

Extending and Continuing the Development of an Integrated  
Development Environment for Answer Set Programming

Robert Ibbotson

Bachelor of Science in Computer Science with Honours  
The University of Bath  
May 2010

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed:

# **Extending and Continuing the Development of an Integrated Development Environment for Answer Set Programming**

Submitted by: Robert Ibbotson

## **COPYRIGHT**

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see <http://www.bath.ac.uk/ordinances/#intelprop>).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

## **Declaration**

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

## **Abstract**

Work has previously been done to support the logic programming paradigm of Answer Set Programming (ASP) in the form of an Integrated Development Environment (IDE) called AnsProlog\* Programming Environment (APE). APE is a Eclipse plug-in that provides support through features such as toggle commenting, dependency graphs and the ability to launch programs such as LPARSE and SMOBELS. This dissertation extends upon APE, investigating further the tools used by the ASP community and the possible support an IDE can provide. Improvements are made to APE, as well as providing a scalable maintainable architecture for future developers of APE.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Literature Survey</b>	<b>4</b>
2.1	Introduction . . . . .	4
2.2	Logic Programming . . . . .	4
2.2.1	Parts of a Program . . . . .	5
2.3	Answer Set Programming . . . . .	6
2.3.1	AnsProlog* vs Prolog . . . . .	8
2.3.2	AnsProlog* Usage . . . . .	9
2.4	IDEs . . . . .	9
2.4.1	Eclipse Plug-in . . . . .	10
2.4.2	APE . . . . .	10
2.4.3	Other IDEs . . . . .	11
2.5	ASP Tools . . . . .	11
2.5.1	DLV . . . . .	12
2.5.2	SMODELS . . . . .	12
2.5.3	LPARSE . . . . .	12
2.5.4	Cmodels . . . . .	13
2.5.5	Sup . . . . .	13
2.5.6	Gringo . . . . .	13
2.5.7	clasp . . . . .	14
2.5.8	ASPViz . . . . .	14
2.5.9	Debugging . . . . .	14

2.6	Common Language . . . . .	16
2.7	Regression Testing . . . . .	16
2.8	Usability Evaluation . . . . .	17
2.9	Summary . . . . .	18
<b>3</b>	<b>Requirements</b>	<b>19</b>
3.1	Gathering Process . . . . .	20
3.2	Discussion of Results . . . . .	21
3.2.1	Experience . . . . .	21
3.2.2	Current Editor . . . . .	21
3.2.3	APE . . . . .	23
3.2.4	ASP Tools . . . . .	24
3.2.5	Features . . . . .	26
3.3	Summary . . . . .	27
<b>4</b>	<b>Design</b>	<b>29</b>
4.1	Architecture . . . . .	29
4.1.1	Eclipse Plug-ins . . . . .	29
4.1.2	APE . . . . .	31
4.2	Developer Framework . . . . .	33
4.3	Conversion . . . . .	34
4.4	Predicate Completion . . . . .	34
4.5	Testing . . . . .	35
4.5.1	APE Tests . . . . .	36
<b>5</b>	<b>Detailed Design and Implementation</b>	<b>38</b>
5.1	Approach . . . . .	38
5.2	Improvements . . . . .	39
5.3	Installation . . . . .	40
5.4	Framework . . . . .	40
5.4.1	Launch Configuration Tab Group . . . . .	41
5.4.2	Launch Configuration Type . . . . .	45

5.4.3	Launch Shortcuts . . . . .	46
5.5	Gringo and Clasp . . . . .	46
5.5.1	Gringo Editor Problems . . . . .	46
5.6	Conversion . . . . .	48
5.6.1	Process . . . . .	48
5.6.2	Gringo to Lparse . . . . .	49
5.6.3	Lparse to Gringo . . . . .	51
5.7	Auto Predicate Completion . . . . .	52
5.7.1	Predicate View . . . . .	53
5.8	Summary . . . . .	55
<b>6</b>	<b>Testing</b>	<b>57</b>
6.1	Test Environment . . . . .	57
6.1.1	Automatic Tests . . . . .	57
6.1.2	Manual Tests . . . . .	58
6.2	Location . . . . .	58
6.3	Automatic Testing . . . . .	60
6.3.1	Project Tests . . . . .	60
6.3.2	External Program Testing . . . . .	60
6.3.3	Conversion Between Solvers . . . . .	62
6.3.4	Predicate View . . . . .	63
6.4	Manual Testing . . . . .	63
6.4.1	Preference Dialog . . . . .	63
6.4.2	Platform Compatibility & Installation . . . . .	64
6.4.3	Predicate View & Predicate Auto Completion . . . . .	65
6.5	Requirements . . . . .	67
6.5.1	Framework To Add New Features . . . . .	67
6.5.2	Support for Gringo and Clasp . . . . .	67
6.5.3	Auto Predicate Completion . . . . .	67
6.5.4	Conversion . . . . .	68
6.5.5	Regression Testing Ability . . . . .	68

6.5.6	User Friendly . . . . .	68
<b>7</b>	<b>Conclusions</b>	<b>69</b>
7.1	Project Summary . . . . .	69
7.2	Limitations . . . . .	71
7.3	Further Work . . . . .	71
7.4	Summary . . . . .	73
	<b>Bibliography</b>	<b>78</b>
<b>A</b>	<b>Questionnaire</b>	<b>79</b>
<b>B</b>	<b>Questionnaire Results</b>	<b>81</b>
<b>C</b>	<b>Requirements Specification</b>	<b>85</b>
C.1	Functional . . . . .	85
C.2	Non-functional . . . . .	86
<b>D</b>	<b>Installing APE</b>	<b>87</b>
D.1	Update Site . . . . .	87
D.2	Supplied CD or Website . . . . .	87
<b>E</b>	<b>Implementation Screenshots</b>	<b>88</b>
<b>F</b>	<b>Testing</b>	<b>91</b>
F.1	Running The Tests . . . . .	91
F.1.1	Screencast . . . . .	91
F.2	New Feature Test . . . . .	92
<b>G</b>	<b>Predicate Completion Tests</b>	<b>93</b>
G.1	Base File . . . . .	93
G.1.1	File: puzzle1_base.lp . . . . .	93
G.2	New Predicate Declaration . . . . .	94
G.2.1	File: puzzle1_new.lp . . . . .	94



