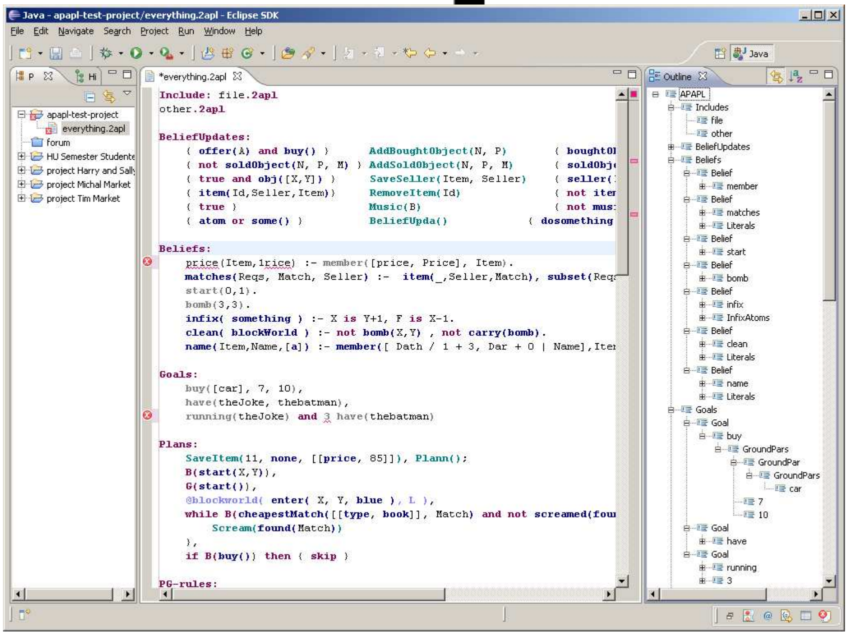
An Integrated Development Environment for A Practical Agent Programming Language

Tim J. Theeuwes

March 2010

2apl



```
Include: file.2apl
other.2apl

BeliefUpdates:
    { offer(A) and buy() }        AddBoughtObject(N, P)        { boughtOl
    { not soldObject(N, P, M) } AddSoldObject(N, P, M)         { soldObj
    { true and obj([X,Y]) }       SaveSeller(Item, Seller)     { seller(
    { item(Id,Seller,Item)}       RemoveItem(Id)               { not iter
    { true }                      Music(B)                     { not mus
    { atom or some() }            BeliefUpda()              { dosomething

Beliefs:
    price(Item,1rice) :- member([price, Price], Item).
    matches(Reqs, Match, Seller) :-  item(_,Seller,Match), subset(Reqs
    start(0,1).
    bomb(3,3).
    infix( something ) :- X is Y+1, F is X-1.
    clean( blockWorld ) :- not bomb(X,Y) , not carry(bomb).
    name(Item,Name,[a]) :- member([ Dath / 1 + 3, Dar + 0 | Name],Iter

Goals:
    buy([car], 7, 10),
    have(theJoke, thebatman),
    running(theJoke) and 3 have(thebatman)

Plans:
    SaveItem(11, none, [[price, 85]]), Plann();
    B(start(X,Y)),
    G(start()),
    @blockworld( enter( X, Y, blue ), L ),
    while B(cheapestMatch([[type, book]], Match) and not screamed(fou
        Scream(found(Match))
    ),
    if B(buy()) then { skip }

PG-rules:
```

Bachelor Thesis

Supervisors: drs. L. J. M. van Moergestel
Dr. M. M. Dastani

HOGESCHOOL UTRECHT

Universiteit Utrecht

Abstract

Nowadays almost all software developers are using Integrated Development Environments (IDE's), to write their code. Big examples of these editors are Eclipse [1], NetBeans [2] and TextPad [3]. Editors like these provide indents lines, matching words, highlight source code syntactically and have auto code completion. Using an IDE as a developer improves the quality of code, syntax errors are less made and code can be written in a faster way due to the code completion function. In addition, the existence of an outline view helps to walk trough the code will writing it.

The editor in 2APL, A Practical Agent Programming Language, a multi-agent programming language developed by Utrecht University, uses Jext [4] as an editor to provide the code with some basic coloring. There is no check on whether the code is valid and other options such as auto completion are not available.

This project aims to create an IDE for 2APL that has syntax highlighting, indents lines, content assist, an outline view, constraint violations checking, and the ability to run the project from the editor. The IDE will be created as an Eclipse plug-in so the look and feel should be familiar with most of the code developers. This Eclipse plug-in will be created with the use of Xtext [5], a Eclipse plug-in creator for textual domain specific languages. With the use of this IDE, end users who are writing code for 2APL should be able to write code without the full knowledge of the 2APL syntax and enjoy all the other advantages that are build in the IDE to help the programmer.

Tim Theeuwes
Utrecht, February 2010

Get graded on your ideas, not on your grammar!
I. Thies, 2009

Contents

List of Figures

1 Introduction

Attending the multi-agent programming [31] course at Utrecht University during my research semester, January 2009 till June 2009, I got my first hands on experience with 2APL. Participating in the practicum of this course brought to my attention that the 2APL platform did not have an IDE to create 2APL code. Working with a default notepad editor, the code was created during the various practicum sessions.

In my search to a final project for my bachelor study, I spoke with dr. M. M. Dastani, a lecturer at the Intelligent Systems Group of Utrecht University [32], on the possibilities to create an IDE for 2APL. My proposal to create an IDE for 2APL was received gratefully, and soon after the proposal a first drawing of a project proposal was created.

From Hogeschool Utrecht drs. L. J. M. van Moergestel supported this idea, as he also had the appropriate experience with 2APL and multi-agent systems since he attended various courses at Utrecht University, including the multi-agent programming course, and was willing to be my first examinator during my final bachelor project.

This thesis is composed of four parts. At first, I will describe the background that is needed to understand the details in this project in chapter 2. In chapter 3, I will explain how and why various tools and techniques are used. The actions taken in chapter 3 result in the final product, the 2APL IDE plug-in. How this plug-in works is described in chapter 4, and finally the syntax of 2APL, that has had the various updates during the project, is described to the bone in chapter 5. Ultimately, I will evaluate the project and I will finish with my conclusion and ideas for future work.

I would like to take this opportunity to thank Mehdi Dastani for his time I stole to get the 2APL syntax fully updated and his vision and advice throughout during this project. Most of all, I am grateful that I was asked to keep on developing on 2APL as a member of the 2APL team. I would also like to express my gratitude to the people of itemis [40], the company behind the Xtext project, for supporting me with their tutorials and newsgroups. Furthermore, I would like to thank Leo van Moergestel for his assistance in each possible way.

2 Background

In this chapter, some theoretical background is provided that is necessary to understand how this project came to an existence. Next, an overview of the programs that were used to create the 2APL IDE plug-in, will be described, together with a detailed explanation of those programs.

2.1 Multi-agent systems

The IDE was created for the 2APL platform, a multi-agent system oriented platform.

Multi-agent systems are systems composed of multiple interacting computing elements known as agents. An agent is a computer system that is capable of independent action on behalf of its user or owner. A multi-agent system is one that consists of a number of agents, which interact with one another, typically by exchanging messages through some computer network infrastructure. In order to successfully interact, these agents will thus require the ability to cooperate, coordinate and negotiate witch each other. [10]

Characteristics of multi-agent systems [10]:

1. Multi-agent systems consist of a number of interacting agents;
2. Multi-agent systems are designed to achieve some global goal;
3. Agents need abilities to cooperate, coordinate, and negotiate in order to achieve their objectives;
4. Multi-agent systems are specified in terms of high-level abstract concepts such as role, permission, responsibility, and interaction.

Multi-agent systems and Artificial Intelligence are not the same. Understanding and modeling social intelligence and emergent behavior are essential in multi-agent systems. [20]

Artificial Intelligence	**Multi-agent systems**
Planning	Interaction and communication
Learning	Social concepts: obligation, norms, responsibilities
Vision	Optimal solutions can be obtained by co-ordination and co-operation
Language understanding	Simulation can verify social and economic theories

2.2 About 2APL

2APL (A Practical Agent Programming Language) is an agent-oriented programming language created by the Utrecht University that facilitates the implementation of multi-agent systems. It provides programming constructs to implement an agent's beliefs, goals, plans, actions, events and a set of rules through which the agent can decide which actions to perform. [8]

To develop multi-agent programs, the 2APL platform is standard shipped with a editor that provides only syntax coloring. The 2APL platform itself provides a step-by-step walkthrough while executing the program, a message overview and a state tracker that shows the `Beliefs`, `Goals` and `Plans` for each step.

2.2.1 2APL agent example

A very small example of a 2APL agent, which at runtime will make a number of pushups, looks like:

```
BeliefUpdates:
      {true}      DoPushup(S, X)    { not count(S), count(S+X) }

Beliefs:
      count(0).

Goals:
      Pushup(10)

PG-rules:
      Pushup(T) <- not count(T) |
      {
            [B(count(R));
            DoPushup(R, 1)]
      }
```

2.2.2 Example 2APL agent explained in detail

The pushup agent consists of 4 parts;

1. BeliefUpdates
2. Beliefs
3. Goals
4. PG-rules

The **BeliefUpdates** rule is responsible for updating the current count in his belief base.

```
{true}              DoPushup(S, X)     { not count(S), count(S+X) }
Precondition        Name               Postcondition
```

In our case there is no precondition.

The `S` in the `DoPushup` function stands for the current count in the beliefbase where `X` stands for the number of pushups that needs to be added to this `S` count.

The postcondition updates the beliefbase by first removing the current count and then add a new belief with the new count.

The **Beliefs** are a Prolog notation of the facts within the program.

```
count(0).
```

The belief `count` has variable `0` at the time the agent executes, the **BeliefUpdates** are able to change this variable.

Goals within 2APL tell the agent what state should be reached by executing actions.

```
pushup(10)
```

Where this looks like Prolog code, it is not, the goal is a state that the agent wants to believe, where in our case the agent wants to pushup 10 times. To reach this the agent uses **PG-rules** explained next.

PG-rules are Planning Goal rules that are executed when the agent has the goal and this belief.

```
pushup(T) <- not count(T)
```

Variable `T` is grounded by the value 10 because the goal `pushup(10)` is still in place, where the belief `not count(10)` is also true because when executing the agent this belief states `count(0)`.

Because the PG-rule states true the plan is adopted.

```
B( count(R) );
DoPushup(R, 1)
```

First we want to know the current count of pushups in the beliefbase of the agent, we'll do this by a beliefquery `B(count(R))` and after this query the `R` variable will contain the current count of pushups. Now the agent executes the BeliefUpdate `DoPushup(R, 1)` where the current count of pushups is added with 1.

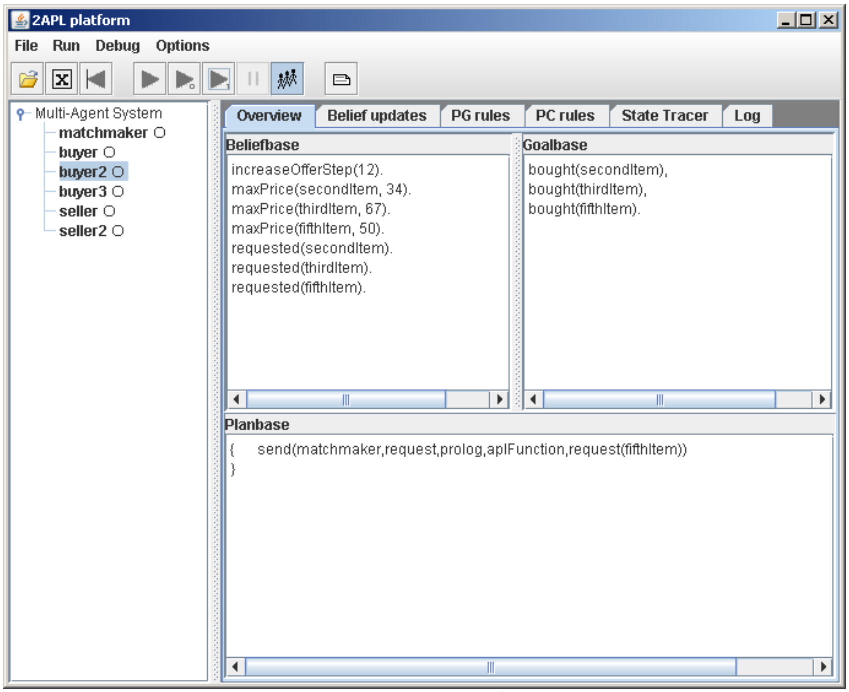
2.2.3 The 2APL platform

The 2APL platform, as shown in figure 2.1, consists of the loaded agents in the Multi-Agent System in the left column, some menu buttons to run the agent , load a MAS file or to reload the initial states of the agent, and on the right column different overviews of the selected agent in the left column.

The right column has the following views available:

Name	Function
Overview	Shows the beliefbase, goalbase and the planbase of the selected agent
Belief updates	Shows all the belief update rules that the selected agent has
PG rules	Shows all the PG rules that the selected agent has
PC rules	Shows all the PC rules that the selected agent has
State Tracer	For every state in the past of the agent the Step, Beliefs, Goals and Plans can be shown
Log	Shows messages for every step the agent has taken

Not that all of agent codes in the 2APL platform the cannot be changed during runtime, but should be changed in the 2APL files and then executed again to affect the changes.



```
increaseOfferStep(12).
maxPrice(secondItem, 34).
maxPrice(thirdItem, 67).
maxPrice(fifthItem, 50).
requested(secondItem).
requested(thirdItem).
requested(fifthItem).
```

```
bought(secondItem),
bought(thirdItem),
bought(fifthItem).
```

```
{    send(matchmaker,request,prolog,aplFunction,request(fifthItem))
}
```

Figure 2.1 - 2APL platform

2.3 An Integrated Development Environment

As shown in the pushup agent example above, the code for the agent is plain, black on white with no checks if the code is valid. An Integrated Development Environment (IDE) adds features to make the programmer creating the code as simple as possible. So what should an IDE have or do?

Figure 2.2 - Some options in an IDE [12]

A lot of words/terms, I will now show an overview with an explanation of the functions that will be available in the 2APL IDE in the next six subchapters.

2.3.1 CodeFormatting

Since code formatting is more a personal flavor, i.e. everybody likes his code formatted a different way, I will give an example or unformatted code and formatted code of a 2APL source.
Unformatted:

```
Include: file .2apl other .2apl BeliefUpdates: { offer ( A ) and buy ( ) }
AddBoughtObject ( N , P ) { boughtObject ( N , P ) }
```

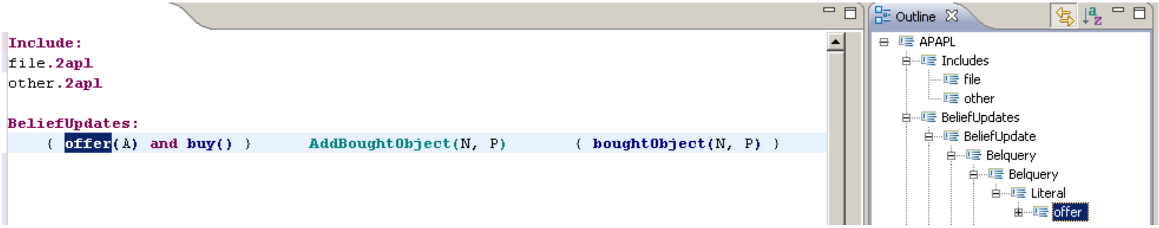
Formatted:

```
Include:
file.2apl
other.2apl

BeliefUpdates:
       { offer(A) and buy() }     AddBoughtObject(N, P)      { boughtObject(N, P) }
```

The meaning of code formatting is that code is easier readable. In the example above it is made clear that formatting the code improves the ability to read the code. An IDE is able to accept whitespaces, tabbing and new lines between the rules without throwing out an error.

2.3.2 OutlineView

An outline view gives a structured view of the code; the code is splitted into the different rules in a hierarchical tree structure. The example below shows that selecting a rule in the outline view triggers the code to be selected in the 2APL source, in this case the Atom rule `offer`.

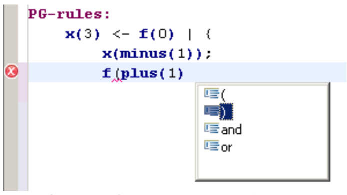


```
Include:
file.2apl
other.2apl

BeliefUpdates:
    { offer(A) and buy() }       AddBoughtObject(N, P)        { boughtObject(N, P) }
```

Figure 2.3 - Outline view example

2.3.3 ContentAssist

A content assistor helps the writer of the code selecting the available input, i.e. all the available options, in the IDE. It can be invoked by `ctrl + space`. The example shows what options are available in the middle of this rule.

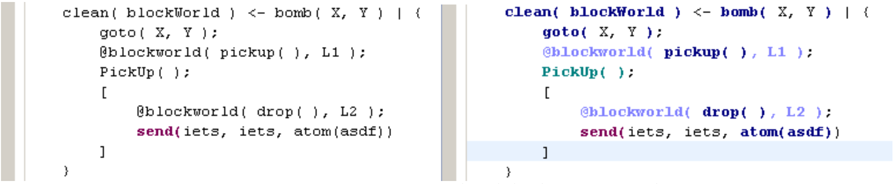


```
PG-rules:
    x(3) <- f(0) | {
        x(minus(1));
        f(plus(1)
```

Figure 2.4 - Content assist example

2.3.4 SyntaxHighlighting

By highlighting the code syntactical the code gets even more readable as per default all the code is black and the keywords are red. To change the color preferences; see chapter 4.4 for the manual. The example will show the difference between uncolored and colored code.



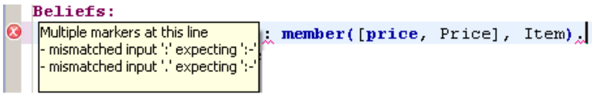
```
clean( blockWorld ) <- bomb( X, Y ) | {
    goto( X, Y );
    @blockworld( pickup( ), L1 );
    PickUp( );
    [
        @blockworld( drop( ), L2 );
        send(iets, iets, atom(asdf))
    ]
}
```

```
clean( blockWorld ) <- bomb( X, Y ) | {
    goto( X, Y );
    @blockworld( pickup( ), L1 );
    PickUp( );
    [
        @blockworld( drop( ), L2 );
        send(iets, iets, atom(asdf))
    ]
}
```

Figure 2.5 - Syntax highlight example

2.3.5 CodeValidation

The code validation function checks if the code applies to the syntax rules. If it does not an error in the form of a red line will be given. By clicking on the red x in front of the line an error message will be shown, in the example it also gives the solution.



```
Beliefs:
: member([price, Price], Item).
```

Multiple markers at this line
- mismatched input ':' expecting ':-'
- mismatched input '.' expecting ':-'

Figure 2.6 - Code validation example

2.3.6 ProjectCreator

A project creator helps the end user setup a new 2APL project. By running the project creator, the user will be able to start off quicker than when he has to create a new project from the ground. The project creator in the 2APL IDE plug-in also creates an example agent, the push-up agent, that is an out of the box finished example.

2.4 Plug-ins

This project creates an IDE in the form of a plug-in for Eclipse. A plug-in consists of a computer program that interacts with a host application, in this project the Eclipse platform, to provide a certain specific, function on demand. [34]

In this tutorial [35] it is described how to create a basic plug-in for the Eclipse platform and how this plug-in is functioning, as in this project Xtext creates the plug-in with the use of a MWE workflow file I will only describe how to create the specific Xtext plug-in and how this can be installed within the Eclipse platform.

2.5 Editors in other Agent Programming Languages

A quick look at the editors used by other Agent Programming Languages is shown below. This also gives an inside look in the other available Agent Programming Languages out there.

Table comparing the editors/IDE's:

	2APL IDE	2APL editor	Jason	Jade*	3APL
Syntax highlighting	Yes	Yes	Yes	Yes	Yes
Auto code completion	Yes	No	No	No	No
Outline view	Yes	No	No	Yes	No
Code validation	Yes	No	Yes	No	No
Code templates	Yes	No	Yes	No	No
Run project from IDE	Yes	No	Yes	No	Yes

* For Jade a standard Java editor [1] was used, Jade itself is not shipped with an IDE or Editor.

2.6 Created work

The following work was created during this project:

1. IDE plug-in for 2APL
2. New EBNF syntax for 2APL
3. Manual and installation guide for the 2APL IDE plug-in
4. Tutorial for 2APL (partly created during the multi-agent programming course)
5. Bachelor Thesis

2.7 Project timeline

The official start of this project was 1st of September 2009, but due to private circumstances I started two weeks earlier so that I could go on an external training of two weeks at the end of September. The deadline for this thesis was originally 15th of December 2009, with a presentation in January 2010.

The following planning was made in the project proposal:

1.	Orientation	4-6 weeks
2.	Definition	2-3 weeks
3.	Design	2-4 weeks
4.	Realization	3-6 weeks
5.	Implementation	5-7 weeks
6.	Documentation	2-3 weeks

Due to the fact that I attended the “Multi-agent programming” course at Utrecht University before starting with this project, I have got a good impression of agent oriented languages, 2APL and the BDI (Belief, Desire and Intentions) model. The experience I acquired during this course helped me getting started faster than I planned. More information about this course can be found on the website of the Computer Science department [31] of the Utrecht University.

When my thesis was disapproved for graduating in January 2010, I had to hand in a redefined version of my thesis. The deadline for the second version of this thesis is 16th of March 2010, with a presentation in the end of March or the beginning of April 2010.

The timeline, splitted in the different phases, looks at the end of the project like this:

#	Phase	Start date	Explanation
1	Orientation	20th of August 2009	Looking into existing IDE’s, reading 2APL documentation, making tutorials of Eclipse
2	Start Xtext implementation	31st of August 2009	Working on Xtext to create the IDE, looking into Xtext documentation
3	Update 2APL syntax	2nd of September 2009	Updating the 2APL syntax with Mehdi
4	Left-factoring	9th of September 2009	Starting to left-factor the updated 2APL rules to fit in the Xtext project
5	Thesis drafts	Mid November 2009	Working on my thesis and review the structure with Leo
6	Implementing Xtext features	Mid November 2009	Adding the outline view, syntax coloring and project creator
7	Finishing the 2APL syntax	Begin December 2009	Making the final changes to the 2APL syntax
8	Finishing the Xtext implementation	Mid December 2009	Add the run project from IDE feature
9	Thesis hand in	16th of December 2009	
10	Creating 2APL IDE manual	End of December 2009	Creating the installation and user guide for the 2APL IDE plug-in
11	Thesis disapproval	20th of January 2010	My thesis version of 16th of December was disapproved for graduating
12	Updating thesis	21st of January 2010	Updating my thesis for graduating, talking with Leo and Martin how to change my thesis in an appropriate way
13	Thesis hand in 2nd version	16th of March 2010	
14	Project presentation	End of March 2010	

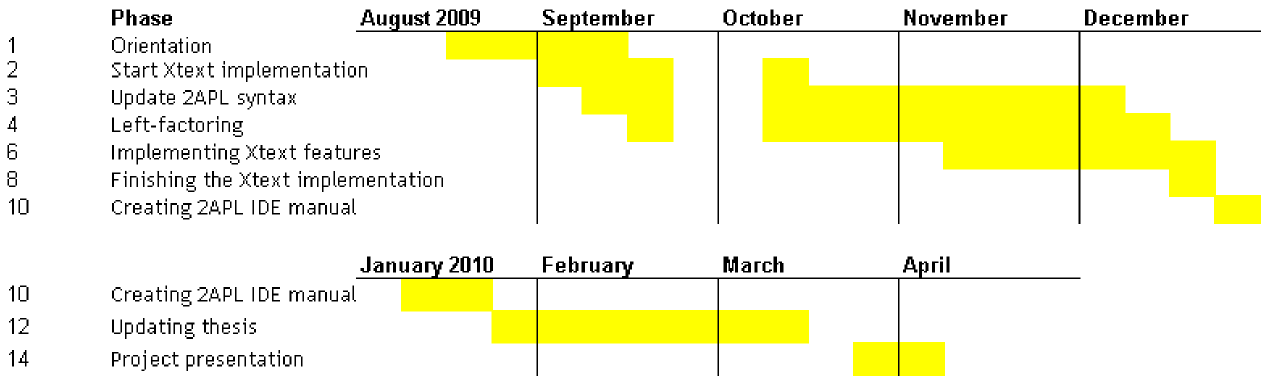


Figure 2.7 - Project timeline

3 Creating the IDE

In this chapter all the steps that are taken to get from the idea to the IDE will be explained. Starting with an overview of all the programs and techniques used and how these are all related to each other. Next, the programs and techniques will be discussed and explained, in some cases accompanied by an example to explain it in detail.

