# 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. Kernen tilbyder funktionalitet til at parse en OML specifiation og opbygge et AST (Abstrakt Syntaks Træ), gendanne kildekoden fra et AST, samt mulighed for at eksportere og importere et AST til/fra XML. Kernen 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 kernen 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

# List of Figures

# 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 stategies 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 a 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

```
class Alarm

types
  public String = seq of char;

instance variables

  descr     : String;
  reqQuali  : Expert`Qualification

operations

  public Alarm: Expert`Qualification * String ==> Alarm
  Alarm(quali,str) ==
  ( descr := str;
    reqQuali := quali
  );

  public GetReqQuali: () ==> Expert`Qualification
  GetReqQuali() ==
    return reqQuali;

end Alarm

class Expert

instance variables

  quali : set of Qualification

types

  public Qualification = <Mech> | <Chem> | <Bio> | <Elec>;

operations

  public Expert: set of Qualification ==> Expert
  Expert(qs) ==
    quali := qs;

  public GetQuali: () ==> set of Qualification
  GetQuali() ==
    return quali;

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. FME[1] 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.

[1] Formal 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.

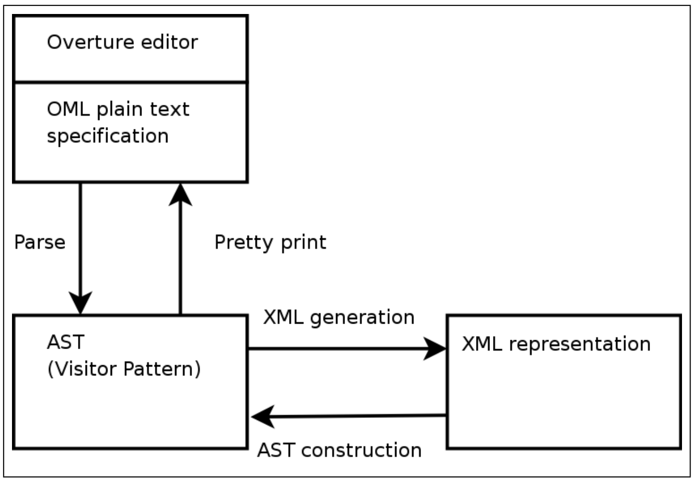



Figure 3.1: Overview of what the kernel should include

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 specification[1] 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.

[1]The 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 week, 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

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

Listing 4.2: XML code example of an extension

```
<extension point="org.overturetool.eclipse.editor.extensionParser">
   <parser
      name="ParserExtension"
      class="org.overturetool.eclipse.parser.OvertureParser"
      id="org.overturetool.eclipse.parser.OvertureParser">
   </parser>
</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 facitities 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

```
<extension
      point="org.eclipse.ui.perspectives">
   <perspective
         name="Overture Perspective"
         icon="icons/sample.gif"
         class="org.overturetool.eclipse.editor.OverturePerspective"
         id="org.overturetool.eclipse.editor.OverturePerspective">
   </perspective>
</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.

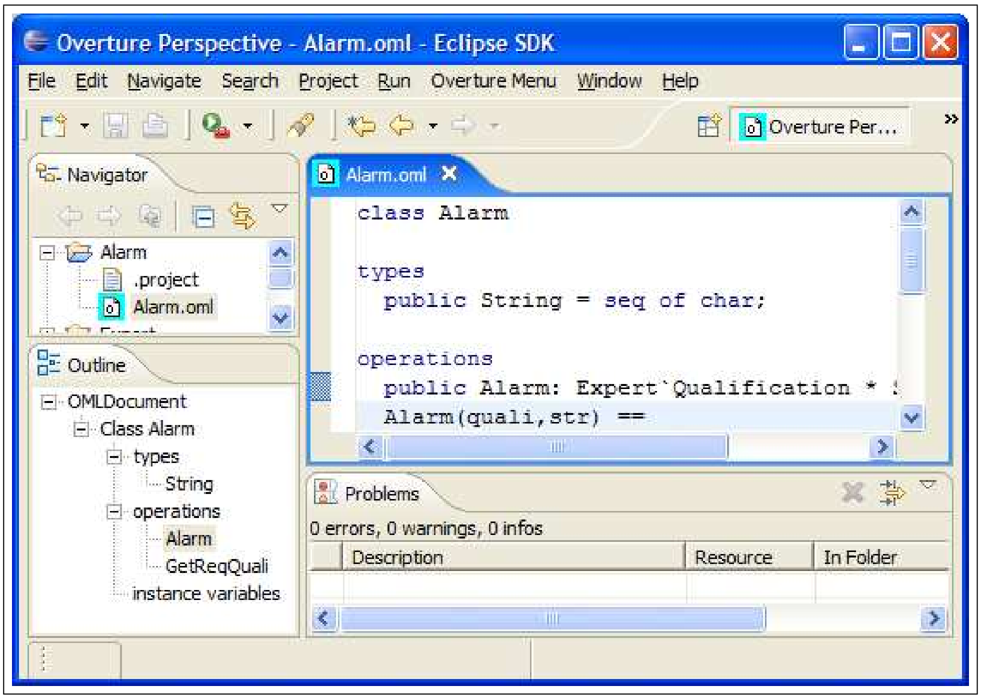



Figure 4.1: Overview of the the Eclipse GUI

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 pathes. 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 pathes, 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.

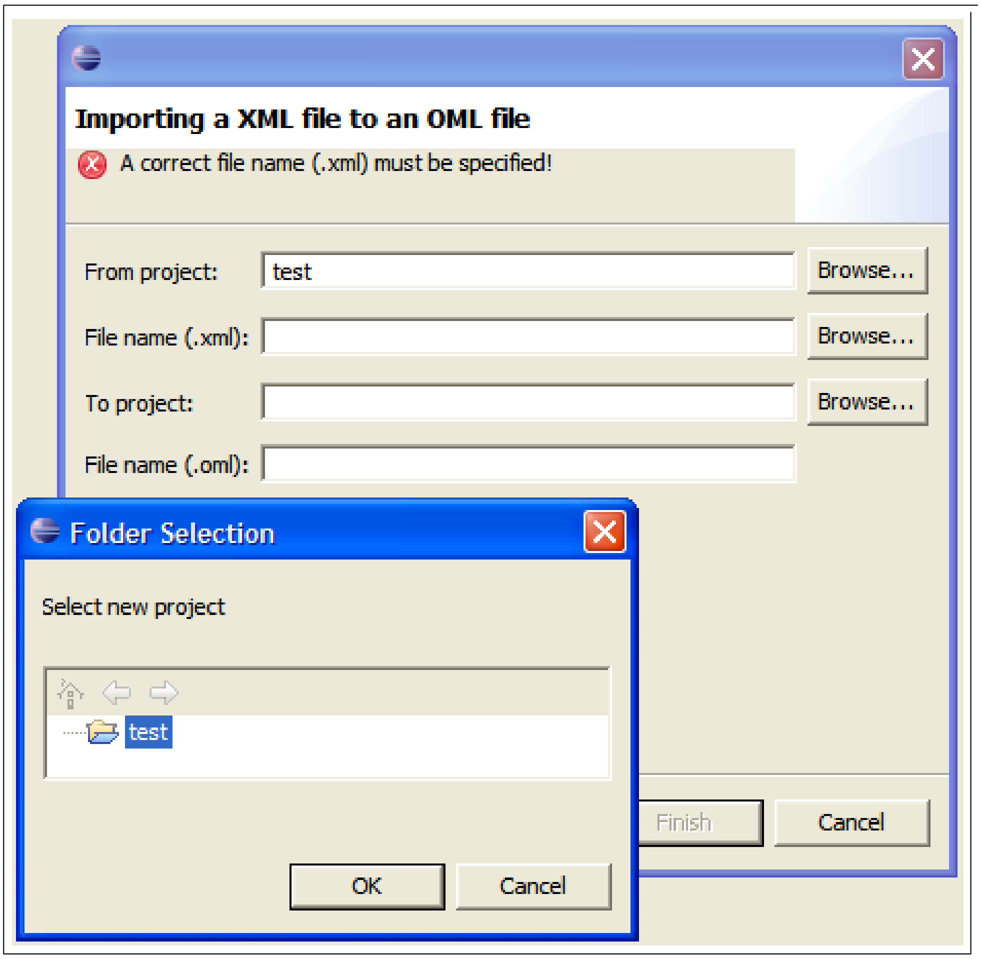



Figure 4.2: Example of a customized wizard

### 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 occurences 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

```
type definition = identifier , '=', type;

type = bracketed type
     | basic type;

bracketed type = '(', type, ')';

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

```
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 approach[1] 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.

[1]Used 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.

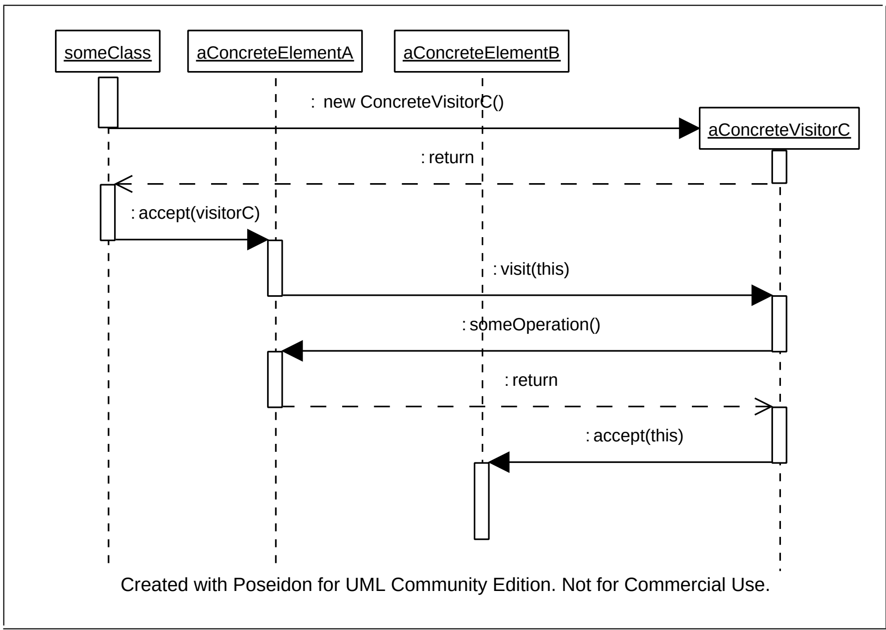



Figure 5.1: Sequence diagram: Example of visitor interaction

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 button up parser starts by looking at the tokens, tries to mach 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, button 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).

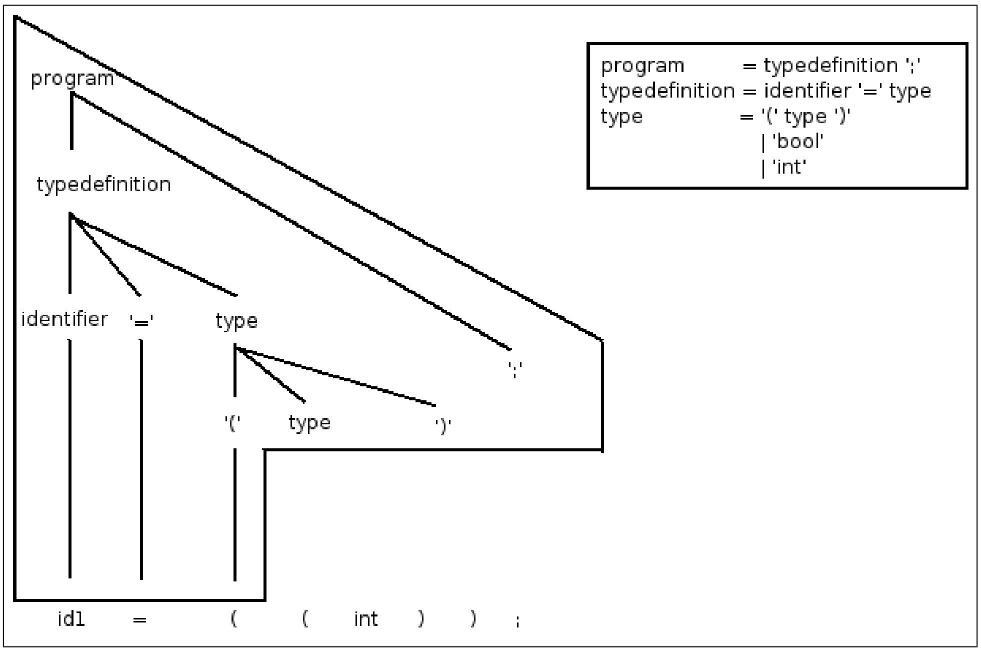



Figure 5.2: Illustration of top down parsing

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 producitons.

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.

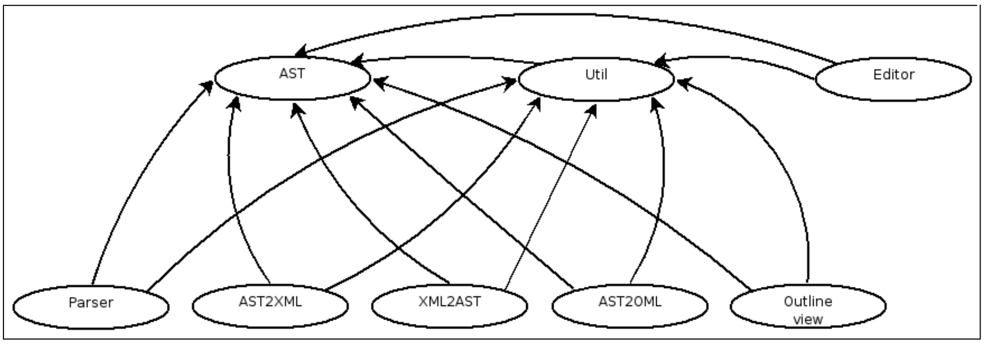



Figure 7.1: Overview of dependencies

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.

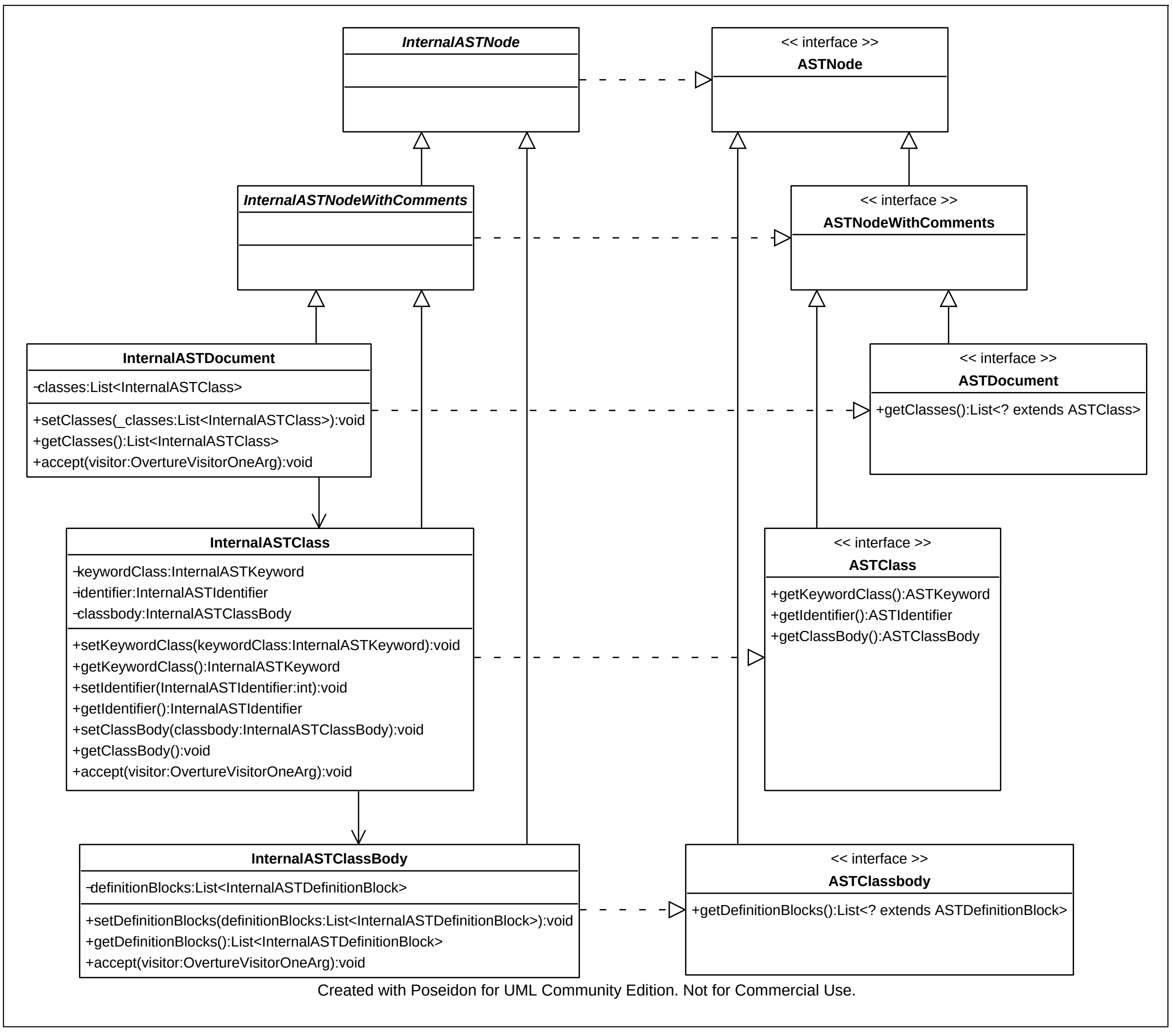



Figure 7.2: Example of selected AST classes based on inheritance and interfaces

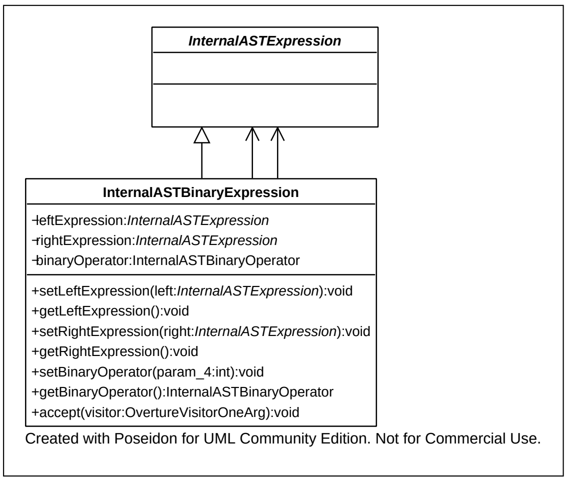



Figure 7.3: Example of an AST with inheritance

### 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.

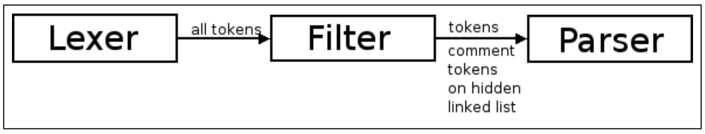



Figure 7.4: Filtering comments - Filter between lexer and parser

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

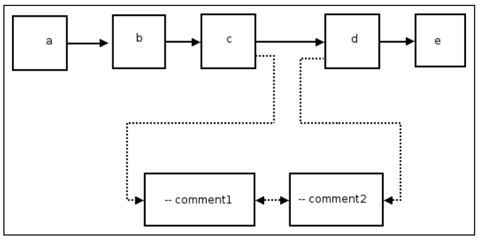



Figure 7.5: Filtering comments - Comments stored in hidden linked list

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 note, `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 Spy[1][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:

[1]Here 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.

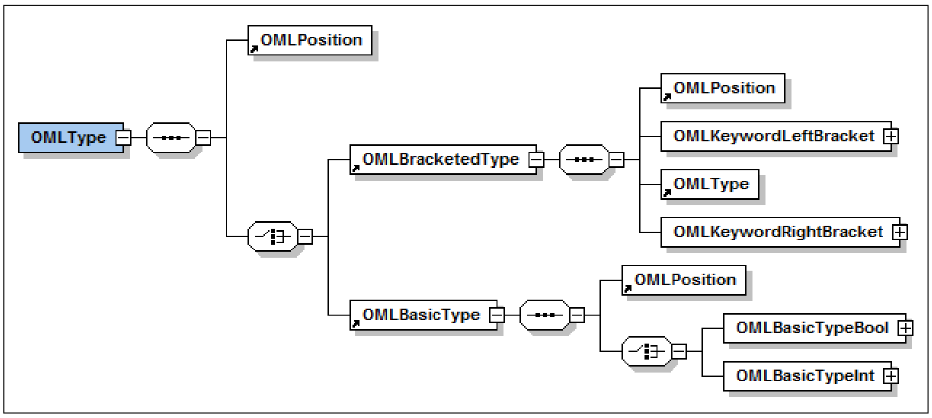



Figure 7.6: XML Schema example (XML-Spy diagram notation)

OMLPosition, OMLIdentifier, OMLKeywordEqualsign and a choice between OMLBracketedType and OMLBasicType. An OMLComments element is optional and therefore has a doted 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 OML-Comment. The element called OMLComment can contain serveral elements of type OMLComment. Figure 7.7 shows how OMLComments elements is related to other elements.

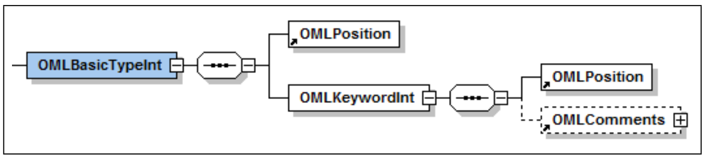



Figure 7.7: XML Schema example (How comments are stored)

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 implantations 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

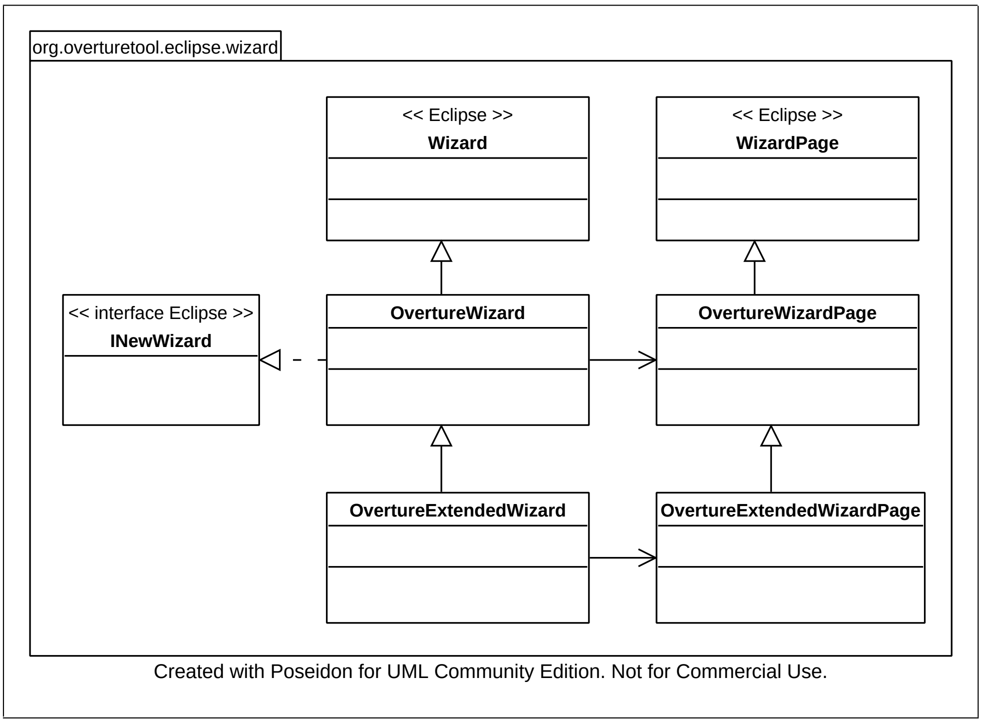



Figure 7.8: Eclipse wizard overview - Class diagram

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.

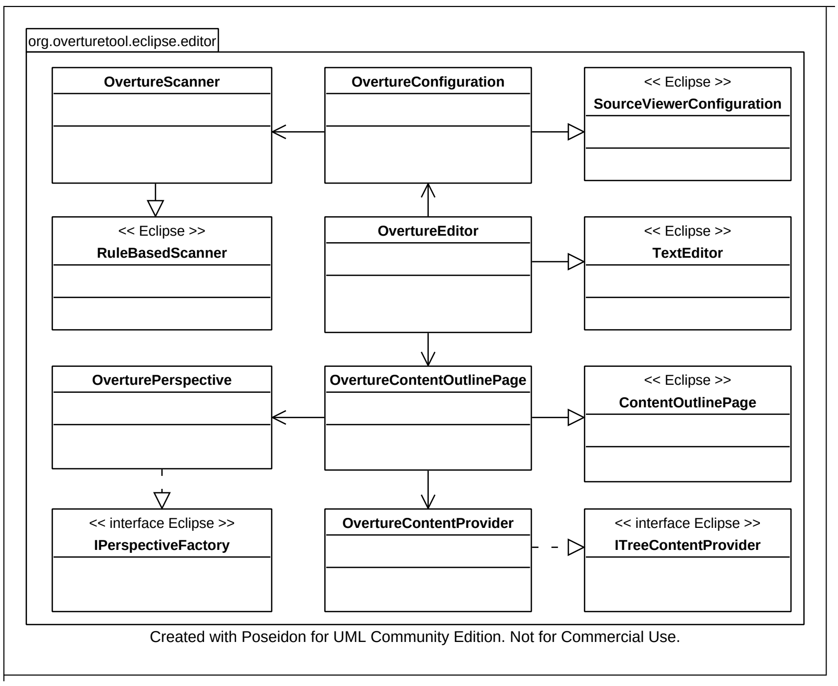



Figure 7.9: Eclipse editor overview - The most important use of classes

Figure 7.10 shows how a menu item to an editors menu can start execution 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.

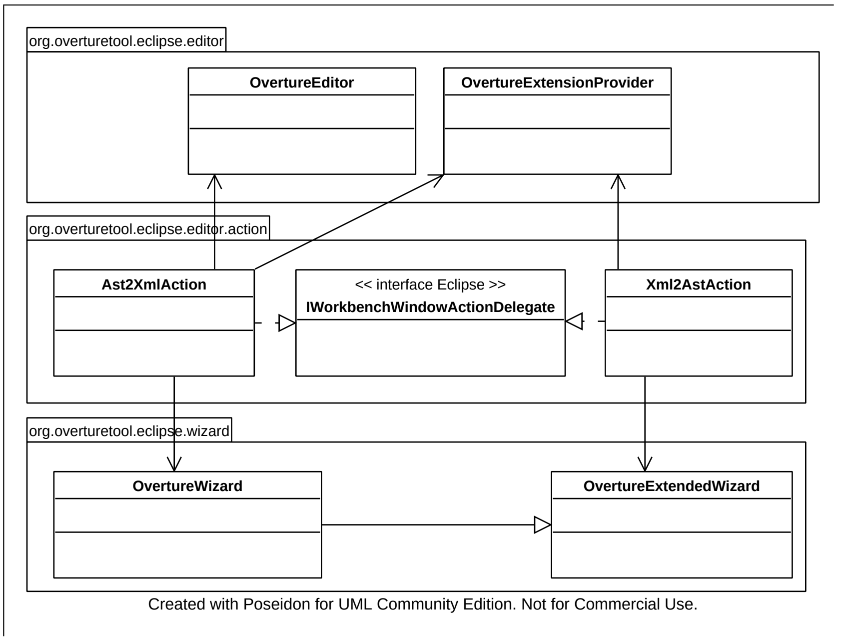



Figure 7.10: Eclipse action overview - Class diagram

## 7.11 Design of Extension Possibilities

A central point is how to design the extension possibilities. We have created a design, shown in figreffigure: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.

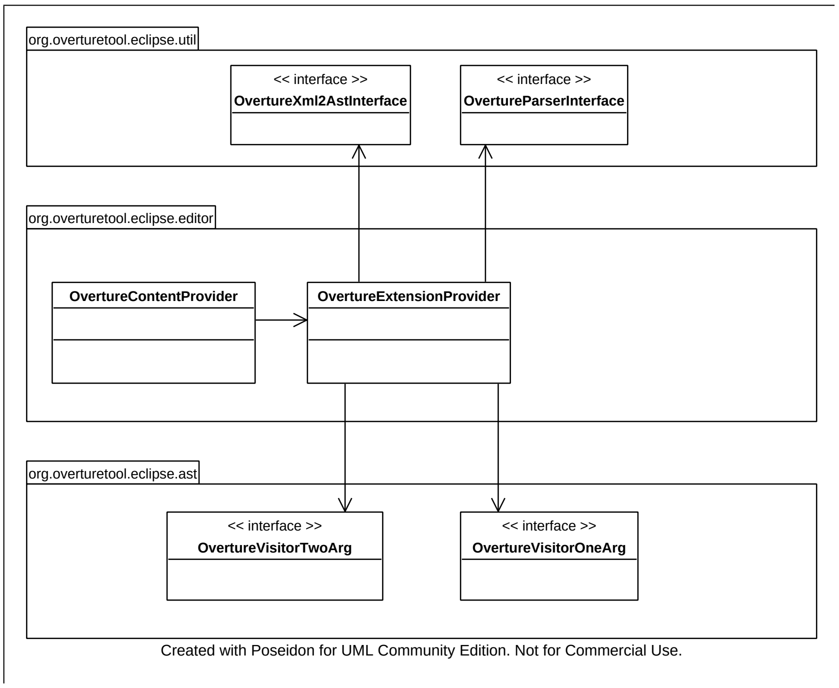



Figure 7.11: Eclipse editor overview - Loading classes for extensions

## 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-in[1]. 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 a 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

```
...
public var1    -- Comment1
       = int ; -- Comment2
               -- Comment3
...
```

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

[1] org.overturetool.eclipse.ast
[2] org.overturetool.eclipse.ast
[3] org.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 constructer 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

```
...
internalASTAccessTypeDefinition.setStartPositionFromNode(
    internalASTAccess);
...
internalASTAccessTypeDefinition.setEndPositionFromNode(
    internalASTTypeDefinition);
...
```

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

```java
package org.overturetool.eclipse.ast;

import java.util.List;

public interface ASTDocument extends ASTNodeWithComments {
    List<? extends ASTClass> getClasses();
}
```

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 specifications. 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
9          | internalASTType = overtureTypePre3
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 convensions.

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

```
expressionPlusRule returns [ExpressionPlus expressionPlus] :
expression1 = expressionHigherPrecedence
PLUS
expression2 = expressionPlusRule
{ expressionPlus = new ExpressionPlus(expression1, expression2);
}
;
```

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

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

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

Listing 8.8: Example of an accept method in InternalASTClass

```
public void accept(OvertureVisitorOneArg visitor) {
     visitor.visit(this);
}
```

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

```
public class OvertureAst2XmlVisitor implements OvertureVisitorTwoArg<
    Element>{

    public void visit(ASTDocument astDocument, Element parent) {

        storePosition( astDocument, parent);
        for (ASTClass astClass : astDocument.getClasses()) {
            astClass.accept(this, parent);
        }
    }

    public void visit(ASTClass astClass, Element parent) {

        Element omlClass = new Element("OMLClass");
        storePosition( astClass, omlClass);
        astClass.getKeywordClass().accept(this, omlClass);
        astClass.getIdentifier().accept(this, omlClass);
        ASTInheritanceClause inheritanceClause = astClass.
            getInheritanceClause();
        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          storePosition( astIdentifier, omlIdentifier);
30          astIdentifier.getComments().accept(this, omlIdentifier);
31          parent.addContent(omlIdentifier);
32      }
33
34      public void visit(ASTKeyword astKeyword, Element parent){
35
36          Element omlKeyword = new Element("OMLKeyword" + astKeyword.getID
                ());
37          omlKeyword.setAttribute("value", astKeyword.getValue());
38          storePosition( astKeyword, omlKeyword);
39          astKeyword.getComments().accept(this, omlKeyword);
40          parent.addContent(omlKeyword);
41      }
42
43      public void storePosition(ASTNode astNode, Element parent) {
44
45          Element omlPosition = new Element("OMLPosition");
46          omlPosition.setAttribute("startLine",astNode.getStartLine()+"");
47          omlPosition.setAttribute("startColumn",astNode.getStartColumn()+"
                ");
48          omlPosition.setAttribute("endLine",astNode.getEndLine()+"");
49          parent.addContent(omlPosition);
50      }
51  }
```

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 is 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

```
class simple
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

```
<?xml version="1.0" encoding="UTF-8"?>
<OMLDocument>
  <OMLPosition startLine="1" startColumn="1" endLine="2" />
  <OMLClass>
    <OMLPosition startLine="1" startColumn="1" endLine="2" />
    <OMLKeywordClass value="class">
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </OMLKeywordClass>
    <OMLIdentifier name="simple">
      <OMLPosition startLine="1" startColumn="7" endLine="1" />
    </OMLIdentifier>
    <OMLKeywordEnd value="end">
      <OMLPosition startLine="2" startColumn="1" endLine="2" />
    </OMLKeywordEnd>
    <OMLIdentifier2>
      <OMLIdentifier name="simple">
        <OMLPosition startLine="2" startColumn="5" endLine="2" />
      </OMLIdentifier>
    </OMLIdentifier2>
  </OMLClass>
</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

```java
public String visit(ASTIdentifier astIdentifier){
    setPosition(astIdentifier);
    doc = doc + astIdentifier.getIdentifierName();
    columnCount = columnCount + astIdentifier.getIdentifierName().
        length();
    setComments(astIdentifier);
}
```

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

```java
public String visit(ASTNode astNode, String parent){
   int startLine = astNode.getStartLine();
   int startColumn = astNode.getStartColumn();
   while (lineCount < startLine){
      parent = parent + "/n";
      lineCount++;
```

[4] org.overturetool.eclipse.util

```
        columnCount = 1;
    }
    while (columnCount < startColumn){
        parent = parent + " ";
        columnCount++;
    }
    return parent;
}
```

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

```
public class OvertureXml2AstConverter implements
    OvertureXml2AstConverterInterface{

     public final InternalASTDocument overtureXMLDocument(Element
         xmlDocument){
         InternalASTDocument internalASTDocument = null;

         internalASTDocument = new InternalASTDocument(xmlDocument);
         List<InternalASTClass> internalASTClassList =
             internalASTDocument.getClasses();
         List<Element> classlist = xmlDocument.getChildren("OMLClass");
         for(Element xmlClass : classlist){
              InternalASTClass internalASTClass = overtureXMLClass(
                  xmlClass);
              internalASTClassList.add(internalASTClass);
         }
         return internalASTDocument;
     }
```

```

    public final InternalASTClass overtureXMLClass(Element xmlClass){
        InternalASTClass internalASTClass = null;

        internalASTClass = new InternalASTClass(xmlClass);

        Element xmlKeywordClass = xmlClass.getChild("OMLKeywordClass");
        InternalASTKeyword internalASTKeywordClass = new
            InternalASTKeyword(xmlKeywordClass);
        internalASTClass.setKeywordClass(internalASTKeywordClass);

        Element xmlIdentifier1 = xmlClass.getChild("OMLIdentifier");
        InternalASTIdentifier internalASTIdentifier1 = new
            InternalASTIdentifier(xmlIdentifier1);
        internalASTClass.setIdentifier(internalASTIdentifier1);

        Element xmlKeywordEnd = xmlClass.getChild("OMLKeywordEnd");
        InternalASTKeyword internalASTKeywordEnd = new
            InternalASTKeyword(xmlKeywordEnd);
        internalASTClass.setKeywordEnd(internalASTKeywordEnd);

        Element xmlIdentifier2 = xmlClass.getChild("OMLIdentifier2");
        Element xmlIdentifierEnd = xmlIdentifier2.getChild("
            OMLIdentifier");
        InternalASTIdentifier internalASTIdentifier2 = new
            InternalASTIdentifier(xmlIdentifierEnd);
        internalASTClass.setIdentifierEnd(internalASTIdentifier2);

        return internalASTClass;
    }
}
```

First a new `InternalASTDocument` node is created. The position and string information in the XML element is transferred in the constructer 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 `InteralASTDocument`.

## 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

```
<?xml version="1.0" encoding="UTF-8"?>
<?eclipse version="3.0"?>
<plugin
   id="org.overturetool.eclipse.editor"
   name="Overture Editor Plug-in"
   version="0.0.1"
   provider-name="Overture"
   class="org.overturetool.eclipse.editor.OvertureEditorPlugin">
   <runtime>
      <library name="antlr.jar"><export name="*"/></library>
   </runtime>
   <requires>
      <import plugin="org.eclipse.ui"/>
   </requires>
   <extension-point id="extensionParser" name="ExtPoint.extensionParser
      "/>
   <extension point="org.eclipse.ui.editors">
      <editor
         name="Overture Editor"
         extensions="oml"
         icon="icons/sample.gif"
         class="org.overturetool.eclipse.editor.OvertureEditor"
         id="org.overturetool.eclipse.editor.OvertureEditor">
      </editor>
   </extension>
   <extension point="org.eclipse.ui.perspectives">
```

```
        <perspective
            name="Overture Perspective"
            icon="icons/sample.gif"
            class="org.overturetool.eclipse.editor.OverturePerspective"
            id="org.overturetool.eclipse.editor.OverturePerspective">
        </perspective>
    </extension>
</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-in[5]. Diagrams presenting a design overview is given in Section 7.10. One of the packages [6] 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

[5] org.overturetool.eclipse.editor

[6] org.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 parser 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

```
private ASTDocument parseOML(IDocument document) {
        OvertureParserInterface overtureParser = null;
        try{
            overtureParser = overtureExtensionProvider.loadParser();
            astDocument = (InternalASTDocument) overtureParser.parse(
                new ByteArrayInputStream(document.get().getBytes()));
            errors = overtureParser.getErrors();
            if (!errors.isEmpty()) {
                for (OvertureParseException error: errors) {
                    addMarker(input.getFile(),error.getError(),
                            new Integer(error.getLine()),
                            new Integer(IMarker.SEVERITY_ERROR));
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return astDocument;
    }

    private void addMarker(IFile file, String message, int lineNumber,
        int severity) {
        try {
            IMarker marker = file.createMarker(IMarker.PROBLEM);
            marker.setAttribute(IMarker.MESSAGE, message);
            marker.setAttribute(IMarker.SEVERITY, severity);
            if (lineNumber == -1) { lineNumber = 1; }
            marker.setAttribute(IMarker.LINE_NUMBER, lineNumber);
        } catch (CoreException e) {
}}
```

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`

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

[7] org.overturetool.eclipse.editor

```
    return (OvertureParserInterface) parser;
    }
}

public IConfigurationElement loadExtensionRegistry(String extensionName){
    IExtensionRegistry pluginRegistry = Platform.getExtensionRegistry();
    IExtensionPoint point = pluginRegistry.getExtensionPoint(OvertureConstants.PLUGIN_ID, extensionName);
    IExtension[] extensions = point.getExtensions();
    IExtension currentExtension = extensions[0];
    IConfigurationElement[] configElements = currentExtension.getConfigurationElements();
    return configElements[0];
}
```

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

```
-- case13          : bracket type
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected

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

```
-- case219 um          : chapter9/Full/Alphabet
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class Alphabet

instance variables
  alph : seq of char := [];

inv AlphabetInv(alph)

```

```
functions
  AlphabetInv: seq of char -> bool
  AlphabetInv (palph) ==
    len palph mod 2 = 0 and
    card elems palph = len palph

operations
  public Alphabet: seq of char ==> Alphabet
  Alphabet (pa) == alph := pa
  pre AlphabetInv(pa);

  public GetChar: nat ==> char
  GetChar (pidx) == return alph(pidx)
  pre pidx in set inds alph;

  public GetIndex: char ==> nat
  GetIndex (pch) ==
    let pidx in set {i | i in set inds alph
                         & alph(i) = pch} in
      return pidx
  pre pch in set elems alph;

  public GetIndices: () ==> set of nat
  GetIndices () == return inds alph;

  public GetSize: () ==> nat
  GetSize () == return len alph;

  public Shift: nat * nat ==> nat
  Shift (pidx, poffset) ==
    if pidx + poffset > len alph
    then return pidx + poffset - len alph
    else return pidx + poffset
  pre pidx in set inds alph and
      poffset <= len alph;

  public Shift: nat ==> nat
  Shift (pidx) == Shift(pidx, 1)
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 way as
12 --               expected. However it finds the line where the error occurs.
13 --               The reason is of cause that it is unaware of that the user
14 --               wants to specify.
15
16 class TestCase
17   is subclass of Test
18
19 instance variables
20   name : seq of char
21
22 operations
23   public TestCase: seq of char ==> TestCase
24   TestCase(nm) == name := nm;
25
26   public GetName: () ==> seq of char
27   GetName () == return name;
28
29   protected AssertTrue: bool ==> ()
30     --AssertTrue (pb) == if not pb then exit <FAILURE>;
31     AssertTrue (pb) == if not pb hen exit <FAILURE>;
32
33   protected AssertFalse: bool ==> ()
34   AssertFalse (pb) == if pb then exit <FAILURE>;
35
36   public Run: TestResult ==> ()
37   Run (ptr) ==
38     trap <FAILURE>
```

```
39         with
40           --ptr.AddFailure(self)
41           ptr.AddFailure(selv 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 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 groupding.

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

```
-- case400          : precedence
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected
class precedence
values
var1 = true or false and (true);
var2 = true and false or (true);
end precedence

-- case401          : grouping
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected
class grouping
types
var3 = int | bool | nat | char;
var4 = int -> bool -> nat -> char ;
end grouping
```

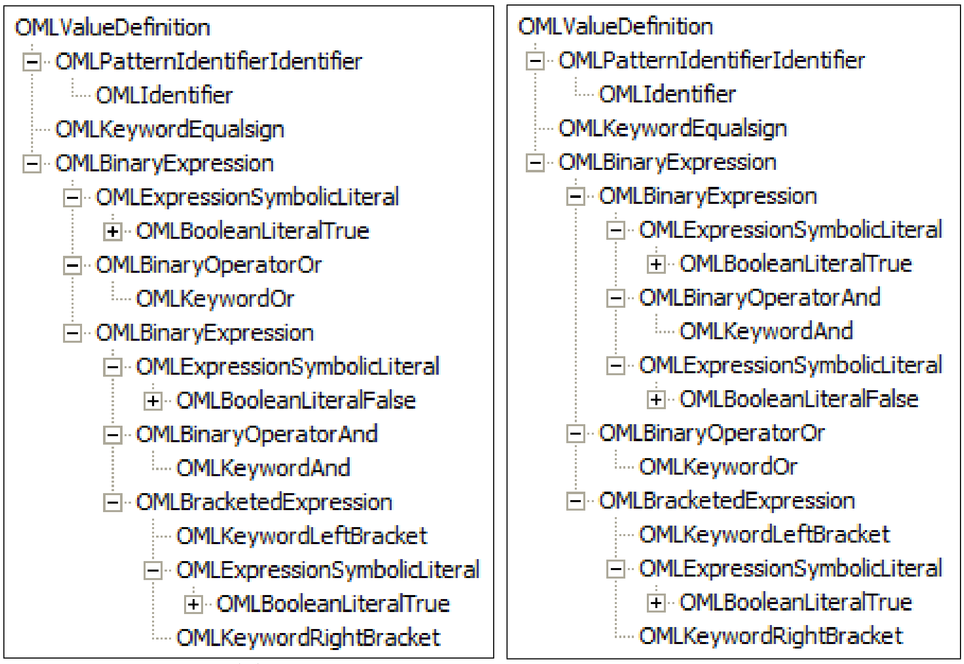
Precedence is tested by constructing an expression containing different combinations of the two binary operators `and` and `or` and verifying that the AST is correctly build. The two type constructs `union type` and `partial function type` has left and right grouping respectively and they have therefore 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.




(a) (b)

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

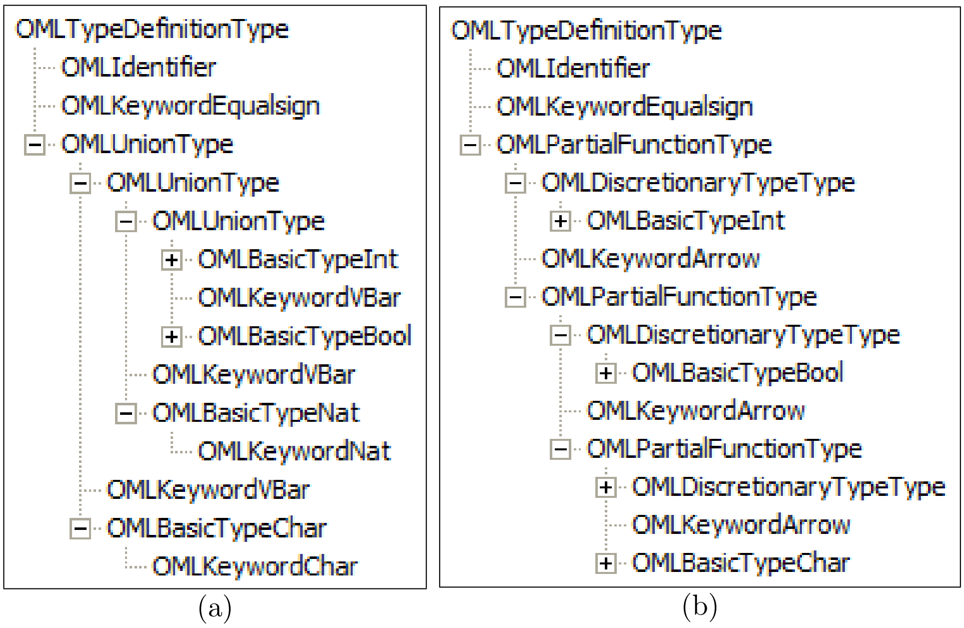



Figure 9.2: Outline view showing the structure of selected parts of an AST.

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 'prof 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 'prof 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 'prof 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 cause 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

```
class Info

types
private age = int

end Info
```

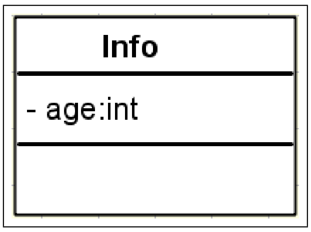



Figure 10.1: Example of an UML class diagram

### 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.

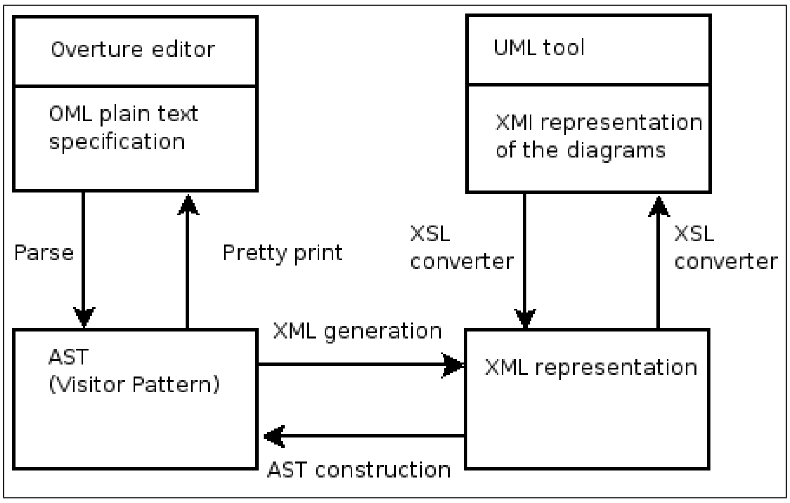



Figure 10.2: Overview of how to convert between OML and UML.