G.3	Previous Predicate Declaration . . . . .	95
G.3.1	File: puzzle1_previous.lp . . . . .	95
G.4	Variable Predicate Declaration . . . . .	95
G.4.1	File: puzzle1_variable.lp . . . . .	95
G.5	Expected Results . . . . .	97
G.5.1	New Predicate Declaration . . . . .	97
G.5.2	Previous Predicate Declaration . . . . .	98
G.5.3	Variable Predicate Declaration . . . . .	99
<b>H</b>	<b>Code</b>	<b>101</b>
H.1	Framework . . . . .	102
H.1.1	File: AbstractArgumentConstants.java . . . . .	102
H.1.2	File: AbstractLaunchConfigurationDelegate.java . . . . .	103
H.1.3	File: AbstractLaunchShortcut.java . . . . .	106
H.1.4	File: AbstractMultipleFileArgumentsTab.java . . . . .	108
H.1.5	File: AbstractSingleFileArgumentsTab.java . . . . .	109
H.1.6	File: CommandLineSwitch.java . . . . .	113
H.2	Gringo . . . . .	114
H.2.1	File: GringoProgramArgumentConstants.java . . . . .	114
H.2.2	File: GringoArgumentsTab.java . . . . .	115
H.2.3	File: GringoLaunchConfigurationDelegate.java . . . . .	116
H.2.4	File: GringoLaunchConfigurationTabGroup.java . . . . .	117
H.2.5	File: GringoLaunchShortcut.java . . . . .	118
H.2.6	File: GringoPreferenceConstants.java . . . . .	118
H.2.7	File: GringoPreferencePage.java . . . . .	119
H.3	Conversion . . . . .	120
H.3.1	File: AbstractProgramConverter.java . . . . .	120
H.3.2	File: AbstractProgramConverterAction.java . . . . .	121
H.3.3	File: AggregateLparseToGringoConverter.java . . . . .	122
H.3.4	File: PoolingGringoToLparseConverter.java . . . . .	124
H.3.5	File: PoolingLparseToGringoConverter.java . . . . .	125

H.3.6	File: RemoveRestrictedNames.java . . . . .	127
H.4	Auto Completion . . . . .	128
H.4.1	File: AtomProposalGenerator.java . . . . .	128
H.4.2	File: IProposalGenerator.java . . . . .	129
H.4.3	File: LparseContentAssistProcessor.java . . . . .	130
H.4.4	File: NumericConstantProposalGenerator.java . . . . .	132
H.4.5	File: ProposalGenerator.java . . . . .	133
H.4.6	File: SymbolProposalGenerator.java . . . . .	135
H.4.7	File: VariableProposalGenerator.java . . . . .	136
H.4.8	File: PredicateView.java . . . . .	137
H.4.9	File: PredicateViewContentProvider.java . . . . .	140
H.4.10	File: PredicateViewCurrentFilter.java . . . . .	141
H.4.11	File: PredicateViewItem.java . . . . .	141
H.4.12	File: PredicateViewLabelProvider.java . . . . .	144
H.4.13	File: PredicateViewPredictedFilter.java . . . . .	144
H.5	Testing . . . . .	145
H.5.1	File: AbstractRegressionTest.java . . . . .	145
H.5.2	File: AbstractTest.java . . . . .	146
H.5.3	File: CommonTests.java . . . . .	148

# List of Figures

2.1	A Logic Programming System [GG85]	5
2.2	Declarative Problem Solving in ASP [GKK <sup>+</sup> ]	7
2.3	System Architecture Of Clasp [GKS09]	15
3.1	Experience With ASP	21
3.2	Editors Currently Used For ASP Development	22
3.3	ASP Tools Currently Used	24
4.1	Eclipse Architecture As Seen In [RB06]	30
5.1	Tab Group Structure	42
5.2	Single File Arguments	44
5.3	Multiple File Arguments	45
5.4	Conversion Options	49
5.5	Predicate Completion Ordering	53
5.6	Predicate Completion Proposals	54
5.7	Predicate View	55
5.8	Predicate View Options	55
6.1	Testing Plug-in Structure	59
E.1	New AnsProlog* Project	88
E.2	New LPARSE File	89
E.3	Installing APE	90
G.1	Predicate View Suggestions	97

G.2	Auto Completion Suggestions . . . . .	98
G.3	Predicate View Suggestions . . . . .	98
G.4	Auto Completion Suggestions . . . . .	99
G.5	Predicate View Suggestions . . . . .	99
G.6	Auto Completion Suggestions . . . . .	100

# List of Tables

3.1	Most Desired Features . . . . .	26
6.1	Project Creation & Deletion Tests . . . . .	60
6.2	Program Tests . . . . .	61
6.3	Predicate View Tests . . . . .	63
6.4	Preference Tests . . . . .	64
6.5	Installation Tests . . . . .	64
6.6	Predicate Completion Tests . . . . .	65
6.7	Predicate View Tests . . . . .	66

# Acknowledgements

I would like to thank my supervisor Dr. Marina De Vos for her support and guidance throughout this project. Additionally, thanks must be given to my friends and family who have supported me throughout.

# Chapter 1

## Introduction

Answer Set Programming (ASP) is a declarative form of programming, where the programmer specifies what needs to happen rather than how it should happen. Given a set of rules and facts, ASP will attempt to solve these, finding the consistent rules within the program which are known as the answer sets. The possible answer sets are computed in a two stage process. Firstly a grounder is run on the program, converting the variables within the program into atoms. This is followed by a solver running on the grounded program to compute the answer sets. Examples of grounder and solver program pairs are LPARSE/SMODELS and Gringo/clasp.

With the increasing efficiency of answer set solvers and novel application areas for answer set programming, ASP is becoming a useful programming paradigm for declarative problem solving and knowledge representation systems. [CDVBP08]

ASP programs are written in the language of AnsProlog\*. “AnsProlog\* is a short form for answer set programming in logic, and the \* denotes that we do not place any restrictions on the rules [Bar03].” Programs can consist of facts with a set of rules from which other facts can be derived. Programs look syntactically similar to Prolog programs with rules consisting of both a head and body. However differences are present between the two: AnsProlog\* allows rules to be placed anywhere, AnsProlog\* does not have the cut operator and AnsProlog\* programs cannot get stuck in an infinite loop.

