeyword keywordAll = new InternalASTKeyword(kwall,"All");
5568         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
5569         InternalASTKeyword keywordSet = new InternalASTKeyword(kwset,"Set");
5570         InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
5571         internalASTSetForLoop = new InternalASTSetForLoop(keywordFor, keywordAll, internalASTPattern
5572             , keywordIn, keywordSet, internalASTExpression, keywordDo, internalASTStatement);
5573     }
5574 ;
5575 //index for loop = 'for', identifier, '=', expression, 'to', expression,
5576 //     [ 'by', expression ],
5577 //     'do', statement ;
5578 overtureIndexForLoop returns [InternalASTIndexForLoop internalASTIndexForLoop]
5579 {
5580     internalASTIndexForLoop = null;
5581     InternalASTExpression internalASTExpression1 = null;
5582     InternalASTExpression internalASTExpression2 = null;
5583     InternalASTExpression internalASTExpression3 = null;

```

I.1. ANTLR GRAMMAR FILE

```

5583     InternalASTStatement internalASTStatement = null;
5584     InternalASTKeyword keywordBy = null;
5585 }
5586 :
5587     kwfor:FOR
5588     kwid:IDENTIFIER
5589     kweq:EQUALSIGN
5590     internalASTExpression1 = overtureExpression
5591     kwto:TO
5592     internalASTExpression2 = overtureExpression
5593     (
5594         kwby:BY
5595         internalASTExpression3 = overtureExpression
5596         {
5597             keywordBy = new InternalASTKeyword(kwby, "By");
5598         }
5599     )?
5600     kwdo:DO
5601     internalASTStatement = overtureStatement
5602     {
5603         InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor, "For");
5604         InternalASTIdentifier identifier = new InternalASTIdentifier(kwid);
5605         InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq, "Equalsign");
5606         InternalASTKeyword keywordTo = new InternalASTKeyword(kwto, "To");
5607         InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo, "Do");
5608         internalASTIndexForLoop = new InternalASTIndexForLoop(keywordFor, identifier,
5609             keywordEqualsign, internalASTExpression1, keywordTo, internalASTExpression2, keywordBy,
5610             internalASTExpression3, keywordDo, internalASTStatement);
5611     }
5612 ;
5613 //while loop = 'while', expression, 'do', statement ;
5614 overtureWhileLoop returns [InternalASTWhileLoop internalASTWhileLoop]
5615 {
5616     internalASTWhileLoop = null;
5617     InternalASTExpression internalASTExpression = null;
5618     InternalASTStatement internalASTStatement = null;
5619 }
5620 :
5621     kwwhile:WHILE
5622     internalASTExpression = overtureExpression
5623     kwdo:DO
5624     internalASTStatement = overtureStatement
5625     {
5626         InternalASTKeyword keywordWhile = new InternalASTKeyword(kwwhile, "While");
5627         InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo, "Do");
5628         internalASTWhileLoop = new InternalASTWhileLoop(keywordWhile, internalASTExpression,
5629             keywordDo, internalASTStatement);
5630     }
5631 ;
5632 //nondeterministic statement = '//' , '(' , statement ,
5633 // { ',' , statement } , ')' ;
5634 overtureNondeterministicStatement returns [InternalASTNondeterministicStatement
5635     internalASTNondeterministicStatement]
5636 {
5637     internalASTNondeterministicStatement = null;
5638     InternalASTStatement internalASTStatement = null;
5639     InternalASTNondeterministicStatementElement internalASTNondeterministicStatementElement = null;
5640     List<InternalASTNondeterministicStatementElement> nondeterministicStatementElements = new
5641         ArrayList<InternalASTNondeterministicStatementElement>();
5642 }
5643 :
5644     kwvbars:VBARS
5645     kwlb:LBACKET
5646     internalASTStatement = overtureStatement
5647     {
5648         internalASTNondeterministicStatementElement = new
5649             InternalASTNondeterministicStatementElement(internalASTStatement);
5650         nondeterministicStatementElements.add(internalASTNondeterministicStatementElement);
5651     }
5652     (options {greedy=true};
5653     kwcomma:COMMA
5654     internalASTStatement = overtureStatement
5655     {
5656         InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5657         internalASTNondeterministicStatementElement = new
5658             InternalASTNondeterministicStatementElement(keywordComma, internalASTStatement);
5659         nondeterministicStatementElements.add(internalASTNondeterministicStatementElement);
5660     }
5661     )*
5662     kwrb:RBRACKET
5663     {
5664         InternalASTKeyword keywordVBars = new InternalASTKeyword(kwvbars, "VBars");

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5660     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
5661     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
5662     internalASTNondeterministicStatement = new InternalASTNondeterministicStatement(keywordVBars
5663         , keywordLeftBracket , nondeterministicStatementElements , keywordRightBracket);
5664 ;
5665
5666 //call statement = [ object designator , '.' ] ,
5667 // name , '(' , [ expression list ] , ')' ;
5668 overturingCallStatement returns [InternalASTCallStatement internalASTCallStatement]
5669 {
5670     internalASTCallStatement = null;
5671     InternalASTObjectDesignator internalASTObjectDesignator = null;
5672     InternalASTName internalASTName = null;
5673     InternalASTExpressionList internalASTExpressionList = null;
5674     InternalASTKeyword keywordDot = null;
5675 }
5676 :
5677 (
5678     (overtureObjectDesignator DOT overtureName LBRACKET)
5679 =>(
5680     internalASTObjectDesignator = overtureObjectDesignator
5681     kwdot:DOT
5682     {
5683         keywordDot = new InternalASTKeyword(kwdot,"Dot");
5684     }
5685     )
5686     |
5687     ()
5688     )
5689     internalASTName = overtureName
5690     kwlb:LBRACKET
5691     (
5692     internalASTExpressionList = overtureExpressionList
5693     )?
5694     kwrb:RBRACKET
5695     {
5696         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
5697         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
5698         internalASTCallStatement = new InternalASTCallStatement(internalASTObjectDesignator ,
5699             keywordDot , internalASTName , keywordLeftBracket , internalASTExpressionList ,
5700             keywordRightBracket);
5701     }
5702 ;
5703 //object designator = name
5704 // | self expression
5705 // | new expression
5706 // | object field reference
5707 // | object apply ;
5708 overturingObjectDesignator returns [InternalASTObjectDesignator internalASTObjectDesignator]
5709 {
5710     internalASTObjectDesignator = null;
5711     InternalASTName name = null;
5712     InternalASTSelfExpression selfExpression = null;
5713     InternalASTNewExpression newExpression = null;
5714 }
5715 :
5716 (
5717     (overtureName)=> name = overtureName
5718     {
5719         internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
5720     }
5721     |(overtureObjectApply)=> internalASTObjectDesignator = overtureObjectApply
5722     | internalASTObjectDesignator = overtureObjectFieldReference
5723     | newExpression = overtureNewExpression
5724     {
5725         internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(newExpression);
5726     }
5727     | selfExpression = overtureSelfExpression
5728     {
5729         internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(selfExpression
5730             );
5731     }
5732     )
5733 ;
5734 //object field reference = object designator , '.' , identifier ;
5735 overturingObjectFieldReference returns [InternalASTObjectFieldReference
5736     internalASTObjectFieldReference]
5737 {

```

I.1. ANTLR GRAMMAR FILE

```

5739 internalASTObjectFieldReference = null;
5740 InternalASTObjectDesignator internalASTObjectDesignator = null;
5741 InternalASTName name = null;
5742 InternalASTSelfExpression selfExpression = null;
5743 InternalASTNewExpression newExpression = null;
5744 }
5745 :
5746 (
5747 (
5748     name = overtureName
5749     {
5750         internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
5751     }
5752     newExpression = overtureNewExpression
5753     {
5754     internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(
5755     newExpression);
5756      }
5757     selfExpression = overtureSelfExpression
5758     {
5759     internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(
5760     selfExpression);
5761      }
5762 )
5763 )
5764 kwdot:DOT
5765 id:IDENTIFIER
5766 {
5767     InternalASTKeyword keywordDot = new InternalASTKeyword(kwdot,"Dot");
5768     InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
5769     internalASTObjectFieldReference = new InternalASTObjectFieldReference(
5770         internalASTObjectDesignator, keywordDot, internalASTIdentifier);
5771 }
5772 ;
5773 //object apply = object designator, '(', [ expression list ], ')';
5774 overtureObjectApply returns [InternalASTObjectApply internalASTObjectApply]
5775 {
5776     internalASTObjectApply = null;
5777     InternalASTObjectDesignator internalASTObjectDesignator = null;
5778     InternalASTExpressionList internalASTExpressionList = null;
5779     InternalASTName name = null;
5780     InternalASTSelfExpression selfExpression = null;
5781     InternalASTNewExpression newExpression = null;
5782 }
5783 :
5784 (
5785     (overtureObjectFieldReference)=> internalASTObjectDesignator =
5786     overtureObjectFieldReference
5787
5788     | name = overtureName
5789     {
5790         internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
5791     }
5792     newExpression = overtureNewExpression
5793     {
5794     internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(
5795     newExpression);
5796      }
5797     selfExpression = overtureSelfExpression
5798     {
5799     internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(
5800     selfExpression);
5801      }
5802 )
5803 kwlb:LBRACKET
5804 (
5805     internalASTExpressionList = overtureExpressionList
5806 )?
5807 kwrb:RBRACKET
5808 {
5809     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
5810     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
5811     internalASTObjectApply = new InternalASTObjectApply(internalASTObjectDesignator,
5812         keywordLeftBracket, internalASTExpressionList, keywordRightBracket);
5813 }
5814 ;
5815 //return statement = 'return', [ expression ];

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5816 | overtureReturnStatement returns [InternalASTReturnStatement internalASTReturnStatement]
5817 | {
5818 |     internalASTReturnStatement = null;
5819 |     InternalASTExpression internalASTExpression = null;
5820 | }
5821 | :
5822 |     kwreturn:RETURN
5823 |     (
5824 |         internalASTExpression = overtureExpression
5825 |     )?
5826 |     {
5827 |         InternalASTKeyword keywordReturn = new InternalASTKeyword(kwreturn, "Return");
5828 |         internalASTReturnStatement = new InternalASTReturnStatement(keywordReturn,
5829 |             internalASTExpression);
5830 |     }
5831 | ;
5832 | //specification statement = '/', implicit operation body, '/' ;
5833 | overtureSpecificationStatement returns [InternalASTSpecificationStatement
5834 |     internalASTSpecificationStatement]
5835 | {
5836 |     internalASTSpecificationStatement = null;
5837 |     InternalASTImplicitOperationBody internalASTImplicitOperationBody = null;
5838 | }
5839 | :
5840 |     kwlb:LBRACK
5841 |     internalASTImplicitOperationBody = overtureImplicitOperationBody
5842 |     kwrb:RBRACK
5843 |     {
5844 |         InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb, "LeftBrack");
5845 |         InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb, "RightBrack");
5846 |         internalASTSpecificationStatement = new InternalASTSpecificationStatement(keywordLeftBrack,
5847 |             internalASTImplicitOperationBody, keywordRightBrack);
5848 |     }
5849 | ;
5850 | //start statement = 'start', '(', expression, ')' ;
5851 | overtureStartStatement returns [InternalASTStartStatement internalASTStartStatement]
5852 | {
5853 |     internalASTStartStatement = null;
5854 |     InternalASTExpression internalASTExpression = null;
5855 | }
5856 | :
5857 |     kwstart:START
5858 |     kwlb:LBRACKET
5859 |     internalASTExpression = overtureExpression
5860 |     kwrb:RBRACKET
5861 |     {
5862 |         InternalASTKeyword keywordStart = new InternalASTKeyword(kwstart, "Start");
5863 |         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
5864 |         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
5865 |         internalASTStartStatement = new InternalASTStartStatement(keywordStart, keywordLeftBracket,
5866 |             internalASTExpression, keywordRightBracket);
5867 |     }
5868 | ;
5869 | //start list statement = 'startlist', '(', expression, ')' ;
5870 | overtureStartListStatement returns [InternalASTStartListStatement internalASTStartListStatement]
5871 | {
5872 |     internalASTStartListStatement = null;
5873 |     InternalASTExpression internalASTExpression = null;
5874 | }
5875 | :
5876 |     kwstartlist:STARTLIST
5877 |     kwlb:LBRACKET
5878 |     internalASTExpression = overtureExpression
5879 |     kwrb:RBRACKET
5880 |     {
5881 |         InternalASTKeyword keywordStartlist = new InternalASTKeyword(kwstartlist, "Startlist");
5882 |         InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb, "LeftBracket");
5883 |         InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb, "RightBracket");
5884 |         internalASTStartListStatement = new InternalASTStartListStatement(keywordStartlist,
5885 |             keywordLeftBracket, internalASTExpression, keywordRightBracket);
5886 |     }
5887 | ;
5888 | //always statement = 'always', statement, 'in', statement ;
5889 | overtureAlwaysStatement returns [InternalASTAlwaysStatement internalASTAlwaysStatement]
5890 | {
5891 |     internalASTAlwaysStatement = null;
5892 |     InternalASTStatement internalASTStatement1 = null;
5893 |     InternalASTStatement internalASTStatement2 = null;
5894 | }
5895 | :

```

I.1. ANTLR GRAMMAR FILE

```

5895     kqualways:ALWAYS
5896     internalASTStatement1 = overtureStatement
5897     kwin:IN
5898     internalASTStatement2 = overtureStatement
5899     {
5900         InternalASTKeyword keywordAlways = new InternalASTKeyword(kqualways,"Always");
5901         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
5902         internalASTAlwaysStatement = new InternalASTAlwaysStatement(keywordAlways,
5903             internalASTStatement1, keywordIn, internalASTStatement2);
5904     }
5905 ;
5906 //trap statement = 'trap', pattern bind, 'with', statement,
5907 // 'in', statement ;
5908 overtureTrapStatement returns [InternalASTTrapStatement internalASTTrapStatement]
5909 {
5910     internalASTTrapStatement = null;
5911     InternalASTPatternBind internalASTPatternBind = null;
5912     InternalASTStatement internalASTStatement1 = null;
5913     InternalASTStatement internalASTStatement2 = null;
5914 }
5915 :
5916     kwtrap:TRAP
5917     internalASTPatternBind = overturePatternBind
5918     kwwith:WITH
5919     internalASTStatement1 = overtureStatement
5920     kwin:IN
5921     internalASTStatement2 = overtureStatement
5922     {
5923         InternalASTKeyword keywordTrap = new InternalASTKeyword(kwtrap,"Trap");
5924         InternalASTKeyword keywordWith = new InternalASTKeyword(kwwith,"With");
5925         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
5926         internalASTTrapStatement = new InternalASTTrapStatement(keywordTrap, internalASTPatternBind,
5927             keywordWith, internalASTStatement1, keywordIn, internalASTStatement2);
5928     }
5929 ;
5930 //recursive trap statement = 'tixe', traps, 'in', statement ;
5931 overtureRecursiveTrapStatement returns [InternalASTRecursiveTrapStatement
5932     internalASTRecursiveTrapStatement]
5933 {
5934     internalASTRecursiveTrapStatement = null;
5935     InternalASTTraps internalASTTraps = null;
5936     InternalASTStatement internalASTStatement = null;
5937 }
5938 :
5939     kwtixe:TIXE
5940     internalASTTraps = overtureTraps
5941     kwin:IN
5942     internalASTStatement = overtureStatement
5943     {
5944         InternalASTKeyword keywordTixe = new InternalASTKeyword(kwtixe,"Tixe");
5945         InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
5946         internalASTRecursiveTrapStatement = new InternalASTRecursiveTrapStatement(keywordTixe,
5947             internalASTTraps, keywordIn, internalASTStatement);
5948     }
5949 ;
5950 //traps = '{', pattern bind, '|->', statement,
5951 // {',', pattern bind, '|->', statement }, '}' ;
5952 overtureTraps returns [InternalASTTraps internalASTTraps]
5953 {
5954     internalASTTraps = null;
5955     InternalASTPatternBind internalASTPatternBind = null;
5956     InternalASTStatement internalASTStatement = null;
5957     InternalASTTrapsElement internalASTTrapsElement = null;
5958     List<InternalASTTrapsElement> trapsElements = new ArrayList<InternalASTTrapsElement>();
5959 }
5960 :
5961     kwlb:LBACE
5962     internalASTPatternBind = overturePatternBind
5963     kwvbararrow:VBARARROW
5964     internalASTStatement = overtureStatement
5965     {
5966         InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
5967         InternalASTTrapsElement = new InternalASTTrapsElement(internalASTPatternBind,
5968             keywordVBarArrow, internalASTStatement);
5969         trapsElements.add(internalASTTrapsElement);
5970     }
5971 (
5972     kwcomma:COMMA
5973     internalASTPatternBind = overturePatternBind
5974     kwvbararrow2:VBARARROW
5975     internalASTStatement = overtureStatement

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

5974     {
5975         InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma, "Comma");
5976         InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow2, "VBarArrow");
5977         internalASTTrapsElement = new InternalASTTrapsElement(keywordComma,
                    internalASTPatternBind, keywordVBarArrow, internalASTStatement);
                    trapsElements.add(internalASTTrapsElement);
5978     }
5979 }*
5980 kwrbr:RBRACE
5981 {
5982     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(kwlb, "LeftBrace");
5983     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(kwrb, "RightBrace");
5984     internalASTTraps = new InternalASTTraps(keywordLeftBrace, trapsElements, keywordRightBrace);
5985 }
5986 }
5987 ;
5988
5989 //exit statement = 'exit', [ expression ] ;
5990 overtureExitStatement returns [InternalASTExitStatement internalASTExitStatement]
5991 {
5992     internalASTExitStatement = null;
5993     InternalASTExpression internalASTExpression = null;
5994 }
5995 :
5996     kwexit:EXIT
5997     (
5998         internalASTExpression = overtureExpression
5999     )?
6000 {
6001     InternalASTKeyword keywordExit = new InternalASTKeyword(kwexit, "Exit");
6002     internalASTExitStatement = new InternalASTExitStatement(keywordExit, internalASTExpression);
6003 }
6004 ;
6005
6006 //error statement = 'error' ;
6007 overtureErrorStatement returns [InternalASTErrorStatement internalASTErrorStatement]
6008 {
6009     internalASTErrorStatement = null;
6010 }
6011 :
6012     kwerror:ERROR
6013     {
6014         InternalASTKeyword keywordError = new InternalASTKeyword(kwerror, "Error");
6015         internalASTErrorStatement = new InternalASTErrorStatement(keywordError);
6016     }
6017 ;
6018
6019 //identity statement = 'skip' ;
6020 overtureIdentityStatement returns [InternalASTIdentityStatement internalASTIdentityStatement]
6021 {
6022     internalASTIdentityStatement = null;
6023 }
6024 :
6025     kwskip:SKIP
6026     {
6027         InternalASTKeyword keywordSkip = new InternalASTKeyword(kwskip, "Skip");
6028         internalASTIdentityStatement = new InternalASTIdentityStatement(keywordSkip);
6029     }
6030 ;
6031
6032 // PATTERNS AND BINDINGS!!!
6033
6034 overturePattern returns [InternalASTPattern internalASTPattern]
6035 {
6036     internalASTPattern = null;
6037 }
6038 :
6039     (overtureSetUnionPattern)=> internalASTPattern = overtureSetUnionPattern
6040     | internalASTPattern = overturePatternB
6041 ;
6042
6043 // overture pattern - used in left side of pattern union expression
6044 overturePatternB returns [InternalASTPattern internalASTPattern]
6045 {
6046     internalASTPattern = null;
6047 }
6048 :
6049     (overtureSeqConcPattern) => internalASTPattern = overtureSeqConcPattern
6050     | internalASTPattern = overturePatternC
6051 ;
6052
6053 overturePatternC returns [InternalASTPattern internalASTPattern]
6054 {
6055     internalASTPattern = null;
6056 }

```


I.1. ANTLR GRAMMAR FILE

```
6057 :
6058 | internalASTPattern = overturePatternIdentifier
6059 | internalASTPattern = overtureMatchValue
6060 | internalASTPattern = overtureSetEnumPattern
6061 | internalASTPattern = overtureSeqEnumPattern
6062 | internalASTPattern = overtureTuplePattern
6063 | internalASTPattern = overtureRecordPattern
6064 ;
6065
6066 overturePatternIdentifier returns [InternalASTPatternIdentifier internalASTPatternIdentifier]
6067 {
6068   internalASTPatternIdentifier = null;
6069 }
6070 :
6071 id: IDENTIFIER
6072 {
6073   internalASTPatternIdentifier = new InternalASTPatternIdentifierIdentifier(new
6074     InternalASTIdentifier(id));
6075 }
6076 | li: MINUS
6077 {
6078   internalASTPatternIdentifier = new InternalASTPatternIdentifierLine(new InternalASTKeyword(li
6079     , "Minus"));
6080 }
6081 ;
6082
6083 overtureMatchValue returns [InternalASTMatchValue internalASTMatchValue]
6084 {
6085   internalASTMatchValue = null;
6086   InternalASTExpression expression = null;
6087   InternalASTSymbolicLiteral literal = null;
6088 }
6089 :
6090 (
6091   lb:LBRACKET
6092   expression = overtureExpression
6093   rb:RBRACKET
6094   {
6095     InternalASTKeyword leftbracket = new InternalASTKeyword(lb, "LeftBracket");
6096     InternalASTKeyword rightbracket = new InternalASTKeyword(rb, "RightBracket");
6097     internalASTMatchValue = new InternalASTMatchValueExpression(leftbracket, expression,
6098       rightbracket);
6099 }
6100 )
6101 | literal = overtureSymbolicLiteral
6102 {
6103   internalASTMatchValue = new InternalASTMatchValueSymbolicLiteral(literal);
6104 }
6105 ;
6106
6107 overtureSetEnumPattern returns [InternalASTSetEnumPattern internalASTSetEnumPattern]
6108 {
6109   internalASTSetEnumPattern = null;
6110   InternalASTPatternList patternlist = null;
6111 }
6112 :
6113 lb: LBRACE
6114 ( patternlist = overturePatternList )?
6115 rb: RBRACE
6116 {
6117   InternalASTKeyword leftbrace = new InternalASTKeyword(lb, "LeftBrace");
6118   InternalASTKeyword rightbrace = new InternalASTKeyword(rb, "RightBrace");
6119   internalASTSetEnumPattern = new InternalASTSetEnumPattern(leftbrace, patternlist, rightbrace)
6120   ;
6121 }
6122 ;
6123
6124 overtureSetUnionPattern returns [InternalASTSetUnionPattern internalASTSetUnionPattern]
6125 {
6126   internalASTSetUnionPattern = null;
6127   InternalASTPattern pattern1 = null;
6128   InternalASTPattern pattern2 = null;
6129 }
6130 :
6131 pattern1 = overturePatternB
6132 un: UNION
6133 pattern2 = overturePattern
6134 {
6135   InternalASTKeyword keywordunion = new InternalASTKeyword(un, "Union");
6136   internalASTSetUnionPattern = new InternalASTSetUnionPattern(pattern1, keywordunion, pattern2)
6137   ;
6138 }
6139 ;
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