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

```
Document = class

class = 'class', identifier,

        [class body]

        'end', identifier;

```

```
class body = definition block

definition block = type definitions

type definitions = 'types', access type definition;

access type definition = [access], type definition;

access = 'private',

       |  'public',

       |  'protected',

type definition = identifier = type;

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 necessairy 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. automaticly 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 VDMtools. 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 intensions of the project. A discussion of the results is given, and the projets 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.

[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 Aliance.* 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 an the corresponding Java representation.

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

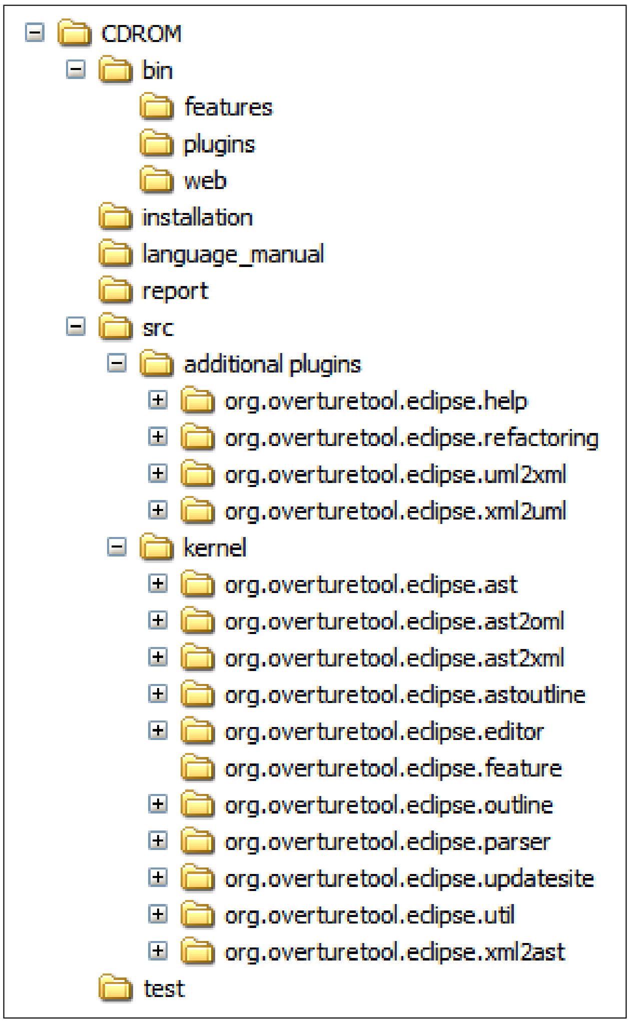



Figure B.1: Cd-rom contents

**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 (se 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' bottom. 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 places 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 possible to run the Overture kernel in the developer mode and here chose 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 chose 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 chose '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 Eclips

**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 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. FME[1] 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.

[1] Formal 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

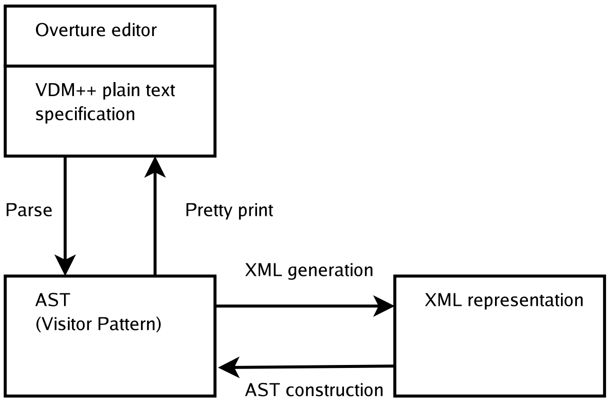



**Fig. 1.** Insert caption

## 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 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 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

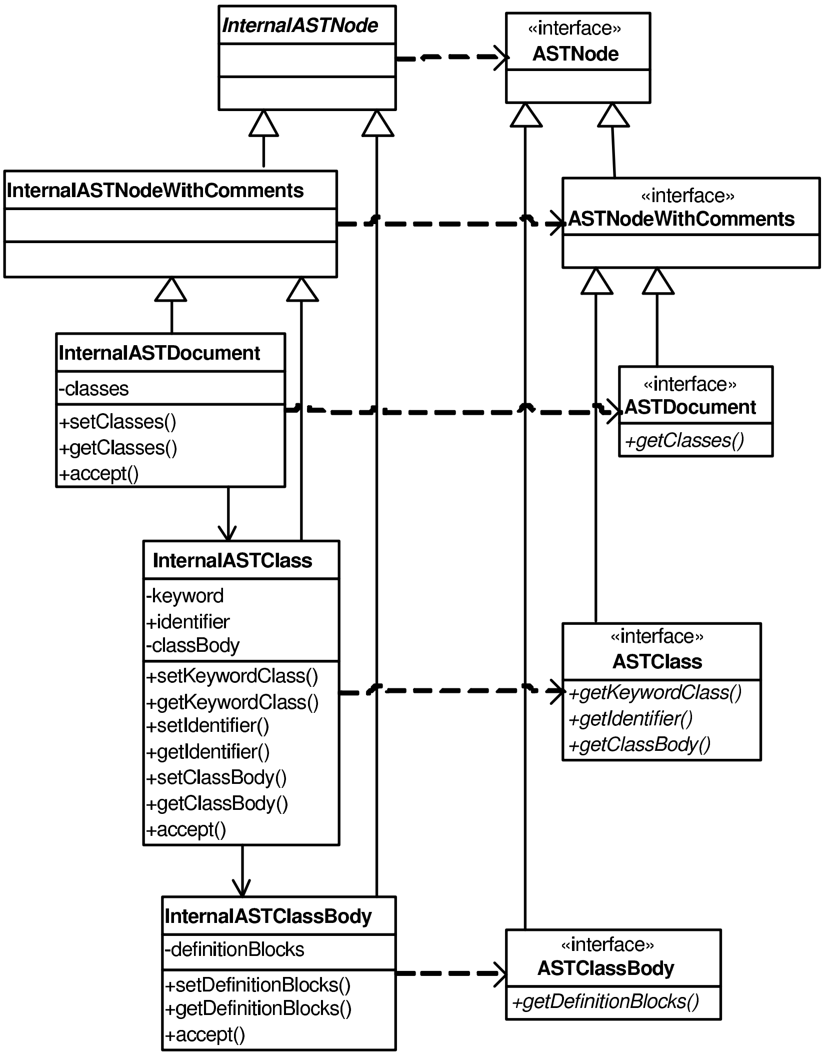



**Fig. 2.** Design of the AST classes

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 | |

| **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 |

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

[2]ANTLR version 2.7.5 [18]

[3]JavaCC 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 |

| **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 | | |

| **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 seams 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 seams 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. |

| | |
|---|---|
| **Recommended Solution** | ANTLR used for parser generation. Parser constructing AST based on manually written AST classes. |
| **Reason for Recommendation** | The selected criterions in mind, ANTLR fulfills all criterions. JavaCC supports most of the criterions, but has weaker error handling than ANTLR. ANTLR rises no problems, taking the additional criterions in mind. JavaCC might give problems as the documentation seams 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 criterions, have 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 criterions where JavaCC has advantages over ANTLR. The available documentation seams 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 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

```
header {
package org.overturetool.eclipse.parser;

import org.overturetool.eclipse.internal.ast.*;
import java.util.List;
import java.util.ArrayList;
}


class OvertureInternalParser extends Parser;
options {
 k = 2;                             // one token lookahead
 exportVocab=Overture;              // Call its vocabulary "Overture"
 codeGenMakeSwitchThreshold = 2;    // Some optimizations
 codeGenBitsetTestThreshold = 3;
 defaultErrorHandler = true;        // Generate parser error handlers
 buildAST = false;
}

// KEYWORDS
tokens {
 A    = "a_"   ;
 ABS    = "abs"   ;
 ALL    = "all"   ;
 ALLSUPER = "allsuper";
 ALWAYS  = "always" ;
 AND    = "and"   ;
 ANSWER  = "answer" ;
 ASSUMPTION  = "assumption";
 ATOMIC  = "atomic" ;
 BE    = "be"   ;
 BOOL   = "bool" ;
 BY    = "by"   ;
 CARD   = "card" ;
 CASES  = "cases" ;
 CHAR   = "char" ;
 CLASS  = "class" ;
 COMP   = "comp" ;
 COMPOSE  = "compose" ;
 CONC   = "conc" ;
 DCL    = "dcl"   ;
 DEF    = "def"   ;
 DEL    = "del"   ;
 DINTER  = "dinter" ;
 DIV    = "div"   ;
 DO    = "do"   ;
 DOM    = "dom"   ;
 DUNION  = "dunion" ;
 EFFECT  = "effect" ;
 ELEMS  = "elems" ;
 ELSE   = "else" ;
 ELSEIF  = "elseif"   ;
 END    = "end"   ;
 ERRROR  = "error" ;
 ERRS   = "errs" ;
 EXISTS  = "exists" ;
 EXISTS1  = "exists1" ;
 EXIT   = "exit" ;
 EXT    = "ext"   ;
 FALSE  = "false" ;
 FLOOR  = "floor" ;
 FOR    = "for"   ;
 FORALL  = "forall"   ;
 FROM   = "from" ;
 FUNCTIONS  = "functions";
 GENERAL  = "general" ;
 HD    = "hd"    ;
 IF     = "if"   ;
 IN    = "in"   ;
 INDS   = "inds" ;
 INIT   = "init" ;
 INMAP  = "inmap" ;
 INPUT  = "input" ;
 INSTANCE = "instance";
 INT    = "int"   ;
 INTER  = "inter" ;
```

```
INV    = "inv"   ;
INVERSE   = "inverse" ;
IOTA   = "iota" ;
IS    = "is"   ;
ISU    = "is_"   ;
ISUBOOL   = "is_bool"   ;
ISUCHAR   = "is_char"   ;
ISUINT   = "is_int"   ;
ISUNAT   = "is_nat"   ;
ISUNAT1   = "is_nat1"   ;
ISURAT   = "is_rat"   ;
ISUREAL   = "is_real"   ;
ISUTOKEN = "is_token" ;
ISOFBASECLASS="isofbaseclass";
ISOFCLASS = "isofclass";
LAMBDA   = "lambda" ;
LEN    = "len"   ;
LET    = "let"   ;
MAKE   = "mk_"   ;
MAP    = "map"   ;
MERGE   = "merge" ;
MOD    = "mod"   ;
MU    = "mu"   ;
MUNION   = "munion" ;
MUTEX   = "mutex" ;
NAT    = "nat"   ;
NAT1   = "nat1" ;
NEW    = "new"   ;
NIL    = "nil"   ;
NOT    = "not"   ;
OF    = "of"   ;
OPERATIONS = "operations";
OR    = "or"   ;
OTHERS   = "others" ;
PER    = "per"   ;
PERIODIC = "periodic";
POST   = "post" ;
POWER   = "power" ;
PRE    = "pre"   ;
PREF   = "pref" ;
PRIVATE   = "private" ;
PROTECTED = "protected"  ;
PSUBSET   = "psubset" ;
PUBLIC   = "public" ;
QSYNC   = "qsync" ;
RAT    = "rat"   ;
RD    = "rd"   ;
REAL   = "real" ;
REM    = "rem"   ;
RESPONSIBILITY="responsibility";
RETURN   = "return" ;
REVERSE   = "reverse" ;
RNG    = "rng"   ;
SAMEBASECLASS="samebaseclass";
SAMECLASS = "sameclass";
SEL    = "sel"   ;
SELF   = "self" ;
SEQ    = "seq"   ;
SEQ1   = "seq1" ;
SET    = "set"   ;
SKIP   = "skip" ;
SPECIFIED = "specified";
ST    = "st"   ;
START   = "start" ;
STARTLIST = "startlist";
SUBCLASS = "subclass";
SUBSET   = "subset" ;
SUBTRACE = "subtrace";
SYNC   = "sync" ;
SYNONYM   = "synonym" ;
T    = "t_"   ;
THEN   = "then" ;
THREAD   = "thread" ;
THREADID = "threadid";
TIXE   = "tixe" ;
TL    = "tl"   ;
TO    = "to"   ;
TOKEN   = "token" ;
TRAP   = "trap" ;
TRUE   = "true" ;
TYPES   = "types" ;
UNDEFINED = "undefined";
UNION   = "union" ;
USING   = "using" ;
```

```
 VALUES   = "values" ;
 VARIABLES = "variables";
 W   = "w_"   ;
 WHILE  = "while" ;
 WITH  = "with" ;
 WR   = "wr"  ;
 YET   = "yet"  ;
 RESULT  = "RESULT" ;
}


// overtureDocument: This is the ASTERIX
//    rule for this parser
// document = class , { class };
overtureDocument returns [InternalASTDocument internalASTDocument]
{
 internalASTDocument = null;
 List<InternalASTClass> internalASTClassList = new ArrayList<InternalASTClass>();
 InternalASTClass internalASTClass = null;
}
 :

  internalASTClass = overtureClass
   {
       internalASTClassList.add(internalASTClass);
   }

  (
       internalASTClass = overtureClass
        {
            internalASTClassList.add(internalASTClass);
        }
  )*
  EOF!
   {
    internalASTDocument = new InternalASTDocument(internalASTClassList);
   }
 ;

// class = 'class', identifier, [inheritance clause],
//          [class body],
//          'end', identifier;
overtureClass returns [InternalASTClass internalASTClass]
{
 internalASTClass = null;
 InternalASTClassBody internalASTClassBody = null;
 InternalASTInheritanceClause internalASTInheritanceClause = null;
}
 : cl:CLASS id:IDENTIFIER
  (
       internalASTInheritanceClause = overtureInheritanceClause
  )?
  (
       internalASTClassBody = overtureClassBody
  )?

  en:END id2:IDENTIFIER
   {
    InternalASTKeyword keywordClass = new InternalASTKeyword(cl,"Class");
    InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
      InternalASTKeyword keywordEnd = new InternalASTKeyword(en,"End");
     InternalASTIdentifier internalASTIdentifierEnd = new InternalASTIdentifier(id2);
    internalASTClass = new InternalASTClass(keywordClass, internalASTIdentifier,
         internalASTInheritanceClause, internalASTClassBody, keywordEnd, internalASTIdentifierEnd
         );
   }
 ;

// inheritance clause = 'is subclass of', identifier, {identifier};
overtureInheritanceClause returns [InternalASTInheritanceClause internalASTInheritanceClause]
{
 InternalASTKeyword keywordIs = null;
 InternalASTKeyword keywordSubclass = null;
 InternalASTKeyword keywordOf = null;
 InternalASTIdentifier internalASTIdentifier1 = null;
 internalASTInheritanceClause = null;
 List<InternalASTIdentifier> inheritanceIdentifierList = new ArrayList<InternalASTIdentifier>();
}
 :

  is:IS
  su:SUBCLASS
  of:OF
  id1:IDENTIFIER
```

```
  {
   keywordIs = new InternalASTKeyword(is,"Is");
   keywordSubclass = new InternalASTKeyword(su,"Subclass");
   keywordOf = new InternalASTKeyword(of,"Of");
   internalASTIdentifier1 = new InternalASTIdentifier(id1);
   inheritanceIdentifierList.add(internalASTIdentifier1);
  }
  (
   id:IDENTIFIER
    {
       InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
     inheritanceIdentifierList.add(internalASTIdentifier);
    }
  )*
  {
   internalASTInheritanceClause = new InternalASTInheritanceClause(keywordIs, keywordSubclass,
       keywordOf, inheritanceIdentifierList);
  }
 ;


// class body = definition block, {definition block} ;
overtureClassBody returns [InternalASTClassBody internalASTClassBody]
{
 internalASTClassBody = null;
 List<InternalASTDefinitionBlock> internalASTDefinitionBlockList = new ArrayList<
      InternalASTDefinitionBlock>();
 InternalASTDefinitionBlock internalASTDefinitionBlock = null;
}
 :
  internalASTDefinitionBlock = overtureDefinitionBlock
   {
      internalASTDefinitionBlockList.add(internalASTDefinitionBlock);
   }

  (
   internalASTDefinitionBlock = overtureDefinitionBlock
    {
      internalASTDefinitionBlockList.add(internalASTDefinitionBlock);
    }
  )*
  {
   internalASTClassBody = new InternalASTClassBody(internalASTDefinitionBlockList);
  }
 ;

// definition block = type definitions
//      | value definitions
//      | function definitions
//      | operation definitions;
//      | instance variable definitions;
//      | synchronization definitions;
//      | thread definitions;
overtureDefinitionBlock returns [InternalASTDefinitionBlock internalASTDefinitionBlock]
{
 //ASTDefinitionBlock is abstract...
 internalASTDefinitionBlock = null;
}
 :
  (
   internalASTDefinitionBlock = overtureTypeDefinitions
  | internalASTDefinitionBlock = overtureValueDefinitions
  | internalASTDefinitionBlock = overtureFunctionDefinitions
  | internalASTDefinitionBlock = overtureOperationDefinitions
  | internalASTDefinitionBlock = overtureInstanceVariableDefinitions
  | internalASTDefinitionBlock = overtureSynchronizationDefinitions
  | internalASTDefinitionBlock = overtureThreadDefinitions
  )
 ;


// type definitions = 'types',[access type definition,
//                      {';', access type definition},[';']];
overtureTypeDefinitions returns [InternalASTTypeDefinitions internalASTTypeDefinitions]
{
 internalASTTypeDefinitions = null;
 InternalASTKeyword keywordTypes = null;
 List<InternalASTTypeDefinitionsElement> typeDefinitionsElements = new ArrayList<
      InternalASTTypeDefinitionsElement>();
 InternalASTTypeDefinitionsElement currentElement = null;
 InternalASTAccessTypeDefinition internalASTAccessTypeDefinition = null;
}
 :
  ty:TYPES
```

```
    {
     keywordTypes = new InternalASTKeyword(ty,"Types");
    }

   (
    internalASTAccessTypeDefinition = overtureAccessTypeDefinition
     {
      currentElement = new InternalASTTypeDefinitionsElement(internalASTAccessTypeDefinition);
      typeDefinitionsElements.add(currentElement);
     }

    (options {greedy=true;}:
     seNoN:SEMICOLON
     {
      InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonNoN);
      currentElement.setEndPositionFromNode(semicolonNoN);
     }
     internalASTAccessTypeDefinition = overtureAccessTypeDefinition
     {
      currentElement = new InternalASTTypeDefinitionsElement(internalASTAccessTypeDefinition);
         typeDefinitionsElements.add(currentElement);
     }
    )*

    (seLast:SEMICOLON
              {
      InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonLast);
      currentElement.setEndPositionFromNode(semicolonLast);
               }
    )?

   )?
   {
    internalASTTypeDefinitions = new InternalASTTypeDefinitions(keywordTypes,
         typeDefinitionsElements);
   }
  ;

// access type definition = [access], type definition;
overtureAccessTypeDefinition returns [InternalASTAccessTypeDefinition
      internalASTAccessTypeDefinition]
{
 internalASTAccessTypeDefinition = null;
 InternalASTAccess internalASTAccess = null;
 InternalASTTypeDefinition internalASTTypeDefinition = null;
}
  :
   (
    internalASTAccess = overtureAccess
   )?
   internalASTTypeDefinition = overtureTypeDefinition
   {
      internalASTAccessTypeDefinition = new InternalASTAccessTypeDefinition(internalASTAccess,
           internalASTTypeDefinition);
   }
  ;

// access = 'public',
//         | 'private',
//         | 'protected';
overtureAccess returns [InternalASTAccess internalASTAccess]
{
 //ASTAccess is abstract...
 internalASTAccess = null;
}
  :
   ( pub:PUBLIC
    {
     internalASTAccess = new InternalASTAccessPublic(new InternalASTKeyword(pub,"Public"));
    }
   | pri:PRIVATE
    {
     internalASTAccess = new InternalASTAccessPrivate(new InternalASTKeyword(pri,"Private"));
    }
   | pro:PROTECTED
    {
     internalASTAccess = new InternalASTAccessProtected(new InternalASTKeyword(pro,"Protected"));
    }
   )
  ;

// type definition = identifier, '=', type, [invariant]
```

```
//        | identifier, '::', field list, [invariant];
overtureTypeDefinition returns [InternalASTTypeDefinition internalASTTypeDefinition]
{
 internalASTTypeDefinition = null;
}
  :
   (
    internalASTTypeDefinition = overtureTypeDefinitionType
   |
    internalASTTypeDefinition = overtureTypeDefinitionFieldList
   )
  ;

// Matches identifier, '=', type, [invariant]
overtureTypeDefinitionType returns [InternalASTTypeDefinitionType internalASTTypeDefinitionType]
{
 internalASTTypeDefinitionType = null;
 InternalASTType internalASTType = null;
 InternalASTInvariant internalASTInvariant = null;
}
  :
   id:IDENTIFIER
   eq:EQUALSIGN
   internalASTType = overtureType
   (
    internalASTInvariant = overtureInvariant
   )?
   {
       InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
    InternalASTKeyword keywordEqualsign = new InternalASTKeyword(eq,"Equalsign");
    internalASTTypeDefinitionType = new InternalASTTypeDefinitionType(internalASTIdentifier,
        keywordEqualsign, internalASTType, internalASTInvariant);
   }
  ;

// Matches identifier, '::', field list, [invariant]
overtureTypeDefinitionFieldList returns [InternalASTTypeDefinitionFieldList
    internalASTTypeDefinitionFieldList]
{
 internalASTTypeDefinitionFieldList = null;
 InternalASTFieldList internalASTFieldList = null;
 InternalASTInvariant internalASTInvariant = null;
}
  :
   id:IDENTIFIER
   dc:DOUBLECOLON
   internalASTFieldList = overtureFieldList
   (
    internalASTInvariant = overtureInvariant
   )?
   {
       InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
    InternalASTKeyword keywordDoubleColon = new InternalASTKeyword(dc,"DoubleColon");
    internalASTTypeDefinitionFieldList = new InternalASTTypeDefinitionFieldList(
        internalASTIdentifier, keywordDoubleColon, internalASTFieldList, internalASTInvariant);
   }
  ;

// type = bracketed type
//        | basic type
//        | quote type
//        | composite type
//        | union type
//        | product type
//        | optional type
//        | set type
//        | seq type
//        | map type
//        | partial function type
//        | type name
//        | type variable;
overtureType returns [InternalASTType internalASTType]
{
 //ASTAccess is abstract...
 internalASTType = null;
}
  :
     internalASTType = overtureTypePre1
  ;

overtureTypePre6 returns [InternalASTType internalASTType]
{
 internalASTType = null;
}
```

```
  :
        (
   //Type type expressions - base cases
     internalASTType = overtureBracketedType
   | internalASTType = overtureBasicType
   | internalASTType = overtureQuoteType
      | internalASTType = overtureCompositeType
        | internalASTType = overtureOptionalType
        | internalASTType = overtureTypeName
      | internalASTType = overtureTypeVariable
  )
  ;

overtureTypePre5 returns [InternalASTType internalASTType]
{
 //Type operators with precedence 5 (highest precedence)
 internalASTType = null;
}
  :
        internalASTType = overtureSetType
       |  internalASTType = overtureSeqType

   | internalASTType = overtureTypePre6
  ;

overtureTypePre4 returns [InternalASTType internalASTType]
{
 //Type operators with precedence 4 (high, but not highest precedence)
 internalASTType = null;
}
  :
       internalASTType = overtureMapType

   | internalASTType = overtureTypePre5
  ;

overtureTypePre3 returns [InternalASTType internalASTType]
{
 //Type operators with precedence 3 (middle precedence)
 internalASTType = null;
}
  :
       (overtureProductType)=> internalASTType = overtureProductType

   | internalASTType = overtureTypePre4
  ;

overtureTypePre2 returns [InternalASTType internalASTType]
{
 //Type operators with precedence 2 (low, but not lowest precedence)
 internalASTType = null;
}
  :
       (overtureUnionType)=> internalASTType = overtureUnionType

   | internalASTType = overtureTypePre3
  ;

overtureTypePre1 returns [InternalASTType internalASTType]
{
 //Type operators with precedence 1 (lowest precedence)
 internalASTType = null;
}
  :
       (overturePartialFunctionType)=> internalASTType = overturePartialFunctionType

   | internalASTType = overtureTypePre2
  ;

// bracketed type = '(', type, ')';
overtureBracketedType returns [InternalASTBracketedType internalASTBracketedType]
{
 internalASTBracketedType = null;
 InternalASTType internalASTType = null;
}
  :
   lb:LBRACKET
   internalASTType = overtureType
   rb:RBRACKET
     {
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
      internalASTBracketedType = new InternalASTBracketedType(keywordLeftBracket, internalASTType,
          keywordRightBracket);
```

```
    }
  ;

// basic type = 'bool' | 'int';
overtureBasicType returns [InternalASTBasicType internalASTBasicType]
{
 //ASTAccess is abstract...
 internalASTBasicType = null;
}
  :
   ( bo:BOOL
     {
      internalASTBasicType = new InternalASTBasicTypeBool(new InternalASTKeyword(bo,"Bool"));
     }
   | na:NAT
     {
      internalASTBasicType = new InternalASTBasicTypeNat(new InternalASTKeyword(na,"Nat"));
     }
   | na1:NAT1
     {
      internalASTBasicType = new InternalASTBasicTypeNat1(new InternalASTKeyword(na1,"Nat1"));
     }
   | in:INT
     {
      internalASTBasicType = new InternalASTBasicTypeInt(new InternalASTKeyword(in,"Int"));
     }
   | ra:RAT
     {
      internalASTBasicType = new InternalASTBasicTypeRat(new InternalASTKeyword(ra,"Rat"));
     }
   | re:REAL
     {
      internalASTBasicType = new InternalASTBasicTypeReal(new InternalASTKeyword(re,"Real"));
     }
   | ch:CHAR
     {
      internalASTBasicType = new InternalASTBasicTypeChar(new InternalASTKeyword(ch,"Char"));
     }
   | to:TOKEN
     {
      internalASTBasicType = new InternalASTBasicTypeToken(new InternalASTKeyword(to,"Token"));
     }
   )
  ;

// quote type =
overtureQuoteType returns [InternalASTQuoteType internalASTQuoteType]
{
 internalASTQuoteType = null;
 InternalASTQuoteLiteral qliteral = null;
}
  :
   qliteral = overtureQuoteLiteral
     {
      internalASTQuoteType = new InternalASTQuoteType(qliteral);
     }

  ;

// composite type =
overtureCompositeType returns [InternalASTCompositeType internalASTCompositeType]
{
 internalASTCompositeType = null;
 InternalASTFieldList fieldlist = null;
}
  :
   cp:COMPOSE
   id:IDENTIFIER
   of:OF
   fieldlist = overtureFieldList
   en:END
     {
      InternalASTKeyword keywordCompose = new InternalASTKeyword(cp,"Compose");
      InternalASTIdentifier identifier = new InternalASTIdentifier(id);
      InternalASTKeyword keywordOf = new InternalASTKeyword(of,"Of");
      InternalASTKeyword keywordEnd = new InternalASTKeyword(en,"End");
      internalASTCompositeType = new InternalASTCompositeType(keywordCompose, identifier, keywordOf
           , fieldlist, keywordEnd);
     }
  ;

// field list = { field }
overtureFieldList returns [InternalASTFieldList internalASTFieldList]
{
```

```
 internalASTFieldList = null;
 List<InternalASTField> fields = new ArrayList<InternalASTField>();
 InternalASTField internalASTField = null;
}
 :
  (
   internalASTField = overtureField
    {
     fields.add(internalASTField);
    }
  )*
  {
   internalASTFieldList = new InternalASTFieldList(fields);
  }
 ;

// field = [ idenifier , ':'], type
overtureField returns [InternalASTField internalASTField]
{
 internalASTField = null;
 InternalASTType type = null;
}
 :
  (
   (
    (
     id1:IDENTIFIER
     co:COLON
     type = overtureType
      {
       InternalASTIdentifier identifier = new InternalASTIdentifier(id1);
       InternalASTKeyword operator = new InternalASTKeyword(co,"Colon");
       internalASTField = new InternalASTFieldColon(identifier, operator, type);
      }
    )
    |
    (
     id2:IDENTIFIER
     coli:COLONLINE
     type = overtureType
      {
       InternalASTIdentifier identifier = new InternalASTIdentifier(id2);
       InternalASTKeyword operator = new InternalASTKeyword(coli,"ColonLine");
       internalASTField = new InternalASTFieldColonLine(identifier, operator, type);
      }
    )
    |
    (
     type = overtureType
      {
       internalASTField = new InternalASTFieldBasic(type);
      }
    )
   )
  )
 ;

// union type =
overtureUnionType returns [InternalASTUnionType internalASTUnionType]
{
 internalASTUnionType = null;
 InternalASTType type1 = null;
 InternalASTKeyword keywordVBar = null;
 InternalASTType type2 = null;
}
 :
  (
   type1 = overtureTypePre3
  )
  vb:VBAR
   {
    keywordVBar = new InternalASTKeyword(vb,"VBar");
   }
  (
   type2 = overtureTypePre3
    {
     internalASTUnionType = new InternalASTUnionType(type1,keywordVBar,type2);
    }
  )


  (options {greedy=true;}:
   vbn:VBAR
    {
```

```
      keywordVBar = new InternalASTKeyword(vbn,"VBar");
     }
    (
     type2 = overtureTypePre3
      {
       internalASTUnionType = new InternalASTUnionType(internalASTUnionType,keywordVBar,type2);
      }
    )
   )*
  ;

// product type =
overtureProductType returns [InternalASTProductType internalASTProductType]
{
 internalASTProductType = null;
 InternalASTType type1 = null;
 InternalASTKeyword keywordAsterix = null;
 InternalASTType type2 = null;
}
  :
   (
    type1 = overtureTypePre4
   )
   as:ASTERIX
    {
     keywordAsterix = new InternalASTKeyword(as,"Asterix");
    }
   (
    type2 = overtureTypePre3
     {
      internalASTProductType = new InternalASTProductType(type1, keywordAsterix, type2);
     }
   )
  ;

// optional type =
overtureOptionalType returns [InternalASTOptionalType internalASTOptionalType]
{
 internalASTOptionalType = null;
 InternalASTType type = null;
}
  :
   lbc:LBRACK
   type = overtureType
   rbc:RBRACK
    {
     InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lbc,"LeftBrack");
     InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rbc,"RightBrack");
     internalASTOptionalType = new InternalASTOptionalType(keywordLeftBrack, type,
          keywordRightBrack);
    }
  ;

// set type =
overtureSetType returns [InternalASTSetType internalASTSetType]
{
 internalASTSetType = null;
 InternalASTType type = null;
}
  :
   se:SET
   of:OF
   type = overtureTypePre5
    {
     InternalASTKeyword keywordSet = new InternalASTKeyword(se, "Set");
     InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
     internalASTSetType = new InternalASTSetType(keywordSet, keywordOf, type);
    }
  ;

// seq type =
overtureSeqType returns [InternalASTSeqType internalASTSeqType]
{
 internalASTSeqType = null;
}
  :
   internalASTSeqType = overtureSeq0Type
   |
   internalASTSeqType = overtureSeq1Type
  ;

// sec0 type = 'seq of', type;
overtureSeq0Type returns [InternalASTSeq0Type internalASTSeq0Type]
{
```

```
 internalASTSeq0Type = null;
 InternalASTType type;
}
 :
  se0:SEQ
  of:OF
  type = overtureTypePre5
   {
    InternalASTKeyword keywordSeq = new InternalASTKeyword(se0, "Seq");
    InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
    internalASTSeq0Type = new InternalASTSeq0Type(keywordSeq, keywordOf, type);
   }
 ;

// sec1 type = 'seq1 of', type;
overtureSeq1Type returns [InternalASTSeq1Type internalASTSeq1Type]
{
 internalASTSeq1Type = null;
 InternalASTType type;
}
 :
  se1:SEQ1
  of:OF
  type = overtureTypePre5
   {
    InternalASTKeyword keywordSeq1 = new InternalASTKeyword(se1, "Seq1");
    InternalASTKeyword keywordOf = new InternalASTKeyword(of, "Of");
    internalASTSeq1Type = new InternalASTSeq1Type(keywordSeq1, keywordOf, type);
   }
 ;

// map type =
overtureMapType returns [InternalASTMapType internalASTMapType]
{
 internalASTMapType = null;
}
 :
  internalASTMapType = overtureGeneralMapType
  |
  internalASTMapType = overtureInjectiveMapType
 ;

// general map type = 'map', type, 'to', type;
overtureGeneralMapType returns [InternalASTGeneralMapType internalASTGeneralMapType]
{
 internalASTGeneralMapType = null;
 InternalASTType type1;
 InternalASTType type2;

}
 :
  ma:MAP
  type1 = overtureTypePre4
  to:TO
  type2 = overtureTypePre4
   {
    InternalASTKeyword keywordMap = new InternalASTKeyword(ma,"Map");
    InternalASTKeyword keywordTo = new InternalASTKeyword(to,"To");
    internalASTGeneralMapType = new InternalASTGeneralMapType(keywordMap, type1, keywordTo, type2
        );
   }
 ;

// injective map type = 'inmap', type, 'to', type;
overtureInjectiveMapType returns [InternalASTInjectiveMapType internalASTInjectiveMapType]
{
 internalASTInjectiveMapType = null;
 InternalASTType type1;
 InternalASTType type2;
}
 :
  im:INMAP
  type1 = overtureTypePre4
  to:TO
  type2 = overtureTypePre4
   {
    InternalASTKeyword keywordInmap = new InternalASTKeyword(im,"Inmap");
    InternalASTKeyword keywordTo = new InternalASTKeyword(to,"To");
    internalASTInjectiveMapType = new InternalASTInjectiveMapType(keywordInmap, type1, keywordTo,
        type2);
   }
 ;

// function type = partial function type
```

```
//      | total function type;
overtureFunctionType returns [InternalASTFunctionType internalASTFunctionType]
{
 internalASTFunctionType = null;
}
 :
      (
    (overturePartialFunctionType)=> internalASTFunctionType = overturePartialFunctionType
    | internalASTFunctionType = overtureTotalFunctionType
  )
 ;

// partial function type = discretionary type '->' type
overturePartialFunctionType returns [InternalASTPartialFunctionType
     internalASTPartialFunctionType]
{
 internalASTPartialFunctionType = null;
 InternalASTDiscretionaryType disctype = null;
 InternalASTType type = null;
}
 :
  disctype = overtureDiscretionaryType
  arw : LINEARROW
  type = overtureType
   {
    InternalASTKeyword keywordLineArrow = new InternalASTKeyword(arw, "Arrow");
    internalASTPartialFunctionType = new InternalASTPartialFunctionType(disctype,
         keywordLineArrow, type);
   }
 ;

// Total total type = discretionary type '+>' type
overtureTotalFunctionType returns [InternalASTTotalFunctionType internalASTTotalFunctionType]
{
 internalASTTotalFunctionType = null;
 InternalASTDiscretionaryType disctype = null;
 InternalASTType type = null;
}
 :
  disctype = overtureDiscretionaryType
  kwpa : PLUSARROW
  type = overtureType
   {
    InternalASTKeyword keywordPlusArrow = new InternalASTKeyword(kwpa, "PlusArrow");
    internalASTTotalFunctionType = new InternalASTTotalFunctionType(disctype, keywordPlusArrow,
         type);
   }
 ;

// discretionary type = type | '(',')' ;
overtureDiscretionaryType returns [InternalASTDiscretionaryType internalASTDiscretionaryType]
{
 internalASTDiscretionaryType = null;
 InternalASTType type2 = null;
}
 :
   (
   type2 = overtureTypePre2
    {
     internalASTDiscretionaryType = new InternalASTDiscretionaryTypeType(type2);
    }
  )
  |
  (
   lb:LBRACKET
   rb:RBRACKET
    {
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
     internalASTDiscretionaryType = new InternalASTDiscretionaryTypeBrackets(keywordLeftBracket,
          keywordRightBracket);
    }
  )
 ;

// type name =
overtureTypeName returns [InternalASTTypeName internalASTTypeName]
{
 internalASTTypeName = null;
 InternalASTName internalASTName = null;
}
 :
  internalASTName = overtureName
   {
```

```
      internalASTTypeName = new InternalASTTypeName(internalASTName);
     }
    ;




// type variable =
overtureTypeVariable returns [InternalASTTypeVariable internalASTTypeVariable]
{
 internalASTTypeVariable = null;
 InternalASTTypeVariableIdentifier typeVariableIdentifier = null;
}
  :
   typeVariableIdentifier = overtureTypeVariableIdentifier
     {
      internalASTTypeVariable = new InternalASTTypeVariable(typeVariableIdentifier);
     }
  ;

// invariant = 'inv', invariant initial function;
overtureInvariant returns [InternalASTInvariant internalASTInvariant]
{
 internalASTInvariant = null;
 InternalASTKeyword keywordInv = null;
 InternalASTInvariantInitialFunction invariantInitialFunction = null;
}
  :
   in:INV
   invariantInitialFunction = overtureInvariantInitialFunction
     {
      keywordInv = new InternalASTKeyword(in,"Inv");
      internalASTInvariant = new InternalASTInvariant(keywordInv, invariantInitialFunction);
     }
  ;

// invariant initial function = pattern, '==', expression;
overtureInvariantInitialFunction returns [InternalASTInvariantInitialFunction
     internalASTInvariantInitialFunction]
{
 internalASTInvariantInitialFunction = null;
 InternalASTKeyword keywordDoubleEqual = null;
 InternalASTPattern pattern = null;
 InternalASTExpression expression = null;
}
  :
   pattern = overturePattern
   de:DOUBLEEQUAL
   expression = overtureExpression
   {
    keywordDoubleEqual = new InternalASTKeyword(de,"DoubleEqual");
    internalASTInvariantInitialFunction = new InternalASTInvariantInitialFunction(pattern,
         keywordDoubleEqual, expression);
   }
  ;

// VALUE DEFINITIONS

// value definitions = 'values', [access value definition,
//        {';', access value definition },
//        [';']];
overtureValueDefinitions returns [InternalASTValueDefinitions internalASTValueDefinitions]
{
 internalASTValueDefinitions = null;
 InternalASTKeyword keywordValues = null;
 List<InternalASTValueDefinitionsElement> valueDefinitionsElements = new ArrayList<
       InternalASTValueDefinitionsElement>();
 InternalASTValueDefinitionsElement currentElement = null;
 InternalASTAccessValueDefinition internalASTAccessValueDefinition = null;
}
  :
   va:VALUES
     {
      keywordValues = new InternalASTKeyword(va,"Values");
     }

   (
    internalASTAccessValueDefinition = overtureAccessValueDefinition
     {
      currentElement = new InternalASTValueDefinitionsElement(internalASTAccessValueDefinition);
      valueDefinitionsElements.add(currentElement);
     }

    (options {greedy=true;}:
```

```
      seNoN:SEMICOLON
      {
       InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
       currentElement.setKeywordSemicolon(semicolonNoN);
       currentElement.setEndPositionFromNode(semicolonNoN);
      }
      internalASTAccessValueDefinition = overtureAccessValueDefinition
      {
       currentElement = new InternalASTValueDefinitionsElement(internalASTAccessValueDefinition);
         valueDefinitionsElements.add(currentElement);
      }
     )*

     (seLast:SEMICOLON
                {
       InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
       currentElement.setKeywordSemicolon(semicolonLast);
       currentElement.setEndPositionFromNode(semicolonLast);
                 }
     )?

   )?

   {
    internalASTValueDefinitions = new InternalASTValueDefinitions(keywordValues,
         valueDefinitionsElements);
   }
  ;

// access value definition = [access] value definition;
overtureAccessValueDefinition returns [InternalASTAccessValueDefinition
     internalASTAccessValueDefinition]
{
 internalASTAccessValueDefinition = null;
 InternalASTAccess internalASTAccess = null;
 InternalASTValueDefinition internalASTValueDefinition = null;
}
  :
   (
    internalASTAccess = overtureAccess
   )?
   internalASTValueDefinition = overtureValueDefinition
     {
        internalASTAccessValueDefinition = new InternalASTAccessValueDefinition(internalASTAccess,
             internalASTValueDefinition);
     }
  ;

// value definition = pattern [':', type], '=', expression
overtureValueDefinition returns [InternalASTValueDefinition internalASTValueDefinition]
{
 internalASTValueDefinition = null;
 InternalASTPattern pattern = null;
 InternalASTKeyword keywordColon = null;
 InternalASTType internalASTType = null;
 InternalASTExpression expression = null;
}
  :
   pattern = overturePattern
   (
    co:COLON
    internalASTType = overtureType
        {
         keywordColon = new InternalASTKeyword(co,"Colon");
        }
   )?

   kweq:EQUALSIGN
   expression = overtureExpression
   {
    if(expression == null) {System.err.println("expression i valuedefinition er NULL"); }
    InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq,"Equalsign");
    internalASTValueDefinition = new InternalASTValueDefinition(pattern, keywordColon,
         internalASTType, keywordEqualsign, expression);
   }
  ;

// FUNCTION DEFINITIONS

// function definitions = 'function', [access function definition,
//        {';', access function definition },
//        [';']];
overtureFunctionDefinitions returns [InternalASTFunctionDefinitions
     internalASTFunctionDefinitions]
```

```
{
 internalASTFunctionDefinitions = null;
 InternalASTKeyword keywordFunctions = null;
 List<InternalASTFunctionDefinitionsElement> FunctionDefinitionsElements = new ArrayList<InternalASTFunctionDefinitionsElement>();
 InternalASTFunctionDefinitionsElement currentElement = null;
 InternalASTAccessFunctionDefinition internalASTAccessFunctionDefinition = null;
}
 :
  fu:FUNCTIONS
  {
   keywordFunctions = new InternalASTKeyword(fu,"Functions");
  }
  (
   internalASTAccessFunctionDefinition = overtureAccessFunctionDefinition
    {
     currentElement = new InternalASTFunctionDefinitionsElement(internalASTAccessFunctionDefinition);
     FunctionDefinitionsElements.add(currentElement);
    }

   (options {greedy=true;}:
    seNoN:SEMICOLON
    {
     InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
     currentElement.setKeywordSemicolon(semicolonNoN);
     currentElement.setEndPositionFromNode(semicolonNoN);
    }
    internalASTAccessFunctionDefinition = overtureAccessFunctionDefinition
    {
     currentElement = new InternalASTFunctionDefinitionsElement(internalASTAccessFunctionDefinition);
       FunctionDefinitionsElements.add(currentElement);
    }
   )*

   (seLast:SEMICOLON
            {
     InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
     currentElement.setKeywordSemicolon(semicolonLast);
     currentElement.setEndPositionFromNode(semicolonLast);
             }
   )?