After reading this chapter it should be possible for the reader to recreate the project totally from scratch and understand how and why the various choices have been made.

3.1 Used programs structure

To create the 2APL IDE a lot of programs, techniques, tools and methods are used during the project. An overview is shown below with the paragraph reference followed:

1.	Eclipse	3.2 and 3.4
2.	Xtext	3.3 and 3.4
3.	ANTLR	3.5
4.	EMF	3.6
5.	ECore Tools	3.8
6.	MWE	3.7
7.	Left-factoring	3.10
8.	Outline view	3.11
9.	DSL	3.12
10.	EBNF	3.9.1
11.	Multiple Alternatives	3.13
12.	Backtracking	3.14
13.	Project creator	3.15
14.	Syntax highlighting	3.16
15.	Run configuration	3.18

Using this, resulted in the following outputs:

1.	ANTLR Syntax diagrams	3.5
2.	Xtext diagram	3.8
3.	Xtext grammar	3.9.2
4.	2APL Syntax	3.17
5.	IDE plug-in	3.7

How all the programs work together that are used during this project.

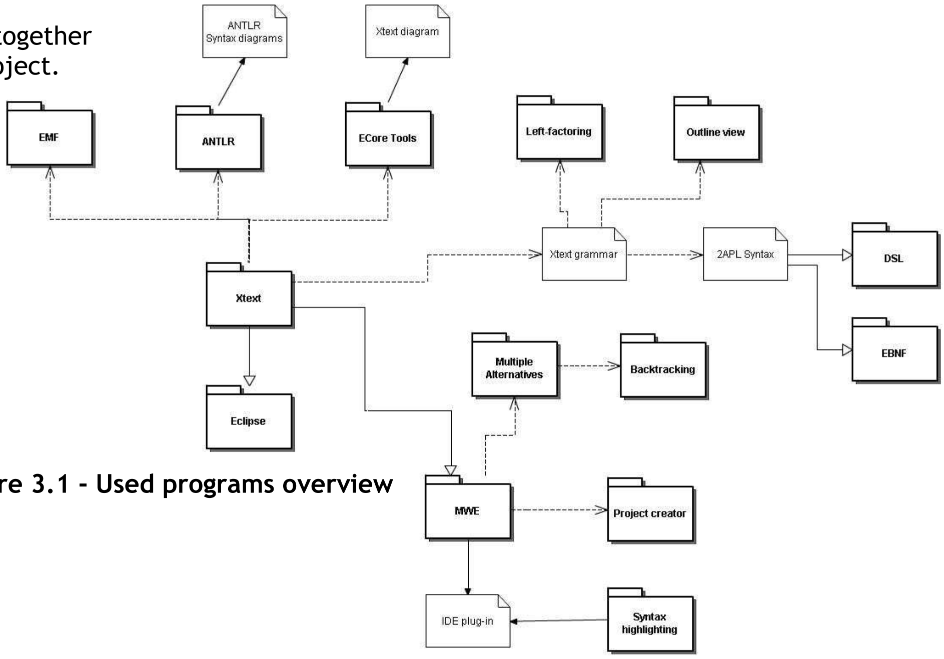


Figure 3.1 - Used programs overview

3.2 Eclipse

Eclipse is an open source community, whose projects are focused on building an open development platform comprised of extensible frameworks, tools and runtimes for building, deploying and managing software across the lifecycle. [17]

The choice for Eclipse as a framework for the 2APL IDE was made quickly. Although there are many other editors available nowadays [2, 3, 4], Eclipse is well known under the end-users that will program in 2APL and Eclipse also supports the usage of EMF (Eclipse Modeling Framework) that is used by Xtext [5].

3.3 Xtext

Xtext is a framework for the development of domain-specific languages and other textual programming languages. It is tightly integrated with the Eclipse Modeling Framework (EMF) and leverages the Eclipse Platform in order to provide a language-specific integrated development environment (IDE). [11]

It is implemented in Java and is based on Eclipse, EMF and ANTLR. A view of how the Xtext plug-in works with the other tools is shown here.

I found Xtext on the Eclipse forum [41] in the Newcomers topic when I was asking for documentation or examples of existing Eclipse IDE plug-ins.

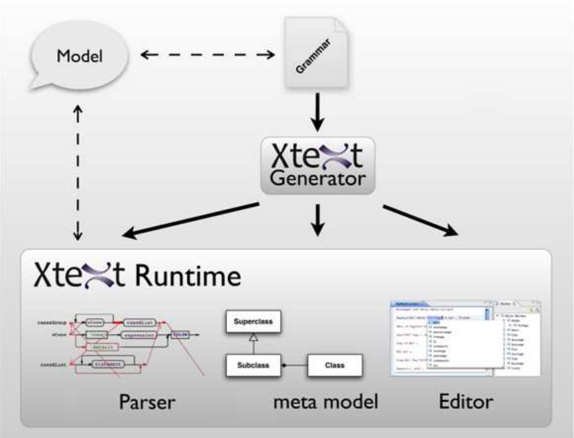


Figure 3.2 - Xtext overview [13]

Create a new Xtext project by following the next five steps:

1. Open Eclipse Xtext [5] en select `File`, `New`, `Project`.
2. Select `Xtext Project` in the New Project wizard to create a new Xtext project.
3. In the `Xtext project wizard` fill in:
   a. Main project name;
      (e.g. `nl.uu.cs.apapl.ide`)
   b. Language name;
      (e.g. `nl.uu.cs.apapl.ide.APAPL`)
   c. DSL-File extension;
      (e.g. `2apl`)
   d. Deselect the `Create generator project` checkbox (this option is for IDE plug-ins that generate code from the created DSL).
4. By finishing the project wizard 2 new project will be available in the `Package Explorer`.

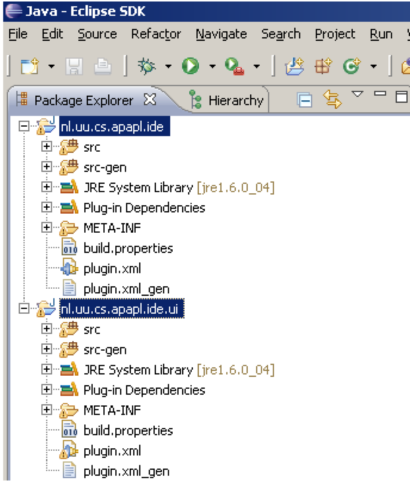


Figure 3.1 - Xtext projects

5. In the `nl.uu.cs.apapl.ide src` folder a default MWE Workflow file is created and also a Xtext file, that holds the Xtext grammar, that can be found in appendix C, is created.

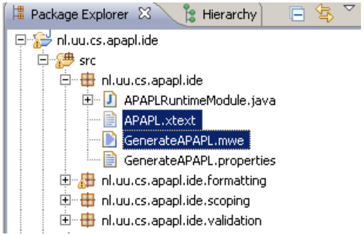
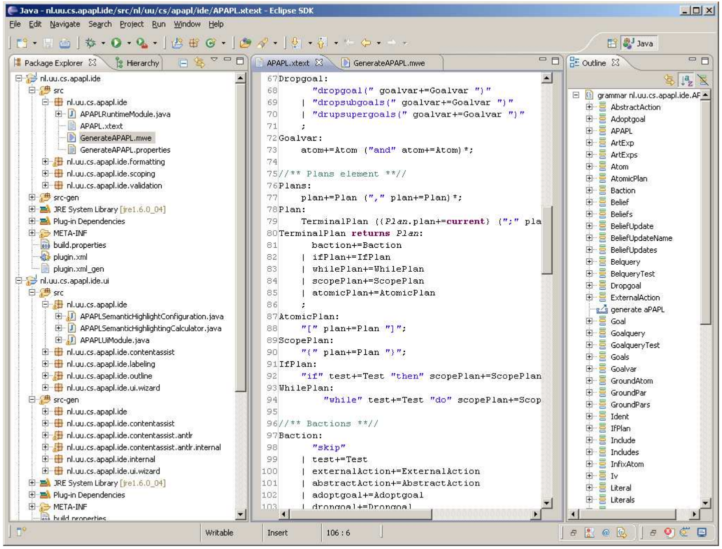


Figure 3.2 - Xtext project files

3.4 The Eclipse with Xtext platform

During this project I worked with Eclipse version 3.5.1 and Xtext version 0.7.2. The project consist of 2 parts, the language rules project and the GUI preferences project. Both of the projects are created during the setup of a new Xtext project.



```
Dropgoal:
      "dropgoal(" goalvar+=Goalvar ")"
    | "dropsubgoals(" goalvar+=Goalvar ")"
    | "drupsupergoals(" goalvar+=Goalvar ")"
    ;
Goalvar:
    atom+=Atom ("and" atom+=Atom)*;

//** Plans element **//
Plans:
    plan+=Plan ("," plan+=Plan)*;
Plan:
    TerminalPlan ({Plan.plan+=current} (";" pla
TerminalPlan returns Plan:
      baction+=Baction
    | ifPlan+=IfPlan
    | whilePlan+=WhilePlan
    | scopePlan+=ScopePlan
    | atomicPlan+=AtomicPlan
    ;
AtomicPlan:
    "[" plan+=Plan "]";
ScopePlan:
    "{" plan+=Plan "}";
IfPlan:
    "if" test+=Test "then" scopePlan+=ScopePlan
WhilePlan:
        "while" test+=Test "do" scopePlan+=Scop

//** Bactions **//
Baction:
      "skip"
    | test+=Test
    | externalAction+=ExternalAction
    | abstractAction+=AbstractAction
    | adoptgoal+=Adoptgoal
    | dropgoal+=Dropgoal
```

Figure 3.3 - Eclipse with Xtext platform

The editor has a `package explorer` that allows you to navigate through all the `src` and `src-gen` files, an outline view to easily oversee all the rules in the currently opened file and the editor with the opened file itself.

3.5 ANTLR

ANTLR, ANother Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, interpreters, compilers, and translators from grammatical descriptions containing actions in a variety of target languages. [18] It is used in this project to create the syntax diagrams and is also used by Xtext to auto generate the parser.

Other examples of language tools are YACC [37], GNU bison [38] or Coco/R [39], but due to the fact that ANTLR is also used by Xtext, it was more then obvious to use ANTLR for our syntax diagrams. More information about ANTLR can be found on the ANTLR website [18].

3.5.1 Parser

A parser is one of the many components within the IDE, it checks for correct syntax and builds a abstract syntax tree implicit in the input rules. [36] In this project, the parser is automatically generated by Xtext by inserting the EBNF-like Xtext grammar, see chapter 3.9.2, and executing the MWE workflow, see chapter 3.7. To get an impression of how complicated a parser can get if it would be build by hand, the parser for 2APL, with a syntax that contains 50 rules, is 30302 lines of code and called `InternalAPAPLParser.java` in this project.

3.5.2 ANTLR syntax diagrams

To create the ANTLR syntax diagrams, the Xtext code had to be loaded in the ANTLR platform, which is easily accomplished by importing the Xtext grammar file in the ANTLR platform, the full Xtext grammar file can be found in appendix C. A syntax diagram of every rule is then generated. Below you will find an example of such output, called an ANTLR syntax diagram.

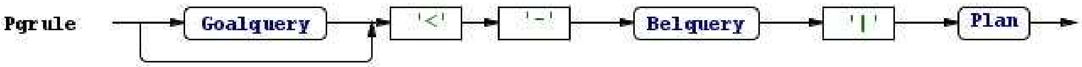


Figure 3.4 - Example of a syntax diagram created by ANTLR

3.6 EMF

EMF stands for Eclipse Modeling Framework. The EMF project is a modeling framework and code generation facility for building tools and other applications based on a structured data model. [28]

In the 2APL IDE plug-in, Xtext uses EMF to create an AST(abstract syntax tree)-meta model. Xtext validates the AST for structural correctness, performs various modifications (linking, etc.) and optimizations on it before eventually some other representation is created (typically code in some target language). [29]

3.7 MWE & IDE plug-in

The modeling workflow engine (MWE) supports orchestration of different Eclipse modeling components to be executed within Eclipse as well as standalone. Based on a dependency injection framework, one can simply configure and wire up workflows using a declarative XML-based language. The project provides the runtime used to execute workflows as well as the IDE tooling used to edit, start and debug them. [22]

During the creation of a new Xtext project, a workflow file (in this project named `GenerateAPAPL.mwe`) was created. I see this file as a kind of an `apache ant` [23] build file. In order to create the IDE, this workflow file needs to be executed.

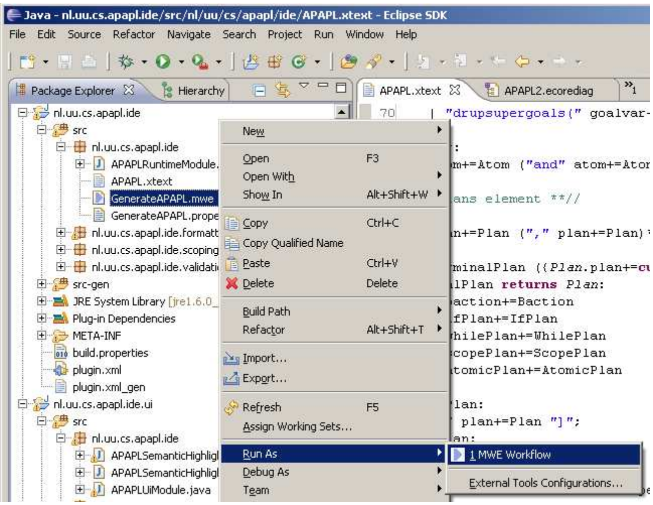


Figure 3.5 - Running the MWE Workflow

In appendix D the full `GenerateAPAPL.mwe` file can be found. Parts changed in this file can be found in chapters 3.14 Backtracking and 3.15 Project creator.

When executing the `MWE Workflow` the files from the folder `src` will be compiled, the result will be stored in the folder `src-gen`, this folder then contains the generated java files that are the basis for the IDE.

To finally export the generated java code to a plug-in, the Xtext project has to be exported.

1. Select the 2 projects that where created during and right click to select `Export;`

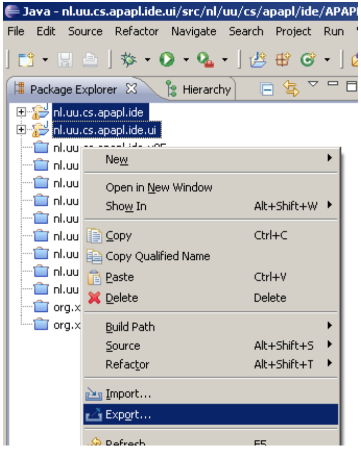
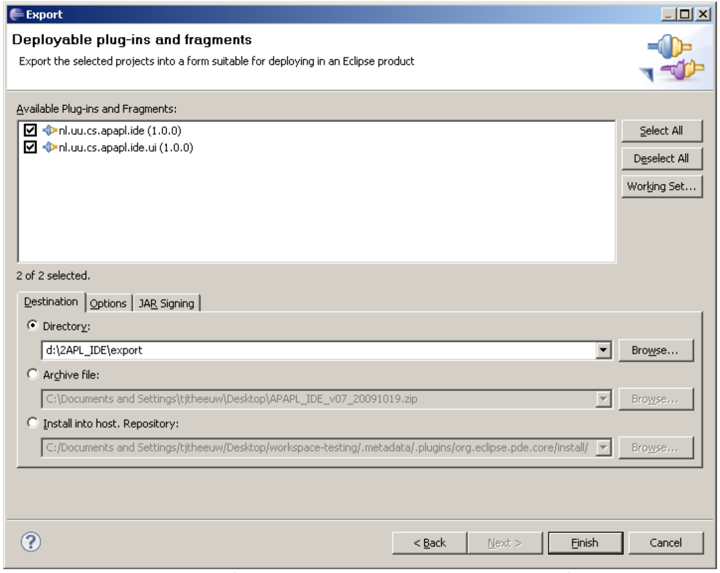


Figure 3.6 - Export Xtext projects 1

2. In the Export wizard select `Deployable plug-ins and fragments;`
3. Select a `Destination directory` and click `Finish.`



Export

Deployable plug-ins and fragments

Export the selected projects into a form suitable for deploying in an Eclipse product

Available Plug-ins and Fragments:

☑ nl.uu.cs.apapl.ide (1.0.0)
☑ nl.uu.cs.apapl.ide.ui (1.0.0)

2 of 2 selected.

● Directory: d:\2APL_IDE\export

○ Archive file: C:\Documents and Settings\tjtheeuw\Desktop\APAPL_IDE_v07_20091019.zip

○ Install into host. Repository: C:/Documents and Settings/tjtheeuw/Desktop/workspace-testing/.metadata/.plugins/org.eclipse.pde.core/install/

Figure 3.7 - Export Xtext projects 2

The result is a set of two jar files; these files are the end product of this project. The files can now be used by end users to enjoy all the features the 2APL IDE has to offer. In chapter 4.1 it is described how to install the 2APL IDE plug-in files.

Note that executing the MWE workflow can be very time-consuming. A way to reduce the time that worked for me was moving the entire Xtext project from a remote file system to my local desktop. This action resulted in a executing time of 5 minutes to just over less than 25 seconds.

3.8 ECore Tools & Xtext diagram

The ECore Tools component provides a complete environment to create, edit and maintain ECore diagrams. [21] In this project, an ECore diagram is created to show how all the 2APL rules have their specific relations with other rules. To create an ECore diagram, the MWE Workflow has to be executed first (see chapter 3.7 for this), then the `src-gen` folder holds an `APAPL.ecore` file that can be transferred to an ECore diagram, in this thesis called the Xtext diagram.

To create the Xtext diagram follow the three steps below:

1. Execute the `MWE workflow` file;
2. Open the `src-gen` folder and find the ecore file (in this project called `APAPL.ecore`);
3. Right-click the ecore file and select `Initialize Ecore Diagram file`;

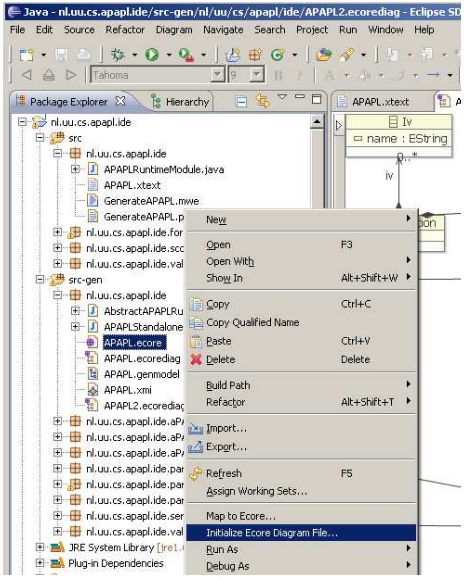


Figure 3.8 - Initialize Ecore Diagram

After step three there is a new file, called in our case `APAPL.ecorediag`, this gives a graphical overview of all the various 2APL syntax rules and their relations with each other. In appendix F is a full size version of the Xtext diagram attached.

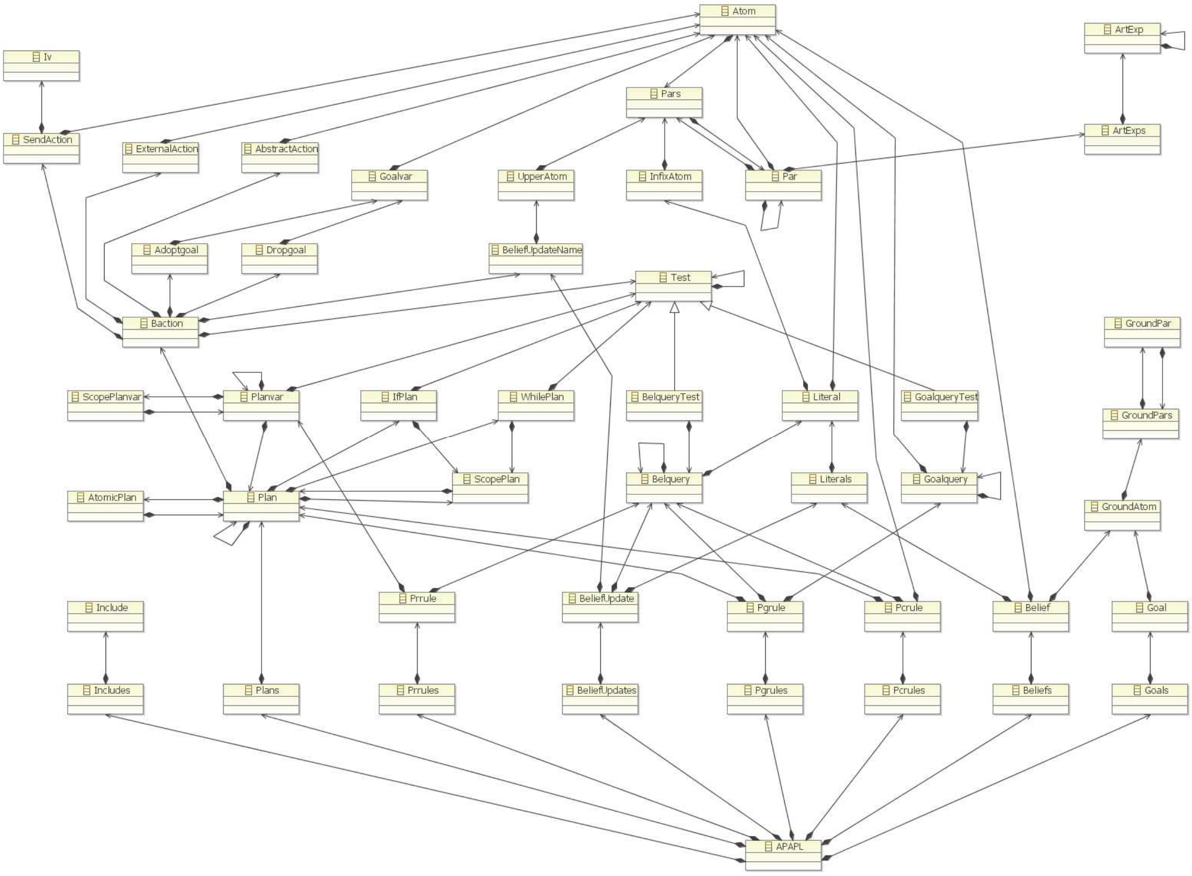


Figure 3.9 - Xtext diagram

3.9 Syntax grammar

During this project I have been using two types of grammar; the EBNF grammar that is used to describe the 2APL syntax, and the Xtext grammar that is used to create the Xtext project. In a simple way I see the Xtext grammar as an add-on to the EBNF grammar, where the Xtext has more options to modify the final 2APL IDE plug-in.

The next two subchapters will describe the formal way of the different syntaxes used in this project.

3.9.1 EBNF grammar

EBNF stands for Extended Backus Naur Form and is used to describe the formal way of computer programming languages. [42] The explanation of the used notations in the 2APL grammar are shown below:

```
<name>        = rule name
{<rule>}      = 0..* (This rule can persist zero or multiple times)
[<rule>]      = 0..1 (This rule is optional)
<rule>+       = 1..* (This rule can persist one or multiple times)
<a> | <b>     = choice between rule name a and b
(<a> | <b>) = choice between rule name a and b within brackets
"var"         = static text
;             = end of the rule
```

Combinations of the differed notations are allowed as well (e.g. `<ident> ["(" <ident> ")"];` which would mean that this rule should start with an `<ident>` followed optional by `"(" <ident> ")"`).

3.9.2 Xtext grammar

The Xtext grammar rules are described using Extended Backus-Naur Form-like expressions. [27] The explanation of the used notations in the Xtext grammar are shown below:

```
Name:         = rule name (Xtext rule names start with a capital letter)
Rule?         = 0..1 (This rule is optional)
Rule*         = 0..* (This rule can persist zero or multiple times)
Rule+         = 1..* (This rule can persist one or multiple times)
A | B         = choice between rule name A and B
(A | B)       = choice between rules within brackets
"var"         = static text
name+=Rule    = a multi valued feature and adds the value on the right hand to
                that feature, which is a list feature
name=Rule     = straight forward assignment, and used for features which take only
                one element
;             = end of the rule
```

The Xtext grammar is an implementation of the new 2APL syntax that I updated during this project, the new syntax can be found in chapter 5. The Xtext grammar is left-factored where needed. The full Xtext grammar can be found in appendix C. An example of a rule is described below:

EBNF syntax

```
<beliefupdate>          =
      "{"<belquery>"}" <beliefupdatename> "{"<literals>"}";
```

Xtext syntax

```
BeliefUpdate:
      "{" belquery+=Belquery "}" beliefUpdateName+=BeliefUpdateName
"{" literals+=Literals "}";
```

Rules itself start with capital letters and the class where the rule is part of starts with a lowercase letter. In the example shown above the rules are all multi valued, which means that there is an other rule after `Belquery`, `BeliefUpdateName` or `Literals`.