6136 | ;
6137 |
6138 |
6139 | overtureSeqEnumPattern returns [InternalASTSeqEnumPattern internalASTSeqEnumPattern]
6140 | {
6141 |   internalASTSeqEnumPattern = null;
6142 |   InternalASTPatternList patternlist = null;
6143 | }
6144 | :
6145 |   lbb: LBRACK
6146 |   ( patternlist = overturePatternList )?
6147 |   rbb: RBRACK
6148 |   {
6149 |     InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lbb, "LeftBrack");
6150 |     InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rbb, "RightBrack");
6151 |     internalASTSeqEnumPattern = new InternalASTSeqEnumPattern(keywordLeftBrack, patternlist,
6152 |       keywordRightBrack);
6153 |   }
6154 | ;
6155 | overtureSeqConcPattern returns [InternalASTSeqConcPattern internalASTSeqConcPattern]
6156 | {
6157 |   internalASTSeqConcPattern = null;
6158 |   InternalASTPattern pattern1 = null;
6159 |   InternalASTPattern pattern2 = null;
6160 | }
6161 | :
6162 |   pattern1 = overturePatternC
6163 |   ha: HAT
6164 |   pattern2 = overturePatternB
6165 |   {
6166 |     InternalASTKeyword keywordhat = new InternalASTKeyword(ha, "Hat");
6167 |     internalASTSeqConcPattern = new InternalASTSeqConcPattern(pattern1, keywordhat, pattern2);
6168 |   }
6169 | ;
6170 |
6171 | overtureTuplePattern returns [InternalASTTuplePattern internalASTTuplePattern]
6172 | {
6173 |   internalASTTuplePattern = null;
6174 |   InternalASTPattern pattern = null;
6175 |   InternalASTPatternList patternlist = null;
6176 | }
6177 | :
6178 |   mk: MAKE
6179 |   lb: LBRACKET
6180 |   pattern = overturePattern
6181 |   co: COMMA
6182 |   patternlist = overturePatternList
6183 |   rb: RBRACKET
6184 |   {
6185 |     InternalASTKeyword keywordMk = new InternalASTKeyword(mk, "Mk");
6186 |     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
6187 |     InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
6188 |     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
6189 |     internalASTTuplePattern = new InternalASTTuplePattern(keywordMk, keywordLeftBracket, pattern,
6190 |       keywordComma, patternlist, keywordRightBracket);
6191 |   }
6192 | ;
6193 | overtureRecordPattern returns [InternalASTRecordPattern internalASTRecordPattern]
6194 | {
6195 |   internalASTRecordPattern = null;
6196 |   InternalASTName name = null;
6197 |   InternalASTPatternList patternlist = null;
6198 | }
6199 | :
6200 |   mk: MAKE
6201 |   name = overtureName
6202 |   lb: LBRACKET
6203 |   (
6204 |     patternlist = overturePatternList
6205 |   )?
6206 |   rb: RBRACKET
6207 |   {
6208 |     InternalASTKeyword keywordMk = new InternalASTKeyword(mk, "Mk");
6209 |     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
6210 |     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
6211 |     internalASTRecordPattern = new InternalASTRecordPattern(keywordMk, name, keywordLeftBracket,
6212 |       patternlist, keywordRightBracket);
6213 |   }
6214 | ;
6215 | overturePatternList returns [InternalASTPatternList internalASTPatternList]
6216 | {

```

I.1. ANTLR GRAMMAR FILE

```
6217 internalASTPatternList = null;
6218 List<InternalASTPatternListElement> patternListElements = new ArrayList<
        InternalASTPatternListElement>();
6219 InternalASTPatternListElement currentElement = null;
6220 InternalASTPattern currentpattern = null;
6221 }
6222 :
6223 {
6224     currentpattern = overturePattern
6225     {
6226         currentElement = new InternalASTPatternListElement(currentpattern);
6227         patternListElements.add(currentElement);
6228     }
6229     (
6230     co: COMMA
6231     currentpattern = overturePattern
6232     {
6233         InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
6234         currentElement = new InternalASTPatternListElement(keywordComma, currentpattern);
6235         patternListElements.add(currentElement);
6236     }
6237     )*
6238     {
6239         internalASTPatternList = new InternalASTPatternList(patternListElements);
6240     }
6241 )
6242 ;
6243 ;
6244 ;
6245 // BINDINGS
6246
6247 overturePatternBind returns [InternalASTPatternBind internalASTPatternBind]
6248 {
6249     internalASTPatternBind = null;
6250 }
6251 :
6252     (overtureBind)=> internalASTPatternBind = overtureBind
6253     |
6254     internalASTPatternBind = overturePattern
6255 ;
6256
6257 overtureBind returns [InternalASTBind internalASTBind]
6258 {
6259     internalASTBind = null;
6260 }
6261 :
6262     (overtureSetBind)=> internalASTBind = overtureSetBind
6263     |
6264     internalASTBind = overtureTypeBind
6265 ;
6266
6267 overtureSetBind returns [InternalASTSetBind internalASTSetBind]
6268 {
6269     internalASTSetBind = null;
6270     InternalASTPattern pattern = null;
6271     InternalASTExpression expression = null;
6272 }
6273 :
6274     pattern = overturePattern
6275     in: IN
6276     set: SET
6277     expression = overtureExpression
6278     {
6279         InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
6280         InternalASTKeyword keywordSet = new InternalASTKeyword(set, "Set");
6281         internalASTSetBind = new InternalASTSetBind(pattern, keywordIn, keywordSet, expression);
6282     }
6283 ;
6284
6285 overtureTypeBind returns [InternalASTTypeBind internalASTTypeBind]
6286 {
6287     internalASTTypeBind = null;
6288     InternalASTPattern pattern = null;
6289     InternalASTType type = null;
6290 }
6291 :
6292     pattern = overturePattern
6293     co: COLON
6294     type = overtureType
6295     {
6296         InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
6297         internalASTTypeBind = new InternalASTTypeBind(pattern, keywordColon, type);
6298     }
6299 ;
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

6300 | overtureBindList returns [InternalASTBindList internalASTBindList]
6301 | {
6302 |   internalASTBindList = null;
6303 |   List<InternalASTBindListElement> bindListElements = new ArrayList<InternalASTBindListElement>();
6304 |   InternalASTBindListElement currentElement = null;
6305 |   InternalASTMultipleBind multipleBind = null;
6306 | }
6307 | :
6308 |   multipleBind = overtureMultipleBind
6309 |   {
6310 |     {
6311 |       currentElement = new InternalASTBindListElement(multipleBind);
6312 |       bindListElements.add(currentElement);
6313 |     }
6314 |     (
6315 |       co: COMMA
6316 |       multipleBind = overtureMultipleBind
6317 |       {
6318 |         InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
6319 |         currentElement = new InternalASTBindListElement(keywordComma, multipleBind);
6320 |         bindListElements.add(currentElement);
6321 |       }
6322 |     )*
6323 |     {
6324 |       internalASTBindList = new InternalASTBindList(bindListElements);
6325 |     }
6326 |   ;
6327 |
6328 | overtureMultipleBind returns [InternalASTMultipleBind internalASTMultipleBind]
6329 | {
6330 |   internalASTMultipleBind = null;
6331 | }
6332 | :
6333 |   (overtureMultipleTypeBind)=> internalASTMultipleBind = overtureMultipleTypeBind
6334 |   |
6335 |   internalASTMultipleBind = overtureMultipleSetBind
6336 | ;
6337 |
6338 | overtureMultipleSetBind returns [InternalASTMultipleSetBind internalASTMultipleSetBind]
6339 | {
6340 |   internalASTMultipleSetBind = null;
6341 |   InternalASTPatternList patternList = null;
6342 |   InternalASTExpression expression = null;
6343 | }
6344 | :
6345 |   patternList = overturePatternList
6346 |   in: IN
6347 |   set: SET
6348 |   expression = overtureExpression
6349 |   {
6350 |     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
6351 |     InternalASTKeyword keywordSet = new InternalASTKeyword(set, "Set");
6352 |     internalASTMultipleSetBind = new InternalASTMultipleSetBind(patternList, keywordIn,
6353 |       keywordSet, expression);
6354 |   }
6355 | ;
6356 |
6357 | overtureMultipleTypeBind returns [InternalASTMultipleTypeBind internalASTMultipleTypeBind]
6358 | {
6359 |   internalASTMultipleTypeBind = null;
6360 |   InternalASTPatternList patternList = null;
6361 |   InternalASTType type = null;
6362 | }
6363 | :
6364 |   patternList = overturePatternList
6365 |   co: COLON
6366 |   type = overtureType
6367 |   {
6368 |     InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
6369 |     internalASTMultipleTypeBind = new InternalASTMultipleTypeBind(patternList, keywordColon, type
6370 |       );
6371 |   }
6372 | ;
6373 |
6374 | overtureTypeBindList returns [InternalASTTypeBindList internalASTTypeBindList]
6375 | {
6376 |   internalASTTypeBindList = null;
6377 |   List<InternalASTTypeBindListElement> typeBindListElements = new ArrayList<
6378 |     InternalASTTypeBindListElement>();
6379 |   InternalASTTypeBindListElement currentElement = null;
6380 |   InternalASTTypeBind typeBind = null;
6381 | }
6382 | :
6383 |   typeBind = overtureTypeBind

```

```

6381     {
6382         currentElement = new InternalASTTypeBindListElement(typeBind);
6383         typeBindListElements.add(currentElement);
6384     }
6385     (
6386     co: COMMA
6387     typeBind = overtуреTypeBind
6388     {
6389         InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
6390         currentElement = new InternalASTTypeBindListElement(keywordComma, typeBind);
6391         typeBindListElements.add(currentElement);
6392     }
6393     )*
6394     {
6395         internalASTTypeBindList = new InternalASTTypeBindList(typeBindListElements);
6396     }
6397     ;
6398
6399     // MISC
6400
6401     // type variable identifier = '@', identifier;
6402     overtуреTypeVariableIdentifier returns [InternalASTTypeVariableIdentifier
        internalASTTypeVariableIdentifier]
6403     {
6404         internalASTTypeVariableIdentifier = null;
6405         InternalASTKeyword keywordAt = null;
6406     }
6407     :
6408     at: AT
6409     {
6410         keywordAt = new InternalASTKeyword(at, "At");
6411     }
6412     id: IDENTIFIER
6413     {
6414         InternalASTIdentifier identifier = new InternalASTIdentifier(id);
6415         internalASTTypeVariableIdentifier = new InternalASTTypeVariableIdentifier(keywordAt,
            identifier);
6416     }
6417     ;
6418
6419     // is basic type = 'is_', ( 'bool' | 'nat' | 'nat1' | 'int' | 'rat'
6420     // | 'real' | 'char' | 'token' );
6421     overtуреIsBasicType returns [InternalASTIsBasicType internalASTIsBasicType]
6422     {
6423         internalASTIsBasicType = null;
6424     }
6425     :
6426     bo: ISUBOOL
6427     {
6428         InternalASTKeyword keyword = new InternalASTKeyword(bo, "IsUBool");
6429         internalASTIsBasicType = new InternalASTIsBasicTypeBool(keyword);
6430     }
6431     | na: ISUNAT
6432     {
6433         InternalASTKeyword keyword = new InternalASTKeyword(na, "IsUNat");
6434         internalASTIsBasicType = new InternalASTIsBasicTypeNat(keyword);
6435     }
6436     | na1: ISUNAT1
6437     {
6438         InternalASTKeyword keyword = new InternalASTKeyword(na1, "IsUNat1");
6439         internalASTIsBasicType = new InternalASTIsBasicTypeNat1(keyword);
6440     }
6441     | in: ISUINT
6442     {
6443         InternalASTKeyword keyword = new InternalASTKeyword(in, "IsUInt");
6444         internalASTIsBasicType = new InternalASTIsBasicTypeInt(keyword);
6445     }
6446     | ra: ISURAT
6447     {
6448         InternalASTKeyword keyword = new InternalASTKeyword(ra, "IsURat");
6449         internalASTIsBasicType = new InternalASTIsBasicTypeRat(keyword);
6450     }
6451     | re: ISUREAL
6452     {
6453         InternalASTKeyword keyword = new InternalASTKeyword(re, "IsUReal");
6454         internalASTIsBasicType = new InternalASTIsBasicTypeReal(keyword);
6455     }
6456     | ch: ISUCHAR
6457     {
6458         InternalASTKeyword keyword = new InternalASTKeyword(ch, "IsUChar");
6459         internalASTIsBasicType = new InternalASTIsBasicTypeChar(keyword);
6460     }
6461     | to: ISUTOKEN
6462     {

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

6463     InternalASTKeyword keyword = new InternalASTKeyword(to, "IsUToken");
6464     internalASTIsBasicType = new InternalASTIsBasicTypeToken(keyword);
6465 }
6466 ;
6467
6468
6469 overtureSymbolicLiteral returns [InternalASTSymbolicLiteral internalASTSymbolicLiteral]
6470 {
6471     internalASTSymbolicLiteral = null;
6472 }
6473 :
6474     internalASTSymbolicLiteral = overtureNumericLiteral
6475     | internalASTSymbolicLiteral = overtureBooleanLiteral
6476     | internalASTSymbolicLiteral = overtureNilLiteral
6477     | internalASTSymbolicLiteral = overtureCharacterLiteral
6478     | internalASTSymbolicLiteral = overtureTextLiteral
6479     | internalASTSymbolicLiteral = overtureQuoteLiteral
6480 ;
6481
6482 overtureNumericLiteral returns [InternalASTNumericLiteral internalASTNumericLiteral]
6483 {
6484     internalASTNumericLiteral = null;
6485     InternalASTNumeral numeral1 = null;
6486     InternalASTKeyword keywordDot = null;
6487     InternalASTNumeral numeral2 = null;
6488     InternalASTNumeral numeral3 = null;
6489     InternalASTExponent exponent = null;
6490     InternalASTExponentElement1 element1 = null;
6491     InternalASTExponentElement2 element2 = null;
6492 }
6493 :
6494     numeral1 = overtureNumeral
6495     (options {greedy=true};)
6496     dot:DOT
6497     {
6498         keywordDot = new InternalASTKeyword(dot, "Dot");
6499     }
6500     numeral2 = overtureNumeral
6501     )?
6502     (
6503     (
6504         e1:"E"
6505         {
6506             InternalASTKeyword keyword = new InternalASTKeyword(e1, "E");
6507             element1 = new InternalASTExponentLowerCaseE(keyword);
6508         }
6509         | e2:"e"
6510         {
6511             InternalASTKeyword keyword = new InternalASTKeyword(e2, "e");
6512             element1 = new InternalASTExponentUpperCaseE(keyword);
6513         }
6514     )
6515     (
6516         pl:PLUS
6517         {
6518             InternalASTKeyword keyword = new InternalASTKeyword(pl, "Plus");
6519             element2 = new InternalASTExponentPlus(keyword);
6520         }
6521         | mi:MINUS
6522         {
6523             InternalASTKeyword keyword = new InternalASTKeyword(mi, "Minus");
6524             element2 = new InternalASTExponentMinus(keyword);
6525         }
6526     )?
6527     numeral3 = overtureNumeral
6528     {
6529         exponent = new InternalASTExponent(element1, element2, numeral3);
6530     }
6531     )?
6532     {
6533         internalASTNumericLiteral = new InternalASTNumericLiteral(numeral1, keywordDot, numeral2,
6534             exponent);
6535     }
6536 ;
6537
6538 overtureNumeral returns [InternalASTNumeral internalASTNumeral]
6539 {
6540     internalASTNumeral = null;
6541 }
6542 :
6543     nu:NUMERAL
6544     {
6545         internalASTNumeral = new InternalASTNumeral(nu);

```

I.1. ANTLR GRAMMAR FILE

```
6546     }
6547     ;
6548
6549
6550 overtureBooleanLiteral returns [InternalASTBooleanLiteral internalASTBooleanLiteral]
6551 {
6552     internalASTBooleanLiteral = null;
6553 }
6554 :
6555     tr:TRUE
6556     {
6557         internalASTBooleanLiteral = new InternalASTBooleanLiteralTrue(new InternalASTKeyword(tr, "
6558             True"));
6559     }
6560     |
6561     fa:FALSE
6562     {
6563         internalASTBooleanLiteral = new InternalASTBooleanLiteralFalse(new InternalASTKeyword(fa, "
6564             False"));
6565     }
6566     ;
6567
6568 overtureNilLiteral returns [InternalASTNilLiteral internalASTNilLiteral]
6569 {
6570     internalASTNilLiteral = null;
6571 }
6572 :
6573     ni:NIL
6574     {
6575         internalASTNilLiteral = new InternalASTNilLiteral(new InternalASTKeyword(ni, "Nil"));
6576     }
6577     ;
6578
6579 overtureCharacterLiteral returns [InternalASTCharacterLiteral internalASTCharacterLiteral]
6580 {
6581     internalASTCharacterLiteral = null;
6582 }
6583 :
6584     chlit:CHARACTERLITERAL
6585     {
6586         internalASTCharacterLiteral = new InternalASTCharacterLiteral(chlit);
6587     }
6588     ;
6589
6590 overtureTextLiteral returns [InternalASTTextLiteral internalASTTextLiteral]
6591 {
6592     internalASTTextLiteral = null;
6593 }
6594 :
6595     tlit:TEXTLITERAL
6596     {
6597         internalASTTextLiteral = new InternalASTTextLiteral(tlit);
6598     }
6599     ;
6600
6601 overtureQuoteLiteral returns [InternalASTQuoteLiteral internalASTQuoteLiteral]
6602 {
6603     internalASTQuoteLiteral = null;
6604 }
6605 :
6606     qlit:QUOTELITERAL
6607     {
6608         internalASTQuoteLiteral = new InternalASTQuoteLiteral(qlit);
6609     }
6610     ;
6611
6612 //-----
6613 // The Overture Scanner
6614 //-----
6615 class OvertureInternalLexer extends Lexer;
6616
6617 options {
6618     exportVocab=Overture; // call the vocabulary "Overture"
6619     testLiterals=true; // don't automatically test for literals
6620     k=5; // three characters of lookahead
6621     charVocabulary = '\0'..'377';
6622     //charVocabulary='\u0000'..'u7FFE';
6623     // without inlining some bitset tests, couldn't do unicode;
6624     // I need to make ANTLR generate smaller bitsets; see
6625     // bottom of JavaLexer.java
6626     codeGenBitsetTestThreshold=20;
6627     // '\'
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

6628 }
6629
6630 ACT      : "#act" ;
6631 ACTIVE  : "#active" ;
6632 FIN     : "#fin" ;
6633 REQ     : "#req" ;
6634 WAITING : "#waiting";
6635
6636 COLON   : ':' ;
6637 DOUBLECOLON : '::' ;
6638 COLONLINE : ':' ;
6639
6640 ASTERIX : '*' ;
6641 VBAR    : '|' ;
6642 LBRACK  : '[' ;
6643 RBRACK  : ']' ;
6644
6645 LINEARROW : '->' ;
6646
6647 LBRACKET : '(' ;
6648 RBRACKET : ')' ;
6649 LBRACE   : '{' ;
6650 RBRACE   : '}' ;
6651
6652 SEMICOLON : ';' ;
6653 COMMA     : ',' ;
6654 DOTS      : '.' ;
6655 TILDE     : '~' ;
6656 DOT       : '.' ;
6657 DOTSHARP : '#.' ;
6658 COLONEQUAL : ':' ;
6659 VBARS     : '||' ;
6660
6661 EQUALSIGN : '=' ;
6662 NOTEQUAL  : '<' ;
6663 APPROX    : '~=' ;
6664 DOUBLEEQUAL : '==' ;
6665
6666 HAT       : '^' ;
6667
6668 AT        : '@' ;
6669 MARK     : '^' ;
6670 PLUS     : '+' ;
6671 MINUS    : '-' ;
6672
6673 DOUBLEPLUS : '++' ;
6674 ITERATE    : '**' ;
6675
6676 LESSTHANCOLON : '<' ;
6677 LESSTHANLINECOLON : '<-' ;
6678 COLONGREATERTHAN : '>' ;
6679 COLONLINEGREATERTHAN : ':->' ;
6680
6681 PLUSARROW : '+>' ;
6682
6683 SLASH     : '/' ;
6684 BACKSLASH : '\\ ' ;
6685
6686 LESSTHAN : '<' ;
6687 LESSTHANEQUAL : '<=' ;
6688 GREATERTHAN : '>' ;
6689 GREATERTHANEQUAL : '>=' ;
6690 EQUALIMPLY : '==>' ;
6691
6692 LOGICALEQUIVALENCE : '<=>' ;
6693 IMPLY : '=>' ;
6694
6695 VBARARROW : '|->' ;
6696 ANDSIGN   : '&' ;
6697
6698
6699
6700 // an identifier. Note that testLiterals is set to true! This means
6701 // that after we match the rule, we look in the literals table to see
6702 // if it's a literal or really an identifier
6703 IDENTIFIER
6704   options {testLiterals=true;}
6705   :
6706   ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'0'..'9'|'_')*
6707   ;
6708
6709 QUOTELITERAL
6710   :
6711   '<' ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')* '>'

```


I.1. ANTLR GRAMMAR FILE

```

6712 ;
6713
6714 NUMERAL
6715 :
6716   ('0'..'9')+
6717 ;
6718
6719 CHARACTERLITERAL
6720 :
6721   '\',
6722   (
6723     ('a'..'z'|'A'..'Z')
6724     | ESCAPESEQUENCE
6725     | ('<' '='
6726        | '>' '='
6727        | '<' '>'
6728        | '-' '>'
6729        | '+' '>'
6730        | '=' '=' '>'
6731        | '|' '|'
6732        | '=' '>'
6733        | '<' '-' '>'
6734        | '|' '-' '>'
6735        | '<' ':'
6736        | ':' '>'
6737        | '<' '-' ':'
6738        | ':' '-' '>'
6739        | '=' '='
6740        | '*' '*'
6741        | '+' '+'
6742     )
6743   )
6744   '\',
6745 ;
6746
6747
6748
6749
6750 ESCAPESEQUENCE
6751 :
6752   '\\', '\\',
6753   | '\\', 'r',
6754   | '\\', 'n',
6755   | '\\', 't',
6756   | '\\', 'f',
6757   | '\\', 'e',
6758   | '\\', 'a',
6759   | '\\', 'x' '0'..'9' '0'..'9',
6760   | '\\', 'c' ('a'..'z'|'A'..'Z')
6761   | '\\', '0'..'7' '0'..'7' '0'..'7',
6762   | '\\', '\n',
6763   | '\\', '\r',
6764 ;
6765
6766
6767 TEXTLITERAL
6768 :
6769   '\"',
6770   (
6771     ('\n'|'\r')
6772     | ('a'..'z'|'A'..'Z')
6773     | ESCAPESEQUENCE
6774   )*
6775   '\"',
6776 ;
6777
6778
6779
6780 // Single-line comments
6781 SL_COMMENT
6782 :
6783   '-' '-'
6784   | '-' ('\n'|'\r')*
6785 ;
6786
6787 // Whitespace -- ignored
6788 WS
6789 :
6790   options {testLiterals=true;}
6791   :
6792   | '\t'
6793   | '\f'
6794   // handle newlines
6795

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
6796 | ( options {generateAmbigWarnings=false;}
6797 | : "\r\n" // Evil DOS
6798 | | '\r' // Macintosh
6799 | | '\n' // Unix (the right way)
6800 | )
6801 | { newline(); }
6802 | )+
6803 | { _ttype = Token.SKIP; }
6804 | ;
```

I.2 Selected AST Classes and Interfaces

For the kernel, 366 AST classes and 366 AST class interfaces has been created. The classes and interfaces shown in Listing I.10 to Listing I.21 represents all applied techniques used in the rest of the classes and interfaces.

I.2.1 ASTNode

Listing I.2: ASTNode.java located in org.overturetool.eclipse.ast

```

1 package org.overturetool.eclipse.ast;
2
3 import java.util.List;
4
5
6 public interface ASTNode {
7     public void accept(OvertureVisitorOneArg visitor);
8     public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A a);
9     public void getContextInfo();
10    public List<ASTAdditionalInformation> getAdditionalInformation();
11    public int getStartLine();
12    public int getStartColumn();
13    public int getEndLine();
14 }

```

I.2.2 InternalASTNode

Listing I.3: InternalASTNode.java located in org.overturetool.eclipse.internal.ast

```

1 package org.overturetool.eclipse.internal.ast;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import org.jdom.Element;
7 import org.overturetool.eclipse.ast.ASTAdditionalInformation;
8 import org.overturetool.eclipse.ast.ASTNode;
9
10 import antlr.Token;
11
12 public abstract class InternalASTNode implements ASTNode {
13     private int startLine = -1;
14     private int startColumn = -1;
15     private int endLine = -1;
16     private List<ASTAdditionalInformation> astAdditionalInformation = new ArrayList<
17         ASTAdditionalInformation>();
18
19     public InternalASTNode(){
20     }
21
22     public InternalASTNode(Element xmlTok){
23         try{
24             Element xmlPosition = xmlTok.getChild("OMLPosition");
25             setStartLine(new Integer(xmlPosition.getAttributeValue("startLine")));
26             setStartColumn(new Integer(xmlPosition.getAttributeValue("startColumn")));
27             setEndLine(new Integer(xmlPosition.getAttributeValue("endLine")));
28         } catch (NullPointerException e){
29             //NO ACTION! PROBABLY PARSE ERROR!!!
30             System.err.println("nullpointer in InternalASTNode");
31             e.printStackTrace();
32         }
33     }
34
35     public List<ASTAdditionalInformation> getAdditionalInformation() {
36         return astAdditionalInformation;
37     }
38
39     public void setAdditionalInformation(