Investigation was done in 2006 by Sureshkumar [Sur06] to add support for ASP in the form of an Integrated Development Environment (IDE). IDEs exist to enable and assist the programmer in achieving their goal of writing the program in the most efficient manner possible. Typical features include syntax highlighting, automating repetitive tasks, compiling code, debugging, as well as speeding up development by being able to run external programs from within the IDE. Nourie [Nou05] sums up an IDE as “a set of tools that aids application development.”

Sureshkumar developed a plug-in for the Eclipse platform called APE, standing for AnsProlog\* Programming Environment. Support exists within APE to run the external grounder LPARSE and solver SMOELS. APE also supports commonly found IDE features such as syntax highlighting, syntax checking to find compile errors as well as being able to generate a dependency graph of the program being developed. ASP is an emerging field with tools and programs being continually developed. Since 2006 further tools such as AspViz, SPOCK as well as Gringo and clasp have been developed. APE does not provide any support to run these.

This dissertation aims to take the work done by Sureshkumar, extending it to provide an IDE that is usable with the features most desired by potential users. As ASP is still an emerging field the project does not aim to add support for every possible external program found in the domain. Instead this project aims to produce an architecture that will allow future developers to quickly add programs in the future. Supporting this methodology further, the project aims to promote a testing strategy such that any future tools added will not cause existing features found within APE to break.

Previous development of APE has followed a evolutionary approach which Sommerville [Som06a] describes as “interleaving the activities of specification, development and validation.” However for this dissertation the approach was changed to use the more traditional waterfall model software process model. Sommerville describes this as taking the fundamental activities of the project and representing them as separate process phases. This process has been documented in the rest of the dissertation, which follows the structure below:

**Literature Survey** The dissertation starts off with a survey of relevant literature. This section covers previous research in the field of ASP as well as IDE development. APE is looked at in more depth, as well as logic programming in general.

**Requirements** This chapter documents the elicitation process of requirement gathering for the project. Gathering choices are explained, as well as documenting the key requirements that the APE must implement. The scope of the project is defined given the time scale.

**Design** The design decisions made for the project are explained in this chapter. The system architecture is introduced as well as the decisions made from the requirements.

**Detailed Design & Implementation** This chapter follows the same approach as the design chapter. A high level overview of the implementation process is given with key algorithms explained. Approaches are reflected upon, with decisions made throughout the project justified.

**Testing** This chapter explains the testing process and strategies that have been implemented for the project. Automatic and manual tests are performed as well as explanation as to how this will help future developers. Additionally the requirements are revisited to ensure the project has succeeded in meeting them.



**Conclusion** This chapter documents any findings from the project as well as providing a critical evaluation of the dissertation as a whole. The future direction of APE is considered with possible improvements and suggestions for future developers.

## Chapter 2

# Literature Survey

### 2.1 Introduction

An Integrated Development Environment (IDE) currently exists for Answer Set Programming (ASP) called APE. This is limited and needs to be further developed before it is widely accepted. Before extending upon APE, it is important to fully understand the subject by reading relevant literature. This will result in effective requirements being gathered. “If these requirements are not satisfied, it may be impossible to make the system (APE) work satisfactorily” [Som06b]. To begin this literature survey the wider field of logic programming will be researched before focussing on ASP, then onto APE.

### 2.2 Logic Programming

“The logic programming paradigm has its roots in automated theorem proving” [Apt03] and “Robinson’s resolution rule” [Bar03]. “Logic programming differs substantially from other programming paradigms” [Apt03]. Apt [Apt03] gives a good summary of the differences as:

- Computing takes place over the domain of all terms defined over a universal alphabet.
- Values are assigned to variables by means of automatically generated substitutions (general unifiers).
- Control is provided by a single mechanism, automatic backtracking.

Logic programming is both declarative and interactive. “A declarative program is viewed as a formula and one can reason about its correctness without any reference to the underlying computational mechanism” [Apt03]. “Benefits of this approach means programs are easier to create and enable machines to explain their results and actions” [GG85]. “Interactive

programming allows the user to write a single program and interact with it by means of various queries of interest to which answers are produced” [Apt03]. Visualisation of a logic program can be seen in Figure 2.1.

Baral [Bar03] explains that classical logic was the initial choice to represent declarative knowledge. Classical logic embodies the monotonic property with conclusions being made, often with incomplete knowledge. The monotonic property means that once a conclusion is reached it remains valid even when additional knowledge is known. This is not like human reasoning, where conclusions can be withdrawn or changed upon receiving more knowledge. This led to the development of non-monotonic logic to enable true declarative logic programming to occur. One of the main declarative logic languages used today is Prolog (Programmation en Logique [CR93]). “Prolog is a logic programming language in which the programmer states what is to be done rather than how it is to be carried out” [WC01]. Wilson and Clark [WC01] observe that the approach is goal orientated and dominates the artificial intelligence scene. However as Prolog evolved to be a fully fledged programming language with efficient implementations, it came at the cost of sacrificing the declarativeness[Bar03]. That was why AnsProlog\* was developed (see Section 2.3, page 6).

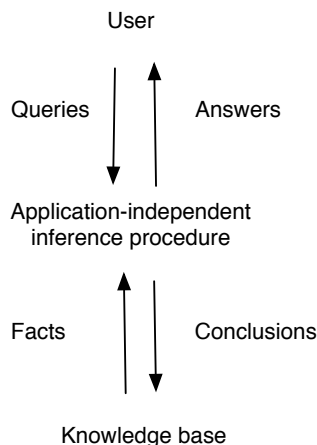


Figure 2.1: A Logic Programming System [GG85]

### 2.2.1 Parts of a Program

Syrjänen [Syr] states that there are four parts of a logic program: constants, variables, atoms and rules.

**Constants** Constants can be either numerical or symbolic. They are the individual things that exist in the problem domain universe. Symbolic constants normally begin with a lower case letter. e.g. `foo`, `bar`, `2`, `3`, `a`.