  )?
  {
   internalASTFunctionDefinitions = new InternalASTFunctionDefinitions(keywordFunctions, FunctionDefinitionsElements);
  }
 ;


// access function definition = [access], function definition;
overtureAccessFunctionDefinition returns [InternalASTAccessFunctionDefinition internalASTAccessFunctionDefinition]
{
 internalASTAccessFunctionDefinition = null;
 InternalASTAccess internalASTAccess = null;
 InternalASTFunctionDefinition internalASTFunctionDefinition = null;
}
 :
  (
   internalASTAccess = overtureAccess
  )?
  internalASTFunctionDefinition = overtureFunctionDefinition
  {
     internalASTAccessFunctionDefinition = new InternalASTAccessFunctionDefinition(internalASTAccess, internalASTFunctionDefinition);
  }
 ;

// function definition = explicit function definition
//           | implicit function definition;
//        | extended explicit function definition
overtureFunctionDefinition returns [InternalASTFunctionDefinition internalASTFunctionDefinition]
{
 internalASTFunctionDefinition = null;
}
 :
      (
  (overtureExplicitFunctionDefinition)=> internalASTFunctionDefinition = overtureExplicitFunctionDefinition
  | (overtureImplicitFunctionDefinition)=> internalASTFunctionDefinition = overtureImplicitFunctionDefinition
```

```
   | internalASTFunctionDefinition = overtureExtendedExplicitFunctionDefinition
  )
 ;



//explicit function defnition = identifier, [ type variable list ], ':',
//          function type,
//          identifier, parameters list,
//          '==', function body,
//          [ 'pre', expression ] ,
//          [ 'post', expression ] ;

overtureExplicitFunctionDefinition returns [InternalASTExplicitFunctionDefinition
     internalASTExplicitFunctionDefinition]
{
 internalASTExplicitFunctionDefinition = null;
 InternalASTTypeVariableList internalASTTypeVariableList = null;
 InternalASTFunctionType internalASTFunctionType = null;
 InternalASTParametersList internalASTParametersList = null;
 InternalASTFunctionBody internalASTFunctionBody = null;
 InternalASTKeyword keywordPre = null;
 InternalASTExpression internalASTExpressionPre = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression internalASTExpressionPost = null;
}
  : id1:IDENTIFIER
  (
      internalASTTypeVariableList = overtureTypeVariableList
  )?
   kwc:COLON
     internalASTFunctionType = overtureFunctionType
  id2:IDENTIFIER
  internalASTParametersList = overtureParametersList
  kwde:DOUBLEEQUAL
  internalASTFunctionBody = overtureFunctionBody
  (
   kwpre:PRE
      internalASTExpressionPre = overtureExpression
   {
    keywordPre = new InternalASTKeyword(kwpre,"Pre");
   }
  )?
  (
  kwpost:POST
      internalASTExpressionPost = overtureExpression
   {
    keywordPost = new InternalASTKeyword(kwpost,"Post");
   }
  )?
   {
    InternalASTIdentifier internalASTIdentifier1 = new InternalASTIdentifier(id1);
    InternalASTKeyword keywordColon = new InternalASTKeyword(kwc,"Colon");
    InternalASTIdentifier internalASTIdentifier2 = new InternalASTIdentifier(id2);
    InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde,"DoubleEqual");
    internalASTExplicitFunctionDefinition = new InternalASTExplicitFunctionDefinition(
         internalASTIdentifier1, internalASTTypeVariableList, keywordColon,
         internalASTFunctionType, internalASTIdentifier2, internalASTParametersList,
         keywordDoubleEqual, internalASTFunctionBody, keywordPre, internalASTExpressionPre,
         keywordPost, internalASTExpressionPost);
  }
 ;


//implicit function defnition = identifier, [ type variable list ],
//          parameter types,
//          identifier type pair list,
//          [ 'pre', expression ] ,
//          [ 'post', expression ] ;

overtureImplicitFunctionDefinition returns [InternalASTImplicitFunctionDefinition
     internalASTImplicitFunctionDefinition]
{
 internalASTImplicitFunctionDefinition = null;
 InternalASTTypeVariableList internalASTTypeVariableList = null;
 InternalASTParameterTypes internalASTParameterTypes = null;
 InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
 InternalASTKeyword keywordPre = null;
 InternalASTExpression internalASTExpressionPre = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression internalASTExpressionPost = null;
}
  : id:IDENTIFIER
  (
```

```
        internalASTTypeVariableList = overtureTypeVariableList
    )?
     internalASTParameterTypes = overtureParameterTypes
    internalASTIdentifierTypePairList = overtureIdentifierTypePairList
    (
     kwpre:PRE
        internalASTExpressionPre = overtureExpression
     {
      keywordPre = new InternalASTKeyword(kwpre,"Pre");
      }
    )?
    kwpost:POST
    internalASTExpressionPost = overtureExpression
    {
     keywordPost = new InternalASTKeyword(kwpost,"Post");
      InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
      internalASTImplicitFunctionDefinition = new InternalASTImplicitFunctionDefinition(
           internalASTIdentifier, internalASTTypeVariableList, internalASTParameterTypes,
           internalASTIdentifierTypePairList, keywordPre, internalASTExpressionPre, keywordPost,
           internalASTExpressionPost);
    }
  ;


//extended explicit function defnition = identifier, [ type variable list ],
//             parameter types,
//             identifier type pair list,
//             '==', function body,
//             [ 'pre', expression ] ,
//             [ 'post', expression ] ;

overtureExtendedExplicitFunctionDefinition returns [InternalASTExtendedExplicitFunctionDefinition
      internalASTExtendedExplicitFunctionDefinition]
{
 internalASTExtendedExplicitFunctionDefinition = null;
 InternalASTTypeVariableList internalASTTypeVariableList = null;
 InternalASTParameterTypes internalASTParameterTypes = null;
 InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
 InternalASTFunctionBody internalASTFunctionBody = null;
 InternalASTKeyword keywordPre = null;
 InternalASTExpression internalASTExpressionPre = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression internalASTExpressionPost = null;
}
    : id:IDENTIFIER
    (
        internalASTTypeVariableList = overtureTypeVariableList
    )?
     internalASTParameterTypes = overtureParameterTypes
    internalASTIdentifierTypePairList = overtureIdentifierTypePairList
    de:DOUBLEEQUAL
    internalASTFunctionBody = overtureFunctionBody
    (
     kwpre:PRE
        internalASTExpressionPre = overtureExpression
     {
      keywordPre = new InternalASTKeyword(kwpre,"Pre");
      }
    )?
     (
     kwpost:POST
        internalASTExpressionPost = overtureExpression
     {
      keywordPost = new InternalASTKeyword(kwpost,"Post");
      }
    )?
     {
      InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
      InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(de,"DoubleEqual");
      internalASTExtendedExplicitFunctionDefinition = new
           InternalASTExtendedExplicitFunctionDefinition(internalASTIdentifier,
           internalASTTypeVariableList, internalASTParameterTypes,
           internalASTIdentifierTypePairList, keywordDoubleEqual, internalASTFunctionBody,
           keywordPre, internalASTExpressionPre, keywordPost, internalASTExpressionPost);
    }
  ;

// type variable list = '[', type variable identifier,
//         { ',', type variable identifier }, ']' ;
overtureTypeVariableList returns [InternalASTTypeVariableList internalASTTypeVariableList]
{
 internalASTTypeVariableList = null;
 List<InternalASTTypeVariableListElement> typeVariableListElements = new ArrayList<
      InternalASTTypeVariableListElement>();
```

```
 InternalASTTypeVariableListElement currentElement = null;
 InternalASTTypeVariableIdentifier typeVariableIdentifier = null;
}
  :
    kwlb:LBRACK
    typeVariableIdentifier = overtureTypeVariableIdentifier
     {
      currentElement = new InternalASTTypeVariableListElement(typeVariableIdentifier);
      typeVariableListElements.add(currentElement);
     }
    (
     kwc:COMMA
     typeVariableIdentifier = overtureTypeVariableIdentifier
      {
       InternalASTKeyword keywordComma = new InternalASTKeyword(kwc,"Comma");
       currentElement = new InternalASTTypeVariableListElement(keywordComma,
            typeVariableIdentifier);
       typeVariableListElements.add(currentElement);
      }
    )*
    kwrb:RBRACK

     {
      InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb,"LeftBrack");
      InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb,"RightBrack");
      internalASTTypeVariableList = new InternalASTTypeVariableList(keywordLeftBrack,
           typeVariableListElements, keywordRightBrack);
     }
  ;


//identifier type pair = identifier, ':' type;
overtureIdentifierTypePair returns [InternalASTIdentifierTypePair internalASTIdentifierTypePair]
{
 internalASTIdentifierTypePair = null;
 InternalASTType internalASTType = null;
}
   : id:IDENTIFIER
    kwc:COLON
      internalASTType = overtureType
   {
    InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
     InternalASTKeyword keywordColon = new InternalASTKeyword(kwc,"Colon");
     internalASTIdentifierTypePair = new InternalASTIdentifierTypePair(internalASTIdentifier,
          keywordColon, internalASTType);
   }
   ;


//pattern list type pair = pattern list, ':' type;
overturePatternListTypePair returns [InternalASTPatternListTypePair
      internalASTPatternListTypePair]
{
 internalASTPatternListTypePair = null;
 InternalASTType internalASTType = null;
 InternalASTPatternList patternList = null;
}
   :
   patternList = overturePatternList
    kwc:COLON
      internalASTType = overtureType
   {
    InternalASTKeyword keywordColon = new InternalASTKeyword(kwc,"Colon");
     internalASTPatternListTypePair = new InternalASTPatternListTypePair(patternList, keywordColon
          , internalASTType);
   }
   ;

//parameter types = '(', [ pattern type pair list ], ')';
overtureParameterTypes returns [InternalASTParameterTypes internalASTParameterTypes]
{
 internalASTParameterTypes = null;
 InternalASTPatternTypePairList patternTypePairList = null;
}
   : kwlb:LBRACKET
   (
       patternTypePairList = overturePatternTypePairList
   )?
   kwrb:RBRACKET
   {
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTParameterTypes = new InternalASTParameterTypes(keywordLeftBracket,
          patternTypePairList, keywordRightBracket);
```

```
  }
  ;

//identifier type pair list = identifier type pair,
//          { ',', identifier type pair } ;
overtureIdentifierTypePairList returns [InternalASTIdentifierTypePairList
      internalASTIdentifierTypePairList]
{
 internalASTIdentifierTypePairList = null;
 InternalASTIdentifierTypePairListElement internalASTIdentifierTypePairListElement = null;
 List<InternalASTIdentifierTypePairListElement> identifierTypePairListElements = new ArrayList<
       InternalASTIdentifierTypePairListElement>();
 InternalASTIdentifierTypePair internalASTIdentifierTypePair = null;
}
  :
  internalASTIdentifierTypePair = overtureIdentifierTypePair
   {
    internalASTIdentifierTypePairListElement = new InternalASTIdentifierTypePairListElement(
         internalASTIdentifierTypePair);
    identifierTypePairListElements.add(internalASTIdentifierTypePairListElement);
   }
  (options {greedy=true;}:
   kwcomma:COMMA
     internalASTIdentifierTypePair = overtureIdentifierTypePair
    {
     InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
     internalASTIdentifierTypePairListElement = new InternalASTIdentifierTypePairListElement(
          keywordComma, internalASTIdentifierTypePair);
     identifierTypePairListElements.add(internalASTIdentifierTypePairListElement);
    }
  )*
  {
   internalASTIdentifierTypePairList = new InternalASTIdentifierTypePairList(
        identifierTypePairListElements);
  }
  ;

//pattern type pair list = pattern list type pair,
//         { ',', pattern list type pair } ;
overturePatternTypePairList returns [InternalASTPatternTypePairList
      internalASTPatternTypePairList]
{
 internalASTPatternTypePairList = null;
 InternalASTPatternListTypePair internalASTPatternListTypePair = null;
 InternalASTPatternTypePairListElement internalASTPatternTypePairListElement = null;
 List<InternalASTPatternTypePairListElement> patternTypePairListElements = new ArrayList<
       InternalASTPatternTypePairListElement>();
}
  :
  internalASTPatternListTypePair = overturePatternListTypePair
  {
   internalASTPatternTypePairListElement = new InternalASTPatternTypePairListElement(
        internalASTPatternListTypePair);
   patternTypePairListElements.add(internalASTPatternTypePairListElement);
  }
  (
   kwcomma:COMMA
     internalASTPatternListTypePair = overturePatternListTypePair
  {
   InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
   internalASTPatternTypePairListElement = new InternalASTPatternTypePairListElement(keywordComma
        , internalASTPatternListTypePair);
   patternTypePairListElements.add(internalASTPatternTypePairListElement);
  }
  )*
  {
   internalASTPatternTypePairList = new InternalASTPatternTypePairList(
        patternTypePairListElements);
  }
  ;


//parameters list = parameters, { parameters };
overtureParametersList returns [InternalASTParametersList internalASTParametersList]
{
 internalASTParametersList = null;
 InternalASTParameters internalASTParameters = null;
 List<InternalASTParameters> parameters = new ArrayList<InternalASTParameters>();
}
  :
  internalASTParameters = overtureParameters
  {
   parameters.add(internalASTParameters);
  }
```

```
  (
    internalASTParameters = overtureParameters
   {
    parameters.add(internalASTParameters);
   }
   )*
   {
    internalASTParametersList = new InternalASTParametersList(parameters);
   }
  ;

// parameters = '(', [ pattern list ], ')';
overtureParameters returns [InternalASTParameters internalASTParameters]
{
 internalASTParameters = null;
 InternalASTPatternList internalASTPatternList = null;
}
   :
   kwlb:LBRACKET
   (
    internalASTPatternList = overturePatternList
   )?
   kwrb:RBRACKET
   {
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
    internalASTParameters = new InternalASTParameters(keywordLeftBracket, internalASTPatternList,
        keywordRightBracket);
   }
  ;

// function body = expression
//          | 'Is not yet specified';
//      | 'is subclass responsibility'
overtureFunctionBody returns [InternalASTFunctionBody internalASTFunctionBody]
{
 internalASTFunctionBody = null;
}
  :
       (
       (overtureExpression)=> internalASTFunctionBody = overtureExpression
   |
    (
     is:IS
     no:NOT
     ye:YET
     sp:SPECIFIED
      {
       InternalASTKeyword keywordIs = new InternalASTKeyword(is,"Is");
       InternalASTKeyword keywordNot = new InternalASTKeyword(no,"Not");
       InternalASTKeyword keywordYet = new InternalASTKeyword(ye,"Yet");
       InternalASTKeyword keywordSpecified = new InternalASTKeyword(sp,"Specified");
       internalASTFunctionBody = new InternalASTFunctionBodyIsNotYetSpecified(keywordIs,
            keywordNot, keywordYet, keywordSpecified);
      }
    )
   |
    (
     is2:IS
     su:SUBCLASS
     re:RESPONSIBILITY
      {
       InternalASTKeyword keywordIs = new InternalASTKeyword(is2,"Is");
       InternalASTKeyword keywordSubclass = new InternalASTKeyword(su,"Subclass");
       InternalASTKeyword keywordResponsibility = new InternalASTKeyword(re,"Responsibility");
       internalASTFunctionBody = new InternalASTFunctionBodyIsSubclassResponsibility(keywordIs,
            keywordSubclass, keywordResponsibility);
      }
    )
   )
  ;

// OPERATION DEFINITIONS

// operation definitions = 'values', [access operation definition,
//        {';', access operation definition },
//        [';']];
overtureOperationDefinitions returns [InternalASTOperationDefinitions
      internalASTOperationDefinitions]
{
 internalASTOperationDefinitions = null;
 InternalASTKeyword keywordOperations = null;
 List<InternalASTOperationDefinitionsElement> operationDefinitionsElements = new ArrayList<
       InternalASTOperationDefinitionsElement>();
```

```
 InternalASTOperationDefinitionsElement currentElement = null;
 InternalASTAccessOperationDefinition internalASTAccessOperationDefinition = null;
}
  :
   op:OPERATIONS
   {
    keywordOperations = new InternalASTKeyword(op,"Operations");
   }
   (
    internalASTAccessOperationDefinition = overtureAccessOperationDefinition
     {
      currentElement = new InternalASTOperationDefinitionsElement(
           internalASTAccessOperationDefinition);
      operationDefinitionsElements.add(currentElement);
     }

    (options {greedy=true;}:
     seNoN:SEMICOLON
     {
      InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonNoN);
      currentElement.setEndPositionFromNode(semicolonNoN);
     }
     internalASTAccessOperationDefinition = overtureAccessOperationDefinition
     {
      currentElement = new InternalASTOperationDefinitionsElement(
           internalASTAccessOperationDefinition);
        operationDefinitionsElements.add(currentElement);
     }
    )*

    (seLast:SEMICOLON
              {
      InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonLast);
      currentElement.setEndPositionFromNode(semicolonLast);
              }
    )?

   )?
   {
    internalASTOperationDefinitions = new InternalASTOperationDefinitions(keywordOperations,
         operationDefinitionsElements);
   }
  ;


// access operation definition = [access], operation definition;
overtureAccessOperationDefinition returns [InternalASTAccessOperationDefinition
      internalASTAccessOperationDefinition]
{
 internalASTAccessOperationDefinition = null;
 InternalASTAccess internalASTAccess = null;
 InternalASTOperationDefinition internalASTOperationDefinition = null;
}
  :
   (
    internalASTAccess = overtureAccess
   )?
   internalASTOperationDefinition = overtureOperationDefinition
   {
      internalASTAccessOperationDefinition = new InternalASTAccessOperationDefinition(
           internalASTAccess, internalASTOperationDefinition);
   }
  ;

// operation definition = explicit operation definition
//                  | implicit operation definition;
//           | extended explicit operation definition
overtureOperationDefinition returns [InternalASTOperationDefinition
      internalASTOperationDefinition]
{
 internalASTOperationDefinition = null;
}
  :
       (
   internalASTOperationDefinition = overtureExplicitOperationDefinition
   | (overtureImplicitOperationDefinition) => internalASTOperationDefinition =
        overtureImplicitOperationDefinition
   | internalASTOperationDefinition = overtureExtendedExplicitOperationDefinition
   )
  ;

//explicit operation defnition = identifier, ':', operation type,
```

```
//          identifier, parameters,
//          '==', operation body,
//          [ 'pre', expression ] ,
//          [ 'post', expression ] ;
overtureExplicitOperationDefinition returns [InternalASTExplicitOperationDefinition
    internalASTExplicitOperationDefinition]
{
 internalASTExplicitOperationDefinition = null;
 InternalASTOperationType internalASTOperationType = null;
 InternalASTParameters internalASTParameters = null;
 InternalASTOperationBody internalASTOperationBody = null;
 InternalASTKeyword keywordPre = null;
 InternalASTExpression internalASTExpressionPre = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression internalASTExpressionPost = null;
}
  : id1:IDENTIFIER
  kwc:COLON
   internalASTOperationType = overtureOperationType
  id2:IDENTIFIER
  internalASTParameters = overtureParameters
  kwde:DOUBLEEQUAL
  internalASTOperationBody = overtureOperationBody
  (
   kwpre:PRE
      internalASTExpressionPre = overtureExpression
   {
    keywordPre = new InternalASTKeyword(kwpre,"Pre");
    }
  )?
   (
   kwpost:POST
      internalASTExpressionPost = overtureExpression
   {
    keywordPost = new InternalASTKeyword(kwpost,"Post");
    }
  )?
   {
    InternalASTIdentifier internalASTIdentifier1 = new InternalASTIdentifier(id1);
    InternalASTKeyword keywordColon = new InternalASTKeyword(kwc,"Colon");
    InternalASTIdentifier internalASTIdentifier2 = new InternalASTIdentifier(id2);
    InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde,"DoubleEqual");
    internalASTExplicitOperationDefinition = new InternalASTExplicitOperationDefinition(
        internalASTIdentifier1, keywordColon, internalASTOperationType, internalASTIdentifier2,
        internalASTParameters, keywordDoubleEqual, internalASTOperationBody, keywordPre,
        internalASTExpressionPre, keywordPost, internalASTExpressionPost);
  }
 ;


//implicit operation definition = identifier, parameter types,
//         [ identifier type pair list ],
//         implicit operation body;
overtureImplicitOperationDefinition returns [InternalASTImplicitOperationDefinition
    internalASTImplicitOperationDefinition]
{
 internalASTImplicitOperationDefinition = null;
 InternalASTParameterTypes internalASTParameterTypes = null;
 InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
 InternalASTImplicitOperationBody internalASTImplicitOperationBody = null;
}
  : id:IDENTIFIER
  internalASTParameterTypes = overtureParameterTypes
  (
   internalASTIdentifierTypePairList = overtureIdentifierTypePairList
  )?
  internalASTImplicitOperationBody = overtureImplicitOperationBody
  {
    InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
    internalASTImplicitOperationDefinition = new InternalASTImplicitOperationDefinition(
        internalASTIdentifier, internalASTParameterTypes, internalASTIdentifierTypePairList,
        internalASTImplicitOperationBody);
  }
 ;

//implicit operation body = [ externals ],
//                        [ 'pre', expression ],
//           'post', expression,
//        [ exceptions] ;
overtureImplicitOperationBody returns [InternalASTImplicitOperationBody
    internalASTImplicitOperationBody]
{
 internalASTImplicitOperationBody = null;
 InternalASTExternals internalASTExternals = null;
```

```
 InternalASTKeyword keywordPre = null;
 InternalASTExpression internalASTExpressionPre = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression internalASTExpressionPost = null;
 InternalASTExceptions internalASTExceptions = null;
}
   :
   (
    internalASTExternals = overtureExternals
   )?
   (
    kwpre:PRE
       internalASTExpressionPre = overtureExpression
       {
        keywordPre = new InternalASTKeyword(kwpre,"Pre");
       }
   )?
   kwpost:POST
      internalASTExpressionPost = overtureExpression
      (
    internalASTExceptions = overtureExceptions
   )?
    {
     keywordPost = new InternalASTKeyword(kwpost,"Post");
       internalASTImplicitOperationBody = new InternalASTImplicitOperationBody(
            internalASTExternals, keywordPre, internalASTExpressionPre, keywordPost,
            internalASTExpressionPost, internalASTExceptions);
   }
  ;

//extended explicit operation definition = identifier, parameter types,
//           [ identifier type pair list ],
//              '==', operation body,
//             [externals],
//           ['pre', expression],
//           ['post', expression],
//           [exceptions]  ;
overtureExtendedExplicitOperationDefinition returns [
     InternalASTExtendedExplicitOperationDefinition
     internalASTExtendedExplicitOperationDefinition]
{
 internalASTExtendedExplicitOperationDefinition = null;
 InternalASTParameterTypes internalASTParameterTypes = null;
 InternalASTIdentifierTypePairList internalASTIdentifierTypePairList = null;
 InternalASTOperationBody internalASTOperationBody = null;
 InternalASTExternals externals = null;
 InternalASTKeyword keywordPre = null;
 InternalASTExpression preExpression = null;
 InternalASTKeyword keywordPost = null;
 InternalASTExpression postExpression = null;
 InternalASTExceptions exceptions = null;
}
   : id:IDENTIFIER
   internalASTParameterTypes = overtureParameterTypes
   (
    internalASTIdentifierTypePairList = overtureIdentifierTypePairList
   )?
   kwde:DOUBLEEQUAL
   internalASTOperationBody = overtureOperationBody
   (
    externals = overtureExternals
   )?
   (
    pr:PRE
    preExpression = overtureExpression
     {
      keywordPre = new InternalASTKeyword(pr, "Pre");
     }
   )?
   (
    po:POST
    postExpression = overtureExpression
     {
      keywordPost = new InternalASTKeyword(po, "Post");
     }
   )?
   (
    exceptions = overtureExceptions
   )?
     {
      InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
      InternalASTKeyword keywordDoubleEqual = new InternalASTKeyword(kwde,"DoubleEqual");
      internalASTExtendedExplicitOperationDefinition = new
           InternalASTExtendedExplicitOperationDefinition(internalASTIdentifier,
```

```
        internalASTParameterTypes, internalASTIdentifierTypePairList, keywordDoubleEqual,
        internalASTOperationBody, externals, keywordPre, preExpression, keywordPost,
        postExpression, exceptions);
  }
 ;

//operation type = discritionary type, '==>', discritionary type;
overtureOperationType returns [InternalASTOperationType internalASTOperationType]
{
 internalASTOperationType = null;
 InternalASTDiscretionaryType internalASTDiscretionaryType1 = null;
 InternalASTDiscretionaryType internalASTDiscretionaryType2 = null;
}
  :
  internalASTDiscretionaryType1 = overtureDiscretionaryType
  kwei:EQUALIMPLY
  internalASTDiscretionaryType2 = overtureDiscretionaryType
  {
    InternalASTKeyword keyword = new InternalASTKeyword(kwei,"EqualImply");
    internalASTOperationType = new InternalASTOperationType(internalASTDiscretionaryType1,
        keyword, internalASTDiscretionaryType2);
  }
 ;

// operation body = statement
//         | 'Is not yet specified';
//     | 'is subclass responsibility'
overtureOperationBody returns [InternalASTOperationBody internalASTOperationBody]
{
 internalASTOperationBody = null;
}
 :
      (
  internalASTOperationBody = overtureStatement
  |
   (
    is:IS
    no:NOT
    ye:YET
    sp:SPECIFIED
     {
      InternalASTKeyword keywordIs = new InternalASTKeyword(is,"Is");
      InternalASTKeyword keywordNot = new InternalASTKeyword(no,"Not");
      InternalASTKeyword keywordYet = new InternalASTKeyword(ye,"Yet");
      InternalASTKeyword keywordSpecified = new InternalASTKeyword(sp,"Specified");
      internalASTOperationBody = new InternalASTOperationBodyIsNotYetSpecified(keywordIs,
          keywordNot, keywordYet, keywordSpecified);
     }
   )
  |
   (
    is2:IS
    su:SUBCLASS
    re:RESPONSIBILITY
     {
      InternalASTKeyword keywordIs = new InternalASTKeyword(is2,"Is");
      InternalASTKeyword keywordSubclass = new InternalASTKeyword(su,"Subclass");
      InternalASTKeyword keywordResponsibility = new InternalASTKeyword(re,"Responsibility");
      internalASTOperationBody = new InternalASTOperationBodyIsSubclassResponsibility(keywordIs,
          keywordSubclass, keywordResponsibility);
     }
   )
  )
 ;

//externals = 'ext', var information, {var information};
overtureExternals returns [InternalASTExternals internalASTExternals]
{
 internalASTExternals = null;
 List<InternalASTVarInformation> varInformations = new ArrayList<InternalASTVarInformation>();

 InternalASTVarInformation internalASTVarInformation = null;
}
  : kwext:EXT
     internalASTVarInformation = overtureVarInformation
     {
      varInformations.add(internalASTVarInformation);
     }
  (
   internalASTVarInformation = overtureVarInformation
     {
      varInformations.add(internalASTVarInformation);
     }
  )*
```

```
   {
    InternalASTKeyword keywordExt = new InternalASTKeyword(kwext,"Ext");
     internalASTExternals = new InternalASTExternals(keywordExt, varInformations);
   }
  ;

//var information = mode, name list , [ ':', type ];
overtureVarInformation returns [InternalASTVarInformation internalASTVarInformation]
{
 internalASTVarInformation = null;
 InternalASTMode internalASTMode = null;
 InternalASTNameList internalASTNameList = null;
 InternalASTType internalASTType = null;
 InternalASTKeyword keywordColon = null;
}
   :
   internalASTMode = overtureMode
   internalASTNameList = overtureNameList

   (
       kwc:COLON
    internalASTType = overtureType
    {
    keywordColon = new InternalASTKeyword(kwc,"Colon");
     }
   )?

    {
     internalASTVarInformation = new InternalASTVarInformation(internalASTMode,
          internalASTNameList, keywordColon, internalASTType);
   }
  ;

//mode = 'rd' | 'wr';
overtureMode returns [InternalASTMode internalASTMode]
{
 internalASTMode = null;
 InternalASTModeRd internalASTModeRd = null;
 InternalASTModeWr internalASTModeWr = null;
}
  :
   ( rd:RD
    {
     InternalASTKeyword keywordRd = new InternalASTKeyword(rd,"Rd");
     internalASTMode = new InternalASTModeRd(keywordRd);
    }
   | wr:WR
    {
     InternalASTKeyword keywordWr = new InternalASTKeyword(wr,"Wr");
     internalASTMode = new InternalASTModeWr(keywordWr);
    }
   )
  ;

//exceptions = 'errs', error list ;
overtureExceptions returns [InternalASTExceptions internalASTExceptions]
{
 internalASTExceptions = null;
 InternalASTErrorList errorList = null;
}
  :
   errs:ERRS
   errorList = overtureErrorList
    {
     InternalASTKeyword keywordErrs = new InternalASTKeyword(errs, "Errs");
     internalASTExceptions = new InternalASTExceptions(keywordErrs, errorList);
    }
  ;

//error list = error, {error} ;
overtureErrorList returns [InternalASTErrorList internalASTErrorList]
{
 internalASTErrorList = null;
 List<InternalASTError> errors = new ArrayList<InternalASTError>();
 InternalASTError error = null;
}
  :
   (
    error = overtureError
     {
      errors.add(error);
     }
    (
    error = overtureError
```

```
      {
       errors.add(error);
      }
     )*
    )
    {
     internalASTErrorList = new InternalASTErrorList(errors);
    }
  ;

//error = identifier, ':', expression, '->', expression ;
overtureError returns [InternalASTError internalASTError]
{
 internalASTError = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   id:IDENTIFIER
   co:COLON
   expression1 = overtureExpression
   la:LINEARROW
   expression2 = overtureExpression
    {
     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
     InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
     InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
     internalASTError = new InternalASTError(identifier, keywordColon, expression1,
          keywordLineArrow, expression2);
    }
  ;

// Instance Variable Definitions

//instance variable definitions = 'instance', 'variables',
//                                [ instance variable definition,
//                                { ';', instance variable definition } ];
overtureInstanceVariableDefinitions returns [InternalASTInstanceVariableDefinitions
     internalASTInstanceVariableDefinitions]
{
 internalASTInstanceVariableDefinitions = null;
 List<InternalASTInstanceVariableDefinitionsElement> elementList = new ArrayList<
      InternalASTInstanceVariableDefinitionsElement>();
 InternalASTInstanceVariableDefinition iVDefinition = null;
 InternalASTInstanceVariableDefinitionsElement currentElement = null;
}
  :
   ( in:INSTANCE
   va:VARIABLES
   (
    iVDefinition = overtureInstanceVariableDefinition
     {
      currentElement = new InternalASTInstanceVariableDefinitionsElement(iVDefinition);
      elementList.add(currentElement);
     }
    (
     se:SEMICOLON
     iVDefinition = overtureInstanceVariableDefinition
      {
       InternalASTKeyword keywordSemicolon = new InternalASTKeyword(se, "Semicolon");
       currentElement = new InternalASTInstanceVariableDefinitionsElement(keywordSemicolon,
            iVDefinition);
       elementList.add(currentElement);
      }
    )*
   )? )
   {
    InternalASTKeyword keywordInstance = new InternalASTKeyword(in, "Instance");
    InternalASTKeyword keywordVariables = new InternalASTKeyword(va, "Variables");
    internalASTInstanceVariableDefinitions = new InternalASTInstanceVariableDefinitions(
         keywordInstance, keywordVariables, elementList);
   }
  ;

// instance variable definition = access assignment definition
//            | invariant definition
//            | init definition;
overtureInstanceVariableDefinition returns [InternalASTInstanceVariableDefinition
     internalASTInstanceVariableDefinition]
{
 internalASTInstanceVariableDefinition = null;
}
  :
   internalASTInstanceVariableDefinition = overtureAccessAssignmentDefinition
```

```
  | internalASTInstanceVariableDefinition = overtureInvariantDefinition
  | internalASTInstanceVariableDefinition = overtureInitStatement
 ;

// access assignment definition = [ access ], assignment definition;
overtureAccessAssignmentDefinition returns [InternalASTAccessAssignmentDefinition
    internalASTAccessAssignmentDefinition]
{
 internalASTAccessAssignmentDefinition = null;
 InternalASTAccess access = null;
 InternalASTAssignmentDefinition assignmentDefinition = null;
}
 :
  (
   access = overtureAccess
  )?
  assignmentDefinition = overtureAssignmentDefinition
   {
    internalASTAccessAssignmentDefinition = new InternalASTAccessAssignmentDefinition(access,
         assignmentDefinition);
   }
 ;

// invariant definition = 'inv', expression ;
overtureInvariantDefinition returns [InternalASTInvariantDefinition
    internalASTInvariantDefinition]
{
 internalASTInvariantDefinition = null;
 InternalASTExpression expression = null;
}
 :
  in:INV
  expression = overtureExpression
   {
    InternalASTKeyword keywordInv = new InternalASTKeyword(in, "Inv");
    internalASTInvariantDefinition = new InternalASTInvariantDefinition(keywordInv, expression);
   }
 ;

// init statement = 'init', statement ;
overtureInitStatement returns [InternalASTInitStatement internalASTInitStatement]
{
 internalASTInitStatement = null;
 InternalASTStatement statement = null;
}
 :
  in:INIT
  statement = overtureStatement
   {
    InternalASTKeyword keywordInit = new InternalASTKeyword(in, "Init");
    internalASTInitStatement = new InternalASTInitStatement(keywordInit, statement);
   }
 ;

// SYNCHRONIZATION DEFINITIONS

//synchronization definitions = 'sync', [ synchronization ];
overtureSynchronizationDefinitions returns [InternalASTSynchronizationDefinitions
    internalASTSynchronizationDefinitions]
{
 internalASTSynchronizationDefinitions = null;
 InternalASTSynchronization synchronization = null;
}
 :
  (
   sy:SYNC
   (
     synchronization = overtureSynchronization
   )?
  )
  {
   InternalASTKeyword keywordSync = new InternalASTKeyword(sy, "Sync");
   internalASTSynchronizationDefinitions = new InternalASTSynchronizationDefinitions(keywordSync,
         synchronization);
  }
 ;

//synchronization = permission predicates;
overtureSynchronization returns [InternalASTSynchronization internalASTSynchronization]
{
 internalASTSynchronization = null;
 InternalASTPermissionPredicates permPredicates = null;
}
 :
```

```
  permPredicates = overturePermissionPredicates
   {
    internalASTSynchronization = new InternalASTSynchronization(permPredicates);
   }
 ;

//permission predicates = permission predicate
//          { ';', permission predicate };
overturePermissionPredicates returns [InternalASTPermissionPredicates internalASTPermissionPredicates]
{
 internalASTPermissionPredicates = null;
 List<InternalASTPermissionPredicatesElement> permissionPredicatesElements = new ArrayList<InternalASTPermissionPredicatesElement>();
 InternalASTPermissionPredicatesElement currentElement = null;
 InternalASTPermissionPredicate currentPredicate = null;
}
 :
  currentPredicate = overturePermissionPredicate
   {
    currentElement = new InternalASTPermissionPredicatesElement(currentPredicate);
    permissionPredicatesElements.add(currentElement);
   }
  (
   se:SEMICOLON
   currentPredicate=overturePermissionPredicate
   {
    InternalASTKeyword  keywordSemicolon = new InternalASTKeyword(se, "Semicolon");
    currentElement = new InternalASTPermissionPredicatesElement(keywordSemicolon, currentPredicate);
    permissionPredicatesElements.add(currentElement);
   }
  )*
  {
   internalASTPermissionPredicates = new InternalASTPermissionPredicates(permissionPredicatesElements);
  }
 ;

//permission predicate = 'per', name '=>' expression
//          | mutex predicate ;
overturePermissionPredicate returns [InternalASTPermissionPredicate internalASTPermissionPredicate]
{
 internalASTPermissionPredicate = null;
 InternalASTName name = null;
 InternalASTExpression expression = null;
 InternalASTMutexPredicate mutexpredicate = null;
}
 :
  (
   (
    pe:PER
    name = overtureName
    im:IMPLY
    expression = overtureExpression
   )
    {
     InternalASTKeyword keywordPer = new InternalASTKeyword(pe, "Per");
     InternalASTKeyword keywordImply = new InternalASTKeyword(im, "Imply");
     internalASTPermissionPredicate = new InternalASTPermissionPredicateElement1(keywordPer, name, keywordImply, expression);
    }
  )
  |
  (
   (
    mutexpredicate = overtureMutexPredicate
   )
    {
     internalASTPermissionPredicate = new InternalASTPermissionPredicateElement2(mutexpredicate);
    }
  )
 ;

//mutex predicate = 'mutex', '(', 'all', ')'
//        | 'mutex', '(', name list, ')';
overtureMutexPredicate returns [InternalASTMutexPredicate internalASTMutexPredicate]
{
 internalASTMutexPredicate = null;
 InternalASTNameList namelist = null;
}
 :
  (MUTEX LBRACKET ALL RBRACKET)
```

```
    =>((
     mu:MUTEX
     lb:LBRACKET
     al:ALL
     rb:RBRACKET
    )
     {
      InternalASTKeyword keywordMutex = new InternalASTKeyword(mu, "Mutex");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
      InternalASTKeyword keywordAll = new InternalASTKeyword(al, "All");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
      internalASTMutexPredicate = new InternalASTMutexPredicateAll(keywordMutex,
           keywordLeftBracket, keywordAll, keywordRightBracket);
     }
   )
   |
   (
    (
     mu2:MUTEX
     lb2:LBRACKET
     namelist = overtureNameList
     rb2:RBRACKET
    )
     {
      InternalASTKeyword keywordMutex = new InternalASTKeyword(mu2, "Mutex");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2, "LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2, "RightBracket");
      internalASTMutexPredicate = new InternalASTMutexPredicateNameList(keywordMutex,
           keywordLeftBracket, namelist, keywordRightBracket);
     }
   )
  ;

// THREAD DEFINITIONS

//thread definitions = 'thread', [thread definition];
overtureThreadDefinitions returns [InternalASTThreadDefinitions internalASTThreadDefinitions]
{
 internalASTThreadDefinitions = null;
 InternalASTThreadDefinition threadDefinition = null;
}
  :
   th:THREAD
   ( threadDefinition = overtureThreadDefinition )?
   {
    InternalASTKeyword keywordThread = new InternalASTKeyword(th, "Thread");
    internalASTThreadDefinitions = new InternalASTThreadDefinitions(keywordThread,
         threadDefinition);
   }
  ;

//thread definition = procedural thread definition
overtureThreadDefinition returns [InternalASTThreadDefinition internalASTThreadDefinition]
{
 internalASTThreadDefinition = null;
 InternalASTProceduralThreadDefinition proceduralThreadDefinition = null;
}
  :
   proceduralThreadDefinition = overtureProceduralThreadDefinition
    {
     internalASTThreadDefinition = new InternalASTThreadDefinition(proceduralThreadDefinition);
    }
  ;

//proceduralthread definition = statement
overtureProceduralThreadDefinition returns [InternalASTProceduralThreadDefinition
     internalASTProceduralThreadDefinition]
{
 internalASTProceduralThreadDefinition = null;
 InternalASTStatement statement = null;
}
  :
   statement = overtureStatement
    {
     internalASTProceduralThreadDefinition = new InternalASTProceduralThreadDefinition(statement);
    }
  ;

// EXPRESSIONS

// expression list = expression , { ',', expression };
overtureExpressionList returns [InternalASTExpressionList internalASTExpressionList]
{
 internalASTExpressionList = null;
```

```
 List<InternalASTExpressionListElement> expressionListElements = new ArrayList<
      InternalASTExpressionListElement>();
 InternalASTExpressionListElement currentElement = null;
 InternalASTExpression expression = null;
}
 :
  (
   expression = overtureExpression
    {
     currentElement = new InternalASTExpressionListElement(expression);
     expressionListElements.add(currentElement);
    }
   (options {greedy=true;}:
    co:COMMA
    expression = overtureExpression
     {
      InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
      currentElement = new InternalASTExpressionListElement(keywordComma, expression);
      expressionListElements.add(currentElement);
     }
   )*
  )
  {
   internalASTExpressionList = new InternalASTExpressionList(expressionListElements);
  }
 ;

// expression = ...
overtureExpression returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
}
 :
  internalASTExpression = overtureExpressionPre00
 ;

overtureExpressionPre00 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
}
 :
    (overtureQuantifiedExpression)=> internalASTExpression = overtureQuantifiedExpression
    | (overtureIotaExpression)=> internalASTExpression = overtureIotaExpression
    | (overtureSetEnumeration)=> internalASTExpression = overtureSetEnumeration
    | (overtureSetComprehension)=> internalASTExpression = overtureSetComprehension
    | (overtureSetRangeExpression)=> internalASTExpression = overtureSetRangeExpression
    | (overtureSequenceEnumeration)=> internalASTExpression = overtureSequenceEnumeration
    | (overtureSequenceComprehension)=> internalASTExpression = overtureSequenceComprehension
    | (overtureMapEnumeration)=> internalASTExpression = overtureMapEnumeration
    | (overtureMapComprehension)=> internalASTExpression = overtureMapComprehension
    | (overtureTupleConstructor)=> internalASTExpression = overtureTupleConstructor
    | (overtureRecordConstructor)=> internalASTExpression = overtureRecordConstructor
    | (overtureRecordModifier)=> internalASTExpression = overtureRecordModifier
    | (overtureTupleSelect)=> internalASTExpression = overtureTupleSelect
    | internalASTExpression = overtureLambdaExpression
    | internalASTExpression = overtureNewExpression
    | internalASTExpression = overtureGeneralIsExpression
    | internalASTExpression = overtureIsofbaseclassExpression
    | internalASTExpression = overtureIsofclassExpression
    | internalASTExpression = overtureSamebaseclassExpression
    | internalASTExpression = overtureSameclassExpression
    | internalASTExpression = overtureActExpression
    | internalASTExpression = overtureFinExpression
    | internalASTExpression = overtureActiveExpression
    | internalASTExpression = overtureReqExpression
    | internalASTExpression = overtureWaitingExpression
    | internalASTExpression = overtureExpressionPre21
 ;

// Optional grouping... - right grouping chosen!
overtureExpressionPre21 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
 :
  (overtureExpressionPre22 overtureBinaryOperatorLogicalEquivalence overtureExpressionPre21) =>
   (
    expression1 = overtureExpressionPre22
    operator = overtureBinaryOperatorLogicalEquivalence
    expression2 = overtureExpressionPre21
     {
```

```
      internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
     }
    )
   |
   internalASTExpression = overtureExpressionPre22
  ;

overtureExpressionPre22 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre23 overtureBinaryOperatorImply overtureExpressionPre22) =>
    (
     expression1 = overtureExpressionPre23
     operator = overtureBinaryOperatorImply
     expression2 = overtureExpressionPre22
      {
      internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
     }
    )
   |
   internalASTExpression = overtureExpressionPre23
  ;

overtureExpressionPre23 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre24 overtureBinaryOperatorOr overtureExpressionPre23) =>
    (
     expression1 = overtureExpressionPre24
     operator = overtureBinaryOperatorOr
     expression2 = overtureExpressionPre23
      {
      internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
     }
    )
   |
   internalASTExpression = overtureExpressionPre24
  ;

overtureExpressionPre24 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre25 overtureBinaryOperatorAnd overtureExpressionPre24) =>
    (
     expression1 = overtureExpressionPre25
     operator = overtureBinaryOperatorAnd
     expression2 = overtureExpressionPre24
      {
      internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
     }
    )
   |
   internalASTExpression = overtureExpressionPre25
  ;

overtureExpressionPre25 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTUnaryOperator operator = null;
 InternalASTExpression expression = null;
}
  :
   (
    operator = overtureUnaryOperatorNot
    expression = overtureExpressionPre31
```

```
    {
     internalASTExpression = new InternalASTPrefixExpression(operator, expression);
    }
   )
   |
   internalASTExpression = overtureExpressionPre31
  ;

overtureExpressionPre31 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre41
   (
    overtureBinaryOperatorLessThan
     | overtureBinaryOperatorLessThanOrEqual
     | overtureBinaryOperatorGreaterThan
     | overtureBinaryOperatorGreaterThanOrEqual
     | overtureBinaryOperatorEqual
     | overtureBinaryOperatorApprox
     | overtureBinaryOperatorNotEqual
     | overtureBinaryOperatorInSet
     | overtureBinaryOperatorNotInSet
     | overtureBinaryOperatorSubset
     | overtureBinaryOperatorProperSubset
   )
   overtureExpressionPre41
   ) =>
    (
     expression1 = overtureExpressionPre41
     (
      operator = overtureBinaryOperatorLessThan
       | operator = overtureBinaryOperatorLessThanOrEqual
       | operator = overtureBinaryOperatorGreaterThan
       | operator = overtureBinaryOperatorGreaterThanOrEqual
       | operator = overtureBinaryOperatorEqual
       | operator = overtureBinaryOperatorApprox
       | operator = overtureBinaryOperatorNotEqual
       | operator = overtureBinaryOperatorInSet
       | operator = overtureBinaryOperatorNotInSet
       | operator = overtureBinaryOperatorSubset
       | operator = overtureBinaryOperatorProperSubset
     )
     expression2 = overtureExpressionPre41
      {
       internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2)
           ;
      }
     )
   |
   internalASTExpression = overtureExpressionPre41
  ;

overtureExpressionPre41 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre41u
   (
    overtureBinaryOperatorArithmeticPlus
     | overtureBinaryOperatorArithmeticMinus
     | overtureBinaryOperatorSetUnion
     | overtureBinaryOperatorSetDifference
     | overtureBinaryOperatorSequenceConcatenate
     | overtureBinaryOperatorMapOrSequenceModify
     | overtureBinaryOperatorMapMerge
   )
   overtureExpressionPre41u) =>
    (
     expression1 = overtureExpressionPre41u
     (
      operator = overtureBinaryOperatorArithmeticPlus
       | operator = overtureBinaryOperatorArithmeticMinus
       | operator = overtureBinaryOperatorSetUnion
       | operator = overtureBinaryOperatorSetDifference
       | operator = overtureBinaryOperatorSequenceConcatenate
```

```
     | operator = overtureBinaryOperatorMapOrSequenceModify
     | operator = overtureBinaryOperatorMapMerge
    )
    expression2 = overtureExpressionPre41u
     {
      expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
     }

    (options {greedy=true;}:
     (
      operator = overtureBinaryOperatorArithmeticPlus
      | operator = overtureBinaryOperatorArithmeticMinus
      | operator = overtureBinaryOperatorSetUnion
      | operator = overtureBinaryOperatorSetDifference
      | operator = overtureBinaryOperatorSequenceConcatenate
      | operator = overtureBinaryOperatorMapOrSequenceModify
      | operator = overtureBinaryOperatorMapMerge
     )
     expression2 = overtureExpressionPre41u
      {
       expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
      }
    )*

    {
     internalASTExpression = expression1;
    }
   )
  |
  internalASTExpression = overtureExpressionPre41u
 ;

overtureExpressionPre41u returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTUnaryOperator operator = null;
 InternalASTExpression expression = null;
}
 :
  (
   (
    operator = overtureUnaryOperatorDistributedSetUnion
    | operator = overtureUnaryOperatorDistributedSetIntersection
   )
   expression = overtureExpressionPre41
   {
    internalASTExpression = new InternalASTPrefixExpression(operator, expression);
   }
  )
  |
  internalASTExpression = overtureExpressionPre42
 ;

overtureExpressionPre42 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
 :
  (overtureExpressionPre44
  (
   overtureBinaryOperatorArithmeticMultiplication
   | overtureBinaryOperatorArithmeticDivide
   | overtureBinaryOperatorArithmeticIntegerDivision
   | overtureBinaryOperatorArithmeticRem
   | overtureBinaryOperatorArithmeticMod
   | overtureBinaryOperatorSetIntersection
  )
  overtureExpressionPre44) =>
   (
    expression1 = overtureExpressionPre44
    (
     operator = overtureBinaryOperatorArithmeticMultiplication
     | operator = overtureBinaryOperatorArithmeticDivide
     | operator = overtureBinaryOperatorArithmeticIntegerDivision
     | operator = overtureBinaryOperatorArithmeticRem
     | operator = overtureBinaryOperatorArithmeticMod
     | operator = overtureBinaryOperatorSetIntersection
    )
    expression2 = overtureExpressionPre44
     {
      expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
```

```
      }

     (options {greedy=true;}:
      (

       operator = overtureBinaryOperatorArithmeticMultiplication
       | operator = overtureBinaryOperatorArithmeticDivide
       | operator = overtureBinaryOperatorArithmeticIntegerDivision
       | operator = overtureBinaryOperatorArithmeticRem
       | operator = overtureBinaryOperatorArithmeticMod
       | operator = overtureBinaryOperatorSetIntersection
      )
      expression2 = overtureExpressionPre44
       {
        expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
       }
     )*

     {
      internalASTExpression = expression1;
     }
    )
   |
   internalASTExpression = overtureExpressionPre44
  ;

overtureExpressionPre44 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
  (overtureExpressionPre45
   (
    overtureBinaryOperatorMapDomainRestrictTo
    | overtureBinaryOperatorMapDomainRestrictBy
   )
   overtureExpressionPre44) =>
    (
     expression1 = overtureExpressionPre45
     (
      operator = overtureBinaryOperatorMapDomainRestrictTo
      | operator = overtureBinaryOperatorMapDomainRestrictBy
     )
     expression2 = overtureExpressionPre44
      {
       internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
      }
     )
   |
   internalASTExpression = overtureExpressionPre45
  ;

overtureExpressionPre45 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
  (overtureExpressionPre46u
   (
    overtureBinaryOperatorMapRangeRestrictTo
    | overtureBinaryOperatorMapRangeRestrictBy
   )
   overtureExpressionPre46u) =>
    (
     expression1 = overtureExpressionPre46u
     (
      operator = overtureBinaryOperatorMapRangeRestrictTo
      | operator = overtureBinaryOperatorMapRangeRestrictBy
     )
     expression2 = overtureExpressionPre46u
      {
       expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
      }

     (options {greedy=true;}:
      (

```

```
        operator = overtureBinaryOperatorMapRangeRestrictTo
        | operator = overtureBinaryOperatorMapRangeRestrictBy
       )
       expression2 = overtureExpressionPre46u
        {
         expression1 = new InternalASTBinaryExpression(expression1, operator, expression2);
        }
      )*

      {
       internalASTExpression = expression1;
      }
     )
    |
    internalASTExpression = overtureExpressionPre46u
  ;

overtureExpressionPre46u returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTUnaryOperator operator = null;
 InternalASTExpression expression = null;
}
  :
   (
    (
     operator = overtureUnaryOperatorUnaryPlus
     | operator = overtureUnaryOperatorUnaryMinus
     | operator = overtureUnaryOperatorArithmeticAbs
     | operator = overtureUnaryOperatorFloor
     | operator = overtureUnaryOperatorSetCardinality
     | operator = overtureUnaryOperatorFinitePowerSet
     | operator = overtureUnaryOperatorSequenceHead
     | operator = overtureUnaryOperatorSequenceTail
     | operator = overtureUnaryOperatorSequenceLength
     | operator = overtureUnaryOperatorSequenceElements
     | operator = overtureUnaryOperatorSequenceIndices
     | operator = overtureUnaryOperatorDistributedSequenceConcatenation
     | operator = overtureUnaryOperatorMapDomain
     | operator = overtureUnaryOperatorMapRange
     | operator = overtureUnaryOperatorDistributedMapMerge
    )
    expression = overtureExpressionPre46u
    {
     internalASTExpression = new InternalASTPrefixExpression(operator, expression);
    }
   )

   |
   internalASTExpression = overtureMapInverse
   |
   internalASTExpression = overtureExpressionPre50
  ;

overtureExpressionPre50 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
}
  :
   (overtureApply)=> internalASTExpression = overtureApply
   | (overtureSubsequence)=> internalASTExpression = overtureSubsequence
   | (overtureFieldSelect)=> internalASTExpression = overtureFieldSelect
   | (overtureFunctionTypeInstantiation)=> internalASTExpression =
       overtureFunctionTypeInstantiation
   | internalASTExpression = overtureExpressionPre61
  ;

overtureExpressionPre61 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
  :
   (overtureExpressionPre62
   overtureBinaryOperatorComposition
   overtureExpressionPre61)=>
   (expression1 = overtureExpressionPre62
   operator = overtureBinaryOperatorComposition
   expression2 = overtureExpressionPre61
    {
     internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
    }
```

```
  )
  | internalASTExpression = overtureExpressionPre62
 ;

overtureExpressionPre62 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTBinaryOperator operator = null;
 InternalASTExpression expression2 = null;
}
 :
  (overtureExpressionPre70
  overtureBinaryOperatorIterate
  overtureExpressionPre62)=>
  (expression1 = overtureExpressionPre70
  operator = overtureBinaryOperatorIterate
  expression2 = overtureExpressionPre62
   {
    internalASTExpression = new InternalASTBinaryExpression(expression1, operator, expression2);
   }
  )
  | internalASTExpression = overtureExpressionPre70
 ;

overtureExpressionPre70 returns [InternalASTExpression internalASTExpression]
{
 internalASTExpression = null;
 InternalASTSymbolicLiteral symbolicLiteral = null;
}
 :
    internalASTExpression = overtureBracketedExpression
  | (overtureLetBeExpression)=> internalASTExpression = overtureLetBeExpression
  | (overtureLetExpression)=> internalASTExpression = overtureLetExpression
  | internalASTExpression = overtureDefExpression
  | internalASTExpression = overtureIfExpression
  | internalASTExpression = overtureCasesExpression
  | internalASTExpression = overtureQuantifiedExpression
  //Names
  | (overtureOldName)=> internalASTExpression = overtureOldName
  | (overtureName)=> internalASTExpression = overtureName
  | symbolicLiteral = overtureSymbolicLiteral
    {
     internalASTExpression = new InternalASTExpressionSymbolicLiteral(symbolicLiteral);
    }
  | internalASTExpression = overtureSelfExpression
  | internalASTExpression = overtureThreadidExpression
  | internalASTExpression = overtureUndefinedExpression
 ;

overtureBracketedExpression returns [InternalASTBracketedExpression internalASTBracketedExpression]
{
 internalASTBracketedExpression = null;
 InternalASTExpression expression = null;
}
 :
  lb:LBRACKET
  expression = overtureExpression
  rb:RBRACKET
   {
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
    internalASTBracketedExpression = new InternalASTBracketedExpression(keywordLeftBracket, expression, keywordRightBracket);
   }
 ;