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

40     List<ASTAdditionalInformation> astAdditionalInformation) {
41     this.astAdditionalInformation = astAdditionalInformation;
42     }
43
44     public int getStartLine() {
45     return startLine;
46     }
47
48     public void setStartLine(int linenr) {
49     startLine = linenr;
50     }
51
52     public int getStartColumn() {
53     return startColumn;
54     }
55
56     public void setStartColumn(int linenr) {
57     startColumn = linenr;
58     }
59
60     public int getEndLine() {
61     return endLine;
62     }
63
64     public void setEndLine(int linenr) {
65     endLine = linenr;
66     }
67
68     public void setPositionsFromXMLToken(Element xmlTok){
69     try{
70     setStartLine(new Integer(xmlTok.getAttributeValue("startLine")));
71     setStartColumn(new Integer(xmlTok.getAttributeValue("startColumn")));
72     setEndLine(new Integer(xmlTok.getAttributeValue("endLine")));
73     } catch (NullPointerException e){
74     //NO ACTION! PROBABLY PARSE ERROR!!!
75     //System.err.println("nullpointer in InternalASTNode1");
76     }
77     }
78
79     public void setStartPositionFromToken(Token tok){
80     try{
81     setStartLine(tok.getLine());
82     setStartColumn(tok.getColumn());
83     } catch (NullPointerException e){
84     //NO ACTION! PROBABLY PARSE ERROR!!!
85     //System.err.println("nullpointer in InternalASTNode2");
86     }
87     }
88
89     public void setStartPositionFromNode(ASTNode node){
90     try {
91     setStartLine(node.getStartLine());
92     setStartColumn(node.getStartColumn());
93     } catch (NullPointerException e){
94     //NO ACTION! PROBABLY PARSE ERROR!!!
95     //System.err.println("nullpointer in InternalASTNode3");
96     }
97     }
98
99     public void setEndPositionFromToken(Token tok){
100    try {
101    setEndLine(tok.getLine());
102    } catch (NullPointerException e){
103    //NO ACTION! PROBABLY PARSE ERROR!!!
104    //System.err.println("nullpointer in InternalASTNode4");
105    }
106    }
107
108    public void setEndPositionFromNode(ASTNode node){
109    try {
110    setEndLine(node.getEndLine());
111    } catch (NullPointerException e){
112    //NO ACTION! PROBABLY PARSE ERROR!!!
113    //System.err.println("nullpointer in InternalASTNode5");
114    }
115    }
116
117    public void setPositionsFromToken(Token tok){
118    try{
119    setStartLine(tok.getLine());
120    setStartColumn(tok.getColumn());
121    setEndLine(tok.getLine());
122    } catch (NullPointerException e){
123    //NO ACTION! PROBABLY PARSE ERROR!!!

```

I.2. SELECTED AST CLASSES AND INTERFACES

```
124     //System.err.println("nullpointer in InternalASTNode6");
125     }
126 }
127
128 public void setPositionsFromNode(ASTNode node){
129     try{
130         setStartLine(node.getStartLine());
131         setStartColumn(node.getStartColumn());
132         setEndLine(node.getEndLine());
133     } catch (NullPointerException e){
134         //NO ACTION! PROBABLY PARSE ERROR!!!
135         //System.err.println("nullpointer in InternalASTNode7");
136     }
137 }
138
139
140
141 public void getContextInfo(){
142 }
143 }
```

I.2.3 ASTNodeWithComments

Listing I.4: ASTNodeWithComments.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;
2
3
4 public interface ASTNodeWithComments extends ASTNode{
5     public ASTComments getComments();
6     public boolean hasComments();
7 }
```

I.2.4 InternalASTNodeWithComments

Listing I.5: InternalASTNodeWithComments.java located in org.overturetool.eclipse.internal.ast

```
1 package org.overturetool.eclipse.internal.ast;
2
3 import org.jdom.Element;
4 import org.overturetool.eclipse.ast.ASTNodeWithComments;
5
6 import antlr.Token;
7
8 public abstract class InternalASTNodeWithComments extends InternalASTNode implements
9     ASTNodeWithComments {
10     InternalASTComments comments = new InternalASTComments();
11
12     public InternalASTNodeWithComments(){
13     }
14
15     public InternalASTNodeWithComments(Element xmlTok){
16         super(xmlTok);
17         try{
18             Element xmlComments = xmlTok.getChild("OMLComments");
19             if(xmlComments!=null) comments.addCommentsFromXMLToken(xmlComments);
20         } catch (NullPointerException e){
21             //NO ACTION! PROBABLY PARSE ERROR!!!
22             System.err.println("nullpointer in InternalASTNodeWithComments");
23         }
24     }
25
26     public InternalASTNodeWithComments(Token tok){
27         setPositionsFromToken(tok);
28         addCommentsFromToken(tok);
29     }
30
31     public void addCommentsFromToken(antlr.CommonHiddenStreamToken token){
32         comments.addCommentsFromToken(token);
33     }
34 }
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
33
34 public void addCommentsFromToken(antlr.Token token){
35     this.addCommentsFromToken((antlr.CommonHiddenStreamToken) token);
36 }
37
38 public InternalASTComments getComments(){
39     return comments;
40 }
41
42 public boolean hasComments(){
43     return (!comments.isEmpty());
44 }
45
46 public InternalASTComment getLastComment(){
47     return comments.getLastComment();
48 }
49
50 }
```

I.2.5 ASTKeyword

Listing I.6: ASTKeyword.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;
2
3 public interface ASTKeyword extends ASTNodeWithComments {
4     public String getValue();
5     public String getID();
6 }
```

I.2.6 InternalASTKeyword

Listing I.7: InternalASTKeyword.java located in org.overturetool.eclipse.internal.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.internal.ast;
5
6 import org.jdom.Element;
7 import org.overturetool.eclipse.ast.ASTKeyword;
8 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
9 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
10
11 import antlr.Token;
12
13 public class InternalASTKeyword extends InternalASTNodeWithComments
14     implements ASTKeyword {
15
16     private String value;
17     private String id;
18
19     public InternalASTKeyword(Token tok, String id){
20         value = tok.getText();
21         setPositionsFromToken(tok);
22         addCommentsFromToken(tok);
23         this.id = id;
24     }
25
26     public InternalASTKeyword(Element xmlTok){
27         super(xmlTok);
28         //id = xmlTok.getName().substring(10, id.length());
29         value = xmlTok.getAttributeValue("value");
30     }
31
32     public String getValue() {
33         return value;
34     }
35
36     public void setValue(String value){
37         this.value = value;
38     }
39 }
```

I.2. SELECTED AST CLASSES AND INTERFACES

```
39 public String getID(){
40     return id;
41 }
42
43 public void setID(String id){
44     this.id = id;
45 }
46
47 public void accept(OvertureVisitorOneArg visitor) {
48     visitor.visit(this);
49 }
50
51 public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
52     visitor.visit(this,obj);
53 }
54 }
55 }
```

I.2.7 ASTIdentifier

Listing I.8: ASTIdentifier.java located in org.overturetool.eclipse.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.ast;
5
6 public interface ASTIdentifier extends ASTNodeWithComments {
7     public String getIdentifierName();
8 }
```

I.2.8 InternalASTIdentifier

Listing I.9: InternalASTIdentifier.java located in org.overturetool.eclipse.internal.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.internal.ast;
5
6 import org.jdom.Element;
7 import org.overturetool.eclipse.ast.ASTIdentifier;
8 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
9 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
10
11 import antlr.Token;
12
13 public class InternalASTIdentifier extends InternalASTNodeWithComments implements
14     ASTIdentifier {
15     private String identifierName;
16
17     public InternalASTIdentifier(Token tok){
18         setIdentifierName(tok.getText());
19         setPositionsFromToken(tok);
20         addCommentsFromToken(tok);
21     }
22
23     public InternalASTIdentifier(Element xmlTok){
24         super(xmlTok);
25         identifierName = xmlTok.getAttributeValue("name");
26     }
27
28     public String getIdentifierName() {
29         return identifierName;
30     }
31
32     public void setIdentifierName(String _identifierName){
33         identifierName = _identifierName;
34     }
35
36
37     public void accept(OvertureVisitorOneArg visitor) {
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
38     visitor.visit(this);
39 }
40
41 public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
42     visitor.visit(this, obj);
43 }
44 }
```

I.2.9 ASTDocument

Listing I.10: ASTDocument.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;
2
3 import java.util.List;
4
5 public interface ASTDocument extends ASTNodeWithComments {
6     List<? extends ASTClass> getClasses();
7 }
```

I.2.10 InternalASTDocument

Listing I.11: InternalASTDocument.java located in org.overturetool.eclipse.internal.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.internal.ast;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 import org.jdom.Element;
10 import org.overturetool.eclipse.ast.ASTDocument;
11 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
12 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
13
14 import antlr.CommonHiddenStreamToken;
15
16 public class InternalASTDocument extends InternalASTNodeWithComments implements ASTDocument {
17     List<InternalASTClass> classes = new ArrayList<InternalASTClass>();
18
19     public InternalASTDocument(List<InternalASTClass> classes){
20         this.classes = classes;
21         if (!classes.isEmpty()) {
22             this.setStartPositionFromNode(classes.get(0));
23             this.setEndPositionFromNode(classes.get(classes.size()-1));
24         }
25     }
26
27     public InternalASTDocument(Element xmlTok){
28         super(xmlTok);
29     }
30
31     public List<InternalASTClass> getClasses(){
32         return classes;
33     }
34
35     public void setClasses(List<InternalASTClass> classes){
36         this.classes = classes;
37     }
38
39     public void addCommentsFromInitialHiddenToken(CommonHiddenStreamToken initialHiddenToken) {
40         comments.addCommentsFromInitialHiddenToken(initialHiddenToken);
41     }
42
43     public void accept(OvertureVisitorOneArg visitor) {
44         visitor.visit(this);
45     }
46
47     public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
48         visitor.visit(this, obj);
49     }
49 }
```


I.2. SELECTED AST CLASSES AND INTERFACES

```
49 }  
50 }
```

I.2.11 ASTClass

Listing I.12: ASTClass.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;  
2  
3 public interface ASTClass extends ASTNode {  
4     public ASTKeyword getKeywordClass();  
5     public ASTIdentifier getIdentifier();  
6     public ASTInheritanceClause getInheritanceClause();  
7     public ASTClassBody getClassBody();  
8     public ASTKeyword getKeywordEnd();  
9     public ASTIdentifier getIdentifierEnd();  
10 }
```

I.2.12 InternalASTClass

Listing I.13: InternalASTClass.java located in org.overturetool.eclipse.internal.ast

```
1 package org.overturetool.eclipse.internal.ast;  
2  
3 import org.jdom.Element;  
4 import org.overturetool.eclipse.ast.ASTClass;  
5 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;  
6 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;  
7  
8 public class InternalASTClass extends InternalASTNode implements ASTClass {  
9     private InternalASTKeyword keywordClass;  
10    private InternalASTIdentifier identifier;  
11    private InternalASTInheritanceClause inheritanceClause;  
12    private InternalASTClassBody classBody;  
13    private InternalASTKeyword keywordEnd;  
14    private InternalASTIdentifier identifierEnd;  
15  
16    public InternalASTClass() {  
17    }  
18  
19    public InternalASTClass(InternalASTKeyword keywordClass, InternalASTIdentifier identifier,  
20                           InternalASTInheritanceClause inheritanceClause, InternalASTClassBody classBody,  
21                           InternalASTKeyword keywordEnd, InternalASTIdentifier identifierEnd) {  
22        this.keywordClass = keywordClass;  
23        this.identifier = identifier;  
24        this.inheritanceClause = inheritanceClause;  
25        this.classBody = classBody;  
26        this.keywordEnd = keywordEnd;  
27        this.identifierEnd = identifierEnd;  
28        this.setStartPositionFromNode(keywordClass);  
29        this.setEndPositionFromNode(identifierEnd);  
30    }  
31  
32    public InternalASTClass(Element xmlTok) {  
33        super(xmlTok);  
34    }  
35  
36    public InternalASTKeyword getKeywordClass() {  
37        return keywordClass;  
38    }  
39  
40    public void setKeywordClass(InternalASTKeyword keywordClass) {  
41        this.keywordClass = keywordClass;  
42    }  
43  
44    public InternalASTIdentifier getIdentifier() {  
45        return identifier;  
46    }  
47  
48    public void setIdentifier(InternalASTIdentifier identifier) {  
49        this.identifier = identifier;  
50    }  
51 }
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
49
50 public InternalASTInheritanceClause getInheritanceClause() {
51     return inheritanceClause;
52 }
53
54 public void setInheritanceClause(InternalASTInheritanceClause inheritanceClause) {
55     this.inheritanceClause = inheritanceClause;
56 }
57
58 public InternalASTClassBody getClassBody() {
59     return classBody;
60 }
61
62 public void setClassBody(InternalASTClassBody classBody) {
63     this.classBody = classBody;
64 }
65
66 public InternalASTKeyword getKeywordEnd() {
67     return keywordEnd;
68 }
69
70 public void setKeywordEnd(InternalASTKeyword keywordEnd) {
71     this.keywordEnd = keywordEnd;
72 }
73
74 public InternalASTIdentifier getIdentifierEnd() {
75     return identifierEnd;
76 }
77
78 public void setIdentifierEnd(InternalASTIdentifier identifierEnd){
79     this.identifierEnd = identifierEnd;
80 }
81
82 public void accept(OvertureVisitorOneArg visitor) {
83     visitor.visit(this);
84 }
85
86 public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
87     visitor.visit(this, obj);
88 }
89
90 }
```

I.2.13 ASTExpression

Listing I.14: ASTExpression.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;
2
3 public interface ASTExpression extends ASTNode{
4
5 }
```

I.2.14 InternalASTExpression

Listing I.15: InternalASTExpression.java located in org.overturetool.eclipse.internal.ast

```
1 package org.overturetool.eclipse.internal.ast;
2
3 import org.jdom.Element;
4 import org.overturetool.eclipse.ast.ASTExpression;
5
6 public abstract class InternalASTExpression extends InternalASTFunctionBody implements
7     ASTExpression {
8
9     public InternalASTExpression() {
10
11
12     }
13
14     public InternalASTExpression(Element xmlTok){
15         super(xmlTok);
16     }
17 }
```

15 }
}

I.2.15 ASTBinaryExpression

Listing I.16: ASTBinaryExpression.java located in org.overturetool.eclipse.ast

```

1  /**
2  *
3  */
4  package org.overturetool.eclipse.ast;
5
6  public interface ASTBinaryExpression extends ASTExpression {
7
8      public ASTExpression getExpression1 ();
9      public ASTBinaryOperator getBinaryOperator ();
10     public ASTExpression getExpression2 ();
11 }

```

I.2.16 InternalASTBinaryExpression

Listing I.17: InternalASTBinaryExpression.java located in org.overturetool.eclipse.internal.ast

```

1  /**
2  *
3  */
4  package org.overturetool.eclipse.internal.ast;
5
6  import org.jdom.Element;
7  import org.overturetool.eclipse.ast.ASTBinaryExpression;
8  import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
9  import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
10
11 public class InternalASTBinaryExpression extends InternalASTExpression implements
12 ASTBinaryExpression {
13
14     private InternalASTExpression expression1;
15     private InternalASTBinaryOperator binaryOperator;
16     private InternalASTExpression expression2;
17
18     public InternalASTBinaryExpression(InternalASTExpression expression1, InternalASTBinaryOperator
19         binaryOperator, InternalASTExpression expression2){
20         this.expression1 = expression1;
21         this.binaryOperator = binaryOperator;
22         this.expression2 = expression2;
23         this.setStartPositionFromNode(expression1);
24         this.setEndPositionFromNode(expression2);
25     }
26
27     public InternalASTBinaryExpression(Element xmlTok) {
28         super(xmlTok);
29     }
30
31     public InternalASTExpression getExpression1 () {
32         return expression1;
33     }
34
35     public void setExpression(InternalASTExpression expression1) {
36         this.expression1 = expression1;
37     }
38
39     public InternalASTBinaryOperator getBinaryOperator () {
40         return binaryOperator;
41     }
42
43     public void setBinaryOperator(InternalASTBinaryOperator binaryOperator) {
44         this.binaryOperator = binaryOperator;
45     }
46
47     public InternalASTExpression getExpression2 () {
48         return expression2;
49     }

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
48     }
49
50     public void setExpression2(InternalASTExpression expression2) {
51         this.expression2 = expression2;
52     }
53
54     public void accept(OvertureVisitorOneArg visitor) {
55         visitor.visit(this);
56     }
57
58     public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
59         visitor.visit(this, obj);
60     }
61 }
```

I.2.17 ASTBinaryOperator

Listing I.18: ASTBinaryOperator.java located in org.overturetool.eclipse.ast

```
1 package org.overturetool.eclipse.ast;
2
3 public interface ASTBinaryOperator extends ASTNode{
4     public ASTKeyword getKeyword();
5 }
6 }
```

I.2.18 InternalASTBinaryOperator

Listing I.19: InternalASTBinaryOperator.java located in org.overturetool.eclipse.internal.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.internal.ast;
5
6 import org.jdom.Element;
7 import org.overturetool.eclipse.ast.ASTBinaryOperator;
8
9 public abstract class InternalASTBinaryOperator extends InternalASTNode implements
10 ASTBinaryOperator {
11
12     private InternalASTKeyword keyword;
13
14     public InternalASTBinaryOperator(InternalASTKeyword keyword) {
15         this.keyword = keyword;
16         this.setPositionsFromNode(keyword);
17     }
18
19     public InternalASTBinaryOperator(Element xmlTok) {
20         super(xmlTok);
21     }
22
23     public InternalASTKeyword getKeyword(){
24         return keyword;
25     }
26
27     public void setKeyword(InternalASTKeyword keyword){
28         this.keyword = keyword;
29     }
30 }
```

I.2.19 ASTBinaryOperatorArithmeticPlus

Listing I.20: ASTBinaryOperatorArithmeticPlus.java located in org.overturetool.eclipse.ast

I.2. SELECTED AST CLASSES AND INTERFACES

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.ast;
5
6 public interface ASTBinaryOperatorArithmeticPlus extends ASTBinaryOperator {
7
8 }
```

I.2.20 InternalASTBinaryOperatorArithmeticPlus

Listing I.21: InternalASTBinaryOperatorArithmeticPlus.java located in org.overturetool.eclipse.internal.ast

```
1 /**
2  *
3  */
4 package org.overturetool.eclipse.internal.ast;
5
6 import org.jdom.Element;
7 import org.overturetool.eclipse.ast.ASTBinaryOperatorArithmeticPlus;
8 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
9 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
10
11 public class InternalASTBinaryOperatorArithmeticPlus extends InternalASTBinaryOperator implements
12 ASTBinaryOperatorArithmeticPlus {
13
14     public InternalASTBinaryOperatorArithmeticPlus(InternalASTKeyword operator) {
15         super(operator);
16     }
17
18     public InternalASTBinaryOperatorArithmeticPlus(Element xmlTok) {
19         super(xmlTok);
20     }
21
22     public void accept(OvertureVisitorOneArg visitor) {
23         visitor.visit(this);
24     }
25
26     public <A> void accept(OvertureVisitorTwoArg<A> visitor, A a) {
27         visitor.visit(this, a);
28     }
29 }
```


I.3 Visitor Interface

Visitor interfaces has been created both with and without arguments of generic types. In Listing I.22 is shown a visitor interface taking one generic argument.