**Variables** Variables usually begin with a capital letter. Unlike other forms of programming no value is assigned to the variable. The underlying engine assigns the correct values or substitutes the constants in the correct place of the variable. e.g. `X, Bar, Foo`.

**Atoms** An atom consists of a predicate symbol followed by a parenthesised list of constants or variable. Atoms are used to express relationships between constants. e.g. `parent(rob,brian)`. This atom may say that Rob is Brian's parent. An atom is boolean, that is to say its value is either true or false.

**Rules** Rules allow inferences to be made based on the predicate. A rule consists of a head and body. In formal notation the rule is written as `head ← body` whilst in programming notation it is written as `head :- body`. An example of a rule is `sibling(X,Y) :- parent(Z,X), parent(Z,Y)` which tells us that X and Y are siblings if they have the same parent.

## 2.3 Answer Set Programming

To fully understand how APE can be used, the field of ASP needs to be fully understood. “In answer set programming a logic program is used to describe the requirements that must be fulfilled by the solutions of a certain problem” [Bar03]. “The answer sets of the program are usually defined through the stable model semantics” [GL88]. These sets then correspond to the solutions of the problem. ASP programs are written in AnsProlog\*.

“AnsProlog\* is short form for answer set programming in logic, and the \* denotes that there are no restrictions on any of the rules” [Bar03]. With the increasing efficiency of answer set solvers and novel application areas for answer set programming, “ASP is becoming a useful programming paradigm for declarative problem-solving and knowledge representation systems” [CDVBP08]. The process of programming in ASP is explained in the Gringo and clasp user manual [GKK<sup>+</sup>]. This describes the process as a four box model as can be seen in Figure 2.2, page 7.

Baral [Bar03] specifies that AnsProlog\* programs can be considered to be a collection of rules of the form:

$$L_0 \text{ or } \dots \text{ or } L_k \leftarrow L_{k+1}, \dots, L_m \text{ not } L_{m+1}, \dots, \text{ not } L_n.$$

Where each  $L_i$  is a literal. A positive literal is an atom, whilst a negative literal is an atom preceded by  $\neg$ .  $L_{m+1}, \dots, L_n$  are considered true if evidence cannot support the truth of  $L_{m+1}, \dots, L_n$ . **not** stands for negation as failure. With negation as failure, **not**  $L_i$  does not mean  $L_i$  is false, rather it cannot be proven to be true. This allows the answer set to be computed in the absence of knowledge rather than with what is known.

The answer sets of a program are defined as the answer sets of the ground program. “A term is said to be ground if no variables occur in it” [Bar03]. An atom is considered ground

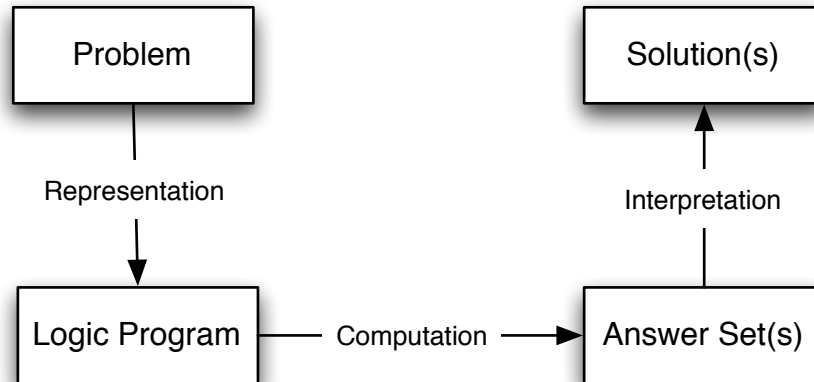


Figure 2.2: Declarative Problem Solving in ASP [GKK<sup>+</sup>]

if it only contains grounded terms. The Herbrand Universe  $HU_{\mathcal{L}}$  of a language  $\mathcal{L}$  is the set of all ground terms which can be formed with the functions and constants of  $\mathcal{L}$ . “The Herbrand Base  $HB_{\mathcal{L}}$  is the set of all ground atoms that can be formed with predicates from  $\mathcal{L}$  and terms from  $HU_{\mathcal{L}}$ ” [Bar03].

“For positive answer set programs (programs without negation as failure), the answer set is computed from the deductive closure” [DV09]. The answer set is computed by the following algorithm:

- Take all rules off which the bodies are true
- Assume the heads to be true
- Continue until a fixed-point is reached

This fixed-point is the unique answer set,  $\text{lm}(P)$  of the program.

**Example program:**

$a :- c, d.$

$c :- e.$

$f :- g.$

$d.$

$e.$

answer set = {e,d,c,a}

For programs with negation as failure the process is slightly more involved. Given a potential answer set  $S$ , the program is converted to  $\text{AnsProlog}^{not}$  using the Gelfond-Lifschitz (GL) transformation.  $\text{AnsProlog}^{not}$  is a subset of  $\text{AnsProlog}^*$  where  $k = 0$  and  $m = n$  in the context of the literal formula above (Section 2.3, page 6).

De Vos [DV09] defines the Gelfond-Lifschitz transformation as:

Let  $P$  be a logic program and  $M$  a set of atoms. The GL reduct  $P^M$  is obtained by:

- Removing each rule with some negative literal ‘**not**  $a$ ’ in the body such that  $a \in M$
- Remove all negative literals from remaining rules

If the answer set  $S'$  of the reduced program is  $S$  after reduction, then  $S'$  is an answer set of the original non-reduced program.

**Example program:**

$a$  :-  $b$ .  
 $b$  :- .  
 $c$  :- not  $d$ .  
 $d$  :- not  $c$ .

**Trying  $S = \{a, b, c, d\}$**

$a$  :-  $b$ .  
 $b$  :- .  
 ~~$c$  :- not  $d$ .~~  
 ~~$d$  :- not  $c$ .~~

Answer set  $S' = \{a, b\}$ .  $S' \neq S$ , therefore not an answer set of the original program.

**Trying  $S = \{a, b, c\}$**

$a$  :-  $b$ .  
 $b$  :- .  
 $c$  :- not  $d$ .  
 ~~$d$  :- not  $c$ .~~

Answer set  $S' = \{a, b, c\}$ .  $S' = S$ , therefore an answer set of the original program.