//let expression = 'let', local definition, { ',', local definition },
//'in', expression ;
overtureLetExpression returns [InternalASTLetExpression internalASTLetExpression]
{
 internalASTLetExpression = null;
 List<InternalASTLetExpressionElement> definitions = new ArrayList<InternalASTLetExpressionElement>();
 InternalASTExpression expression = null;
 InternalASTLocalDefinition localDefinition = null;
 InternalASTLetExpressionElement currentElement = null;
}
 :
  let:LET
  localDefinition = overtureLocalDefinition
   {
    currentElement = new InternalASTLetExpressionElement(localDefinition);
```

```
      definitions.add(currentElement);
     }
    (
     co:COMMA
     localDefinition = overtureLocalDefinition
      {
       InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
       currentElement = new InternalASTLetExpressionElement(keywordComma, localDefinition);
       definitions.add(currentElement);
      }
    )*
    in:IN
    expression = overtureExpression
     {
      InternalASTKeyword keywordLet = new InternalASTKeyword(let, "Let");
      InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
      internalASTLetExpression = new InternalASTLetExpression(keywordLet, definitions, keywordIn,
          expression);
     }
  ;

overtureLetBeExpression returns [InternalASTLetBeExpression internalASTLetBeExpression]
{
 internalASTLetBeExpression = null;
 List<InternalASTLetExpressionElement> definitions = new ArrayList<
      InternalASTLetExpressionElement>();
 InternalASTBind bind = null;
 InternalASTKeyword keywordBe = null;
 InternalASTKeyword keywordSt = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   let:LET
   bind = overtureBind

   (
   be:BE
   st:ST
   expression1 = overtureExpression
    {
     keywordBe = new InternalASTKeyword(be, "Be");
     keywordSt = new InternalASTKeyword(st, "St");
    }
   )?
   in:IN
   expression2 = overtureExpression
    {
     InternalASTKeyword keywordLet = new InternalASTKeyword(let, "Let");
     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
     internalASTLetBeExpression = new InternalASTLetBeExpression(keywordLet, bind, keywordBe,
         keywordSt, expression1, keywordIn, expression2);
    }
  ;

overtureDefExpression returns [InternalASTDefExpression internalASTDefExpression]
{
 internalASTDefExpression = null;
 List<InternalASTDefExpressionElement> elements = new ArrayList<InternalASTDefExpressionElement
      >();
 InternalASTPatternBind currentPatternBind = null;
 InternalASTExpression currentexpression = null;
 InternalASTDefExpressionElement currentelement = null;
 InternalASTExpression expression = null;
}
  :
   def:DEF
   currentPatternBind = overturePatternBind
   eq1:EQUALSIGN
   currentexpression = overtureExpression
    {
     InternalASTKeyword keywordEqualSign = new InternalASTKeyword(eq1, "Equalsign");
     currentelement = new InternalASTDefExpressionElement(currentPatternBind, keywordEqualSign,
         currentexpression);
     elements.add(currentelement);
    }

   (options {greedy=true;}:
    sc:SEMICOLON
     {
      InternalASTKeyword keywordSemicolon = new InternalASTKeyword(sc, "Semicolon");
      currentelement.setKeywordSemicolon(keywordSemicolon);
      currentelement.setEndPositionFromNode(keywordSemicolon);
     }
```

```


    currentPatternBind = overturePatternBind
    eq:EQUALSIGN
    currentexpression = overtureExpression
     {
      InternalASTKeyword keywordEqualsign = new InternalASTKeyword(eq, "Equalsign");
      currentelement = new InternalASTDefExpressionElement(currentPatternBind, keywordEqualsign,
           currentexpression);
      elements.add(currentelement);
     }
   )*

   (
    selast:SEMICOLON
     {
       InternalASTKeyword keywordSemicolon = new InternalASTKeyword(selast, "Semicolon");
       currentelement.setKeywordSemicolon(keywordSemicolon);
       currentelement.setEndPositionFromNode(keywordSemicolon);
     }
   )?


   in:IN
   expression = overtureExpression
    {
     InternalASTKeyword keywordDef = new InternalASTKeyword(def, "Def");
     InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
     internalASTDefExpression = new InternalASTDefExpression(keywordDef, elements, keywordIn,
          expression);
    }
  ;

overtureIfExpression returns [InternalASTIfExpression internalASTIfExpression]
{
 internalASTIfExpression = null;
 InternalASTKeyword keywordIf = null;
 InternalASTExpression ifexpression = null;
 InternalASTKeyword keywordThen = null;
 InternalASTExpression thenexpression = null;
 List<InternalASTElseifExpression> elseifExpressions = new ArrayList<InternalASTElseifExpression
      >();
 InternalASTElseifExpression elseifexpression = null;
 InternalASTKeyword keywordElse = null;
 InternalASTExpression elseexpression = null;
}
  :
   kwif:IF
   ifexpression = overtureExpression

   kwthen:THEN
   thenexpression = overtureExpression

   (
    elseifexpression = overtureElseifExpression
     {
      elseifExpressions.add(elseifexpression);
     }
   )*

   kwelse:ELSE
   elseexpression = overtureExpression
    {
     keywordIf = new InternalASTKeyword(kwif, "If");
     keywordThen = new InternalASTKeyword(kwthen, "Then");
     keywordElse = new InternalASTKeyword(kwelse, "Else");
     internalASTIfExpression = new InternalASTIfExpression(keywordIf, ifexpression, keywordThen,
          thenexpression, elseifExpressions, keywordElse, elseexpression);
    }
  ;

overtureElseifExpression returns [InternalASTElseifExpression internalASTElseifExpression]
{
 internalASTElseifExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   elseif:ELSEIF
   expression1 = overtureExpression
   then:THEN
   expression2 = overtureExpression
    {
     InternalASTKeyword keywordElseif = new InternalASTKeyword(elseif, "Elseif");
```

```
        InternalASTKeyword keywordThen = new InternalASTKeyword(then, "Then");
        internalASTElseifExpression = new InternalASTElseifExpression(keywordElseif, expression1, keywordThen, expression2);
      }
  ;

overtureCasesExpression returns [InternalASTCasesExpression internalASTCasesExpression]
{
 internalASTCasesExpression = null;
 InternalASTExpression expression = null;
 InternalASTCasesExpressionAlternatives casealternatives = null;
 InternalASTKeyword keywordComma = null;
 InternalASTOthersExpression othersExpression = null;
}
  :
   ca:CASES

   expression = overtureExpression

   co:COLON

   casealternatives = overtureCasesExpressionAlternatives

   (
    com:COMMA
    othersExpression = overtureOthersExpression
     {
      keywordComma = new InternalASTKeyword(com, "Comma");
     }
   )?

   en:END
    {
      InternalASTKeyword keywordCases = new InternalASTKeyword(ca, "Cases");
      InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
      InternalASTKeyword keywordEnd = new InternalASTKeyword(en, "End");
      internalASTCasesExpression = new InternalASTCasesExpression(keywordCases, expression, keywordColon, casealternatives, keywordComma, othersExpression, keywordEnd);
    }
  ;

overtureCasesExpressionAlternatives returns [InternalASTCasesExpressionAlternatives internalASTCasesExpressionAlternatives]
{
 internalASTCasesExpressionAlternatives = null;
 List<InternalASTCasesExpressionAlternativesElement> alternatives = new ArrayList<InternalASTCasesExpressionAlternativesElement>();
 InternalASTCasesExpressionAlternativesElement element = null;
 InternalASTCasesExpressionAlternative alternative = null;
}
  :
   alternative = overtureCasesExpressionAlternative
    {
     element = new InternalASTCasesExpressionAlternativesElement(alternative);
     alternatives.add(element);
    }

   (options {greedy=true;}:
    co:COMMA
    alternative = overtureCasesExpressionAlternative
     {
      InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
      element = new InternalASTCasesExpressionAlternativesElement(keywordComma, alternative);
      alternatives.add(element);
     }
   )*

   {
    internalASTCasesExpressionAlternatives = new InternalASTCasesExpressionAlternatives(alternatives);
   }
  ;

overtureCasesExpressionAlternative returns [InternalASTCasesExpressionAlternative internalASTCasesExpressionAlternative]
{
 internalASTCasesExpressionAlternative = null;
 InternalASTPatternList patternlist = null;
 InternalASTExpression expression = null;
}
  :
   patternlist = overturePatternList
   la:LINEARROW
   expression = overtureExpression
```

```
      {
       InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
       internalASTCasesExpressionAlternative = new InternalASTCasesExpressionAlternative(patternlist
            , keywordLineArrow, expression);
      }
  ;

overtureOthersExpression returns [InternalASTOthersExpression internalASTOthersExpression]
{
 internalASTOthersExpression = null;
 InternalASTExpression expression = null;
}
  :
   ot:OTHERS
   la:LINEARROW
   expression = overtureExpression
      {
       InternalASTKeyword keywordOthers = new InternalASTKeyword(ot, "Others");
       InternalASTKeyword keywordLineArrow = new InternalASTKeyword(la, "Arrow");
       internalASTOthersExpression = new InternalASTOthersExpression(keywordOthers, keywordLineArrow
            , expression);
      }
  ;


overtureUnaryOperatorUnaryPlus returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   up:PLUS
      {
       internalASTUnaryOperator = new InternalASTUnaryOperatorUnaryPlus(
               new InternalASTKeyword(up, "Plus"));
      }
  ;

overtureUnaryOperatorUnaryMinus returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   mi:MINUS
      {
       internalASTUnaryOperator = new InternalASTUnaryOperatorUnaryMinus(
               new InternalASTKeyword(mi, "Minus"));
      }
  ;

overtureUnaryOperatorArithmeticAbs returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   abs:ABS
      {
       internalASTUnaryOperator = new InternalASTUnaryOperatorArithmeticAbs(
               new InternalASTKeyword(abs, "Abs"));
      }
  ;

overtureUnaryOperatorFloor returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   floor:FLOOR
      {
       internalASTUnaryOperator = new InternalASTUnaryOperatorFloor(
               new InternalASTKeyword(floor, "Floor"));
      }
  ;

overtureUnaryOperatorNot returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   not:NOT
      {
       internalASTUnaryOperator = new InternalASTUnaryOperatorNot(
               new InternalASTKeyword(not, "Not"));
      }
  ;
```

```

overtureUnaryOperatorSetCardinality returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   card:CARD
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSetCardinality(
              new InternalASTKeyword(card, "Card"));
     }
  ;

overtureUnaryOperatorFinitePowerSet returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   pwr:POWER
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorFinitePowerSet(
              new InternalASTKeyword(pwr, "Power"));
     }
  ;

overtureUnaryOperatorDistributedSetUnion returns[InternalASTUnaryOperator
     internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   du:DUNION
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSetUnion(
              new InternalASTKeyword(du, "DUnion"));
     }
  ;

overtureUnaryOperatorDistributedSetIntersection returns[InternalASTUnaryOperator
     internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   di:DINTER
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSetIntersection(
              new InternalASTKeyword(di, "DInter"));
     }
  ;

overtureUnaryOperatorSequenceHead returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   hd:HD
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceHead(
              new InternalASTKeyword(hd, "Hd"));
     }
  ;

overtureUnaryOperatorSequenceTail returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   tl:TL
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceTail(
              new InternalASTKeyword(tl, "Tl"));
     }
  ;

overtureUnaryOperatorSequenceLength returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   len:LEN
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceLength(
              new InternalASTKeyword(len, "Len"));
```

```
    }
  ;

overtureUnaryOperatorSequenceElements returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   el:ELEMS
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceElements(
              new InternalASTKeyword(el, "Elems"));
     }
  ;

overtureUnaryOperatorSequenceIndices returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   inds:INDS
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorSequenceIndices(
              new InternalASTKeyword(inds, "Inds"));
     }
  ;

overtureUnaryOperatorDistributedSequenceConcatenation returns[InternalASTUnaryOperator
      internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   conc:CONC
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedSequenceConcatenation(
              new InternalASTKeyword(conc, "Conc"));
     }
  ;

overtureUnaryOperatorMapDomain returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   dom:DOM
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorMapDomain(
              new InternalASTKeyword(dom, "Dom"));
     }
  ;

overtureUnaryOperatorMapRange returns[InternalASTUnaryOperator internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   rng:RNG
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorMapRange(
              new InternalASTKeyword(rng, "Rng"));
     }
  ;

overtureUnaryOperatorDistributedMapMerge returns[InternalASTUnaryOperator
      internalASTUnaryOperator]
{
 internalASTUnaryOperator = null;
}
  :
   mer:MERGE
     {
      internalASTUnaryOperator = new InternalASTUnaryOperatorDistributedMapMerge(
              new InternalASTKeyword(mer, "Merge"));
     }
  ;

overtureMapInverse returns[InternalASTMapInverse internalASTMapInverse]
{
 internalASTMapInverse = null;
 InternalASTExpression expression = null;
}
  :
   in:INVERSE
```

```
    expression = overtureExpression
      {
       InternalASTKeyword keywordInverse = new InternalASTKeyword(in, "Inverse");
       internalASTMapInverse = new InternalASTMapInverse(keywordInverse, expression);
      }
  ;

overtureBinaryOperatorArithmeticPlus returns [InternalASTBinaryOperatorArithmeticPlus
     internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:PLUS
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Plus");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticPlus(operator);
     }
  ;

overtureBinaryOperatorArithmeticMinus returns [InternalASTBinaryOperatorArithmeticMinus
     internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:MINUS
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Minus");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMinus(operator);
     }
  ;

overtureBinaryOperatorArithmeticMultiplication returns [
     InternalASTBinaryOperatorArithmeticMultiplication internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:ASTERIX
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Asterix");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMultiplication(operator);
     }
  ;

overtureBinaryOperatorArithmeticDivide returns [InternalASTBinaryOperatorArithmeticDivide
     internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:SLASH
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Slash");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticDivide(operator);
     }
  ;

overtureBinaryOperatorArithmeticIntegerDivision returns [
     InternalASTBinaryOperatorArithmeticIntegerDivision internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:DIV
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Div");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticIntegerDivision(operator);
     }
  ;

overtureBinaryOperatorArithmeticRem returns [InternalASTBinaryOperatorArithmeticRem
     internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:REM
     {
      InternalASTKeyword operator = new InternalASTKeyword(op, "Rem");
      internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticRem(operator);
     }
  ;
```

```
overtureBinaryOperatorArithmeticMod returns [InternalASTBinaryOperatorArithmeticMod
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:MOD
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Mod");
     internalASTBinaryOperator = new InternalASTBinaryOperatorArithmeticMod(operator);
    }
 ;

overtureBinaryOperatorLessThan returns [InternalASTBinaryOperatorLessThan
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:LESSTHAN
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThan");
     internalASTBinaryOperator = new InternalASTBinaryOperatorLessThan(operator);
    }
 ;

overtureBinaryOperatorLessThanOrEqual returns [InternalASTBinaryOperatorLessThanOrEqual
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:LESSTHANEQUAL
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanEqual");
     internalASTBinaryOperator = new InternalASTBinaryOperatorLessThanOrEqual(operator);
    }
 ;

overtureBinaryOperatorGreaterThan returns [InternalASTBinaryOperatorGreaterThan
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:GREATERTHAN
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "GreaterThan");
     internalASTBinaryOperator = new InternalASTBinaryOperatorGreaterThan(operator);
    }
 ;

overtureBinaryOperatorGreaterThanOrEqual returns [InternalASTBinaryOperatorGreaterThanOrEqual
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:GREATERTHANEQUAL
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "GreaterThanEqual");
     internalASTBinaryOperator = new InternalASTBinaryOperatorGreaterThanOrEqual(operator);
    }
 ;

overtureBinaryOperatorEqual returns [InternalASTBinaryOperatorEqual internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:EQUALSIGN
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Equalsign");
     internalASTBinaryOperator = new InternalASTBinaryOperatorEqual(operator);
    }
 ;

overtureBinaryOperatorNotEqual returns [InternalASTBinaryOperatorNotEqual
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
```

```
  op:NOTEQUAL
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "NotEqual");
     internalASTBinaryOperator = new InternalASTBinaryOperatorNotEqual(operator);
    }
  ;

overtureBinaryOperatorApprox returns [InternalASTBinaryOperatorApprox internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:APPROX
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Approx");
     internalASTBinaryOperator = new InternalASTBinaryOperatorApprox(operator);
    }
  ;

overtureBinaryOperatorOr returns [InternalASTBinaryOperatorOr internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:OR
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Or");
     internalASTBinaryOperator = new InternalASTBinaryOperatorOr(operator);
    }
  ;

overtureBinaryOperatorAnd returns [InternalASTBinaryOperatorAnd internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:AND
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "And");
     internalASTBinaryOperator = new InternalASTBinaryOperatorAnd(operator);
    }
  ;

overtureBinaryOperatorImply returns [InternalASTBinaryOperatorImply internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:IMPLY
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Imply");
     internalASTBinaryOperator = new InternalASTBinaryOperatorImply(operator);
    }
  ;

overtureBinaryOperatorLogicalEquivalence returns [InternalASTBinaryOperatorLogicalEquivalence internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:LOGICALEQUIVALENCE
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "LogicalEquivalence");
     internalASTBinaryOperator = new InternalASTBinaryOperatorLogicalEquivalence(operator);
    }
  ;

overtureBinaryOperatorInSet returns [InternalASTBinaryOperatorInSet internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   in:IN
   set:SET
    {
     InternalASTKeyword operatorIn = new InternalASTKeyword(in, "In");
     InternalASTKeyword operatorSet = new InternalASTKeyword(set, "Set");
     internalASTBinaryOperator = new InternalASTBinaryOperatorInSet(operatorIn, operatorSet);
    }
  ;

overtureBinaryOperatorNotInSet returns [InternalASTBinaryOperatorNotInSet internalASTBinaryOperator]
```

```
{
 internalASTBinaryOperator = null;
}
  :
   not:NOT
   in:IN
   set:SET
    {
     InternalASTKeyword operatorNot = new InternalASTKeyword(not, "Not");
     InternalASTKeyword operatorIn = new InternalASTKeyword(in, "In");
     InternalASTKeyword operatorSet = new InternalASTKeyword(set, "Set");
     internalASTBinaryOperator = new InternalASTBinaryOperatorNotInSet(operatorNot, operatorIn, operatorSet);
    }
  ;

overtureBinaryOperatorSubset returns [InternalASTBinaryOperatorSubset internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:SUBSET
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Subset");
     internalASTBinaryOperator = new InternalASTBinaryOperatorSubset(operator);
    }
  ;

overtureBinaryOperatorProperSubset returns [InternalASTBinaryOperatorProperSubset internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:PSUBSET
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "PSubset");
     internalASTBinaryOperator = new InternalASTBinaryOperatorProperSubset(operator);
    }
  ;

overtureBinaryOperatorSetUnion returns [InternalASTBinaryOperatorSetUnion internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:UNION
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Union");
     internalASTBinaryOperator = new InternalASTBinaryOperatorSetUnion(operator);
    }
  ;

overtureBinaryOperatorSetDifference returns [InternalASTBinaryOperatorSetDifference internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:BACKSLASH
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Backslash");
     internalASTBinaryOperator = new InternalASTBinaryOperatorSetDifference(operator);
    }
  ;

overtureBinaryOperatorSetIntersection returns [InternalASTBinaryOperatorSetIntersection internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:INTER
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Inter");
     internalASTBinaryOperator = new InternalASTBinaryOperatorSetIntersection(operator);
    }
  ;

overtureBinaryOperatorSequenceConcatenate returns [InternalASTBinaryOperatorSequenceConcatenate internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
```

```
}
  :
   op:HAT
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "Hat");
     internalASTBinaryOperator = new InternalASTBinaryOperatorSequenceConcatenate(operator);
    }
  ;

overtureBinaryOperatorMapOrSequenceModify returns [InternalASTBinaryOperatorMapOrSequenceModify
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:DOUBLEPLUS
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "DoublePlus");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapOrSequenceModify(operator);
    }
  ;

overtureBinaryOperatorMapMerge returns [InternalASTBinaryOperatorMapMerge
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:MUNION
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "MUnion");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapMerge(operator);
    }
  ;

overtureBinaryOperatorMapDomainRestrictTo returns [InternalASTBinaryOperatorMapDomainRestrictTo
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:LESSTHANCOLON
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanColon");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapDomainRestrictTo(operator);
    }
  ;

overtureBinaryOperatorMapDomainRestrictBy returns [InternalASTBinaryOperatorMapDomainRestrictBy
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:LESSTHANLINECOLON
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "LessThanLineColon");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapDomainRestrictBy(operator);
    }
  ;

overtureBinaryOperatorMapRangeRestrictTo returns [InternalASTBinaryOperatorMapRangeRestrictTo
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:COLONGREATERTHAN
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "ColonGreaterThan");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapRangeRestrictTo(operator);
    }
  ;

overtureBinaryOperatorMapRangeRestrictBy returns [InternalASTBinaryOperatorMapRangeRestrictBy
      internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
  :
   op:COLONLINEGREATERTHAN
    {
     InternalASTKeyword operator = new InternalASTKeyword(op, "ColonLineGreaterThan");
     internalASTBinaryOperator = new InternalASTBinaryOperatorMapRangeRestrictBy(operator);
```

```
  }
 ;

overtureBinaryOperatorComposition returns [InternalASTBinaryOperatorComposition
     internalASTBinaryOperator]
{
 internalASTBinaryOperator = null;
}
 :
  op:COMP
   {
    InternalASTKeyword operator = new InternalASTKeyword(op, "Comp");
    internalASTBinaryOperator = new InternalASTBinaryOperatorComposition(operator);
   }
 ;

overtureBinaryOperatorIterate returns [InternalASTBinaryOperatorIterate internalASTBinaryOperator
     ]
{
 internalASTBinaryOperator = null;
}
 :
  op:ITERATE
   {
    InternalASTKeyword operator = new InternalASTKeyword(op, "Iterate");
    internalASTBinaryOperator = new InternalASTBinaryOperatorIterate(operator);
   }
 ;


// QUANTIFIED EXPRESSION

//quantified expression = all expression
//          | exists expression
//          | exists unique expression ;
overtureQuantifiedExpression returns [InternalASTQuantifiedExpression
     internalASTQuantifiedExpression]
{
 internalASTQuantifiedExpression = null;
}
 :
      (
   internalASTQuantifiedExpression = overtureAllExpression
   | internalASTQuantifiedExpression = overtureExistsExpression
   | internalASTQuantifiedExpression = overtureExistsUniqueExpression
  )
 ;

//all expression = 'forall', bind list, '&', expression ;
overtureAllExpression returns [InternalASTAllExpression internalASTAllExpression]
{
 internalASTAllExpression = null;
 InternalASTBindList bindList = null;
 InternalASTExpression expression = null;
}
 :
  kwforall:FORALL
  bindList = overtureBindList
  kwand:ANDSIGN
  expression = overtureExpression
  {
   InternalASTKeyword keywordForall = new InternalASTKeyword(kwforall, "Forall");
   InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
   internalASTAllExpression = new InternalASTAllExpression(keywordForall, bindList, keywordAnd,
        expression);
  }

 ;

//exists expression = 'exists', bind list, '&', expression ;
overtureExistsExpression returns [InternalASTExistsExpression internalASTExistsExpression]
{
 internalASTExistsExpression = null;
 InternalASTBindList bindList = null;
 InternalASTExpression expression = null;
}
 :
  kwexists:EXISTS
  bindList = overtureBindList
  kwand:ANDSIGN
  expression = overtureExpression
  {
   InternalASTKeyword keywordExists = new InternalASTKeyword(kwexists, "Exists");
   InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
```

```
    internalASTExistsExpression = new InternalASTExistsExpression(keywordExists, bindList, keywordAnd, expression);
  }

  ;

//exists unique expression = 'exists1', bind, '&', expression ;
overtureExistsUniqueExpression returns [InternalASTExistsUniqueExpression internalASTExistsUniqueExpression]
{
 internalASTExistsUniqueExpression = null;
 InternalASTBind bind = null;
 InternalASTExpression expression = null;
 }
 :
  kwexists1:EXISTS1
  bind = overtureBind
  kwand:ANDSIGN
  expression = overtureExpression
  {
   InternalASTKeyword keywordExists1 = new InternalASTKeyword(kwexists1, "Exists1");
   InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
   internalASTExistsUniqueExpression = new InternalASTExistsUniqueExpression(keywordExists1, bind, keywordAnd, expression);
  }
  ;

//iota expression = 'iota', bind, '&', expression ;
overtureIotaExpression returns [InternalASTIotaExpression internalASTIotaExpression]
{
 internalASTIotaExpression = null;
 InternalASTBind bind = null;
 InternalASTExpression expression = null;
}
 :
  kwiota:IOTA
  bind = overtureBind
  kwand:ANDSIGN
  expression = overtureExpression
  {
   InternalASTKeyword keywordIota = new InternalASTKeyword(kwiota, "Iota");
   InternalASTKeyword keywordAnd = new InternalASTKeyword(kwand, "Andsign");
   internalASTIotaExpression = new InternalASTIotaExpression(keywordIota, bind, keywordAnd, expression);
  }
  ;

//set enumeration = '{', [ expression list ], '}' ;
overtureSetEnumeration returns [InternalASTSetEnumeration internalASTSetEnumeration]
{
 internalASTSetEnumeration = null;
 InternalASTExpressionList internalASTExpressionList = null;
}
 :
  lb:LBRACE
  (
     internalASTExpressionList = overtureExpressionList
  )?
  rb:RBRACE
  {
   InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb,"LeftBrace");
   InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb,"RightBrace");
   internalASTSetEnumeration = new InternalASTSetEnumeration(keywordLeftBrace, internalASTExpressionList, keywordRightBrace);
  }
 ;

//set comprehension = '{', expression, '|', bind list,
//        [ '&', expression ], '}' ;
overtureSetComprehension returns [InternalASTSetComprehension internalASTSetComprehension]
{
 internalASTSetComprehension = null;
 InternalASTExpression expression1 = null;
 InternalASTBindList bindList = null;
 InternalASTKeyword keywordAndsign = null;
 InternalASTExpression expression2 = null;
}
 :
  lb:LBRACE
  expression1 = overtureExpression
  vb:VBAR
  bindList = overtureBindList
  (
   an:ANDSIGN
```

```
      expression2 = overtureExpression
       {
        keywordAndsign = new InternalASTKeyword(an, "Andsign");
       }
    )?
    rb:RBRACE
    {
     InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb,"LeftBrace");
     InternalASTKeyword keywordVBar = new InternalASTKeyword(vb, "VBar");
     InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb,"RightBrace");
     internalASTSetComprehension = new InternalASTSetComprehension(keywordLeftBrace, expression1,
          keywordVBar, bindList, keywordAndsign, expression2, keywordRightBrace);
    }
  ;

//set range expression = '{', expression, ',', '...', ',',
//          expression, '}' ;
overtureSetRangeExpression returns [InternalASTSetRangeExpression internalASTSetRangeExpression]
{
 internalASTSetRangeExpression = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   lb:LBRACE
   expression1 = overtureExpression
   co1:COMMA
   dots:DOTS
   co2:COMMA
   expression2 = overtureExpression
   rb:RBRACE
   {
    InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lb,"LeftBrace");
    InternalASTKeyword keywordComma1 = new InternalASTKeyword(co1, "Comma1");
    InternalASTKeyword keywordDots = new InternalASTKeyword(dots, "Dots");
    InternalASTKeyword keywordComma2 = new InternalASTKeyword(co2, "Comma2");
    InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rb,"RightBrace");
    internalASTSetRangeExpression = new InternalASTSetRangeExpression(keywordLeftBrace,
         expression1, keywordComma1, keywordDots, keywordComma2, expression2, keywordRightBrace);
   }
  ;

//sequence enumeration = '[', [ expression list ], ']' ;
overtureSequenceEnumeration returns [InternalASTSequenceEnumeration
     internalASTSequenceEnumeration]
{
 internalASTSequenceEnumeration = null;
 InternalASTExpressionList internalASTExpressionList = null;
}
  :
   lb:LBRACK
   (
       internalASTExpressionList = overtureExpressionList
   )?
   rb:RBRACK
   {
    InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lb,"LeftBrack");
    InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rb,"RightBrack");
    internalASTSequenceEnumeration = new InternalASTSequenceEnumeration(keywordLeftBrack,
         internalASTExpressionList, keywordRightBrack);
   }
  ;

//sequence comprehension = '[', expression, '|', set bind, [ '&', expression ], ']' ;
overtureSequenceComprehension returns [InternalASTSequenceComprehension
     internalASTSequenceComprehension]
{
 internalASTSequenceComprehension = null;
 InternalASTExpression expression1 = null;
 InternalASTSetBind setBind = null;
 InternalASTKeyword keywordAndsign = null;
 InternalASTExpression expression2 = null;
}
  :
   lb:LBRACK
   expression1 = overtureExpression
   vb:VBAR
   setBind = overtureSetBind
   (
    an:ANDSIGN
    expression2 = overtureExpression
     {
      keywordAndsign = new InternalASTKeyword(an, "Andsign");
     }
```

```
    )?
    rb:RBRACK
    {
     InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lb,"LeftBrack");
     InternalASTKeyword keywordVBar = new InternalASTKeyword(vb, "VBar");
     InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rb,"RightBrack");
     internalASTSequenceComprehension = new InternalASTSequenceComprehension(keywordLeftBrack, expression1, keywordVBar, setBind, keywordAndsign, expression2, keywordRightBrack);
    }
  ;

//subsequence = expression, ‘(’, expression, ‘,’, ‘...’, ‘,’, expression, ‘)’ ;
overtureSubsequence returns [InternalASTSubsequence internalASTSubsequence]
{
 internalASTSubsequence = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
 InternalASTExpression expression3 = null;
}
  :
    expression1 = overtureExpressionPre61
    lb:LBRACKET
    expression2 = overtureExpression
    co1:COMMA
    dots:DOTS
    co2:COMMA
    expression3 = overtureExpression
    rb:RBRACKET
    {
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
     InternalASTKeyword keywordComma1 = new InternalASTKeyword(co1, "Comma1");
     InternalASTKeyword keywordDots = new InternalASTKeyword(dots, "Dots");
     InternalASTKeyword keywordComma2 = new InternalASTKeyword(co2, "Comma2");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
     internalASTSubsequence = new InternalASTSubsequence(expression1, keywordLeftBracket, expression2, keywordComma1, keywordDots, keywordComma2, expression3, keywordRightBracket);
    }
  ;

//map enumeration = '{', maplet, { ',', maplet }, '}'
//        | '{', '|->', '}' ;
overtureMapEnumeration returns [InternalASTMapEnumeration internalASTMapEnumeration]
{
 internalASTMapEnumeration = null;
 List<InternalASTMapEnumerationMapletElement> mapEnumerationMapletElements = new ArrayList<InternalASTMapEnumerationMapletElement>();
 InternalASTMapEnumerationMaplet mapEnumerationMaplet = null;
 InternalASTMapEnumerationMapletElement currentElement = null;
 InternalASTMapEnumerationArrow mapEnumerationArrow = null;
 InternalASTMaplet maplet = null;
}
  :
    (
      (
       lbr:LBRACE
       maplet = overtureMaplet
        {
         currentElement = new InternalASTMapEnumerationMapletElement(maplet);
         mapEnumerationMapletElements.add(currentElement);
        }
       (
        co:COMMA
        maplet = overtureMaplet
         {
          InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
          currentElement = new InternalASTMapEnumerationMapletElement(keywordComma, maplet);
          mapEnumerationMapletElements.add(currentElement);
         }
       )*
       rbr:RBRACE
        {
         InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lbr, "LeftBrace");
         InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rbr, "RightBrace");
         internalASTMapEnumeration = new InternalASTMapEnumerationMaplet(keywordLeftBrace, mapEnumerationMapletElements, keywordRightBrace);
        }
      )
    |
     (
      lbr2:LBRACE
        vba:VBARARROW
      rbr2:RBRACE
        {
```

```
      InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(lbr2, "LeftBrace");
      InternalASTKeyword keywordRightBrace = new InternalASTKeyword(rbr2, "RightBrace");
        InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(vba, "VBarArrow");
      internalASTMapEnumeration = new InternalASTMapEnumerationArrow(keywordLeftBrace,
           keywordVBarArrow, keywordRightBrace);
    }
   )
  )
 ;

//maplet = expression, '|->', expression ;s
overtureMaplet returns [InternalASTMaplet internalASTMaplet]
{
 internalASTMaplet = null;
 InternalASTExpression internalASTExpression1 = null;
 InternalASTExpression internalASTExpression2 = null;
}
  :
  internalASTExpression1 = overtureExpression
  kwvbararrow:VBARARROW
  internalASTExpression2 = overtureExpression
  {
   InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
    internalASTMaplet = new InternalASTMaplet(internalASTExpression1, keywordVBarArrow,
         internalASTExpression2);
  }
 ;

//map comprehension = '{', maplet, '|', bind list,
//      [ '&', expression ], '}' ;
overtureMapComprehension returns [InternalASTMapComprehension internalASTMapComprehension]
{
 internalASTMapComprehension = null;
 InternalASTMaplet internalASTMaplet = null;
 InternalASTBindList internalASTBindList = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTKeyword keywordAndsign = null;
}
  :
  kwlb:LBRACE
  internalASTMaplet = overtureMaplet
  kwvbar:VBAR
  internalASTBindList = overtureBindList
  (
    kwandsign:ANDSIGN
    internalASTExpression = overtureExpression
     {
      keywordAndsign = new InternalASTKeyword(kwandsign,"Andsign");
     }
  )?
  kwrb:RBRACE
  {
   InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(kwlb,"LeftBrace");
    InternalASTKeyword keywordVBar = new InternalASTKeyword(kwvbar,"VBar");
    InternalASTKeyword keywordRightBrace = new InternalASTKeyword(kwrb,"RightBrace");
    internalASTMapComprehension = new InternalASTMapComprehension(keywordLeftBrace,
         internalASTMaplet, keywordVBar, internalASTBindList, keywordAndsign,
         internalASTExpression, keywordRightBrace);
  }
 ;

//tuple constructor = 'mk_', '(', expression, expression list, ')' ;
overtureTupleConstructor returns [InternalASTTupleConstructor internalASTTupleConstructor]
{
 internalASTTupleConstructor = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTExpressionList internalASTExpressionList = null;
}
  :
  kwmk:MAKE
  kwlb:LBRACKET
      internalASTExpression = overtureExpression
  internalASTExpressionList = overtureExpressionList
  kwrb:RBRACKET
   {
   InternalASTKeyword keywordMake = new InternalASTKeyword(kwmk,"Mk");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
    internalASTTupleConstructor = new InternalASTTupleConstructor(keywordMake, keywordLeftBracket
         , internalASTExpression, internalASTExpressionList, keywordRightBracket);
  }
 ;

//record constructor = 'mk_', name, '(', [ expression list ], ')' ;
```

```
overtureRecordConstructor returns [InternalASTRecordConstructor internalASTRecordConstructor]
{
 internalASTRecordConstructor = null;
 InternalASTName internalASTName = null;
 InternalASTExpressionList internalASTExpressionList = null;
}
   :
   kwmk:MAKE
   internalASTName = overtureName
   kwlb:LBRACKET
   (
        internalASTExpressionList = overtureExpressionList
   )?
   kwrb:RBRACKET
    {
    InternalASTKeyword keywordMake = new InternalASTKeyword(kwmk,"Mk");
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTRecordConstructor = new InternalASTRecordConstructor(keywordMake, internalASTName,
           keywordLeftBracket, internalASTExpressionList, keywordRightBracket);
   }
  ;

//record modifier = 'mu', '(', expression, ',',
//          record modification,
//          { ',', record modification }, ')' ;
overtureRecordModifier returns [InternalASTRecordModifier internalASTRecordModifier]
{
 internalASTRecordModifier = null;
 InternalASTRecordModifierElement internalASTRecordModifierElement = null;
 List<InternalASTRecordModifierElement> recordModifierElements = new ArrayList<
      InternalASTRecordModifierElement>();
 InternalASTExpression internalASTExpression = null;
 InternalASTRecordModification internalASTRecordModification = null;
}
   : kwmu:MU
   kwlb:LBRACKET
      internalASTExpression = overtureExpression
      kwcomma:COMMA
      internalASTRecordModification = overtureRecordModification
      {
       InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
     internalASTRecordModifierElement = new InternalASTRecordModifierElement(keywordComma,
          internalASTRecordModification);
     recordModifierElements.add(internalASTRecordModifierElement);
    }
      (
       kwcomma2:COMMA
       internalASTRecordModification = overtureRecordModification
       {
        InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma2,"Comma");
      internalASTRecordModifierElement = new InternalASTRecordModifierElement(keywordComma,
           internalASTRecordModification);
      recordModifierElements.add(internalASTRecordModifierElement);
     }
      )*
      kwrb:RBRACKET
      {
       InternalASTKeyword keywordMu = new InternalASTKeyword(kwmu,"Mu");
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTRecordModifier = new InternalASTRecordModifier(keywordMu, keywordLeftBracket,
          internalASTExpression, recordModifierElements, keywordRightBracket);
   }
  ;

//record modification = identifier, '|->', expression ;
overtureRecordModification returns [InternalASTRecordModification internalASTRecordModification]
{
 internalASTRecordModification = null;
 InternalASTExpression internalASTExpression = null;
}
   :
   kwid:IDENTIFIER
   kwvbararrow:VBARARROW
       internalASTExpression = overtureExpression
    {
    InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(kwid);
    InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
     internalASTRecordModification = new InternalASTRecordModification(internalASTIdentifier,
          keywordVBarArrow, internalASTExpression);
   }
  ;
```

```
//apply = expression, '(', [ expression list ], ')' ;
overtureApply returns [InternalASTApply internalASTApply]
{
 internalASTApply = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTExpressionList internalASTExpressionList = null;
}
    :
    (
     (overtureSubsequence)=> internalASTExpression = overtureSubsequence
     | (overtureFunctionTypeInstantiation) => internalASTExpression =
          overtureFunctionTypeInstantiation
     | internalASTExpression = overtureExpressionPre61
    )
    kwlb:LBRACKET
    (
      internalASTExpressionList = overtureExpressionList
     )?
     kwrb:RBRACKET
    {
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
      internalASTApply = new InternalASTApply(internalASTExpression, keywordLeftBracket,
           internalASTExpressionList, keywordRightBracket);
    }
  ;

//field select = expression, '.', identifier ;
overtureFieldSelect returns [InternalASTFieldSelect internalASTFieldSelect]
{
 internalASTFieldSelect = null;
 InternalASTExpression internalASTExpression = null;
}
    :
    (
     (overtureSubsequence)=> internalASTExpression = overtureSubsequence
     | (overtureApply)=> internalASTExpression = overtureApply
     | (overtureFunctionTypeInstantiation)=> internalASTExpression =
          overtureFunctionTypeInstantiation
     | internalASTExpression = overtureExpressionPre61
    )
    kwdot:DOT
    id:IDENTIFIER
    {
     InternalASTKeyword keywordDot = new InternalASTKeyword(kwdot,"Dot");
     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
      internalASTFieldSelect = new InternalASTFieldSelect(internalASTExpression, keywordDot,
           identifier);
    }
  ;

//tuple select = expression, '.#', numeral ;
overtureTupleSelect returns [InternalASTTupleSelect internalASTTupleSelect]
{
 internalASTTupleSelect = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTNumeral internalASTNumeral = null;
 InternalASTSymbolicLiteral symbolicLiteral = null;
}
    :
    (
      (overtureQuantifiedExpression)=> internalASTExpression = overtureQuantifiedExpression
     | internalASTExpression = overtureIotaExpression
     | (overtureSetEnumeration)=> internalASTExpression = overtureSetEnumeration
     | (overtureSetComprehension)=> internalASTExpression = overtureSetComprehension
     | (overtureSetRangeExpression)=> internalASTExpression = overtureSetRangeExpression
     | (overtureSequenceEnumeration)=> internalASTExpression = overtureSequenceEnumeration
     | (overtureSequenceComprehension)=> internalASTExpression = overtureSequenceComprehension
     | (overtureMapEnumeration)=> internalASTExpression = overtureMapEnumeration
     | (overtureMapComprehension)=> internalASTExpression = overtureMapComprehension
     | internalASTExpression = overtureTupleConstructor
     | internalASTExpression = overtureRecordConstructor
     | internalASTExpression = overtureRecordModifier
     | internalASTExpression = overtureLambdaExpression
     | internalASTExpression = overtureNewExpression
     | internalASTExpression = overtureGeneralIsExpression
     | internalASTExpression = overtureIsofbaseclassExpression
     | internalASTExpression = overtureIsofclassExpression
     | internalASTExpression = overtureSamebaseclassExpression
     | internalASTExpression = overtureSameclassExpression
     | internalASTExpression = overtureActExpression
     | internalASTExpression = overtureFinExpression
     | internalASTExpression = overtureActiveExpression
     | internalASTExpression = overtureReqExpression
```

```
      | internalASTExpression = overtureWaitingExpression
       {
        internalASTExpression = new InternalASTExpressionSymbolicLiteral(symbolicLiteral);
       }
      | internalASTExpression = overtureExpressionPre21
    )
    kwdotsharp:DOTSHARP
    internalASTNumeral = overtureNumeral
    {
     InternalASTKeyword keywordDotSharp = new InternalASTKeyword(kwdotsharp,"DotSharp");
     internalASTTupleSelect = new InternalASTTupleSelect(internalASTExpression, keywordDotSharp, internalASTNumeral);
    }
  ;

//function type instantiation = name, '[', type, { ',', type }, ']' ;
overtureFunctionTypeInstantiation returns [InternalASTFunctionTypeInstantiation internalASTFunctionTypeInstantiation]
{
 internalASTFunctionTypeInstantiation = null;
 InternalASTFunctionTypeInstantiationElement internalASTFunctionTypeInstantiationElement = null;
 List<InternalASTFunctionTypeInstantiationElement> functionTypeInstantiationElements = new ArrayList<InternalASTFunctionTypeInstantiationElement>();
 InternalASTName internalASTName = null;
 InternalASTType internalASTType = null;
}
    :
    internalASTName = overtureName
        kwlb:LBRACK
        internalASTType = overtureType
        {
         internalASTFunctionTypeInstantiationElement = new InternalASTFunctionTypeInstantiationElement(internalASTType);
         functionTypeInstantiationElements.add(internalASTFunctionTypeInstantiationElement);
        }
        (
         kwcomma:COMMA
         internalASTType = overtureType
         {
          InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
        internalASTFunctionTypeInstantiationElement = new InternalASTFunctionTypeInstantiationElement(keywordComma, internalASTType);
          functionTypeInstantiationElements.add(internalASTFunctionTypeInstantiationElement);
         }
        )*
        kwrb:RBRACK
        {
         InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb,"LeftBrack");
         InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb,"RightBrack");
       internalASTFunctionTypeInstantiation = new InternalASTFunctionTypeInstantiation(internalASTName, keywordLeftBrack, functionTypeInstantiationElements, keywordRightBrack);
    }
  ;

//lambda expression = 'lambda', type bind list, '&', expression ;
overtureLambdaExpression returns [InternalASTLambdaExpression internalASTLambdaExpression]
{
 internalASTLambdaExpression = null;
 InternalASTTypeBindList internalASTTypeBindList = null;
 InternalASTExpression internalASTExpression = null;
}
    :
    kwlambda:LAMBDA
        internalASTTypeBindList = overtureTypeBindList
        kwandsign:ANDSIGN
        internalASTExpression = overtureExpression
      {
        InternalASTKeyword keywordLambda = new InternalASTKeyword(kwlambda,"Lambda");
        InternalASTKeyword keywordAndsign = new InternalASTKeyword(kwandsign,"Andsign");
      internalASTLambdaExpression = new InternalASTLambdaExpression(keywordLambda, internalASTTypeBindList, keywordAndsign, internalASTExpression);
    }
  ;

//new expression = 'new', name, '(', [ instvarinit expression ], ')' ;
overtureNewExpression returns [InternalASTNewExpression internalASTNewExpression]
{
 internalASTNewExpression = null;
 InternalASTName internalASTName = null;
 InternalASTInstvarinitExpression internalASTInstvarinitExpression = null;
}
    :
    kwnew:NEW
```

```
      internalASTName = overtureName
      kwlb:LBRACKET
      (
       internalASTInstvarinitExpression = overtureInstvarinitExpression
      )?
      kwrb:RBRACKET
      {
       InternalASTKeyword keywordNew = new InternalASTKeyword(kwnew,"New");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTNewExpression = new InternalASTNewExpression(keywordNew, internalASTName,
          keywordLeftBracket, internalASTInstvarinitExpression, keywordRightBracket);
  }
 ;

//instvarinit expression = name, '|->', expression ;
overtureInstvarinitExpression returns [InternalASTInstvarinitExpression
     internalASTInstvarinitExpression]
{
 internalASTInstvarinitExpression = null;
 InternalASTName internalASTName = null;
 InternalASTExpression internalASTExpression = null;
}
  :
  internalASTName = overtureName
      kwvbararrow:VBARARROW
      internalASTExpression = overtureExpression
      {
       InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
     internalASTInstvarinitExpression = new InternalASTInstvarinitExpression(internalASTName,
          keywordVBarArrow, internalASTExpression);
  }
 ;

//self expression = 'self' ;
overtureSelfExpression returns [InternalASTSelfExpression internalASTSelfExpression]
{
 internalASTSelfExpression = null;
}
  :
      kwself:SELF
      {
       InternalASTKeyword keywordSelf = new InternalASTKeyword(kwself,"Self");
     internalASTSelfExpression = new InternalASTSelfExpression(keywordSelf);
  }
 ;

//threadid expression = 'threadid' ;
overtureThreadidExpression returns [InternalASTThreadidExpression internalASTThreadidExpression]
{
 internalASTThreadidExpression = null;
}
  :
      kwthreadid:THREADID
      {
       InternalASTKeyword keywordThreadid = new InternalASTKeyword(kwthreadid,"Threadid");
     internalASTThreadidExpression = new InternalASTThreadidExpression(keywordThreadid);
  }
 ;

//general is expression = is expression
//        | type judgement ;
overtureGeneralIsExpression returns [InternalASTGeneralIsExpression
     internalASTGeneralIsExpression]
{
 internalASTGeneralIsExpression = null;
}
  :
     (
      (overtureIsExpression)=> internalASTGeneralIsExpression = overtureIsExpression
        | internalASTGeneralIsExpression = overtureTypeJudgement
     )
 ;

//is expression = 'is_', name, '(', expression, ')'
//     | is basic type, '(', expression, ')' ;
overtureIsExpression returns [InternalASTIsExpression internalASTIsExpression]
{
 internalASTIsExpression = null;
 InternalASTName internalASTName = null;
 InternalASTIsBasicType internalASTIsBasicType = null;
 InternalASTExpression internalASTExpression = null;

}
```

```
    :
    (
     kwis:ISU
     internalASTName = overtureName
     kwlb:LBRACKET
        internalASTExpression = overtureExpression
     kwrb:RBRACKET
     {
      InternalASTKeyword keywordIs = new InternalASTKeyword(kwis,"Is");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
        internalASTIsExpression = new InternalASTIsExpressionIsName(keywordIs, internalASTName,
             keywordLeftBracket, internalASTExpression, keywordRightBracket);
     }
    )
    |
    (
     internalASTIsBasicType = overtureIsBasicType
     kwlb2:LBRACKET
        internalASTExpression = overtureExpression
     kwrb2:RBRACKET
     {
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb2,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb2,"RightBracket");
        internalASTIsExpression = new InternalASTIsExpressionIsBasicType(internalASTIsBasicType,
             keywordLeftBracket, internalASTExpression, keywordRightBracket);
     }
    )
    ;