Listing I.22: OvertureVisitorTwoArg.java located in org.overturetool.eclipse.ast

```

1 package org.overturetool.eclipse.ast;
2
3
4 public interface OvertureVisitorTwoArg<A> {
5
6     public void visit(ASTAccessAssignmentDefinition astAccessAssignmentDefinition, A obj);
7     public void visit(ASTAccessFunctionDefinition astAccessFunctionDefinition, A obj);
8     public void visit(ASTAccessOperationDefinition astAccessOperationDefinition, A obj);
9     public void visit(ASTAccessPrivate astPrivate, A obj);
10    public void visit(ASTAccessProtected astProtected, A obj);
11    public void visit(ASTAccessPublic astPublic, A obj);
12    public void visit(ASTAccessTypeDefinition astAccessTypeDefinition, A obj);
13    public void visit(ASTAccessValueDefinition astAccessValueDefinition, A obj);
14    public void visit(ASTActExpressionName astActExpressionName, A obj);
15    public void visit(ASTActExpressionNameList astActExpressionNameList, A obj);
16    public void visit(ASTActiveExpressionName astActiveExpressionName, A obj);
17    public void visit(ASTActiveExpressionNameList astActiveExpressionNameList, A obj);
18    public void visit(ASTAllExpression astAllExpression, A obj);
19    public void visit(ASTAlwaysStatement astAlwaysStatement, A obj);
20    public void visit(ASTApply astApply, A obj);
21    public void visit(ASTAssignmentDefinition astAssignmentDefinition, A obj);
22    public void visit(ASTAssignStatement astAssignStatement, A obj);
23    public void visit(ASTBasicTypeBool astBoolType, A obj);
24    public void visit(ASTBasicTypeChar astCharType, A obj);
25    public void visit(ASTBasicTypeInt astIntType, A obj);
26    public void visit(ASTBasicTypeNat astNatType, A obj);
27    public void visit(ASTBasicTypeNat1 astNat1Type, A obj);
28    public void visit(ASTBasicTypeRat astRatType, A obj);
29    public void visit(ASTBasicTypeReal astRealType, A obj);
30    public void visit(ASTBasicTokenType astTokenType, A obj);
31    public void visit(ASTBinaryExpression astBinaryExpression, A obj);
32    public void visit(ASTBinaryOperatorAnd astBinaryOperatorAnd, A obj);
33    public void visit(ASTBinaryOperatorApprox astBinaryOperatorApprox, A obj);
34    public void visit(ASTBinaryOperatorArithmeticDivide astBinaryOperatorArithmeticDivide, A obj);
35    public void visit(ASTBinaryOperatorArithmeticIntegerDivision
36        astBinaryOperatorArithmeticIntegerDivision, A obj);
37    public void visit(ASTBinaryOperatorArithmeticMinus astBinaryOperatorArithmeticMinus, A obj);
38    public void visit(ASTBinaryOperatorArithmeticMod astBinaryOperatorArithmeticMod, A obj);
39    public void visit(ASTBinaryOperatorArithmeticMultiplication
40        astBinaryOperatorArithmeticMultiplication, A obj);
41    public void visit(ASTBinaryOperatorArithmeticPlus astBinaryOperatorArithmeticPlus, A obj);
42    public void visit(ASTBinaryOperatorArithmeticRem astBinaryOperatorArithmeticRem, A obj);
43    public void visit(ASTBinaryOperatorComposition astBinaryOperatorComposition, A obj);
44    public void visit(ASTBinaryOperatorEqual astBinaryOperatorEqual, A obj);
45    public void visit(ASTBinaryOperatorGreaterThan astBinaryOperatorGreaterThan, A obj);
46    public void visit(ASTBinaryOperatorGreaterThanOrEqual astBinaryOperatorGreaterThanOrEqual, A
47        obj);
48    public void visit(ASTBinaryOperatorImply astBinaryOperatorImply, A obj);
49    public void visit(ASTBinaryOperatorInSet astBinaryOperatorInSet, A obj);
50    public void visit(ASTBinaryOperatorIterate astBinaryOperatorIterate, A obj);
51    public void visit(ASTBinaryOperatorLessThan astBinaryOperatorLessThan, A obj);
52    public void visit(ASTBinaryOperatorLessThanOrEqual astBinaryOperatorLessThanOrEqual, A obj);
53    public void visit(ASTBinaryOperatorLogicalEquivalence astBinaryOperatorLogicalEquivalence, A
54        obj);
55    public void visit(ASTBinaryOperatorMapDomainRestrictBy astBinaryOperatorMapDomainRestrictBy, A
56        obj);
57    public void visit(ASTBinaryOperatorMapDomainRestrictTo astBinaryOperatorMapDomainRestrictTo, A
58        obj);
59    public void visit(ASTBinaryOperatorMapMerge astBinaryOperatorMapMerge, A obj);
60    public void visit(ASTBinaryOperatorMapOrSequenceModify astBinaryOperatorMapOrSequenceModify, A
61        obj);
62    public void visit(ASTBinaryOperatorMapRangeRestrictBy astBinaryOperatorMapRangeRestrictBy, A
63        obj);
64    public void visit(ASTBinaryOperatorMapRangeRestrictTo astBinaryOperatorMapRangeRestrictTo, A
65        obj);
66    public void visit(ASTBinaryOperatorNotEqual astBinaryOperatorNotEqual, A obj);
67    public void visit(ASTBinaryOperatorNotInSet astBinaryOperatorNotInSet, A obj);
68    public void visit(ASTBinaryOperatorOr astBinaryOperatorOr, A obj);
69    public void visit(ASTBinaryOperatorProperSubset astBinaryOperatorProperSubset, A obj);

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

61 public void visit(ASTBinaryOperatorSequenceConcatenate astBinaryOperatorSequenceConcatenate, A
    obj);
62 public void visit(ASTBinaryOperatorSetDifference astBinaryOperatorSetDifference, A obj);
63 public void visit(ASTBinaryOperatorSetIntersection astBinaryOperatorSetIntersection, A obj);
64 public void visit(ASTBinaryOperatorSetUnion astBinaryOperatorSetUnion, A obj);
65 public void visit(ASTBinaryOperatorSubset astBinaryOperatorSubset, A obj);
66 public void visit(ASTBindList astBindList, A obj);
67 public void visit(ASTBindListElement astBindListElement, A obj);
68 public void visit(ASTBlockStatement astBlockStatement, A obj);
69 public void visit(ASTBlockStatementElement astBlockStatementElement, A obj);
70 public void visit(ASTBooleanLiteralFalse astBooleanLiteralFalse, A obj);
71 public void visit(ASTBooleanLiteralTrue astBooleanLiteralTrue, A obj);
72 public void visit(ASTBracketedExpression astBracketedExpression, A obj);
73 public void visit(ASTBracketedType astBracketedType, A obj);
74 public void visit(ASTCallStatement astCallStatement, A obj);
75 public void visit(ASTCasesExpression astCasesExpression, A obj);
76 public void visit(ASTCasesExpressionAlternative astCasesExpressionAlternative, A obj);
77 public void visit(ASTCasesExpressionAlternatives astCasesExpressionAlternatives, A obj);
78 public void visit(ASTCasesExpressionAlternativesElement astCasesExpressionAlternativesElement,
    A obj);
79 public void visit(ASTCasesStatement astCasesStatement, A obj);
80 public void visit(ASTCasesStatementAlternative astCasesStatementAlternative, A obj);
81 public void visit(ASTCasesStatementAlternatives astCasesStatementAlternatives, A obj);
82 public void visit(ASTCasesStatementAlternativesElement astCasesStatementAlternativesElement, A
    obj);
83 public void visit(ASTCharacterLiteral astCharacterLiteral, A obj);
84 public void visit(ASTClass astClass, A obj);
85 public void visit(ASTClassBody astClassBody, A obj);
86 public void visit(ASTComment astComment, A obj);
87 public void visit(ASTComments astComments, A obj);
88 public void visit(ASTCompositeType astCompositeType, A obj);
89 public void visit(ASTDclStatement astDclStatement, A obj);
90 public void visit(ASTDclStatementElement astDclStatementElement, A obj);
91 public void visit(ASTDefExpression astDefExpression, A obj);
92 public void visit(ASTDefExpressionElement astDefExpressionElement, A obj);
93 public void visit(ASTDefStatement astDefStatement, A obj);
94 public void visit(ASTDefStatementElement astDefStatementElement, A obj);
95 public void visit(ASTDiscretionaryTypeBrackets astDiscretionaryTypeBrackets, A obj);
96 public void visit(ASTDiscretionaryTypeType astDiscretionaryTypeType, A obj);
97 public void visit(ASTDocument astDocument, A obj);
98 public void visit(ASTElseifExpression astElseifExpression, A obj);
99 public void visit(ASTElseifStatement astElseifStatement, A obj);
100 public void visit(ASTEqualsDefinition astEqualsDefinition, A obj);
101 public void visit(ASTError astError, A obj);
102 public void visit(ASTErrorList astErrorList, A obj);
103 public void visit(ASTErrorStatement astErrorStatement, A obj);
104 public void visit(ASTExceptions astExceptions, A obj);
105 public void visit(ASTExistsExpression astExistsExpression, A obj);
106 public void visit(ASTExistsUniqueExpression astExistsUniqueExpression, A obj);
107 public void visit(ASTExitStatement astExitStatement, A obj);
108 public void visit(ASTExplicitFunctionDefinition astExplicitFunctionDefinition, A obj);
109 public void visit(ASTExplicitOperationDefinition astExplicitOperationDefinition, A obj);
110 public void visit(ASTExponent astExponent, A obj);
111 public void visit(ASTExponentLowerCaseE astExponentLowerCaseE, A obj);
112 public void visit(ASTExponentMinus astExponentMinus, A obj);
113 public void visit(ASTExponentPlus astExponentPlus, A obj);
114 public void visit(ASTExponentUpperCaseE astExponentUpperCaseE, A obj);
115 public void visit(ASTExpressionList astExpressionList, A obj);
116 public void visit(ASTExpressionListElement astExpressionListElement, A obj);
117 public void visit(ASTExpressionSymbolicLiteral astExpressionSymbolicLiteral, A obj);
118 public void visit(ASTExtendedExplicitFunctionDefinition astExtendedExplicitFunctionDefinition,
    A obj);
119 public void visit(ASTExtendedExplicitOperationDefinition astExtendedExplicitOperationDefinition
    , A obj);
120 public void visit(ASTExternals astExternals, A obj);
121 public void visit(ASTFieldBasic astFieldBasic, A obj);
122 public void visit(ASTFieldColon astFieldColon, A obj);
123 public void visit(ASTFieldColonLine astFieldColonLine, A obj);
124 public void visit(ASTFieldList astFieldList, A obj);
125 public void visit(ASTFieldReference astFieldReference, A obj);
126 public void visit(ASTFieldSelect astFieldSelect, A obj);
127 public void visit(ASTFinExpressionName astFinExpressionName, A obj);
128 public void visit(ASTFinExpressionNameList astFinExpressionNameList, A obj);
129 public void visit(ASTFunctionBodyIsNotYetSpecified astIsNotYetSpecified, A obj);
130 public void visit(ASTFunctionBodyIsSubclassResponsibility astIsSubclassResponsibility, A obj);
131 public void visit(ASTFunctionDefinitions astFunctionDefinitions, A obj);
132 public void visit(ASTFunctionDefinitionsElement astFunctionDefinitionsElement, A obj);
133 public void visit(ASTFunctionTypeInstantiation astFunctionTypeInstantiation, A obj);
134 public void visit(ASTFunctionTypeInstantiationElement astFunctionTypeInstantiationElement, A
    obj);
135 public void visit(ASTGeneralMapType astGeneralMapType, A obj);
136 public void visit(ASTIdentifier astIdentifier, A obj);
137 public void visit(ASTIdentifierTypePair astIdentifierTypePair, A obj);
138 public void visit(ASTIdentifierTypePairList astIdentifierTypePairList, A obj);

```


I.3. VISITOR INTERFACE

```
139 public void visit(ASTIdentifierTypePairListElement astIdentifierTypePairListElement, A obj);
140 public void visit(ASTIdentityStatement astIdentityStatement, A obj);
141 public void visit(ASTIfExpression astIfExpression, A obj);
142 public void visit(ASTIfStatement astIfStatement, A obj);
143 public void visit(ASTImplicitFunctionDefinition astImplicitFunctionDefinition, A obj);
144 public void visit(ASTImplicitOperationBody astImplicitOperationBody, A obj);
145 public void visit(ASTImplicitOperationDefinition astImplicitOperationDefinition, A obj);
146 public void visit(ASTIndexForLoop astIndexForLoop, A obj);
147 public void visit(ASTInheritanceClause astInheritanceClause, A obj);
148 public void visit(ASTInitStatement astInitStatement, A obj);
149 public void visit(ASTInjectiveMapType astInjectiveMapType, A obj);
150 public void visit(ASTInstanceVariableDefinitions astInstanceVariableDefinitions, A obj);
151 public void visit(ASTInstanceVariableDefinitionsElement astInstanceVariableDefinitionsElement,
    A obj);
152 public void visit(ASTInstvarinitExpression astInstvarinitExpression, A obj);
153 public void visit(ASTInvariant astInvariant, A obj);
154 public void visit(ASTInvariantDefinition astInvariantDefinition, A obj);
155 public void visit(ASTInvariantInitialFunction astInvariantInitialFunction, A obj);
156 public void visit(ASTIotaExpression astIotaExpression, A obj);
157 public void visit(ASTIsBasicTypeBool astIsBasicTypeBool, A obj);
158 public void visit(ASTIsBasicTypeChar astIsBasicTypeChar, A obj);
159 public void visit(ASTIsBasicTypeInt astIsBasicTypeInt, A obj);
160 public void visit(ASTIsBasicTypeNat astIsBasicTypeNat, A obj);
161 public void visit(ASTIsBasicTypeNat1 astIsBasicTypeNat1, A obj);
162 public void visit(ASTIsBasicTypeRat astIsBasicTypeRat, A obj);
163 public void visit(ASTIsBasicTypeReal astIsBasicTypeReal, A obj);
164 public void visit(ASTIsBasicTokenType astIsBasicTokenType, A obj);
165 public void visit(ASTIsExpressionIsBasicType astIsExpressionIsBasicType, A obj);
166 public void visit(ASTIsExpressionIsName astIsExpressionIsName, A obj);
167 public void visit(ASTIsOfClassExpression astIsOfClassExpression, A obj);
168 public void visit(ASTIsOfClassExpression astIsOfClassExpression, A obj);
169 public void visit(ASTKeyword astKeyword, A obj);
170 public void visit(ASTLetBeExpression astLetBeExpression, A obj);
171 public void visit(ASTLetExpression astLetExpression, A obj);
172 public void visit(ASTLetExpressionElement astLetExpressionElement, A obj);
173 public void visit(ASTLetBeStatement astLetBeStatement, A obj);
174 public void visit(ASTLetStatement astLetStatement, A obj);
175 public void visit(ASTLetStatementElement astLetStatementElement, A obj);
176 public void visit(ASTLambdaExpression astLambdaExpression, A obj);
177 public void visit(ASTMapComprehension astMapComprehension, A obj);
178 public void visit(ASTMapEnumerationArrow astMapEnumerationArrow, A obj);
179 public void visit(ASTMapEnumerationMaplet astMapEnumerationMaplet, A obj);
180 public void visit(ASTMapEnumerationMapletElement astMapEnumerationMapletElement, A obj);
181 public void visit(ASTMapInverse astMapInverse, A obj);
182 public void visit(ASTMaplet astMaplet, A obj);
183 public void visit(ASTMapOrSequenceReference astMapOrSequenceReference, A obj);
184 public void visit(ASTMatchValueExpression astMatchValueExpression, A obj);
185 public void visit(ASTMatchValueSymbolicLiteral astMatchValueSymbolicLiteral, A obj);
186 public void visit(ASTModeRd astASTModeRd, A obj);
187 public void visit(ASTModeWr astASTModeWr, A obj);
188 public void visit(ASTMultipleAssignStatement astMultipleAssignStatement, A obj);
189 public void visit(ASTMultipleAssignStatementElement astMultipleAssignStatementElement, A obj);
190 public void visit(ASTMultipleSetBind astMultipleSetBind, A obj);
191 public void visit(ASTMultipleTypeBind astMultipleTypeBind, A obj);
192 public void visit(ASTMutexPredicateAll astMutexPredicateAll, A obj);
193 public void visit(ASTMutexPredicateNameList astMutexPredicateNameList, A obj);
194 public void visit(ASTName astName, A obj);
195 public void visit(ASTNameList astNameList, A obj);
196 public void visit(ASTNameListElement astNameListElement, A obj);
197 public void visit(ASTNewExpression astNewExpression, A obj);
198 public void visit(ASTNilLiteral astNilLiteral, A obj);
199 public void visit(ASTNondeterministicStatement astNondeterministicStatement, A obj);
200 public void visit(ASTNondeterministicStatementElement astNondeterministicStatementElement, A
    obj);
201 public void visit(ASTNumeral astNumeral, A obj);
202 public void visit(ASTNumericLiteral astNumericLiteral, A obj);
203 public void visit(ASTObjectApply astObjectApply, A obj);
204 public void visit(ASTObjectDesignatorName astObjectDesignatorName, A obj);
205 public void visit(ASTObjectDesignatorNewExpression astObjectDesignatorNewExpression, A obj);
206 public void visit(ASTObjectDesignatorSelfExpression astObjectDesignatorSelfExpression, A obj);
207 public void visit(ASTObjectFieldReference astObjectFieldReference, A obj);
208 public void visit(ASTOldName astOldName, A obj);
209 public void visit(ASTOperationBodyIsNotYetSpecified astOperationBodyIsNotYetSpecified, A obj);
210 public void visit(ASTOperationDefinitions astOperationDefinitions, A obj);
211 public void visit(ASTOperationDefinitionsElement astOperationDefinitionsElement, A obj);
212 public void visit(ASTOperationType astOperationType, A obj);
213 public void visit(ASTOperationBodyIsSubclassResponsibility
    astOperationBodyIsSubclassResponsibility, A obj);
214 public void visit(ASTOptionalType astOptionalType, A obj);
215 public void visit(ASTOthersExpression astOthersExpression, A obj);
216 public void visit(ASTOthersStatement astOthersStatement, A obj);
217 public void visit(ASTParameters astParameters, A obj);
218 public void visit(ASTParametersList astParametersList, A obj);
219 public void visit(ASTParameterTypes astParameterTypes, A obj);
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

220 public void visit(ASTPartialFunctionType astPartialFunctionType, A obj);
221 public void visit(ASTPatternIdentifierIdentifier astPatternIdentifierIdentifier, A obj);
222 public void visit(ASTPatternIdentifierLine astPatternIdentifierLine, A obj);
223 public void visit(ASTPatternList astPatternList, A obj);
224 public void visit(ASTPatternListElement astPatternListElement, A obj);
225 public void visit(ASTPatternListTypePair astPatternListTypePair, A obj);
226 public void visit(ASTPatternTypePairList astPatternTypePairList, A obj);
227 public void visit(ASTPatternTypePairListElement astPatternTypePairListElement, A obj);
228 public void visit(ASTPermissionPredicateElement1 astPermissionPredicatesElement1, A obj);
229 public void visit(ASTPermissionPredicateElement2 astPermissionPredicatesElement2, A obj);
230 public void visit(ASTPermissionPredicates astPermissionPredicates, A obj);
231 public void visit(ASTPermissionPredicatesElement astPermissionPredicatesElement, A obj);
232 public void visit(ASTPrefixExpression astPrefixExpression, A obj);
233 public void visit(ASTProceduralThreadDefinition astProceduralThreadDefinition, A obj);
234 public void visit(ASTProductType astProductType, A obj);
235 public void visit(ASTQuoteLiteral astQuoteLiteral, A obj);
236 public void visit(ASTQuoteType astQuoteType, A obj);
237 public void visit(ASTRecordConstructor astRecordConstructor, A obj);
238 public void visit(ASTRecordModifier ASTRecordModifier, A obj);
239 public void visit(ASTRecordModifierElement ASTRecordModifierElement, A obj);
240 public void visit(ASTRecordModification ASTRecordModification, A obj);
241 public void visit(ASTRecordPattern astRecordPattern, A obj);
242 public void visit(ASTRecursiveTrapStatement astRecursiveTrapStatement, A obj);
243 public void visit(ASTReqExpressionName astReqExpressionName, A obj);
244 public void visit(ASTReqExpressionNameList astReqExpressionNameList, A obj);
245 public void visit(ASTReturnStatement astReturnStatement, A obj);
246 public void visit(ASTSamebaseclassExpression astSamebaseclassExpression, A obj);
247 public void visit(ASTSameclassExpression astSameclassExpression, A obj);
248 public void visit(ASTSelfExpression astSelfExpression, A obj);
249 public void visit(ASTSeq0Type astSeq0Type, A obj);
250 public void visit(ASTSeq1Type astSeq1Type, A obj);
251 public void visit(ASTSeqConcPattern astSeqConcPattern, A obj);
252 public void visit(ASTSeqEnumPattern astSeqEnumPattern, A obj);
253 public void visit(ASTSequenceComprehension astSequenceComprehension, A obj);
254 public void visit(ASTSequenceEnumeration astSequenceEnumeration, A obj);
255 public void visit(ASTSequenceForLoop astSequenceForLoop, A obj);
256 public void visit(ASTSetBind astSetBind, A obj);
257 public void visit(ASTSetComprehension astSetComprehension, A obj);
258 public void visit(ASTSetEnumeration astSetEnumeration, A obj);
259 public void visit(ASTSetEnumPattern astSetEnumPattern, A obj);
260 public void visit(ASTSetForLoop astSetForLoop, A obj);
261 public void visit(ASTSetRangeExpression astSetRangeExpression, A obj);
262 public void visit(ASTSetType astSetType, A obj);
263 public void visit(ASTSetUnionPattern astSetUnionPattern, A obj);
264 public void visit(ASTSpecificationStatement astSpecificationStatement, A obj);
265 public void visit(ASTStartListStatement astStartListStatement, A obj);
266 public void visit(ASTStartStatement astStartStatement, A obj);
267 public void visit(ASTStateDesignatorName astStateDesignatorName, A obj);
268 public void visit(ASTSubsequence astSubsequence, A obj);
269 public void visit(ASTSynchronization astSynchronization, A obj);
270 public void visit(ASTSynchronizationDefinitions astSynchronizationDefinitions, A obj);
271 public void visit(ASTTextLiteral astTextLiteral, A obj);
272 public void visit(ASTThreadDefinition astThreadDefinition, A obj);
273 public void visit(ASTThreadDefinitions astThreadDefinitions, A obj);
274 public void visit(ASTThreadidExpression astThreadidExpression, A obj);
275 public void visit(ASTTotalFunctionType astTotalFunctionType, A obj);
276 public void visit(ASTTraps astTraps, A obj);
277 public void visit(ASTTrapsElement astTrapsElement, A obj);
278 public void visit(ASTTrapStatement astTrapStatement, A obj);
279 public void visit(ASTTupleConstructor astTupleConstructor, A obj);
280 public void visit(ASTTuplePattern astTuplePattern, A obj);
281 public void visit(ASTTupleSelect astTupleSelect, A obj);
282 public void visit(ASTTypeBind astTypeBind, A obj);
283 public void visit(ASTTypeBindList astTypeBindList, A obj);
284 public void visit(ASTTypeBindListElement astTypeBindListElement, A obj);
285 public void visit(ASTTypeDefinitionFieldList astTypeFieldListDefinition, A obj);
286 public void visit(ASTTypeDefinitions astTypeDefinitions, A obj);
287 public void visit(ASTTypeDefinitionsElement astTypeDefinitionsElement, A obj);
288 public void visit(ASTTypeDefinitionType astTypeTypeDefinition, A obj);
289 public void visit(ASTTypeJudgement ASTTypeJudgement, A obj);
290 public void visit(ASTTypeName astTypeName, A obj);
291 public void visit(ASTTypeVariable astTypeVariable, A obj);
292 public void visit(ASTTypeVariableIdentifier astTypeVariableIdentifier, A obj);
293 public void visit(ASTTypeVariableList astTypeVariableList, A obj);
294 public void visit(ASTTypeVariableListElement astTypeVariableListElement, A obj);
295 public void visit(ASTUnaryOperatorArithmeticAbs astUnaryOperatorArithmeticAbs, A obj);
296 public void visit(ASTUnaryOperatorDistributedMapMerge astUnaryOperatorDistributedMapMerge, A
obj);
297 public void visit(ASTUnaryOperatorDistributedSequenceConcatenation
astUnaryOperatorDistributedSequenceConcatenation, A obj);
298 public void visit(ASTUnaryOperatorDistributedSetIntersection
astUnaryOperatorDistributedSetIntersection, A obj);
299 public void visit(ASTUnaryOperatorDistributedSetUnion astUnaryOperatorDistributedSetUnion, A
obj);

```

I.3. VISITOR INTERFACE

```
300 public void visit(ASTUnaryOperatorFinitePowerSet astUnaryOperatorFinitePowerSet, A obj);
301 public void visit(ASTUnaryOperatorFloor astUnaryOperatorFloor, A obj);
302 public void visit(ASTUnaryOperatorMapDomain astUnaryOperatorMapDomain, A obj);
303 public void visit(ASTUnaryOperatorMapRange astUnaryOperatorMapRange, A obj);
304 public void visit(ASTUnaryOperatorNot astUnaryOperatorNot, A obj);
305 public void visit(ASTUnaryOperatorSequenceElements astUnaryOperatorSequenceElements, A obj);
306 public void visit(ASTUnaryOperatorSequenceHead astUnaryOperatorSequenceHead, A obj);
307 public void visit(ASTUnaryOperatorSequenceIndices astUnaryOperatorSequenceIndices, A obj);
308 public void visit(ASTUnaryOperatorSequenceLength astUnaryOperatorSequenceLength, A obj);
309 public void visit(ASTUnaryOperatorSequenceTail astUnaryOperatorSequenceTail, A obj);
310 public void visit(ASTUnaryOperatorSetCardinality astUnaryOperatorSetCardinality, A obj);
311 public void visit(ASTUnaryOperatorUnaryMinus astUnaryOperatorUnaryMinus, A obj);
312 public void visit(ASTUnaryOperatorUnaryPlus astUnaryOperatorUnaryPlus, A obj);
313 public void visit(ASTUnionType astUnionType, A obj);
314 public void visit(ASTUndefinedExpression astSelfExpression, A obj);
315 public void visit(ASTValueDefinition astValueDefinition, A obj);
316 public void visit(ASTValueDefinitions astValueDefinitions, A obj);
317 public void visit(ASTValueDefinitionsElement astValueDefinitionsElement, A obj);
318 public void visit(ASTVarInformation astVarInformation, A obj);
319 public void visit(ASTWaitingExpressionName astWaitingExpressionName, A obj);
320 public void visit(ASTWaitingExpressionNameList astWaitingExpressionNameList, A obj);
321 public void visit(ASTWhileLoop astWhileLoop, A obj);
322 }
```