3.10 Left-recursive and left-factoring

During the creation of the IDE plug-in I ran into the following problem, left-recursive rules, this are rules that call itself e.g.

```
<belquery> =
        "true"
      | <literal>
      | <belquery> "and" <belquery>
      ;
```

The Xtext parser uses a `LL parser` that parses the input from left to right. In the `<belquery>` rule it can therefore never find a match for e.g. `dummy()` (an example of a literal) because it can be a `<literal>` or a `<belquery>` (that has next a `<literal>` or `<belquery>` etc.) rule.

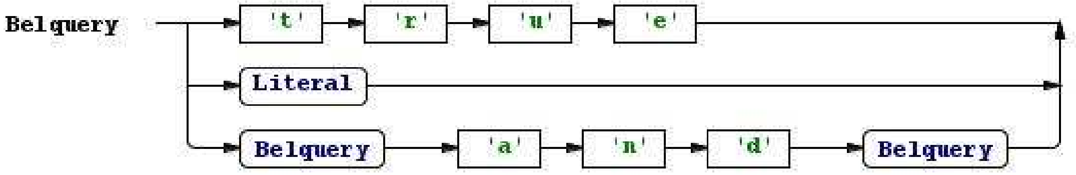


Figure 3.10 - Left-recursive rule example

This has to be rewritten with left-factoring. The rule would than look like:

```
<belquery> =
      <terminalbelquery> ["and" <belquery>];

<terminalbelquery> =
        "true"
      | <literal>
      ;
```

The example `dummy()` would then parse to `<termianlbelquery>`, that will pars it to a <literal> and finish.

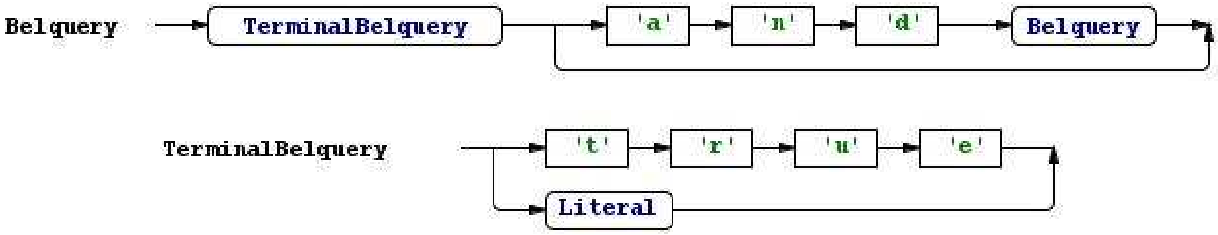


Figure 3.11 - Left-factored rule example

In this project seven 2APL rules had to be rewritten with the help of left-factoring, these rules are:

1. test
2. plan
3. planvar
4. par
5. goalquery
6. belquery
7. artexp

In chapter 5 all the 2APL rules will be described, there these seven rules are left-factored, more information about left factoring can be found in my posts on the Eclipse forum [43].

3.11 Outline view

As described in chapter 2.3.3, the outline view greatly improves the overview structure of the created code. In the 2APL IDE plug-in, the outline view is made out of two parts. The first part is responsible for the multi valued features, whereas the second part is responsible for the features that only take one element. [30]

In the next two subchapters I will describe the difference between these two parts and how they both are implemented in the 2APL IDE plug-in.

3.11.1 Multi valued features

The += sign, the add operator, expects a multi valued feature and adds the value on the right hand to that feature, which is a list feature. [30]

In the 2APL IDE plug-in this is used by rules that have the option to contain a different rule, which in the 2APL case will be practical almost all of the rules. The Xtext grammar describes how this is implemented, see chapter 3.9.2 for this.

A multi valued feature is shown as a collapsed item in the outline view, by expanding the item one or more rules can be applicable, the example below shows that the rule `Goals` and `Plans` can contain more than one `Goal` or `Plan` rule, that are now part of a list inside the outline view.

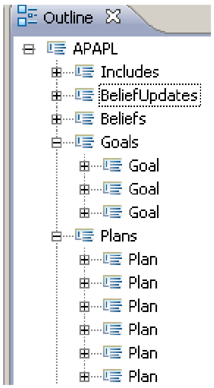


Figure 3.12 - Multi valued features

3.11.2 Single valued features

The simple equal sign = is the straight forward assignment, and used for features which take only one element. [30]

By assigning a rule with a single valued feature, the outline is not given the name of the rule, as with a multi valued feature, but is given the name of the value inside the rule.

An example is shown below where the `Goal` rule starts with a `GroundAtom` that starts with an `Ident` that then gives the name of the `GroundAtom` in the outline view, in this case `receiveMoney`. Next, it is for the `GroundAtom` rule possible to have one or more `GroundPars`, so this rule is given the name of the rule itself, in the `GroundPars` rule it is possible to have a `Num` rule, in our case with the value `100`, this value is than the name of the `Num` rule in our outline view.

```
receiveMoney(100)
```

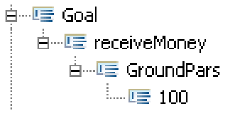


Figure 3.13 - Single valued features

The single valued rules in the 2APL IDE plug-in are:

1. Num
2. Var
3. Ident

I have chosen for this construction in order to make sure that the outline view is really representing the code, the end user can now distinguish the multiple same rules by the single valued features inside the rules.

3.12 DSL

A domain-specific language (DSL) is a small programming language, which focuses on a particular domain. Such a domain can be more or less anything, in this case it is the 2APL syntax language. The idea is that its concepts and notation is as close as possible to what you have in mind when you think about a solution in that domain. Of course this concerns problems which can be solved or processed by computers somehow. [16]

The opposite of a DSL is a so called GPL, a General Purpose Language such as Java or any other common programming language. With a GPL you can solve every computer problem, but it might not always be the best way to solve it. [16]

3.13 Multiple Alternatives

The problem with multiple alternatives is that the parser cannot make a decision between different rules, e.g. the static text `true` can also be a `<literal>`, that can be an `<atom>`, that can be an `<ident>` what may contain the text `true`. This example is shown below:

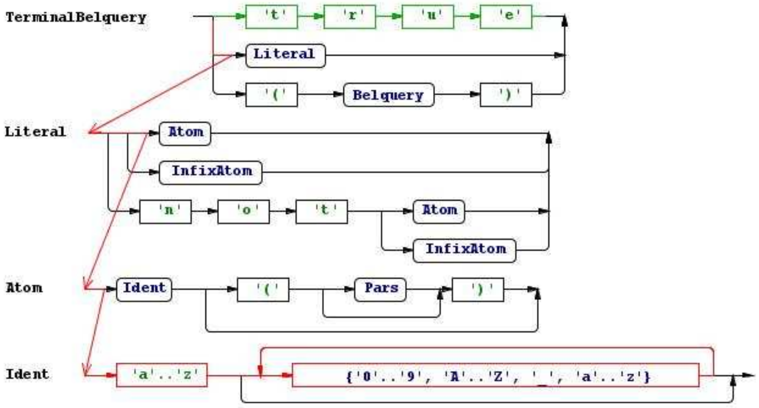


Figure 3.14 - Multiple alternatives example

It can also happen that there are multiple rules applicable, e.g. the rule `<terminalpar>` has `<artexps>` and `<pars>` that when you follow the rules both can have a `<num>` rule. The parser is unable to distinguish the difference. This example is shown below.

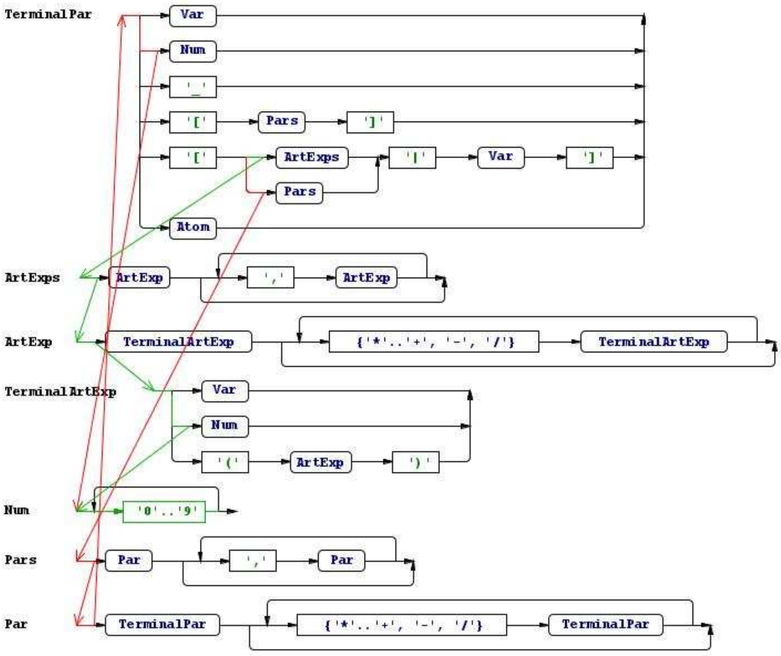


Figure 3.15 - Multiple applicable rules example

By using the backtrack option this problem can be tackled, see chapter 3.14 about backtracking and how to enable this feature.

3.14 Backtracking

Backtracking is a general algorithm for finding all, or some, solutions to computational problem. The backtracking algorithm enumerates a set of partial candidates that, in principle, could be completed in various ways to give all the possible solutions to the given problem. The completion is done incrementally, by a sequence of candidate extension steps. [24]

To enable the backtracking algorithm in this project the Xtext manual provides the following solution [25], this solution was also used in the 2APL IDE plug-in:

1. In the MWE workflow file, in this case `GenerateAPAPL.mwe`, replace:
   `<fragment class="org.eclipse.xtext.generator.AntlrDelegatingFragment"/>`
   with:
   `<fragment class="de.itemis.xtext.antlr.XtextAntlrGeneratorFragment">`
   `<options backtrack="true"/></fragment>`
2. Execute the `MWE workflow` file, see chapter 4.7.

3.15 Project creator

The project creator can be enabled by modifying the `MWE workflow` file and adding two files to the `ui` package (in this case `nl.uu.cs.apapl.ide.ui`) by following the steps in the Xtext user guide [44].

1. Add the following line to the `MWE workflow` file:
   ```
   <fragment
   class="org.eclipse.xtext.ui.generator.projectWizard.SimpleProjectWizard
   Fragment" generatorProjectName="${projectName}.generator"
   modelFileExtension="2apl"/>
   ```
2. Modify the `APAPLCustomNewProjectWizard.java` file in the `nl.uu.cs.apapl.ide.ui.wizard` package, change the `addPages()` function to:

   ```
   public void addPages() {
     mainPage = new WizardNewProjectCreationPage("basicNewProjectPage");
     mainPage.setTitle("2APL Project");
     mainPage.setDescription("Create a new 2APL project.");
     addPage(mainPage);
   }
   ```
3. Modify the `APAPLNewProject.xpt` file, this file holds a default project with 2APL source code that will be created during the project setup. The default source that is shipped with the 2APL IDE plug-in is the push-up agent that was described in chapter 2.2.1. The full `APAPLNewProject.xpt` file can be found in part two of appendix G.

3.16 Syntax highlighting

After executing the `MWE workflow` the generated files will have some default settings regarding syntax highlighting. To change the behavior of the highlighting functionality in the plug-in some changes have to be made to the generated java files. The two files, `APAPLSemanticHighlightConfiguration.java` and `APAPLSemanticHighlightingCalculator.java`, are responsible for the color settings and identifying the various rules that will be given a color configuration. How this is implemented will be explained in the following two subchapters, extracted from chapter 5 of the Xtext manual [26]. To register the two new files in the 2APL IDE plug-in the Xtext generated `APAPLUiModule.java` file has to be modified. In the first part of appendix E the full `APAPLUiModule.java` file can be found.

3.16.1 Highlight configuration

In the APAPLSemanticHighlightConfiguration.java file the various text styles will be defined. In the 2APL IDE plug-in there are four rules that can be given different text styles:

1. Atom
2. UpperAtom
3. GroundAtom
4. ExternalAction

For every of the four rules there is a text style created, this text style holds settings like the background color, text color and font style (e.g. bold or italic). The atom text style source code looks like: [26]

```
public TextStyle atom() {
      TextStyle textStyle = new TextStyle();
      textStyle.setBackgroundColor(new RGB(255, 255, 255));
      textStyle.setColor(new RGB(0, 0, 0));
      return textStyle;
}
```

As per default I ship all the text styles of the four rules in this default format, the end users are free the change the styles in the preferred way their selves. The full source code can be found in the second part of appendix E.

3.16.2 Highlighting calculator

In the APAPLSemanticHighlightingCalculator.java file the different rules are identified and given the text style settings specified earlier. To identify the various predefined rules all the content of the 2APL file will be parsed into an EObject, that can then check if the object contains a rule that needs to be given a text style. To walk through all these objects the following code is used to identify an atom rule: [26]

```
EObject current = iter.next();
if(current instanceof Atom){
      AbstractNode node = null;
      NodeAdapter adapter = NodeUtil.getNodeAdapter(current);
      if (adapter != null) {
            CompositeNode nodeC = adapter.getParserNode();
            if (nodeC != null) {
                  for (AbstractNode child: nodeC.getChildren()) {
                        if (child instanceof LeafNode) {
                              node = child;
                              highlightNode(node,
APAPLSemanticHighlightConfiguration.ATOM, acceptor);
                        }
                  }
            }
      }
}
```

3.17 2APL syntax

The 2APL syntax is written in the EBNF standard, in chapter 6 it is described how the syntax was at the start of this project, what and why changes to the syntax have been made and a description of all the rules is given.

By making a code example for every rule a formal test is executed, every rule has therefore tested as shown in chapter 5.5 and appendix H.

3.18 Run configuration

A run configuration is made to execute the 2APL project from the IDE. In chapter 4.5 it is described how to configure the 2APL IDE plug-in to execute the project. To obtain this functionality, a small change to the 2APL platform had to be made.

The run configuration starts the 2APL platform and sends some arguments to the `2apl.jar` file, the full arguments in the 2APL IDE plug-in are `-jar 2apl.jar –nojade “${selected_recourse_loc}”` this executes the jar file and in this case disables the jade functionality inside the 2APL platform, then the parameter `${selected_recourse_loc}` is send that contains the full path to the 2APL project that is active in the 2APL IDE.

In the 2APL platform, this parameter is used to load the MAS file that contains the agent configuration of the 2APL project, the 2APL platform searches through the directory that is given by this parameter for a file that has a `.mas` extension. If found, this file is loaded and executed in the 2APL platform, and if there is no MAS file found, an error will be thrown to inform the user.

In appendix I the java code that has been added to the `APAPL.java` file is attached. Note that adding this functionality has no impact on the way the 2APL platform works.

The run configuration option in the 2APL platform was first implemented by Michal Cap [33], a master student at Utrecht University. This functionality only provided the loading of a selected file, what is enough to run the 2APL platform from the command line, to have the desired functionality for the 2APL IDE plug-in I modified his code so that a directory is checked to find a MAS file.

4 2APL IDE plug-in

In the previous two chapters it has been described how to create an IDE. This chapter covers the question of how to use the 2APL IDE plug-in. In this chapter, the various steps to install, work with and tune the IDE as you like will be described. To use the 2APL IDE plug-in, it is required to have basic knowledge of the Eclipse platform.

4.1 Install the 2APL IDE plug-in

In order to install the 2APL IDE plug-in you need to follow the next procedure:

1. Download the Eclipse Xtext application from its homepage at: http://www.eclipse.org/Xtext/download
2. Extract the contents of the file into a directory. In this sequel, we assume that this directory is named `eclipse`.
3. Download the 2APL IDE plug-in files from the 2APL website at: http://apapl.sourceforge.net/?page_id=7
4. Extract the files to the eclipse plug-in directory named `eclipse\plugins`.

You can now use the 2APL IDE as part of the Eclipse framework by executing Eclipse, the 2APL IDE plug-in is then automatically loaded. The 2APL IDE plug-in has been tested with `Eclipse 3.5.1`, `Xtext 0.7.2`, under `Windows` and `Mac OS X`.

4.1.1 In the background of the plug-in

This plug-in is created by Xtext, Xtext creates, as described in chapter 3.3, a plug-in for Eclipse. As I did not change any of the code to get the plug-in working with Eclipse, i.e. Xtext takes fully care of this part, I will only show how Xtext is integrated with EMF.

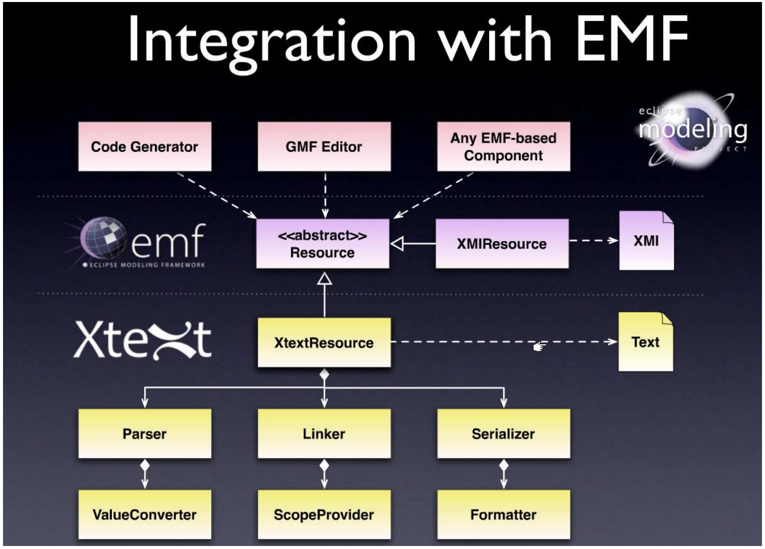
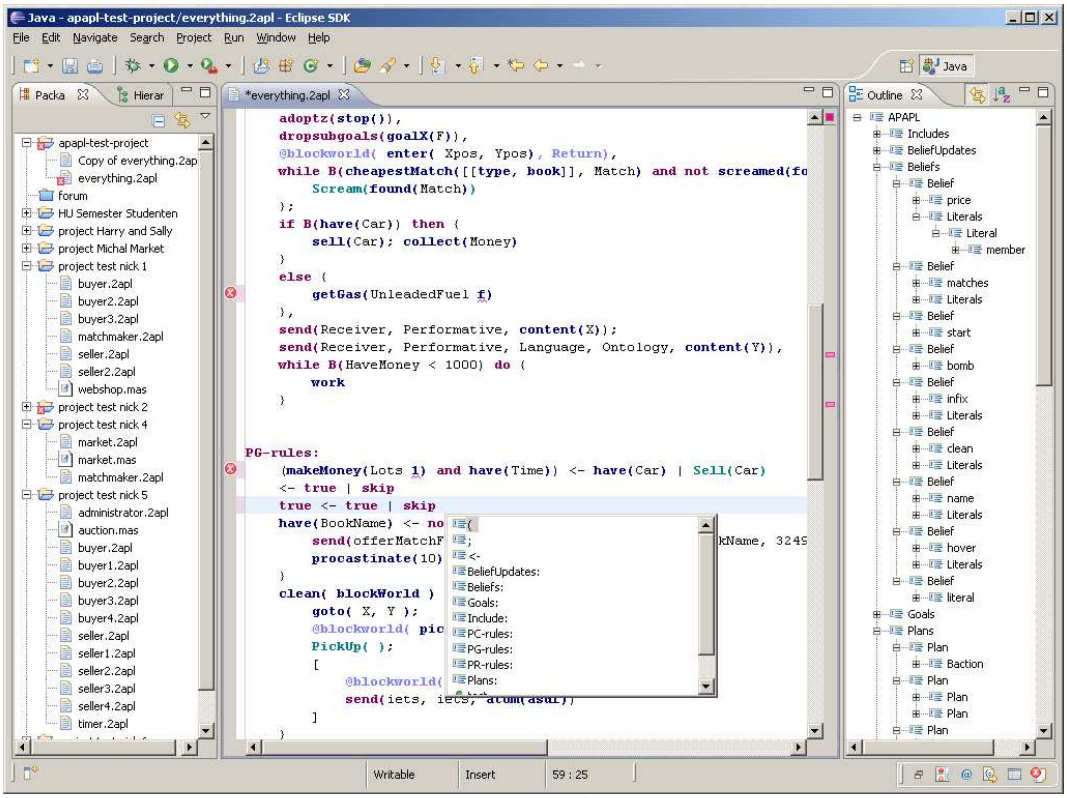


Figure 4.1 - Xtext integration with EMF [45]

For more information regarding the integration of the plug-in with Eclipse see the auto generated `AbstractAPAPLRuntimeModule.java` file, attached in appendix J, how the various options are binded in the framework.

4.2 Look and feel of the IDE



```
    adoptz(stop()),
    dropsubgoals(goalX(F)),
    @blockworld( enter( Xpos, Ypos), Return),
    while B(cheapestMatch([[type, book]], Match) and not screamed(fo
        Scream(found(Match))
    );
    if B(have(Car)) then {
        sell(Car); collect(Money)
    }
    else {
        getGas(UnleadedFuel f)
    },
    send(Receiver, Performative, content(X));
    send(Receiver, Performative, Language, Ontology, content(Y)),
    while B(HaveMoney < 1000) do {
        work
    }


PG-rules:
    (makeMoney(Lots 1) and have(Time)) <- have(Car) | Sell(Car)
    <- true | skip
    true <- true | skip
    have(BookName) <- no
        send(offerMatchF                                    kName, 3249
        procastinate(10)
    }
    clean( blockWorld )
        goto( X, Y );
        @blockworld( pic
        PickUp( );
        [
            @blockworld(
            send(iets, iets, atom(asdf))
        ]
    }
```

Figure 4.2 - IDE for 2APL

In the left column the Package explorer is shown, this gives an overview of all the available projects, shown as folders, available in the IDE. A project can be opened to get an overview of the files inside the project.

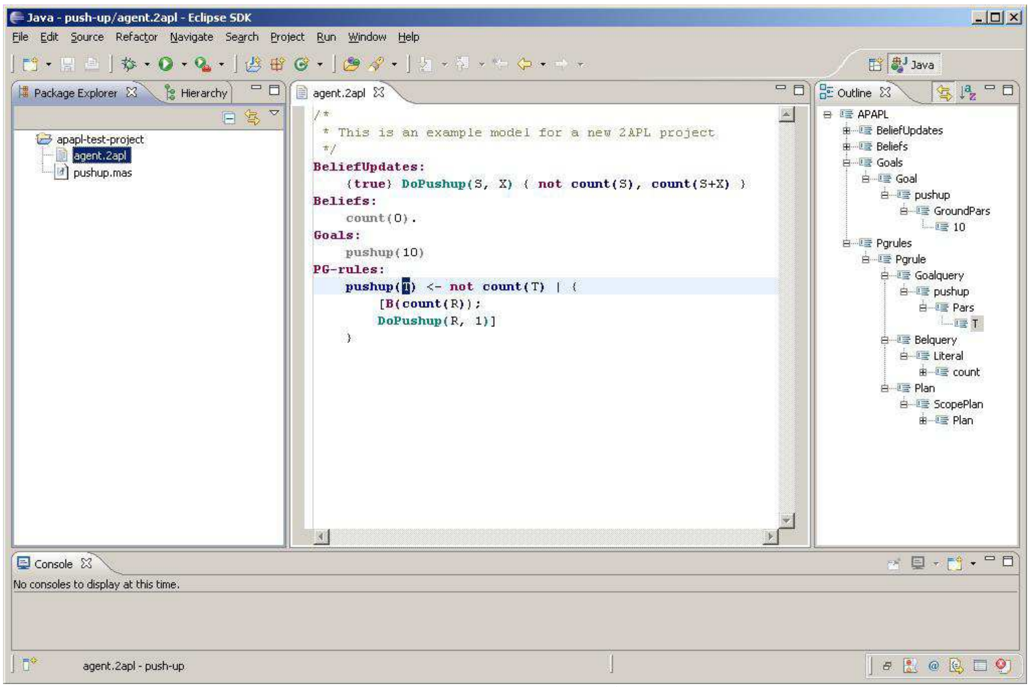
In the center column the editor is active, this editor shows the opened file and applies all the futures available in the IDE such as syntax coloring and code validation checks.