The full answer sets for the example program are  $\{a, b, c\}$  and  $\{a, b, d\}$

### 2.3.1 AnsProlog\* vs Prolog

“AnsProlog\* was born as a declarative alternative to Prolog” [Bar03]. Baral [Bar03] describes the differences as:

**Positioning of rules in AnsProlog\* does not matter** Prolog processes literals from left to right and top to bottom. In AnsProlog\* the ordering of literals does not matter, a program is considered a set of AnsProlog\* rules.

**The cut operator is not present in AnsProlog\*** The cut operator in Prolog is a goal that cannot be backtracked past. It is used to stop extra solutions being found or additional calculations being performed.

**Prolog can get stuck in infinite loops** Prolog has problems with programs that have recursion through the negation as failure operator. These problems are not found in AnsProlog\* as answer set semantics are used to characterise negation as failure.

### 2.3.2 AnsProlog\* Usage

“Answer set programming has been applied to several areas of science and technology” [Lif08]. Lifschitz [Lif08] gives three usages where ASP has already been used:

**Automated product configuration** ASP has been used for the creation of a web-based product configurator.

**Decision support for the space shuttle** An ASP system capable of planning and diagnostic tasks has been used for the operation of the space shuttle.

**Inferring phylogenetic trees** An ASP based method for reconstructing a phylogeny for a set of taxa has been applied to historical analysis of languages and parasite systems.

The decision to use ASP is due to the advantages that ASP provides. De Vos [DV09] gives the following advantages of programming in ASP:

- Programs are simple and quick to write
- Portable to different machine architectures
- Specification is equal to the implementation

## 2.4 IDEs

“A programming environment is a suite of programming tools designed to simplify programming and thereby enhance programmer productivity” [Rei96]. The programming environment should appear to the programmer as a single tool, “there should be no firewalls separating the various functions provided by the environment” [DMS84].

“Eclipse has become the dominant IDE for Java” [Gee05]. Geer [Gee05] states that the power of Eclipse is due to being inexpensive whilst being a common platform that different tools can be integrated with. Applying the same principles to APE will hopefully allow it to become widely accepted by the community. APE will come pre-packaged with a variety of ASP tools based on user feedback. However like Eclipse it must allow different tools to be added at a later date. By doing this it will allow it to stay future proof.

As the aim for this project is to build upon APE it is pointless to talk about ways other than Eclipse that an IDE for ASP could have been developed. Instead we will focus on the Eclipse plug-in framework, then what APE provides before looking at other IDE's in the logic field.

### 2.4.1 Eclipse Plug-in

The Eclipse platform developer guide [Ecl09] states that the whole Eclipse platform is structured as a core runtime engine with a set of additional features that are installed as platform plug-ins. Plug-ins contribute by adding functionality to pre-defined extension points. The plug-in identifies what extension points should be used. In addition the plug-in provides a manifest file (`MANIFEST.MF`) that describes the packaging and prerequisites. It also provides a plug-in manifest (`plugin.xml`) that describes the extensions the plug-in is defining.

APE was first developed in 2006. Since then the Eclipse plug-in API has changed considerably. The first job when extending upon APE will be to find any incompatibilities that may have been introduced by the newer versions of Eclipse. The Eclipse developer guide [Ecl09] defines the changes between different Eclipse versions. This will be the first place to look when identifying an incompatibility bug.

### 2.4.2 APE

APE was developed in 2006 by Adrian Sureshkumar. Sureshkumar [Sur06] stated that due to time constraints not all features that he would have liked had been implemented. The features that were implemented in APE were:

- Syntax highlighting
- Error and warning underlining
- Block commenting
- Launching LPARSE and SMOBELS from within a graphical user interface for program arguments
- Piping SMOBELS output to another program
- Multiple file support
- Common Eclipse features such as source control are also applicable to APE

Sureshkumar [Sur06] added that before APE can be widely used other solvers such as DLV need to be supported. This is so it will appeal to more users and will enable a bigger take-up of the tool. Sureshkumar [Sur06] also stated that rigorous testing should be performed



before releasing to the community. When first developed due to the small nature of the project no unit testing had been performed. When extending upon APE it is important that testing is added as a priority to enable maintainable code. It is also important to note that since 2006 a lot may have changed in the field of ASP and what was important then may be different now. Before developing any new functionality it is important to see what has changed and what is still relevant in the original tool.

### 2.4.3 Other IDEs

As Sureshkumar [Sur06] commented upon, Komorowski and Omori[KO85] and Francez et al.[FGP<sup>+</sup>85] present what the IDE scene for logic programming languages was like in the 1980's. Answer Set Programming is currently in a similar state. Doing a Google<sup>1</sup> search for *Answer Set Programming integrated development environment* presents no noticeable results. At the time of writing the top hit is in fact the knowledge representation and reasoning at the University of Bath website<sup>2</sup> where APE is currently documented. In the three years that have passed since APE was first developed it seems that no noticeable further work has been done in this area.

Looking at what features exist for Prolog will allow similar logic features to be analysed, deciding what features should to be added into APE. A Google search for *Prolog integrated development environment* presents many results. Tools such as the Prolog Development Tool (PDT)<sup>3</sup> exist as a plug-in to Eclipse. PDT's main features include syntax highlighting, searching for previous predicates as well as references to the predicate. It has features such as code completion as well as making development easier with keyboard shortcuts.

Other environments such as Visual Prolog<sup>4</sup> exist. This provides the ability to group items into packages giving an extra level of abstraction and a text editor allowing browsing to declaration and implementation.

From the two Prolog environments documented here it seems that adding support to APE to support searching for declarations and implementations of constants, variables and atoms would be a worthwhile feature. In addition code completion would be a useful feature for the IDE. If the user has already defined a constant, variable or atom then APE should recognise this and as such be helpful to the user by showing code completion possibilities.

## 2.5 ASP Tools

Many tools exist within the field of ASP. The ones added to APE will be the most desired from the community decided through means of a questionnaire. Some of the more commonly used ASP tools are documented here.