I.4 Outline Visitor

Listing I.23: OvertureOutlineVisitor.java located in org.overturetool.eclipse.outline

```

1  /*****
2  * Overturetool
3  * Jens Kielsinggaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.outline;
8
9  import java.util.List;
10
11 import org.overturetool.eclipse.ast.*;
12 import org.overturetool.eclipse.internal.ast.InternalASTDefExpressionElement;
13 import org.overturetool.eclipse.internal.ast.OvertureVisitorTwoArgImpl;
14 import org.overturetool.eclipse.util.TreeParent;
15
16 public class OvertureOutlineVisitor extends OvertureVisitorTwoArgImpl<TreeParent> implements
17     OvertureVisitorTwoArg<TreeParent>{
18
19     public void visit(ASTClass astClass, TreeParent parent) {
20         try {
21             ASTIdentifier astIdentifier = astClass.getIdentifier();
22             TreeParent om1Class = new TreeParent("Class "+astIdentifier.getIdentifierName(), astClass.
23                 getStartLine(), astClass.getStartColumn(), astClass.getEndLine());
24             ASTClassBody classBody = astClass.getClassBody();
25             if(classBody != null){
26                 classBody.accept(this, om1Class);
27             }
28             parent.addChild(om1Class);
29         } catch (Exception e) {
30             e.printStackTrace();
31         }
32     }
33
34     public void visit(ASTExplicitFunctionDefinition astExplicitFunctionDefinition, TreeParent
35         parent){
36         try {
37             ASTIdentifier astIdentifier = astExplicitFunctionDefinition.getIdentifier1();
38             TreeParent om1Identifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
39                 getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
40             parent.addChild(om1Identifier);
41         } catch (Exception e) {
42             e.printStackTrace();
43         }
44     }
45
46     public void visit(ASTExplicitOperationDefinition astExplicitOperationDefinition, TreeParent
47         parent){
48         try {
49             ASTIdentifier astIdentifier = astExplicitOperationDefinition.getIdentifier1();
50             TreeParent om1Identifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
51                 getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
52             parent.addChild(om1Identifier);
53         } catch (Exception e) {
54             e.printStackTrace();
55         }
56     }
57
58     public void visit(ASTExtendedExplicitFunctionDefinition astExtendedExplicitFunctionDefinition,
59         TreeParent parent){
60         try {
61             ASTIdentifier astIdentifier = astExtendedExplicitFunctionDefinition.getIdentifier();
62             TreeParent om1Identifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
63                 getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
64             parent.addChild(om1Identifier);
65         } catch (Exception e) {
66             e.printStackTrace();
67         }
68     }
69
70     public void visit(ASTExtendedExplicitOperationDefinition astExtendedExplicitOperationDefinition
71         , TreeParent parent){
72         try {
73             ASTIdentifier astIdentifier = astExtendedExplicitOperationDefinition.getIdentifier();
74             TreeParent om1Identifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
75                 getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

67     parent.addChild(omlIdentifier);
68   } catch (Exception e) {
69     e.printStackTrace();
70   }
71 }
72
73 public void visit(ASTFunctionDefinitions astFunctionDefinitions, TreeParent parent) {
74   try {
75     TreeParent omlFunctionDefinitions = new TreeParent("functions", astFunctionDefinitions.
76       getStartLine(), astFunctionDefinitions.getStartColumn(), astFunctionDefinitions.
77       getEndLine());
78     for (ASTFunctionDefinitionsElement astFunctionDefinitionsElement : astFunctionDefinitions.
79       getFunctionDefinitionsElements()) {
80       astFunctionDefinitionsElement.accept(this, omlFunctionDefinitions);
81     }
82     parent.addChild(omlFunctionDefinitions);
83   } catch (Exception e) {
84     e.printStackTrace();
85   }
86 }
87
88 public void visit(ASTImplicitFunctionDefinition astImplicitFunctionDefinition, TreeParent
89   parent){
90   try {
91     ASTIdentifier astIdentifier = astImplicitFunctionDefinition.getIdentifier();
92     TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
93       getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
94     parent.addChild(omlIdentifier);
95   } catch (Exception e) {
96     e.printStackTrace();
97   }
98 }
99
100 public void visit(ASTImplicitOperationDefinition astImplicitOperationDefinition, TreeParent
101   parent){
102   try {
103     ASTIdentifier astIdentifier = astImplicitOperationDefinition.getIdentifier();
104     TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
105       getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
106     parent.addChild(omlIdentifier);
107   } catch (Exception e) {
108     e.printStackTrace();
109   }
110 }
111
112 public void visit(ASTInstanceVariableDefinitions astInstanceVariableDefinitions, TreeParent
113   parent) {
114   try {
115     TreeParent omlInstanceVariableDefinitions = new TreeParent("instance variables",
116       astInstanceVariableDefinitions.getStartLine(), astInstanceVariableDefinitions.
117       getStartColumn(), astInstanceVariableDefinitions.getEndLine());
118     for (ASTInstanceVariableDefinitionsElement astInstanceVariableDefinitionsElement :
119       astInstanceVariableDefinitions.getInstanceVariableDefinitionsElement()) {
120       astInstanceVariableDefinitionsElement.accept(this, omlInstanceVariableDefinitions);
121     }
122     parent.addChild(omlInstanceVariableDefinitions);
123   } catch (Exception e) {
124     e.printStackTrace();
125   }
126 }
127
128 public void visit(ASTOperationDefinitions astOperationDefinitions, TreeParent parent){
129   try {
130     TreeParent omlOperationDefinitions = new TreeParent("operations", astOperationDefinitions.
131       getStartLine(), astOperationDefinitions.getStartColumn(), astOperationDefinitions.
132       getEndLine());
133     for (ASTOperationDefinitionsElement astOperationDefinitionsElement :
134       astOperationDefinitions.getOperationDefinitionsElements()) {
135       astOperationDefinitionsElement.accept(this, omlOperationDefinitions);
136     }
137     parent.addChild(omlOperationDefinitions);
138   } catch (Exception e) {
139     e.printStackTrace();
140   }
141 }
142
143 public void visit(ASTSynchronizationDefinitions astSynchronizationDefinitions, TreeParent
144   parent) {
145   try {

```

I.4. OUTLINE VISITOR

```
135     TreeParent omlSynchronizationDefinitions = new TreeParent("sync",
136         astSynchronizationDefinitions.getStartLine(), astSynchronizationDefinitions.
137         getStartColumn(), astSynchronizationDefinitions.getEndLine());
138     } catch (Exception e) {
139         e.printStackTrace();
140     }
141 }
142 public void visit(ASTThreadDefinitions astThreadDefinitions, TreeParent parent) {
143     try {
144         TreeParent omlThreadDefinition = new TreeParent("thread", astThreadDefinitions.getStartLine()
145             (), astThreadDefinitions.getStartColumn(), astThreadDefinitions.getEndLine());
146         parent.addChild(omlThreadDefinition);
147     } catch (Exception e) {
148         e.printStackTrace();
149     }
150 }
151 public void visit(ASTTypeDefinitions astTypeDefinitions, TreeParent parent) {
152     try {
153         TreeParent omlTypeDefinitions = new TreeParent("types", astTypeDefinitions.getStartLine(),
154             astTypeDefinitions.getStartColumn(), astTypeDefinitions.getEndLine());
155         for (ASTTypeDefinitionsElement astTypeDefinitionsElement : astTypeDefinitions.
156             getTypeDefinitionsElements()) {
157             astTypeDefinitionsElement.accept(this, omlTypeDefinitions);
158         }
159         parent.addChild(omlTypeDefinitions);
160     } catch (Exception e) {
161         e.printStackTrace();
162     }
163 }
164 public void visit(ASTTypeDefinitionFieldList astTypeDefinitionFieldList, TreeParent parent) {
165     try {
166         ASTIdentifier astIdentifier = astTypeDefinitionFieldList.getIdentifier();
167         TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
168             getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
169         parent.addChild(omlIdentifier);
170     } catch (Exception e) {
171         e.printStackTrace();
172     }
173 }
174 public void visit(ASTTypeDefinitionType astTypeDefinitionType, TreeParent parent) {
175     try {
176         ASTIdentifier astIdentifier = astTypeDefinitionType.getIdentifier();
177         TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(), astIdentifier.
178             getStartLine(), astIdentifier.getStartColumn(), astIdentifier.getEndLine());
179         parent.addChild(omlIdentifier);
180     } catch (Exception e) {
181         e.printStackTrace();
182     }
183 }
184 public void visit(ASTValueDefinitions astValueDefinitions, TreeParent parent) {
185     try {
186         TreeParent omlValueDefinitions = new TreeParent("values", astValueDefinitions.getStartLine()
187             (), astValueDefinitions.getStartColumn(), astValueDefinitions.getEndLine());
188         for (ASTValueDefinitionsElement astValueDefinitionsElement : astValueDefinitions.
189             getValueDefinitionsElements()) {
190             astValueDefinitionsElement.accept(this, omlValueDefinitions);
191         }
192         parent.addChild(omlValueDefinitions);
193     } catch (Exception e) {
194         e.printStackTrace();
195     }
196 }
```


I.5 Ast2Oml Pretty Print Visitor

Listing I.24: OvertureAst2OmlVisitor.java located in org.overturetool.eclipse.ast2oml

```

1  /*****
2  * Overturetool
3  * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.ast2oml;
8
9  import org.overturetool.eclipse.ast.ASTCharacterLiteral;
10 import org.overturetool.eclipse.ast.ASTClass;
11 import org.overturetool.eclipse.ast.ASTComment;
12 import org.overturetool.eclipse.ast.ASTDocument;
13 import org.overturetool.eclipse.ast.ASTIdentifier;
14 import org.overturetool.eclipse.ast.ASTKeyword;
15 import org.overturetool.eclipse.ast.ASTNode;
16 import org.overturetool.eclipse.ast.ASTNumeral;
17 import org.overturetool.eclipse.ast.ASTQuoteLiteral;
18 import org.overturetool.eclipse.ast.ASTTextLiteral;
19 import org.overturetool.eclipse.internal.ast.OvertureVisitorOneArgImpl;
20
21 public class OvertureAst2OmlVisitor extends OvertureVisitorOneArgImpl{
22
23     private int lineCount = 1;
24     private int columnCount = 1;
25     private String doc = new String();
26
27     public void addTokenString(String keyword) {
28         try {
29             doc = doc + keyword;
30             columnCount = columnCount + keyword.length();
31         } catch (Exception e) {
32             e.printStackTrace();
33         }
34     }
35
36     public void setPosition(ASTNode astNode) {
37         try {
38             int startLine = astNode.getStartLine();
39             int startColumn = astNode.getStartColumn();
40             while (lineCount < startLine) {
41                 doc = doc + "\n";
42                 lineCount++;
43                 columnCount = 1;
44             }
45             while (columnCount < startColumn) {
46                 doc = doc + " ";
47                 columnCount++;
48             }
49         } catch (Exception e) {
50             e.printStackTrace();
51         }
52     }
53
54     public void visit(ASTComment astComment) {
55         try {
56             setPosition(astComment);
57             addTokenString(astComment.getComment());
58         } catch (Exception e) {
59             e.printStackTrace();
60         }
61     }
62
63     public String visit(ASTDocument astDocument) {
64         try {
65             visit(astDocument.getComments());
66             for (ASTClass astClass : astDocument.getClasses()) {
67                 astClass.accept(this);
68             }
69         } catch (Exception e) {
70             e.printStackTrace();
71         }
72         return doc;
73     }
74
75     public void visit(ASTIdentifier astIdentifier) {
76         try {

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
77     setPosition(astIdentifier);
78     addTokenString(astIdentifier.getIdentifierName());
79     astIdentifier.getComments().accept(this);
80 } catch (Exception e) {
81     e.printStackTrace();
82 }
83 }
84
85 public void visit(ASTKeyword astKeyword) {
86     try {
87         setPosition(astKeyword);
88         addTokenString(astKeyword.getValue());
89         astKeyword.getComments().accept(this);
90     } catch (Exception e) {
91         e.printStackTrace();
92     }
93 }
94
95 public void visit(ASTNumeral astNumeral) {
96     try {
97         setPosition(astNumeral);
98         addTokenString(astNumeral.getValue());
99         astNumeral.getComments().accept(this);
100    } catch (Exception e) {
101        e.printStackTrace();
102    }
103 }
104
105 public void visit(ASTQuoteLiteral astQuoteLiteral) {
106     try {
107         setPosition(astQuoteLiteral);
108         addTokenString(astQuoteLiteral.getQuoteLitteralName());
109         astQuoteLiteral.getComments().accept(this);
110     } catch (Exception e) {
111         e.printStackTrace();
112     }
113 }
114
115 public void visit(ASTTextLiteral astTextLiteral) {
116     try {
117         setPosition(astTextLiteral);
118         addTokenString("\\"+astTextLiteral.getTextLitteralName()+"\"");
119         astTextLiteral.getComments().accept(this);
120     } catch (Exception e) {
121         e.printStackTrace();
122     }
123 }
124
125 public void visit(ASTCharacterLiteral astCharacterLiteral) {
126     try {
127         setPosition(astCharacterLiteral);
128         addTokenString("\'+ astCharacterLiteral.getCharacterLitteralName()+"\'");
129         astCharacterLiteral.getComments().accept(this);
130     } catch (Exception e) {
131         e.printStackTrace();
132     }
133 }
134 }
```

I.6 XML Schema – Sample of the XML Schema

Listing I.25: vdm0_1.xsd located in plug-in org.overturetool.eclipse.editor in the folder XMLSchema. The file is in total 7832 lines, but only the first 217 lines are shown here.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by jens (student) -->
3 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
4   attributeFormDefault="unqualified">
5   <xs:element name="OMLDocument">
6     <xs:complexType>
7       <xs:sequence>
8         <xs:element ref="OMLPosition"/>
9         <xs:element ref="OMLComments" minOccurs="0"/>
10        <xs:sequence maxOccurs="unbounded">
11          <xs:element ref="OMLClass"/>
12        </xs:sequence>
13      </xs:sequence>
14    </xs:complexType>
15  </xs:element>
16  <xs:element name="OMLClass">
17    <xs:complexType>
18      <xs:sequence>
19        <xs:element ref="OMLPosition"/>
20        <xs:element name="OMLKeywordClass">
21          <xs:complexType>
22            <xs:sequence>
23              <xs:element ref="OMLPosition"/>
24              <xs:element ref="OMLComments" minOccurs="0"/>
25            </xs:sequence>
26            <xs:attribute name="value" type="xs:string" use="required" fixed="class"/>
27          </xs:complexType>
28        </xs:element>
29        <xs:element ref="OMLIdentifier"/>
30        <xs:sequence minOccurs="0">
31          <xs:element ref="OMLInheritanceClause"/>
32        </xs:sequence>
33        <xs:sequence minOccurs="0">
34          <xs:element ref="OMLClassBody"/>
35        </xs:sequence>
36        <xs:element name="OMLKeywordEnd">
37          <xs:complexType>
38            <xs:sequence>
39              <xs:element ref="OMLPosition"/>
40              <xs:element ref="OMLComments" minOccurs="0"/>
41            </xs:sequence>
42            <xs:attribute name="value" type="xs:string" use="required" fixed="end"/>
43          </xs:complexType>
44        </xs:element>
45        <xs:element name="OMLIdentifier2">
46          <xs:complexType>
47            <xs:sequence>
48              <xs:element ref="OMLIdentifier"/>
49            </xs:sequence>
50          </xs:complexType>
51        </xs:element>
52        </xs:sequence>
53      </xs:complexType>
54    </xs:element>
55    <xs:element name="OMLInheritanceClause">
56      <xs:complexType>
57        <xs:sequence>
58          <xs:element ref="OMLPosition"/>
59          <xs:element name="OMLKeywords">
60            <xs:complexType>
61              <xs:sequence>
62                <xs:element ref="OMLPosition"/>
63                <xs:element ref="OMLComments" minOccurs="0"/>
64              </xs:sequence>
65              <xs:attribute name="value" type="xs:string" use="required" fixed="is"/>
66            </xs:complexType>
67          </xs:element>
68          <xs:element name="OMLKeywordSubclass">
69            <xs:complexType>
70              <xs:sequence>
71                <xs:element ref="OMLPosition"/>
72                <xs:element ref="OMLComments" minOccurs="0"/>
73              </xs:sequence>
74              <xs:attribute name="value" type="xs:string" use="required" fixed="subclass"/>

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

74     </xs:complexType>
75 </xs:element>
76 <xs:element name="OMLKeywordOf">
77   <xs:complexType>
78     <xs:sequence>
79       <xs:element ref="OMLPosition"/>
80       <xs:element ref="OMLComments" minOccurs="0"/>
81     </xs:sequence>
82     <xs:attribute name="value" type="xs:string" use="required" fixed="of"/>
83   </xs:complexType>
84 </xs:element>
85 <xs:sequence maxOccurs="unbounded">
86   <xs:element ref="OMLIdentifier"/>
87 </xs:sequence>
88 </xs:sequence>
89 </xs:complexType>
90 </xs:element>
91 <xs:element name="OMLClassBody">
92   <xs:complexType>
93     <xs:sequence>
94       <xs:element ref="OMLPosition"/>
95       <xs:sequence maxOccurs="unbounded">
96         <xs:element ref="OMLDefinitionBlock"/>
97       </xs:sequence>
98     </xs:sequence>
99   </xs:complexType>
100 </xs:element>
101 <xs:element name="OMLDefinitionBlock">
102   <xs:complexType>
103     <xs:sequence>
104       <xs:element ref="OMLPosition"/>
105       <xs:choice>
106         <xs:element ref="OMLTypeDefinitions"/>
107         <xs:element ref="OMLValueDefinitions"/>
108         <xs:element ref="OMLFunctionDefinitions"/>
109         <xs:element ref="OMLOperationDefinitions"/>
110         <xs:element ref="OMLInstanceVariableDefinitions"/>
111         <xs:element ref="OMLSynchronizationDefinitions"/>
112         <xs:element ref="OMLThreadDefinitions"/>
113       </xs:choice>
114     </xs:sequence>
115   </xs:complexType>
116 </xs:element>
117 <xs:element name="OMLTypeDefinitions">
118   <xs:complexType>
119     <xs:sequence>
120       <xs:element ref="OMLPosition"/>
121       <xs:element name="OMLKeywordTypes">
122         <xs:complexType>
123           <xs:sequence>
124             <xs:element ref="OMLPosition"/>
125             <xs:element ref="OMLComments" minOccurs="0"/>
126           </xs:sequence>
127           <xs:attribute name="value" type="xs:string" use="required" fixed="types"/>
128         </xs:complexType>
129       </xs:element>
130       <xs:sequence minOccurs="0" maxOccurs="unbounded">
131         <xs:element name="OMLTypeDefinitionsElement">
132           <xs:complexType>
133             <xs:sequence>
134               <xs:element ref="OMLPosition"/>
135               <xs:element ref="OMLAccessTypeDefinition"/>
136               <xs:element name="OMLKeywordSemicolon" minOccurs="0">
137                 <xs:complexType>
138                   <xs:sequence>
139                     <xs:element ref="OMLPosition"/>
140                     <xs:element ref="OMLComments" minOccurs="0"/>
141                   </xs:sequence>
142                   <xs:attribute name="value" type="xs:string" use="required" fixed=";"/>
143                 </xs:complexType>
144               </xs:element>
145             </xs:sequence>
146           </xs:complexType>
147         </xs:element>
148       </xs:sequence>
149     </xs:sequence>
150   </xs:complexType>
151 </xs:element>
152 <xs:element name="OMLAccessTypeDefinition">
153   <xs:complexType>
154     <xs:sequence>
155       <xs:element ref="OMLPosition"/>
156       <xs:element ref="OMLAccess" minOccurs="0"/>
157       <xs:element ref="OMLTypeDefinition"/>

```

I.6. XML SCHEMA – SAMPLE OF THE XML SCHEMA

```
158     </xs:sequence>
159 </xs:complexType>
160 </xs:element>
161 <xs:element name="OMLAccess">
162   <xs:complexType>
163     <xs:sequence>
164       <xs:element ref="OMLPosition"/>
165       <xs:choice>
166         <xs:element name="OMLAccessPublic">
167           <xs:complexType>
168             <xs:sequence>
169               <xs:element ref="OMLPosition"/>
170               <xs:element name="OMLKeywordPublic">
171                 <xs:complexType>
172                   <xs:sequence>
173                     <xs:element ref="OMLPosition"/>
174                     <xs:element ref="OMLComments" minOccurs="0"/>
175                   </xs:sequence>
176                   <xs:attribute name="value" type="xs:string" use="required" fixed="public"/>
177                 </xs:complexType>
178               </xs:element>
179             </xs:sequence>
180           </xs:complexType>
181         </xs:element>
182         <xs:element name="OMLAccessPrivate">
183           <xs:complexType>
184             <xs:sequence>
185               <xs:element ref="OMLPosition"/>
186               <xs:element name="OMLKeywordPrivate">
187                 <xs:complexType>
188                   <xs:sequence>
189                     <xs:element ref="OMLPosition"/>
190                     <xs:element ref="OMLComments" minOccurs="0"/>
191                   </xs:sequence>
192                   <xs:attribute name="value" type="xs:string" use="required" fixed="private"/>
193                 </xs:complexType>
194               </xs:element>
195             </xs:sequence>
196           </xs:complexType>
197         </xs:element>
198         <xs:element name="OMLAccessProtected">
199           <xs:complexType>
200             <xs:sequence>
201               <xs:element ref="OMLPosition"/>
202               <xs:element name="OMLKeywordProtected">
203                 <xs:complexType>
204                   <xs:sequence>
205                     <xs:element ref="OMLPosition"/>
206                     <xs:element ref="OMLComments" minOccurs="0"/>
207                   </xs:sequence>
208                   <xs:attribute name="value" type="xs:string" use="required" fixed="protected"/>
209                 </xs:complexType>
210               </xs:element>
211             </xs:sequence>
212           </xs:complexType>
213         </xs:element>
214       </xs:choice>
215     </xs:sequence>
216   </xs:complexType>
217 </xs:element>
218 ...
```


I.7 Selected Editor Files

I.7.1 Ast2XmlAction.java

Listing I.26: Ast2XmlAction.java located in org.overturetool.eclipse.ast2xml

```

1  /*****
2  * Overturetool
3  * Jens Kielsingaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.editor.action;
8
9  import org.eclipse.core.runtime.IPath;
10 import org.eclipse.jface.action.IAction;
11 import org.eclipse.jface.dialogs.MessageDialog;
12 import org.eclipse.jface.viewers.ISelection;
13 import org.eclipse.jface.wizard.WizardDialog;
14 import org.eclipse.ui.IWorkbenchWindow;
15 import org.eclipse.ui.IWorkbenchWindowActionDelegate;
16 import org.eclipse.ui.part.FileEditorInput;
17 import org.jdom.Document;
18 import org.jdom.Element;
19 import org.jdom.output.Format;
20 import org.jdom.output.XMLOutputter;
21 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
22 import org.overturetool.eclipse.editor.OvertureEditor;
23 import org.overturetool.eclipse.editor.OvertureExtensionProvider;
24 import org.overturetool.eclipse.editor.wizard.OvertureWizard;
25 import org.overturetool.eclipse.util.OvertureAst2XmlVisitorInterface;
26
27 /**
28  * This class is used when the user pushes the 'Export to XML' item
29  * in the Overture menu. This is done by implementing
30  * Eclipse's IWorkbenchWindowActionDelegate interface and refer
31  * to this file from the plugin.xml file. The run method calls
32  * the method parseASTtoXML in the OvertureContentProvider class.
33  * The functionality is moved here in order to check for syntax
34  * before exporting to XML.
35  */
36
37 public class Ast2XmlAction implements IWorkbenchWindowActionDelegate {
38     private IWorkbenchWindow window;
39     OvertureEditor editor;
40     private XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
41
42     public Ast2XmlAction() {
43     }
44
45     public void run(IAction action) {
46         editor = (OvertureEditor) window.getActivePage().getActiveEditor();
47         if(editor == null){
48             MessageDialog.openInformation(
49                 window.getShell(),
50                 "Overture: Export to XML",
51                 "An OML file has to be active in the editor window in order to do this operation!\n Please
52                 open an OML file and try again");
53         }else{
54
55             Document doc = null;
56             editor.doSave(null);
57             if(editor.getOvertureContentOutlinePage().getOvertureContentProvider().getErrors().isEmpty()){
58                 try{
59
60                     IPath path = ((FileEditorInput) editor.getEditorInput()).getFile().getLocation();
61                     String fileName = path.removeFileExtension().lastSegment();
62                     String containerName = path.removeFileExtension().removeLastSegments(1).lastSegment();
63                     OvertureWizard overtureWizard = new OvertureWizard("ExportWizardPage", "Exporting an OML file
64                     to a XML file", containerName, fileName+".xml", "xml");
65                     WizardDialog dialog = new WizardDialog(window.getShell(), overtureWizard);
66                     dialog.open();
67                     if(overtureWizard.performFinish){
68                         OvertureExtensionProvider overtureExtensionProvider = new OvertureExtensionProvider();
69                         OvertureVisitorTwoArg<Element> ast2XmlVisitor = overtureExtensionProvider
70                             loadAst2XmlVisitor();
71                         Element omlDocument = new Element("OMLDocument");
72                         ast2XmlVisitor.visit(editor.getOvertureContentOutlinePage().getOvertureContentProvider().
73                             getASTDocument(), omlDocument);
74                         String documentString = outputter.outputString(new Document(omlDocument));