In the right column the outline view is presented that shows the outline view of the current opened file, that is also the file opened in the center column.

4.3 Create a new 2APL project

In order to work with the IDE you need to create a project. A project consists of one MAS file and one or multiple 2APL files. Follow the instructions to create a new project, when finished you can modify the files the way you like, add new files and run the project from the IDE.

1. Create a new project in the Eclipse Xtext version that has the 2APL IDE plug-in installed
2. In the project wizard, choose APAPL project and give the project a name (e.g. push-up)
3. A template 2APL and MAS file are created within the new project



```
/*
 * This is an example model for a new 2APL project
 */
BeliefUpdates:
    {true} DoPushup(S, X) { not count(S), count(S+X) }
Beliefs:
    count(0).
Goals:
    pushup(10)
PG-rules:
    pushup(T) <- not count(T) | {
        [B(count(R));
        DoPushup(R, 1)]
    }
```

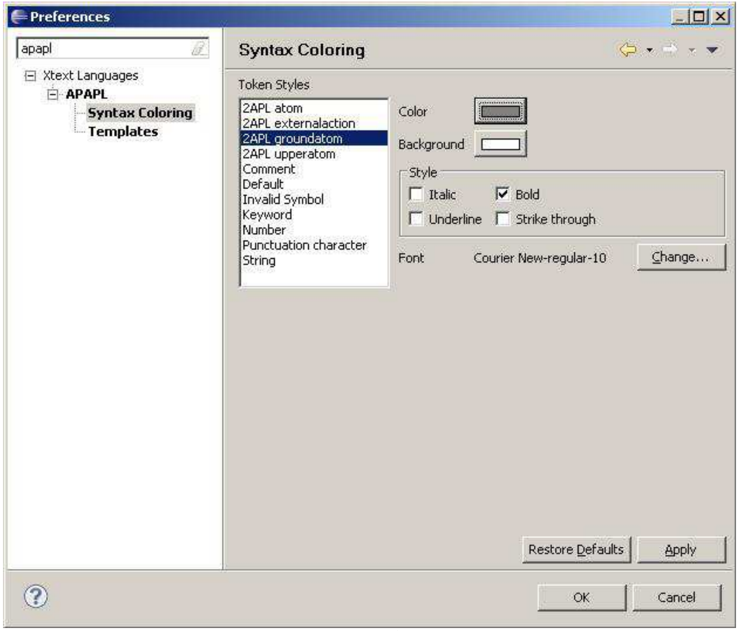
```
APAPL
    BeliefUpdates
    Beliefs
    Goals
        Goal
            pushup
                GroundPars
                    10
    Pgrules
        Pgrule
            Goalquery
                pushup
                    Pars
                        T
            Belquery
                Literal
                    count
            Plan
                ScopePlan
                    Plan
```

No consoles to display at this time.

Figure 4.3 - New created project

4.4 Change syntax colors in the IDE

In the IDE it is possible to change the color of the 2APL code. To do this go to the `Preference` page, under the menu `Window`, and select `Syntax Coloring` under `Xtext Languages` and then `APAPL`.



Token Styles
- 2APL atom
- 2APL externalaction
- 2APL groundatom
- 2APL upperatom
- Comment
- Default
- Invalid Symbol
- Keyword
- Number
- Punctuation character
- String

Color
Background
Style
☐ Italic ☑ Bold
☐ Underline ☐ Strike through
Font Courier New-regular-10 Change...

Figure 4.4 - Color settings

Default only the `Keyword` token has a color (Red and bold), all the other tokens have the color black. It is possible to change the token appearance of `2APL atom`, `2APL externalaction`, `2APL groundatom` and `2APL upperatom`.

4.5 Run the project from the IDE

It is possible to run the project form the IDE, this makes the sequence of `trial and error` a lot faster. In order to run the project from the IDE a `run configuration` has to be added to the IDE.

1. Open the `External Tools Configurations`
2. In the External Tools Configurations window, right click `program` and select `new`

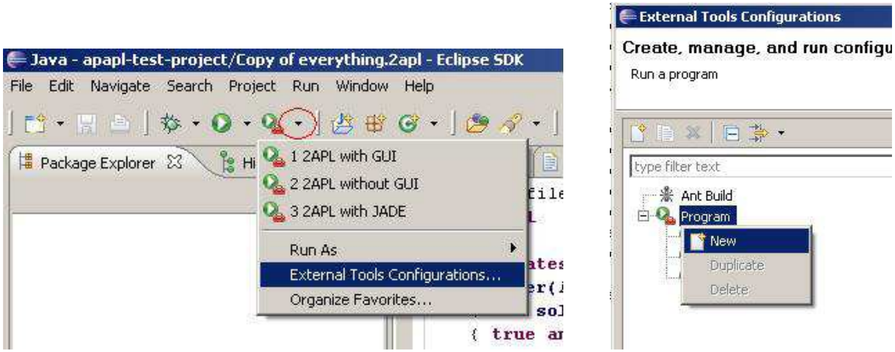
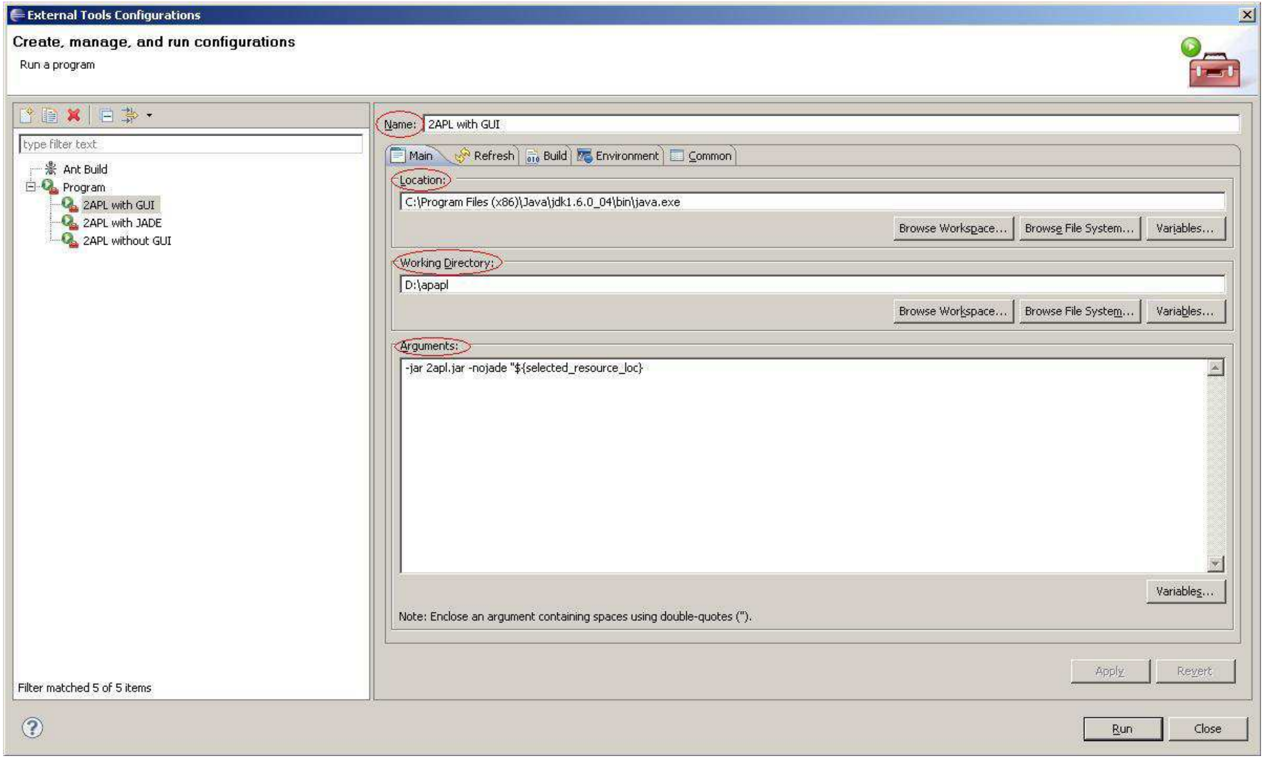

Java - apapl-test-project/Copy of everything.2apl - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Package Explorer
1 2APL with GUI
2 2APL without GUI
3 2APL with JADE
Run As
External Tools Configurations...
Organize Favorites...
External Tools Configurations
Create, manage, and run configu
Run a program
type filter text
Ant Build
Program
New
Duplicate
Delete

Figure 4.5 - External tools configuration 1

3. In the configurations window:
   a. Name: a name for the run configuration
      (e.g. `2APL with GUI`)
   b. Location: the full path to your java.exe
      (e.g. `C:\Program Files (x86)\Java\jdk1.6.0_04\bin\java.exe`)
   c. Working Directory: the path to your 2APL directory
      (e.g. `D:\apapl`)
   d. Arguments: `-jar 2apl.jar –nojade “${selected_recourse_loc }”`



Name: 2APL with GUI

Location:
C:\Program Files (x86)\Java\jdk1.6.0_04\bin\java.exe

Working Directory:
D:\apapl

Arguments:
-jar 2apl.jar -nojade "${selected_resource_loc}"

Note: Enclose an argument containing spaces using double-quotes (").

Figure 4.6 - External tools configuration 2

5 2APL syntax

This chapter will review the 2APL syntax. During the project, many changes in the syntax have been made. An overview of these changes can be found in chapter 5.4. The syntax is expressed in the EBNF, Extended Backus-Naur Form [6], standard. Note that changing the syntax of the rules are made in a way that the 2APL platform does not have to be modified.

5.1 EBNF legenda

```
<name>       = rule name
{<rule>}     = 0..* (This rule can persist zero or multiple times)
[<rule>]     = 0..1 (This rule is optional)
<rule>+      = 1..* (This rule can persist one or multiple times)
<A> | <B>    = choice between rule name A and B
(<A> | <B>) = choice between rule name A and B within brackets
"var"        = static text
;            = end of the rule
```

5.2 Current EBNF 2APL syntax

The current 2APL syntax specifies <atom> as a Prolog like atomic formula starting with a lowercase letter, <Atom> to denote a Prolog like atomic formula starting with a capital letter, <ident> to denote a string, <Var> to denote a string starting with a capital letter and <ground_atom> to denote a grounded atomic formula. [7, 19]

Because of the missing specification of these rules, it is unclear what an atom rule should look like. This problem does not only exist for the atom rule, but for all the unspecified rules in the current EBNF 2APL syntax. The full syntax can be found in appendix A. [19]

5.3 New EBNF 2APL syntax

The new syntax is a rewrite of the current syntax. It can also be found in appendix B.

```
<APAPL>               =  { "Include:" <includes>
                      |  "BeliefUpdates:" <beliefupdates>
                      |  "Beliefs:" <beliefs>
                      |  "Goals:" <goals>
                      |  "Plans:" <plans>
                      |  "PG-rules:" <pgrules>
                      |  "PC-rules:" <pcrules>
                      |  "PR-rules:" <prrules> };
<includes>            =  <include>+;
<include>             =  <ident> ".2apl";
<beliefupdates>       =  <beliefupdate>+;
<beliefupdate>        =  "{" [<belquery>] "}" <beliefupdatename> "{" <literals> "}";
<beliefupdatename>    =  <upperatom>;
<beliefs>             =  <belief>+;
<belief>              =  <ground_atom>"."
                      |  <atom> ":-" <literals> ".";
<goals>               =  <goal> {"," <goal>};
<goal>                =  <ground_atom> {"and" <ground_atom>};
<baction>             =  "skip"
                      |  <beliefupdatename>
                      |  <sendaction>
                      |  <externalaction>
                      |  <abstractaction>
                      |  <test>
                      |  <adoptgoal>
                      |  <dropgoal>;
<plans>               =  <plan> {"," <plan>};
```

```
<plan>              =  <baction>
                    |  <sequenceplan>
                    |  <ifplan>
                    |  <whileplan>
                    |  <atomicplan>
                    |  <scopeplan>;
<sendaction>        =  "send(" <iv> "," <iv> "," <atom> ")"
                    |  "send(" <iv> "," <iv> "," <iv> "," <iv> "," <atom> ")";
<externalaction>    =  "@" <ident> "(" <atom> "," <var> ")";
<abstractaction>    =  <atom>;
<test>              =  "B(" <belquery> ")"
                    |  "G(" <goalquery> ")"
                    |  <test> "&" <test>
                    |  "(" <test> ")";
<adoptgoal>         =  "adopta(" <goalvar> ")"
                    |  "adoptz(" <goalvar> ")";
<dropgoal>          =  "dropgoal(" <goalvar> ")"
                    |  "dropsubgoals(" <goalvar> ")"
                    |  "dropsupergoals(" <goalvar> ")";
<ifplan>            =  "if" <test> "then" <scopeplan> ["else" <scopeplan>];
<whileplan>         =  "while" <test> "do" <scopeplan>;
<atomicplan>        =  "[" <plan> "]";
<scopeplan>         =  "{" <plan> "}";
<pgrules>           =  <pgrule>+;
<pgrule>            =  [<goalquery>] "<-" <belquery> "|" <plan>;
<pcrules>           =  <pcrule>+;
<pcrule>            =  <atom> "<-" <belquery> "|" <plan>;
<prrules>           =  <prrule>+;
<prrule>            =  <planvar> "<-" <belquery> "|" <planvar>;
<goalvar>           =  <atom> {"and" <atom>};
<planvar>           =  <plan>
                    |  <var>
                    |  "if" <test> "then" <scopeplanvar> ["else" <scopelanvar>]
                    |  "while" <test> "do" <scopeplanvar>
                    |  <planvar> ";" <planvar>;
<scopeplanvar>      =  "{" <planvar> "}";
<literals>          =  <literal> {"," <literal>};
<literal>           =  <atom>
                    |  <infixatom>
                    |  "not" <atom>
                    |  "not" <infixatom>;
<belquery>          =  "true"
                    |  <belquery> "and" <belquery>
                    |  <belquery> "or" <belquery>
                    |  "(" <belquery ")"
                    |  <literal>;
<goalquery>         =  "true"
                    |  <goalquery> "and" <goalquery>
                    |  <goalquery> "or" <goalquery>
                    |  "(" <goalquery> ")"
                    |  <atom>;
<iv>                =  <ident> | <var>;
<groundatom>        =  <ident> "(" <groundpars> ")";
<groundpars>        =  <groundpar> {"," <groundpar>};
<groundpar>         =  <ident> | <num> | "_"
                    |  "[" [<groundpars>] "]"
                    |  "[" <groundpars> "|" <var> "]";
<upperatom>         =  <var> "(" [<pars>] ")";
<atom>              =  <ident> ["(" [<pars>] ")"];
<infixatom>         =  <par> ("=" | ">" | "<" | "=" | "<=" | ">=" | "=>" | "=<")
                       <par>;
<pars>              =  <par> {"," <par>};
<par>               =  <var> | <num> | "_" | <atom>
                    |  <par> ("+" | "-" | "*" | "/") <par>
                    |  "[" <pars> "]"
                    |  "[" (<artexps> | <pars>) "|" <var> "]";
```

```
<artexps>            =   <artexp> {"," <artexp>};
<artexp>             =   <var> | <num>
                     |   <artexp> ("+" | "-" | "*" | "/") <artexp>
                     |   "(" <artexp> ")";
<var>                =   "A".."Z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
<ident>              =   "a".."z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
<num>                =   ("0".."9")+;
```

5.4 Differences

The differences are made without the need to change the existing program and can be grouped in 2 parts, one is the group that holds the changes for a better overview, like includes and beliefupdates and the other group contain the rules that where not specified like atom and var.

In the new version of the 2APL syntax the following rules have been added or specified:

	Rule name	Description
1.	<APAPL>	This rule overrides the old <Agent_Prog> rule
2.	<includes>	This rule is created to have multiple <include> rules
3.	<include>	This rule specifies one singe file that is included
4.	<beliefupdates>	This rule is created to have multiple <beliefupdate> rules
5.	<beliefupdatename>	This rule is created to have a better overview in the <beliefupdate> rule itself
6.	<beliefs>	This rule is created to have multiple <belief> rules
7.	<groundatom>	This rule specifies an <atom> like rule that should have one or more parameters
8.	<groundpars>	This rule is created to have multiple <groundpar> rules
9.	<groundpar>	This rule specifies all the parameters within a <groundatom> rule
10.	<upperatom>	This rule is identical to the <atom> rule but it starts with a capital letter
11.	<atom>	This rule specifies how an <atom> rule should be implemented
12.	<infixatom>	This rule is a special type of <atom>
13.	<pars>	This rule is created to have multiple <par> rules
14.	<par>	This rule specifies all the parameters within an <atom>, <upperatom> or <infixatom> rules
15.	<artexps>	This rule is created to have multiple <artexp> rules
16.	<artexp>	This rule specifies the arithmetic expressions within a <par> rule
17.	<var>	This rule specifies a string starting with a capital letter
18.	<ident>	This rule specifies a string starting with a lowercase letter
19.	<num>	This rule specifies a number

The following rules have been deleted:

	Rule name	Description
1.	<Agent_Prog>	This rule is replaced by the <APAPL> rule
2.	<BelUpSpec>	This rule is replaced by the <beliefupdate> rule

The following rules have been modified:

	Rule name
1.	<beliefupdate>
2.	<plan>
3.	<sendaction>
4.	<test>
5.	<literal>

In appendix H is described what has changed for this five rules.

5.5 2APL rules in detail

In total there are 50 rules in the 2APL syntax, all of these rules are well documented in appendix H. Every rule is described in the following seven ways:

1. Old EBNF syntax:
   If the rule syntax has been updated this area shows the old syntax EBNF syntax
2. EBNF syntax:
   The full syntax for the rule is shown here
3. Xtext syntax:
   The Xtext syntax is shown here, if needed rules are left-factored and operators are added to create the wanted outline view
4. ANTLR syntax diagram:
   This diagram is added to have a graphical view of the rule, it shows that when a rule is chosen what various steps are available to take next
5. Description:
   In the description I briefly describe how the rule can be used, here I do not describe why a rule exists in the 2APL platform, for this I suggest to read the 2APL user manual [7, 19]
6. Outline view:
   The 2APL IDE plug-in gives an outline view for the rule that I here show to have a better oversight of the rule
7. Example:
   An example of the rule is given, here I use different coloring settings for the various rule to distinguish the rules better

6 Evaluation

Overall, I am very satisfied with the result of the project. When I started my search for existing IDE's, Eclipse and Xtext came up very fast, resulting in a good start of my project. Then I found out that not all the rules of 2APL where described in the existing syntax, so these had to be added or existing rules had to be modified. This took more time then I anticipated, and at the start of the project I did not even calculate the time for this research in the project. Adding and modifying the 2APL syntax got me right into the left-recursion problem, some extra changes had to be made to the Xtext grammar, but the documentation available for this problem on the internet was more then enough to solve this matter.

If I look at the end product, the 2APL IDE plug-in that is now available for all the 2APL developers, I could not have thought that all the options I wanted in the plug-in have been realized. Of course there is more functionality to be added in a later stage, but considering the short time for this project I am more then pleased with the result.

When I would have to make an other IDE, I would definitely use the same structure as I did in this project, but I would require a complete syntax of the DSL before starting. During this project I had the time to work on the syntax, but would have asked the 2APL development team to work on this if I came in a short of time as this task was very time-consuming. I think that because of the use of a Prolog engine in 2APL the syntax is incomplete, if I check what rules are described in the existing syntax, all of these rules are specific 2APL rules that refer in some cases to a Prolog rule. For this project all the available rules had to be described, so just referring to a rule in Prolog would not work within the Xtext project. Working on the 2APL syntax gained me quite some experience with the EBNF standard and the left-recursion problem.

Not only was this project the end of my Bachelor study, it is also the beginning of my Master study and while working on this project I attended the Intelligent agents course [14] from John-Jules Meyer and the Multi-agent systems course [15] from Mehdi Dastani. Attending these courses gave me a better insight into the world of agents and into how 2APL is used in the agent-oriented world. All this will hopefully result in a jump start towards my Master study.

7 Conclusion and future work

During this project I have gained a good impression about how hard it is to create a programming language. Although the 2APL language is relatively small compared to Java or C, it still manages to design agent applications on a BDI model. Due to the fact that a good IDE was missing for 2APL, it was very hard for programmers to create a bug free application without having to study the full 2APL syntax. Since the creation of the IDE I received quite some positive feedback from people that used it, or even heard about it. I hope that the use my IDE in the next Multi-agent programming course at Utrecht University will help 2APL developers to create, test and execute their projects.

To optimize the created IDE some additional features can be created, and of course it is possible that during the Multi-agent programming course bugs are found in the IDE that need to be solved.

Additional futures that can be implemented later are:

1. Auto code formatting
   The java editor in Eclipse has the function to press `cltr + shift + F` to format the code automatically, this can be added in a later version of the 2APL IDE plug-in as well.
2. MAS file editor
   To create a project in 2APL a mas file is required, an editor could be created for the relative small mas files in projects. The latest development shows that the possibility exists that this mas file will be replaced by a xml file. But also for this xml file a modified editor can be build.
3. 2APL pull down menu with help options
   This menu should be available next to the Help section of Eclipse, it can hold a link to the documentation of 2APL an other references.
4. Environment creator for the 2APL platform
   A new editor can be build to create environments for 2APL. Recent development indicates that this environment will be changed to a default standard so the environment will also work with multi-agent programming languages such as Jason, agentspeak or jadex.
5. Negative integers and doubles
   It is possible to run the 2APL platform with a negative integer, but the 2APL IDE does not recognize a negative number as valid, this simply because it was not described in the 2APL syntax. This syntax should be changed in a way to allow negative numbers and doubles.

8 References

[1] www.eclipse.org accessed 19 November 2009

[2] http://netbeans.org accessed 19 November 2009

[3] www.textpad.com accessed 19 November 2009

[4] www.jext.org accessed 19 November 2009

[5] www.eclipse.org/Xtext accessed 19 November 2009 and 1 December 2009

[6] http://www.cs.cmu.edu/~pattis/misc/ebnf.pdf accessed 19 November 2009

[7] 2APL user guide http://www.cs.uu.nl/2apl/downloads/userguide.pdf accessed 19 November 2009

[8] www.cs.uu.nl/2apl accessed 1 December 2009

[9] http://www.eclipse.org/modeling/emft/?project=ecoretools accessed 7 December 2009

[10] M. Wooldridge. An Introduction to Multiagent Systems. John Wiley and Sons Ltd, February 2002.

[11] http://www.eclipse.org/Xtext/documentation/0_7_2/xtext.html#WhatisXtext accessed 8 December 2009

[12] http://www.slideshare.net/HeikoB/xtext-webinar slide 28 accessed 8 December 2009

[13] http://www.slideshare.net/peterfriese/building-dsls-with-eclipse-1916333 (Slide 65) Accessed 2 December 2009

[14] http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOIAG&jaar=2009

[15] http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOMAS&jaar=2009

[16] http://www.eclipse.org/Xtext/documentation/0_7_2/xtext.html#DSL accessed 1 December 2009

[17] http://www.eclipse.org/org/ accessed 8 December 2009

[18] http://www.antlr.org/ accessed 11 December 2009

[19] Mehdi Dastani, 2APL: a practical agent programming language, International Journal of Autonomous Agents and Multi-Agent Systems (JAAMAS), 16(3):214-248, Special Issue on Computational Logic-based Agents, (eds.) Francesca Toni and Jamal Bentahar, 2008.