---

<sup>1</sup><http://www.google.com>

<sup>2</sup>[http://krr.cs.bath.ac.uk/index.php/Student\\_Projects](http://krr.cs.bath.ac.uk/index.php/Student_Projects)

<sup>3</sup><http://sewiki.iai.uni-bonn.de/research/pdt/start>

<sup>4</sup><http://www.visual-prolog.com/vip6/default.htm#VDE>

### 2.5.1 DLV

“DLV is a system for disjunctive data-log with constraints, true negation (à la Gelfond & Lifschitz) and queries” [FP]. “The DLV system supports disjunctive logic programming, where a program may contain disjunction in the head of rules and negation in the body” [LPF<sup>+</sup>06]. Baral [Bar03] explains that the DLV system processes the input program in several steps. Firstly it converts it to an internal representation, then it converts it to a program without variables. In the next step possible answer sets are computed with these answer sets going through post processing according to the front-end.

Currently APE does not support DLV due to time constraints in the initial development. Sureshkumar [Sur06] mentioned that the project is not open source meaning only binary builds are available. However this should not pose a problem for integrating DLV into APE as DLV will be used as it is, rather than developed further.

### Java Wrapper

A wrapper exists, as explained by Ricca [Ric] that wraps around DLV so that DLV programs can be developed as Java programs. APE will need to support a similar architecture in order to call DLV from within a Java program. Ricca [Ric] uses a `DlvHandler` class which provides the functionality to call DLV. The `DlvHandler` manages a DLV instance, which is a native process. This is done by using the `Runtime.exec()` method and `java.lang.Process` class. All DLV input and output operations go to the `DlvHandler` instance (inside the Java Virtual Machine) through a system pipe.

### 2.5.2 SMODELS

“SMODELS is the engine that performs the computation of the answer sets of the input program” [Bar03]. A SMODELS program may have none, one or many stable models, known as the answer sets. The stable models are seen as a rational belief about the program. SMODELS can be used either as a C++ library that can be called from user programs or as a stand alone program together with a suitable front end. The main front end known as a grounder is LPARSE. Other solvers that accept LPARSE input include `assat`, `nomore++` and `SMODELScc`.

### 2.5.3 LPARSE

“LPARSE programs are written using standard, though extended (such as `AnsProlog*`) logic programming notation” [Syr]. “LPARSE adds a layer of syntactic sugar on top of SMODELS” [Syr]. It is the most feature rich of the different parsers and front ends and is the default to use when writing SMODELS programs. LPARSE provides a grounded input program that can be used within a solver such as SMODELS. To ensure fast grounding by processing each rule only once the input must satisfy the property of being strongly range restricted

[Bar03]. This means that any variable that appears in the rule must also appear in the domain literal in the body.

“Other front ends exist in addition to LPARSE” [SYT]:

**smodels API** A C++ library that allows it to be called within any C++ program.

**parse** Original parser of SMODELS producing output in 1.x format

**primitive parse (pparse)** Accepts only ground programs producing output in 2.x format

**model checking models (mcsmodels)** A deadlock and reachability checker

**dlsmodels** An older version of mcSMODELS which detects deadlocks

#### 2.5.4 Cmodels

Cmodels is a system that computes answer sets for either “disjunctive logic programs or logic programs containing choice rules” [Liea]. It uses satisfiability (SAT) solvers to enumerate models of the program. Like SMODELS the input to the solver is a grounded logic program achieved by using LPARSE. Cmodels computes the supported model  $M$  of a logic program  $P$ . “Computing  $M$  is accomplished by forming the completion of  $P$ , classifying it (additional atoms can be introduced) and then invoking a SAT solver” [Lieb].

#### 2.5.5 Sup

SUP is a native Answer Set Programming (ASP) solver that can be seen as a “combination of computational ideas behind cmodels and SMODELS” [Lieb]. It can be thought of Cmodels without completion. Sup computes the supported models of  $P$  but does not form the completion of  $P$ . “Sup applies Minisat<sup>5</sup> to  $P$  and the non-clausal constraint mechanism of Minisat is employed for expressing the supportedness restriction” [Lieb].

#### 2.5.6 Gringo

Gringo, is a grounder capable of translating logic programs provided by users into equivalent propositional logic programs. Gringo is attractive as it “significantly extends the input language of LPARSE while supporting a compatible output format, recognised by many state-of-the-art ASP solvers” [GKO<sup>+</sup>09].

As Gebser et al. [GST07] explain, before Gringo there were only two major grounders LPARSE and DLV’s grounding component. Gringo combines and extends from both of these grounders. “A salient design principle of GrinGo is its extensibility that aims at facilitating the incorporation of additional language constructs” [GST07].

---

<sup>5</sup><http://minisat.se/>

Gebser et al. [GST07] describe the features that Gringo combines:

- The input language features normal logic rule, cardinality constraints and further LPARSE constructs
- It offers the new class of  $\lambda$ -restricted programs that extends LPARSE's  $\omega$ -restricted programs,
- Its instantiation procedure uses back-jumping and improves on the technique used in dlvs grounder
- Its primary output language is textual as with dlvs grounding component

$\lambda$ -restricted means that the variables in the program are bound by the rules of a smaller program. These features of Gringo allow for more optimised programs to be grounded. “GrinGo on most instances is faster than LPARSE” [GST07].

### 2.5.7 clasp

clasp works on logic programs in Gringo's output format. This numerical format, which is not supposed to be human readable is output by Gringo and can be piped into clasp [GKK<sup>+</sup>]. It is the best performing solver as described by Gebser et al. [GKS09]. Gebser et al. [GKS09] also explain that clasp was originally designed and optimised for conflict driven ASP solving centred around the concept of a nogood from the area of constraint processing.

The system architecture of clasp can be seen in Figure 2.3, page 15.

### 2.5.8 ASPViz

ASPVIZ is a Java program that “constructs two-dimensional images from the answer sets of a given program” [CDVBP08]. As explained by Cliffe et al. [CDVBP08] the tool takes two ASP programs,  $P_a$  and  $P_b$ . The former represents the given problem and the latter elaborates on the conclusions drawn by  $P_a$  concluding the necessary literals to render a graphic. To use this within APE would mean allowing the user to supply both parts of the program so that ASPVIZ can draw its output.