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
72     overtureWizard.doFinish(overtureWizard.containerNameTo, overtureWizard.fileNameTo,
73         documentString);
74     }
75     }catch (Exception e) {
76         System.out.println("OvertureContentProvider -> parseASTtoXML");
77         e.printStackTrace();
78     }
79 }
80 }
81
82 public void selectionChanged(IAction action, ISelection selection) {
83 }
84
85 public void dispose() {
86 }
87
88 public void init(IWorkbenchWindow window) {
89     this.window = window;
90 }
91 }
```


I.7.2 Xml2AstAction.java

Listing I.27: Xml2AstAction.java located in org.overturetool.eclipse.xml2ast

```

1  /*****
2  * Overturetool
3  * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.editor.action;
8
9  import java.net.URL;
10
11 import org.eclipse.core.runtime.CoreException;
12 import org.eclipse.core.runtime.Path;
13 import org.eclipse.core.runtime.Platform;
14 import org.eclipse.jface.action.IAction;
15 import org.eclipse.jface.dialogs.MessageDialog;
16 import org.eclipse.jface.viewers.ISelection;
17 import org.eclipse.jface.wizard.WizardDialog;
18 import org.eclipse.ui.IWorkbenchWindow;
19 import org.eclipse.ui.IWorkbenchWindowActionDelegate;
20 import org.jdom.Document;
21 import org.jdom.Element;
22 import org.jdom.input.SAXBuilder;
23 import org.overturetool.eclipse.ast.ASTDocument;
24 import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
25 import org.overturetool.eclipse.editor.OvertureConstants;
26 import org.overturetool.eclipse.editor.OvertureExtensionProvider;
27 import org.overturetool.eclipse.editor.wizard.OvertureExtendedWizard;
28 import org.overturetool.eclipse.editor.wizard.OvertureWizard;
29 import org.overturetool.eclipse.util.OvertureXml2AstConverterInterface;
30
31 /**
32  * This class is used when the user pushes the 'Import from XML'
33  * item in the Overture menu. This is done by implementing
34  * Eclipse's IWorkbenchWindowActionDelegate interface and refer
35  * to this file from the plugin.xml file. In the run
36  * method a XML file is imported to an OML file from a workspace
37  * location to another workspace location. The import is done in
38  * three steps. First the XML file is loaded into a jdom document
39  * and validated according to a XML schema. Then the jdom document
40  * is converted to our AST representation using a visitor provided
41  * by another plugin. Finilly, the AST is converted to a text string
42  * using a visitor provided by another plugin. This string if formatted
43  * so that the created OML file is similar to the original OML file.
44  */
45 public class Xml2AstAction implements IWorkbenchWindowActionDelegate {
46     private IWorkbenchWindow window;
47     //OvertureEditor editor;
48     OvertureExtensionProvider overtureExtensionProvider;
49     ASTDocument astDocument;
50
51     public Xml2AstAction() {
52     }
53
54     public void run(IAction action) {
55         overtureExtensionProvider = new OvertureExtensionProvider();
56         Document doc = null;
57         Document doc1 = null;
58         URL url = null;
59         String containerNameFrom = "";
60         String fileNameFrom = "";
61         String containerNameTo = "";
62         String fileNameTo = "";
63
64         OvertureExtendedWizard overtureExtendedWizard = new OvertureExtendedWizard("ExportWizardPage", "
        Importing a XML file to an OML file","xml","oml");
65         WizardDialog dialog = new WizardDialog(window.getShell(), overtureExtendedWizard);
66         dialog.open();
67         if (overtureExtendedWizard.performFinish){
68             containerNameFrom = overtureExtendedWizard.containerNameFrom;
69             fileNameFrom = overtureExtendedWizard.fileNameFrom;
70             containerNameTo = overtureExtendedWizard.containerNameTo;
71             fileNameTo = overtureExtendedWizard.fileNameTo;
72
73             SAXBuilder builder1 = null;
74             try{
75                 builder1= new SAXBuilder(true);
76                 builder1.setFeature("http://apache.org/xml/features/validation/schema", true);
77                 builder1.setFeature("http://xml.org/sax/features/validation", true);

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
78     url = Platform.find(
79         Platform.getBundle(OvertureConstants.PLUGIN_ID),
80         new Path("XMLSchema/vdm0_1.xsd")
81     );
82     builder1.setProperty(
83         "http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation",
84         Platform.resolve(url).getPath()
85     );
86     doc = builder1.build(getWorkspaceLocation() + containerNameFrom + "/" + fileNameFrom);
87 } catch (Exception e) {
88     System.err.println(e);
89     MessageDialog.openError(window.getShell(),
90         "Overture: Error validating the file '" + containerNameFrom + "'",
91         e.toString()
92     );
93 }
94 }
95 OvertureXml2AstConverterInterface xml2AstParser = (OvertureXml2AstConverterInterface)
96     overtureExtensionProvider.loadXml2AstParser();
97 if (doc != null) {
98     try {
99         Element omlDocument = doc.getRootElement();
100         if (omlDocument == null) System.err.println("DETTE ER ET STORT PROBLEM");
101         astDocument = xml2AstParser.overtureXMLDocument(omlDocument);
102         OvertureWizard overtureNewWizard = new OvertureWizard();
103         OvertureVisitorOneArg<String> ast2OmlVisitor = overtureExtensionProvider.loadAst2OmlVisitor
104             ();
105         String docContent = ast2OmlVisitor.visit(astDocument);
106         overtureNewWizard.doFinish(containerNameTo, fileNameTo, docContent);
107     } catch (CoreException e) {
108         e.printStackTrace();
109     } catch (Exception e) {
110         e.printStackTrace();
111     }
112 }
113 }
114 }
115
116 public void selectionChanged(IAction action, ISelection selection) {
117 }
118
119 public void dispose() {
120 }
121
122 public void init(IWorkbenchWindow window) {
123     this.window = window;
124 }
125
126 private String getWorkspaceLocation() {
127     return Platform.getLocation().toString();
128 }
129
130 }
```

I.7.3 OvertureContentProvider.java

Listing I.28: OvertureContentProvider.java located in org.overturetool.eclipse.editor

```

1  /*****
2  * Overturetool
3  * Jens Kielsingaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.editor;
8
9  import java.io.ByteArrayInputStream;
10 import java.util.List;
11
12 import org.eclipse.core.resources.IFile;
13 import org.eclipse.core.resources.IMarker;
14 import org.eclipse.core.runtime.CoreException;
15 import org.eclipse.core.runtime.IPath;
16 import org.eclipse.core.runtime.Platform;
17 import org.eclipse.jface.text.IDocument;
18 import org.eclipse.jface.viewers.ITreeContentProvider;
19 import org.eclipse.jface.viewers.Viewer;
20 import org.eclipse.ui.IWorkbenchWindow;
21 import org.eclipse.ui.part.FileEditorInput;
22 import org.eclipse.ui.texteditor.IDocumentProvider;
23 import org.jdom.output.Format;
24 import org.jdom.output.XMLOutputter;
25 import org.overturetool.eclipse.ast.ASTDokument;
26 import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
27 import org.overturetool.eclipse.internal.ast.InternalASTDocument;
28 import org.overturetool.eclipse.util.OvertureOutlineVisitorInterface;
29 import org.overturetool.eclipse.util.OvertureParseException;
30 import org.overturetool.eclipse.util.OvertureParserInterface;
31 import org.overturetool.eclipse.util.TreeObject;
32 import org.overturetool.eclipse.util.TreeParent;
33
34 /**
35  * This class has several functions.
36  *
37  * - Parses the text in the active Overture editor and builds an AST.
38  * - Report syntax errors to the user.
39  * - Exports the AST to XML
40  * - Creates an outline view by extending the ITreeContentProvider
41  *
42  * The parsing, the export to XML and the outline view is all done
43  * by other plugins. This class is therefore only responsible for
44  * handling the data from these plugins. Uses the OvertureExtensionProvider
45  * class to load the plugins. By defining extension points and loading
46  * the corresponding extensions the user is encouraged to e.g. create
47  * he/her own AST outline view instead of the traditional outline view.
48  * The inputChanged method is called every time the user saves an OML
49  * document.
50  */
51
52 public class OvertureContentProvider implements ITreeContentProvider {
53
54     private TreeParent outlineRoot = null;
55     private List<OvertureParseException> errors = null;
56     private ASTDocument astDocument = null;
57     private XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
58     private FileEditorInput input;
59     private IDocumentProvider overtureDocumentProvider;
60     private OvertureExtensionProvider overtureExtensionProvider;
61     private IWorkbenchWindow window;
62
63     public OvertureContentProvider(IDocumentProvider overtureDocumentProvider){
64         this.overtureDocumentProvider = overtureDocumentProvider;
65         overtureExtensionProvider = new OvertureExtensionProvider();
66     }
67
68     public List<OvertureParseException> getErrors(){
69         return errors;
70     }
71
72     public ASTDocument getASTDocument(){
73         return astDocument;
74     }
75
76     private String getWorkspaceLocation(){

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

77     return Platform.getLocation().toString();
78 }
79
80 private IPath getWorkspaceLocation2(){
81     return Platform.getLocation();
82 }
83 private ASTDocument parseOML(IDocument document) {
84     OvertureParserInterface overtureParser = null;
85     try{
86         overtureParser = overtureExtensionProvider.loadParser();
87         astDocument = (InternalASTDocument) overtureParser.parse(new ByteArrayInputStream(document.get
88             ().getBytes()));
89         errors = overtureParser.getErrors();
90         if (!errors.isEmpty()) {
91             for (OvertureParseException error: errors) {
92                 addMarker(input.getFile(), error.getError(),
93                     new Integer(error.getLine()),
94                     new Integer(IMarker.SEVERITY_ERROR));
95             }
96         } catch (Exception e) {
97             e.printStackTrace();
98         }
99         return astDocument;
100     }
101
102 private void addMarker(IFile file, String message, int lineNumber, int severity) {
103     try {
104         IMarker marker = file.createMarker(IMarker.PROBLEM);
105         marker.setAttribute(IMarker.MESSAGE, message);
106         marker.setAttribute(IMarker.SEVERITY, severity);
107         if (lineNumber == -1) {
108             lineNumber = 1;
109         }
110         marker.setAttribute(IMarker.LINE_NUMBER, lineNumber);
111     } catch (CoreException e) {
112     }
113 }
114
115 private void createOutlineView() {
116     if (errors.isEmpty()) {
117         OvertureVisitorTwoArg<TreeParent> overtureOutlineVisitor = overtureExtensionProvider.
118             loadOutlineVisitor();
119         TreeParent omlDocument = new TreeParent("OMLDocument", astDocument.getStartLine(), astDocument.
120             getStartColumn(), astDocument.getEndLine());
121         overtureOutlineVisitor.visit(astDocument, omlDocument);
122         outlineRoot = new TreeParent("", 10, 10, 10);
123         outlineRoot.addChild(omlDocument);
124     } else {
125         initialize();
126     }
127 }
128
129 public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {
130     ASTDocument astDocument;
131
132     try {
133         if (input != null) {
134             input.getFile().deleteMarkers(IMarker.PROBLEM, true, 1);
135         } catch (CoreException ce) {
136             System.out.println("OvertureContentProvider -> clearProblemView");
137         }
138
139         input = (FileEditorInput) newInput;
140
141         if (newInput != null) {
142             IDocument document = overtureDocumentProvider.getDocument(newInput);
143             if (document != null) {
144                 parseOML(document);
145                 createOutlineView();
146             }
147         }
148
149     public void dispose() {
150     }
151
152     public boolean isDeleted(Object element) {
153         return false;
154     }
155
156     private void initialize() {
157         TreeParent treeRoot = new TreeParent("No outline available", 0, 0, 0);

```

I.7. SELECTED EDITOR FILES

```
158     outlineRoot = new TreeParent("",0,0,0);
159     outlineRoot.addChild(treeRoot);
160 }
161
162 public Object[] getElements(Object parent) {
163     if (outlineRoot == null) {
164         initialize();
165     }
166     return getChildren(outlineRoot);
167 }
168
169 public Object getParent(Object child) {
170     if (child instanceof TreeObject) {
171         return ((TreeObject) child).getParent();
172     }
173     return null;
174 }
175
176 public Object[] getChildren(Object parent) {
177     if (parent instanceof TreeParent) {
178         return ((TreeParent) parent).getChildren();
179     }
180     return new Object[0];
181 }
182
183 public boolean hasChildren(Object parent) {
184     if (parent instanceof TreeParent)
185         return ((TreeParent) parent).hasChildren();
186     return false;
187 }
188 }
```


I.7.4 OvertureContentOutlinePage.java

Listing I.29: OvertureContentOutlinePage.java located in org.overturetool.eclipse.editor

```

1  /*****
2  * Overturetool
3  * Jens Kielsingaard Hansen & Jacob Porsborg Nielsen
4  * Spring 2005, DTU Denmark
5  *****/
6
7  package org.overturetool.eclipse.editor;
8
9  import org.eclipse.jface.text.IDocument;
10 import org.eclipse.jface.viewers.ISelection;
11 import org.eclipse.jface.viewers.IStructuredSelection;
12 import org.eclipse.jface.viewers.ITableLabelProvider;
13 import org.eclipse.jface.viewers.LabelProvider;
14 import org.eclipse.jface.viewers.SelectionChangedEvent;
15 import org.eclipse.jface.viewers.TreeViewer;
16 import org.eclipse.swt.graphics.Image;
17 import org.eclipse.swt.widgets.Composite;
18 import org.eclipse.swt.widgets.Control;
19 import org.eclipse.ui.IEditorInput;
20 import org.eclipse.ui.ISharedImages;
21 import org.eclipse.ui.IWorkbench;
22 import org.eclipse.ui.IWorkbenchWindow;
23 import org.eclipse.ui.PlatformUI;
24 import org.eclipse.ui.WorkbenchException;
25 import org.eclipse.ui.texteditor.IDocumentProvider;
26 import org.eclipse.ui.texteditor.ITextEditor;
27 import org.eclipse.ui.views.contentoutline.ContentOutlinePage;
28 import org.overturetool.eclipse.util.TreeObject;
29
30 /**
31  * Creates and updates an outline view for the OvertureEditor.
32  * Extends Eclipse's ContentOutlinePage providing the standard
33  * outline view. The viewer is based on Eclipse's TreeViewer.
34  * The view is updated when saving an OML file in the OvertureEditor.
35  * The class OvertureContentProvider is used to provide the content
36  * for this view.
37  */
38
39 public class OvertureContentOutlinePage extends ContentOutlinePage {
40
41     protected ITextEditor overtureEditor;
42     private OvertureContentProvider overtureContentProvider;
43     protected IDocumentProvider overtureDocumentProvider;
44     private IWorkbenchWindow window;
45     private TreeViewer viewer;
46     protected IEditorInput input;
47
48     /**
49      * Creates a content outline page using the an OvertureEditor
50      * and an OvertureDocumentProvider
51      */
52     public OvertureContentOutlinePage(IDocumentProvider overtureDocumentProvider, ITextEditor
53         overtureEditor) {
54         super();
55         this.overtureDocumentProvider = overtureDocumentProvider;
56         this.overtureEditor = overtureEditor;
57     }
58
59     public OvertureContentProvider getOvertureContentProvider(){
60         return overtureContentProvider;
61     }
62
63     /**
64      * Configures the TreeViewer with the overtureContentProvider.
65      * Is called when the user opens an OML file.
66      */
67     public void createControl(Composite parent) {
68         super.createControl(parent);
69
70         doSetPerspective();
71
72         try{
73             viewer = getTreeViewer();
74             overtureContentProvider = new OvertureContentProvider(overtureDocumentProvider);
75             viewer.setContentProvider(overtureContentProvider);
76             if (input != null) {

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
76     viewer.setInput(input);
77     }
78     } catch (Exception e) {
79     e.printStackTrace();
80     }
81     }
82
83     /**
84     * Sets the input of the outline view
85     */
86     public void setInput(Object input) {
87     this.input = (IEditorInput) input;
88     update();
89     }
90
91     public void doSetPerspective(){
92     try {
93     IWorkbench workbench= PlatformUI.getWorkbench();
94     workbench.showPerspective(
95     "org.overturetool.eclipse.editor.OverturePerspective",
96     PlatformUI.getWorkbench().getActiveWorkbenchWindow()
97     );
98     } catch (WorkbenchException e) {
99     e.printStackTrace();
100    }
101    }
102
103    /**
104    * Updates the outline view
105    */
106    public void update() {
107
108
109
110    try{
111    if (viewer != null) {
112    Control control = viewer.getControl();
113    if (control != null && !control.isDisposed()) {
114    control.setRedraw(false);
115    viewer.setInput(input);
116    viewer.expandAll();
117    control.setRedraw(true);
118    }
119    }
120    } catch (Exception e) {
121    e.printStackTrace();
122    }
123    }
124
125    /**
126    * Registers when a user click on one of the nodes in the
127    * outline view. Gets the chosen tree element and highlights
128    * the the corresponding lines in the editor.
129    */
130    public void selectionChanged(SelectionChangedEvent event) {
131    super.selectionChanged(event);
132    ISelection selection = event.getSelection();
133    if (selection.isEmpty())
134    overtureEditor.resetHighlightRange();
135    else {
136    TreeObject treeObject = (TreeObject) ((IStructuredSelection) selection).getFirstElement();
137    IDocument doc = this.overtureDocumentProvider.getDocument(input);
138    try {
139    overtureEditor.setHighlightRange(doc.getLineOffset(treeObject.getStartLine())-1,doc.
140    getLineOffset(treeObject.getEndLine()-treeObject.getStartLine()), true);
141    } catch (Exception x) {
142    overtureEditor.resetHighlightRange();
143    }
144    }
145    }
146 }
```


I.8 Export/Import to/from UML

I.8.1 XML to XMI/UML Conversion

Listing I.30: convertXml2Uml.xml located in org.overturetool.eclipse.xml2uml

```

1 <?xml version="1.0"?>
2
3 <xsl:stylesheet
4   version="1.0"
5   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xmlns:UML="org.omg.xmi.namespace.UML"
8 >
9 <xsl:output indent="yes"/>
10 <xsl:template match="OMLDocument">
11 <XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Sat Aug 06 00:42:38
    CEST 2005'>
12   <XMI.header>
13     <XMI.documentation>
14       <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
15       <XMI.exporterVersion>1.0</XMI.exporterVersion>
16     </XMI.documentation>
17   </XMI.header>
18   <XMI.content>
19     <UML:Model xmi.id = 'I1a666bfm10588cd000amm7f55' name = 'model 1' isSpecification = 'false'
20       isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
21       <UML:Namespace.ownedElement>
22         <xsl:apply-templates select="OMLClass"/>
23     </UML:Namespace.ownedElement>
24   </UML:Model>
25   <UML:Diagram xmi.id = 'I1a666bfm10588cd000amm7f54' isVisible = 'true' name = 'Class Diagram_1
    ,
26     zoom = '1.0'>
27     <UML:GraphElement.position>
28       <XMI.field>0.0</XMI.field>
29       <XMI.field>0.0</XMI.field>
30     </UML:GraphElement.position>
31     <UML:GraphNode.size>
32       <XMI.field>160.0</XMI.field>
33       <XMI.field>131.0</XMI.field>
34     </UML:GraphNode.size>
35     <UML:Diagram.viewport>
36       <XMI.field>0.0</XMI.field>
37       <XMI.field>0.0</XMI.field>
38     </UML:Diagram.viewport>
39     <UML:GraphElement.semanticModel>
40     <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f53' presentation = ''
41       typeInfo = 'ClassDiagram'/>
42     </UML:GraphElement.semanticModel>
43     <UML:GraphElement.contained>
44     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f50' isVisible = 'true'>
45     <UML:GraphElement.position>
46       <XMI.field>70.0</XMI.field>
47       <XMI.field>40.0</XMI.field>
48     </UML:GraphElement.position>
49     <UML:GraphNode.size>
50       <XMI.field>100.0</XMI.field>
51       <XMI.field>71.0</XMI.field>
52     </UML:GraphNode.size>
53     <UML:GraphElement.semanticModel>
54     <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f4f' presentation = ''>
55     <UML:Uml1SemanticModelBridge.element>
56     <UML:Class xmi.idref = 'I1a666bfm10588cd000amm7f51'>
57     </UML:Uml1SemanticModelBridge.element>
58     </UML:Uml1SemanticModelBridge>
59     </UML:GraphElement.semanticModel>
60     <UML:GraphElement.contained>
61     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4e' isVisible = 'true'>
62     <UML:GraphElement.position>
63       <XMI.field>1.0</XMI.field>
64       <XMI.field>1.0</XMI.field>
65     </UML:GraphElement.position>
66     <UML:GraphNode.size>
67       <XMI.field>98.0</XMI.field>
68       <XMI.field>19.0</XMI.field>
69     </UML:GraphNode.size>
70     <UML:GraphElement.semanticModel>

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

71         <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f4d'
72             presentation = ''
73             typeInfo = 'NameCompartment'/>
74     </UML:GraphElement.semanticModel>
75     <UML:GraphElement.contained>
76     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4c' isVisible = 'true'>
77         <UML:GraphElement.position>
78             <XMI.field>30.3542</XMI.field>
79             <XMI.field>2.0</XMI.field>
80         </UML:GraphElement.position>
81         <UML:GraphNode.size>
82             <XMI.field>37.2915</XMI.field>
83             <XMI.field>15.0</XMI.field>
84         </UML:GraphNode.size>
85     </UML:GraphElement.semanticModel>
86     <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f4b'
87         presentation = ''
88         typeInfo = 'Name'/>
89     </UML:GraphElement.semanticModel>
90     <UML:GraphElement.contained>
91     <UML:GraphNode>
92     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4a' isVisible = 'true'>
93         <UML:GraphElement.position>
94             <XMI.field>1.0</XMI.field>
95             <XMI.field>20.0</XMI.field>
96         </UML:GraphElement.position>
97         <UML:GraphNode.size>
98             <XMI.field>98.0</XMI.field>
99             <XMI.field>1.0</XMI.field>
100        </UML:GraphNode.size>
101    </UML:GraphElement.semanticModel>
102    <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f49'
103        presentation = ''
104        typeInfo = 'CompartmentSeparator'/>
105    </UML:GraphElement.semanticModel>
106    <UML:GraphElement.contained>
107    <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f48' isVisible = 'true'>
108        <UML:GraphElement.position>
109            <XMI.field>1.0</XMI.field>
110            <XMI.field>21.0</XMI.field>
111        </UML:GraphElement.position>
112        <UML:GraphNode.size>
113            <XMI.field>98.0</XMI.field>
114            <XMI.field>24.0</XMI.field>
115        </UML:GraphNode.size>
116    </UML:GraphElement.semanticModel>
117    <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f47'
118        presentation = ''
119        typeInfo = 'AttributeCompartment'/>
120    </UML:GraphElement.semanticModel>
121    <UML:GraphElement.contained>
122    <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f46' isVisible = 'true'>
123        <UML:GraphElement.position>
124            <XMI.field>2.0</XMI.field>
125            <XMI.field>2.0</XMI.field>
126        </UML:GraphElement.position>
127        <UML:GraphNode.size>
128            <XMI.field>94.0</XMI.field>
129            <XMI.field>20.0</XMI.field>
130        </UML:GraphNode.size>
131    </UML:GraphElement.semanticModel>
132    <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f45'
133        presentation = ''
134        typeInfo = 'DelimitedSection'/>
135    </UML:GraphElement.semanticModel>
136    <UML:GraphElement.contained>
137    <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f39' isVisible = 'true'>
138        <UML:GraphElement.position>
139            <XMI.field>2.0</XMI.field>
140            <XMI.field>2.0</XMI.field>
141        </UML:GraphElement.position>
142        <UML:GraphNode.size>
143            <XMI.field>90.0</XMI.field>
144            <XMI.field>15.0</XMI.field>
145        </UML:GraphNode.size>
146    </UML:DiagramElement.property>
147    <UML:Property xmi.id = 'I1a666bfm10588cd000amm7f37' key = 'gentleware-
        custom-width'
        value = '0.0'/>
148    <UML:Property xmi.id = 'I1a666bfm10588cd000amm7f36' key = 'gentleware-
        custom-height'
        value = '0.0'/>
149    </UML:DiagramElement.property>

```

I.8. EXPORT/IMPORT TO/FROM UML