[20] http://www.cs.uu.nl/docs/vakken/mas/slides/introduction.pdf slide 10 and 11 accessed 29 January 2010

[21] http://www.eclipse.org/modeling/emft/?project=ecoretools accessed 1 February 2010

[22] http://wiki.eclipse.org/Modeling_Workflow_Engine_(MWE) accessed 1 February 2010

[23] http://ant.apache.org/ accessed 1 February 20103

[24] http://en.wikipedia.org/wiki/Backtracking accessed 10 February 2010

[25] http://wiki.eclipse.org/Xtext/FAQ#OK.2C_but_I_didn.27t_get_these_warnings_in_oAW_Xtext.C2.A0.21 accessed 10 February 2010

[26] http://www.eclipse.org/Xtext/documentation/0_7_0/xtext.html#highlighting accessed 10 February 2010

[27] http://www.eclipse.org/Xtext/documentation/0_7_0/xtext.html#rules accessed 11 February 2010

[28] http://www.eclipse.org/modeling/emf/ accessed 11 February 2010

[29] http://www.theserverside.com/tt/articles/article.tss?l=PragmaticGen accessed 11 February 2010

[30] http://www.eclipse.org/Xtext/documentation/0_7_0/xtext.html#rules accessed 11 February 2010

[31] http://www.cs.uu.nl/education/vak.php?stijl=2&vak=INFOMAP&jaar=2008 accessed 15 February 2010

[32] http://people.cs.uu.nl/mehdi/ accessed 16 February 2010

[33] http://apapl.svn.sourceforge.net/viewvc/apapl/trunk/src/APAPL.java?view=markup&pathrev=25 accessed 17 February 2010

[34] http://en.wikipedia.org/wiki/Plug-in_(computing) accessed 19 February 2010

[35] http://www.eclipse.org/articles/Article-Your%20First%20Plug-in/YourFirstPlugin.html accessed 19 February 2010

[36] http://en.wikipedia.org/wiki/Parsing accessed 19 February 2010

[37] http://dinosaur.compilertools.net/yacc/index.html accessed 19 February 2010

[38] http://www.gnu.org/software/bison/ accessed 19 February 2010

[39] http://www.ssw.uni-linz.ac.at/Coco/ accessed 19 February 2010

[40] http://xtext.itemis.com/ accessed 2 March 2010

[41] http://www.eclipse.org/forums/index.php?t=msg&goto=480994&#msg_480994 accessed August 2009
[42] http://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_Form accessed 2 March 2010
[43] http://www.eclipse.org/forums/index.php?t=showposts&id=67905& accessed 2 Marc 2010
[44] http://www.eclipse.org/Xtext/documentation/latest/xtext.html#projectwizard accessed 2 March 2010
[45] http://www.slideshare.net/HeikoB/xtext-webinar slide 33, accessed 4 March 2010

9 Appendices

9.1 Appendix A - Old EBNF 2APL syntax

```
<Agent_Prog>        =   { "Include:" <ident>
                    |   "BeliefUpdates:" <BelUpSpec>
                    |   "Beliefs:" <belief>
                    |   "Goals:" <goals>
                    |   "Plans:" <plans>
                    |   "PG-rules:" <pgrules>
                    |   "PC-rules:" <pcrules>
                    |   "PR-rules:" <prrules> };
<BelUpSpec>         =   "{"<belquery>"}" <beliefupdate> "{"<literals>"}";
<beliefupdate>      =   <Atom>;
<belief>            =   <ground_atom>"."
                    |   <atom> ":-" <literals>".";
<goals>             =   <goal> {"," <goal>};
<goal>              =   <ground_atom> {"and" <ground_atom>};
<baction>           =   "skip"
                    |   <beliefupdatename>
                    |   <sendaction>
                    |   <externalaction>
                    |   <abstractaction>
                    |   <test>
                    |   <adoptgoal>
                    |   <dropgoal>;
<plans>             =   <plan> {"," <plan>};
<plan>              =   <baction>
                    |   <sequenceplan>
                    |   <ifplan>
                    |   <whileplan>
                    |   <atomicplan>;
<sendaction>        =   "send(" <iv> "," <iv> "," <atom> ")";
                    |   "send(" <iv> "," <iv> "," <iv> "," <iv> "," <atom> ")";
<externalaction>    =   "@" <ident> "(" <atom> "," <var> ")";
<abstractaction>    =   <atom>;
<test>              =   "B(" <belquery> ")"
                    |   "G(" <goalquery> ")"
                    |   <test> "&" <test>;
<adoptgoal>         =   "adopta(" <goalvar> ")"
                    |   "adoptz(" <goalvar> ")";
<dropgoal>          =   "dropgoal(" <goalvar> ")"
                    |   "dropsubgoals(" <goalvar> ")"
                    |   "dropsupergoals(" <goalvar> ")";
<ifplan>            =   "if" <test> "then" <scopeplan> ["else" <scopeplan>];
<whileplan>         =   "while" <test> "do" <scopeplan>;
<atomicplan>        =   "[" <plan> "]";
<scopeplan>         =   "{" <plan> "}";
<pgrules>           =   <pgrule>+;
<pgrule>            =   [<goalquery>] "<-" <belquery> "|" <plan>;
<pcrules>           =   <pcrule>+;
<pcrule>            =   <atom> "<-" <belquery> "|" <plan>;
<prrules>           =   <prrule>+;
<prrule>            =   <planvar> "<-" <belquery> "|" <planvar>;
<goalvar>           =   <atom> {"and" <atom>};
<planvar>           =   <plan>
                    |   <var>
                    |   "if" <test> "then" <scopeplanvar> ["else" <scopelanvar>]
                    |   "while" <test> "do" <scopeplanvar>
                    |   <planvar> ";" <planvar>;
<scopeplanvar>      =   "{" <planvar> "}";
<literals>          =   <literal> {"," <literal>};
```

```
<literal>           =  <atom>
                    |  "not" <atom>;
<belquery>          =  "true"
                    |  <belquery> "and" <belquery>
                    |  <belquery> "or" <belquery>
                    |  "(" <belquery ")"
                    |  <literal>;
<goalquery>         =  "true"
                    |  <goalquery> "and" <goalquery>
                    |  <goalquery> "or" <goalquery>
                    |  "(" <goalquery> ")"
                    |  <atom>;
<iv>                =  <ident> | <var>;
```

9.2 Appendix B – New EBNF 2APL syntax

```
<APAPL>             =   { "Include:" <includes>
                    |   "BeliefUpdates:" <beliefupdates>
                    |   "Beliefs:" <beliefs>
                    |   "Goals:" <goals>
                    |   "Plans:" <plans>
                    |   "PG-rules:" <pgrules>
                    |   "PC-rules:" <pcrules>
                    |   "PR-rules:" <prrules> };
<includes>          =   <include>+;
<include>           =   <ident> ".2apl";
<beliefupdates>     =   <beliefupdate>+;
<beliefupdate>      =   "{" [<belquery>] "}" <beliefupdatename> "{" <literals> "}";
<beliefupdatename>  =   <upperatom>;
<beliefs>           =   <belief>+;
<belief>            =   <ground_atom>"."
                    |   <atom> ":-" <literals> ".";
<goals>             =   <goal> {"," <goal>};
<goal>              =   <ground_atom> {"and" <ground_atom>};
<baction>           =   "skip"
                    |   <beliefupdatename>
                    |   <sendaction>
                    |   <externalaction>
                    |   <abstractaction>
                    |   <test>
                    |   <adoptgoal>
                    |   <dropgoal>;
<plans>             =   <plan> {"," <plan>};
<plan>              =   <baction>
                    |   <sequenceplan>
                    |   <ifplan>
                    |   <whileplan>
                    |   <atomicplan>
                    |   <scopeplan>;
<sendaction>        =   "send(" <iv> "," <iv> "," <atom> ")"
                    |   "send(" <iv> "," <iv> "," <iv> "," <iv> "," <atom> ")";
<externalaction>    =   "@" <ident> "(" <atom> "," <var> ")";
<abstractaction>    =   <atom>;
<test>              =   "B(" <belquery> ")"
                    |   "G(" <goalquery> ")"
                    |   <test> "&" <test>
                    |   "(" <test> ")";
<adoptgoal>         =   "adopta(" <goalvar> ")"
                    |   "adoptz(" <goalvar> ")";
<dropgoal>          =   "dropgoal(" <goalvar> ")"
                    |   "dropsubgoals(" <goalvar> ")"
                    |   "dropsupergoals(" <goalvar> ")";
<ifplan>            =   "if" <test> "then" <scopeplan> ["else" <scopeplan>];
<whileplan>         =   "while" <test> "do" <scopeplan>;
<atomicplan>        =   "[" <plan> "]";
<scopeplan>         =   "{" <plan> "}";
<pgrules>           =   <pgrule>+;
<pgrule>            =   [<goalquery>] "<-" <belquery> "|" <plan>;
<pcrules>           =   <pcrule>+;
<pcrule>            =   <atom> "<-" <belquery> "|" <plan>;
<prrules>           =   <prrule>+;
<prrule>            =   <planvar> "<-" <belquery> "|" <planvar>;
<goalvar>           =   <atom> {"and" <atom>};
<planvar>           =   <plan>
                    |   <var>
                    |   "if" <test> "then" <scopeplanvar> ["else" <scopelanvar>]
                    |   "while" <test> "do" <scopeplanvar>
                    |   <planvar> ";" <planvar>;
```

```
<scopeplanvar>      =  "{" <planvar> "}";
<literals>          =  <literal> {"," <literal>};
<literal>           =  <atom>
                    |  <infixatom>
                    |  "not" <atom>
                    |  "not" <infixatom>;
<belquery>          =  "true"
                    |  <belquery> "and" <belquery>
                    |  <belquery> "or" <belquery>
                    |  "(" <belquery ")"
                    |  <literal>;
<goalquery>         =  "true"
                    |  <goalquery> "and" <goalquery>
                    |  <goalquery> "or" <goalquery>
                    |  "(" <goalquery> ")"
                    |  <atom>;
<iv>                =  <ident> | <var>;
<groundatom>        =  <ident> "(" <groundpars> ")";
<groundpars>        =  <groundpar> {"," <groundpar>};
<groundpar>         =  <ident> | <num> | "_"
                    |  "[" [<groundpars>] "]"
                    |  "[" <groundpars> "|" <var> "]";
<upperatom>         =  <var> "(" [<pars>] ")";
<atom>              =  <ident> ["(" [<pars>] ")"];
<infixatom>         =  <par> ("=" | ">" | "<" | "=" | "<=" | ">=" | "=>" | "=<")
                       <par>;
<pars>              =  <par> {"," <par>};
<par>               =  <var> | <num> | "_" | <atom>
                    |  <par> ("+" | "-" | "*" | "/") <par>
                    |  "[" <pars> "]"
                    |  "[" (<artexps> | <pars>) "|" <var> "]";
<artexps>           =  <artexp> {"," <artexp>};
<artexp>            =  <var> | <num>
                    |  <artexp> ("+" | "-" | "*" | "/") <artexp>
                    |  "(" <artexp> ")";
<var>               =  "A".."Z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
<ident>             =  "a".."z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
<num>               =  ("0".."9")+;
```

9.3 Appendix C – Xtext grammar

```
grammar nl.uu.cs.apapl.ide.APAPL with org.eclipse.xtext.common.Terminals

generate aPAPL "http://www.uu.nl/cs/apapl/ide/APAPL"

//** APAPL **//
APAPL:
        (elements+=
          "Include:" includes+=Includes
        | "BeliefUpdates:" beliefUpdates+=BeliefUpdates
        | "Beliefs:" beliefs+=Beliefs
        | "Goals:" goals+=Goals
        | "Plans:" plans+=Plans
        | "PG-rules:" pgrules+=Pgrules
        | "PC-rules:" pcrules+=Pcrules
        | "PR-rules:" prrules+=Prrules
        )*
        ;

//** Include elemenet **//
Includes:
        (include+=Include)+;
Include:
        includeName=Ident ".2apl";

//** BeliefUpdates element **//
BeliefUpdates:
        (beliefUpdate+=BeliefUpdate)+;
BeliefUpdate:
        "{" (belquery+=Belquery)? "}" beliefUpdateName+=BeliefUpdateName "{"
literals+=Literals "}";
BeliefUpdateName:
        upperatom+=UpperAtom;

//** Belief element **//
Beliefs:
        (belief+=Belief)+;
Belief:
          groundAtom+=GroundAtom "."
        | atom+=Atom ":-" literals+=Literals "."
        ;

//** Belief Query **//
Belquery:
        TerminalBelquery ({Belquery.belquery+=current} (("and" | "or")
belquery+=TerminalBelquery)+)?;
TerminalBelquery returns Belquery:
          "true"
        | literal+=Literal
        | '(' belquery+=Belquery ')'
        ;

//** Goal Query **//
Goalquery:
        TerminalGoalquery ({Goalquery.goalquery+=current} (("and" | "or")
goalquery+=TerminalGoalquery)+)?;
TerminalGoalquery returns Goalquery:
          "true"
        | atom+=Atom
        | '(' goalquery+=Goalquery ')'
        ;
```

```
//** Goals element **//
Goals:
       goal+=Goal ("," goal+=Goal)*;
Goal:
       groundAtom+=GroundAtom ("and" groundAtom+=GroundAtom)*;
Adoptgoal:
         "adopta(" goalvar+=Goalvar ")"
       | "adoptz(" goalvar+=Goalvar ")"
       ;
Dropgoal:
         "dropgoal(" goalvar+=Goalvar ")"
       | "dropsubgoals(" goalvar+=Goalvar ")"
       | "drupsupergoals(" goalvar+=Goalvar ")"
       ;
Goalvar:
       atom+=Atom ("and" atom+=Atom)*;

//** Plans element **//
Plans:
    plan+=Plan ("," plan+=Plan)*;
Plan:
       TerminalPlan ({Plan.plan+=current} (";" plan+=TerminalPlan)+)?;
TerminalPlan returns Plan:
         baction+=Baction
       | ifPlan+=IfPlan
       | whilePlan+=WhilePlan
       | scopePlan+=ScopePlan
       | atomicPlan+=AtomicPlan
       ;
AtomicPlan:
       "[" plan+=Plan "]";
ScopePlan:
       "{" plan+=Plan "}";
IfPlan:
       "if" test+=Test "then" scopePlan+=ScopePlan ("else" scopePlan+=ScopePlan)?;
WhilePlan:
        "while" test+=Test "do" scopePlan+=ScopePlan;

//** Bactions **//
Baction:
         "skip"
       | test+=Test
       | externalAction+=ExternalAction
    | abstractAction+=AbstractAction
       | adoptgoal+=Adoptgoal
       | dropgoal+=Dropgoal
       | sendAction+=SendAction
       | beliefUpdateName+=BeliefUpdateName
       ;

//** Actions **//
ExternalAction:
       "@" name=Ident "(" atom+=Atom "," varname=Var ")";
AbstractAction:
       atom+=Atom;
SendAction:
         "send(" iv+=Iv "," iv+=Iv "," atom+=Atom ")"
       | "send(" iv+=Iv "," iv+=Iv "," iv+=Iv "," iv+=Iv "," atom+=Atom ")"
       ;

//** PG-rules **//
Pgrules:
       (pgrule+=Pgrule)+;
Pgrule:
       (goalquery+=Goalquery)? "<-" belquery+=Belquery "|" plan+=Plan;
```

```
//** PC-rules **//
Pcrules:
        (pcrule+=Pcrule)+;
Pcrule:
        atom+=Atom "<-" belquery+=Belquery "|" plan+=Plan;

//** PR-rules **//
Prrules:
        (prrule+=Prrule)+;
Prrule:
        planvar+=Planvar "<-" belquery+=Belquery "|" planvar+=Planvar;

//** Planvar **//
Planvar:
        TerminalPlanvar ({Planvar.planvar+=current} (";" planvar+=TerminalPlanvar)+)?;
TerminalPlanvar returns Planvar:
          name=Var
        | name="if" test+=Test "then" scopePlanvar+=ScopePlanvar ("else"
scopePlanvar+=ScopePlanvar)?
        | name="while" test+=Test "do" scopePlanvar+=ScopePlanvar
        | plan+=TerminalPlan
        ;
ScopePlanvar:
        "{" planvar+=Planvar "}";

//** Test **//
Test:
        TerminalTest ({Test.test+=current} ("&" test+=TerminalTest)+)?;
TerminalTest returns Test:
        BelqueryTest | GoalqueryTest | "(" test+=Test ")";
BelqueryTest:
        "B(" belquery+=Belquery ")";
GoalqueryTest:
        "G(" goalquery+=Goalquery ")";

Literals:
        literal+=Literal ("," literal+=Literal)*;
Literal:
          atom+=Atom
        | infixatom+=InfixAtom
        | "not" atom+=Atom
        | "not" infixatom+=InfixAtom
        ;
Iv:
        name=Ident | name=Var;

//** Terminal en static datatype rules **//
GroundAtom:
        name=Ident "(" groundpars+=GroundPars ")";
GroundPars:
        groundpar+=GroundPar ("," groundpar+=GroundPar)*;
GroundPar:
          name=Ident
        | name=Num
        | "_"
        | "[" (groundpars+=GroundPars)? "]"
        | "[" groundpars+=GroundPars "|" name=Var "]"
        ;

UpperAtom:
        name=Var '(' (pars+=Pars)? ')';

Atom:
        name=Ident ("(" (pars+=Pars)? ")")?;
```

```
InfixAtom:
       par+=Par ("=" | ">" | "<" | "<=" | ">=" | "=>" | "=<") par+=Par;

Pars:
       par+=Par ("," par+=Par)*;
Par:
       TerminalPar ({Par.par+=current} (("+" | "-" | "*" | "/") par+=TerminalPar)+)?;
TerminalPar returns Par:
         name=Var
       | name=Num
       | name='_'
       | '[' (pars+=Pars)? ']'
       | '[' (artexps+=ArtExps | pars+=Pars) '|' name=Var ']'
       | atom+=Atom
       ;

ArtExps:
       artexp+=ArtExp (',' artexp+=ArtExp)*;
ArtExp:
       TerminalArtExp ({ArtExp.artexp+=current} (("+" | "-" | "*" | "/")
artexp+=TerminalArtExp)+)?;
TerminalArtExp returns ArtExp:
         name=Var
       | name=Num
       | '(' artexp+=ArtExp ')'
       ;

terminal Var:
       ('A'..'Z') ('a'..'z' | 'A'..'Z' | '0'..'9' | "_")*;
terminal Ident:
       ('a'..'z') ('a'..'z' | 'A'..'Z' | '0'..'9' | "_")*;
terminal Num:
       ('0'..'9')+;

terminal SL_COMMENT:
       ('//' | '%') !('\n'|'\r')* ('\r'? '\n')? ;
```

9.4 Appendix D – MWE workflow file

```
<workflow>
       <property file="nl/uu/cs/apapl/ide/GenerateAPAPL.properties"/>

       <property name="runtimeProject" value="../${projectName}"/>

       <bean class="org.eclipse.emf.mwe.utils.StandaloneSetup"
platformUri="${runtimeProject}/.."/>

       <component class="org.eclipse.emf.mwe.utils.DirectoryCleaner"
directory="${runtimeProject}/src-gen"/>
       <component class="org.eclipse.emf.mwe.utils.DirectoryCleaner"
directory="${runtimeProject}.ui/src-gen"/>

       <component class="org.eclipse.xtext.generator.Generator">
             <pathRtProject value="${runtimeProject}"/>
             <pathUiProject value="${runtimeProject}.ui"/>
             <projectNameRt value="${projectName}"/>
             <projectNameUi value="${projectName}.ui"/>

             <language uri="${grammarURI}" fileExtensions="${file.extensions}">
                    <!-- Java API to access grammar elements (required by several other
fragments) -->
                    <fragment
class="org.eclipse.xtext.generator.grammarAccess.GrammarAccessFragment"/>

                    <!-- generates Java API for the generated EPackages -->
                    <fragment
class="org.eclipse.xtext.generator.ecore.EcoreGeneratorFragment"/>

                    <!-- the serialization component -->
                    <fragment
class="org.eclipse.xtext.generator.parseTreeConstructor.ParseTreeConstructorFragment"/>

                    <!-- a custom ResourceFactory for use with EMF -->
                    <fragment
class="org.eclipse.xtext.generator.resourceFactory.ResourceFactoryFragment"
                           fileExtensions="${file.extensions}"/>

                    <!-- the following fragment tries to use the Antlr Generator fragment
which can be installed via update manager from http://download.itemis.com/updates/ -->
                    <!-- <fragment
class="org.eclipse.xtext.generator.AntlrDelegatingFragment" /> -->
                    <fragment class="de.itemis.xtext.antlr.XtextAntlrGeneratorFragment">
                           <options backtrack="true" memoize="true"/>
                    </fragment>

                    <!-- java-based API for validation -->
                    <fragment
class="org.eclipse.xtext.generator.validation.JavaValidatorFragment">
                <composedCheck value="org.eclipse.xtext.validation.ImportUriValidator"/>
            </fragment>

                    <!-- scoping API -->
                    <fragment
class="org.eclipse.xtext.generator.scoping.JavaScopingFragment"/>

                    <!-- formatter API -->
                    <fragment
class="org.eclipse.xtext.generator.formatting.FormatterFragment"/>

                    <!-- labeling API -->
                    <fragment
class="org.eclipse.xtext.ui.generator.labeling.LabelProviderFragment"/>
```

```
                    <!-- outline API -->
                    <fragment
class="org.eclipse.xtext.ui.generator.outline.TransformerFragment"/>
                    <fragment
class="org.eclipse.xtext.ui.generator.outline.OutlineNodeAdapterFactoryFragment"/>

                    <!-- java-based API for content assistance -->
                    <fragment
class="org.eclipse.xtext.ui.generator.contentAssist.JavaBasedContentAssistFragment"/>
                    <!-- the following fragment tries to use the Antlr based content
assist fragment which can be downloaded from http://www.itemis.com
                         and will be ignored if it's not available. -->
                    <fragment
class="org.eclipse.xtext.generator.DelegatingGeneratorFragment"
                           delegate="de.itemis.xtext.antlr.XtextAntlrUiGeneratorFragment"
                           message="You are generating without ANTLR. It is highly
recommended to download and use the plugin 'de.itemis.xtext.antlr' \n\t using the update
site http://download.itemis.com/updates/.">
                    </fragment>
                    <!-- <fragment
class="de.itemis.xtext.antlr.XtextAntlrUiGeneratorFragment"/> -->
                    <fragment
class="de.itemis.xtext.antlr.XtextAntlrUiGeneratorFragment">
                           <options backtrack="true"  memoize="true"/>
                    </fragment>

                    <!-- project wizard (optional) -->
                    <fragment
class="org.eclipse.xtext.ui.generator.projectWizard.SimpleProjectWizardFragment"
generatorProjectName="${projectName}.generator" modelFileExtension="2apl"/>

             </language>
      </component>
</workflow>
```

9.5 Appendix E –Semantic highlight configuration

APAPLUiModule.java

```
package nl.uu.cs.apapl.ide;

import org.eclipse.xtext.ui.common.editor.syntaxcoloring.*;
import org.eclipse.xtext.ui.core.wizard.IProjectCreator;

/**
 * Use this class to register components to be used within the IDE.
 */
public class APAPLUiModule extends nl.uu.cs.apapl.ide.AbstractAPAPLUiModule {
        //** Insert the configuration file for the syntax highlighting **//
        public Class<? extends ISemanticHighlightingConfiguration>
bindISemanticHighlightingConfiguration() {
                return APAPLSemanticHighlightConfiguration.class;
        }
        //** Insert the highlighting calculator that looks if a token is part of one of the
predifined tokens that need to be highlighted **//
        public Class<? extends ISemanticHighlightingCalculator>
bindISemanticHighlightingCalculator() {
                return APAPLSemanticHighlightingCalculator.class;
        }
}
```

APAPLSemanticHighlightConfiguration.java

```
package nl.uu.cs.apapl.ide;

import org.eclipse.swt.SWT;
import org.eclipse.swt.graphics.RGB;
import org.eclipse.xtext.ui.common.editor.syntaxcoloring.*;
import org.eclipse.xtext.ui.core.editor.utils.TextStyle;

public class APAPLSemanticHighlightConfiguration implements
ISemanticHighlightingConfiguration {
        // provide an id string for the highlighting calculator
        public static final String ATOM = "2APL atom";
        public static final String UPPERATOM = "2APL upperatom";
        public static final String GROUNDATOM = "2APL groundatom";
        public static final String EXTERNALACTION = "2APL externalaction";
        // configure the acceptor providing the id, the description string
        // that will appear in the preference page and the initial text style
        // method for calculating an actual text styles
        public void configure(IHighlightingConfigurationAcceptor acceptor) {
                acceptor.acceptDefaultHighlighting(ATOM, "2APL atom", atom());
                acceptor.acceptDefaultHighlighting(UPPERATOM, "2APL upperatom",
upperatom());
                acceptor.acceptDefaultHighlighting(GROUNDATOM, "2APL groundatom",
groundatom());
                acceptor.acceptDefaultHighlighting(EXTERNALACTION, "2APL externalaction",
externalaction());
        }
        public TextStyle atom() {
                TextStyle textStyle = new TextStyle();
                textStyle.setBackgroundColor(new RGB(255, 255, 255));
                textStyle.setColor(new RGB(0, 0, 0));
                return textStyle;
        }
        public TextStyle upperatom() {
                TextStyle textStyle = new TextStyle();
                textStyle.setBackgroundColor(new RGB(255, 255, 255));
                textStyle.setColor(new RGB(0, 0, 0));
                return textStyle;
        }
```