//type judgement = 'is_', '(', expression, ',', type, ')' ;
overtureTypeJudgement returns [InternalASTTypeJudgement internalASTTypeJudgement]
{
 internalASTTypeJudgement = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTType internalASTType = null;
}
    :
    kwis:ISU
    kwlb:LBRACKET
    internalASTExpression = overtureExpression
    kwcomma:COMMA
    internalASTType = overtureType
    kwrb:RBRACKET
    {
     InternalASTKeyword keywordIs = new InternalASTKeyword(kwis,"Is");
        InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
        InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTTypeJudgement = new InternalASTTypeJudgement(keywordIs, keywordLeftBracket,
          internalASTExpression, keywordComma, internalASTType, keywordRightBracket);
    }
    ;

//undefined expression = 'undefined' ;
overtureUndefinedExpression returns [InternalASTUndefinedExpression
     internalASTUndefinedExpression]
{
 internalASTUndefinedExpression = null;
}
    :
       kwundefined:UNDEFINED
       {
        InternalASTKeyword keywordUndefined = new InternalASTKeyword(kwundefined, "Undefined");
      internalASTUndefinedExpression = new InternalASTUndefinedExpression(keywordUndefined);
    }
  ;

//isofbaseclass expression = 'isofbaseclass', '(', name, expression, ')' ;
overtureIsofbaseclassExpression returns [InternalASTIsofbaseclassExpression
     internalASTIsofbaseclassExpression]
{
 internalASTIsofbaseclassExpression = null;
 InternalASTName name = null;
 InternalASTExpression expression = null;
}
  :
    isobc:ISOFBASECLASS
    lb:LBRACKET
    name = overtureName
    expression = overtureExpression
    rb:RBRACKET
    {
     InternalASTKeyword keywordIsofbaseclass = new InternalASTKeyword(isobc, "Isofbaseclass");
```

```
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTIsofbaseclassExpression = new InternalASTIsofbaseclassExpression(
        keywordIsofbaseclass, keywordLeftBracket, name, expression, keywordRightBracket);
  }
  ;

//isofclass expression = 'isofclass', '(', name, expression, ')' ;
overtureIsofclassExpression returns [InternalASTIsofclassExpression
    internalASTIsofclassExpression]
{
 internalASTIsofclassExpression = null;
 InternalASTName name = null;
 InternalASTExpression expression = null;
}
  :
   isoc:ISOFCLASS
   lb:LBRACKET
   name = overtureName
   expression = overtureExpression
   rb:RBRACKET
   {
    InternalASTKeyword keywordIsofclass = new InternalASTKeyword(isoc, "Isofclass");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTIsofclassExpression = new InternalASTIsofclassExpression(keywordIsofclass,
        keywordLeftBracket, name, expression, keywordRightBracket);
  }
  ;

//samebaseclass expression = 'samebaseclass', '(', expression, expression, ')' ;
overtureSamebaseclassExpression returns [InternalASTSamebaseclassExpression
    internalASTSamebaseclassExpression]
{
 internalASTSamebaseclassExpression = null;
 InternalASTName name = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   sa:SAMEBASECLASS
   lb:LBRACKET
   expression1 = overtureExpression
   expression2 = overtureExpression
   rb:RBRACKET
   {
    InternalASTKeyword keywordSamebaseclass = new InternalASTKeyword(sa, "Samebaseclass");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTSamebaseclassExpression = new InternalASTSamebaseclassExpression(
        keywordSamebaseclass, keywordLeftBracket, expression1, expression2, keywordRightBracket);
  }
  ;

//sameclass expression = 'sameclass', '(', expression, expression, ')' ;
overtureSameclassExpression returns [InternalASTSameclassExpression
    internalASTSameclassExpression]
{
 internalASTSameclassExpression = null;
 InternalASTName name = null;
 InternalASTExpression expression1 = null;
 InternalASTExpression expression2 = null;
}
  :
   sa:SAMECLASS
   lb:LBRACKET
   expression1 = overtureExpression
   expression2 = overtureExpression
   rb:RBRACKET
   {
    InternalASTKeyword keywordSameclass = new InternalASTKeyword(sa, "Sameclass");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTSameclassExpression = new InternalASTSameclassExpression(keywordSameclass,
        keywordLeftBracket, expression1, expression2, keywordRightBracket);
  }
  ;

//act expression = '#act', '(', name, ')'
//       | '#act', '(', name list, ')' ;
overtureActExpression returns [InternalASTActExpression internalASTActExpression]
{
 internalASTActExpression = null;
 InternalASTName name = null;
```

```
 InternalASTNameList namelist = null;
}
 :
  (ACT LBRACKET overtureName RBRACKET)
  =>(
   sa:ACT
   lb:LBRACKET
   name = overtureName
   rb:RBRACKET
   {
    InternalASTKeyword keywordAct = new InternalASTKeyword(sa, "Act");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTActExpression = new InternalASTActExpressionName(keywordAct, keywordLeftBracket,
        name, keywordRightBracket);
   }
  )
  |
  (
   sa2:ACT
   lb2:LBRACKET
   namelist = overtureNameList
   rb2:RBRACKET
   {
    InternalASTKeyword keywordAct = new InternalASTKeyword(sa2, "Act");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2,"RightBracket");
    internalASTActExpression = new InternalASTActExpressionNameList(keywordAct,
        keywordLeftBracket, namelist, keywordRightBracket);
   }
  )
 ;

//fin expression = '#fin', '(', name, ')'
//      | '#fin', '(', name list, ')' ;
overtureFinExpression returns [InternalASTFinExpression internalASTFinExpression]
{
 internalASTFinExpression = null;
 InternalASTName name = null;
 InternalASTNameList namelist = null;
}
 :
  (FIN LBRACKET overtureName RBRACKET)
  =>(
   sa:FIN
   lb:LBRACKET
   name = overtureName
   rb:RBRACKET
   {
    InternalASTKeyword keywordFin = new InternalASTKeyword(sa, "Fin");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
    internalASTFinExpression = new InternalASTFinExpressionName(keywordFin, keywordLeftBracket,
        name, keywordRightBracket);
   }
  )
  |
  (
   sa2:FIN
   lb2:LBRACKET
   namelist = overtureNameList
   rb2:RBRACKET
   {
    InternalASTKeyword keywordFin = new InternalASTKeyword(sa2, "Fin");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2,"LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2,"RightBracket");
    internalASTFinExpression = new InternalASTFinExpressionNameList(keywordFin,
        keywordLeftBracket, namelist, keywordRightBracket);
   }
  )
 ;

//active expression = '#active', '(', name, ')'
//      | '#active', '(', name list, ')' ;
overtureActiveExpression returns [InternalASTActiveExpression internalASTActiveExpression]
{
 internalASTActiveExpression = null;
 InternalASTName name = null;
 InternalASTNameList namelist = null;
}
 :
  (ACTIVE LBRACKET overtureName RBRACKET)
  =>(
   sa:ACTIVE
```

```
      lb:LBRACKET
      name = overtureName
      rb:RBRACKET
      {
       InternalASTKeyword keywordActive = new InternalASTKeyword(sa, "Active");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
       internalASTActiveExpression = new InternalASTActiveExpressionName(keywordActive,
            keywordLeftBracket, name, keywordRightBracket);
      }
     )
     |
     (
      sa2:ACTIVE
      lb2:LBRACKET
      namelist = overtureNameList
      rb2:RBRACKET
      {
       InternalASTKeyword keywordActive = new InternalASTKeyword(sa2, "Active");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2,"LeftBracket");
       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2,"RightBracket");
       internalASTActiveExpression = new InternalASTActiveExpressionNameList(keywordActive,
            keywordLeftBracket, namelist, keywordRightBracket);
      }
     )
   ;

//req expression = '#req', '(', name, ')'
//        | '#req', '(', name list, ')' ;
overtureReqExpression returns [InternalASTReqExpression internalASTReqExpression]
{
 internalASTReqExpression = null;
 InternalASTName name = null;
 InternalASTNameList namelist = null;
}
   :
    (REQ LBRACKET overtureName RBRACKET)
    =>(
      sa:REQ
      lb:LBRACKET
      name = overtureName
      rb:RBRACKET
      {
       InternalASTKeyword keywordReq = new InternalASTKeyword(sa, "Req");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
       internalASTReqExpression = new InternalASTReqExpressionName(keywordReq, keywordLeftBracket,
            name, keywordRightBracket);
      }
     )
     |
     (
      sa2:REQ
      lb2:LBRACKET
      namelist = overtureNameList
      rb2:RBRACKET
      {
       InternalASTKeyword keywordReq = new InternalASTKeyword(sa2, "Req");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2,"LeftBracket");
       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2,"RightBracket");
       internalASTReqExpression = new InternalASTReqExpressionNameList(keywordReq,
            keywordLeftBracket, namelist, keywordRightBracket);
      }
     )
   ;

//waiting expression = '#waiting', '(', name, ')'
//        | '#waiting', '(', name list, ')' ;
overtureWaitingExpression returns [InternalASTWaitingExpression internalASTWaitingExpression]
{
 internalASTWaitingExpression = null;
 InternalASTName name = null;
 InternalASTNameList namelist = null;
}
   :
    (WAITING LBRACKET overtureName RBRACKET)
    =>(
      sa:WAITING
      lb:LBRACKET
      name = overtureName
      rb:RBRACKET
      {
       InternalASTKeyword keywordWaiting = new InternalASTKeyword(sa, "Waiting");
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb,"LeftBracket");
```

```
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb,"RightBracket");
      internalASTWaitingExpression = new InternalASTWaitingExpressionName(keywordWaiting,
           keywordLeftBracket, name, keywordRightBracket);
     }
    )
    |
    (
     sa2:WAITING
     lb2:LBRACKET
     namelist = overtureNameList
     rb2:RBRACKET
     {
      InternalASTKeyword keywordWaiting = new InternalASTKeyword(sa2, "Waiting");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb2,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb2,"RightBracket");
      internalASTWaitingExpression = new InternalASTWaitingExpressionNameList(keywordWaiting,
           keywordLeftBracket, namelist, keywordRightBracket);
     }
    )
  ;

// name = identifier, ['''', identifier ] ;
overtureName returns [InternalASTName internalASTName]
{
 internalASTName = null;
 InternalASTIdentifier identifier1 = null;
 InternalASTKeyword keywordMark = null;
 InternalASTIdentifier identifier2 = null;
}
  :
   id1: IDENTIFIER
    {
     identifier1 = new InternalASTIdentifier(id1);
    }

   (
    ma: MARK
     {
      keywordMark = new InternalASTKeyword(ma, "Mark");
     }
    id2: IDENTIFIER
     {
      identifier2 = new InternalASTIdentifier(id2);
     }
   )?
   {
    internalASTName = new InternalASTName(identifier1, keywordMark, identifier2);
   }
  ;

// name list = name, { ',', name}
overtureNameList returns [InternalASTNameList internalASTNameList]
{
 internalASTNameList = null;
 InternalASTName name = null;
 List<InternalASTNameListElement> nameListElements = new ArrayList<InternalASTNameListElement>();
 InternalASTNameListElement currentElement = null;
}
  :
   (
    name = overtureName
     {
      currentElement = new InternalASTNameListElement(name);
      nameListElements.add(currentElement);
     }

    (
     co:COMMA
     name = overtureName
     {
      InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
      currentElement = new InternalASTNameListElement(keywordComma, name);
      nameListElements.add(currentElement);
     }
    )*
   )
   {
    internalASTNameList = new InternalASTNameList(nameListElements);
   }
  ;

// old name = identifier, '~' ;
overtureOldName returns [InternalASTOldName internalASTOldName]
{
```

```
 internalASTOldName = null;
}
  :
   (
    id:IDENTIFIER
    ti:TILDE
   )
   {
    InternalASTIdentifier identifier = new InternalASTIdentifier(id);
    InternalASTKeyword keywordTilde = new InternalASTKeyword(ti, "Tilde");
    internalASTOldName = new InternalASTOldName(identifier, keywordTilde);
   }
  ;

// state designator = name | field reference | map or sequence reference ;
overtureStateDesignator returns [InternalASTStateDesignator internalASTStateDesignator]
{
 internalASTStateDesignator = null;
 InternalASTName name = null;
}
  :
   (overtureMapOrSequenceReference)=> internalASTStateDesignator = overtureMapOrSequenceReference
   | (overtureFieldReference)=> internalASTStateDesignator = overtureFieldReference
   | name = overtureName
       {
        internalASTStateDesignator = new InternalASTStateDesignatorName(name);
       }
  ;

// field reference = state designator, '.', identifier ;
overtureFieldReference returns [InternalASTFieldReference internalASTFieldReference]
{
 internalASTFieldReference = null;
 InternalASTStateDesignator statedesignator = null;
 InternalASTIdentifier identifier = null;
 InternalASTName name = null;
}
  :
   (
    (
     //(overtureMapOrSequenceReference)=> statedesignator = overtureMapOrSequenceReference
     name = overtureName
       {
        statedesignator = new InternalASTStateDesignatorName(name);
       }
    )
    dot:DOT
    id:IDENTIFIER
   )
   {
    identifier = new InternalASTIdentifier(id);
    InternalASTKeyword keywordDot = new InternalASTKeyword(dot, "Dot");
    internalASTFieldReference = new InternalASTFieldReference(statedesignator, keywordDot,
         identifier);
   }
  ;

// map or sequence reference = state designator, '(', expression, ')' ;
overtureMapOrSequenceReference returns [InternalASTMapOrSequenceReference
     internalASTMapOrSequenceReference]
{
 internalASTMapOrSequenceReference = null;
 InternalASTStateDesignator statedesignator = null;
 InternalASTExpression expression = null;
 InternalASTName name = null;
}
  :
   (
    (
     (overtureFieldReference)=> statedesignator = overtureFieldReference
     |
     name = overtureName
       {
        statedesignator = new InternalASTStateDesignatorName(name);
       }
    )
    lb:LBRACKET
    expression = overtureExpression
    rb:RBRACKET
   )
   {
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
    internalASTMapOrSequenceReference = new InternalASTMapOrSequenceReference(statedesignator,
```

```
        keywordLeftBracket, expression, keywordRightBracket);
  }
  ;


// STATEMENT

//statement = let statement
//    | let be statement
//    | def statement
//    | block statement
//    | general assign statement
//    | if statement
//    | cases statement
//    | sequence for loop
//    | set for loop
//    | index for loop
//    | while loop
//    | nondeterministic statement
//    | call statement
//    | specification statement
//    | start statement
//    | start list statement
//    | return statement
//    | always statement
//    | trap statement
//    | recursive trap statement
//    | exit statement
//    | error statement
//    | identity statement ;
overtureStatement returns [InternalASTStatement internalASTStatement]
{
 internalASTStatement = null;
}
  :
   (
     (overtureGeneralAssignStatement)=> internalASTStatement = overtureGeneralAssignStatement
    | (overtureLetBeStatement)=> internalASTStatement = overtureLetBeStatement
    | internalASTStatement = overtureLetStatement
    | internalASTStatement = overtureDefStatement
    | internalASTStatement = overtureBlockStatement
    | internalASTStatement = overtureIfStatement
    | internalASTStatement = overtureCasesStatement
    | (overtureSequenceForLoop)=> internalASTStatement = overtureSequenceForLoop
    | internalASTStatement = overtureSetForLoop
    | internalASTStatement = overtureIndexForLoop
    | internalASTStatement = overtureWhileLoop
    | internalASTStatement = overtureNondeterministicStatement
    | internalASTStatement = overtureCallStatement
    | internalASTStatement = overtureSpecificationStatement
    | internalASTStatement = overtureStartStatement
    | internalASTStatement = overtureStartListStatement
    | internalASTStatement = overtureReturnStatement
    | internalASTStatement = overtureAlwaysStatement
    | internalASTStatement = overtureTrapStatement
    | internalASTStatement = overtureRecursiveTrapStatement
    | internalASTStatement = overtureExitStatement
    | internalASTStatement = overtureErrorStatement
    | internalASTStatement = overtureIdentityStatement
   )
   ;

//let statement = 'let', local definition, { ',', local definition },
//                'in', statement ;
overtureLetStatement returns [InternalASTLetStatement internalASTLetStatement]
{
 internalASTLetStatement = null;
 List<InternalASTLetStatementElement> letStatementElements = new ArrayList<
      InternalASTLetStatementElement>();
 InternalASTLetStatementElement currentElement = null;
 InternalASTKeyword keywordComma = null;
 InternalASTStatement internalASTStatement = null;
 InternalASTLocalDefinition internalASTLocalDefinition = null;
}
   :
     klet:LET
     internalASTLocalDefinition = overtureLocalDefinition
     {
        currentElement = new InternalASTLetStatementElement(internalASTLocalDefinition);
      letStatementElements.add(currentElement);
     }
     (
      kwcomma:COMMA
      internalASTLocalDefinition = overtureLocalDefinition
```

```
      {
        keywordComma = new InternalASTKeyword(kwcomma,"Comma");
      currentElement = new InternalASTLetStatementElement(keywordComma, internalASTLocalDefinition);
      letStatementElements.add(currentElement);
      }
     )*
     kwin:IN
     internalASTStatement = overtureStatement
     {
      InternalASTKeyword keywordLet = new InternalASTKeyword(klet,"Let");
    InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
     internalASTLetStatement = new InternalASTLetStatement(keywordLet, letStatementElements, keywordIn, internalASTStatement);
     }
  ;

//local definition = value definition
//      | function definition ;
overtureLocalDefinition returns [InternalASTLocalDefinition internalASTLocalDefinition]
{
 internalASTLocalDefinition = null;
}
  :
   (
      (overtureFunctionDefinition) => internalASTLocalDefinition = overtureFunctionDefinition
    | (overtureValueDefinition) => internalASTLocalDefinition = overtureValueDefinition
   )
   ;

//let be statement = 'let', bind, [ 'be', 'st', expression ], 'in',
//        statement ;
overtureLetBeStatement returns [InternalASTLetBeStatement internalASTLetBeStatement]
{
 internalASTLetBeStatement = null;
 InternalASTBind internalASTBind = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement = null;
 InternalASTKeyword keywordBe = null;
 InternalASTKeyword keywordSt = null;
}
   :
     kwlet:LET
     internalASTBind = overtureBind
     (
      kwbe:BE
      kwst:ST
      internalASTExpression = overtureExpression
      {
       keywordBe = new InternalASTKeyword(kwbe,"Be");
        keywordSt = new InternalASTKeyword(kwst,"St");
      }
     )?
     kwin:IN
          internalASTStatement = overtureStatement
     {
    InternalASTKeyword keywordLet = new InternalASTKeyword(kwlet,"Let");
     InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
     internalASTLetBeStatement = new InternalASTLetBeStatement(keywordLet, internalASTBind, keywordBe, keywordSt, internalASTExpression, keywordIn, internalASTStatement);
     }
  ;

// def statement = 'def', equals definition,
//                 { ';', equals definition }, [ ';' ],
//       'in', statement ;
overtureDefStatement returns [InternalASTDefStatement internalASTDefStatement]
{
 internalASTDefStatement = null;
 List<InternalASTDefStatementElement> defStatementElements = new ArrayList<InternalASTDefStatementElement>();
 InternalASTDefStatementElement currentElement = null;
 InternalASTEqualsDefinition internalASTEqualsDefinition = null;
 InternalASTAccessTypeDefinition internalASTAccessTypeDefinition = null;
 InternalASTKeyword keywordDef = null;
 InternalASTStatement internalASTStatement = null;
}
  :
   kwdef:DEF
    {
     keywordDef = new InternalASTKeyword(kwdef,"Def");
    }

   (
```

```
    internalASTEqualsDefinition = overtureEqualsDefinition
     {
      currentElement = new InternalASTDefStatementElement(internalASTEqualsDefinition);
      defStatementElements.add(currentElement);
     }

    (options {greedy=true;}:
     seNoN:SEMICOLON
     {
      InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonNoN);
      currentElement.setEndPositionFromNode(semicolonNoN);
     }
     internalASTEqualsDefinition = overtureEqualsDefinition
     {
      currentElement = new InternalASTDefStatementElement(internalASTEqualsDefinition);
        defStatementElements.add(currentElement);
     }
    )*

    (seLast:SEMICOLON
             {
      InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
      currentElement.setKeywordSemicolon(semicolonLast);
      currentElement.setEndPositionFromNode(semicolonLast);
              }
    )?

   )?
   kwin:IN
   internalASTStatement = overtureStatement
   {
    InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
    internalASTDefStatement = new InternalASTDefStatement(keywordDef, defStatementElements,
         keywordIn, internalASTStatement);
   }
  ;

//equals definition = pattern bind, '=', expression ;
overtureEqualsDefinition returns [InternalASTEqualsDefinition internalASTEqualsDefinition]
{
 internalASTEqualsDefinition = null;
 InternalASTPatternBind internalASTPatternBind = null;
 InternalASTExpression internalASTExpression = null;
}
   :
     internalASTPatternBind = overturePatternBind
     kweq:EQUALSIGN
     internalASTExpression = overtureExpression
     {
    InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq,"Equalsign");
     internalASTEqualsDefinition = new InternalASTEqualsDefinition(internalASTPatternBind,
          keywordEqualsign, internalASTExpression);
     }
  ;

// block statement = '(', { dcl statement },
//                  statement, { ';', statement }, [ ';' ], ')' ;
overtureBlockStatement returns [InternalASTBlockStatement internalASTBlockStatement]
{
 internalASTBlockStatement = null;
 InternalASTDclStatement internalASTDclStatement = null;
 List<InternalASTDclStatement> dclStatements = new ArrayList<InternalASTDclStatement>();
 InternalASTBlockStatementElement currentElement = null;
 InternalASTStatement internalASTStatement = null;
 List<InternalASTBlockStatementElement> blockStatementElements = new ArrayList<
      InternalASTBlockStatementElement>();
}
  :
   kwlb:LBRACKET
    (
     internalASTDclStatement = overtureDclStatement
     {
      dclStatements.add(internalASTDclStatement);
     }
    )*
   (
    internalASTStatement = overtureStatement
     {
      currentElement = new InternalASTBlockStatementElement(internalASTStatement);
      blockStatementElements.add(currentElement);
     }

    (options {greedy=true;}:
```

```
      seNoN:SEMICOLON
       {
        InternalASTKeyword semicolonNoN = new InternalASTKeyword(seNoN,"Semicolon");
        currentElement.setKeywordSemicolon(semicolonNoN);
        currentElement.setEndPositionFromNode(semicolonNoN);
       }
       internalASTStatement = overtureStatement
       {
        currentElement = new InternalASTBlockStatementElement(internalASTStatement);
          blockStatementElements.add(currentElement);
       }
      )*

     (seLast:SEMICOLON
               {
        InternalASTKeyword semicolonLast = new InternalASTKeyword(seLast,"Semicolon");
        currentElement.setKeywordSemicolon(semicolonLast);
        currentElement.setEndPositionFromNode(semicolonLast);
                }
     )?
    )
    kwrb:RBRACKET
    {
     InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
     InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTBlockStatement = new InternalASTBlockStatement(keywordLeftBracket, dclStatements,
          blockStatementElements, keywordRightBracket);
    }
  ;

//dcl statement = 'dcl', assignment definition,
//      { ',', assignment definition }, ';' ;
overtureDclStatement returns [InternalASTDclStatement internalASTDclStatement]
{
 internalASTDclStatement = null;
 InternalASTAssignmentDefinition internalASTAssignmentDefinition = null;
 InternalASTDclStatementElement internalASTDclStatementElement = null;
 List<InternalASTDclStatementElement> dclStatementElements = new ArrayList<
      InternalASTDclStatementElement>();
}
   :
      kwdcl:DCL
      internalASTAssignmentDefinition = overtureAssignmentDefinition
      {
            internalASTDclStatementElement = new InternalASTDclStatementElement(
                 internalASTAssignmentDefinition);
        dclStatementElements.add(internalASTDclStatementElement);
      }
      (
       kwcomma:COMMA
       internalASTAssignmentDefinition = overtureAssignmentDefinition
     {
       InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
       internalASTDclStatementElement = new InternalASTDclStatementElement(keywordComma,
            internalASTAssignmentDefinition);
        dclStatementElements.add(internalASTDclStatementElement);
        }
      )*
      kwsc:SEMICOLON
      {
       InternalASTKeyword keywordDcl = new InternalASTKeyword(kwdcl,"Dcl");
     InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc,"Semicolon");
      internalASTDclStatement = new InternalASTDclStatement(keywordDcl, dclStatementElements,
           keywordSemicolon);
      }
  ;

//assignment definition = identifier, ':', type, [ ':=', expression ] ;
overtureAssignmentDefinition returns [InternalASTAssignmentDefinition
      internalASTAssignmentDefinition]
{
 internalASTAssignmentDefinition = null;
 InternalASTType internalASTType = null;
 InternalASTKeyword keywordColonEqual = null;
 InternalASTExpression internalASTExpression = null;
}
   :
      id:IDENTIFIER
      kwcolon:COLON
      internalASTType = overtureType
      (
       kwcolonequal:COLONEQUAL
       internalASTExpression = overtureExpression
        {
```

```
        keywordColonEqual = new InternalASTKeyword(kwcolonequal,"ColonEqual");
      }
     )?
     {
      InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
      InternalASTKeyword keywordColon = new InternalASTKeyword(kwcolon,"Colon");
      internalASTAssignmentDefinition = new InternalASTAssignmentDefinition(internalASTIdentifier,
           keywordColon, internalASTType, keywordColonEqual, internalASTExpression);
     }
 ;

//general assign statement = assign statement
//          | multiple assign statement ;
overtureGeneralAssignStatement returns [InternalASTGeneralAssignStatement
      internalASTGeneralAssignStatement]
{
 internalASTGeneralAssignStatement = null;
}
  :
   (
      internalASTGeneralAssignStatement = overtureAssignStatement
    | internalASTGeneralAssignStatement = overtureMultipleAssignStatement
   )
   ;

//assign statement = state designator, ':=', expression ;
overtureAssignStatement returns [InternalASTAssignStatement internalASTAssignStatement]
{
 internalASTAssignStatement = null;
 InternalASTStateDesignator internalASTStateDesignator = null;
 InternalASTExpression internalASTExpression;
}
   :
     internalASTStateDesignator = overtureStateDesignator
     kwcolonequal:COLONEQUAL
     internalASTExpression = overtureExpression
     {
      InternalASTKeyword keywordColonEqual = new InternalASTKeyword(kwcolonequal,"ColonEqual");
      internalASTAssignStatement = new InternalASTAssignStatement(internalASTStateDesignator,
           keywordColonEqual, internalASTExpression);
     }
 ;

//multiple assign statement = 'atomic', '(' assign statement, ';',
//          assign statement,
//          [ { ';', assign statement } ], ')' ;
overtureMultipleAssignStatement returns [InternalASTMultipleAssignStatement
      internalASTMultipleAssignStatement]
{
 internalASTMultipleAssignStatement = null;
 InternalASTAssignStatement internalASTAssignStatement = null;
 InternalASTMultipleAssignStatementElement internalASTMultipleAssignStatementElement = null;
 List<InternalASTMultipleAssignStatementElement> multipleAssignStatementElements = new ArrayList<
       InternalASTMultipleAssignStatementElement>();
 InternalASTExpression internalASTExpression;
}
   :
     kwatomic:ATOMIC
     kwlb:LBRACKET
     internalASTAssignStatement = overtureAssignStatement
     {
    internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement(
         internalASTAssignStatement);
      multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
     }
     kwsc:SEMICOLON
     internalASTAssignStatement = overtureAssignStatement
     {
      InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc,"Semicolon");
    internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement(
         keywordSemicolon, internalASTAssignStatement);
      multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
     }
    (
     kwsc2:SEMICOLON
     internalASTAssignStatement = overtureAssignStatement
     {
      InternalASTKeyword keywordSemicolon = new InternalASTKeyword(kwsc2,"Semicolon");
    internalASTMultipleAssignStatementElement = new InternalASTMultipleAssignStatementElement(
         keywordSemicolon, internalASTAssignStatement);
      multipleAssignStatementElements.add(internalASTMultipleAssignStatementElement);
     }
     )*
     kwrb:RBRACKET
```

```
      {
        InternalASTKeyword keywordAtomic = new InternalASTKeyword(kwatomic, "Atomic");;
       InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
       InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
       internalASTMultipleAssignStatement = new InternalASTMultipleAssignStatement(keywordAtomic,
            keywordLeftBracket, multipleAssignStatementElements, keywordRightBracket);
      }
  ;

//if statement = 'if', expression, 'then', statement,
//        { elseif statement },
//        [ 'else', statement ] ;
overtureIfStatement returns [InternalASTIfStatement internalASTIfStatement]
{
 internalASTIfStatement = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement1 = null;
 InternalASTElseifStatement internalASTElseifStatement = null;
 List<InternalASTElseifStatement> elseifStatements = new ArrayList<InternalASTElseifStatement>();
 InternalASTStatement internalASTStatement2 = null;
 InternalASTKeyword keywordElse = null;
}
    :
      kwif:IF
      internalASTExpression = overtureExpression
      kwthen:THEN
      internalASTStatement1 = overtureStatement
      (
      (options {greedy=true;}:
        internalASTElseifStatement = overtureElseifStatement
         {
          elseifStatements.add(internalASTElseifStatement);
         }
      )*
      (options {greedy=true;}:
       kwelse:ELSE
       internalASTStatement2 = overtureStatement
        {
         keywordElse = new InternalASTKeyword(kwelse,"Else");
        }
      )?
      )
      {
       InternalASTKeyword keywordIf = new InternalASTKeyword(kwif,"If");
       InternalASTKeyword keywordThen = new InternalASTKeyword(kwthen,"Then");
     internalASTIfStatement = new InternalASTIfStatement(keywordIf, internalASTExpression,
          keywordThen, internalASTStatement1, elseifStatements, keywordElse, internalASTStatement2)
          ;
      }
  ;

//elseif statement = 'elseif', expression, 'then', statement ;
overtureElseifStatement returns [InternalASTElseifStatement internalASTElseifStatement]
{
 internalASTElseifStatement = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement = null;
}
    :
      kwelseif:ELSEIF
      internalASTExpression = overtureExpression
      kwthen:THEN
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordElseif = new InternalASTKeyword(kwelseif,"Elseif");
       InternalASTKeyword keywordThen = new InternalASTKeyword(kwthen,"Then");
       internalASTElseifStatement = new InternalASTElseifStatement(keywordElseif,
            internalASTExpression, keywordThen, internalASTStatement);
      }
  ;

//cases statement = 'cases', expression, ':',
//          cases statement alternatives,
//          [ ',', others statement ], 'end' ;
overtureCasesStatement returns [InternalASTCasesStatement internalASTCasesStatement]
{
 internalASTCasesStatement = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTCasesStatementAlternatives internalASTCasesStatementAlternatives = null;
 InternalASTOthersStatement internalASTOthersStatement = null;
 InternalASTKeyword keywordComma = null;
}
    :
      kwcases:CASES
```

```
      internalASTExpression = overtureExpression
      kwcolon:COLON
      internalASTCasesStatementAlternatives = overtureCasesStatementAlternatives
      (
       kwcomma:COMMA
       internalASTOthersStatement = overtureOthersStatement
       {
       keywordComma = new InternalASTKeyword(kwcomma,"Comma");
     }
      )?
      kwend:END
      {
       InternalASTKeyword keywordCases = new InternalASTKeyword(kwcases,"Cases");
       InternalASTKeyword keywordColon = new InternalASTKeyword(kwcolon,"Colon");
       InternalASTKeyword keywordEnd = new InternalASTKeyword(kwend,"End");
       internalASTCasesStatement = new InternalASTCasesStatement(keywordCases,
            internalASTExpression, keywordColon, internalASTCasesStatementAlternatives,
            keywordComma, internalASTOthersStatement, keywordEnd);
      }
 ;

 //cases statement alternatives = cases statement alternative,
//                               { ',', cases statement alternative } ;
overtureCasesStatementAlternatives returns [InternalASTCasesStatementAlternatives
     internalASTCasesStatementAlternatives]
{
 internalASTCasesStatementAlternatives = null;
 InternalASTCasesStatementAlternative casesStatementAlternative = null;
 InternalASTCasesStatementAlternativesElement internalASTCasesStatementAlternativesElement = null
     ;
 InternalASTKeyword keywordComma = null;
 List<InternalASTCasesStatementAlternativesElement> casesStatementAlternatives = new ArrayList<
      InternalASTCasesStatementAlternativesElement>();
}
   :
      casesStatementAlternative = overtureCasesStatementAlternative
      {
     internalASTCasesStatementAlternativesElement = new
          InternalASTCasesStatementAlternativesElement(casesStatementAlternative);
       casesStatementAlternatives.add(internalASTCasesStatementAlternativesElement);
      }
      (
      (
        kwcomma:COMMA
       casesStatementAlternative = overtureCasesStatementAlternative
        {
         keywordComma = new InternalASTKeyword(kwcomma, "Comma");
         internalASTCasesStatementAlternativesElement = new
              InternalASTCasesStatementAlternativesElement(keywordComma, casesStatementAlternative)
              ;
         casesStatementAlternatives.add(internalASTCasesStatementAlternativesElement);
      }
      )*
      )
      {
       internalASTCasesStatementAlternatives = new InternalASTCasesStatementAlternatives(
            casesStatementAlternatives);
      }
 ;

//cases statement alternative = pattern list, '->', statement ;
overtureCasesStatementAlternative returns [InternalASTCasesStatementAlternative
     internalASTCasesStatementAlternative]
{
 internalASTCasesStatementAlternative = null;
 InternalASTPatternList internalASTPatternList = null;
 InternalASTStatement internalASTStatement = null;
}
   :
      internalASTPatternList = overturePatternList
      kwarrow:LINEARROW
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordArrow = new InternalASTKeyword(kwarrow,"Arrow");
       internalASTCasesStatementAlternative = new InternalASTCasesStatementAlternative(
            internalASTPatternList, keywordArrow, internalASTStatement);
      }
 ;

//others statement = 'others', '->', statement ;
overtureOthersStatement returns [InternalASTOthersStatement internalASTOthersStatement]
{
 internalASTOthersStatement = null;
 InternalASTStatement internalASTStatement = null;
```

```
}
  :
    kwothers:OTHERS
    kwarrow:LINEARROW
    internalASTStatement = overtureStatement
    {
     InternalASTKeyword keywordOthers = new InternalASTKeyword(kwothers,"Others");
     InternalASTKeyword keywordArrow = new InternalASTKeyword(kwarrow,"Arrow");
     internalASTOthersStatement = new InternalASTOthersStatement(keywordOthers, keywordArrow,
          internalASTStatement);
    }
  ;

//sequence for loop = 'for', pattern bind, 'in', [ 'reverse' ],
//        expression, 'do', statement ;
overtureSequenceForLoop returns [InternalASTSequenceForLoop internalASTSequenceForLoop]
{
 internalASTSequenceForLoop = null;
 InternalASTPatternBind internalASTPatternBind = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement = null;
 InternalASTKeyword keywordReverse = null;
}
  :
    kwfor:FOR
    internalASTPatternBind = overturePatternBind
    kwin:IN
    (
     kwreverse:REVERSE
     {
      keywordReverse = new InternalASTKeyword(kwreverse,"Reverse");
     }
    )?
    internalASTExpression = overtureExpression
    kwdo:DO
    internalASTStatement = overtureStatement
    {
     InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor,"For");
     InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
     InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
     internalASTSequenceForLoop = new InternalASTSequenceForLoop(keywordFor,
          internalASTPatternBind, keywordIn, keywordReverse, internalASTExpression, keywordDo,
          internalASTStatement);
    }
  ;

//set for loop = 'for', 'all', pattern, 'in set', expression,
//      'do', statement ;
overtureSetForLoop returns [InternalASTSetForLoop internalASTSetForLoop]
{
 internalASTSetForLoop = null;
 InternalASTPattern internalASTPattern = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement = null;
}
  :
    kwfor:FOR
    kwall:ALL
    internalASTPattern = overturePattern
    kwin:IN
    kwset:SET
    internalASTExpression = overtureExpression
    kwdo:DO
    internalASTStatement = overtureStatement
    {
     InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor,"For");
     InternalASTKeyword keywordAll = new InternalASTKeyword(kwall,"All");
     InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
     InternalASTKeyword keywordSet = new InternalASTKeyword(kwset,"Set");
     InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
     internalASTSetForLoop = new InternalASTSetForLoop(keywordFor, keywordAll, internalASTPattern
          , keywordIn, keywordSet, internalASTExpression, keywordDo, internalASTStatement);
    }
  ;

//index for loop = 'for', identifier, '=', expression, 'to', expression,
//        [ 'by', expression ],
//       'do', statement ;
overtureIndexForLoop returns [InternalASTIndexForLoop internalASTIndexForLoop]
{
 internalASTIndexForLoop = null;
 InternalASTExpression internalASTExpression1 = null;
 InternalASTExpression internalASTExpression2 = null;
 InternalASTExpression internalASTExpression3 = null;
```

```
 InternalASTStatement internalASTStatement = null;
 InternalASTKeyword keywordBy = null;
}
   :
      kwfor:FOR
      kwid:IDENTIFIER
      kweq:EQUALSIGN
      internalASTExpression1 = overtureExpression
      kwto:TO
      internalASTExpression2 = overtureExpression
      (
       kwby:BY
       internalASTExpression3 = overtureExpression
       {
        keywordBy = new InternalASTKeyword(kwby,"By");
       }
      )?
      kwdo:DO
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordFor = new InternalASTKeyword(kwfor,"For");
       InternalASTIdentifier identifier = new InternalASTIdentifier(kwid);
      InternalASTKeyword keywordEqualsign = new InternalASTKeyword(kweq,"Equalsign");
       InternalASTKeyword keywordTo = new InternalASTKeyword(kwto,"To");
       InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
       internalASTIndexForLoop = new InternalASTIndexForLoop(keywordFor, identifier,
            keywordEqualsign, internalASTExpression1, keywordTo, internalASTExpression2, keywordBy,
             internalASTExpression3, keywordDo, internalASTStatement);
      }
 ;

//while loop = 'while', expression, 'do', statement ;
overtureWhileLoop returns [InternalASTWhileLoop internalASTWhileLoop]
{
 internalASTWhileLoop = null;
 InternalASTExpression internalASTExpression = null;
 InternalASTStatement internalASTStatement = null;
}
   :
      kwwhile:WHILE
      internalASTExpression = overtureExpression
      kwdo:DO
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordWhile = new InternalASTKeyword(kwwhile,"While");
       InternalASTKeyword keywordDo = new InternalASTKeyword(kwdo,"Do");
       internalASTWhileLoop = new InternalASTWhileLoop(keywordWhile, internalASTExpression,
            keywordDo, internalASTStatement);
      }
 ;

//nondeterministic statement = '||', '(', statement,
//            { ',', statement }, ')' ;
overtureNondeterministicStatement returns [InternalASTNondeterministicStatement
     internalASTNondeterministicStatement]
{
 internalASTNondeterministicStatement = null;
 InternalASTStatement internalASTStatement = null;
 InternalASTNondeterministicStatementElement internalASTNondeterministicStatementElement = null;
 List<InternalASTNondeterministicStatementElement> nondeterministicStatementElements = new
      ArrayList<InternalASTNondeterministicStatementElement>();
}
   :
      kwvbars:VBARS
      kwlb:LBRACKET
      internalASTStatement = overtureStatement
       {
        internalASTNondeterministicStatementElement = new
             InternalASTNondeterministicStatementElement(internalASTStatement);
          nondeterministicStatementElements.add(internalASTNondeterministicStatementElement);
       }
      (options {greedy=true;}:
         kwcomma:COMMA
      internalASTStatement = overtureStatement
          {
           InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
           internalASTNondeterministicStatementElement = new
                InternalASTNondeterministicStatementElement(keywordComma, internalASTStatement);
            nondeterministicStatementElements.add(internalASTNondeterministicStatementElement);
       }
      )*
      kwrb:RBRACKET
      {
       InternalASTKeyword keywordVBars = new InternalASTKeyword(kwvbars,"VBars");
```

```
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
      internalASTNondeterministicStatement = new InternalASTNondeterministicStatement(keywordVBars
           , keywordLeftBracket, nondeterministicStatementElements, keywordRightBracket);
     }
  ;

//call statement = [ object designator, '.' ],
//        name, '(', [ expression list ], ')', ;
overtureCallStatement returns [InternalASTCallStatement internalASTCallStatement]
{
 internalASTCallStatement = null;
 InternalASTObjectDesignator internalASTObjectDesignator = null;
 InternalASTName internalASTName = null;
 InternalASTExpressionList internalASTExpressionList = null;
 InternalASTKeyword keywordDot = null;
}
   :
     (
     (overtureObjectDesignator DOT overtureName LBRACKET)
     =>(
      internalASTObjectDesignator = overtureObjectDesignator
      kwdot:DOT
      {
       keywordDot = new InternalASTKeyword(kwdot,"Dot");
      }
     )
     |
     ()
     )
     internalASTName = overtureName
     kwlb:LBRACKET
     (
      internalASTExpressionList = overtureExpressionList
     )?
     kwrb:RBRACKET
     {
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
      internalASTCallStatement = new InternalASTCallStatement(internalASTObjectDesignator,
           keywordDot, internalASTName, keywordLeftBracket, internalASTExpressionList,
           keywordRightBracket);
     }
  ;

//object designator = name
//      | self expression
//      | new expression
//      | object field reference
//      | object apply ;
overtureObjectDesignator returns [InternalASTObjectDesignator internalASTObjectDesignator]
{
 internalASTObjectDesignator = null;
 InternalASTName name = null;
 InternalASTSelfExpression selfExpression = null;
 InternalASTNewExpression newExpression = null;
}
   :
     (
     (overtureName)=> name = overtureName
       {
        internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
       }
     |(overtureObjectApply)=> internalASTObjectDesignator = overtureObjectApply

     | internalASTObjectDesignator = overtureObjectFieldReference

     | newExpression = overtureNewExpression
       {
        internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(newExpression);
       }
     | selfExpression = overtureSelfExpression
       {
        internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(selfExpression
            );
       }

     )
  ;

//object field reference = object designator, '.', identifier ;
overtureObjectFieldReference returns [InternalASTObjectFieldReference
     internalASTObjectFieldReference]
{
```

```
 internalASTObjectFieldReference = null;
 InternalASTObjectDesignator internalASTObjectDesignator = null;
 InternalASTName name = null;
 InternalASTSelfExpression selfExpression = null;
 InternalASTNewExpression newExpression = null;
}
   :
     (
       (

        name = overtureName
         {
          internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
         }
         //| newExpression = overtureNewExpression
        // {
        //   internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(
            newExpression);
        // }
        //| selfExpression = overtureSelfExpression
        // {
        //   internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(
            selfExpression);
        // }

       )
     )
     kwdot:DOT
     id:IDENTIFIER
     {
      InternalASTKeyword keywordDot = new InternalASTKeyword(kwdot,"Dot");
      InternalASTIdentifier internalASTIdentifier = new InternalASTIdentifier(id);
     internalASTObjectFieldReference = new InternalASTObjectFieldReference(
          internalASTObjectDesignator, keywordDot, internalASTIdentifier);
     }
 ;

//object apply = object designator, '(', [ expression list ], ')' ;
overtureObjectApply returns [InternalASTObjectApply internalASTObjectApply]
{
 internalASTObjectApply = null;
 InternalASTObjectDesignator internalASTObjectDesignator = null;
 InternalASTExpressionList internalASTExpressionList = null;
 InternalASTName name = null;
 InternalASTSelfExpression selfExpression = null;
 InternalASTNewExpression newExpression = null;
}
   :
     (
        (overtureObjectFieldReference)=> internalASTObjectDesignator =
             overtureObjectFieldReference


       | name = overtureName
        {
        internalASTObjectDesignator = new InternalASTObjectDesignatorName(name);
        }
        //| newExpression = overtureNewExpression
       // {
       //   internalASTObjectDesignator = new InternalASTObjectDesignatorNewExpression(
            newExpression);
       // }
     //| selfExpression = overtureSelfExpression
       // {
       //   internalASTObjectDesignator = new InternalASTObjectDesignatorSelfExpression(
            selfExpression);
       // }


     )
     kwlb:LBRACKET
     (
      internalASTExpressionList = overtureExpressionList
     )?
     kwrb:RBRACKET
     {
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
     internalASTObjectApply = new InternalASTObjectApply(internalASTObjectDesignator,
          keywordLeftBracket, internalASTExpressionList, keywordRightBracket);
     }
 ;

//return statement = 'return', [ expression ] ;
```

```
overtureReturnStatement returns [InternalASTReturnStatement internalASTReturnStatement]
{
 internalASTReturnStatement = null;
 InternalASTExpression internalASTExpression = null;
}
   :
      kwreturn:RETURN
      (
       internalASTExpression = overtureExpression
      )?
      {
       InternalASTKeyword keywordReturn = new InternalASTKeyword(kwreturn,"Return");
      internalASTReturnStatement = new InternalASTReturnStatement(keywordReturn,
           internalASTExpression);
      }
  ;

//specification statement = '[', implicit operation body, ']' ;
overtureSpecificationStatement returns [InternalASTSpecificationStatement
      internalASTSpecificationStatement]
{
 internalASTSpecificationStatement = null;
 InternalASTImplicitOperationBody internalASTImplicitOperationBody = null;
}
   :
      kwlb:LBRACK
      internalASTImplicitOperationBody = overtureImplicitOperationBody
      kwrb:RBRACK
      {
       InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(kwlb,"LeftBrack");
      InternalASTKeyword keywordRightBrack = new InternalASTKeyword(kwrb,"RightBrack");
      internalASTSpecificationStatement = new InternalASTSpecificationStatement(keywordLeftBrack,
           internalASTImplicitOperationBody, keywordRightBrack);
      }
  ;

//start statement = 'start', '(', expression, ')' ;
overtureStartStatement returns [InternalASTStartStatement internalASTStartStatement]
{
 internalASTStartStatement = null;
 InternalASTExpression internalASTExpression = null;
}
   :
      kwstart:START
      kwlb:LBRACKET
      internalASTExpression = overtureExpression
      kwrb:RBRACKET
      {
       InternalASTKeyword keywordStart = new InternalASTKeyword(kwstart,"Start");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
      internalASTStartStatement = new InternalASTStartStatement(keywordStart, keywordLeftBracket,
           internalASTExpression, keywordRightBracket);
      }
  ;

//start list statement = 'startlist', '(', expression, ')' ;
overtureStartListStatement returns [InternalASTStartListStatement internalASTStartListStatement]
{
 internalASTStartListStatement = null;
 InternalASTExpression internalASTExpression = null;
}
   :
      kwstartlist:STARTLIST
      kwlb:LBRACKET
      internalASTExpression = overtureExpression
      kwrb:RBRACKET
      {
       InternalASTKeyword keywordStartlist = new InternalASTKeyword(kwstartlist,"Startlist");
      InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(kwlb,"LeftBracket");
      InternalASTKeyword keywordRightBracket = new InternalASTKeyword(kwrb,"RightBracket");
      internalASTStartListStatement = new InternalASTStartListStatement(keywordStartlist,
           keywordLeftBracket, internalASTExpression, keywordRightBracket);
      }
  ;

//always statement = 'always', statement, 'in', statement ;
overtureAlwaysStatement returns [InternalASTAlwaysStatement internalASTAlwaysStatement]
{
 internalASTAlwaysStatement = null;
 InternalASTStatement internalASTStatement1 = null;
 InternalASTStatement internalASTStatement2 = null;
}
   :
```

```
      kwalways:ALWAYS
      internalASTStatement1 = overtureStatement
      kwin:IN
      internalASTStatement2 = overtureStatement
      {
       InternalASTKeyword keywordAlways = new InternalASTKeyword(kwalways,"Always");
      InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
      internalASTAlwaysStatement = new InternalASTAlwaysStatement(keywordAlways,
           internalASTStatement1, keywordIn, internalASTStatement2);
      }
  ;