```
148 <UML: GraphElement.semanticModel>
149 <UML: Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f38'
      presentation = ''>
150 <UML: Uml1SemanticModelBridge.element>
151 <UML: Attribute xmi.idref = 'I1a666bfm10588cd000amm7f3a' />
152 </UML: Uml1SemanticModelBridge.element>
153 </UML: Uml1SemanticModelBridge>
154 </UML: GraphElement.semanticModel>
155 <UML: GraphElement.contained>
156 <UML: GraphNode xmi.id = 'I1a666bfm10588cd000amm7f35' isVisible = 'true'>
157 <UML: GraphElement.position>
158 <XMI.field >0.0</XMI.field>
159 <XMI.field >0.0</XMI.field>
160 </UML: GraphElement.position>
161 <UML: GraphNode.size>
162 <XMI.field >6.4238</XMI.field>
163 <XMI.field >15.0</XMI.field>
164 </UML: GraphNode.size>
165 <UML: GraphElement.semanticModel>
166 <UML: SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f34'
      presentation = ''
      typeInfo = 'Visibility' />
167 </UML: GraphElement.semanticModel>
168 </UML: GraphNode>
169 <UML: GraphNode xmi.id = 'I1a666bfm10588cd000amm7f33' isVisible = 'true'>
170 <UML: GraphElement.position>
171 <XMI.field >6.4238</XMI.field>
172 <XMI.field >0.0</XMI.field>
173 </UML: GraphElement.position>
174 <UML: GraphNode.size>
175 <XMI.field >18.353</XMI.field>
176 <XMI.field >15.0</XMI.field>
177 </UML: GraphNode.size>
178 <UML: GraphElement.semanticModel>
179 <UML: SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f32'
      presentation = ''
      typeInfo = 'Name' />
181 </UML: GraphElement.semanticModel>
182 </UML: GraphNode>
183 <UML: GraphNode xmi.id = 'I1a666bfm10588cd000amm7f31' isVisible = 'true'>
184 <UML: GraphElement.position>
185 <XMI.field >24.7769</XMI.field>
186 <XMI.field >0.0</XMI.field>
187 </UML: GraphElement.position>
188 <UML: GraphNode.size>
189 <XMI.field >3.0562</XMI.field>
190 <XMI.field >15.0</XMI.field>
191 </UML: GraphNode.size>
192 <UML: GraphElement.semanticModel>
193 <UML: SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f30'
      presentation = ''
      typeInfo = 'TypeSeparator' />
195 </UML: GraphElement.semanticModel>
196 </UML: GraphNode>
197 <UML: GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2f' isVisible = 'true'>
198 <UML: GraphElement.position>
199 <XMI.field >27.833</XMI.field>
200 <XMI.field >0.0</XMI.field>
201 </UML: GraphElement.position>
202 <UML: GraphNode.size>
203 <XMI.field >11.6177</XMI.field>
204 <XMI.field >15.0</XMI.field>
205 </UML: GraphNode.size>
206 <UML: GraphElement.semanticModel>
207 <UML: SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f2e'
      presentation = ''
      typeInfo = 'StructuralFeatureType' />
209 </UML: GraphElement.semanticModel>
210 <UML: GraphElement.contained>
211 <UML: GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2d' isVisible = '
      true'>
212 <UML: GraphElement.position>
213 <XMI.field >0.0</XMI.field>
214 <XMI.field >0.0</XMI.field>
215 </UML: GraphElement.position>
216 <UML: GraphNode.size>
217 <XMI.field >11.6177</XMI.field>
218 <XMI.field >15.0</XMI.field>
219 </UML: GraphNode.size>
220 <UML: GraphElement.semanticModel>
221 <UML: Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f2c'
      presentation = ''>
222 </UML: Uml1SemanticModelBridge>
223 <UML: Uml1SemanticModelBridge.element>
224 <UML: DataType xmi.idref = 'I1a666bfm10588cd000amm7f3e' />
```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

225     </UML:Uml1SemanticModelBridge.element>
226     </UML:Uml1SemanticModelBridge>
227     </UML:GraphElement.semanticModel>
228     <UML:GraphElement.contained>
229     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2b' isVisible =
        'true'>
230         <UML:GraphElement.position>
231         <XMI.field>0.0</XMI.field>
232         <XMI.field>0.0</XMI.field>
233         </UML:GraphElement.position>
234         <UML:GraphNode.size>
235         <XMI.field>11.6177</XMI.field>
236         <XMI.field>15.0</XMI.field>
237         </UML:GraphNode.size>
238         <UML:GraphElement.semanticModel>
239         <UML:SimpleSemanticModelElement xmi.id = '
            I1a666bfm10588cd000amm7f2a' presentation = ''
            typeInfo = 'Name'/>
240         </UML:GraphElement.semanticModel>
241         </UML:GraphNode>
242     </UML:GraphElement.contained>
243     </UML:GraphElement.contained>
244     </UML:GraphElement.contained>
245     </UML:GraphElement.contained>
246     </UML:GraphElement.contained>
247     </UML:GraphElement.contained>
248     </UML:GraphElement.contained>
249     </UML:GraphElement.contained>
250     </UML:GraphElement.contained>
251     </UML:GraphElement.contained>
252     </UML:GraphElement.contained>
253     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f44' isVisible = 'true'>
254     <UML:GraphElement.position>
255     <XMI.field>1.0</XMI.field>
256     <XMI.field>45.0</XMI.field>
257     </UML:GraphElement.position>
258     <UML:GraphNode.size>
259     <XMI.field>98.0</XMI.field>
260     <XMI.field>1.0</XMI.field>
261     </UML:GraphNode.size>
262     <UML:GraphElement.semanticModel>
263     <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f43'
        presentation = ''
        typeInfo = 'CompartmentSeparator'/>
264     </UML:GraphElement.semanticModel>
265     </UML:GraphNode>
266     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f42' isVisible = 'true'>
267     <UML:GraphElement.position>
268     <XMI.field>1.0</XMI.field>
269     <XMI.field>46.0</XMI.field>
270     </UML:GraphElement.position>
271     <UML:GraphElement.size>
272     <XMI.field>98.0</XMI.field>
273     <XMI.field>24.0</XMI.field>
274     </UML:GraphElement.size>
275     <UML:GraphElement.semanticModel>
276     <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f41'
        presentation = ''
        typeInfo = 'OperationCompartment'/>
277     </UML:GraphElement.semanticModel>
278     </UML:GraphElement.contained>
279     <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f40' isVisible = 'true'>
280     <UML:GraphElement.position>
281     <XMI.field>2.0</XMI.field>
282     <XMI.field>2.0</XMI.field>
283     </UML:GraphElement.position>
284     <UML:GraphElement.size>
285     <XMI.field>94.0</XMI.field>
286     <XMI.field>20.0</XMI.field>
287     </UML:GraphElement.size>
288     <UML:GraphElement.semanticModel>
289     <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f3f'
        presentation = ''
        typeInfo = 'DelimitedSection'/>
290     </UML:GraphElement.semanticModel>
291     </UML:GraphNode>
292     </UML:GraphElement.contained>
293     </UML:GraphElement.contained>
294     </UML:GraphElement.contained>
295     </UML:GraphElement.contained>
296     </UML:GraphElement.contained>
297     </UML:GraphElement.contained>
298     </UML:GraphElement.contained>
299     </UML:Diagram.owner>
300     <UML:Diagram.owner>
301     <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f52' presentation = ''>
302     <UML:Uml1SemanticModelBridge.element>
303     <UML:Model xmi.idref = 'I1a666bfm10588cd000amm7f55'/>

```

I.8. EXPORT/IMPORT TO/FROM UML

```

304         </UML:Uml1SemanticModelBridge.element>
305     </UML:Uml1SemanticModelBridge>
306 </UML:Diagram.owner>
307 </UML:Diagram>
308 </XMI.content>
309 </XMI>
310 </xsl:template>
311
312 <xsl:template match="OMLClass">
313 <xsl:element name="UML:Class">
314     <xsl:attribute name="xmi.id">
315         <xsl:value-of select='concat("I1a666bfm10588cd000amm7f51")'/>
316     </xsl:attribute>
317
318     <xsl:attribute name="name">
319         <xsl:value-of select="OMLIdentifier/@name"/>
320     </xsl:attribute>
321
322 <xsl:attribute name="visibility">
323     <xsl:value-of select='concat("public")'/>
324 </xsl:attribute>
325     <xsl:attribute name="isSpecification">
326         <xsl:value-of select='concat("false")'/>
327 </xsl:attribute>
328     <xsl:attribute name="isRoot">
329         <xsl:value-of select='concat("false")'/>
330 </xsl:attribute>
331     <xsl:attribute name="isLeaf">
332         <xsl:value-of select='concat("false")'/>
333 </xsl:attribute>
334     <xsl:attribute name="isAbstract">
335         <xsl:value-of select='concat("false")'/>
336 </xsl:attribute>
337     <xsl:attribute name="isActive">
338         <xsl:value-of select='concat("false")'/>
339 </xsl:attribute>
340 <UML:Classifier.feature>
341
342     <xsl:element name="UML:Attribute">
343         <xsl:attribute name="xmi.id">
344             <xsl:value-of select='concat("I1a666bfm10588cd000amm7f3a")'/>
345         </xsl:attribute>
346         <xsl:attribute name="name">
347             <xsl:value-of select="OMLClassBody/OMLDefinitionBlock/OMLTypeDefinitions/
348                 OMLTypeDefinitionsElement/OMLAccessTypeDefinition/OMLTypeDefinition/
349                 OMLTypeDefinitionType/OMLIdentifier/@name"/>
350         </xsl:attribute>
351
352         <xsl:attribute name="visibility">
353             <xsl:value-of select="OMLClassBody/OMLDefinitionBlock/OMLTypeDefinitions/
354                 OMLTypeDefinitionsElement/OMLAccessTypeDefinition/OMLAccess/**/@value"/>
355         </xsl:attribute>
356
357         <xsl:attribute name="isSpecification">
358             <xsl:value-of select='concat("false")'/>
359         </xsl:attribute>
360
361         <xsl:attribute name="ownerScope">
362             <xsl:value-of select='concat("instance")'/>
363         </xsl:attribute>
364
365         <xsl:attribute name="changeability">
366             <xsl:value-of select='concat("changeable")'/>
367         </xsl:attribute>
368
369         <UML:StructuralFeature.type>
370             <UML:DataType xmi.idref = 'I1a666bfm10588cd000amm7f3e'/>
371         </UML:StructuralFeature.type>
372     </xsl:element>
373
374 </UML:Classifier.feature>
375 </xsl:element>
376
377 <UML:Package xmi.id = 'I1a666bfm10588cd000amm7f3c' name = 'java' isSpecification = 'false'
378     isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
379 <UML:Namespace.ownedElement>
380     <UML:Package xmi.id = 'I1a666bfm10588cd000amm7f3d' name = 'lang' isSpecification = '
381         false'
382         isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
383     <UML:Namespace.ownedElement>
384         <UML:DataType xmi.id = 'I1a666bfm10588cd000amm7f3e' name = 'int' isSpecification
385             = 'false'

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```
382         isRoot = 'false' isLeaf = 'false' isAbstract = 'false' />
383         <UML:DataType xmi.id = 'I1a666bfm10588cd000amm7f3b' name = 'void' isSpecification
384             = 'false'
385             isRoot = 'false' isLeaf = 'false' isAbstract = 'false' />
386         </UML:Namespace.ownedElement>
387     </UML:Package>
388 </UML:Namespace.ownedElement>
389 </UML:Package>
390 </xsl:template>
</xsl:stylesheet>
```

I.8.2 XMI/UML to XML Conversion

Listing I.31: convertUml2Xml.xsl located in
org.overturetool.eclipse.uml2xml

```

1 <?xml version="1.0"?>
2
3 <xsl:stylesheet
4   version="1.0"
5   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
6   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
7   xmlns:UML="org.omg.xmi.namespace.UML"
8 >
9 <xsl:output indent="yes"/>
10 <xsl:template match="XMI[@xmi.version='1.2']">
11 <OMLDocument>
12   <OMLPosition startLine="1" startColumn="1" endLine="1" />
13   <xsl:apply-templates select="XMI.content/UML:Model"/>
14 </OMLDocument>
15 </xsl:template>
16
17 <xsl:template match="XMI">
18   <xsl:message terminate="yes">Unknown XMI version</xsl:message>
19 </xsl:template>
20
21 <xsl:template match="UML:Model/UML:Namespace.ownedElement/UML:Class">
22 <OMLClass>
23   <OMLPosition startLine="1" startColumn="1" endLine="1" />
24   <xsl:element name="OMLKeywordClass">
25     <xsl:attribute name="value">
26       <xsl:value-of select="concat('class')"/>
27     </xsl:attribute>
28     <OMLPosition startLine="1" startColumn="1" endLine="1" />
29   </xsl:element>
30   <xsl:element name="OMLIdentifier">
31     <xsl:attribute name="name">
32       <xsl:value-of select="@name"/>
33     </xsl:attribute>
34     <OMLPosition startLine="1" startColumn="1" endLine="1" />
35   </xsl:element>
36
37   <OMLClassBody>
38     <OMLPosition startLine="1" startColumn="1" endLine="1" />
39     <OMLDefinitionBlock>
40       <OMLPosition startLine="1" startColumn="1" endLine="1" />
41       <OMLTypeDefinitions>
42         <OMLPosition startLine="1" startColumn="1" endLine="1" />
43         <xsl:element name="OMLKeywordTypes">
44           <xsl:attribute name="value">
45             <xsl:value-of select="concat('types')"/>
46           </xsl:attribute>
47           <OMLPosition startLine="1" startColumn="1" endLine="1" />
48         </xsl:element>
49         <xsl:apply-templates select="UML:Classifier.feature/UML:Attribute"/>
50       </OMLTypeDefinitions></OMLDefinitionBlock></OMLClassBody>
51
52       <xsl:element name="OMLKeywordEnd">
53         <xsl:attribute name="value">
54           <xsl:value-of select="concat('end')"/>
55         </xsl:attribute>
56         <OMLPosition startLine="1" startColumn="1" endLine="1" />
57       </xsl:element>
58     </OMLIdentifier2>
59     <xsl:element name="OMLIdentifier">
60       <xsl:attribute name="name">
61         <xsl:value-of select="@name"/>
62       </xsl:attribute>
63       <OMLPosition startLine="1" startColumn="1" endLine="1" />
64     </xsl:element>
65   </OMLIdentifier2>
66 </OMLClass>
67 </xsl:template>
68
69 <xsl:template match="UML:Attribute">
70 <OMLTypeDefinitionsElement>
71 <OMLPosition startLine="1" startColumn="1" endLine="1" />
72 <OMLAccessTypeDefinition>
73 <OMLPosition startLine="1" startColumn="1" endLine="1" />
74 <OMLAccess>
75 <OMLPosition startLine="1" startColumn="1" endLine="1" />

```

APPENDIX I. SOURCE CODE - SELECTED SAMPLES

```

77 <xsl:if test="@visibility = 'private'">
78 <OMLAccessPrivate>
79 <OMLPosition startLine="1" startColumn="1" endLine="1" />
80 <xsl:element name="OMLKeywordPrivate">
81 <xsl:attribute name="value">
82 <xsl:value-of select='concat(" private")'/>
83 </xsl:attribute>
84 <OMLPosition startLine="1" startColumn="1" endLine="1" />
85 </xsl:element>
86 </OMLAccessPrivate>
87 </xsl:if>
88 <xsl:if test="@visibility = 'public'">
89 <OMLAccessPublic>
90 <OMLPosition startLine="1" startColumn="1" endLine="1" />
91 <xsl:element name="OMLKeywordPublic">
92 <xsl:attribute name="value">
93 <xsl:value-of select='concat(" public")'/>
94 </xsl:attribute>
95 <OMLPosition startLine="1" startColumn="1" endLine="1" />
96 </xsl:element>
97 </OMLAccessPublic>
98 </xsl:if>
99 <xsl:if test="@visibility = 'protected'">
100 <OMLAccessProtected>
101 <OMLPosition startLine="1" startColumn="1" endLine="1" />
102 <xsl:element name="OMLKeywordProtected">
103 <xsl:attribute name="value">
104 <xsl:value-of select='concat(" protected")'/>
105 </xsl:attribute>
106 <OMLPosition startLine="1" startColumn="1" endLine="1" />
107 </xsl:element>
108 </OMLAccessProtected>
109 </xsl:if>
110 </OMLAccess>
111
112 <OMLTypeDefinition>
113 <OMLPosition startLine="1" startColumn="1" endLine="1" />
114 <OMLTypeDefinitionType>
115 <OMLPosition startLine="1" startColumn="1" endLine="1" />
116 <xsl:element name="OMLIdentifier">
117 <xsl:attribute name="name">
118 <xsl:value-of select="@name"/>
119 </xsl:attribute>
120 <OMLPosition startLine="1" startColumn="1" endLine="1" />
121 </xsl:element>
122 <xsl:element name="OMLKeywordEqualsign">
123 <xsl:attribute name="value">
124 <xsl:value-of select='concat("=")'/>
125 </xsl:attribute>
126 <OMLPosition startLine="1" startColumn="1" endLine="1" />
127 </xsl:element>
128 <OMLType>
129 <OMLPosition startLine="1" startColumn="1" endLine="1" />
130 <OMLBasicType>
131 <OMLPosition startLine="1" startColumn="1" endLine="1" />
132 <OMLBasicTypeInt>
133 <OMLPosition startLine="1" startColumn="1" endLine="1" />
134 <OMLKeywordInt value="int">
135 <OMLPosition startLine="1" startColumn="1" endLine="1" />
136 </OMLKeywordInt>
137 </OMLBasicTypeInt>
138 </OMLBasicType>
139 </OMLType>
140
141 </OMLTypeDefinitionType></OMLTypeDefinition></OMLAccessTypeDefinition></OMLTypeDefinitionsElement
142 >
143 </xsl:template>
144 </xsl:stylesheet>

```


Appendix J

Tests

This appendix shows some OML specifications that has been used in test of the kernel.

All test cases – including the full systematic functional test – are available on the cd-rom, see Section B. Principles of how the kernel has been tested is described in Appendix 9

J.1 Test – Part2.oml

Listing J.1 shows examples of OML specifications.

Listing J.1: Examples of OML specifications

```
1  -- case200 um           : chapter2/Alarm
2  -- tested using       : -
3  -- expected output   : correctly built tree
4  -- result            : as expected
5  class Alarm
6
7  types
8    public String = seq of char;
9
10 instance variables
11
12   descr      : String;
13   reqQuali   : Expert ' Qualification
14
15 operations
16
17   public Alarm: Expert ' Qualification * String ==> Alarm
18   Alarm(quali , str) ==
19   ( descr := str;
20     reqQuali := quali
21   );
22
```

```
23  public GetReqQuali: () ==> Expert 'Qualification
24  GetReqQuali() ==
25      return reqQuali;
26
27  end Alarm
28
29  — case201 um           : chapter2/Expert
30  — tested using       : —
31  — expected output : correctly built tree
32  — result            : as expected
33  class Expert
34
35  instance variables
36
37      quali : set of Qualification
38
39  types
40
41      public Qualification = <Mech> | <Chem> | <Bio> | <Elec>;
42
43  operations
44
45      public Expert: set of Qualification ==> Expert
46      Expert(qs) ==
47          quali := qs;
48
49      public GetQuali: () ==> set of Qualification
50      GetQuali() ==
51          return quali;
52
53  end Expert
54
55  — case202 um           : chapter6/EmergencyBrake
56  — tested using       : —
57  — expected output : correctly built tree
58  — result            : as expected
59  class EmergencyBrake
60
61  instance variables
62      enabled: bool := false
63
64  operations
65
66      public Enable: () ==> ()
67      Enable() ==
68          enabled := true;
69
70      public IsEnabled: () ==> bool
71      IsEnabled() ==
```

```

72     return enabled;
73
74 end EmergencyBrake
75
76 — case203 um           : chapter6/ObstacleSensor
77 — tested using       : -
78 — expected output   : correctly built tree
79 — result             : as expected
80 class ObstacleSensor
81
82 operations
83
84 public GetData: () ==> set of Controller ‘Obstacle
85 GetData() ==
86     is not yet specified;
87
88 end ObstacleSensor
89
90 — case204 um           : chapter6/PositionSensor
91 — tested using       : -
92 — expected output   : correctly built tree
93 — result             : as expected
94 class PositionSensor
95
96 operations
97 public GetDirection: () ==> Vector
98 GetDirection() ==
99     is not yet specified;
100
101 public GetPosition: () ==> Point
102 GetPosition() ==
103     return Point(1,1,1);
104
105 end PositionSensor
106
107 — case206 m           : chapter6/SteeringController
108 — tested using       : -
109 — expected output   : correctly built tree
110 — result             : as expected
111 class SteeringController
112 operations
113
114 public GetPosition: () ==> Vector
115 GetPosition() ==
116     is not yet specified;
117
118 public SendCommand: Controller ‘SteeringCommands ==> ()
119 SendCommand(-) ==
120     is not yet specified;

```

APPENDIX J. TESTS

```
121 |
122 | sync
123 |
124 | per GetPosition => #active(SendCommand);
125 | per SendCommand => #active(GetPosition)
126 |
127 | end SteeringController
128 |
129 | — case207 m : chapter6/SteeringMonitor
130 | — tested using : —
131 | — expected output : correctly built tree
132 | — result : as expected
133 | class SteeringMonitor
134 |
135 | instance variables
136 |
137 | steering : SteeringController;
138 | emergencyBrake: EmergencyBrake;
139 | active : bool := true;
140 | log : seq of seq of char
141 |
142 | operations
143 |
144 | CheckSteering: () ==> bool
145 | CheckSteering() =
146 | def sp = GetPosition();
147 | in if ExceedsLimits(<Zero>)
148 | then
149 | ( LogCondition(beyondSafetyLimits);
150 | return false;
151 | )
152 | else
153 | ( if abs sp.Angle > MATH*pi/4
154 | then LogCondition(nearSafetyLimit);
155 | return true;
156 | );
157 |
158 | LogCondition: seq1 of char ==> ()
159 | LogCondition(str) =
160 | log := log ^ str;
161 |
162 | thread
163 |
164 | while active do
165 | if not CheckSteering()
166 | then ( emergencyBrake.Enable();
167 | active := false;
168 | )
169 |
```

```

170 end SteeringMonitor
171
172 — case208           : chapter7/Actuator
173 — tested using     : —
174 — expected output  : correctly built tree
175 — result           : as expected
176 class Actuator
177
178 instance variables
179
180 signalshown: CongestionMonitor ‘Signal
181
182 operations
183
184 public GetSignal: () ==> CongestionMonitor ‘Signal
185 GetSignal() ==
186     return signalshown;
187
188 public SetSignal: CongestionMonitor ‘Signal ==> ()
189 SetSignal(signal) ==
190     signalshown := signal;
191
192 end Actuator
193
194 — case209           : chapter7/CongestionMonitor
195 — tested using     : —
196 — expected output  : correctly built tree
197 — result           : as expected
198 class CongestionMonitor is subclass of TrafficControl
199
200 instance variables
201
202 sensor: CongestionSensor;
203 actuator: ActuatorManager;
204 status: CongestionSensor ‘CongestionStatus;
205 previousstatus: CongestionSensor ‘CongestionStatus;
206 operator: OperatorControl
207
208 types
209
210 public Signal = <NoWarning>|<PreAnnouncement>|
211                 <CongestionWarning>;
212
213 operations
214
215 public CongestionMonitor: CWS ‘Location * PassageSensor *
216         ActuatorManager *
217         OperatorControl ==> CongestionMonitor
218 CongestionMonitor(loc , sen , act , op) ==