### 2.5.9 Debugging

Debugging is a major feature of an IDE and would be especially useful within APE. “Current ASP systems are still mostly experimental tools and their support for debugging is limited” [Syr06]. “Debugging answer set programs is a task of supporting a programmer in investigating why a program does not behave as expected, rather than a series of static tests that can be performed” [BD05].

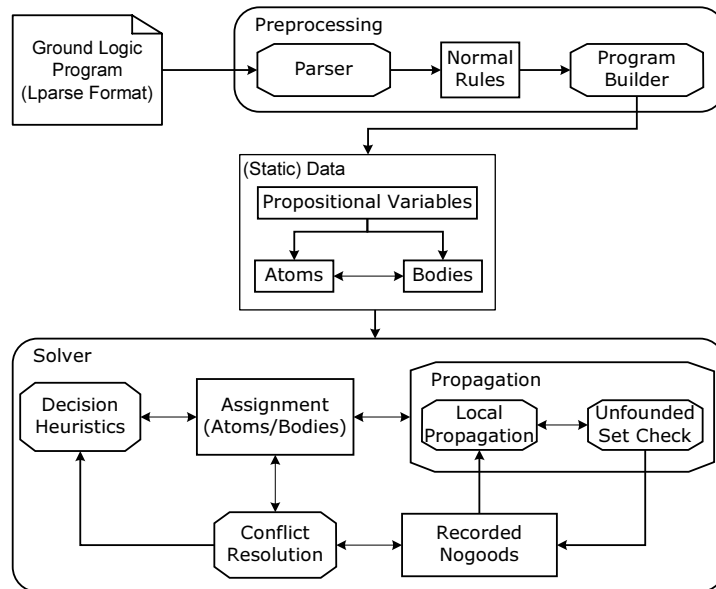


Figure 2.3: System Architecture Of Clasp [GKS09]

APE currently checks user input syntactically. This ensures that the syntax the user is entering is correct for the formal language. For semantic errors (where the program is syntactically correct) APE does not support any tools that allow the user to see why the behaviour is different.

Syrjänen [Syr06] says that semantic errors can be categorised into two types:

1. Typographical errors such as a misspelling of a variable or predicate
2. Logical errors where a rule behaves different than the intention

Currently in APE the most practical approach is to remove rules from the program until the resulting program has an answer set and then examining the removed rules to see what caused the error. “It is a non exact process, as it is subject to the potential differences between what the programmer wrote, what the programmer meant to write and what the programmer actually meant” [BD05].

A debugging tool SPOCK<sup>6</sup> explained by Brain et al. [BGP<sup>+</sup>07] exists for debugging ASP programs. The input of SPOCK is a logic program in the core language of DLV or SMODELS, read from the system standard input and/or from (multiple) files. The output varies according to the selected functionalities, determined by a set of options. The most impor-

<sup>6</sup><http://www.kr.tuwien.ac.at/research/systems/debug/index.html>

tant syntax extension of the input programs is the labelling of rules, allowing debugging mechanisms to explicitly refer to certain rules

## 2.6 Common Language

As discussed above a variety of different tools exist to allow users to ground and solve ASP programs. Some users may prefer to use `dlv`, others `LPARSE` with `SMODELS` and others `Gringo` with `clasp`. In some situations it may be desirable for users to convert between one or the other.

`Gringo` is built on top of `LPARSE` meaning programs written for `LPARSE` should work fine with `Gringo`. As `LPARSE` is more level restricted, programs tend to be more verbose than the ones for `Gringo`. “It is not suggested writing programs in the input language of `Gringo` for compatibility with `LPARSE`” [GKK<sup>+</sup>].

The key differences between `LPARSE` and `Gringo` are:

- `Gringo` supports min and max aggregates
- Variables within compound terms are supported e.g. `p(f(x))` `LPARSE` would treat it as `f(x)`
- `LPARSE` supports primed atoms such as `p'`
- Symbolic names are built into `LPARSE` such as `plus`. This is not the case for `Gringo` to not conflict with user defined names.

The user may want to convert between different solvers. As a first step APE should be able to highlight what is incompatible. This should be done by changing the tool type in APE and the syntax highlighting should be able to identify what would cause problems. Future steps could be to automatically convert between different solvers if any incompatibility issues occur.

If the user wanted to convert between different programs then APE should show what is common between the two and what will cause incompatibility for the first step. Future steps for APE would be to develop a common language and to convert to that in order for it to work between different solvers.

## 2.7 Regression Testing

Regression testing as explained by Wahl [Wah99] is a testing process that is used to determine if a modified program still meets its specifications or if new errors have been introduced. As the field of ASP is changing with new tools, debuggers, solvers etc. being developed then these will need to be added into APE. Having a good regression testing

framework will make the developers job straightforward as it will be easier to maintain. Additionally it will allow APE to be tested for compatibility issues when new versions of Eclipse are released.

Two forms of regression testing need to occur within APE. Firstly end to end input and output must be consistent between different versions. For example when running tools within APE such as LPARSE and SMOBELS, when given the same input, the output should be identical for the previous and current version of the code. Secondly the user interface should be tested after any change. This is significantly harder to test but it will ensure that developers have more confidence when making changes in future releases.

Commercial tools exist such as Window Tester<sup>7</sup> that enable automated regression and coverage tests to occur. However in the case of APE it is more appropriate to use open source tools such as SWTBot<sup>8</sup>. SWTBot provides an API that allows rigorous functional testing. SWTBot allows tests that interact with the applications user interface. Using a tool such as this would allow regression tests to be produced, testing the user interaction of APE.

## 2.8 Usability Evaluation

For APE to be deemed a success we must ensure that is accepted by the ASP community. For it to be accepted the factors behind what motivates people to use particular software must be researched. Seeley [See03] states that a good tool should promote novice learning and yet also be extremely efficient for experts. In order to ensure that the APE is being used correctly, evaluation of users using the system must be performed.

“Interface evaluation is the process of assessing the usability of an interface and checking that it meets user requirements” [Som06c]. The following attributes can be used to assess usability:

- Learnability
- Speed of operation
- Robustness
- Recoverability
- Adaptability

Observation of these attributes can be achieved through questionnaires, observations of users, video snapshots as well as embedding into the software, usage collection tools. Sommerville also mentions that it is good to give the users of the system a ‘gripe’ command.