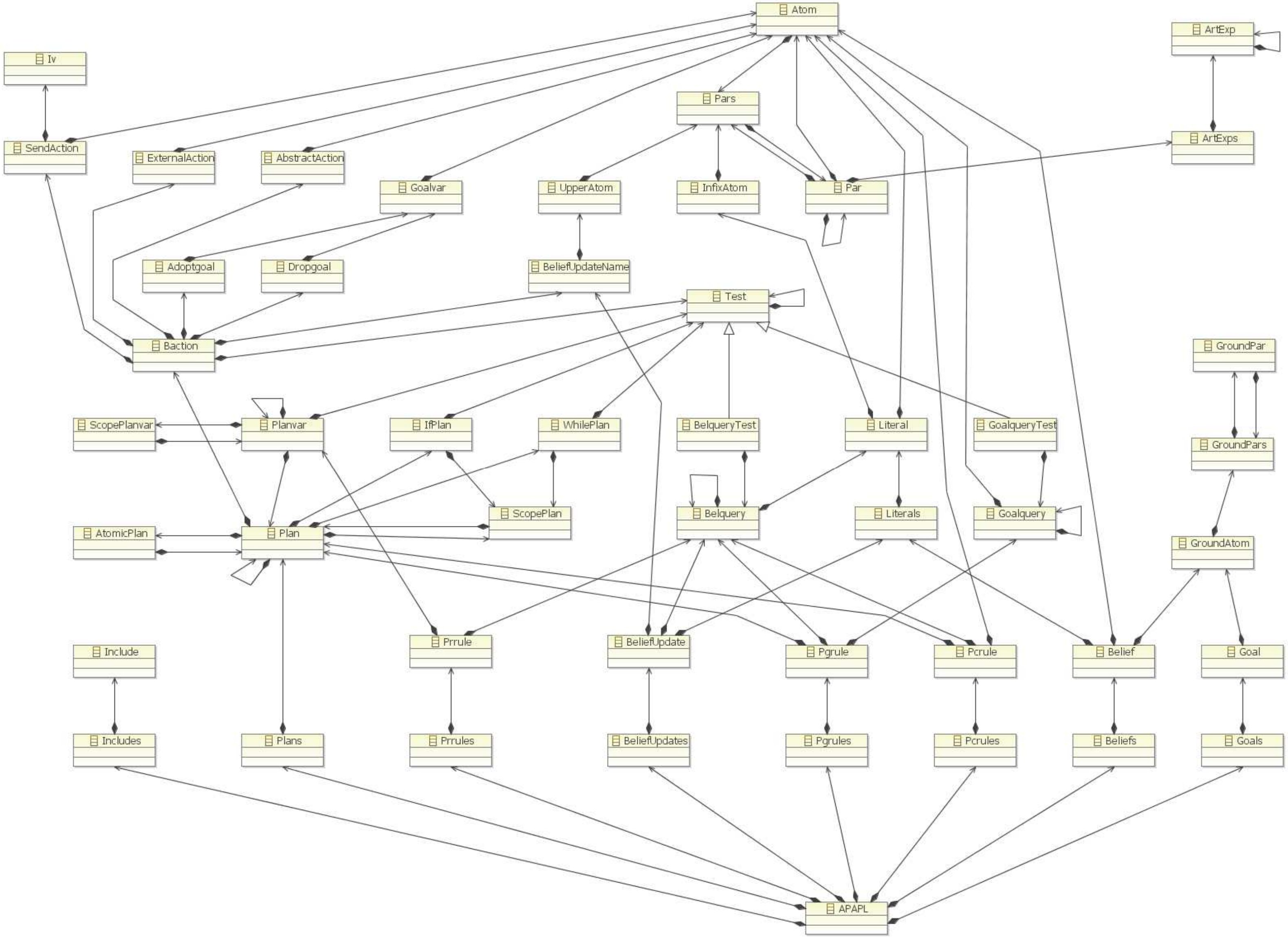
```
	public TextStyle groundatom() {
		TextStyle textStyle = new TextStyle();
		textStyle.setBackgroundColor(new RGB(255, 255, 255));
		textStyle.setColor(new RGB(0, 0, 0));
		return textStyle;
	}

	public TextStyle externalaction() {
		TextStyle textStyle = new TextStyle();
		textStyle.setBackgroundColor(new RGB(255, 255, 255));
		textStyle.setColor(new RGB(0, 0, 0));
		return textStyle;
	}
}
```

APAPLSemanticHighlightingCalculator.java

```
package nl.uu.cs.apapl.ide;

import java.util.*;
import nl.uu.cs.apapl.ide.aPAPL.*;
import org.eclipse.emf.ecore.*;
import org.eclipse.emf.ecore.util.EcoreUtil;
import org.eclipse.xtext.parsetree.*;
import org.eclipse.xtext.resource.*;
import org.eclipse.xtext.ui.common.editor.syntaxcoloring.*;

public class APAPLSemanticHighlightingCalculator implements
ISemanticHighlightingCalculator {

	public void provideHighlightingFor(XtextResource resource,
IHighlightedPositionAcceptor acceptor) {
		if (resource == null)
			return;
		Iterator<EObject> iter = EcoreUtil.getAllContents(resource, true);
		while(iter.hasNext()) {
			EObject current = iter.next();
			if(current instanceof Atom){
				AbstractNode node = null;
				NodeAdapter adapter = NodeUtil.getNodeAdapter(current);
				if (adapter != null) {
					CompositeNode nodeC = adapter.getParserNode();
					if (nodeC != null) {
						for (AbstractNode child: nodeC.getChildren()) {
							if (child instanceof LeafNode) {
								node = child;
								highlightNode(node,
APAPLSemanticHighlightConfiguration.ATOM, acceptor);
							}
						}
					}
				}
			}
			else if(current instanceof UpperAtom){
				AbstractNode node = null;
				NodeAdapter adapter = NodeUtil.getNodeAdapter(current);
				if (adapter != null) {
					CompositeNode nodeC = adapter.getParserNode();
					if (nodeC != null) {
						for (AbstractNode child: nodeC.getChildren()) {
							if (child instanceof LeafNode) {
								node = child;
								highlightNode(node,
APAPLSemanticHighlightConfiguration.UPPERATOM, acceptor);
							}
						}
					}
```

```
                }
            }
            else if(current instanceof GroundAtom){
                AbstractNode node = null;
                NodeAdapter adapter = NodeUtil.getNodeAdapter(current);
                if (adapter != null) {
                    CompositeNode nodeC = adapter.getParserNode();
                    if (nodeC != null) {
                        for (AbstractNode child: nodeC.getChildren()) {
                            if (child instanceof LeafNode) {
                                node = child;
                                highlightNode(node,
APAPLSemanticHighlightConfiguration.GROUNDATOM, acceptor);
                            }
                        }
                    }
                }
            }
            else if(current instanceof ExternalAction){
                AbstractNode node = null;
                NodeAdapter adapter = NodeUtil.getNodeAdapter(current);
                if (adapter != null) {
                    CompositeNode nodeC = adapter.getParserNode();
                    if (nodeC != null) {
                        for (AbstractNode child: nodeC.getChildren()) {
                            if (child instanceof LeafNode) {
                                node = child;
                                highlightNode(node,
APAPLSemanticHighlightConfiguration.EXTERNALACTION, acceptor);
                            }
                        }
                    }
                }
            }
        }
    }
    private void highlightNode(AbstractNode node, String id,
IHighlightedPositionAcceptor acceptor) {
        if (node == null)
            return;
        if (node instanceof LeafNode) {
            acceptor.addPosition(node.getOffset(), node.getLength(), id);
        } else {
            for (LeafNode leaf: node.getLeafNodes()) {
                if (!leaf.isHidden()) {
                    acceptor.addPosition(leaf.getOffset(), leaf.getLength(),
id);
                }
            }
        }
    }


}
```

9.6 Appendix F – Xtext diagram



9.7 Appendix G – Project wizard

APAPLCustomNewProjectWizard.java

```
package nl.uu.cs.apapl.ide.ui.wizard;

import org.eclipse.ui.dialogs.WizardNewProjectCreationPage;
import org.eclipse.xtext.ui.core.wizard.IProjectInfo;
import org.eclipse.xtext.ui.core.wizard.XtextNewProjectWizard;

public class APAPLCustomNewProjectWizard extends XtextNewProjectWizard {

        private WizardNewProjectCreationPage mainPage;

        public APAPLCustomNewProjectWizard() {
                super();
                setWindowTitle("New 2APL Project");
        }

        /**
         * Use this method to add pages to the wizard.
         * The one-time generated version of this class will add a default new project page
to the wizard.
         */
        public void addPages() {
                mainPage = new WizardNewProjectCreationPage("basicNewProjectPage");
                mainPage.setTitle("2APL Project");
                mainPage.setDescription("Create a new 2APL project.");
                addPage(mainPage);
        }

        /**
         * Use this method to read the project settings from the wizard pages and feed them
into the project info class.
         */
        @Override
        protected IProjectInfo getProjectInfo() {
                nl.uu.cs.apapl.ide.ui.wizard.APAPLProjectInfo projectInfo = new
nl.uu.cs.apapl.ide.ui.wizard.APAPLProjectInfo();
                projectInfo.setProjectName(mainPage.getProjectName());
                return projectInfo;
        }

}
```

APAPLNewProject.xpt

```
«IMPORT nl::uu::cs::apapl::ide::ui::wizard»

«DEFINE main FOR APAPLProjectInfo»
«EXPAND model FOR this»
«EXPAND project FOR this»
«ENDDEFINE»

«DEFINE model FOR APAPLProjectInfo»
«FILE "agent.2apl"-»
/*
 * This is an example model for a new 2APL project
 */
BeliefUpdates:
       {true} DoPushup(S, X) { not count(S), count(S+X) }
Beliefs:
       count(0).
Goals:
       pushup(10)
PG-rules:
       pushup(T) <- not count(T) | {
              [B(count(R));
              DoPushup(R, 1)]
       }
«ENDFILE»
«ENDDEFINE»
«DEFINE project FOR APAPLProjectInfo»
«FILE "pushup.mas"-»
/*
 * This is an example project mas (Multi Agent Systems) file for a new 2APL project
 */
pushup : agent.2apl
«ENDFILE»
«ENDDEFINE»
```

9.8 Appendix H – 2APL rules in detail

9.8.1 APAPL

Old EBNF syntax

```
<Agent_Prog>       =  { "Include:" <ident>
                   |  "BeliefUpdates:" <BelUpSpec>
                   |  "Beliefs:" <belief>
                   |  "Goals:" <goals>
                   |  "Plans:" <plans>
                   |  "PG-rules:" <pgrules>
                   |  "PC-rules:" <pcrules>
                   |  "PR-rules:" <prrules> };
```

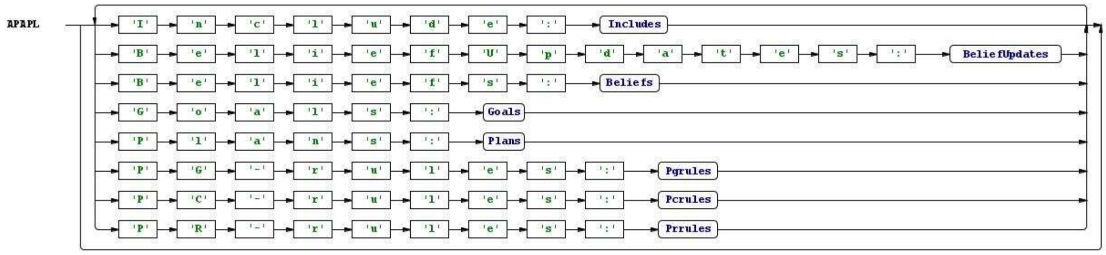
EBNF syntax

```
<APAPL>            =     { "Include:" <includes>
                   |     "BeliefUpdates:" <beliefupdates>
                   |     "Beliefs:" <beliefs>
                   |     "Goals:" <goals>
                   |     "Plans:" <plans>
                   |     "PG-rules:" <pgrules>
                   |     "PC-rules:" <pcrules>
                   |     "PR-rules:" <prrules> };
```

Xtext syntax

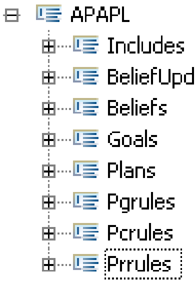
```
APAPL:
      (elements+=
        "Include:" includes+=Includes
      | "BeliefUpdates:" beliefUpdates+=BeliefUpdates
      | "Beliefs:" beliefs+=Beliefs
      | "Goals:" goals+=Goals
      | "Plans:" plans+=Plans
      | "PG-rules:" pgrules+=Pgrules
      | "PC-rules:" pcrules+=Pcrules
      | "PR-rules:" prrules+=Prrules
      )*
      ;
```

ANTLR syntax diagram


APAPL
'I' 'n' 'c' 'l' 'u' 'd' 'e' ':' Includes
'B' 'e' 'l' 'i' 'e' 'f' 'U' 'p' 'd' 'a' 't' 'e' 's' ':' BeliefUpdates
'B' 'e' 'l' 'i' 'e' 'f' 's' ':' Beliefs
'G' 'o' 'a' 'l' 's' ':' Goals
'P' 'l' 'a' 'n' 's' ':' Plans
'P' 'G' '-' 'r' 'u' 'l' 'e' 's' ':' Pgrules
'P' 'C' '-' 'r' 'u' 'l' 'e' 's' ':' Pcrules
'P' 'R' '-' 'r' 'u' 'l' 'e' 's' ':' Prrules

Description

The first rule that should be used is the <APAPL>, this rule can include zero or more elements. Changes from the old version <Agent_Prog> are made to have a concurrent style.



9.8.2 includes

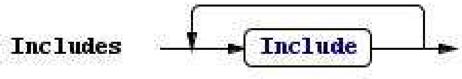
EBNF syntax

```
<includes>  =     <include>+;
```

Xtext syntax

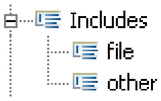
```
Includes:
      (include+=Include)+;
```

ANTLR syntax diagram



Description

If the <includes> rule is called there should be at least one <include> rule present.



9.8.3 include

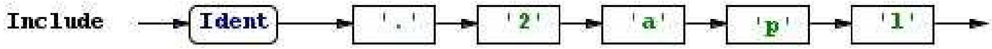
EBNF syntax

```
<include>   =     <ident> ".2apl";
```

Xtext syntax

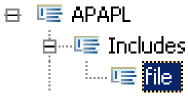
```
Include:
      includeName=Ident ".2apl";
```

ANTLR syntax diagram



Description

An <include> rule starts with an <ident> followed by .2apl. An example could be:



```
Include: file.2apl
```

9.8.4 beliefupdates

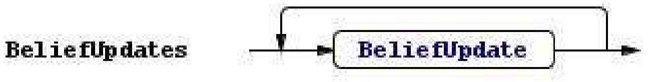
EBNF syntax

```
<beliefupdates>   =     <beliefupdate>+;
```

Xtext syntax

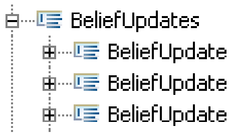
```
BeliefUpdates:
      (beliefUpdate+=BeliefUpdate)+;
```

ANTLR syntax diagram



Description

If the <beliefupdates> rule is called there should be at lease one <beliefupdate> rule present.



```
⊟ BeliefUpdates
   ⊞ BeliefUpdate
   ⊞ BeliefUpdate
   ⊞ BeliefUpdate
```

9.8.5 beliefupdate

Old EBNF syntax

```
<BelUpSpec>       =
      "{"<belquery>"}" <beliefupdate> "{"<literals>"}";

<beliefupdate>    =  <Atom>;
```

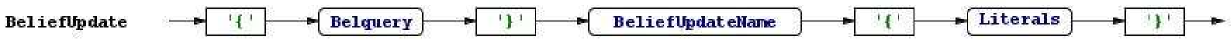
EBNF syntax

```
<beliefupdate>          =
      "{"<belquery>"}" <beliefupdatename> "{"<literals>"}";
```

Xtext syntax

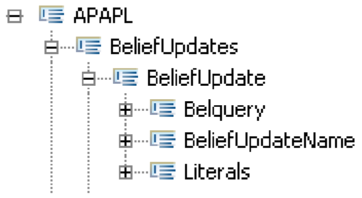
```
BeliefUpdate:
      "{" belquery+=Belquery "}" beliefUpdateName+=BeliefUpdateName
"{" literals+=Literals "}";
```

ANTLR syntax diagram



Description

An <beliefupdate> rule has 3 parts. The first part, between the { and } brackets, is the so called per-condition, the second part is the <beliefupdatename> that holds the name for the beliefupdate and the 3rd part, between the { and } brackets. In the old version of the 2APL syntax this rule was called <BelUpSpec>. An example could be:



```
⊟ APAPL
   ⊟ BeliefUpdates
      ⊟ BeliefUpdate
         ⊞ Belquery
         ⊞ BeliefUpdateName
         ⊞ Literals
```

```
{ not soldObject(N, P, M) }
AddSoldObject(N, P, M)
{ soldObject(N, P, M) }
```

9.8.6 beliefupdatename

EBNF syntax

```
<beliefupdatename>  =     <upperatom>;
```

Xtext syntax

```
BeliefUpdateName:
      upperatom+=UpperAtom;
```

ANTLR syntax diagram

BeliefUpdateName
UpperAtom

Description

A <beliefupdatename> consists of exactly one <upperatom>.

```
AddSoldObject(N, P, M)
```

9.8.7 beliefs

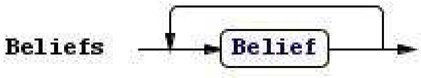
EBNF syntax

```
<beliefs>   =     <belief>+;
```

Xtext syntax

```
Beliefs:
      (belief+=Belief)+;
```

ANTLR syntax diagram



Description

If the <beliefs> rule is called there should be at lease one <belief> rule present. This rule didn't exist in the old 2APL syntax and is added to have a better view if multiple beliefs are implemented.

```
belief1(slow).
belief2(fast) :- walking(Hard).
```

9.8.8 belief

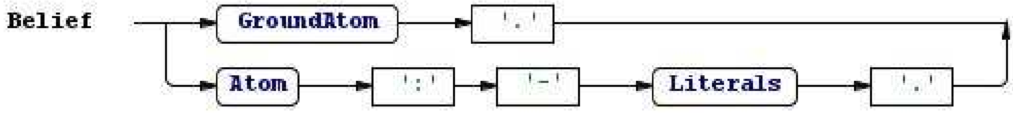
EBNF syntax

```
<belief>    =     <ground_atom>"." | <atom> ":-" <literals> ".";
```

Xtext syntax

```
Belief:
        groundAtom+=GroundAtom "."
      | atom+=Atom ":-" literals+=Literals "."
      ;
```

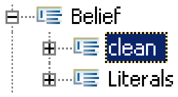
ANTLR syntax diagram



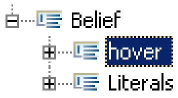
Description

The `<belief>` rule holds a `<groundatom>` followed by a `.` or an `<atom>` followed by `:-` `<literals>`..

```
bomb(3,3).
```



```
clean( blockWorld ) :- not bomb(X,Y) , not
carry(bomb).
```



```
hover( something ) :- X is Y+1, F is X-1.
```

9.8.9 goals

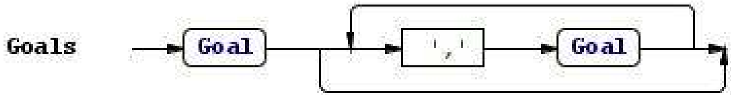
EBNF syntax

```
<goals>     =     <goal> {"," <goal>};
```

Xtext syntax

```
Goals:
      goal+=Goal ("," goal+=Goal)*;
```

ANTLR syntax diagram



Description

If the `<goals>` rule is called there should be at lease one `<goal>` rule present.

```
goal1(slow),
goal2(fast)
```

9.8.10 goal

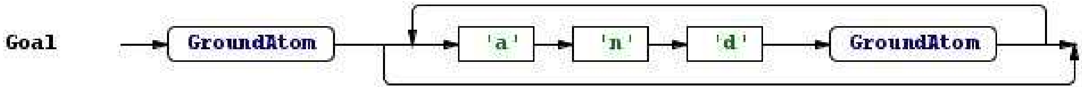
EBNF syntax

```
<goal>      =     <ground_atom> {"and" <ground_atom>};
```

Xtext syntax

```
Goal:
      groundAtom+=GroundAtom ("and" groundAtom+=GroundAtom)*;
```

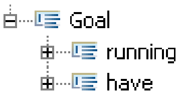
ANTLR syntax diagram



Description

A <goal> starts with <groundatom> and optional followed by multiple times and <groundatom>.

```
buy([car], 7, 10)
```



```
running(fast) and have(drink)
```

9.8.11 baction

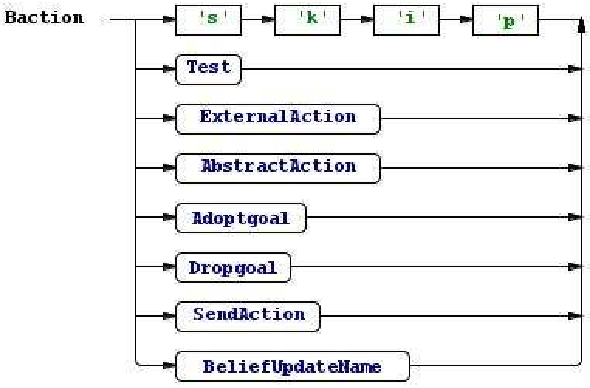
EBNF syntax

```
<baction>   =     "skip"
                  | <beliefupdatename>
                  | <sendaction>
                  | <externalaction>
                  | <abstractaction>
                  | <test>
                  | <adoptgoal>
                  | <dropgoal>
                  ;
```

Xtext syntax

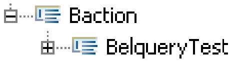
```
Baction:
        "skip"
      | test+=Test
      | externalAction+=ExternalAction
      | abstractAction+=AbstractAction
      | adoptgoal+=Adoptgoal
      | dropgoal+=Dropgoal
      | sendAction+=SendAction
      | beliefUpdateName+=BeliefUpdateName
      ;
```

ANTLR syntax diagram



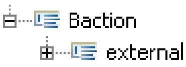
Description

The basic actions are grouped together in <baction> and can have exactly one of the following rules skip, <upperatom>, <sendaction>, <externalaction>, <abstractaction>, <test>, <adoptgoal>, <dropgoal>.



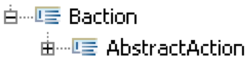
Baction
BelqueryTest

```
B(test)
```



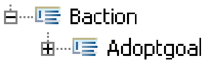
Baction
external

```
@external(action(), L)
```



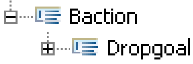
Baction
AbstractAction

```
abstract(Action)
```



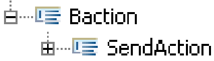
Baction
Adoptgoal

```
adopta(goal)
```



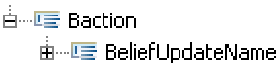
Baction
Dropgoal

```
dropgoal(goal)
```



Baction
SendAction

```
send(a, b, c())
```



Baction
BeliefUpdateName

```
Update(X, Y)
```

9.8.12 plans

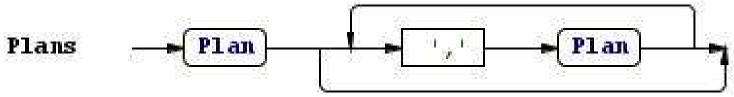
EBNF syntax

```
<plans>     =     <plan> {"," <plan>};
```

Xtext syntax

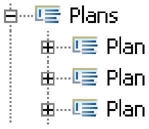
```
Plans:
      plan+=Plan ("," plan+=Plan)*;
```

ANTLR syntax diagram



Description

<plans> start with <plan> and optional followed by multiple times , <plan>.