```

APPENDIX J. TESTS

```

218 | ( location := loc;
219 |   sensor := new CongestionSensor();
220 |   actuator := act;
221 |   status := <NoCongestion>;
222 |   previousstatus := <NoCongestion>;
223 |   operator := op;
224 | );
225 |
226 | public UpdateCongestionStatus: () ==> ()
227 | UpdateCongestionStatus() ==
228 | (
229 |   def newstatus = sensor.IssueCongestionStatus;
230 |   in
231 |     ( cases mk_(previousstatus status, newstatus):
232 |       mk_(-, <NoCongestion>, <Congestion>),
233 |       mk_(<NoCongestion>, <Doubt>, <Congestion>) ->
234 |         ( actuator.ShowSignal(location, <CongestionWarning>);
235 |         ),
236 |       mk_(-, <Congestion>, <NoCongestion>),
237 |       mk_(<Congestion>, <Doubt>, <NoCongestion>) ->
238 |         ( actuator.ShowSignal(location, <NoWarning>);
239 |         )
240 |     )
241 |   );
242 | );
243 |
244 | end CongestionMonitor
245 |
246 | — case210 : chapter7/CongestionSensor
247 | — tested using : —
248 | — expected output : correctly built tree
249 | — result : as expected
250 | class CongestionSensor is subclass of Sensor
251 |
252 | values
253 |
254 | congestionThreshold : nat = 100;
255 | noCongestionThreshold: nat = 150;
256 | noPassages: nat1 = 4
257 |
258 | instance variables
259 |
260 | passageSensor: PassageSensor
261 |
262 | types
263 |
264 | public CongestionStatus = <Congestion>|<NoCongestion>|
265 |   <Doubt>
266 |

```

```

267 operations
268
269 public CongestionSensor: PassageSensor ==>
270                               CongestionSensor
271 CongestionSensor(sensor) ==
272   passageSensor := sensor;
273
274 public IssueCongestionStatus: () ==> CongestionStatus
275 IssueCongestionStatus() ==
276   def averageSpeed = passageSensor .
277                       AverageSpeed
278   in
279     if averageSpeed < congestionThreshold
280     then return <Congestion>
281     elseif averageSpeed > noCongestionThreshold
282     then return <NoCongestion>
283     else return <Doubt>
284
285 end CongestionSensor
286
287 — case211           : chapter7/CWS
288 — tested using     : —
289 — expected output : correctly built tree
290 — result           : as expected
291 class CWS
292
293 types
294
295 public Speed = nat;
296 public Location = nat1;
297
298 instance variables
299
300 roadNetwork: seq of CongestionMonitor := [];
301 sensors      : seq of PassageSensor := [];
302 inv len roadNetwork = len sensors;
303 am: ActuatorManager := new ActuatorManager();
304 op: OperatorControl := new OperatorControl()
305
306 operations
307
308 public AddCongestionMonitor: Location ==> ()
309 AddCongestionMonitor(loc) ==
310 ( def sensor = new PassageSensor();
311   cm = new CongestionMonitor();
312   in
313     let numberOfWarners = len roadNetwork
314     in
315       atomic( roadNetwork := roadNetwork(1, ..., loc);

```

```

316         sensors := sensors (loc , ... , numberOfWarners)
317     );
318     am.AddActuator (loc);
319 );
320
321 end CWS
322
323 — case212           : chapter7/DrivingTimesSensor
324 — tested using     : —
325 — expected output : correctly built tree
326 — result           : as expected
327 class DrivingTimesSensor is subclass of PassageSensor
328
329 operations
330
331 public CarPassingEvent: nat1 ==> ()
332 CarPassingEvent(-) ==
333     is subclass responsibility;
334
335 end DrivingTimesSensor
336
337 — case213           : chapter7/LoopDetector
338 — tested using     : —
339 — expected output : correctly built tree
340 — result           : as expected
341 class LoopDetector is subclass of DrivingTimesSensor
342
343 values
344
345 distanceBetweenLoops: nat1 = 2500;
346
347 operations
348
349 public CarPassingEvent: nat1 ==> ()
350 CarPassingEvent (drivingTime) ==
351     NewPassage (distanceBetweenLoops * drivingTime);
352
353 end LoopDetector
354
355 — case215           : chapter7/OperatorControl
356 — tested using     : —
357 — expected output : correctly built tree
358 — result           : as expected
359 class OperatorControl
360
361 instance variables
362 messageLog: seq of seq1 of char := [];
363 locations : seq of CWS' Location := [];
364 inv len messageLog = len locations

```



```

365
366 operations
367
368 public ResetLog: () ==> ()
369 ResetLog() ==
370   messageLog := [];
371
372 public WriteLog: seq1 of char * CWS' Location ==> ()
373 WriteLog(message, location) ==
374 ( messageLog := messageLog ^
375   ConvertNum2String(location);
376   locations := locations;
377 );
378
379 public CongestionSpots: () ==> set of CWS' Location
380 CongestionSpots() ==
381   return elems locations;
382
383 ConvertLog2File: () ==> seq of char
384 ConvertLog2File() ==
385   return conc messageLog;
386
387 functions
388
389 ConvertNum2String: nat1 -> seq1 of char
390 ConvertNum2String(n) ==
391   is not yet specified;
392
393 end OperatorControl
394
395 — case216           : chapter7/Sensor
396 — tested using    : —
397 — expected output : correctly built tree
398 — result         : as expected
399 class Sensor
400
401 instance variables
402
403 protected location: CWS' Location
404
405 end Sensor
406
407 — case217           : chapter8/CongestionMonitor
408 — tested using    : —
409 — expected output : correctly built tree
410 — result         : as expected
411 class CongestionMonitor is subclass of TrafficControl
412
413 types

```

APPENDIX J. TESTS

```

414 public Signal = <NoWarning>|<PreAnnouncement>|<CongestionWarning>;
415
416 instance variables
417 sensor: CongestionSensor;
418 nameServer: NameServer;
419 status: CongestionSensor 'CongestionStatus := <NoCongestion>;
420 previousstatus: CongestionSensor 'CongestionStatus := <NoCongestion>;
421 operator: OperatorControl
422
423 operations
424
425 public CongestionMonitor: CWS' Location * (inmap CWS' Lane to
      PassageSensor) *
426                               NameServer * OperatorControl ==>
427                               CongestionMonitor
428 CongestionMonitor (loc , sen , ns , op) ==
429 ( location := loc;
430   sensor := new CongestionSensor ();
431   nameServer := ns;
432   operator := op;
433 );
434
435 public UpdateCongestionStatus: () ==> ()
436 UpdateCongestionStatus () ==
437 ( def newstatus = sensor.IssueCongestionStatus ;
438   in
439   ( cases mk_(previousstatus status , newstatus):
440     mk_(-, <NoCongestion>, <Congestion>),
441     mk_(<NoCongestion>, <Doubt>, <Congestion>) ->
442       ( nameServer.GetActuatorManager(location , <CongestionWarning>)
443         ;
444         operator.WriteLog (CongestionOccurred , location);) ,
445     mk_(<Congestion>, <Doubt>, <NoCongestion>) ->
446       ( nameServer.GetActuatorManager(location , <NoWarning>);
447         operator.WriteLog (CongestionResolved , location);)
448   end;
449 );
450 );
451 );
452
453 end CongestionMonitor
454
455 — case218           : chapter8/NameServer
456 — tested using     : —
457 — expected output  : correctly built tree
458 — result          : as expected
459 class NameServer
460

```

```

461 instance variables
462 am: map ActuatorManager to (set of CWS' Location) := {|->}
463
464 operations
465
466 public SetActuatorManager: ActuatorManager *
467                               set of CWS' Location ==> ()
468 SetActuatorManager(actuatorManager, locations) ==
469   am := {actuatorManager |-> locations};
470
471 public SetLocation: ActuatorManager * CWS' Location ==> ()
472 SetLocation(actuatorManager, location) ==
473   am(actuatorManager) := {location}
474 pre actuatorManager in set dom am;
475
476 public GetActuatorManager: [CWS' Location] ==> [ActuatorManager]
477 GetActuatorManager(loc) ==
478   if loc = nil
479   then return nil
480   else let locations = inverse am
481         in
482           let locationSet in set dom locations be st
483             loc in set locationSet
484           in
485             return locations (locationSet);
486
487 public GetLocations: () ==> set of CWS' Location
488 GetLocations() ==
489   return dunion rng am;
490
491 end NameServer
492
493 — case219 um : chapter9/Full/Alphabet
494 — tested using : —
495 — expected output : correctly built tree
496 — result : as expected
497 class Alphabet
498
499 instance variables
500 alph : seq of char := [];
501
502 inv AlphabetInv(alph)
503
504 functions
505 AlphabetInv: seq of char -> bool
506 AlphabetInv (palph) ==
507   len palph mod 2 = 0 and
508   card elems palph = len palph
509

```

```

510 operations
511 public Alphabet: seq of char ==> Alphabet
512 Alphabet (pa) = alph := pa
513 pre AlphabetInv(pa);
514
515 public GetChar: nat ==> char
516 GetChar (pidx) = return alph(pidx)
517 pre pidx in set inds alph;
518
519 public GetIndex: char ==> nat
520 GetIndex (pch) =
521   let pidx in set {i | i in set inds alph
522     & alph(i) = pch} in
523     return pidx
524 pre pch in set elems alph;
525
526 public GetIndices: () ==> set of nat
527 GetIndices () = return inds alph;
528
529 public GetSize: () ==> nat
530 GetSize () = return len alph;
531
532 public Shift: nat * nat ==> nat
533 Shift (pidx, poffset) =
534   if pidx + poffset > len alph
535   then return pidx + poffset - len alph
536   else return pidx + poffset
537 pre pidx in set inds alph and
538   poffset <= len alph;
539
540 public Shift: nat ==> nat
541 Shift (pidx) = Shift(pidx, 1)
542 end Alphabet
543
544 — case221 um : chapter9/Full/SimpleEnigma
545 — tested using : —
546 — expected output : correctly built tree
547 — result : as expected
548 class SimpleEnigma
549   is subclass of Component
550
551 values
552   refcfg : inmap nat to nat =
553     {1 |-> 3, 2 |-> 4};
554   rotcfg : inmap nat to nat =
555     {1 |-> 2, 2 |-> 4, 3 |-> 3, 4 |-> 1};
556   pbcfg : inmap nat to nat =
557     {2 |-> 3}
558

```

```

559 operations
560   public SimpleEnigma: () ==> SimpleEnigma
561   SimpleEnigma () ==
562     (dcl cp : Component ;
563     alph := new Alphabet ();
564     next := new Reflector ();
565     cp := new Rotor ();
566     cp.SetNext(next);
567     next := cp;
568     cp := new Rotor ();
569     cp.SetNext(next);
570     next := cp;
571     cp := new Rotor ();
572     cp.SetNext(next);
573     next := cp;
574     cp := new Plugboard ();
575     cp.SetNext(next);
576     next := cp);
577
578 end SimpleEnigma
579
580 — case222 um : chapter9/Full/TestCase
581 — tested using : —
582 — expected output : correctly built tree
583 — result : as expected
584 class TestCase
585   is subclass of Test
586
587 instance variables
588   name : seq of char
589
590 operations
591   public TestCase: seq of char ==> TestCase
592   TestCase(nm) == name := nm;
593
594   public GetName: () ==> seq of char
595   GetName () == return name;
596
597   protected AssertTrue: bool ==> ()
598   AssertTrue (pb) == if not pb then exit <FAILURE>;
599
600   protected AssertFalse: bool ==> ()
601   AssertFalse (pb) == if pb then exit <FAILURE>;
602
603   public Run: TestResult ==> ()
604   Run (ptr) ==
605     trap <FAILURE>
606     with
607       ptr.AddFailure(self)

```

```

608     in
609         (SetUp());
610         RunTest();
611         TearDown());
612
613     protected SetUp: () ==> ()
614     SetUp () = is subclass responsibility;
615
616     protected RunTest: () ==> ()
617     RunTest () = is subclass responsibility;
618
619     protected TearDown: () ==> ()
620     TearDown () = is subclass responsibility
621
622 end TestCase
623
624 — case223 : chapter9/Full/TestCase
625 — tested using : —
626 — expected output : correctly built tree
627 — result : as expected
628 class TestSuite
629     is subclass of Test
630
631 instance variables
632     tests : seq of Test := []
633
634 operations
635     public Run: () ==> ()
636     Run () =
637         (dcl ntr : TestResult := new TestResult());
638         Run(ntr);
639         ntr.Show());
640
641     public Run: TestResult ==> ()
642     Run (result) =
643         for test in tests do
644             test.Run(result);
645
646     public AddTest: Test ==> ()
647     AddTest(test) =
648         tests := tests ^ test;
649
650 end TestSuite
651
652 — case224 : chapter9/Initial/Enigma
653 — tested using : —
654 — expected output : correctly built tree
655 — result : as expected
656 class Enigma

```

```

657 operations
658
659   public Keystroke : char ==> char
660   Keystroke(-) ==
661     is not yet specified;
662
663 instance variables
664   plugboard : Plugboard
665
666 end Enigma
667
668   — case225           : chapter9/Full/Plugboard
669   — tested using     : —
670   — expected output : correctly built tree
671   — result           : as expected
672
673 class Plugboard
674 operations
675
676   private Decode : nat ==> nat
677   Decode(-) ==
678     is not yet specified;
679
680   private Encode : nat ==> nat
681   Encode(-) ==
682     is not yet specified;
683
684   public Substitute : nat ==> nat
685   Substitute(-) ==
686     is not yet specified;
687
688
689 instance variables
690   first_rotor : Rotor;
691   config : inmap nat to nat
692
693 end Plugboard
694
695   — case226           : chapter9/Full/Reflector
696   — tested using     : —
697   — expected output : correctly built tree
698   — result           : as expected
699
700 class Reflector
701
702 instance variables
703   cur_pos : nat;
704   config : inmap nat to nat
705

```

```
706 operations
707
708   private Decode : nat  $\implies$  nat
709   Decode(-) ==
710     is not yet specified;
711
712   private Encode : nat  $\implies$  nat
713   Encode(-) ==
714     is not yet specified;
715
716   public Substitute : nat  $\implies$  nat
717   Substitute(-) ==
718     is not yet specified;
719
720 end Reflector
721
722 — case227 : chapter9/Full/Rotor
723 — tested using : —
724 — expected output : correctly built tree
725 — result : as expected
726
727 class Rotor
728 operations
729
730   private Decode : nat  $\implies$  nat
731   Decode(-) ==
732     is not yet specified;
733
734   private Encode : nat  $\implies$  nat
735   Encode(-) ==
736     is not yet specified;
737
738   public Rotate : nat  $\implies$  nat
739   Rotate(-) ==
740     is not yet specified;
741
742   public Substitute : nat  $\implies$  nat
743   Substitute(-) ==
744     is not yet specified;
745
746
747 instance variables
748   next_rotor : Rotor;
749   reflector : Reflector;
750   cur_pos : nat;
751   config : inmap nat to nat;
752   latch_pos : nat
753
754 end Rotor
```



```

755
756 — case228 : chapter10/AnnounceBeacon
757 — tested using : —
758 — expected output : correctly built tree
759 — result : as expected
760 class AnnounceBeacon is subclass of Beacon
761
762 instance variables
763 targetspeed: real
764
765 operations
766
767 public AnnounceBeacon: real ==> AnnounceBeacon
768 AnnounceBeacon(ts) ==
769     targetspeed := ts;
770
771 public GetTargetSpeed : () ==> real
772 GetTargetSpeed() ==
773     return targetspeed;
774
775 end AnnounceBeacon
776
777 — case229 : chapter10/CabDisplay
778 — tested using : —
779 — expected output : correctly built tree
780 — result : as expected
781 class CabDisplay
782
783 instance variables
784
785 instance variables
786
787 alarm : bool := false;
788 emergencyBrake: bool := false;
789 groundFault : bool := false
790
791 operations
792
793 public SetAlarm: () ==> ()
794 SetAlarm() ==
795     alarm := true
796     pre not emergencyBrake and not groundFault;
797
798 public UnsetAlarm: () ==> ()
799 UnsetAlarm() ==
800     alarm := false;
801
802 public SetEmergencyBrake: () ==> ()
803 SetEmergencyBrake() ==

```

APPENDIX J. TESTS

```

804 | ( alarm := false;
805 |   emergencyBrake := true ););
806 |
807 | public UnsetEmergencyBrake: () ==> ()
808 | UnsetEmergencyBrake() ==
809 |   emergencyBrake := false;
810 |
811 | public SetGroundFault: () ==> ()
812 | SetGroundFault() ==
813 |   groundFault := true;
814 |
815 | public UnsetGroundFault: () ==> ()
816 | UnsetGroundFault() ==
817 |   groundFault := false;
818 |
819 | public GetDisplay: () ==> bool
820 | GetDisplay() ==
821 |   return (display);
822 |
823 | end CabDisplay
824 |
825 | — case230           : chapter10/Driver
826 | — tested using     : —
827 | — expected output : correctly built tree
828 | — result           : as expected
829 | class Driver
830 |
831 | instance variables
832 |
833 | faults: set of Fault := {}
834 |
835 | types
836 |
837 | public DriverId = token;
838 |
839 | operations
840 |
841 | public AddFaults: set of Fault ==> ()
842 | AddFaults(newfaults) ==
843 |   faults := faults union newfaults;
844 |
845 | public GetFaults: () ==> set of Fault
846 | GetFaults() ==
847 |   return faults;
848 |
849 | end Driver
850 |
851 | — case231           : chapter10/DriverCard
852 | — tested using     : —

```

```
853 — expected output : correctly built tree
854 — result : as expected
855 class DriverCard
856
857 instance variables
858
859 id: DriverId
860
861 types
862
863 public DriverId = token;
864
865 operations
866
867 public SetId: DriverId ==> ()
868 SetId(newid) ==
869     id := newid;
870
871 public GetId: () ==> DriverId
872 GetId() ==
873     return id;
874
875 end DriverCard
876
877 — case232 : chapter10/EmergencyBrake
878 — tested using : -
879 — expected output : correctly built tree
880 — result : as expected
881 class EmergencyBrake
882
883 instance variables
884
885 emergencybrake: bool := false
886
887 operations
888
889 public SetEmergencyBrake: () ==> ()
890 SetEmergencyBrake() ==
891     emergencybrake := true;
892
893 public UnsetEmergencyBrake: () ==> ()
894 UnsetEmergencyBrake() ==
895     emergencybrake := false;
896
897 public GetEmergencyBrake: () ==> bool
898 GetEmergencyBrake() ==
899     return emergencybrake;
900
901 end EmergencyBrake
```

APPENDIX J. TESTS

```
902 |
903 | — case233 : chapter10/Event
904 | — tested using : —
905 | — expected output : correctly built tree
906 | — result : as expected
907 |
908 | class Event
909 |
910 | operations
911 |
912 | public Execute: CSL  $\implies$  Test ‘TestResult
913 | Execute(csl) ==
914 |   is subclass responsibility;
915 |
916 | end Event
917 |
918 | — case234 : chapter10/Fault
919 | — tested using : —
920 | — expected output : correctly built tree
921 | — result : as expected
922 | class Fault
923 |
924 | instance variables
925 |
926 | speedlimit : real;
927 | actualspeed : real
928 |
929 | operations
930 |
931 | public Fault: real * real  $\implies$  Fault
932 | Fault(max, act) ==
933 | ( speedlimit := max;
934 |   actualspeed := act;
935 | );
936 |
937 | end Fault
938 |
939 | — case235 : chapter10/LimitBeacon
940 | — tested using : —
941 | — expected output : correctly built tree
942 | — result : as expected
943 | class LimitBeacon is subclass of Beacon
944 |
945 | instance variables
946 |
947 | speed: [real] := nil
948 |
949 | operations
950 |
```

```

951 public SetSpeedRestriction: real ==> ()
952 SetSpeedRestriction(s) ==
953   speed := s;
954
955 public GetSpeedRestriction: () ==> real
956 GetSpeedRestriction() ==
957   return speed
958 pre speed <> nil;
959
960 end LimitBeacon
961
962 — case236           : chapter10/MaxSpeedEvent
963 — tested using     : —
964 — expected output : correctly built tree
965 — result          : as expected
966 class MaxSpeedEvent is subclass of Event
967
968 operations
969
970 public Execute: CSL ==> Test ‘TestResult
971 Execute(csl) ==
972 ( let ms = GetMaxSpeed()
973   in
974     return mk_Test ‘MaxSpeed(ms);
975 );
976
977 end MaxSpeedEvent
978
979 — case237           : chapter12-13/MessageChannelBuffer
980 — tested using     : —
981 — expected output : correctly built tree
982 — result          : as expected
983 class MessageChannelBuffer
984
985 instance variables
986 data : [MessageChannel] := nil
987
988 operations
989 public Put: MessageChannel ==> ()
990 Put(msg) ==
991   data := msg;
992
993 public Get: () ==> MessageChannel
994 Get() ==
995 let d = data in
996   ( data := nil;
997     return d;
998   );
999

```

APPENDIX J. TESTS

```
1000 sync
1001
1002 per Get => data <> nil;
1003 per Put  => data = nil
1004
1005 sync
1006   mutex(Put, Get);
1007   mutex(Put);
1008   mutex(Get)
1009
1010 end MessageChannelBuffer
1011
1012 — case238           : chapter12-13/POP3Message
1013 — tested using     : —
1014 — expected output : correctly built tree
1015 — result          : as expected
1016 class POP3Message
1017
1018 instance variables
1019   header : seq of char;
1020   body   : seq of char;
1021   deleted : bool;
1022   uniqueId : seq of char
1023
1024
1025 operations
1026
1027 public POP3Message: seq of char * seq of char * seq of char ==>
1028   POP3Message
1029   POP3Message(nheader, nbody, nuniqueId) ==
1030   ( header := nheader;
1031     body   := nbody;
1032     deleted := false;
1033     uniqueId := nuniqueId;
1034   );
1035
1036 public GetBody: () ==> seq of char
1037   GetBody() ==
1038   return body;
1039
1040 public GetHeader: () ==> seq of char
1041   GetHeader() ==
1042   return header;
1043
1044 public GetText: () ==> seq of char
1045   GetText() ==
1046   return header ^"\n"^body;
1047
1048 public Delete: () ==> POP3Message
```

```

1048 Delete () ==
1049 ( deleted := true;
1050   return self;
1051 );
1052
1053 public IsDeleted: () ==> bool
1054 IsDeleted () ==
1055   return deleted;
1056
1057 public Undelete: () ==> POP3Message
1058 Undelete () ==
1059 ( deleted := false;
1060   return self;
1061 );
1062
1063 public GetSize: () ==> nat
1064 GetSize () ==
1065   return len body + len header;
1066
1067 public GetUniqueId: () ==> seq of char
1068 GetUniqueId () ==
1069   return uniqueId;
1070
1071 end POP3Message
1072
1073 — case239           : chapter12-13/POP3Types
1074 — tested using     : -
1075 — expected output : correctly built tree
1076 — result          : as expected
1077 class POP3Types
1078 types
1079 public ClientCommand = StandardClientCommand |
1080                        OptionalClientCommand;
1081 public StandardClientCommand = QUIT | STAT | LIST | RETR | DELE | NOOP
1082   | RSET;
1083 public OptionalClientCommand = TOP | UIDL | USER | PASS | APOP;
1084 public QUIT :: ;
1085 public STAT :: ;
1086 public LIST :: messageNumber : [nat];
1087 public RETR :: messageNumber : nat;
1088 public DELE :: messageNumber : nat;
1089 public NOOP :: ;
1090 public RSET :: ;
1091 public TOP :: messageNumber : nat
1092               numLines      : nat;
1093 public UIDL :: messageNumber : [nat];
1094 public USER :: name : UserName;
1095 public PASS :: string : seq of char;
1096 public APOP :: name : seq of char

```

APPENDIX J. TESTS

```
1096         digest : seq of char;  
1097 public UserName = seq of char;  
1098 public Password = seq of char;  
1099 public ServerResponse = OkResponse | ErrResponse;  
1100 public OkResponse :: data : seq of char;  
1101 public ErrResponse :: data : seq of char;  
1102  
1103 end POP3Types
```