//trap statement = 'trap', pattern bind, 'with', statement,
//        'in', statement ;
overtureTrapStatement returns [InternalASTTrapStatement internalASTTrapStatement]
{
 internalASTTrapStatement = null;
 InternalASTPatternBind internalASTPatternBind = null;
 InternalASTStatement internalASTStatement1 = null;
 InternalASTStatement internalASTStatement2 = null;
}
   :
      kwtrap:TRAP
      internalASTPatternBind = overturePatternBind
      kwwith:WITH
      internalASTStatement1 = overtureStatement
      kwin:IN
      internalASTStatement2 = overtureStatement
      {
       InternalASTKeyword keywordTrap = new InternalASTKeyword(kwtrap,"Trap");
      InternalASTKeyword keywordWith = new InternalASTKeyword(kwwith,"With");
      InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
      internalASTTrapStatement = new InternalASTTrapStatement(keywordTrap, internalASTPatternBind,
           keywordWith, internalASTStatement1, keywordIn, internalASTStatement2);
      }
  ;

//recursive trap statement = 'tixe', traps, 'in', statement ;
overtureRecursiveTrapStatement returns [InternalASTRecursiveTrapStatement
      internalASTRecursiveTrapStatement]
{
 internalASTRecursiveTrapStatement = null;
 InternalASTTraps internalASTTraps = null;
 InternalASTStatement internalASTStatement = null;
}
   :
      kwtixe:TIXE
      internalASTTraps = overtureTraps
      kwin:IN
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordTixe = new InternalASTKeyword(kwtixe,"Tixe");
      InternalASTKeyword keywordIn = new InternalASTKeyword(kwin,"In");
      internalASTRecursiveTrapStatement = new InternalASTRecursiveTrapStatement(keywordTixe,
           internalASTTraps, keywordIn, internalASTStatement);
      }
  ;

//traps = '{', pattern bind, '|->', statement,
//  { ',', pattern bind, '|->', statement }, '}' ;
overtureTraps returns [InternalASTTraps internalASTTraps]
{
 internalASTTraps = null;
 InternalASTPatternBind internalASTPatternBind = null;
 InternalASTStatement internalASTStatement = null;
 InternalASTTrapsElement internalASTTrapsElement = null;
 List<InternalASTTrapsElement> trapsElements = new ArrayList<InternalASTTrapsElement>();
}
   :
      kwlb:LBRACE
      internalASTPatternBind = overturePatternBind
      kwvbararrow:VBARARROW
      internalASTStatement = overtureStatement
      {
       InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow,"VBarArrow");
        internalASTTrapsElement = new InternalASTTrapsElement(internalASTPatternBind,
           keywordVBarArrow, internalASTStatement);
        trapsElements.add(internalASTTrapsElement);
      }
      (
       kwcomma:COMMA
       internalASTPatternBind = overturePatternBind
       kwvbararrow2:VBARARROW
       internalASTStatement = overtureStatement
```

```
        {
          InternalASTKeyword keywordComma = new InternalASTKeyword(kwcomma,"Comma");
           InternalASTKeyword keywordVBarArrow = new InternalASTKeyword(kwvbararrow2,"VBarArrow");
           internalASTTrapsElement = new InternalASTTrapsElement(keywordComma,
                internalASTPatternBind, keywordVBarArrow, internalASTStatement);
           trapsElements.add(internalASTTrapsElement);
        }
       )*
       kwrb:RBRACE
       {
        InternalASTKeyword keywordLeftBrace = new InternalASTKeyword(kwlb,"LeftBrace");
        InternalASTKeyword keywordRightBrace = new InternalASTKeyword(kwrb,"RightBrace");
        internalASTTraps = new InternalASTTraps(keywordLeftBrace, trapsElements, keywordRightBrace);
       }
 ;

//exit statement = 'exit', [ expression ] ;
overtureExitStatement returns [InternalASTExitStatement internalASTExitStatement]
{
 internalASTExitStatement = null;
 InternalASTExpression internalASTExpression = null;
}
   :
       kwexit:EXIT
       (
         internalASTExpression = overtureExpression
       )?
       {
        InternalASTKeyword keywordExit = new InternalASTKeyword(kwexit,"Exit");
       internalASTExitStatement = new InternalASTExitStatement(keywordExit, internalASTExpression);
       }
 ;

//error statement = 'error' ;
overtureErrorStatement returns [InternalASTErrorStatement internalASTErrorStatement]
{
 internalASTErrorStatement = null;
}
   :
       kwerror:ERROR
       {
        InternalASTKeyword keywordError = new InternalASTKeyword(kwerror,"Error");
       internalASTErrorStatement = new InternalASTErrorStatement(keywordError);
       }
 ;

//identity statement = 'skip' ;
overtureIdentityStatement returns [InternalASTIdentityStatement internalASTIdentityStatement]
{
 internalASTIdentityStatement = null;
}
   :
       kwskip:SKIP
       {
        InternalASTKeyword keywordSkip = new InternalASTKeyword(kwskip,"Skip");
       internalASTIdentityStatement = new InternalASTIdentityStatement(keywordSkip);
       }
 ;

// PATTERNS AND BINDINGS!!!

overturePattern returns [InternalASTPattern internalASTPattern]
{
 internalASTPattern = null;
}
 :
   (overtureSetUnionPattern)=> internalASTPattern = overtureSetUnionPattern
   | internalASTPattern = overturePatternB
 ;

// overture pattern - used in left side of pattern union expression
overturePatternB returns [InternalASTPattern internalASTPattern]
{
 internalASTPattern = null;
}
 :
   (overtureSeqConcPattern) => internalASTPattern = overtureSeqConcPattern
   | internalASTPattern = overturePatternC
 ;

overturePatternC returns [InternalASTPattern internalASTPattern]
{
 internalASTPattern = null;
}
```

```
  :
   internalASTPattern = overturePatternIdentifier
   | internalASTPattern = overtureMatchValue
   | internalASTPattern = overtureSetEnumPattern
   | internalASTPattern = overtureSeqEnumPattern
   | internalASTPattern = overtureTuplePattern
   | internalASTPattern = overtureRecordPattern
  ;

overturePatternIdentifier returns [InternalASTPatternIdentifier internalASTPatternIdentifier]
{
 internalASTPatternIdentifier = null;
}
  :
   id: IDENTIFIER
    {
     internalASTPatternIdentifier = new InternalASTPatternIdentifierIdentifier(new InternalASTIdentifier(id));
    }
   |
   li: MINUS
    {
     internalASTPatternIdentifier = new InternalASTPatternIdentifierLine(new InternalASTKeyword(li, "Minus"));
    }
  ;

overtureMatchValue returns [InternalASTMatchValue internalASTMatchValue]
{
 internalASTMatchValue = null;
 InternalASTExpression expression = null;
 InternalASTSymbolicLiteral literal = null;
}
  :
   (
    lb:LBRACKET
    expression = overtureExpression
    rb:RBRACKET
     {
      InternalASTKeyword leftbracket = new InternalASTKeyword(lb, "LeftBracket");
      InternalASTKeyword rightbracket = new InternalASTKeyword(rb, "RightBracket");
      internalASTMatchValue = new InternalASTMatchValueExpression(leftbracket, expression, rightbracket);
     }
   )
   |
    literal = overtureSymbolicLiteral
     {
      internalASTMatchValue = new InternalASTMatchValueSymbolicLiteral(literal);
     }
  ;

overtureSetEnumPattern returns [InternalASTSetEnumPattern internalASTSetEnumPattern]
{
 internalASTSetEnumPattern = null;
 InternalASTPatternList patternlist = null;
}
  :
   lb: LBRACE
   ( patternlist = overturePatternList )?
   rb: RBRACE
    {
     InternalASTKeyword leftbrace = new InternalASTKeyword(lb, "LeftBrace");
     InternalASTKeyword rightbrace = new InternalASTKeyword(rb, "RightBrace");
     internalASTSetEnumPattern = new InternalASTSetEnumPattern(leftbrace, patternlist, rightbrace);
    }
  ;

overtureSetUnionPattern returns [InternalASTSetUnionPattern internalASTSetUnionPattern]
{
 internalASTSetUnionPattern = null;
 InternalASTPattern pattern1 = null;
 InternalASTPattern pattern2 = null;
}
  :
   pattern1 = overturePatternB
   un: UNION
   pattern2 = overturePattern
    {
     InternalASTKeyword keywordunion = new InternalASTKeyword(un, "Union");
     internalASTSetUnionPattern = new InternalASTSetUnionPattern(pattern1, keywordunion, pattern2);
    }
```

```
 ;


overtureSeqEnumPattern returns [InternalASTSeqEnumPattern internalASTSeqEnumPattern]
{
 internalASTSeqEnumPattern = null;
 InternalASTPatternList patternlist = null;
}
 :
  lbb: LBRACK
  ( patternlist = overturePatternList )?
  rbb: RBRACK
   {
    InternalASTKeyword keywordLeftBrack = new InternalASTKeyword(lbb, "LeftBrack");
    InternalASTKeyword keywordRightBrack = new InternalASTKeyword(rbb, "RightBrack");
    internalASTSeqEnumPattern = new InternalASTSeqEnumPattern(keywordLeftBrack, patternlist,
         keywordRightBrack);
   }
 ;

overtureSeqConcPattern returns [InternalASTSeqConcPattern internalASTSeqConcPattern]
{
 internalASTSeqConcPattern = null;
 InternalASTPattern pattern1 = null;
 InternalASTPattern pattern2 = null;
}
 :
  pattern1 = overturePatternC
  ha: HAT
  pattern2 = overturePatternB
   {
    InternalASTKeyword keywordhat = new InternalASTKeyword(ha, "Hat");
    internalASTSeqConcPattern = new InternalASTSeqConcPattern(pattern1, keywordhat, pattern2);
   }
 ;

overtureTuplePattern returns [InternalASTTuplePattern internalASTTuplePattern]
{
 internalASTTuplePattern = null;
 InternalASTPattern pattern = null;
 InternalASTPatternList patternlist = null;
}
 :
  mk: MAKE
  lb: LBRACKET
  pattern = overturePattern
  co: COMMA
  patternlist = overturePatternList
  rb: RBRACKET
   {
    InternalASTKeyword keywordMk = new InternalASTKeyword(mk, "Mk");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
    InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
    internalASTTuplePattern = new InternalASTTuplePattern(keywordMk, keywordLeftBracket, pattern,
         keywordComma, patternlist, keywordRightBracket);
   }
 ;

overtureRecordPattern returns [InternalASTRecordPattern internalASTRecordPattern]
{
 internalASTRecordPattern = null;
 InternalASTName name = null;
 InternalASTPatternList patternlist = null;
}
 :
  mk: MAKE
  name = overtureName
  lb: LBRACKET
  (
   patternlist = overturePatternList
  )?
  rb: RBRACKET
   {
    InternalASTKeyword keywordMk = new InternalASTKeyword(mk, "Mk");
    InternalASTKeyword keywordLeftBracket = new InternalASTKeyword(lb, "LeftBracket");
    InternalASTKeyword keywordRightBracket = new InternalASTKeyword(rb, "RightBracket");
    internalASTRecordPattern = new InternalASTRecordPattern(keywordMk, name, keywordLeftBracket,
         patternlist, keywordRightBracket);
   }
 ;

overturePatternList returns [InternalASTPatternList internalASTPatternList]
{
```

```
 internalASTPatternList = null;
 List<InternalASTPatternListElement> patternListElements = new ArrayList<
      InternalASTPatternListElement>();
 InternalASTPatternListElement currentElement = null;
 InternalASTPattern currentpattern = null;
}
 :
  (
   currentpattern = overturePattern
    {
     currentElement = new InternalASTPatternListElement(currentpattern);
     patternListElements.add(currentElement);
    }
   (
    co: COMMA
    currentpattern = overturePattern
     {
      InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
      currentElement = new InternalASTPatternListElement(keywordComma, currentpattern);
      patternListElements.add(currentElement);
     }
   )*
    {
     internalASTPatternList = new InternalASTPatternList(patternListElements);
    }
  )

 ;

// BINDINGS

overturePatternBind returns [InternalASTPatternBind internalASTPatternBind]
{
 internalASTPatternBind = null;
}
 :
  (overtureBind)=> internalASTPatternBind = overtureBind
  |
  internalASTPatternBind = overturePattern
 ;

overtureBind returns [InternalASTBind internalASTBind]
{
 internalASTBind = null;
}
 :
  (overtureSetBind)=> internalASTBind = overtureSetBind
  |
  internalASTBind = overtureTypeBind
 ;

overtureSetBind returns [InternalASTSetBind internalASTSetBind]
{
 internalASTSetBind = null;
 InternalASTPattern pattern = null;
 InternalASTExpression expression = null;
}
 :
  pattern = overturePattern
  in: IN
  set: SET
  expression = overtureExpression
   {
    InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
    InternalASTKeyword keywordSet = new InternalASTKeyword(set, "Set");
    internalASTSetBind = new InternalASTSetBind(pattern, keywordIn, keywordSet, expression);
   }
 ;

overtureTypeBind returns [InternalASTTypeBind internalASTTypeBind]
{
 internalASTTypeBind = null;
 InternalASTPattern pattern = null;
 InternalASTType type = null;
}
 :
  pattern = overturePattern
  co: COLON
  type = overtureType
   {
    InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
    internalASTTypeBind = new InternalASTTypeBind(pattern, keywordColon, type);
   }
 ;
```

```

overtureBindList returns [InternalASTBindList internalASTBindList]
{
 internalASTBindList = null;
 List<InternalASTBindListElement> bindListElements = new ArrayList<InternalASTBindListElement>();
 InternalASTBindListElement currentElement = null;
 InternalASTMultipleBind multipleBind = null;
}
 :
  multipleBind = overtureMultipleBind
   {
    currentElement = new InternalASTBindListElement(multipleBind);
    bindListElements.add(currentElement);
   }
  (
   co: COMMA
   multipleBind = overtureMultipleBind
    {
     InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
     currentElement = new InternalASTBindListElement(keywordComma, multipleBind);
     bindListElements.add(currentElement);
    }
  )*
   {
    internalASTBindList = new InternalASTBindList(bindListElements);
   }
 ;

overtureMultipleBind returns [InternalASTMultipleBind internalASTMultipleBind]
{
 internalASTMultipleBind = null;
}
 :
  (overtureMultipleTypeBind)=> internalASTMultipleBind = overtureMultipleTypeBind
  |
  internalASTMultipleBind = overtureMultipleSetBind
 ;

overtureMultipleSetBind returns [InternalASTMultipleSetBind internalASTMultipleSetBind]
{
 internalASTMultipleSetBind = null;
 InternalASTPatternList patternList = null;
 InternalASTExpression expression = null;
}
 :
  patternList = overturePatternList
  in: IN
  set: SET
  expression = overtureExpression
   {
    InternalASTKeyword keywordIn = new InternalASTKeyword(in, "In");
    InternalASTKeyword keywordSet = new InternalASTKeyword(set, "Set");
    internalASTMultipleSetBind = new InternalASTMultipleSetBind(patternList, keywordIn,
         keywordSet, expression);
   }
 ;

overtureMultipleTypeBind returns [InternalASTMultipleTypeBind internalASTMultipleTypeBind]
{
 internalASTMultipleTypeBind = null;
 InternalASTPatternList patternList = null;
 InternalASTType type = null;
}
 :
  patternList = overturePatternList
  co: COLON
  type = overtureType
   {
    InternalASTKeyword keywordColon = new InternalASTKeyword(co, "Colon");
    internalASTMultipleTypeBind = new InternalASTMultipleTypeBind(patternList, keywordColon, type
         );
   }
 ;

overtureTypeBindList returns [InternalASTTypeBindList internalASTTypeBindList]
{
 internalASTTypeBindList = null;
 List<InternalASTTypeBindListElement> typeBindListElements = new ArrayList<
      InternalASTTypeBindListElement>();
 InternalASTTypeBindListElement currentElement = null;
 InternalASTTypeBind typeBind = null;
}
 :
  typeBind = overtureTypeBind
```

```
    {
     currentElement = new InternalASTTypeBindListElement(typeBind);
     typeBindListElements.add(currentElement);
    }
   (
    co: COMMA
    typeBind = overtureTypeBind
     {
      InternalASTKeyword keywordComma = new InternalASTKeyword(co, "Comma");
      currentElement = new InternalASTTypeBindListElement(keywordComma, typeBind);
      typeBindListElements.add(currentElement);
     }
   )*
   {
    internalASTTypeBindList = new InternalASTTypeBindList(typeBindListElements);
   }
  ;

// MISC

// type variable identifier = '@', identifier;
overtureTypeVariableIdentifier returns [InternalASTTypeVariableIdentifier
     internalASTTypeVariableIdentifier]
{
 internalASTTypeVariableIdentifier = null;
 InternalASTKeyword keywordAt = null;
}
  :
   at:AT
    {
     keywordAt = new InternalASTKeyword(at, "At");
    }
   id:IDENTIFIER
    {
     InternalASTIdentifier identifier = new InternalASTIdentifier(id);
     internalASTTypeVariableIdentifier = new InternalASTTypeVariableIdentifier(keywordAt,
          identifier);
    }
  ;

//is basic type = 'is_', ( 'bool' | 'nat' | 'nat1' | 'int' | 'rat'
//      | 'real' | 'char' | 'token' ) ;
overtureIsBasicType returns [InternalASTIsBasicType internalASTIsBasicType]
{
 internalASTIsBasicType = null;
}
  :
   bo:ISUBOOL
    {
     InternalASTKeyword keyword = new InternalASTKeyword(bo, "IsUBool");
     internalASTIsBasicType = new InternalASTIsBasicTypeBool(keyword);
    }
   | na:ISUNAT
    {
     InternalASTKeyword keyword = new InternalASTKeyword(na, "IsUNat");
     internalASTIsBasicType = new InternalASTIsBasicTypeNat(keyword);
    }
   | na1:ISUNAT1
    {
     InternalASTKeyword keyword = new InternalASTKeyword(na1, "IsUNat1");
     internalASTIsBasicType = new InternalASTIsBasicTypeNat1(keyword);
    }
   | in:ISUINT
    {
     InternalASTKeyword keyword = new InternalASTKeyword(in, "IsUInt");
     internalASTIsBasicType = new InternalASTIsBasicTypeInt(keyword);
    }
   | ra:ISURAT
    {
     InternalASTKeyword keyword = new InternalASTKeyword(ra, "IsURat");
     internalASTIsBasicType = new InternalASTIsBasicTypeRat(keyword);
    }
   | re:ISUREAL
    {
     InternalASTKeyword keyword = new InternalASTKeyword(re, "IsUReal");
     internalASTIsBasicType = new InternalASTIsBasicTypeReal(keyword);
    }
   | ch:ISUCHAR
    {
     InternalASTKeyword keyword = new InternalASTKeyword(ch, "IsUChar");
     internalASTIsBasicType = new InternalASTIsBasicTypeChar(keyword);
    }
   | to:ISUTOKEN
    {
```

```
     InternalASTKeyword keyword = new InternalASTKeyword(to, "IsUToken");
     internalASTIsBasicType = new InternalASTIsBasicTypeToken(keyword);
    }
  ;


overtureSymbolicLiteral returns [InternalASTSymbolicLiteral internalASTSymbolicLiteral]
{
 internalASTSymbolicLiteral = null;
}
  :
   internalASTSymbolicLiteral = overtureNumericLiteral
   | internalASTSymbolicLiteral = overtureBooleanLiteral
   | internalASTSymbolicLiteral = overtureNilLiteral
   | internalASTSymbolicLiteral = overtureCharacterLiteral
   | internalASTSymbolicLiteral = overtureTextLiteral
   | internalASTSymbolicLiteral = overtureQuoteLiteral
  ;

overtureNumericLiteral returns [InternalASTNumericLiteral internalASTNumericLiteral]
{
 internalASTNumericLiteral = null;
 InternalASTNumeral numeral1 = null;
 InternalASTKeyword keywordDot = null;
 InternalASTNumeral numeral2 = null;
 InternalASTNumeral numeral3 = null;
 InternalASTExponent exponent = null;
 InternalASTExponentElement1 element1 = null;
 InternalASTExponentElement2 element2 = null;
}
  :
   numeral1 = overtureNumeral
   (options {greedy=true;}:
    dot:DOT
     {
      keywordDot = new InternalASTKeyword(dot, "Dot");
     }
    numeral2 = overtureNumeral
   )?
   (
    (
     e1:"E"
      {
       InternalASTKeyword keyword = new InternalASTKeyword(e1, "E");
       element1 = new InternalASTExponentLowerCaseE(keyword);
      }
      | e2:"e"
       {
        InternalASTKeyword keyword = new InternalASTKeyword(e2, "e");
       element1 = new InternalASTExponentUpperCaseE(keyword);
      }
    )
    (
     pl:PLUS
      {
       InternalASTKeyword keyword = new InternalASTKeyword(pl, "Plus");
       element2 = new InternalASTExponentPlus(keyword);
      }
     | mi:MINUS
      {
       InternalASTKeyword keyword = new InternalASTKeyword(mi, "Minus");
       element2 = new InternalASTExponentMinus(keyword);
      }
    )?
    numeral3 = overtureNumeral
     {
      exponent = new InternalASTExponent(element1, element2, numeral3);
     }
   )?

   {
    internalASTNumericLiteral = new InternalASTNumericLiteral(numeral1, keywordDot, numeral2,
         exponent);
   }
  ;

overtureNumeral returns [InternalASTNumeral internalASTNumeral]
{
 internalASTNumeral = null;
}
  :
   nu:NUMERAL
    {
     internalASTNumeral = new InternalASTNumeral(nu);
```

```
    }
  ;


overtureBooleanLiteral returns [InternalASTBooleanLiteral internalASTBooleanLiteral]
{
 internalASTBooleanLiteral = null;
}
  :
   tr:TRUE
    {
     internalASTBooleanLiteral = new InternalASTBooleanLiteralTrue(new InternalASTKeyword(tr, "
          True"));
    }
   |
   fa:FALSE
    {
     internalASTBooleanLiteral = new InternalASTBooleanLiteralFalse(new InternalASTKeyword(fa, "
          False"));
    }
  ;

overtureNilLiteral returns [InternalASTNilLiteral internalASTNilLiteral]
{
 internalASTNilLiteral = null;
}
  :
   ni:NIL
    {
     internalASTNilLiteral = new InternalASTNilLiteral(new InternalASTKeyword(ni, "Nil"));
    }
  ;

overtureCharacterLiteral returns [InternalASTCharacterLiteral internalASTCharacterLiteral]
{
 internalASTCharacterLiteral = null;
}
  :
   chlit:CHARACTERLITERAL
    {
     internalASTCharacterLiteral = new InternalASTCharacterLiteral(chlit);
    }
  ;

overtureTextLiteral returns [InternalASTTextLiteral internalASTTextLiteral]
{
 internalASTTextLiteral = null;
}
  :
   tlit:TEXTLITERAL
    {
     internalASTTextLiteral = new InternalASTTextLiteral(tlit);
    }
  ;

overtureQuoteLiteral returns [InternalASTQuoteLiteral internalASTQuoteLiteral]
{
 internalASTQuoteLiteral = null;
}
  :
   qlit:QUOTELITERAL
    {
     internalASTQuoteLiteral = new InternalASTQuoteLiteral(qlit);
    }
  ;



//----------------------------------------------------------------------------
// The Overture Scanner
//----------------------------------------------------------------------------
class OvertureInternalLexer extends Lexer;

options {
 exportVocab=Overture;  // call the vocabulary "Overture"
 testLiterals=true;     // don't automatically test for literals
 k=5;                   // three characters of lookahead
 charVocabulary = '\0'..'\377';
 //charVocabulary='\u0000'..'\u7FFE';
 // without inlining some bitset tests, couldn't do unicode;
 // I need to make ANTLR generate smaller bitsets; see
 // bottom of JavaLexer.java
 codeGenBitsetTestThreshold=20;
  // '\''
```

```
}

 ACT        : "#act" ;
 ACTIVE      : "#active" ;
 FIN        : "#fin" ;
 REQ        : "#req" ;
 WAITING      : "#waiting";

 COLON       : ':'   ;
 DOUBLECOLON    : "::"   ;
 COLONLINE     : ":-"   ;

 ASTERIX      : '*'   ;
 VBAR       : '|'   ;
 LBRACK           : '['   ;
 RBRACK           : ']'   ;

 LINEARROW     : "->"   ;

 LBRACKET     : '('   ;
 RBRACKET     : ')'   ;
 LBRACE      : '{'   ;
 RBRACE      : '}'   ;

 SEMICOLON      : ';'   ;
 COMMA      : ','   ;
 DOTS       : "..."   ;
 TILDE      : '~'   ;
 DOT       : '.'   ;
 DOTSHARP     : ".#"   ;
 COLONEQUAL     : ":="   ;
 VBARS      : "||"   ;

 EQUALSIGN     : '='   ;
 NOTEQUAL     : "<>"   ;
 APPROX      : "~="   ;
 DOUBLEEQUAL    : "=="   ;

 HAT        : '^'   ;

 AT        : '@'   ;
 MARK       : '`'   ;
 PLUS       : '+'   ;
 MINUS      : '-'   ;

 DOUBLEPLUS     : "++"   ;
 ITERATE      : "**"   ;

 LESSTHANCOLON   : "<:"   ;
 LESSTHANLINECOLON : "<-:"   ;
 COLONGREATERTHAN : ":>"   ;
 COLONLINEGREATERTHAN : ":->" ;

 PLUSARROW     : "+>"   ;

 SLASH      : '/'   ;
 BACKSLASH     : '\\'   ;

 LESSTHAN     : '<'   ;
 LESSTHANEQUAL   : "<="   ;
 GREATERTHAN    : '>'   ;
 GREATERTHANEQUAL : ">="   ;
 EQUALIMPLY     : "==>"   ;

 LOGICALEQUIVALENCE : "<=>"   ;
 IMPLY      : "=>"   ;

 VBARARROW     : "|->"   ;
 ANDSIGN      : '&'   ;



// an identifier.  Note that testLiterals is set to true!  This means
// that after we match the rule, we look in the literals table to see
// if it's a literal or really an identifer
IDENTIFIER
 options {testLiterals=true;}
 :
  ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z' | '0'..'9' | '_')*
 ;

QUOTELITERAL
 :
  '<' ('a'..'z'|'A'..'Z') ('a'..'z'|'A'..'Z'|'_'|'0'..'9')* '>'
```

```
  ;

NUMERAL
  :
   ('0'..'9')+
  ;

CHARACTERLITERAL
  :
   '\''
   (
    ('a'..'z'|'A'..'Z')
    | ESCAPESEQUENCE
    |     ('<' '='
        | '>' '='
        | '<' '>'
        | '-' '>'
        | '+' '>'
        | '=' '=' '>'
        | '|' '|'
        | '=' '>'
        | '<' '=' '>'
        | '|' '-' '>'
        | '<' ':'
        | ':' '>'
        | '<' '-' ':'
        | ':' '-' '>'
        | '=' '='
        | '*' '*'
        | '+' '+'
        )
   )
   '\''
  ;



ESCAPESEQUENCE
  :
   '\\' '\\'
   | '\\' 'r'
   | '\\' 'n'
   | '\\' 't'
   | '\\' 'f'
   | '\\' 'e'
   | '\\' 'a'
   | '\\' 'x' '0'..'9' '0'..'9'
   | '\\' 'c' ('a'..'z'|'A'..'Z')
   | '\\' '0'..'7' '0'..'7' '0'..'7'
   | '\\' '\"'
   | '\\' '\''
  ;


TEXTLITERAL
  :
   '\"'
   (
    ( '\"' '\"' )
    | ('a'..'z'|'A'..'Z')
    | ESCAPESEQUENCE
   )*
   '\"'
  ;




// Single-line comments
SL_COMMENT
  : "--"
   (~('\n'|'\r'))*
  ;


// Whitespace -- ignored
WS
  options {testLiterals=true;}
  :
  ( ' '
   | '\t'
   | '\f'
    // handle newlines
```

```
  | ( options {generateAmbigWarnings=false;}
   : "\r\n"  // Evil DOS
   | '\r'    // Macintosh
   | '\n'    // Unix (the right way)
   )
   { newline(); }
  )+
  { _ttype = Token.SKIP; }
 ;
```

## 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

```java
package org.overturetool.eclipse.ast;

import java.util.List;


public interface ASTNode {
  public void accept(OvertureVisitorOneArg visitor);
  public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A a);
  public void getContextInfo();
  public List<ASTAdditionalInformation> getAdditionalInformation();
  public int getStartLine();
  public int getStartColumn();
  public int getEndLine();
}
```

### I.2.2 InternalASTNode

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

```java
package org.overturetool.eclipse.internal.ast;

import java.util.ArrayList;
import java.util.List;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTAdditionalInformation;
import org.overturetool.eclipse.ast.ASTNode;

import antlr.Token;

public abstract class InternalASTNode implements ASTNode {
  private int startLine = -1;
  private int startColumn = -1;
  private int endLine = -1;
    private List<ASTAdditionalInformation> astAdditionalInformation = new ArrayList<
        ASTAdditionalInformation>();

  public InternalASTNode(){

  }

  public InternalASTNode(Element xmlTok){
    try{
      Element xmlPosition = xmlTok.getChild("OMLPosition");
      setStartLine(new Integer(xmlPosition.getAttributeValue("startLine")));
      setStartColumn(new Integer(xmlPosition.getAttributeValue("startColumn")));
      setEndLine(new Integer(xmlPosition.getAttributeValue("endLine")));
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      System.err.println("nullpointer in InternalASTNode");
      e.printStackTrace();
    }
  }

  public List<ASTAdditionalInformation> getAdditionalInformation() {
    return astAdditionalInformation;
  }

  public void setAdditionalInformation(
```

```
      List<ASTAdditionalInformation> astAdditionalInformation) {
    this.astAdditionalInformation = astAdditionalInformation;
  }

  public int getStartLine() {
    return startLine;
  }

  public void setStartLine(int linenr) {
    startLine = linenr;
  }

  public int getStartColumn() {
    return startColumn;
  }

  public void setStartColumn(int linenr) {
    startColumn = linenr;
  }

  public int getEndLine() {
    return endLine;
  }

  public void setEndLine(int linenr) {
    endLine = linenr;
  }

  public void setPositionsFromXMLToken(Element xmlTok){
    try{
      setStartLine(new Integer(xmlTok.getAttributeValue("startLine")));
      setStartColumn(new Integer(xmlTok.getAttributeValue("startColumn")));
      setEndLine(new Integer(xmlTok.getAttributeValue("endLine")));
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode1");
    }
  }

  public void setStartPositionFromToken(Token tok){
    try{
      setStartLine(tok.getLine());
      setStartColumn(tok.getColumn());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode2");
    }
  }

  public void setStartPositionFromNode(ASTNode node){
    try {
      setStartLine(node.getStartLine());
      setStartColumn(node.getStartColumn());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode3");
    }
  }

  public void setEndPositionFromToken(Token tok){
    try {
      setEndLine(tok.getLine());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode4");
    }
  }

  public void setEndPositionFromNode(ASTNode node){
    try {
      setEndLine(node.getEndLine());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode5");
    }
  }

  public void setPositionsFromToken(Token tok){
    try{
      setStartLine(tok.getLine());
      setStartColumn(tok.getColumn());
      setEndLine(tok.getLine());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
```

```
      //System.err.println("nullpointer in InternalASTNode6");
    }
  }

  public void setPositionsFromNode(ASTNode node){
    try{
      setStartLine(node.getStartLine());
      setStartColumn(node.getStartColumn());
      setEndLine(node.getEndLine());
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      //System.err.println("nullpointer in InternalASTNode7");
    }
  }



  public void getContextInfo(){
  }
}
```

### I.2.3 ASTNodeWithComments

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

```
package org.overturetool.eclipse.ast;


public interface ASTNodeWithComments extends ASTNode{
  public ASTComments getComments();
  public boolean hasComments();
}
```

### I.2.4 InternalASTNodeWithComments

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

```
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTNodeWithComments;

import antlr.Token;

public abstract class InternalASTNodeWithComments extends InternalASTNode implements
     ASTNodeWithComments {
  InternalASTComments comments = new InternalASTComments();

  public InternalASTNodeWithComments(){
  }

  public InternalASTNodeWithComments(Element xmlTok){
    super(xmlTok);
    try{
      Element xmlComments = xmlTok.getChild("OMLComments");
      if(xmlComments!=null) comments.addCommentsFromXMLToken(xmlComments);
    } catch (NullPointerException e){
      //NO ACTION! PROBABLY PARSE ERROR!!!
      System.err.println("nullpointer in InternalASTNodeWithComments");
    }
  }

  public InternalASTNodeWithComments(Token tok){
    setPositionsFromToken(tok);
    addCommentsFromToken(tok);
  }

  public void addCommentsFromToken(antlr.CommonHiddenStreamToken token){
    comments.addCommentsFromToken(token);
  }
```

```

  public void addCommentsFromToken(antlr.Token token){
    this.addCommentsFromToken((antlr.CommonHiddenStreamToken) token);
  }

  public InternalASTComments getComments(){
    return comments;
  }

  public boolean hasComments(){
    return (!comments.isEmpty());
  }

  public InternalASTComment getLastComment(){
    return comments.getLastComment();
  }

}
```

### I.2.5 ASTKeyword

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

```
package org.overturetool.eclipse.ast;

public interface ASTKeyword extends ASTNodeWithComments {
  public String getValue();
  public String getID();
}
```

### I.2.6 InternalASTKeyword

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

```
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTKeyword;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

import antlr.Token;

public class InternalASTKeyword extends InternalASTNodeWithComments
    implements ASTKeyword {

  private String value;
  private String id;

  public InternalASTKeyword(Token tok, String id){
    value = tok.getText();
    setPositionsFromToken(tok);
    addCommentsFromToken(tok);
    this.id = id;
  }

  public InternalASTKeyword(Element xmlTok){
    super(xmlTok);
    //id = xmlTok.getName().substring(10,id.length());
    value = xmlTok.getAttributeValue("value");
  }

  public String getValue() {
    return value;
  }

  public void setValue(String value){
    this.value = value;
  }
```

```java

  public String getID(){
    return id;
  }

  public void setID(String id){
    this.id = id;
  }

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

  public <A> void accept(OvertureVisitorTwoArg<A> visitor,A obj) {
    visitor.visit(this,obj);
  }
}
```

### I.2.7 ASTIdentifier

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

```java
/**
 *
 */
package org.overturetool.eclipse.ast;

public interface ASTIdentifier extends ASTNodeWithComments {
  public String getIdentifierName();
}
```

### I.2.8 InternalASTIdentifier

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

```java
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTIdentifier;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

import antlr.Token;

public class InternalASTIdentifier extends InternalASTNodeWithComments implements
    ASTIdentifier {
  private String identifierName;

  public InternalASTIdentifier(Token tok){
    setIdentifierName(tok.getText());
    setPositionsFromToken(tok);
    addCommentsFromToken(tok);
  }

  public InternalASTIdentifier(Element xmlTok){
    super(xmlTok);
    identifierName = xmlTok.getAttributeValue("name");
  }

  public String getIdentifierName() {
    return identifierName;
  }

  public void setIdentifierName(String _identifierName){
    identifierName = _identifierName;
  }


  public void accept(OvertureVisitorOneArg visitor) {
```

```
    visitor.visit(this);
  }

  public <A> void accept(OvertureVisitorTwoArg<A> visitor,A obj) {
    visitor.visit(this,obj);
  }
}
```

### I.2.9 ASTDocument

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

```
package org.overturetool.eclipse.ast;

import java.util.List;

public interface ASTDocument extends ASTNodeWithComments {
  List<? extends ASTClass> getClasses();
}
```

### I.2.10 InternalASTDocument

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

```
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import java.util.ArrayList;
import java.util.List;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTDocument;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

import antlr.CommonHiddenStreamToken;

public class InternalASTDocument extends InternalASTNodeWithComments implements ASTDocument {
  List<InternalASTClass> classes = new ArrayList<InternalASTClass>();

  public InternalASTDocument(List<InternalASTClass> classes){
    this.classes = classes;
    if (!classes.isEmpty()) {
      this.setStartPositionFromNode(classes.get(0));
      this.setEndPositionFromNode(classes.get(classes.size()-1));
    }
  }

  public InternalASTDocument(Element xmlTok){
    super(xmlTok);
  }

  public List<InternalASTClass> getClasses(){
    return classes;
  }

  public void setClasses(List<InternalASTClass> classes){
    this.classes = classes;
  }

  public void addCommentsFromInitialHiddenToken(CommonHiddenStreamToken initialHiddenToken) {
    comments.addCommentsFromInitialHiddenToken(initialHiddenToken);
  }

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

  public <A> void accept(OvertureVisitorTwoArg<A> visitor,A obj) {
    visitor.visit(this,obj);
```

```
  }
}
```

### I.2.11 ASTClass

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

```java
package org.overturetool.eclipse.ast;

public interface ASTClass extends ASTNode {
  public ASTKeyword getKeywordClass();
  public ASTIdentifier getIdentifier();
  public ASTInheritanceClause getInheritanceClause();
  public ASTClassBody getClassBody();
  public ASTKeyword getKeywordEnd();
  public ASTIdentifier getIdentifierEnd();
}
```

### I.2.12 InternalASTClass

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

```java
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTClass;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

public class InternalASTClass extends InternalASTNode implements ASTClass {
  private InternalASTKeyword keywordClass;
  private InternalASTIdentifier identifier;
  private InternalASTInheritanceClause inheritanceClause;
  private InternalASTClassBody classBody;
  private InternalASTKeyword keywordEnd;
  private InternalASTIdentifier identifierEnd;

  public InternalASTClass(){
  }

  public InternalASTClass(InternalASTKeyword keywordClass, InternalASTIdentifier identifier,
      InternalASTInheritanceClause inheritanceClause, InternalASTClassBody classBody,
      InternalASTKeyword keywordEnd, InternalASTIdentifier identifierEnd){
    this.keywordClass = keywordClass;
    this.identifier = identifier;
    this.inheritanceClause = inheritanceClause;
    this.classBody = classBody;
    this.keywordEnd = keywordEnd;
    this.identifierEnd = identifierEnd;
    this.setStartPositionFromNode(keywordClass);
    this.setEndPositionFromNode(identifierEnd);
  }

  public InternalASTClass(Element xmlTok){
    super(xmlTok);
  }

  public InternalASTKeyword getKeywordClass() {
    return keywordClass;
  }

  public void setKeywordClass(InternalASTKeyword keywordClass) {
    this.keywordClass = keywordClass;
  }

  public InternalASTIdentifier getIdentifier() {
    return identifier;
  }

  public void setIdentifier(InternalASTIdentifier identifier){
    this.identifier = identifier;
  }
```

```

  public InternalASTInheritanceClause getInheritanceClause() {
    return inheritanceClause;
  }

  public void setInheritanceClause(InternalASTInheritanceClause inheritanceClause) {
    this.inheritanceClause = inheritanceClause;
  }

  public InternalASTClassBody getClassBody() {
    return classBody;
  }

  public void setClassBody(InternalASTClassBody classBody) {
    this.classBody = classBody;
  }

  public InternalASTKeyword getKeywordEnd() {
    return keywordEnd;
  }

  public void setKeywordEnd(InternalASTKeyword keywordEnd) {
    this.keywordEnd = keywordEnd;
  }

  public InternalASTIdentifier getIdentifierEnd() {
    return identifierEnd;
  }

  public void setIdentifierEnd(InternalASTIdentifier identifierEnd){
    this.identifierEnd = identifierEnd;
  }

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

  public <A> void accept(OvertureVisitorTwoArg<A> visitor,A obj) {
    visitor.visit(this,obj);
  }

}
```

### I.2.13 ASTExpression

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

```
package org.overturetool.eclipse.ast;

public interface ASTExpression extends ASTNode{

}
```

### I.2.14 InternalASTExpression

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

```
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTExpression;

public abstract class InternalASTExpression extends InternalASTFunctionBody implements
    ASTExpression {

  public InternalASTExpression(){
  }

  public InternalASTExpression(Element xmlTok){
    super(xmlTok);
  }

```

```
}
```

### I.2.15 ASTBinaryExpression

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

```
/**
 *
 */
package org.overturetool.eclipse.ast;

public interface ASTBinaryExpression extends ASTExpression {

  public ASTExpression getExpression1();
  public ASTBinaryOperator getBinaryOperator();
  public ASTExpression getExpression2();
}
```

### I.2.16 InternalASTBinaryExpression

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

```
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTBinaryExpression;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

public class InternalASTBinaryExpression extends InternalASTExpression implements
ASTBinaryExpression {

  private InternalASTExpression expression1;
  private InternalASTBinaryOperator binaryOperator;
  private InternalASTExpression expression2;

  public InternalASTBinaryExpression(InternalASTExpression expression1, InternalASTBinaryOperator
        binaryOperator, InternalASTExpression expression2){
    this.expression1 = expression1;
    this.binaryOperator = binaryOperator;
    this.expression2 = expression2;
    this.setStartPositionFromNode(expression1);
    this.setEndPositionFromNode(expression2);
  }

  public InternalASTBinaryExpression(Element xmlTok) {
    super(xmlTok);
  }

  public InternalASTExpression getExpression1() {
    return expression1;
  }

  public void setExpression(InternalASTExpression expression1) {
    this.expression1 = expression1;
  }

  public InternalASTBinaryOperator getBinaryOperator() {
    return binaryOperator;
  }

  public void setBinaryOperator(InternalASTBinaryOperator binaryOperator) {
    this.binaryOperator = binaryOperator;
  }

  public InternalASTExpression getExpression2() {
    return expression2;
```

```
  }

  public void setExpression2(InternalASTExpression expression2) {
    this.expression2 = expression2;
  }

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

  public <A> void accept(OvertureVisitorTwoArg<A> visitor ,A obj) {
    visitor.visit(this,obj);
  }
}
```

### I.2.17 ASTBinaryOperator

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

```
package org.overturetool.eclipse.ast;

public interface ASTBinaryOperator extends ASTNode{

  public ASTKeyword getKeyword();
}
```

### I.2.18 InternalASTBinaryOperator

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

```
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTBinaryOperator;

public abstract class InternalASTBinaryOperator extends InternalASTNode implements
ASTBinaryOperator {

  private InternalASTKeyword keyword;

  public InternalASTBinaryOperator(InternalASTKeyword keyword) {
    this.keyword = keyword;
    this.setPositionsFromNode(keyword);
  }

  public InternalASTBinaryOperator(Element xmlTok) {
    super(xmlTok);
  }

  public InternalASTKeyword getKeyword(){
    return keyword;
  }

  public void setKeyword(InternalASTKeyword keyword){
    this.keyword = keyword;
  }
}
```

### I.2.19 ASTBinaryOperatorArithmeticPlus

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

```
/**
 *
 */
package org.overturetool.eclipse.ast;

public interface ASTBinaryOperatorArithmeticPlus extends ASTBinaryOperator {

}
```

### I.2.20 InternalASTBinaryOperatorArithmeticPlus

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

```
/**
 *
 */
package org.overturetool.eclipse.internal.ast;

import org.jdom.Element;
import org.overturetool.eclipse.ast.ASTBinaryOperatorArithmeticPlus;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;

public class InternalASTBinaryOperatorArithmeticPlus extends InternalASTBinaryOperator implements
ASTBinaryOperatorArithmeticPlus {

  public InternalASTBinaryOperatorArithmeticPlus(InternalASTKeyword operator) {
    super(operator);
  }

  public InternalASTBinaryOperatorArithmeticPlus(Element xmlTok) {
    super(xmlTok);
  }

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

  public <A> void accept(OvertureVisitorTwoArg<A> visitor,A a) {
    visitor.visit(this,a);
  }
}
```

## 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

```java
package org.overturetool.eclipse.ast;


public interface OvertureVisitorTwoArg<A> {

  public void visit(ASTAccessAssignmentDefinition astAccessAssignmentDefinition, A obj);
  public void visit(ASTAccessFunctionDefinition astAcccessFunctionDefinition, A obj);
  public void visit(ASTAccessOperationDefinition astAccessOperationDefinition, A obj);
  public void visit(ASTAccessPrivate astPrivate, A obj);
  public void visit(ASTAccessProtected astProtected, A obj);
  public void visit(ASTAccessPublic astPublic, A obj);
  public void visit(ASTAccessTypeDefinition astAccessTypeDefinition, A obj);
  public void visit(ASTAccessValueDefinition astAccessValueDefinition, A obj);
  public void visit(ASTActExpressionName astActExpressionName, A obj);
  public void visit(ASTActExpressionNameList astActExpressionNameList, A obj);
  public void visit(ASTActiveExpressionName astActiveExpressionName, A obj);
  public void visit(ASTActiveExpressionNameList astActiveExpressionNameList, A obj);
  public void visit(ASTAllExpression astAllExpression, A obj);
  public void visit(ASTAlwaysStatement astAlwaysStatement, A obj);
  public void visit(ASTApply astApply, A obj);
  public void visit(ASTAssignmentDefinition astAssignmentDefinition, A obj);
  public void visit(ASTAssignStatement astAssignStatement, A obj);
  public void visit(ASTBasicTypeBool astBoolType, A obj);
  public void visit(ASTBasicTypeChar astCharType, A obj);
  public void visit(ASTBasicTypeInt astIntType, A obj);
  public void visit(ASTBasicTypeNat astNatType, A obj);
  public void visit(ASTBasicTypeNat1 astNat1Type, A obj);
  public void visit(ASTBasicTypeRat astRatType, A obj);
  public void visit(ASTBasicTypeReal astRealType, A obj);
  public void visit(ASTBasicTypeToken astTokenType, A obj);
  public void visit(ASTBinaryExpression astBinaryExpression, A obj);
  public void visit(ASTBinaryOperatorAnd astBinaryOperatorAnd, A obj);
  public void visit(ASTBinaryOperatorApprox astBinaryOperatorApprox, A obj);
  public void visit(ASTBinaryOperatorArithmeticDivide astBinaryOperatorArithmeticDivide, A obj);
  public void visit(ASTBinaryOperatorArithmeticIntegerDivision
      astBinaryOperatorArithmeticIntegerDivision, A obj);
  public void visit(ASTBinaryOperatorArithmeticMinus astBinaryOperatorArithmeticMinus, A obj);
  public void visit(ASTBinaryOperatorArithmeticMod astBinaryOperatorArithmeticMod, A obj);
  public void visit(ASTBinaryOperatorArithmeticMultiplication
      astBinaryOperatorArithmeticMultiplication, A obj);
  public void visit(ASTBinaryOperatorArithmeticPlus astBinaryOperatorArithmeticPlus, A obj);
  public void visit(ASTBinaryOperatorArithmeticRem astBinaryOperatorArithmeticRem, A obj);
  public void visit(ASTBinaryOperatorComposition astBinaryOperatorComposition, A obj);
  public void visit(ASTBinaryOperatorEqual astBinaryOperatorEqual, A obj);
  public void visit(ASTBinaryOperatorGreaterThan astBinaryOperatorGreaterThan, A obj);
  public void visit(ASTBinaryOperatorGreaterThanOrEqual astBinaryOperatorGreaterThanOrEqual, A
      obj);
  public void visit(ASTBinaryOperatorImply astBinaryOperatorImply, A obj);
  public void visit(ASTBinaryOperatorInSet astBinaryOperatorInSet, A obj);
  public void visit(ASTBinaryOperatorIterate astBinaryOperatorIterate, A obj);
  public void visit(ASTBinaryOperatorLessThan astBinaryOperatorLessThan, A obj);
  public void visit(ASTBinaryOperatorLessThanOrEqual astBinaryOperatorLessThanOrEqual, A obj);
  public void visit(ASTBinaryOperatorLogicalEquivalence astBinaryOperatorLogicalEquivalence, A
      obj);
  public void visit(ASTBinaryOperatorMapDomainRestrictBy astBinaryOperatorMapDomainRestrictBy, A
      obj);
  public void visit(ASTBinaryOperatorMapDomainRestrictTo astBinaryOperatorMapDomainRestrictTo, A
      obj);
  public void visit(ASTBinaryOperatorMapMerge astBinaryOperatorMapMerge, A obj);
  public void visit(ASTBinaryOperatorMapOrSequenceModify astBinaryOperatorMapOrSequenceModify, A
      obj);
  public void visit(ASTBinaryOperatorMapRangeRestrictBy astBinaryOperatorMapRangeRestrictBy, A
      obj);
  public void visit(ASTBinaryOperatorMapRangeRestrictTo astBinaryOperatorMapRangeRestrictTo, A
      obj);
  public void visit(ASTBinaryOperatorNotEqual astBinaryOperatorNotEqual, A obj);
  public void visit(ASTBinaryOperatorNotInSet astBinaryOperatorNotInSet, A obj);
  public void visit(ASTBinaryOperatorOr astBinaryOperatorOr, A obj);
  public void visit(ASTBinaryOperatorProperSubset astBinaryOperatorProperSubset, A obj);
```