Plans
Plan
Plan
Plan

```
planA(X, Y),
PlanB(),
@planC(f(), L)
```

9.8.13 plan

Old EBNF syntax

```
<plan>      =     <baction>
                  | <sequenceplan>
                  | <ifplan>
                  | <whileplan>
                  | <atomicplan>
                  ;
```

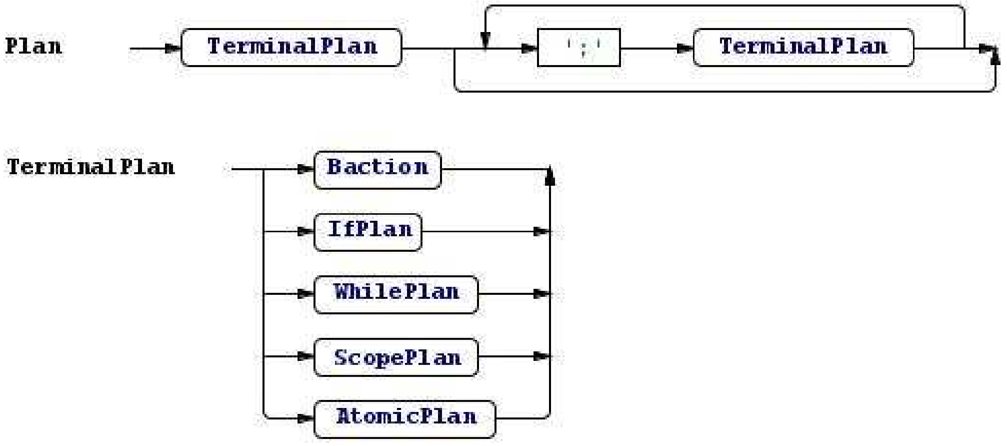
EBNF syntax

```
<plan>      =     <baction>
                  | <sequenceplan>
                  | <ifplan>
                  | <whileplan>
                  | <atomicplan>
                  | <scopeplan>
                  ;
```

Xtext syntax

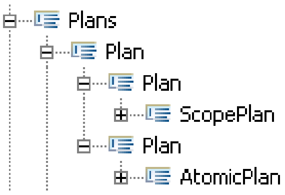
```
Plan:
      TerminalPlan ({Plan.plan+=current} (";"
      plan+=TerminalPlan)+)?;
TerminalPlan returns Plan:
        baction+=Baction
      | ifPlan+=IfPlan
      | whilePlan+=WhilePlan
      | scopePlan+=ScopePlan
      | atomicPlan+=AtomicPlan
      ;
```

ANTLR syntax diagram



Description

The <sequenceplan> has been left out of the Xtext syntax but is implemented as TerminalPlan. This new rule then contains all the other rules that <plan> should have. This change has to be made to make the language left-factored. In the example below a <scopeplan> and a <atomicplan> are both part of the <sequenceplan> because of the ; between the <scopeplan> and <atomicplan>.



```
Plans:
      { scope(yPlan) }
      ;
      [ atomic(xPlan) ]
```

9.8.14 sendaction

Old EBNF syntax

```
<sendaction>       =  "send(" <iv> "," <iv> "," <atom> ")";
                   |  "send(" <iv> "," <iv> "," <iv> "," <iv> ","
<atom> ")";
```

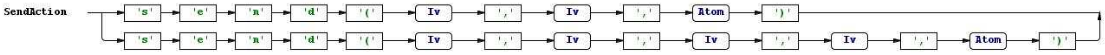
EBNF syntax

```
<sendaction>       =     "send(" <iv> "," <iv> "," <atom> ")"
                   |     "send(" <iv> "," <iv> "," <iv> "," <iv> ","
<atom> ")"
                   ;
```

Xtext syntax

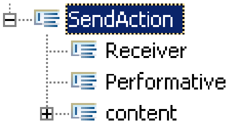
```
SendAction:
        "send(" iv+=Iv "," iv+=Iv "," atom+=Atom ")"
      | "send(" iv+=Iv "," iv+=Iv "," iv+=Iv "," iv+=Iv ","
      atom+=Atom ")"
      ;
```

ANTRL syntax diagram

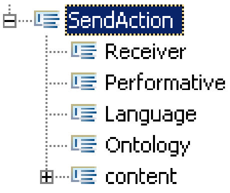


Description

The <sendaction> has two possibilities, one with 2 times <iv> and one with 4 times <iv>. An example is shown below.



```
send(Receiver, Performative, content(X))
```



```
send(Receiver, Performative, Language, Ontology,
content(Y))
```

9.8.15 externalaction

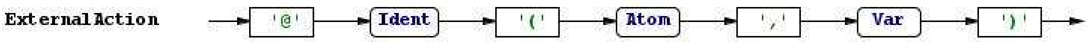
EBNF syntax

```
<externalaction>  =     "@" <ident> "(" <atom> "," <Var> ")";
```

Xtext syntax

```
ExternalAction:
      "@" name=Ident "(" atom+=Atom "," varname=Var ")";
```

ANTRL syntax diagram



Description

The <externalaction> is supposed to change the external environment by sending and receiving messages. This starts with an <ident> that denote the name of the environment, an <atom> to call the method in the external environment and a <var> that stores a return value.

```
@blockworld( enter( Xpos, Ypos), Return)
```

9.8.16 abstractaction

EBNF syntax

```
<abstractaction>  =     <atom>;
```

Xtext syntax

```
AbstractAction:
      atom+=Atom;
```

ANTRL syntax diagram

AbstractAction
Atom

Description

An <abstractaction> contains one <atom>.

```
remove(X)
```

9.8.17 test

Old EBNF syntax

```
<test>              =  "B(" <belquery> ")"
                    |  "G(" <goalquery> ")"
                    |  <test> "&" <test>
                    ;
```

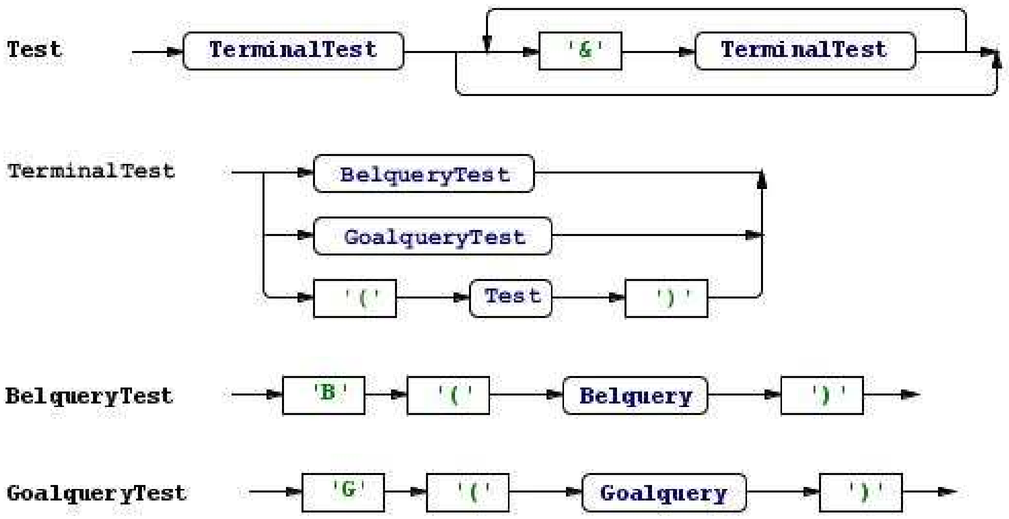
EBNF syntax

```
<test>              =  "B(" <belquery> ")"
                    |  "G(" <goalquery> ")"
                    |  <test> "&" <test>
                    |  "(" <test> ")"
                    ;
```

Xtext syntax

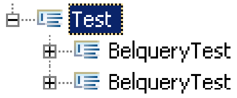
```
Test:
      TerminalTest ({Test.test+=current} ("&"
test+=TerminalTest)+)?;
TerminalTest returns Test:
      BelqueryTest | GoalqueryTest | "(" test+=Test ")";
BelqueryTest:
      "B(" belquery+=Belquery ")";
GoalqueryTest:
      "G(" goalquery+=Goalquery")";
```

ANTRL syntax diagram



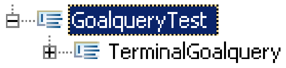
Description

The <test> rule also needed to be left-factored to TerminalTest. Then TerminalTest was devided in BelqueryTest and GoalqueryTest to make sure the outline structure is maintained.



```
Test
    BelqueryTest
    BelqueryTest
```

```
B(me(X,Y)) & B(other(X1,Y1))
```



```
GoalqueryTest
    TerminalGoalquery
```

```
G(walking())
```

9.8.18 adoptgoal

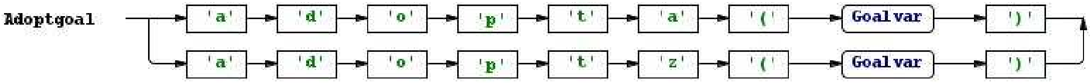
EBNF syntax

```
<adoptgoal> =     "adopta(" <goalvar> ")"
            |     "adoptz(" <goalvar> ")"
            ;
```

Xtext syntax

```
Adoptgoal:
        "adopta(" goalvar+=Goalvar ")"
      | "adoptz(" goalvar+=Goalvar ")"
      ;
```

ANTRL syntax diagram


Adoptgoal
'a' 'd' 'o' 'p' 't' 'a' '(' Goalvar ')'
'a' 'd' 'o' 'p' 't' 'z' '(' Goalvar ')'

Description

<adoptgoal> can implement a goal at the top by adopta or at the end by adoptz.

```
adopta(doCheck())
```

```
adoptz(stop())
```

9.8.19 dropgoal

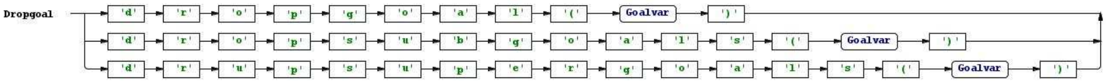
EBNF syntax

```
<dropgoal>  =     "dropgoal(" <goalvar> ")"
            |     "dropsubgoals(" <goalvar> ")"
            |     "dropsupergoals(" <goalvar> ")"
            ;
```

Xtext syntax

```
Dropgoal:
        "dropgoal(" goalvar+=Goalvar ")"
      | "dropsubgoals(" goalvar+=Goalvar ")"
      | "drupsupergoals(" goalvar+=Goalvar ")"
      ;
```

ANTRL syntax diagram


Dropgoal
'd' 'r' 'o' 'p' 'g' 'o' 'a' 'l' '(' Goalvar ')'
'd' 'r' 'o' 'p' 's' 'u' 'b' 'g' 'o' 'a' 'l' 's' '(' Goalvar ')'
'd' 'r' 'u' 'p' 's' 'u' 'p' 'e' 'r' 'g' 'o' 'a' 'l' 's' '(' Goalvar ')'

Description

<dropgoal> can drop a goals from an agent’s goal base.

```
dropgoal(check())
```

```
dropsubgoals(goalX(F))
```

9.8.20 ifplan

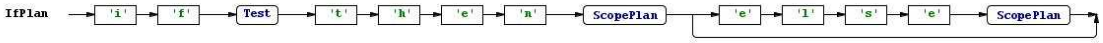
EBNF syntax

```
<ifplan>    =
      "if" <test> "then" <scopeplan> ["else" <scopeplan>];
```

Xtext syntax

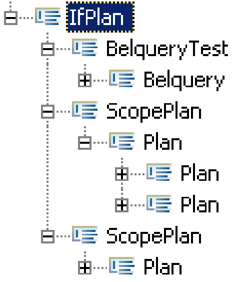
```
IfPlan:
      "if" test+=Test "then" scopePlan+=ScopePlan ("else"
      scopePlan+=ScopePlan)?;
```

ANTRL syntax diagram



Description

The <ifplan> is formatted as if <test> then <scopeplan> and optionally the form else <scopeplan>. The example shows that it is also possible to have a <sequenceplan> within the <scopeplan>.



IfPlan
- BelqueryTest
  - Belquery
- ScopePlan
  - Plan
    - Plan
    - Plan
- ScopePlan
  - Plan

```
if B(have(Car)) then {
      sell(Car); collect(Money)
}
else {
      getGas(UnleadedFuel)
}
```

9.8.21 whileplan

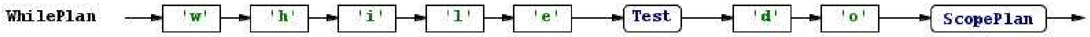
EBNF syntax

```
<whileplan> =     "while" <test> "do" <scopeplan>;
```

Xtext syntax

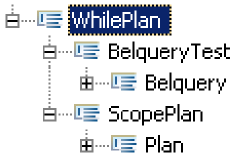
```
WhilePlan:
      "while" test+=Test "do" scopePlan+=ScopePlan;
```

ANTRL syntax diagram



Description

<whileplan> will repeat the <scopeplan> until the <test> returns false.



WhilePlan
- BelqueryTest
  - Belquery
- ScopePlan
  - Plan

```
while B(HaveMoney < 1000) do {
      work
}
```

9.8.22 atomicplan

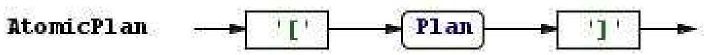
EBNF syntax

```
<atomicplan>      =     "[" <plan> "]";
```

Xtext syntax

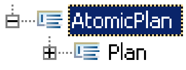
```
AtomicPlan:
      "[" plan+=Plan "]";
```

ANTRL syntax diagram



Description

An <atomicplan> surrounds the <plan> with a [and] bracket.



AtomicPlan
Plan

```
[ plan(Y) ]
```

9.8.23 scopeplan

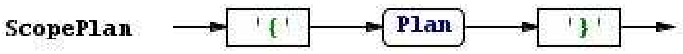
EBNF syntax

```
<scopeplan> =     "{" <plan> "}";
```

Xtext syntax

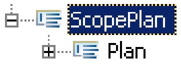
```
ScopePlan:
       "{" plan+=Plan "}";
```

ANTRL syntax diagram



Description

<scopeplan> surrounds the <plan> with a { and } bracket.



ScopePlan
Plan

```
{ plan(Y) }
```

9.8.24 pgrules

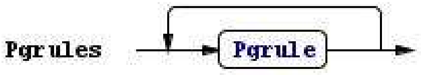
EBNF syntax

```
<pgrules>   =     <pgrule>+;
```

Xtext syntax

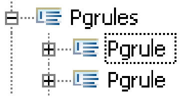
```
Pgrules:
       (pgrule+=Pgrule)+;
```

ANTRL syntax diagram



Description

If <pgrules> is called, at least one <pgrule> should be present.



Pgrules
Pgrule
Pgrule

```
PG-rules:
<- true | skip
true <- true | skip
```

9.8.25 pgrule

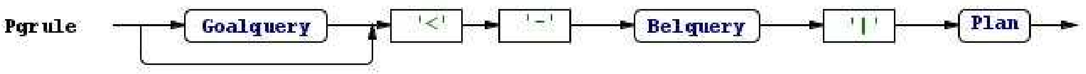
EBNF syntax

```
<pgrule>    =     [<goalquery>] "<-" <belquery> "|" <plan>;
```

Xtext syntax

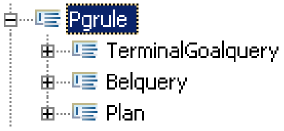
```
Pgrule:
      (goalquery+=Goalquery)? "<-" belquery+=Belquery "|"
      plan+=Plan;
```

ANTRL syntax diagram



Description

If the <pgrule> is called, it starts optional with a <goalquery> followed by <- and a <belquery> ending with | <plan>.



```
PG-rules:
      makeMoney(Lots) <- have(Car) | Sell(Car)
```

9.8.26 pcrules

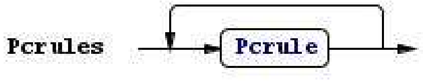
EBNF syntax

```
<pcrules>   =     <pcrule>+;
```

Xtext syntax

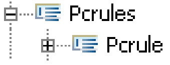
```
Pcrules:
      (pcrule+=Pcrule)+;
```

ANTRL syntax diagram



Description

If <pcrules> is called at least one <pcrule> should be present.



```
PC-rules:
atom <- belquery | plan
```

9.8.27 pcrule

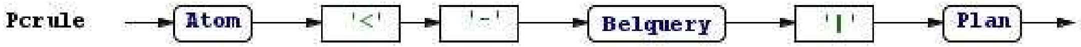
EBNF syntax

```
<pcrule>    =     <atom> "<-" <belquery> "|" <plan>;
```

Xtext syntax

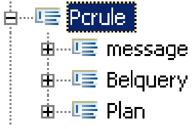
```
Pcrule:
      atom+=Atom "<-" belquery+=Belquery "|" plan+=Plan;
```

ANTRL syntax diagram



Description

A <pcrule> starts with an <atom> followed by <- <belquery> and ends with | <plan>.



```
message(From, Bank, received(X)) <- have(Account) |
{
        SaveInAccount(X)
}
```

9.8.28 prrules

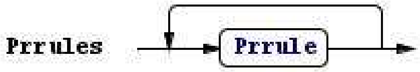
EBNF syntax

```
<prrules>   =     <prrule>+;
```

Xtext syntax

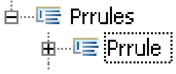
```
Prrules:
      (prrule+=Prrule)+;
```

ANTRL syntax diagram



Description

If <prrules> is called at least one <prrule> should be present.



```
PR-rules:
Planvar <- true | Planvar
```

9.8.29 prrule

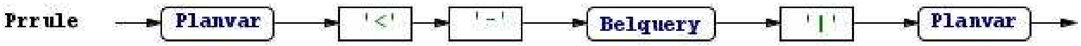
EBNF syntax

```
<prrule>    =     <planvar> "<-" <belquery> "|" <planvar>;
```

Xtext syntax

```
Prrule:
      planvar+=Planvar "<-" belquery+=Belquery "|" planvar+=Planvar;
```

ANTRL syntax diagram



Description

A <prrule> starts with a <planvar> followed by <- <belquery> and ends with | <planvar>.

```
@externalworld( getCar(Toyota), R ) <- not haveCar()
| {
        UpdateCar(R)
}
```

9.8.30 goalvar

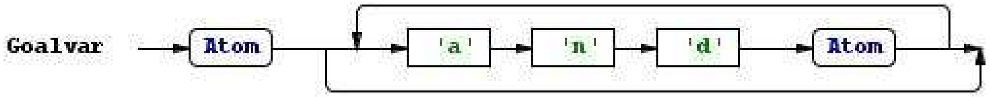
EBNF syntax

```
<goalvar>   =     <atom> {"and" <atom>};
```

Xtext syntax

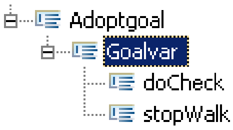
```
Goalvar:
      atom+=Atom ("and" atom+=Atom)*;
```

ANTRL syntax diagram



Description

<goalvar> contains at least one <atom> and can optionally multiple times be followed by and <atom>.


Adoptgoal
Goalvar
doCheck
stopWalk

```
adopta(doCheck() and stopWalk())
```

9.8.31 planvar

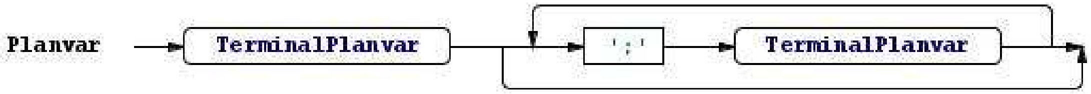
EBNF syntax

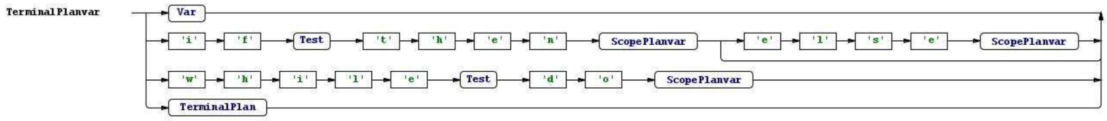
```
<planvar>   =     <plan>
            |     <Var>
            |     "if" <test> "then" <scopeplanvar> ["else"
                  <scopelanvar>]
            |     "while" <test> "do" <scopeplanvar>
            |     <planvar> ";" <planvar>
            ;
```

Xtext syntax

```
Planvar:
      TerminalPlanvar ({Planvar.planvar+=current} (";"
      planvar+=TerminalPlanvar)+)?;
TerminalPlanvar returns Planvar:
        name=Var
      | name="if" test+=Test "then" scopePlanvar+=ScopePlanvar
      ("else" scopePlanvar+=ScopePlanvar)?
      | name="while" test+=Test "do" scopePlanvar+=ScopePlanvar
      | plan+=TerminalPlan
      ;
```

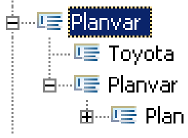
ANTRL syntax diagram





Description

<planvar> also needed to be left-factored, so TerminalPlanvar was created. In TerminalPlanvar it is possible to put a <var> a differend version of <ifplan> and <whileplan> and TerminalPlan. Note that TerminalPlan is differend than TerminalPlanvar.



```
Toyota ; B(Car = Toyota )
```

9.8.32 scopeplanvar

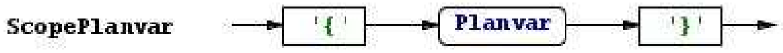
EBNF syntax

```
<scopeplanvar>    =     "{" <planvar> "}";
```

Xtext syntax

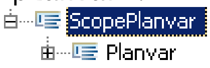
```
ScopePlanvar:
      "{" planvar+=Planvar "}";
```

ANTRL syntax diagram



Description

<scopeplanvar> is almost the same as <scopeplan> only between the { and } brackets now exists <planvar>.



```
{ Toyota ; B(Car = Toyota ) }
```

9.8.33 literals

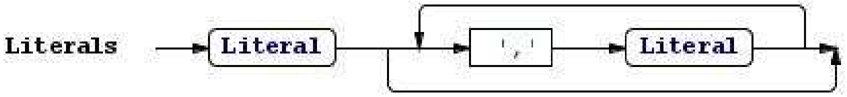
EBNF syntax

```
<literals>  =     <literal> {"," <literal>};
```

Xtext syntax

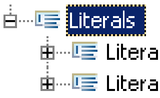
```
Literals:
      literal+=Literal ("," literal+=Literal)*;
```

ANTRL syntax diagram



Description

<literals> consist of <literal> optional followed by multiple times , <literal>.


Literals
Literal
Literal

```
not have(Car) , have(Money)
```

9.8.34 literal

Old EBNF syntax

```
<literal>    =     <atom>
             |     "not" <atom>
             ;
```

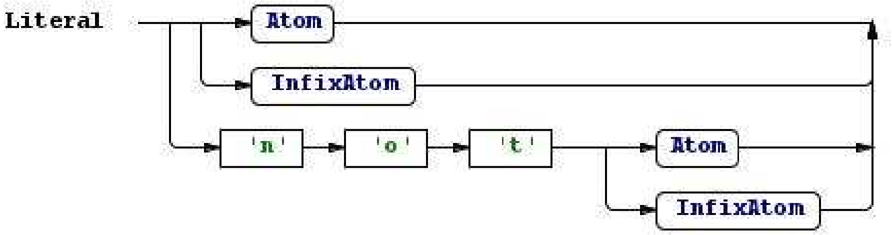
EBNF syntax

```
<literal>    =     (<atom> | <infixatom>)
             |     "not" (<atom> | <infixatom>)
             ;
```

Xtext syntax

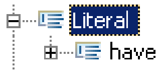
```
Literal:
        atom+=Atom
      | infixatom+=InfixAtom
      | "not" atom+=Atom
      | "not" infixatom+=InfixAtom
      ;
```

ANTRL syntax diagram

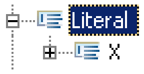


Description

A <literal> has 4 posible options: <atom> or <infixatom> or not <atom> or not <infixatom>.


Literal
have

```
not have(Car)
```


Literal
X

```
X is Y+1
```

9.8.35 belquery

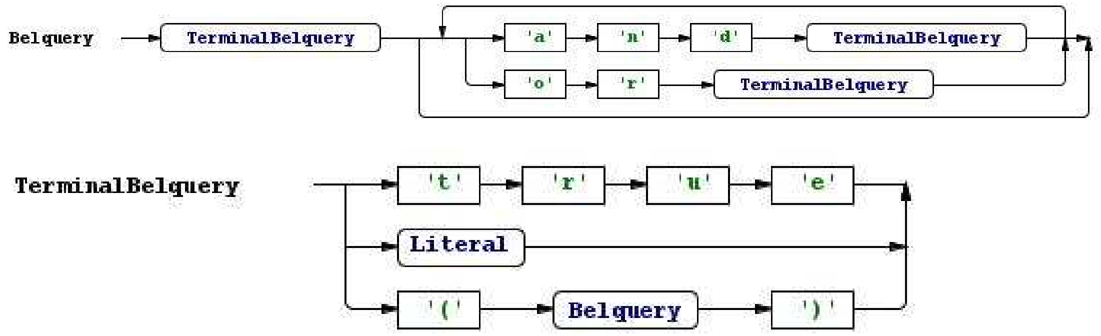
EBNF syntax

```
<belquery>   =     "true"
             |     <belquery> "and" <belquery>
             |     <belquery> "or" <belquery>
             |     "(" <belquery ")"
             |     <literal>
             ;
```

Xtext syntax

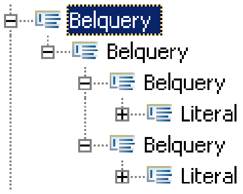
```
Belquery:
      TerminalBelquery ({Belquery.belquery+=current} (("and" | "or")
      belquery+=TerminalBelquery)+)?;
TerminalBelquery returns Belquery:
        "true"
       | literal+=Literal
       | '(' belquery+=Belquery ')'
       ;
```

ANTRL syntax diagram



Description

To remove the left-recursion `TerminalBelquery` was created. `TerminalBelquery` can have true `<literal>` or an `<belquery>` encapsulated by (and).



```
Belquery
  Belquery
    Belquery
      Literal
    Belquery
      Literal
```

```
(have(Account) and status(active))
```

9.8.36 goalquery

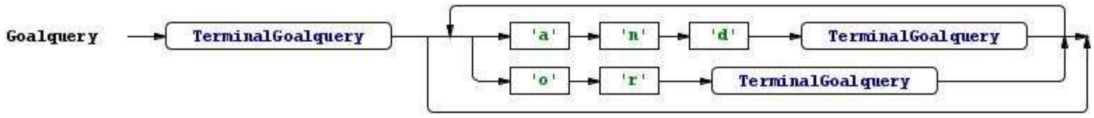
EBNF syntax

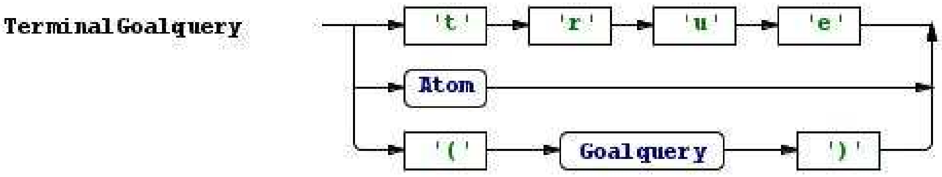
```
<goalquery> =      "true"
              |    <goalquery> "and" <goalquery>
              |    <goalquery> "or" <goalquery>
              |    "(" <goalquery> ")"
              |    <atom>
              ;
```

Xtext syntax

```
Goalquery:
      TerminalGoalquery ({Goalquery.goalquery+=current} (("and" |
      "or") goalquery+=TerminalGoalquery)+)?;
TerminalGoalquery:
        "true"
       | atom+=Atom
       | '(' goalQuery+=Goalquery ')'
       ;
```

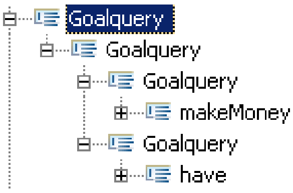
ANTRL syntax diagram

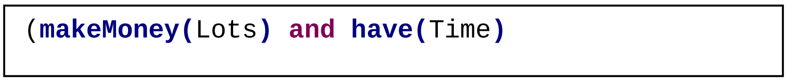




Description

The only difference between `<goalquery>` and `<belquery>` is the `<atom>` and `<literal>`. To remove the left-recursion `TerminalGoalquery` was created.





```
(makeMoney(Lots) and have(Time)
```

9.8.37 iv

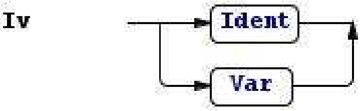
EBNF syntax

```
<iv>  =     <ident> | <Var>;
```

Xtext syntax

```
Iv:
      name=Ident | name=Var;
```

ANTRL syntax diagram



Description

An `<iv>` can be an `<ident>` or a `<var>` this rule is only used in the `<sendaction>`.

9.8.38 groundatom

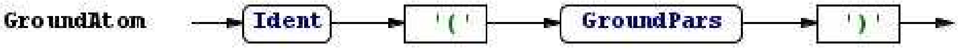
EBNF syntax

```
<groundatom>      =     <ident> "(" <groundpars> ")";
```

Xtext syntax

```
GroundAtom:
      name=Ident "(" groundpars+=GroundPars ")";
```

ANTRL syntax diagram



Description

A <groundatom> is a atom that must have <groundpars> surrounded by (and).

start
GroundPars

```
start(0, xPos, yPos)
```

9.8.39 groundpars

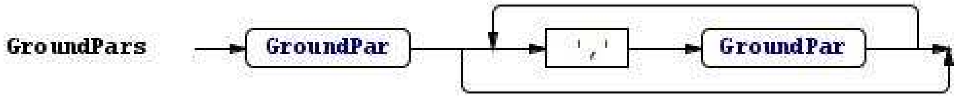
EBNF syntax

```
<groundpars>      =     <groundpar> {"," <groundpar>};
```

Xtext syntax

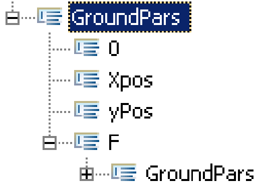
```
GroundPars:
      groundpar+=GroundPar ("," groundpar+=GroundPar)*;
```

ANTRL syntax diagram



Description

<groundpars> consist of <groundpar> optionaly followed by one or multiple times , <groundpar>.



GroundPars
0
Xpos
yPos
F
GroundPars

```
start(0, xPos, yPos, [A, _ | F]).
```

9.8.40 groundpar

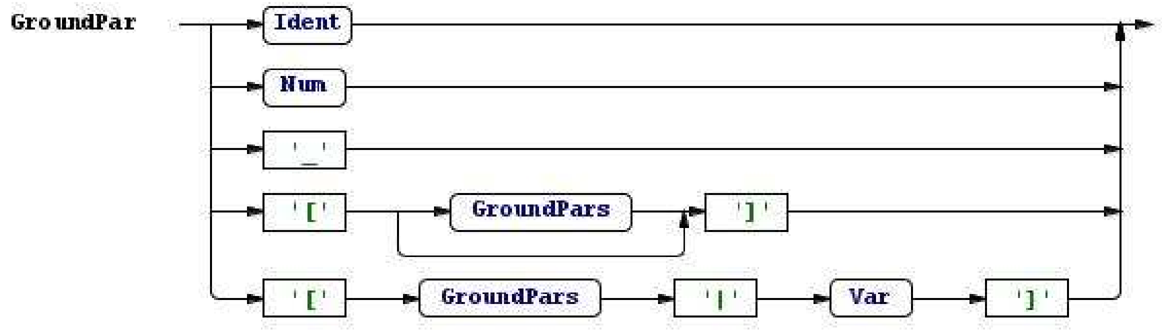
EBNF syntax

```
<groundpar>          =  <ident> | <num> | "_"
                     |  "[" [<groundpars>] "]"
                     |  "[" <groundpars> "|" <var> "]"
                     ;
```

Xtext syntax

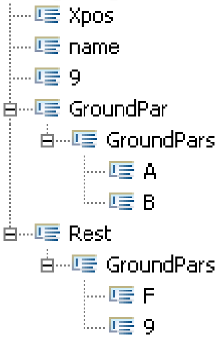
```
GroundPar:
        name=Ident
      | name=Num
      | "_"
      | "[" (groundpars+=GroundPars)? "]"
      | "[" groundpars+=GroundPars "|" name=Var "]"
      ;
```

ANTRL syntax diagram



Description

<groundpar> can contain an <ident> a <num> or a _. Also optional <groundpars> surrounded by [and] or [<groundpars> | <var>].



```
start(
      name,
      9,
      _,
      [A, B],
      [F, 9 | Rest]
)
```

9.8.41 upperatom

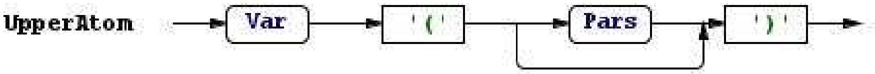
EBNF syntax

```
<upperatom> =     <var> "(" [<pars>] ")";
```

Xtext syntax

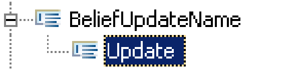
```
UpperAtom:
      name=Var '(' (pars+=Pars)? ')';
```

ANTRL syntax diagram

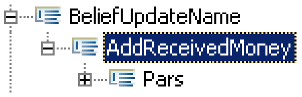


Description

An <upperatom> is used to denote a <beliefupdatename> and starts with a <var> followed by the (and) bracket with optional <pars> between the brackets.



```
Update()
```



```
AddReceivedMoney(N, P, M)
```

9.8.42 atom

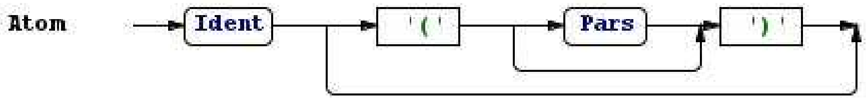
EBNF syntax

```
<atom>      =     <ident> ["(" [<pars>] ")"];
```

Xtext syntax

```
Atom:
      name=Ident ("(" (pars+=Pars)? ")")?;
```

ANTRL syntax diagram



Description

An <atom> starts with <ident> and optional (and) brackets with optional <pars> between the brackets.

```
atom
```

```
running()
```

```
received(Money)
```

9.8.43 infixatom

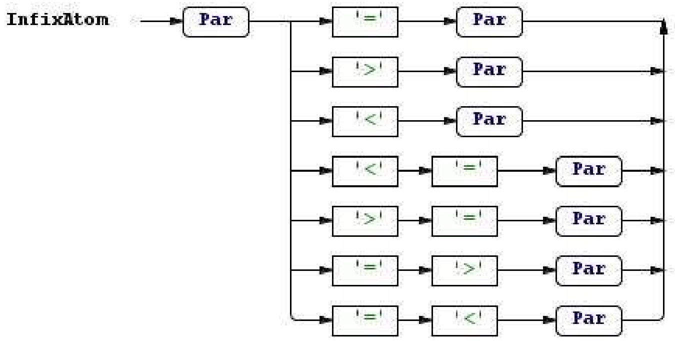
EBNF syntax

```
<infixatom>      =     <par> ("=" | ">" | "<" | "=" | "<=" | ">=" | "=>" |
"=<") <par>;
```

Xtext syntax

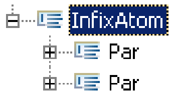
```
InfixAtom:
      par+=Par ("=" | ">" | "<" | "<=" | ">=" | "=>" | "=<")
par+=Par
      ;
```

ANTRL syntax diagram



Description

An <infixatom> is a logic atom starting with <par> followed by an expression and ending with <par>.


InfixAtom
Par
Par

```
X = Y+1
```

```
F-X > X+3
```

9.8.44 pars

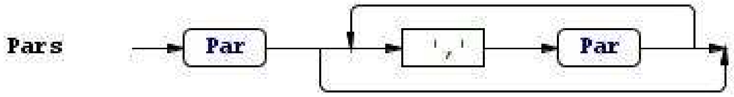
EBNF syntax

```
<pars>      =     <par> {"," <par>};
```

Xtext syntax

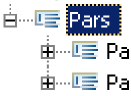
```
Pars:
      par+=Par ("," par+=Par)*;
```

ANTRL syntax diagram



Description

<pars> consists of <par> optionally followed by one or multiple times , <par>.


Pars
Par
Par

```
x,[f]
```

9.8.45 par

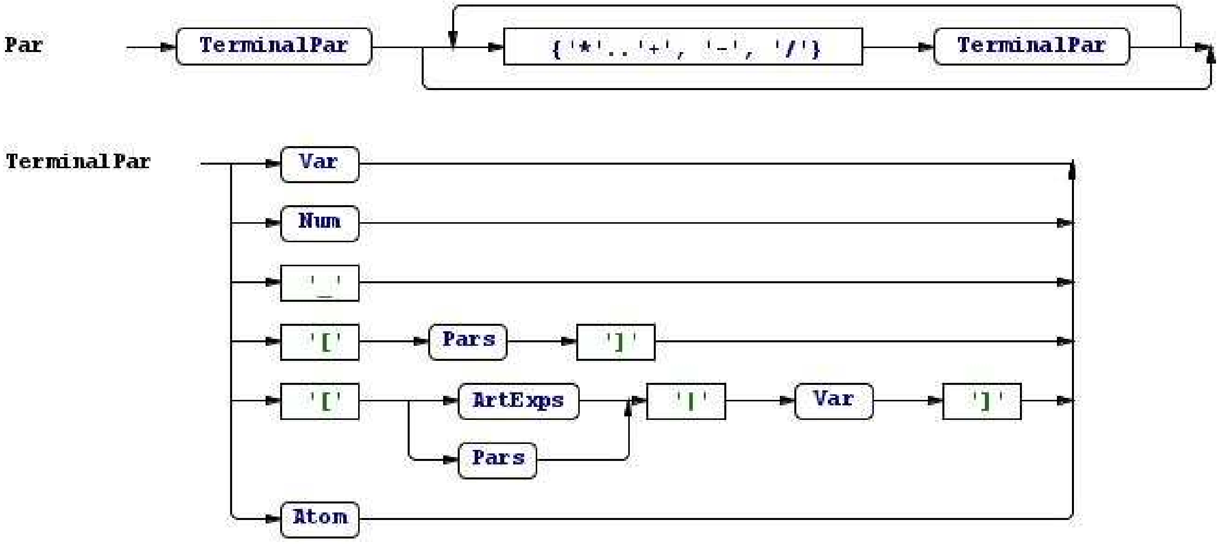
EBNF syntax

```
<par> =     <var> | <num> | "_" | <atom>
      |     "[" <pars> "]"
      |     "[" (<artexps> | <pars>) "|" <var> "]"
      ;
```

Xtext syntax

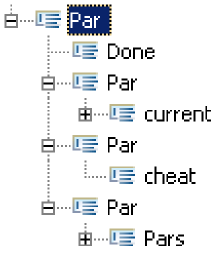
```
Par:
      TerminalPar ({Par.par+=current} (("+" | "-" | "*" | "/")
par+=TerminalPar)+)?;
TerminalPar returns Par:
        name=Var
      | name=Num
      | name='_'
      | '[' pars+=Pars ']'
      | '[' (artexps+=ArtExps | pars+=Pars) '|' name=Var ']'
      | atom+=Atom
      ;
```

ANTRL syntax diagram



Description

To remove the left-recusion, `TerminalPar` was created. `<par>` starts with `TerminalPar` and optional an operation followed by `TerminalPar`. `TerminalPar` can consist of a `<var>` a `<num>` a _ or an `<atom>`. `TerminalPar` can also have [and] brackets with `<pars>` or `<artexps>` inside.



```
Par
  Done
  Par
    current
  Par
    cheat
  Par
    Pars
```

```
Done + current(X) - cheat * [X]
```

9.8.46 artexps

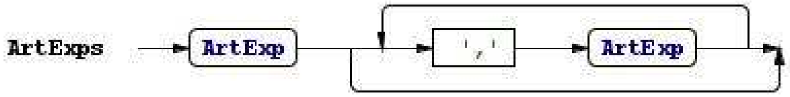
EBNF syntax

```
<artexps>   =    <artexp> {"," <artexp>};
```

Xtext syntax

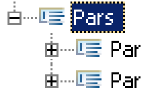
```
ArtExps:
      artexp+=ArtExp (',' artexp+=ArtExp)*;
```

ANTRL syntax diagram



Description

<artexps> consists of <artexp> optionally followed by one or multiple times , <artexp>.



```
X+10 , Y-x
```

9.8.47 artexp

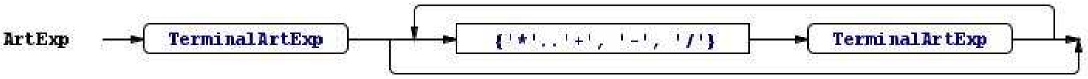
EBNF syntax

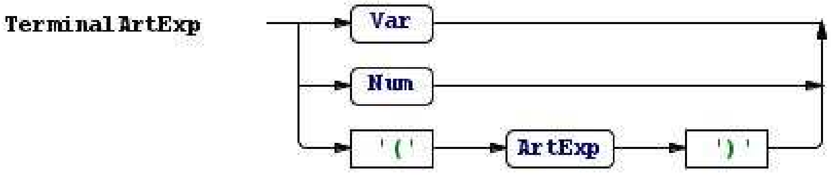
```
<artexp>       =        <var> | <num>
               |        <artexp> ("+" | "-" | "*" | "/") <artexp>
               |        "(" <artexp> ")"
               ;
```

Xtext syntax

```
ArtExp:
      TerminalArtExp ({ArtExp.artexp+=current} (("+" | "-" | "*" |
      "/") artexp+=TerminalArtExp)+)?;
TerminalArtExp returns ArtExp:
        name=Var
      | name=Num
      | '(' artexp+=ArtExp ')'
      ;
```

ANTRL syntax diagram





Description

<artexp> starts with TerminalArtExp that can consist of a <var> a <num> or an <artexp> surrounded by (and) brackets. <artexp> can also consist of TerminalArtExp followed one or multipe times by an operation and TerminalArtExp.

9.8.48 var

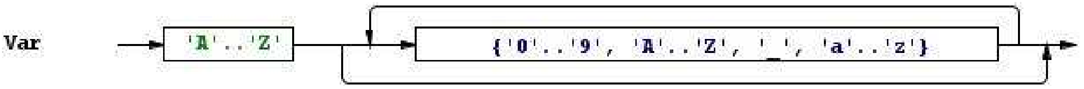
EBNF syntax

```
<var>       =     "A".."Z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
```

Xtext syntax

```
terminal Var:
      ('A'..'Z') ('a'..'z' | 'A'..'Z' | '0'..'9' | "_")*;
```

ANTRL syntax diagram



Description

A <var> denotes a string starting with a capital letter.

Seller

```
Seller
```

9.8.49 ident

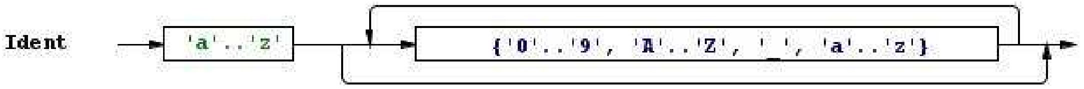
EBNF syntax

```
<ident>     =     "a".."z" {"a".."z" | "A".."Z" | "0".."9" | "_"};
```

Xtext syntax

```
terminal Ident:
      ('a'..'z') ('a'..'z' | 'A'..'Z' | '0'..'9' | "_")*;
```

ANTRL syntax diagram



Description

An <ident> denotes a string starting with a lowercase letter.

seller

```
seller
```

9.8.50 num

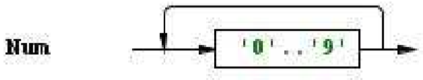
EBNF syntax

```
<num>       =     ("0".."9")+;
```

Xtext syntax

```
terminal Num:
      ('0'..'9')+;
```

ANTRL syntax diagram



Description

A <num> denotes a number value.

900

```
900
```

9.9 Appendix I – Run configuration

APAPL.java

```
if (args.length > 0) {
    if (!args[args.length - 1].startsWith("-")) {
        // Does the file exist?
        masfile = new File(args[args.length - 1]);
        if (!masfile.isFile()) {
            // Try to find the mas file in the directory
            if (masfile.isDirectory()) {
                File[] listOfFiles = masfile.listFiles();
                for (int i = 0; i < listOfFiles.length; i++) {
                    if (listOfFiles[i].isFile() &&
listOfFiles[i].getName().endsWith(".mas")) {
                        System.out.print("Found mas file " +
listOfFiles[i].getName() + " in directory " + args[args.length - 1] + "\n");
                        masfile = new File(args[args.length - 1] +
File.separator + listOfFiles[i].getName());
                        break;
                    }
                }
                // Check again if a mas file is found and loaded
                if (!masfile.isFile()) {
                    System.out.print("Cannot access MAS file: " + masfile +
"\n");

                    System.exit(0);
                }
            } else {
                System.out.print("Cannot access MAS file: " + masfile + "\n");
                System.exit(0);
            }
        }
    }
}
```

9.10 Appendix J – AbstractAPAPLRuntimeModule.java

```java
/*
 * generated by Xtext
 */
package nl.uu.cs.apapl.ide;

import org.eclipse.xtext.Constants;
import org.eclipse.xtext.service.DefaultRuntimeModule;

import com.google.inject.Binder;
import com.google.inject.name.Names;

/**
 * Manual modifications go to {nl.uu.cs.apapl.ide.APAPLRuntimeModule}
 */
public abstract class AbstractAPAPLRuntimeModule extends DefaultRuntimeModule {

      @Override
      public void configure(Binder binder) {
             super.configure(binder);

      binder.bind(String.class).annotatedWith(Names.named(Constants.LANGUAGE_NAME)).toIns
tance(
                    "nl.uu.cs.apapl.ide.APAPL");
      }

      // contributed by org.eclipse.xtext.generator.grammarAccess.GrammarAccessFragment
      public Class<? extends org.eclipse.xtext.IGrammarAccess> bindIGrammarAccess() {
             return nl.uu.cs.apapl.ide.services.APAPLGrammarAccess.class;
      }

      // contributed by
org.eclipse.xtext.generator.parseTreeConstructor.ParseTreeConstructorFragment
      public Class<? extends org.eclipse.xtext.parsetree.reconstr.IParseTreeConstructor>
bindIParseTreeConstructor() {
             return
nl.uu.cs.apapl.ide.parseTreeConstruction.APAPLParsetreeConstructor.class;
      }

      // contributed by de.itemis.xtext.antlr.XtextAntlrGeneratorFragment
      public Class<? extends org.eclipse.xtext.parser.antlr.IAntlrParser>
bindIAntlrParser() {
             return nl.uu.cs.apapl.ide.parser.antlr.APAPLParser.class;
      }

      // contributed by de.itemis.xtext.antlr.XtextAntlrGeneratorFragment
      public Class<? extends org.eclipse.xtext.parser.ITokenToStringConverter>
bindITokenToStringConverter() {
             return org.eclipse.xtext.parser.antlr.AntlrTokenToStringConverter.class;
      }

      // contributed by de.itemis.xtext.antlr.XtextAntlrGeneratorFragment
      public Class<? extends org.eclipse.xtext.parser.antlr.IAntlrTokenFileProvider>
bindIAntlrTokenFileProvider() {
             return nl.uu.cs.apapl.ide.parser.antlr.APAPLAntlrTokenFileProvider.class;
      }

      // contributed by de.itemis.xtext.antlr.XtextAntlrGeneratorFragment
      public Class<? extends org.eclipse.xtext.parser.antlr.Lexer> bindLexer() {
             return nl.uu.cs.apapl.ide.parser.antlr.internal.InternalAPAPLLexer.class;
      }
	// contributed by de.itemis.xtext.antlr.XtextAntlrGeneratorFragment
	public Class<? extends org.eclipse.xtext.parser.antlr.ITokenDefProvider> 
bindITokenDefProvider() {
		return org.eclipse.xtext.parser.antlr.AntlrTokenDefProvider.class;
	}

	// contributed by org.eclipse.xtext.generator.validation.JavaValidatorFragment
	@org.eclipse.xtext.service.SingletonBinding(eager=true)	public Class<? extends 
nl.uu.cs.apapl.ide.validation.APAPLJavaValidator> bindAPAPLJavaValidator() {
		return nl.uu.cs.apapl.ide.validation.APAPLJavaValidator.class;
	}

	// contributed by org.eclipse.xtext.generator.scoping.JavaScopingFragment
	public Class<? extends org.eclipse.xtext.scoping.IScopeProvider> 
bindIScopeProvider() {
		return nl.uu.cs.apapl.ide.scoping.APAPLScopeProvider.class;
	}

	// contributed by org.eclipse.xtext.generator.formatting.FormatterFragment
	public Class<? extends org.eclipse.xtext.formatting.IFormatter> bindIFormatter() {
		return nl.uu.cs.apapl.ide.formatting.APAPLFormatter.class;
	}

}
```