```
    public void visit(ASTBinaryOperatorSequenceConcatenate astBinaryOperatorSequenceConcatenate, A obj);
    public void visit(ASTBinaryOperatorSetDifference astBinaryOperatorSetDifference, A obj);
    public void visit(ASTBinaryOperatorSetIntersection astBinaryOperatorSetIntersection, A obj);
    public void visit(ASTBinaryOperatorSetUnion astBinaryOperatorSetUnion, A obj);
    public void visit(ASTBinaryOperatorSubset astBinaryOperatorSubset, A obj);
    public void visit(ASTBindList astBindList, A obj);
    public void visit(ASTBindListElement astBindListElement, A obj);
    public void visit(ASTBlockStatement astBlockStatement, A obj);
    public void visit(ASTBlockStatementElement astBlockStatementElement, A obj);
    public void visit(ASTBooleanLiteralFalse astBooleanLiteralFalse, A obj);
    public void visit(ASTBooleanLiteralTrue astBooleanLiteralTrue, A obj);
    public void visit(ASTBracketedExpression astBracketedExpression, A obj);
    public void visit(ASTBracketedType astBracketedType, A obj);
    public void visit(ASTCallStatement astCallStatement, A obj);
    public void visit(ASTCasesExpression astCasesExpression, A obj);
    public void visit(ASTCasesExpressionAlternative astCasesExpressionAlternative, A obj);
    public void visit(ASTCasesExpressionAlternatives astCasesExpressionAlternatives, A obj);
    public void visit(ASTCasesExpressionAlternativesElement astCasesExpressionAlternativesElement, A obj);
    public void visit(ASTCasesStatement astCasesStatement, A obj);
    public void visit(ASTCasesStatementAlternative astCasesStatementAlternative, A obj);
    public void visit(ASTCasesStatementAlternatives astCasesStatementAlternatives, A obj);
    public void visit(ASTCasesStatementAlternativesElement astCasesStatementAlternativesElement, A obj);
    public void visit(ASTCharacterLiteral astCharacterLiteral, A obj);
    public void visit(ASTClass astClass, A obj);
    public void visit(ASTClassBody astClassBody, A obj);
    public void visit(ASTComment astComment, A obj);
    public void visit(ASTComments astComments, A obj);
    public void visit(ASTCompositeType astCompositeType, A obj);
    public void visit(ASTDclStatement astDclStatement, A obj);
    public void visit(ASTDclStatementElement astDclStatementElement, A obj);
    public void visit(ASTDefExpression astDefExpression, A obj);
    public void visit(ASTDefExpressionElement astDefExpressionElement, A obj);
    public void visit(ASTDefStatement astDefStatement, A obj);
    public void visit(ASTDefStatementElement astDefStatementElement, A obj);
    public void visit(ASTDiscretionaryTypeBrackets astDiscretionaryTypeBrackets, A obj);
    public void visit(ASTDiscretionaryTypeType astDiscretionaryTypeType, A obj);
    public void visit(ASTDocument astDocument, A obj);
    public void visit(ASTElseifExpression astElseIfExpression, A obj);
    public void visit(ASTElseifStatement astElseifStatement, A obj);
    public void visit(ASTEqualsDefinition astEqualsDefinition, A obj);
    public void visit(ASTError astError, A obj);
    public void visit(ASTErrorList astErrorList, A obj);
    public void visit(ASTErrorStatement astErrorStatement, A obj);
    public void visit(ASTExceptions astExceptions, A obj);
    public void visit(ASTExistsExpression astExistsExpression, A obj);
    public void visit(ASTExistsUniqueExpression astExistsUniqueExpression, A obj);
    public void visit(ASTExitStatement astExitStatement, A obj);
    public void visit(ASTExplicitFunctionDefinition astExplicitFunctionDefinition, A obj);
    public void visit(ASTExplicitOperationDefinition astExplicitOperationDefinition, A obj);
    public void visit(ASTExponent astExponent, A obj);
    public void visit(ASTExponentLowerCaseE astExponentLowerCaseE, A obj);
    public void visit(ASTExponentMinus astExponentMinus, A obj);
    public void visit(ASTExponentPlus astExponentPlus, A obj);
    public void visit(ASTExponentUpperCaseE astExponentUpperCaseE, A obj);
    public void visit(ASTExpressionList astExpressionList, A obj);
    public void visit(ASTExpressionListElement astExpressionListElement, A obj);
    public void visit(ASTExpressionSymbolicLiteral astExpressionSymbolicLiteral, A obj);
    public void visit(ASTExtendedExplicitFunctionDefinition astExtendedExplicitFunctionDefinition, A obj);
    public void visit(ASTExtendedExplicitOperationDefinition astExtendedExplicitOperationDefinition, A obj);
    public void visit(ASTExternals astASTExternals, A obj);
    public void visit(ASTFieldBasic astFieldBasic, A obj);
    public void visit(ASTFieldColon astFieldColon, A obj);
    public void visit(ASTFieldColonLine astFieldColonLine, A obj);
    public void visit(ASTFieldList astFieldList, A obj);
    public void visit(ASTFieldReference astFieldReference, A obj);
    public void visit(ASTFieldSelect astFieldSelect, A obj);
    public void visit(ASTFinExpressionName astFinExpressionName, A obj);
    public void visit(ASTFinExpressionNameList astFinExpressionNameList, A obj);
    public void visit(ASTFunctionBodyIsNotYetSpecified astIsNotYetSpecified, A obj);
    public void visit(ASTFunctionBodyIsSubclassResponsibility astIsSubclassResponsibility, A obj);
    public void visit(ASTFunctionDefinitions astFunctionDefinitions, A obj);
    public void visit(ASTFunctionDefinitionsElement astFunctionDefinitionsElement, A obj);
    public void visit(ASTFunctionTypeInstantiation astFunctionTypeInstantiation, A obj);
    public void visit(ASTFunctionTypeInstantiationElement astFunctionTypeInstantiationElement, A obj);
    public void visit(ASTGeneralMapType astGeneralMapType, A obj);
    public void visit(ASTIdentifier astIdentifier, A obj);
    public void visit(ASTIdentifierTypePair astIdentifierTypePair, A obj);
    public void visit(ASTIdentifierTypePairList astIdentifierTypePairList, A obj);
```

```
    public void visit(ASTIdentifierTypePairListElement astIdentifierTypePairListElement, A obj);
    public void visit(ASTIdentityStatement astIdentityStatement, A obj);
    public void visit(ASTIfExpression astIfExpression, A obj);
    public void visit(ASTIfStatement astIfStatement, A obj);
    public void visit(ASTImplicitFunctionDefinition astImplicitFunctionDefinition, A obj);
    public void visit(ASTImplicitOperationBody astImplicitOperationBody, A obj);
    public void visit(ASTImplicitOperationDefinition astImplicitOperationDefinition, A obj);
    public void visit(ASTIndexForLoop astIndexForLoop, A obj);
    public void visit(ASTInheritanceClause astInheritanceClause, A obj);
    public void visit(ASTInitStatement astInitStatement, A obj);
    public void visit(ASTInjectiveMapType astInjectiveMapType, A obj);
    public void visit(ASTInstanceVariableDefinitions astInstanceVariableDefinitions, A obj);
    public void visit(ASTInstanceVariableDefinitionsElement astInstanceVariableDefinitionsElement,
        A obj);
    public void visit(ASTInstvarinitExpression astInstvarinitExpression, A obj);
    public void visit(ASTInvariant astInvariant, A obj);
    public void visit(ASTInvariantDefinition astInvariantDefinition, A obj);
    public void visit(ASTInvariantInitialFunction astInvariantInitialFunction, A obj);
    public void visit(ASTIotaExpression astIotaExpression, A obj);
    public void visit(ASTIsBasicTypeBool astIsBasicTypeBool, A obj);
    public void visit(ASTIsBasicTypeChar astIsBasicTypeChar, A obj);
    public void visit(ASTIsBasicTypeInt astIsBasicTypeInt, A obj);
    public void visit(ASTIsBasicTypeNat astIsBasicTypeNat, A obj);
    public void visit(ASTIsBasicTypeNat1 astIsBasicTypeNat1, A obj);
    public void visit(ASTIsBasicTypeRat astIsBasicTypeRat, A obj);
    public void visit(ASTIsBasicTypeReal astIsBasicTypeReal, A obj);
    public void visit(ASTIsBasicTypeToken astIsBasicTypeToken, A obj);
    public void visit(ASTIsExpressionIsBasicType astIsExpressionIsBasicType, A obj);
    public void visit(ASTIsExpressionIsName astIsExpressionIsName, A obj);
    public void visit(ASTIsofbaseclassExpression astIsofbaseclassExpression, A obj);
    public void visit(ASTIsofclassExpression astIsofclassExpression, A obj);
    public void visit(ASTKeyword astKeyword, A obj);
    public void visit(ASTLetBeExpression astLetBeExpression, A obj);
    public void visit(ASTLetExpression astLetExpression, A obj);
    public void visit(ASTLetExpressionElement astLetExpressionElement, A obj);
    public void visit(ASTLetBeStatement astLetBeStatement, A obj);
    public void visit(ASTLetStatement astLetStatement, A obj);
    public void visit(ASTLetStatementElement astLetStatementElement, A obj);
    public void visit(ASTLambdaExpression astLambdaExpression, A obj);
    public void visit(ASTMapComprehension astMapComprehension, A obj);
    public void visit(ASTMapEnumerationArrow astMapEnumerationArrow, A obj);
    public void visit(ASTMapEnumerationMaplet astMapEnumerationMaplet, A obj);
    public void visit(ASTMapEnumerationMapletElement astMapEnumerationMapletElement, A obj);
    public void visit(ASTMapInverse astMapInverse, A obj);
    public void visit(ASTMaplet astMaplet, A obj);
    public void visit(ASTMapOrSequenceReference astMapOrSequenceReference, A obj);
    public void visit(ASTMatchValueExpression astMatchValueExpression, A obj);
    public void visit(ASTMatchValueSymbolicLiteral astMatchValueSymbolicLiteral, A obj);
    public void visit(ASTModeRd astASTModeRd, A obj);
    public void visit(ASTModeWr astASTModeWr, A obj);
    public void visit(ASTMultipleAssignStatement astMultipleAssignStatement, A obj);
    public void visit(ASTMultipleAssignStatementElement astMultipleAssignStatementElement, A obj);
    public void visit(ASTMultipleSetBind astMultipleSetBind, A obj);
    public void visit(ASTMultipleTypeBind ASTMultipleTypeBind, A obj);
    public void visit(ASTMutexPredicateAll astMutexPredicateAll, A obj);
    public void visit(ASTMutexPredicateNameList astMutexPredicateNameList, A obj);
    public void visit(ASTName astName, A obj);
    public void visit(ASTNameList astNameList, A obj);
    public void visit(ASTNameListElement astNameListElement, A obj);
    public void visit(ASTNewExpression astNewExpression, A obj);
    public void visit(ASTNilLiteral astNilLiteral, A obj);
    public void visit(ASTNondeterministicStatement astNondeterministicStatement, A obj);
    public void visit(ASTNondeterministicStatementElement astNondeterministicStatementElement, A
        obj);
    public void visit(ASTNumeral astNumeral, A obj);
    public void visit(ASTNumericLiteral astNumericLiteral, A obj);
    public void visit(ASTObjectApply astObjectApply, A obj);
    public void visit(ASTObjectDesignatorName astObjectDesignatorName, A obj);
    public void visit(ASTObjectDesignatorNewExpression astObjectDesignatorNewExpression, A obj);
    public void visit(ASTObjectDesignatorSelfExpression astObjectDesignatorSelfExpression, A obj);
    public void visit(ASTObjectFieldReference astObjectFieldReference, A obj);
    public void visit(ASTOldName astOldName, A obj);
    public void visit(ASTOperationBodyIsNotYetSpecified astOperationBodyIsNotYetSpecified, A obj);
    public void visit(ASTOperationDefinitions astOperationDefinitions, A obj);
    public void visit(ASTOperationDefinitionsElement astOperationDefinitionsElement, A obj);
    public void visit(ASTOperationType astOperationType, A obj);
    public void visit(ASTOperationBodyIsSubclassResponsibility
        astOperationBodyIsSubclassResponsibility, A obj);
    public void visit(ASTOptionalType astOptionalType, A obj);
    public void visit(ASTOthersExpression astOthersExpression, A obj);
    public void visit(ASTOthersStatement astOthersStatement, A obj);
    public void visit(ASTParameters astParameters, A obj);
    public void visit(ASTParametersList astParametersList, A obj);
    public void visit(ASTParameterTypes astParameterTypes, A obj);
```

```
    public void visit(ASTPartialFunctionType astPartialFunctionType, A obj);
    public void visit(ASTPatternIdentifierIdentifier astPatternIdentifierIdentifier, A obj);
    public void visit(ASTPatternIdentifierLine astPatternIdentifierLine, A obj);
    public void visit(ASTPatternList astPatternList, A obj);
    public void visit(ASTPatternListElement astPatternListElement, A obj);
    public void visit(ASTPatternListTypePair astPatternListTypePair, A obj);
    public void visit(ASTPatternTypePairList astPatternTypePairList, A obj);
    public void visit(ASTPatternTypePairListElement astPatternTypePairListElement, A obj);
    public void visit(ASTPermissionPredicateElement1 astPermissionPredicatesElement1, A obj);
    public void visit(ASTPermissionPredicateElement2 astPermissionPredicatesElement2, A obj);
    public void visit(ASTPermissionPredicates astPermissionPredicates, A obj);
    public void visit(ASTPermissionPredicatesElement astPermissionPredicatesElement, A obj);
    public void visit(ASTPrefixExpression astPrefixExpression, A obj);
    public void visit(ASTProceduralThreadDefinition astProceduralThreadDefinition, A obj);
    public void visit(ASTProductType astProductType, A obj);
    public void visit(ASTQuoteLiteral astQuoteLitteral, A obj);
    public void visit(ASTQuoteType astQuoteType, A obj);
    public void visit(ASTRecordConstructor astRecordConstructor, A obj);
    public void visit(ASTRecordModifier ASTRecordModifier, A obj);
    public void visit(ASTRecordModifierElement ASTRecordModifierElement, A obj);
    public void visit(ASTRecordModification ASTRecordModification, A obj);
    public void visit(ASTRecordPattern astRecordPattern, A obj);
    public void visit(ASTRecursiveTrapStatement astRecursiveTrapStatement, A obj);
    public void visit(ASTReqExpressionName astReqExpressionName, A obj);
    public void visit(ASTReqExpressionNameList astReqExpressionNameList, A obj);
    public void visit(ASTReturnStatement astReturnStatement, A obj);
    public void visit(ASTSamebaseclassExpression astSamebaseclassExpression, A obj);
    public void visit(ASTSameclassExpression astSameclassExpression, A obj);
    public void visit(ASTSelfExpression astSelfExpression, A obj);
    public void visit(ASTSeq0Type astSeq0Type, A obj);
    public void visit(ASTSeq1Type astSeq1Type, A obj);
    public void visit(ASTSeqConcPattern astSeqConcPattern, A obj);
    public void visit(ASTSeqEnumPattern astSeqEnumPattern, A obj);
    public void visit(ASTSequenceComprehension astSequenceComprehension, A obj);
    public void visit(ASTSequenceEnumeration astSequenceEnumeration, A obj);
    public void visit(ASTSequenceForLoop astSequenceForLoop, A obj);
    public void visit(ASTSetBind astSetBind, A obj);
    public void visit(ASTSetComprehension astSetComprehension, A obj);
    public void visit(ASTSetEnumeration astSetEnumeration, A obj);
    public void visit(ASTSetEnumPattern astSetEnumPattern, A obj);
    public void visit(ASTSetForLoop astSetForLoop, A obj);
    public void visit(ASTSetRangeExpression astSetRangeExpression, A obj);
    public void visit(ASTSetType astSetType, A obj);
    public void visit(ASTSetUnionPattern astSetUnionPattern, A obj);
    public void visit(ASTSpecificationStatement astSpecificationStatement, A obj);
    public void visit(ASTStartListStatement astStartListStatement, A obj);
    public void visit(ASTStartStatement astStartStatement, A obj);
    public void visit(ASTStateDesignatorName astStateDesignatorName, A obj);
    public void visit(ASTSubsequence astSubsequence, A obj);
      public void visit(ASTSynchronization astSynchronization, A obj);
    public void visit(ASTSynchronizationDefinitions astSynchronizationDefinitions, A obj);
    public void visit(ASTTextLiteral astTextLiteral, A obj);
    public void visit(ASTThreadDefinition astThreadDefinition, A obj);
    public void visit(ASTThreadDefinitions astThreadDefinitions, A obj);
    public void visit(ASTThreadidExpression astThreadidExpression, A obj);
    public void visit(ASTTotalFunctionType astTotalFunctionType, A obj);
    public void visit(ASTTraps astTraps, A obj);
    public void visit(ASTTrapsElement astTrapsElement, A obj);
    public void visit(ASTTrapStatement astTrapStatement, A obj);
    public void visit(ASTTupleConstructor astTupleConstructor, A obj);
    public void visit(ASTTuplePattern astTuplePattern, A obj);
    public void visit(ASTTupleSelect astTupleSelect, A obj);
    public void visit(ASTTypeBind astTypeBind, A obj);
    public void visit(ASTTypeBindList astTypeBindList, A obj);
    public void visit(ASTTypeBindListElement astTypeBindListElement, A obj);
    public void visit(ASTTypeDefinitionFieldList astTypeFieldListDefinition, A obj);
    public void visit(ASTTypeDefinitions astTypeDefinitions, A obj);
    public void visit(ASTTypeDefinitionsElement astTypeDefinitionsElement, A obj);
    public void visit(ASTTypeDefinitionType astTypeTypeDefinition, A obj);
    public void visit(ASTTypeJudgement ASTTypeJudgement, A obj);
    public void visit(ASTTypeName astTypeName, A obj);
    public void visit(ASTTypeVariable astTypeVariable, A obj);
    public void visit(ASTTypeVariableIdentifier astTypeVariableIdentifier, A obj);
    public void visit(ASTTypeVariableList astTypeVariableList, A obj);
    public void visit(ASTTypeVariableListElement astTypeVariableListElement, A obj);
    public void visit(ASTUnaryOperatorArithmeticAbs astUnaryOperatorArithmeticAbs, A obj);
    public void visit(ASTUnaryOperatorDistributedMapMerge astUnaryOperatorDistributedMapMerge, A
        obj);
    public void visit(ASTUnaryOperatorDistributedSequenceConcatenation
        astUnaryOperatorDistributedSequenceConcatenation, A obj);
    public void visit(ASTUnaryOperatorDistributedSetIntersection
        astUnaryOperatorDistributedSetIntersection, A obj);
    public void visit(ASTUnaryOperatorDistributedSetUnion astUnaryOperatorDistributedSetUnion, A
        obj);
```

```java
    public void visit(ASTUnaryOperatorFinitePowerSet astUnaryOperatorFinitePowerSet, A obj);
    public void visit(ASTUnaryOperatorFloor astUnaryOperatorFloor, A obj);
    public void visit(ASTUnaryOperatorMapDomain astUnaryOperatorMapDomain, A obj);
    public void visit(ASTUnaryOperatorMapRange astUnaryOperatorMapRange, A obj);
    public void visit(ASTUnaryOperatorNot astUnaryOperatorNot, A obj);
    public void visit(ASTUnaryOperatorSequenceElements astUnaryOperatorSequenceElements, A obj);
    public void visit(ASTUnaryOperatorSequenceHead astUnaryOperatorSequenceHead, A obj);
    public void visit(ASTUnaryOperatorSequenceIndices astUnaryOperatorSequenceIndices, A obj);
    public void visit(ASTUnaryOperatorSequenceLength astUnaryOperatorSequenceLength, A obj);
    public void visit(ASTUnaryOperatorSequenceTail astUnaryOperatorSequenceTail, A obj);
    public void visit(ASTUnaryOperatorSetCardinality astUnaryOperatorSetCardinality, A obj);
    public void visit(ASTUnaryOperatorUnaryMinus astUnaryOperatorUnaryMinus, A obj);
    public void visit(ASTUnaryOperatorUnaryPlus astUnaryOperatorUnaryPlus, A obj);
    public void visit(ASTUnionType astUnionType, A obj);
    public void visit(ASTUndefinedExpression astSelfExpression, A obj);
    public void visit(ASTValueDefinition astValueDefinition, A obj);
    public void visit(ASTValueDefinitions astValueDefinitions, A obj);
    public void visit(ASTValueDefinitionsElement astValueDefinitionsElement, A obj);
    public void visit(ASTVarInformation astVarInformation, A obj);
    public void visit(ASTWaitingExpressionName astWaitingExpressionName, A obj);
    public void visit(ASTWaitingExpressionNameList astWaitingExpressionNameList, A obj);
    public void visit(ASTWhileLoop astWhileLoop, A obj);
}
```

# I.4 Outline Visitor

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

```
/*****************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 *****************************************************************************/

package org.overturetool.eclipse.outline;

import java.util.List;

import org.overturetool.eclipse.ast.*;
import org.overturetool.eclipse.internal.ast.InternalASTDefExpressionElement;
import org.overturetool.eclipse.internal.ast.OvertureVisitorTwoArgImpl;
import org.overturetool.eclipse.util.TreeParent;

public class OvertureOutlineVisitor extends OvertureVisitorTwoArgImpl<TreeParent> implements
     OvertureVisitorTwoArg<TreeParent>{

  public void visit(ASTClass astClass, TreeParent parent) {
    try {
      ASTIdentifier astIdentifier = astClass.getIdentifier();
      TreeParent omlClass = new TreeParent("Class "+astIdentifier.getIdentifierName(),astClass.
           getStartLine(),astClass.getStartColumn(),astClass.getEndLine());
      ASTClassBody classBody = astClass.getClassBody();
      if(classBody != null){
        classBody.accept(this, omlClass);
      }
      parent.addChild(omlClass);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTExplicitFunctionDefinition astExplicitFunctionDefinition, TreeParent
       parent){
    try {
      ASTIdentifier astIdentifier = astExplicitFunctionDefinition.getIdentifier1();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
           getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
      parent.addChild(omlIdentifier);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }


  public void visit(ASTExplicitOperationDefinition astExplicitOperationDefinition, TreeParent
       parent){
    try {
      ASTIdentifier astIdentifier = astExplicitOperationDefinition.getIdentifier1();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
           getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
      parent.addChild(omlIdentifier);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTExtendedExplicitFunctionDefinition astExtendedExplicitFunctionDefinition,
       TreeParent parent){
    try {
      ASTIdentifier astIdentifier = astExtendedExplicitFunctionDefinition.getIdentifier();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
           getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
      parent.addChild(omlIdentifier);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTExtendedExplicitOperationDefinition astExtendedExplicitOperationDefinition
       , TreeParent parent){
    try {
      ASTIdentifier astIdentifier = astExtendedExplicitOperationDefinition.getIdentifier();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
           getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
```

```
        parent.addChild(omlIdentifier);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    public void visit(ASTFunctionDefinitions astFunctionDefinitions, TreeParent parent) {
      try {
        TreeParent omlFunctionDefinitions = new TreeParent("functions",astFunctionDefinitions.getStartLine(),astFunctionDefinitions.getStartColumn(),astFunctionDefinitions.getEndLine());
        for (ASTFunctionDefinitionsElement astFunctionDefinitionsElement : astFunctionDefinitions.getFunctionDefinitionsElements()) {
          astFunctionDefinitionsElement.accept(this, omlFunctionDefinitions);
        }
        parent.addChild(omlFunctionDefinitions);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    public void visit(ASTImplicitFunctionDefinition astImplicitFunctionDefinition, TreeParent parent){
      try {
        ASTIdentifier astIdentifier = astImplicitFunctionDefinition.getIdentifier();
        TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
        parent.addChild(omlIdentifier);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }


    public void visit(ASTImplicitOperationDefinition astImplicitOperationDefinition, TreeParent parent){
      try {
        ASTIdentifier astIdentifier = astImplicitOperationDefinition.getIdentifier();
        TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
        parent.addChild(omlIdentifier);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }


    public void visit(ASTInstanceVariableDefinitions astInstanceVariableDefinitions, TreeParent parent) {
      try {
        TreeParent omlInstanceVariableDefinitions = new TreeParent("instance variables",astInstanceVariableDefinitions.getStartLine(),astInstanceVariableDefinitions.getStartColumn(),astInstanceVariableDefinitions.getEndLine());
        for (ASTInstanceVariableDefinitionsElement astInstanceVariableDefinitionsElement : astInstanceVariableDefinitions.getInstanceVariableDefinitionsElement()) {
          astInstanceVariableDefinitionsElement.accept(this, omlInstanceVariableDefinitions);
        }
        parent.addChild(omlInstanceVariableDefinitions);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }



    public void visit(ASTOperationDefinitions astOperationDefinitions, TreeParent parent){
      try {
        TreeParent omlOperationDefinitions = new TreeParent("operations",astOperationDefinitions.getStartLine(),astOperationDefinitions.getStartColumn(),astOperationDefinitions.getEndLine());
        for (ASTOperationDefinitionsElement astOperationDefinitionsElement : astOperationDefinitions.getOperationDefinitionsElements()) {
          astOperationDefinitionsElement.accept(this, omlOperationDefinitions);
        }
        parent.addChild(omlOperationDefinitions);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }

    public void visit(ASTSynchronizationDefinitions astSynchronizationDefinitions, TreeParent parent) {
      try {
```

```
      TreeParent omlSynchronizationDefinitions = new TreeParent("sync",
          astSynchronizationDefinitions.getStartLine(),astSynchronizationDefinitions.
          getStartColumn(),astSynchronizationDefinitions.getEndLine());
      parent.addChild(omlSynchronizationDefinitions);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTThreadDefinitions astThreadDefinitions, TreeParent parent) {
    try {
      TreeParent omlThreadDefinition = new TreeParent("thread",astThreadDefinitions.getStartLine
          (),astThreadDefinitions.getStartColumn(),astThreadDefinitions.getEndLine());
      parent.addChild(omlThreadDefinition);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTTypeDefinitions astTypeDefinitions, TreeParent parent) {
    try {
      TreeParent omlTypeDefinitions = new TreeParent("types",astTypeDefinitions.getStartLine(),
          astTypeDefinitions.getStartColumn(),astTypeDefinitions.getEndLine());
      for (ASTTypeDefinitionsElement astTypeDefinitionsElement : astTypeDefinitions.
          getTypeDefinitionsElements()) {
        astTypeDefinitionsElement.accept(this, omlTypeDefinitions);
      }
      parent.addChild(omlTypeDefinitions);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTTypeDefinitionFieldList astTypeDefinitionFieldList, TreeParent parent) {
    try {
      ASTIdentifier astIdentifier = astTypeDefinitionFieldList.getIdentifier();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
          getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
      parent.addChild(omlIdentifier);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTTypeDefinitionType astTypeDefinitionType, TreeParent parent) {
    try {
      ASTIdentifier astIdentifier = astTypeDefinitionType.getIdentifier();
      TreeParent omlIdentifier = new TreeParent(astIdentifier.getIdentifierName(),astIdentifier.
          getStartLine(),astIdentifier.getStartColumn(),astIdentifier.getEndLine());
      parent.addChild(omlIdentifier);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTValueDefinitions astValueDefinitions, TreeParent parent) {
    try {
      TreeParent omlValueDefinitions = new TreeParent("values",astValueDefinitions.getStartLine()
          ,astValueDefinitions.getStartColumn(),astValueDefinitions.getEndLine());
      for (ASTValueDefinitionsElement astValueDefinitionsElement : astValueDefinitions.
          getValueDefinitionsElements()) {
        astValueDefinitionsElement.accept(this, omlValueDefinitions);
      }
      parent.addChild(omlValueDefinitions);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

## I.5 Ast2Oml Pretty Print Visitor

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

```
/*****************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 *****************************************************************************/

package org.overturetool.eclipse.ast2oml;

import org.overturetool.eclipse.ast.ASTCharacterLiteral;
import org.overturetool.eclipse.ast.ASTClass;
import org.overturetool.eclipse.ast.ASTComment;
import org.overturetool.eclipse.ast.ASTDocument;
import org.overturetool.eclipse.ast.ASTIdentifier;
import org.overturetool.eclipse.ast.ASTKeyword;
import org.overturetool.eclipse.ast.ASTNode;
import org.overturetool.eclipse.ast.ASTNumeral;
import org.overturetool.eclipse.ast.ASTQuoteLiteral;
import org.overturetool.eclipse.ast.ASTTextLiteral;
import org.overturetool.eclipse.internal.ast.OvertureVisitorOneArgImpl;

public class OvertureAst2OmlVisitor extends OvertureVisitorOneArgImpl{

  private int lineCount = 1;
  private int columnCount = 1;
  private String doc = new String();

  public void addTokenString(String keyword) {
    try {
      doc = doc + keyword;
      columnCount = columnCount + keyword.length();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void setPosition(ASTNode astNode) {
    try {
      int startLine = astNode.getStartLine();
      int startColumn = astNode.getStartColumn();
      while (lineCount < startLine) {
        doc = doc + "\n";
        lineCount++;
        columnCount = 1;
      }
      while (columnCount < startColumn) {
        doc = doc + " ";
        columnCount++;
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTComment astComment) {
    try {
      setPosition(astComment);
      addTokenString(astComment.getComment());
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public String visit(ASTDocument astDocument) {
    try {
      visit(astDocument.getComments());
      for (ASTClass astClass : astDocument.getClasses()) {
        astClass.accept(this);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
    return doc;
  }

  public void visit(ASTIdentifier astIdentifier) {
    try {
```

```java
      setPosition(astIdentifier);
      addTokenString(astIdentifier.getIdentifierName());
      astIdentifier.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTKeyword astKeyword) {
    try {
      setPosition(astKeyword);
      addTokenString(astKeyword.getValue());
      astKeyword.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTNumeral astNumeral) {
    try {
      setPosition(astNumeral);
      addTokenString(astNumeral.getValue());
      astNumeral.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTQuoteLiteral astQuoteLiteral) {
    try {
      setPosition(astQuoteLiteral);
      addTokenString(astQuoteLiteral.getQuoteLitteralName());
      astQuoteLiteral.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTTextLiteral astTextLiteral) {
    try {
      setPosition(astTextLiteral);
      addTokenString("\""+astTextLiteral.getTextLitteralName()+"\"");
      astTextLiteral.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public void visit(ASTCharacterLiteral astCharacterLiteral) {
    try {
      setPosition(astCharacterLiteral);
      addTokenString("\'"+ astCharacterLiteral.getCharacterLitteralName()+"\'");
      astCharacterLiteral.getComments().accept(this);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

## 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.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by jens (student) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified" attributeFormDefault="unqualified">
 <xs:element name="OMLDocument">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="OMLPosition"/>
    <xs:element ref="OMLComments" minOccurs="0"/>
    <xs:sequence maxOccurs="unbounded">
     <xs:element ref="OMLClass"/>
    </xs:sequence>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="OMLClass">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="OMLPosition"/>
    <xs:element name="OMLKeywordClass">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLPosition"/>
       <xs:element ref="OMLComments" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required" fixed="class"/>
     </xs:complexType>
    </xs:element>
    <xs:element ref="OMLIdentifier"/>
    <xs:sequence minOccurs="0">
     <xs:element ref="OMLInheritanceClause"/>
    </xs:sequence>
    <xs:sequence minOccurs="0">
     <xs:element ref="OMLClassBody"/>
    </xs:sequence>
    <xs:element name="OMLKeywordEnd">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLPosition"/>
       <xs:element ref="OMLComments" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required" fixed="end"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="OMLIdentifier2">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLIdentifier"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="OMLInheritanceClause">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="OMLPosition"/>
    <xs:element name="OMLKeywordIs">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLPosition"/>
       <xs:element ref="OMLComments" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required" fixed="is"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="OMLKeywordSubclass">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLPosition"/>
       <xs:element ref="OMLComments" minOccurs="0"/>
      </xs:sequence>
      <xs:attribute name="value" type="xs:string" use="required" fixed="subclass"/>
```

```
    </xs:complexType>
   </xs:element>
   <xs:element name="OMLKeywordOf">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="OMLPosition"/>
      <xs:element ref="OMLComments" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="value" type="xs:string" use="required" fixed="of"/>
    </xs:complexType>
   </xs:element>
   <xs:sequence maxOccurs="unbounded">
    <xs:element ref="OMLIdentifier"/>
   </xs:sequence>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="OMLClassBody">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="OMLPosition"/>
   <xs:sequence maxOccurs="unbounded">
    <xs:element ref="OMLDefinitionBlock"/>
   </xs:sequence>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="OMLDefinitionBlock">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="OMLPosition"/>
   <xs:choice>
    <xs:element ref="OMLTypeDefinitions"/>
    <xs:element ref="OMLValueDefinitions"/>
    <xs:element ref="OMLFunctionDefinitions"/>
    <xs:element ref="OMLOperationDefinitions"/>
    <xs:element ref="OMLInstanceVariableDefinitions"/>
    <xs:element ref="OMLSynchronizationDefinitions"/>
    <xs:element ref="OMLThreadDefinitions"/>
   </xs:choice>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="OMLTypeDefinitions">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="OMLPosition"/>
   <xs:element name="OMLKeywordTypes">
    <xs:complexType>
     <xs:sequence>
      <xs:element ref="OMLPosition"/>
      <xs:element ref="OMLComments" minOccurs="0"/>
     </xs:sequence>
     <xs:attribute name="value" type="xs:string" use="required" fixed="types"/>
    </xs:complexType>
   </xs:element>
   <xs:sequence minOccurs="0" maxOccurs="unbounded">
    <xs:element name="OMLTypeDefinitionsElement">
     <xs:complexType>
      <xs:sequence>
       <xs:element ref="OMLPosition"/>
       <xs:element ref="OMLAccessTypeDefinition"/>
       <xs:element name="OMLKeywordSemicolon" minOccurs="0">
        <xs:complexType>
         <xs:sequence>
          <xs:element ref="OMLPosition"/>
          <xs:element ref="OMLComments" minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="value" type="xs:string" use="required" fixed=";"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:sequence>
 </xs:complexType>
</xs:element>
<xs:element name="OMLAccessTypeDefinition">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="OMLPosition"/>
   <xs:element ref="OMLAccess" minOccurs="0"/>
   <xs:element ref="OMLTypeDefinition"/>
```

```
    </xs:sequence>
   </xs:complexType>
  </xs:element>
  <xs:element name="OMLAccess">
   <xs:complexType>
    <xs:sequence>
     <xs:element ref="OMLPosition"/>
     <xs:choice>
      <xs:element name="OMLAccessPublic">
       <xs:complexType>
        <xs:sequence>
         <xs:element ref="OMLPosition"/>
         <xs:element name="OMLKeywordPublic">
          <xs:complexType>
           <xs:sequence>
            <xs:element ref="OMLPosition"/>
            <xs:element ref="OMLComments" minOccurs="0"/>
           </xs:sequence>
           <xs:attribute name="value" type="xs:string" use="required" fixed="public"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="OMLAccessPrivate">
       <xs:complexType>
        <xs:sequence>
         <xs:element ref="OMLPosition"/>
         <xs:element name="OMLKeywordPrivate">
          <xs:complexType>
           <xs:sequence>
            <xs:element ref="OMLPosition"/>
            <xs:element ref="OMLComments" minOccurs="0"/>
           </xs:sequence>
           <xs:attribute name="value" type="xs:string" use="required" fixed="private"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
      <xs:element name="OMLAccessProtected">
       <xs:complexType>
        <xs:sequence>
         <xs:element ref="OMLPosition"/>
         <xs:element name="OMLKeywordProtected">
          <xs:complexType>
           <xs:sequence>
            <xs:element ref="OMLPosition"/>
            <xs:element ref="OMLComments" minOccurs="0"/>
           </xs:sequence>
           <xs:attribute name="value" type="xs:string" use="required" fixed="protected"/>
          </xs:complexType>
         </xs:element>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:choice>
    </xs:sequence>
   </xs:complexType>
  </xs:element>
...
```

## I.7 Selected Editor Files

### I.7.1 Ast2XmlAction.java

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

```
/***************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 ***************************************************************************/

package org.overturetool.eclipse.editor.action;

import org.eclipse.core.runtime.IPath;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.eclipse.ui.part.FileEditorInput;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
import org.overturetool.eclipse.editor.OvertureEditor;
import org.overturetool.eclipse.editor.OvertureExtensionProvider;
import org.overturetool.eclipse.editor.wizard.OvertureWizard;
import org.overturetool.eclipse.util.OvertureAst2XmlVisitorInterface;

/**
 * This class is used when the user pushes the 'Export to XML' item
 * in the Overture menu. This is done by implementing
 * Eclipse's IWorkbenchWindowActionDelegate interface and refer
 * to this file from the plugin.xml file. The run method calls
 * the method parseASTtoXML in the OvertureContentProvider class.
 * The functionality is moved here in order to check for syntax
 * before exporting to XML.
 * */

public class Ast2XmlAction implements IWorkbenchWindowActionDelegate {
 private IWorkbenchWindow window;
 OvertureEditor editor;
 private XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());

 public Ast2XmlAction() {
 }

 public void run(IAction action) {
  editor = (OvertureEditor) window.getActivePage().getActiveEditor();
  if(editor == null){
   MessageDialog.openInformation(
     window.getShell(),
     "Overture: Export to XML",
   "An OML file has to be active in the editor window in order to do this operation!\n Please open an OML file and try again");

  }else{

   Document doc = null;
   editor.doSave(null);
   if(editor.getOvertureContentOutlinePage().getOvertureContentProvider().getErrors().isEmpty()){
    try{

     IPath path = ((FileEditorInput) editor.getEditorInput()).getFile().getLocation();
     String fileName = path.removeFileExtension().lastSegment();
     String containerName = path.removeFileExtension().removeLastSegments(1).lastSegment();
     OvertureWizard overtureWizard = new OvertureWizard("ExportWizardPage","Exporting an OML file to a XML file",containerName,fileName+".xml","xml");
     WizardDialog dialog = new WizardDialog(window.getShell(), overtureWizard);
     dialog.open();
     if(overtureWizard.performFinish){
      OvertureExtensionProvider overtureExtensionProvider = new OvertureExtensionProvider();
      OvertureVisitorTwoArg<Element> ast2XmlVisitor = overtureExtensionProvider.loadAst2XmlVisitor();
      Element omlDocument = new Element("OMLDocument");
      ast2XmlVisitor.visit(editor.getOvertureContentOutlinePage().getOvertureContentProvider().getASTDocument(), omlDocument);
      String documentString = outputter.outputString(new Document(omlDocument));
```

```
        overtureWizard.doFinish(overtureWizard.containerNameTo,overtureWizard.fileNameTo,
             documentString);
       }
      }catch (Exception e) {
       System.out.println("OvertureContentProvider -> parseASTtoXML");
       e.printStackTrace();
      }
    }
   }
  }

  public void selectionChanged(IAction action, ISelection selection) {
  }

  public void dispose() {
  }

  public void init(IWorkbenchWindow window) {
   this.window = window;
  }
}
```

### I.7.2 Xml2AstAction.java

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

```
/****************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 ****************************************************************************/

package org.overturetool.eclipse.editor.action;

import java.net.URL;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.Path;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.action.IAction;
import org.eclipse.jface.dialogs.MessageDialog;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.wizard.WizardDialog;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.IWorkbenchWindowActionDelegate;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;
import org.overturetool.eclipse.ast.ASTDocument;
import org.overturetool.eclipse.ast.OvertureVisitorOneArg;
import org.overturetool.eclipse.editor.OvertureConstants;
import org.overturetool.eclipse.editor.OvertureExtensionProvider;
import org.overturetool.eclipse.editor.wizard.OvertureExtendedWizard;
import org.overturetool.eclipse.editor.wizard.OvertureWizard;
import org.overturetool.eclipse.util.OvertureXml2AstConverterInterface;

/**
 * This class is used when the user pushes the 'Import from XML'
 * item in the Overture menu. This is done by implementing
 * Eclipse's IWorkbenchWindowActionDelegate interface and refer
 * to this file from the plugin.xml file. In the run
 * method a XML file is imported to an OML file from a workspace
 * location to another workspace location. The import is done in
 * three steps. First the XML file is loaded into a jdom document
 * and validated according to a XML schema. Then the jdom document
 * is converted to our AST representation using a visitor privided
 * by another plugin. Finilly, the AST is converted to a text string
 * using a visitor privided by another plugin. This string if formated
 * so that the created OML file is similar to the original OML file.
 */
public class Xml2AstAction implements IWorkbenchWindowActionDelegate {
 private IWorkbenchWindow window;
 //OvertureEditor editor;
 OvertureExtensionProvider overtureExtensionProvider;
 ASTDocument astDocument;

 public Xml2AstAction() {
 }

 public void run(IAction action) {
  overtureExtensionProvider = new OvertureExtensionProvider();
  Document doc = null;
  Document doc1 = null;
  URL url = null;
  String containerNameFrom = "";
  String fileNameFrom = "";
  String containerNameTo = "";
  String fileNameTo = "";

  OvertureExtendedWizard overtureExtendedWizard = new OvertureExtendedWizard("ExportWizardPage","
      Importing a XML file to an OML file","xml","oml");
  WizardDialog dialog = new WizardDialog(window.getShell(), overtureExtendedWizard);
  dialog.open();
  if(overtureExtendedWizard.performFinish){
   containerNameFrom = overtureExtendedWizard.containerNameFrom;
   fileNameFrom = overtureExtendedWizard.fileNameFrom;
   containerNameTo = overtureExtendedWizard.containerNameTo;
   fileNameTo = overtureExtendedWizard.fileNameTo;

   SAXBuilder builder1 = null;
   try{
    builder1= new SAXBuilder(true);
    builder1.setFeature("http://apache.org/xml/features/validation/schema", true);
    builder1.setFeature("http://xml.org/sax/features/validation", true);
```

```
      url = Platform.find(
        Platform.getBundle(OvertureConstants.PLUGIN_ID),
        new Path("XMLSchema/vdm0_1.xsd")
      );
      builder1.setProperty(
        "http://apache.org/xml/properties/schema/external-noNamespaceSchemaLocation",
        Platform.resolve(url).getPath()
      );
      doc = builder1.build(getWorkspaceLocation() + containerNameFrom + "/"+fileNameFrom);
     }catch (Exception e) {
      System.err.println(e);
      MessageDialog.openError(window.getShell(),
        "Overture: Error validating the file '"+containerNameFrom+"'!",
        e.toString()
      );

     }
     OvertureXml2AstConverterInterface xml2AstParser = (OvertureXml2AstConverterInterface)
          overtureExtensionProvider.loadXml2AstParser();
     if(doc != null){
      try{

       Element omlDocument = doc.getRootElement();
       if(omlDocument == null) System.err.println("DETTE ER ET STORT PROBLEM");
       astDocument = xml2AstParser.overtureXMLDocument(omlDocument);
       OvertureWizard overtureNewWizard = new OvertureWizard();
       OvertureVisitorOneArg<String> ast2OmlVisitor = overtureExtensionProvider.loadAst2OmlVisitor
           ();
       String docContent = ast2OmlVisitor.visit(astDocument);
       overtureNewWizard.doFinish(containerNameTo, fileNameTo, docContent);
      } catch (CoreException e) {
       e.printStackTrace();
      }catch (Exception e) {
       e.printStackTrace();
      }
     }
    }

  }

  public void selectionChanged(IAction action, ISelection selection) {
  }

  public void dispose() {
  }

  public void init(IWorkbenchWindow window) {
   this.window = window;
  }

  private String getWorkspaceLocation(){
   return Platform.getLocation().toString();
  }

}
```

### I.7.3 OvertureContentProvider.java

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

```
/*****************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 *****************************************************************************/

package org.overturetool.eclipse.editor;

import java.io.ByteArrayInputStream;
import java.util.List;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IMarker;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IPath;
import org.eclipse.core.runtime.Platform;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.viewers.ITreeContentProvider;
import org.eclipse.jface.viewers.Viewer;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.part.FileEditorInput;
import org.eclipse.ui.texteditor.IDocumentProvider;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;
import org.overturetool.eclipse.ast.ASTDocument;
import org.overturetool.eclipse.ast.OvertureVisitorTwoArg;
import org.overturetool.eclipse.internal.ast.InternalASTDocument;
import org.overturetool.eclipse.util.OvertureOutlineVisitorInterface;
import org.overturetool.eclipse.util.OvertureParseException;
import org.overturetool.eclipse.util.OvertureParserInterface;
import org.overturetool.eclipse.util.TreeObject;
import org.overturetool.eclipse.util.TreeParent;

/**
 * This class has serveral functions.
 *
 * - Parses the text in the active Overture editor and builds an AST.
 * - Report syntax errors to the user.
 * - Exports the AST to XML
 * - Creates an outline view by extending the ITreeContentProvider
 *
 * The parsing, the export to XML and the outline view is all done
 * by other plugins. This class is therefore only responsible for
 * handling the data from these plugins. Uses the OvertureExtensionProvider
 * class to load the plugins. By defining extension points and loading
 * the corresponding extensions the user is incouraged to e.g. create
 * he/her own AST outline view instead of the traditional outline view.
 * The inputChanged method is called every time the user saves an OML
 * document.
 */

public class OvertureContentProvider implements ITreeContentProvider {

 private TreeParent outlineRoot = null;
 private List<OvertureParseException> errors = null;
 private ASTDocument astDocument = null;
 private XMLOutputter outputter = new XMLOutputter(Format.getPrettyFormat());
 private FileEditorInput input;
 private IDocumentProvider overtureDocumentProvider;
 private OvertureExtensionProvider overtureExtensionProvider;
 private IWorkbenchWindow window;

 public OvertureContentProvider(IDocumentProvider overtureDocumentProvider){
  this.overtureDocumentProvider = overtureDocumentProvider;
  overtureExtensionProvider = new OvertureExtensionProvider();
 }

 public List<OvertureParseException> getErrors(){
  return errors;
 }

 public ASTDocument getASTDocument(){
  return astDocument;
 }

 private String getWorkspaceLocation(){
```

```
 return Platform.getLocation().toString();
}

private IPath getWorkspaceLocation2(){
 return Platform.getLocation();
}
private ASTDocument parseOML(IDocument document) {
 OvertureParserInterface overtureParser = null;
 try{
  overtureParser = overtureExtensionProvider.loadParser();
  astDocument = (InternalASTDocument) overtureParser.parse(new ByteArrayInputStream(document.get
      ().getBytes()));
  errors = overtureParser.getErrors();
  if (!errors.isEmpty()) {
   for (OvertureParseException error: errors) {
    addMarker(input.getFile(),error.getError(),
      new Integer(error.getLine()),
      new Integer(IMarker.SEVERITY_ERROR));
   }
  }
 } catch (Exception e) {
  e.printStackTrace();
 }
 return astDocument;
}

private void addMarker(IFile file, String message, int lineNumber, int severity) {
 try {
  IMarker marker = file.createMarker(IMarker.PROBLEM);
  marker.setAttribute(IMarker.MESSAGE, message);
  marker.setAttribute(IMarker.SEVERITY, severity);
  if (lineNumber == -1) {
   lineNumber = 1;
  }
  marker.setAttribute(IMarker.LINE_NUMBER, lineNumber);
 } catch (CoreException e) {
 }
}

private void createOutlineView() {
 if (errors.isEmpty()) {
  OvertureVisitorTwoArg<TreeParent> overtureOutlineVisitor = overtureExtensionProvider.
      loadOutlineVisitor();
  TreeParent omlDocument = new TreeParent("OMLDocument",astDocument.getStartLine(),astDocument.
      getStartColumn(),astDocument.getEndLine());
  overtureOutlineVisitor.visit(astDocument, omlDocument);
  outlineRoot = new TreeParent("",10,10,10);
  outlineRoot.addChild(omlDocument);
 } else {
  initialize();
 }
}

public void inputChanged(Viewer viewer, Object oldInput, Object newInput) {
 ASTDocument astDocument;

 try {
  if (input != null) {
   input.getFile().deleteMarkers(IMarker.PROBLEM, true, 1);
  }
 } catch (CoreException ce) {
  System.out.println("OvertureContentProvider -> clearProblemView");
 }

 input = (FileEditorInput) newInput;

 if (newInput != null) {
  IDocument document = overtureDocumentProvider.getDocument(newInput);
  if (document != null) {
   parseOML(document);
   createOutlineView();
  }
 }
}

public void dispose() {
}

public boolean isDeleted(Object element) {
 return false;
}

private void initialize() {
 TreeParent treeRoot = new TreeParent("No outline available",0,0,0);
```

```
   outlineRoot = new TreeParent("",0,0,0);
   outlineRoot.addChild(treeRoot);
  }

  public Object[] getElements(Object parent) {
   if (outlineRoot == null) {
    initialize();
   }
   return getChildren(outlineRoot);
  }

  public Object getParent(Object child) {
   if (child instanceof TreeObject) {
    return ((TreeObject) child).getParent();
   }
   return null;
  }

  public Object[] getChildren(Object parent) {
   if (parent instanceof TreeParent) {
    return ((TreeParent) parent).getChildren();
   }
   return new Object[0];
  }

  public boolean hasChildren(Object parent) {
   if (parent instanceof TreeParent)
    return ((TreeParent) parent).hasChildren();
   return false;
  }
 }
```

### I.7.4 OvertureContentOutlinePage.java

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

```
/******************************************************************************
 * Overturetool
 * Jens Kielsgaard Hansen & Jacob Porsborg Nielsen
 * Spring 2005, DTU Denmark
 ******************************************************************************/

package org.overturetool.eclipse.editor;

import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.ITableLabelProvider;
import org.eclipse.jface.viewers.LabelProvider;
import org.eclipse.jface.viewers.SelectionChangedEvent;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.ISharedImages;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchWindow;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.WorkbenchException;
import org.eclipse.ui.texteditor.IDocumentProvider;
import org.eclipse.ui.texteditor.ITextEditor;
import org.eclipse.ui.views.contentoutline.ContentOutlinePage;
import org.overturetool.eclipse.util.TreeObject;

/**
 * Creates and updates an outline view for the OvertureEditor.
 * Extends Eclipse's ContentOutlinePage providing the standard
 * outline view. The viewer is based on Eclipse's TreeViewer.
 * The view is updated when saving an OML file in the OvertureEditor.
 * The class OvertureContentProvider is used to provide the content
 * for this view.
 */

public class OvertureContentOutlinePage extends ContentOutlinePage {

 protected ITextEditor overtureEditor;
 private OvertureContentProvider overtureContentProvider;
 protected IDocumentProvider overtureDocumentProvider;
 private IWorkbenchWindow window;
 private TreeViewer viewer;
 protected IEditorInput input;

 /**
  * Creates a content outline page using the an OvertureEditor
  * and an OvertureDocumentProvider
  */
 public OvertureContentOutlinePage(IDocumentProvider overtureDocumentProvider, ITextEditor
     overtureEditor) {
  super();
  this.overtureDocumentProvider = overtureDocumentProvider;
  this.overtureEditor = overtureEditor;
 }

 public OvertureContentProvider getOvertureContentProvider(){
  return overtureContentProvider;
 }

 /**
  * Configures the TreeViewer with the overtureContentProvider.
  * Is called when the user opens an OML file.
  */
 public void createControl(Composite parent) {
  super.createControl(parent);

  doSetPerspective();

  try{
   viewer = getTreeViewer();
   overtureContentProvider = new OvertureContentProvider(overtureDocumentProvider);
   viewer.setContentProvider(overtureContentProvider);
   if (input != null) {
```

```java
      viewer.setInput(input);
    }
  }catch (Exception e) {
    e.printStackTrace();
  }
 }

 /**
  * Sets the input of the outline view
  */
 public void setInput(Object input) {
  this.input = (IEditorInput) input;
  update();
 }

 public void doSetPerspective(){
  try {
   IWorkbench workbench= PlatformUI.getWorkbench();
   workbench.showPerspective(
     "org.overturetool.eclipse.editor.OverturePerspective",
     PlatformUI.getWorkbench().getActiveWorkbenchWindow()
   );
  } catch (WorkbenchException e) {
   e.printStackTrace();
  }
 }

 /**
  * Updates the outline view
  */
 public void update() {



  try{
   if (viewer != null) {
    Control control = viewer.getControl();
    if (control != null && !control.isDisposed()) {
     control.setRedraw(false);
     viewer.setInput(input);
     viewer.expandAll();
     control.setRedraw(true);
    }
   }
  }catch (Exception e) {
   e.printStackTrace();
  }
 }

 /**
  * Registers when a user click on one of the nodes in the
  * outline view. Gets the chosen tree element and highlights
  * the the corresponding lines in the editor.
  */
 public void selectionChanged(SelectionChangedEvent event) {
  super.selectionChanged(event);
  ISelection selection = event.getSelection();
  if (selection.isEmpty())
   overtureEditor.resetHighlightRange();
  else {
   TreeObject treeObject = (TreeObject) ((IStructuredSelection) selection).getFirstElement();
   IDocument doc = this.overtureDocumentProvider.getDocument(input);
   try {
    overtureEditor.setHighlightRange(doc.getLineOffset(treeObject.getStartLine())-1,doc.
         getLineOffset(treeObject.getEndLine()-treeObject.getStartLine()), true);
   } catch (Exception x) {
    overtureEditor.resetHighlightRange();
   }
  }
 }

}
```

# I.8 Export/Import to/from UML

## I.8.1 XML to XMI/UML Conversion

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

```
<?xml version="1.0"?>

<xsl:stylesheet
version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:UML="org.omg.xmi.namespace.UML"
>
<xsl:output indent="yes"/>
<xsl:template match="OMLDocument">
<XMI xmi.version = '1.2' xmlns:UML = 'org.omg.xmi.namespace.UML' timestamp = 'Sat Aug 06 00:42:38
      CEST 2005'>
  <XMI.header>
    <XMI.documentation>
      <XMI.exporter>Netbeans XMI Writer</XMI.exporter>
      <XMI.exporterVersion>1.0</XMI.exporterVersion>
    </XMI.documentation>
  </XMI.header>
  <XMI.content>
    <UML:Model xmi.id = 'I1a666bfm10588cd000amm7f55' name = 'model 1' isSpecification = 'false'
      isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
      <UML:Namespace.ownedElement>
       <xsl:apply-templates select="OMLClass"/>
 </UML:Namespace.ownedElement>
    </UML:Model>
    <UML:Diagram xmi.id = 'I1a666bfm10588cd000amm7f54' isVisible = 'true' name = 'Class Diagram_1
      '
      zoom = '1.0'>
      <UML:GraphElement.position>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
      </UML:GraphElement.position>
      <UML:GraphNode.size>
        <XMI.field>160.0</XMI.field>
        <XMI.field>131.0</XMI.field>
      </UML:GraphNode.size>
      <UML:Diagram.viewport>
        <XMI.field>0.0</XMI.field>
        <XMI.field>0.0</XMI.field>
      </UML:Diagram.viewport>
      <UML:GraphElement.semanticModel>
        <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f53' presentation = ''
          typeInfo = 'ClassDiagram'/>
      </UML:GraphElement.semanticModel>
      <UML:GraphElement.contained>
        <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f50' isVisible = 'true'>
          <UML:GraphElement.position>
            <XMI.field>70.0</XMI.field>
            <XMI.field>40.0</XMI.field>
          </UML:GraphElement.position>
          <UML:GraphNode.size>
            <XMI.field>100.0</XMI.field>
            <XMI.field>71.0</XMI.field>
          </UML:GraphNode.size>
          <UML:GraphElement.semanticModel>
            <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f4f' presentation = ''>
              <UML:Uml1SemanticModelBridge.element>
                <UML:Class xmi.idref = 'I1a666bfm10588cd000amm7f51'/>
              </UML:Uml1SemanticModelBridge.element>
            </UML:Uml1SemanticModelBridge>
          </UML:GraphElement.semanticModel>
          <UML:GraphElement.contained>
            <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4e' isVisible = 'true'>
              <UML:GraphElement.position>
                <XMI.field>1.0</XMI.field>
                <XMI.field>1.0</XMI.field>
              </UML:GraphElement.position>
              <UML:GraphNode.size>
                <XMI.field>98.0</XMI.field>
                <XMI.field>19.0</XMI.field>
              </UML:GraphNode.size>
              <UML:GraphElement.semanticModel>
```

```
            <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f4d'
                presentation = ''
              typeInfo = 'NameCompartment'/>
          </UML:GraphElement.semanticModel>
          <UML:GraphElement.contained>
            <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4c' isVisible = 'true'>
              <UML:GraphElement.position>
                <XMI.field>30.3542</XMI.field>
                <XMI.field>2.0</XMI.field>
              </UML:GraphElement.position>
              <UML:GraphNode.size>
                <XMI.field>37.2915</XMI.field>
                <XMI.field>15.0</XMI.field>
              </UML:GraphNode.size>
              <UML:GraphElement.semanticModel>
                <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f4b'
                    presentation = ''
                  typeInfo = 'Name'/>
              </UML:GraphElement.semanticModel>
            </UML:GraphNode>
          </UML:GraphElement.contained>
        </UML:GraphNode>
        <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f4a' isVisible = 'true'>
          <UML:GraphElement.position>
            <XMI.field>1.0</XMI.field>
            <XMI.field>20.0</XMI.field>
          </UML:GraphElement.position>
          <UML:GraphNode.size>
            <XMI.field>98.0</XMI.field>
            <XMI.field>1.0</XMI.field>
          </UML:GraphNode.size>
          <UML:GraphElement.semanticModel>
            <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f49'
                presentation = ''
              typeInfo = 'CompartmentSeparator'/>
          </UML:GraphElement.semanticModel>
        </UML:GraphNode>
        <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f48' isVisible = 'true'>
          <UML:GraphElement.position>
            <XMI.field>1.0</XMI.field>
            <XMI.field>21.0</XMI.field>
          </UML:GraphElement.position>
          <UML:GraphNode.size>
            <XMI.field>98.0</XMI.field>
            <XMI.field>24.0</XMI.field>
          </UML:GraphNode.size>
          <UML:GraphElement.semanticModel>
            <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f47'
                presentation = ''
              typeInfo = 'AttributeCompartment'/>
          </UML:GraphElement.semanticModel>
          <UML:GraphElement.contained>
            <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f46' isVisible = 'true'>
              <UML:GraphElement.position>
                <XMI.field>2.0</XMI.field>
                <XMI.field>2.0</XMI.field>
              </UML:GraphElement.position>
              <UML:GraphNode.size>
                <XMI.field>94.0</XMI.field>
                <XMI.field>20.0</XMI.field>
              </UML:GraphNode.size>
              <UML:GraphElement.semanticModel>
                <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f45'
                    presentation = ''
                  typeInfo = 'DelimitedSection'/>
              </UML:GraphElement.semanticModel>
              <UML:GraphElement.contained>
                <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f39' isVisible = 'true'>
                  <UML:GraphElement.position>
                    <XMI.field>2.0</XMI.field>
                    <XMI.field>2.0</XMI.field>
                  </UML:GraphElement.position>
                  <UML:GraphNode.size>
                    <XMI.field>90.0</XMI.field>
                    <XMI.field>15.0</XMI.field>
                  </UML:GraphNode.size>
                  <UML:DiagramElement.property>
                    <UML:Property xmi.id = 'I1a666bfm10588cd000amm7f37' key = 'gentleware-
                        custom-width'
                      value = '0.0'/>
                    <UML:Property xmi.id = 'I1a666bfm10588cd000amm7f36' key = 'gentleware-
                        custom-height'
                      value = '0.0'/>
                  </UML:DiagramElement.property>
```

```
<UML:GraphElement.semanticModel>
  <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f38'
      presentation = ''>
    <UML:Uml1SemanticModelBridge.element>
      <UML:Attribute xmi.idref = 'I1a666bfm10588cd000amm7f3a'/>
    </UML:Uml1SemanticModelBridge.element>
  </UML:Uml1SemanticModelBridge>
</UML:GraphElement.semanticModel>
<UML:GraphElement.contained>
  <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f35' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>0.0</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>6.4238</XMI.field>
      <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f34'
          presentation = ''
        typeInfo = 'Visibility'/>
    </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f33' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>6.4238</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>18.353</XMI.field>
      <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f32'
          presentation = ''
        typeInfo = 'Name'/>
    </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f31' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>24.7769</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>3.0562</XMI.field>
      <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f30'
          presentation = ''
        typeInfo = 'TypeSeparator'/>
    </UML:GraphElement.semanticModel>
  </UML:GraphNode>
  <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2f' isVisible = 'true'>
    <UML:GraphElement.position>
      <XMI.field>27.833</XMI.field>
      <XMI.field>0.0</XMI.field>
    </UML:GraphElement.position>
    <UML:GraphNode.size>
      <XMI.field>11.6177</XMI.field>
      <XMI.field>15.0</XMI.field>
    </UML:GraphNode.size>
    <UML:GraphElement.semanticModel>
      <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f2e'
          presentation = ''
        typeInfo = 'StructuralFeatureType'/>
    </UML:GraphElement.semanticModel>
    <UML:GraphElement.contained>
      <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2d' isVisible = '
          true'>
        <UML:GraphElement.position>
          <XMI.field>0.0</XMI.field>
          <XMI.field>0.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>11.6177</XMI.field>
          <XMI.field>15.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f2c
              ' presentation = ''>
            <UML:Uml1SemanticModelBridge.element>
              <UML:DataType xmi.idref = 'I1a666bfm10588cd000amm7f3e'/>
```

```
                            </UML:Uml1SemanticModelBridge.element>
                          </UML:Uml1SemanticModelBridge>
                        </UML:GraphElement.semanticModel>
                        <UML:GraphElement.contained>
                          <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f2b' isVisible = 'true'>
                            <UML:GraphElement.position>
                              <XMI.field>0.0</XMI.field>
                              <XMI.field>0.0</XMI.field>
                            </UML:GraphElement.position>
                            <UML:GraphNode.size>
                              <XMI.field>11.6177</XMI.field>
                              <XMI.field>15.0</XMI.field>
                            </UML:GraphNode.size>
                            <UML:GraphElement.semanticModel>
                              <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f2a' presentation = ''
                                typeInfo = 'Name'/>
                            </UML:GraphElement.semanticModel>
                          </UML:GraphNode>
                        </UML:GraphElement.contained>
                      </UML:GraphNode>
                    </UML:GraphElement.contained>
                  </UML:GraphNode>
                </UML:GraphElement.contained>
              </UML:GraphNode>
            </UML:GraphElement.contained>
          </UML:GraphNode>
        </UML:GraphElement.contained>
      </UML:GraphNode>
      <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f44' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>1.0</XMI.field>
          <XMI.field>45.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>98.0</XMI.field>
          <XMI.field>1.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f43' presentation = ''
            typeInfo = 'CompartmentSeparator'/>
        </UML:GraphElement.semanticModel>
      </UML:GraphNode>
      <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f42' isVisible = 'true'>
        <UML:GraphElement.position>
          <XMI.field>1.0</XMI.field>
          <XMI.field>46.0</XMI.field>
        </UML:GraphElement.position>
        <UML:GraphNode.size>
          <XMI.field>98.0</XMI.field>
          <XMI.field>24.0</XMI.field>
        </UML:GraphNode.size>
        <UML:GraphElement.semanticModel>
          <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f41' presentation = ''
            typeInfo = 'OperationCompartment'/>
        </UML:GraphElement.semanticModel>
        <UML:GraphElement.contained>
          <UML:GraphNode xmi.id = 'I1a666bfm10588cd000amm7f40' isVisible = 'true'>
            <UML:GraphElement.position>
              <XMI.field>2.0</XMI.field>
              <XMI.field>2.0</XMI.field>
            </UML:GraphElement.position>
            <UML:GraphNode.size>
              <XMI.field>94.0</XMI.field>
              <XMI.field>20.0</XMI.field>
            </UML:GraphNode.size>
            <UML:GraphElement.semanticModel>
              <UML:SimpleSemanticModelElement xmi.id = 'I1a666bfm10588cd000amm7f3f' presentation = ''
                typeInfo = 'DelimitedSection'/>
            </UML:GraphElement.semanticModel>
          </UML:GraphNode>
        </UML:GraphElement.contained>
      </UML:GraphNode>
    </UML:GraphElement.contained>
  </UML:GraphNode>
</UML:GraphElement.contained>
<UML:Diagram.owner>
  <UML:Uml1SemanticModelBridge xmi.id = 'I1a666bfm10588cd000amm7f52' presentation = ''>
    <UML:Uml1SemanticModelBridge.element>
      <UML:Model xmi.idref = 'I1a666bfm10588cd000amm7f55'/>
```

```
          </UML:Uml1SemanticModelBridge.element>
        </UML:Uml1SemanticModelBridge>
      </UML:Diagram.owner>
    </UML:Diagram>
  </XMI.content>
</XMI>
</xsl:template>

<xsl:template match="OMLClass">
<xsl:element name="UML:Class">
        <xsl:attribute name="xmi.id">
          <xsl:value-of select='concat("I1a666bfm10588cd000amm7f51")'/>
        </xsl:attribute>

   <xsl:attribute name="name">
          <xsl:value-of select="OMLIdentifier/@name"/>
        </xsl:attribute>

 <xsl:attribute name="visibility">
          <xsl:value-of select='concat("public")'/>
        </xsl:attribute>
         <xsl:attribute name="isSpecification">
          <xsl:value-of select='concat("false")'/>
        </xsl:attribute>
         <xsl:attribute name="isRoot">
          <xsl:value-of select='concat("false")'/>
        </xsl:attribute>
         <xsl:attribute name="isLeaf">
          <xsl:value-of select='concat("false")'/>
        </xsl:attribute>
         <xsl:attribute name="isAbstract">
           <xsl:value-of select='concat("false")'/>
        </xsl:attribute>
         <xsl:attribute name="isActive">
          <xsl:value-of select='concat("false")'/>
        </xsl:attribute>
  <UML:Classifier.feature>

        <xsl:element name="UML:Attribute">
        <xsl:attribute name="xmi.id">
          <xsl:value-of select='concat("I1a666bfm10588cd000amm7f3a")'/>
        </xsl:attribute>
        <xsl:attribute name="name">
          <xsl:value-of select="OMLClassBody/OMLDefinitionBlock/OMLTypeDefinitions/
               OMLTypeDefinitionsElement/OMLAccessTypeDefinition/OMLTypeDefinition/
               OMLTypeDefinitionType/OMLIdentifier/@name"/>
        </xsl:attribute>

        <xsl:attribute name="visibility">
          <xsl:value-of select="OMLClassBody/OMLDefinitionBlock/OMLTypeDefinitions/
               OMLTypeDefinitionsElement/OMLAccessTypeDefinition/OMLAccess/*/*/@value"/>
        </xsl:attribute>

        <xsl:attribute name="isSpecification">
          <xsl:value-of select='concat("false")'/>
        </xsl:attribute>

        <xsl:attribute name="ownerScope">
          <xsl:value-of select='concat("instance")'/>
        </xsl:attribute>

        <xsl:attribute name="changeability">
          <xsl:value-of select='concat("changeable")'/>
        </xsl:attribute>

          <UML:StructuralFeature.type>
                <UML:DataType xmi.idref = 'I1a666bfm10588cd000amm7f3e'/>
              </UML:StructuralFeature.type>
              </xsl:element>

         </UML:Classifier.feature>

          </xsl:element>

        <UML:Package xmi.id = 'I1a666bfm10588cd000amm7f3c' name = 'java' isSpecification = 'false'
          isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
          <UML:Namespace.ownedElement>
            <UML:Package xmi.id = 'I1a666bfm10588cd000amm7f3d' name = 'lang' isSpecification = 'false'
              isRoot = 'false' isLeaf = 'false' isAbstract = 'false'>
              <UML:Namespace.ownedElement>
                <UML:DataType xmi.id = 'I1a666bfm10588cd000amm7f3e' name = 'int' isSpecification = 'false'
```

```
                    isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
                  <UML:DataType xmi.id = 'I1a666bfm10588cd000amm7f3b' name = 'void' isSpecification
                      = 'false'
                    isRoot = 'false' isLeaf = 'false' isAbstract = 'false'/>
                </UML:Namespace.ownedElement>
              </UML:Package>
            </UML:Namespace.ownedElement>
          </UML:Package>
      </xsl:template>
</xsl:stylesheet>
```

### I.8.2 XMI/UML to XML Conversion

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

```
<?xml version="1.0"?>

<xsl:stylesheet
version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:UML="org.omg.xmi.namespace.UML"
>
<xsl:output indent="yes"/>
<xsl:template match="XMI[@xmi.version='1.2']">
<OMLDocument>
  <OMLPosition startLine="1" startColumn="1" endLine="1" />
  <xsl:apply-templates select="XMI.content/UML:Model"/>
</OMLDocument>
</xsl:template>

<xsl:template match="XMI">
   <xsl:message terminate="yes">Unknown XMI version</xsl:message>
</xsl:template>

<xsl:template match="UML:Model/UML:Namespace.ownedElement/UML:Class">
<OMLClass>
  <OMLPosition startLine="1" startColumn="1" endLine="1" />
 <xsl:element name="OMLKeywordClass">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("class")'/>
        </xsl:attribute>
        <OMLPosition startLine="1" startColumn="1" endLine="1" />
      </xsl:element>
      <xsl:element name="OMLIdentifier">
        <xsl:attribute name="name">
          <xsl:value-of select="@name"/>
        </xsl:attribute>
          <OMLPosition startLine="1" startColumn="1" endLine="1" />
           </xsl:element>

 <OMLClassBody>
 <OMLPosition startLine="1" startColumn="1" endLine="1" />
    <OMLDefinitionBlock>
    <OMLPosition startLine="1" startColumn="1" endLine="1" />
    <OMLTypeDefinitions>
    <OMLPosition startLine="1" startColumn="1" endLine="1" />
      <xsl:element name="OMLKeywordTypes">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("types")'/>
        </xsl:attribute>
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </xsl:element>
        <xsl:apply-templates select="UML:Classifier.feature/UML:Attribute"/>
   </OMLTypeDefinitions></OMLDefinitionBlock></OMLClassBody>

      <xsl:element name="OMLKeywordEnd">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("end")'/>
        </xsl:attribute>
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </xsl:element>
 <OMLIdentifier2>
      <xsl:element name="OMLIdentifier">
        <xsl:attribute name="name">
          <xsl:value-of select="@name"/>
        </xsl:attribute>
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </xsl:element>
 </OMLIdentifier2>
</OMLClass>

</xsl:template>

<xsl:template match="UML:Attribute">
<OMLTypeDefinitionsElement>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
<OMLAccessTypeDefinition>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
<OMLAccess>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
```

```
<xsl:if test="@visibility = 'private'">
 <OMLAccessPrivate>
  <OMLPosition startLine="1" startColumn="1" endLine="1" />
  <xsl:element name="OMLKeywordPrivate">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("private")'/>
        </xsl:attribute>
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </xsl:element>
 </OMLAccessPrivate>
</xsl:if>
<xsl:if test="@visibility = 'public'">
 <OMLAccessPublic>
  <OMLPosition startLine="1" startColumn="1" endLine="1" />
  <xsl:element name="OMLKeywordPublic">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("public")'/>
        </xsl:attribute>
         <OMLPosition startLine="1" startColumn="1" endLine="1" />
      </xsl:element>
 </OMLAccessPublic>
</xsl:if>
<xsl:if test="@visibility = 'protected'">
 <OMLAccessProtected>
  <OMLPosition startLine="1" startColumn="1" endLine="1" />
  <xsl:element name="OMLKeywordProtected">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("protected")'/>
        </xsl:attribute>
       <OMLPosition startLine="1" startColumn="1" endLine="1" />
       </xsl:element>
    </OMLAccessProtected>
</xsl:if>
</OMLAccess>

<OMLTypeDefinition>
 <OMLPosition startLine="1" startColumn="1" endLine="1" />
 <OMLTypeDefinitionType>
 <OMLPosition startLine="1" startColumn="1" endLine="1" />
   <xsl:element name="OMLIdentifier">
        <xsl:attribute name="name">
          <xsl:value-of select="@name"/>
        </xsl:attribute>
         <OMLPosition startLine="1" startColumn="1" endLine="1" />
      </xsl:element>
         <xsl:element name="OMLKeywordEqualsign">
        <xsl:attribute name="value">
          <xsl:value-of select='concat("=")'/>
        </xsl:attribute>
      <OMLPosition startLine="1" startColumn="1" endLine="1" />
    </xsl:element>
<OMLType>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
<OMLBasicType>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
<OMLBasicTypeInt>
<OMLPosition startLine="1" startColumn="1" endLine="1" />
<OMLKeywordInt value="int">
<OMLPosition startLine="1" startColumn="1" endLine="1" />
</OMLKeywordInt>
</OMLBasicTypeInt>
</OMLBasicType>
</OMLType>

</OMLTypeDefinitionType></OMLTypeDefinition></OMLAccessTypeDefinition></OMLTypeDefinitionsElement
    >
</xsl:template>

</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

```
-- case200 um            : chapter2/Alarm
-- tested using     : -
-- expected output : correctly built tree
-- result          : as expected
class Alarm

types
  public String = seq of char;

instance variables

  descr    : String;
  reqQuali : Expert`Qualification

operations

  public Alarm: Expert`Qualification * String ==> Alarm
  Alarm(quali,str) ==
  ( descr := str;
    reqQuali := quali
  );

```

```
  public GetReqQuali: () ==> Expert`Qualification
  GetReqQuali() ==
    return reqQuali;

end Alarm

-- case201 um            : chapter2/Expert
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class Expert

instance variables

  quali : set of Qualification

types

  public Qualification = <Mech> | <Chem> | <Bio> | <Elec>;

operations

  public Expert: set of Qualification ==> Expert
  Expert(qs) ==
    quali := qs;

  public GetQuali: () ==> set of Qualification
  GetQuali() ==
    return quali;

end Expert

-- case202 um            : chapter6/EmergencyBrake
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class EmergencyBrake

instance variables
  enabled: bool := false

operations

public Enable: () ==> ()
Enable() ==
  enabled := true;

public IsEnabled: () ==> bool
IsEnabled() ==
```

```
  return enabled;

end EmergencyBrake

-- case203 um           : chapter6/ObstacleSensor
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class ObstacleSensor

operations

public GetData: () ==> set of Controller`Obstacle
GetData() ==
  is not yet specified;

end ObstacleSensor

-- case204 um           : chapter6/PositionSensor
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class PositionSensor

operations
public GetDirection: () ==> Vector
GetDirection() ==
  is not yet specified;

public GetPosition: () ==> Point
GetPosition() ==
  return Point(1,1,1);

end PositionSensor

-- case206 m           : chapter6/SteeringController
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class SteeringController
operations

public GetPosition: () ==> Vector
GetPosition() ==
  is not yet specified;

public SendCommand: Controller`SteeringCommands ==> ()
SendCommand(-) ==
  is not yet specified;
```

```

sync

per GetPosition => #active(SendCommand);
per SendCommand => #active(GetPosition)

end SteeringController

-- case207 m            : chapter6/SteeringMonitor
-- tested using         : -
-- expected output : correctly built tree
-- result               : as expected
class SteeringMonitor

instance variables

steering        : SteeringController;
emergencyBrake: EmergencyBrake;
active          : bool := true;
log             : seq of seq of char

operations

CheckSteering: () ==> bool
CheckSteering() ==
def sp = GetPosition();
in if ExceedsLimits(<Zero>)
   then
   ( LogCondition(beyondSafetyLimits);
     return false;
   )
   else
   ( if abs sp.Angle > MATH`pi/4
     then LogCondition(nearSafetyLimit);
     return true;
   );

LogCondition: seq1 of char ==> ()
LogCondition(str) ==
  log := log ^ str;

thread

while active do
  if not CheckSteering()
  then ( emergencyBrake.Enable();
         active := false;
       )

```

```
end SteeringMonitor

-- case208              : chapter7/Actuator
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class Actuator

instance variables

signalshown: CongestionMonitor`Signal

operations

public GetSignal: () ==> CongestionMonitor`Signal
GetSignal() ==
  return signalshown;

public SetSignal: CongestionMonitor`Signal ==> ()
SetSignal(signal) ==
  signalshown := signal;

end Actuator

-- case209              : chapter7/CongestionMonitor
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class CongestionMonitor is subclass of TrafficControl

instance variables

sensor: CongestionSensor;
actuator: ActuatorManager;
status: CongestionSensor`CongestionStatus;
previousstatus: CongestionSensor`CongestionStatus;
operator: OperatorControl

types

public Signal = <NoWarning>|<PreAnnouncement>|
                <CongestionWarning>;

operations

public CongestionMonitor: CWS`Location * PassageSensor *
    ActuatorManager *
                          OperatorControl ==> CongestionMonitor
CongestionMonitor(loc, sen, act, op) ==
```

```
( location := loc;
  sensor := new CongestionSensor();
  actuator := act;
  status := <NoCongestion>;
  previousstatus := <NoCongestion>;
  operator := op;
);

public UpdateCongestionStatus: () ==> ()
UpdateCongestionStatus() ==
(
def newstatus = sensor.IssueCongestionStatus;
 in
  ( cases mk_(previousstatus status, newstatus):
      mk_(-, <NoCongestion>, <Congestion>),
      mk_(<NoCongestion>, <Doubt>, <Congestion>) ->
         ( actuator.ShowSignal(location, <CongestionWarning>);
     ),
      mk_(-, <Congestion>, <NoCongestion>),
      mk_(<Congestion>, <Doubt>, <NoCongestion>) ->
         ( actuator.ShowSignal(location, <NoWarning>);
        )
    end;
  );
);

end CongestionMonitor

-- case210           : chapter7/CongestionSensor
-- tested using      : -
-- expected output   : correctly built tree
-- result            : as expected
class CongestionSensor is subclass of Sensor

values

congestionThreshold   : nat = 100;
noCongestionThreshold: nat = 150;
noPassages: nat1 = 4

instance variables

passageSensor: PassageSensor

types

public CongestionStatus = <Congestion>|<NoCongestion>|
                          <Doubt>
```

```
operations

public CongestionSensor: PassageSensor ==>
                         CongestionSensor
CongestionSensor(sensor) ==
  passageSensor := sensor;

public IssueCongestionStatus: () ==> CongestionStatus
IssueCongestionStatus() ==
  def averageSpeed = passageSensor.
                     AverageSpeed
  in
    if averageSpeed < congestionThreshold
    then return <Congestion>
    elseif averageSpeed > noCongestionThreshold
    then return <NoCongestion>
    else return <Doubt>

end CongestionSensor

-- case211          : chapter7/CWS
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected
class CWS

types

public Speed = nat;
public Location = nat1;

instance variables

roadNetwork: seq of CongestionMonitor := [];
sensors    : seq of PassageSensor := [];
inv len roadNetwork = len sensors;
am: ActuatorManager := new ActuatorManager();
op: OperatorControl := new OperatorControl()

operations

public AddCongestionMonitor: Location ==> ()
AddCongestionMonitor(loc) ==
( def sensor = new PassageSensor();
      cm = new CongestionMonitor();
  in
    let numberOfWarners = len roadNetwork
    in
     atomic( roadNetwork := roadNetwork(1,...,loc);
```

```
                sensors := sensors(loc,...,numberOfWarners)
              );
  am.AddActuator(loc);
);

end CWS

-- case212            : chapter7/DrivingTimesSensor
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected
class DrivingTimesSensor is subclass of PassageSensor

operations

public CarPassingEvent: nat1 ==> ()
CarPassingEvent(-) ==
  is subclass responsibility;

end DrivingTimesSensor

-- case213            : chapter7/LoopDetector
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected
class LoopDetector is subclass of DrivingTimesSensor

values

distanceBetweenLoops: nat1 = 2500;

operations

public CarPassingEvent: nat1 ==> ()
CarPassingEvent(drivingTime) ==
  NewPassage(distanceBetweenLoops * drivingTime);

end LoopDetector

-- case215            : chapter7/OperatorControl
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected
class OperatorControl

instance variables
messageLog: seq of seq1 of char := [];
locations : seq of CWS`Location := [];
inv len messageLog = len locations
```

```

operations

public ResetLog: () ==> ()
ResetLog() ==
  messageLog := [];

public WriteLog: seq1 of char * CWS`Location ==> ()
WriteLog(message, location) ==
( messageLog := messageLog ^
                ConvertNum2String(location);
  locations := locations;
);

public CongestionSpots: () ==> set of CWS`Location
CongestionSpots() ==
  return elems locations;

ConvertLog2File: () ==> seq of char
ConvertLog2File() ==
   return conc messageLog;

functions

ConvertNum2String: nat1 -> seq1 of char
ConvertNum2String(n) ==
  is not yet specified;

end OperatorControl

-- case216          : chapter7/Sensor
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class Sensor

instance variables

protected location: CWS`Location

end Sensor

-- case217          : chapter8/CongestionMonitor
-- tested using     : -
-- expected output : correctly built tree
-- result           : as expected
class CongestionMonitor is subclass of TrafficControl

types
```

```
public Signal = <NoWarning>|<PreAnnouncement>|<CongestionWarning>;

instance variables
sensor: CongestionSensor;
nameServer: NameServer;
status: CongestionSensor`CongestionStatus  := <NoCongestion>;
previousstatus: CongestionSensor`CongestionStatus  := <NoCongestion>;
operator: OperatorControl

operations

public CongestionMonitor: CWS`Location * (inmap CWS`Lane to PassageSensor) *
                          NameServer * OperatorControl ==>
                          CongestionMonitor
CongestionMonitor (loc, sen, ns, op) ==
( location := loc;
  sensor := new CongestionSensor();
  nameServer := ns;
  operator := op;
);

public UpdateCongestionStatus: () ==> ()
UpdateCongestionStatus () ==
( def newstatus = sensor.IssueCongestionStatus ;
  in
  ( cases mk_(previousstatus status, newstatus):
      mk_(-, <NoCongestion>, <Congestion>),
      mk_(<NoCongestion>, <Doubt>, <Congestion>) ->
         ( nameServer.GetActuatorManager(location, <CongestionWarning>);
          operator.WriteLog (CongestionOccurred, location);),
      mk_(-, <Congestion>, <NoCongestion>),
      mk_(<Congestion>, <Doubt>, <NoCongestion>) ->
         ( nameServer.GetActuatorManager(location, <NoWarning>);
          operator.WriteLog (CongestionResolved, location);)
    end;

  );
);

end CongestionMonitor

-- case218         : chapter8/NameServer
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class NameServer

```

```
instance variables
am: map ActuatorManager to (set of CWS`Location) := {|->}

operations

public SetActuatorManager: ActuatorManager *
                           set of CWS`Location ==> ()
SetActuatorManager(actuatorManager, locations) ==
  am := {actuatorManager |-> locations};

public SetLocation: ActuatorManager * CWS`Location ==> ()
SetLocation(actuatorManager, location) ==
  am(actuatorManager) := {location}
pre actuatorManager in set dom am;

public GetActuatorManager: [CWS`Location] ==> [ActuatorManager]
GetActuatorManager(loc) ==
  if loc = nil
  then return nil
  else let locations = inverse am
       in
         let locationSet in set dom locations be st
             loc in set locationSet
         in
           return locations (locationSet);

public GetLocations: () ==> set of CWS`Location
GetLocations() ==
  return dunion rng am;

end NameServer

-- case219 um          : chapter9/Full/Alphabet
-- tested using        : -
-- expected output     : correctly built tree
-- result              : as expected
class Alphabet

instance variables
  alph : seq of char := [];

inv AlphabetInv(alph)

functions
  AlphabetInv: seq of char -> bool
  AlphabetInv (palph) ==
    len palph mod 2 = 0 and
    card elems palph = len palph

```

```
operations
  public Alphabet: seq of char ==> Alphabet
  Alphabet (pa) == alph := pa
  pre AlphabetInv(pa);

  public GetChar: nat ==> char
  GetChar (pidx) == return alph(pidx)
  pre pidx in set inds alph;

  public GetIndex: char ==> nat
  GetIndex (pch) ==
    let pidx in set {i | i in set inds alph
                        & alph(i) = pch} in
      return pidx
  pre pch in set elems alph;

  public GetIndices: () ==> set of nat
  GetIndices () == return inds alph;

  public GetSize: () ==> nat
  GetSize () == return len alph;

  public Shift: nat * nat ==> nat
  Shift (pidx, poffset) ==
    if pidx + poffset > len alph
    then return pidx + poffset - len alph
    else return pidx + poffset
  pre pidx in set inds alph and
      poffset <= len alph;

  public Shift: nat ==> nat
  Shift (pidx) == Shift(pidx, 1)
end Alphabet

-- case221 um          : chapter9/Full/SimpleEnigma
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class SimpleEnigma
  is subclass of Component

values
   refcfg : inmap nat to nat =
     {1 |-> 3, 2 |-> 4};
   rotcfg : inmap nat to nat =
     {1 |-> 2, 2 |-> 4, 3 |-> 3, 4 |-> 1};
   pbcfg : inmap nat to nat =
     {2 |-> 3}

```

```
operations
  public SimpleEnigma: () ==> SimpleEnigma
  SimpleEnigma () ==
    (dcl cp : Component ;
     alph := new Alphabet();
     next := new Reflector();
     cp := new Rotor();
     cp.SetNext(next);
     next := cp;
     cp := new Rotor();
     cp.SetNext(next);
     next := cp;
     cp := new Rotor();
     cp.SetNext(next);
     next := cp;
     cp := new Plugboard();
     cp.SetNext(next);
     next := cp);

end SimpleEnigma

-- case222 um            : chapter9/Full/TestCase
-- tested using    : -
-- expected output : correctly built tree
-- result          : as expected
class TestCase
  is subclass of Test

instance variables
  name : seq of char

operations
  public TestCase: seq of char ==> TestCase
  TestCase(nm) == name := nm;

  public GetName: () ==> seq of char
  GetName () == return name;

  protected AssertTrue: bool ==> ()
  AssertTrue (pb) == if not pb then exit <FAILURE>;

  protected AssertFalse: bool ==> ()
  AssertFalse (pb) == if pb then exit <FAILURE>;

  public Run: TestResult ==> ()
  Run (ptr) ==
    trap <FAILURE>
      with
        ptr.AddFailure(self)
```

```
      in
        (SetUp();
      RunTest();
      TearDown());

  protected SetUp: () ==> ()
  SetUp () == is subclass responsibility;

  protected RunTest: () ==> ()
  RunTest () == is subclass responsibility;

  protected TearDown: () ==> ()
  TearDown () == is subclass responsibility

end TestCase

-- case223            : chapter9/Full/TestCase
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class TestSuite
  is subclass of Test

instance variables
  tests : seq of Test := []

operations
  public Run: () ==> ()
  Run () ==
    (dcl ntr : TestResult := new TestResult();
     Run(ntr);
     ntr.Show());

  public Run: TestResult ==> ()
  Run (result) ==
    for test in tests do
      test.Run(result);

  public AddTest: Test ==> ()
  AddTest(test) ==
    tests := tests ^ test;

end TestSuite

-- case224            : chapter9/Initial/Enigma
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class Enigma
```

```
operations

  public Keystroke : char ==> char
  Keystroke(-) ==
    is not yet specified;

instance variables
  plugboard : Plugboard

end Enigma

-- case225          : chapter9/Full/Plugboard
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected

class Plugboard
operations

  private Decode : nat ==> nat
  Decode(-) ==
    is not yet specified;

  private Encode : nat ==> nat
  Encode(-) ==
    is not yet specified;

  public Substitute : nat ==> nat
  Substitute(-) ==
    is not yet specified;


instance variables
  first_rotor : Rotor;
  config : inmap nat to nat

end Plugboard

-- case226          : chapter9/Full/Reflector
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected

class Reflector

instance variables
  cur_pos : nat;
  config : inmap nat to nat

```

```
operations

  private Decode : nat ==> nat
  Decode(-) ==
    is not yet specified;

  private Encode : nat ==> nat
  Encode(-) ==
    is not yet specified;

  public Substitute : nat ==> nat
  Substitute(-) ==
    is not yet specified;

end Reflector

-- case227            : chapter9/Full/Rotor
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected

class Rotor
operations

  private Decode : nat ==> nat
  Decode(-) ==
    is not yet specified;

  private Encode : nat ==> nat
  Encode(-) ==
    is not yet specified;

  public Rotate : nat ==> nat
  Rotate(-) ==
    is not yet specified;

  public Substitute : nat ==> nat
  Substitute(-) ==
    is not yet specified;


instance variables
  next_rotor : Rotor;
  reflector : Reflector;
  cur_pos : nat;
  config : inmap nat to nat;
  latch_pos : nat

end Rotor
```

```

-- case228            : chapter10/AnnounceBeacon
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class AnnounceBeacon is subclass of Beacon

instance variables
targetspeed: real

operations

public AnnounceBeacon: real ==> AnnounceBeacon
AnnounceBeacon(ts) ==
  targetspeed := ts;

public GetTargetSpeed : () ==> real
GetTargetSpeed() ==
  return targetspeed;

end AnnounceBeacon

-- case229            : chapter10/CabDisplay
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class CabDisplay

instance variables

instance variables

alarm          : bool := false;
emergencyBrake: bool := false;
groundFault    : bool := false

operations

public SetAlarm: () ==> ()
SetAlarm() ==
  alarm := true
pre not emergencyBrake and not groundFault;

public UnsetAlarm: () ==> ()
UnsetAlarm() ==
  alarm := false;

public SetEmergencyBrake: () ==> ()
SetEmergencyBrake() ==
```

```
( alarm := false;
  emergencyBrake := true ;);

public UnsetEmergencyBrake: () ==> ()
UnsetEmergencyBrake() ==
  emergencyBrake := false;

public SetGroundFault: () ==> ()
SetGroundFault() ==
  groundFault := true;

public UnsetGroundFault: () ==> ()
UnsetGroundFault() ==
  groundFault := false;

public GetDisplay: () ==> bool
GetDisplay() ==
  return (display);

end CabDisplay

-- case230            : chapter10/Driver
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class Driver

instance variables

faults: set of Fault := {}

types

public DriverId = token;

operations

public AddFaults: set of Fault ==> ()
AddFaults(newfaults) ==
  faults := faults union newfaults;

public GetFaults: () ==> set of Fault
GetFaults() ==
  return faults;

end Driver

-- case231            : chapter10/DriverCard
-- tested using       : -
```

```
-- expected output : correctly built tree
-- result          : as expected
class DriverCard

instance variables

id: DriverId

types

public DriverId = token;

operations

public SetId: DriverId ==> ()
SetId(newid) ==
  id := newid;

public GetId: () ==> DriverId
GetId() ==
  return id;

end DriverCard

-- case232          : chapter10/EmergencyBrake
-- tested using     : -
-- expected output : correctly built tree
-- result          : as expected
class EmergencyBrake

instance variables

emergencybrake: bool := false

operations

public SetEmergencyBrake: () ==> ()
SetEmergencyBrake() ==
  emergencybrake := true;

public UnsetEmergencyBrake: () ==> ()
UnsetEmergencyBrake() ==
  emergencybrake := false;

public GetEmergencyBrake: () ==> bool
GetEmergencyBrake() ==
  return emergencybrake;

end EmergencyBrake
```

```
-- case233             : chapter10/Event
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected

class Event

operations

public Execute: CSL ==> Test`TestResult
Execute(csl) ==
  is subclass responsibility;

end Event

-- case234             : chapter10/Fault
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected
class Fault

instance variables

speedlimit   : real;
actualspeed : real

operations

public Fault: real * real ==> Fault
Fault(max,act) ==
( speedlimit := max;
  actualspeed := act;
);

end Fault

-- case235             : chapter10/LimitBeacon
-- tested using      : -
-- expected output : correctly built tree
-- result             : as expected
class LimitBeacon is subclass of Beacon

instance variables

speed: [real] := nil

operations

```

```
public SetSpeedRestriction: real ==> ()
SetSpeedRestriction(s) ==
  speed := s;

public GetSpeedRestriction: () ==> real
GetSpeedRestriction() ==
  return speed
pre speed <> nil;

end LimitBeacon

-- case236            : chapter10/MaxSpeedEvent
-- tested using       : -
-- expected output : correctly built tree
-- result             : as expected
class MaxSpeedEvent is subclass of Event

operations

public Execute: CSL ==> Test`TestResult
Execute(csl) ==
( let ms = GetMaxSpeed()
  in
    return mk_Test`MaxSpeed(ms);
);

end MaxSpeedEvent

-- case237            : chapter12-13/MessageChannelBuffer
-- tested using       : -
-- expected output : correctly built tree
-- result             : as expected
class MessageChannelBuffer

instance variables
data : [MessageChannel] := nil

operations
public Put: MessageChannel ==> ()
Put(msg) ==
  data := msg;

public Get: () ==> MessageChannel
Get() ==
let d = data in
  ( data := nil;
    return d;
  );

```

```
sync

per Get => data <> nil;
per Put   => data = nil

sync
  mutex(Put, Get);
  mutex(Put);
  mutex(Get)

end MessageChannelBuffer

-- case238            : chapter12-13/POP3Message
-- tested using       : -
-- expected output    : correctly built tree
-- result             : as expected
class POP3Message

instance variables
  header : seq of char;
  body : seq of char;
  deleted : bool;
  uniqueId : seq of char


operations

public POP3Message: seq of char * seq of char * seq of char ==>
    POP3Message
POP3Message(nheader, nbody, nuniqueId) ==
( header := nheader;
  body := nbody;
  deleted := false;
  uniqueId := nuniqueId;
);

public GetBody: () ==> seq of char
GetBody() ==
  return body;

public GetHeader: () ==> seq of char
GetHeader() ==
  return header;

public GetText: () ==> seq of char
GetText() ==
  return header ^"\n"^body;

public Delete: () ==> POP3Message
```

```
Delete() ==
( deleted := true;
  return self;
);

public IsDeleted: () ==> bool
IsDeleted() ==
  return deleted;

public Undelete: () ==> POP3Message
Undelete() ==
( deleted := false;
  return self;
);

public GetSize: () ==> nat
GetSize() ==
  return len body + len header;

public GetUniqueId: () ==> seq of char
GetUniqueId() ==
  return uniqueId;

end POP3Message

-- case239          : chapter12-13/POP3Types
-- tested using     : -
-- expected output  : correctly built tree
-- result           : as expected
class POP3Types
types
public ClientCommand = StandardClientCommand |
                       OptionalClientCommand;
public StandardClientCommand = QUIT | STAT | LIST | RETR | DELE | NOOP
    | RSET;
public OptionalClientCommand = TOP | UIDL | USER | PASS | APOP;
public QUIT :: ;
public STAT :: ;
public LIST :: messageNumber : [nat];
public RETR :: messageNumber : nat;
public DELE :: messageNumber : nat;
public NOOP :: ;
public RSET :: ;
public TOP :: messageNumber : nat
              numLines      : nat;
public UIDL :: messageNumber : [nat];
public USER :: name : UserName;
public PASS :: string : seq of char;
public APOP :: name   : seq of char
```

```
                    digest : seq of char;
public UserName = seq of char;
public Password = seq of char;
public ServerResponse = OkResponse | ErrResponse;
public OkResponse ::  data : seq of char;
public ErrResponse :: data : seq of char;

end POP3Types
```