---

<sup>7</sup><http://www.windowtester.com/>

<sup>8</sup><http://www.eclipse.org/swtbot/>

This allows users to pass messages to the designer about what they feel could be improved about the system.

## 2.9 Summary

The information outlined above has identified the field of logic programming and where the paradigm of ASP factors. Key texts and papers have been identified that can be looked at further for clarification or further points as the project progresses. Brief descriptions of different tools that are used in the ASP community are provided. Tools such as clasp, ASPViz and SPOCK have been made since APE was first developed. These have been researched and document how they fit into the APE model. Other logic IDE's such as Visual Prolog have been evaluated to see the features that they provide in order to see if relevant features can be added to APE.

The plug-in architecture for Eclipse is described showing how easily it is to add new features to APE. Also outlined is how the tool can be developed more easily in the future. Given the timescale and scope of this project not all tools will be added into APE. For future development work a regression framework would be useful. The different regressions that may occur are documented as well as how these can be tested and what tools exist in order to develop a good testing framework.

Finally documented is the different techniques needed to evaluate the interface design. For acceptance by the community a good user interface is a must. If the tool is good to use then people are likely to stick by the tool, otherwise they will carry on using existing methods. By now understanding the paradigm of ASP as well as the methodologies currently in use, a tool can be developed (APE) that will provide a methodology to support these.



## Chapter 3

# Requirements

In order to provide a successful system, requirements must be identified early on in the project. Sommerville [Som06a] describes requirements elicitation as “the process of deriving the system requirements through observation of existing systems.” Requirements are a vital part of the project with Goguen and Linde [GL93] describing projects failing because of inadequate requirements. The functional requirements define the behaviour that APE should provide. These include new features to be implemented, as well as improvements to existing features in APE. Constraints of the system, also known as non-functional requirements are thought about and documented. This chapter does not aim to give a full set of requirements for APE. The field of ASP is ever changing and it would be impossible to document every possible feature that may be useful in APE. Instead key requirements are documented concluding with the requirements implemented given the time scale limitations of the project.

Goguen and Linde [GL93] say a basic question for requirements gathering is finding out what users really need. The users in this project can also be considered as the stakeholders. As described by Preece [PRS02] these are the people that have an interest in the project. For this dissertation the stakeholders are ASP programmers as well as other members who have an interest in ASP programming. These are the people that will benefit most from successful completion of the project.

Requirements have been gathered from these stakeholders as well as from a developers point of view. In order to think of future developers, introspection as described by Goguen and Linde [GL93] has been used. Introspection is described as being useful but has the problem of being inaccurate if performed by someone in a different field. Given the requirements are thought of by a programmer to help future programmers, then the problems documented by Goguen and Linde [GL93] are not so much a concern. For this dissertation it was deemed a suitable method considering the developer requirements are a subset of the main requirements of the project. In addition time constraints of the project meant other ways of investigating requirements in this field could not be performed in suitable detail.

### 3.1 Gathering Process

Many approaches were considered for the requirements elicitation process. Goguen and Linde [GL93] discuss the various approaches that were available. Examples include questionnaires, face to face interviews, telephone interviews, focus groups and observation sessions. Ruane [Rua04] explains the main choice is choosing whether questions are to be asked via a questionnaire or as an interview.

A questionnaire is an extremely efficient data collection tool with Goguen and Linde [GL93] describing questionnaires as being useful when the population is large enough with a clear issue. Criticisms as explained by Ruane [Rua04] are that words can be put in the mouth of respondents with no meaningful information gathered. In addition no probing questions can be asked as respondents are not inclined to invest time to reply. As Ruane [Rua04] explains, an interview is much more personal. An interviewer can build a rapport with the respondent, being able to listen and interact to build knowledge of the subject.

For this dissertation a questionnaire was deemed suitable. A questionnaire allowed people from all around the world to participate without putting a burden on their time. Face to face interviews would have been a great monetary expense involving travelling and meeting people. In addition like telephone interviews, it would have impeded upon people's time. Therefore it was deemed suitable to allow users the choice of whether to complete the questionnaire. Questionnaires can suffer from low response rates, Ruane [Rua04] gives an example of less than 30%. However it was deemed enough responses would be received due to the large population as well as the advantages APE would provide.

The questionnaire as suggested by Ruane [Rua04], was designed to “stand on its own.” Benefits of this approach are the elimination of time and space barriers. Suchman and Jordan [SJ90] describe that most useful information in requirements gathering is obtained when both interviewers and respondents negotiate the meaning of questions and answers. Unfortunately the questionnaire hinders this process. However in the case of this project, when feedback was received it was acted upon. For example, following feedback the first question was changed from ‘What do you currently use to develop ASP programs?’ to ‘Which text editor do you currently use to develop ASP programs?’ to ensure greater clarity for respondents.

The questionnaire distributed to the stakeholders can be found in Appendix A, page 79. Likewise the results received can be seen in Appendix B, page 81.

One area that was not explored was observation sessions to see how users use the current ASP tools available to them. Again the same problems as face to face interviews would be encountered. Therefore it may have been suitable to limit these sessions to only observe people within the university. This would have relied on people giving up their time to allow observation of them. However due to the suitability of the questionnaire results and time constraints this avenue was not explored.

## 3.2 Discussion of Results

The questionnaire was distributed to forty six people. Seventeen responses were returned, giving a response rate of 37%. This low response rate may be explained by people being unwilling to complete the questionnaire or people on the mailing list now being inactive in the ASP scene. The feedback from the responses received was extremely positive. Although it was a low response rate the justification to carry on with the development of APE was found from the comments received. Such comments received included ‘great initiative’ and ‘this sounds like an excellent idea, could definitely help make developing ASP programs easier.’

### 3.2.1 Experience

In order to better understand the experience of users, they were asked how many years they had been using ASP for. Most people had development experience ranging from 1 to 10 years. A couple of people questioned had experience of 15 years and 20 years. However the majority were considerably less. This suggests that the people questioned have enough experience in ASP to give valuable suggestions about the direction that APE development should head. The respondents experience is widely spread meaning that the answers will be coming from different perspectives. The danger of answers from more experienced users is that they could have workarounds in place and do not feel the need to change. Likewise with inexperienced users the danger is that people do not know the domain well enough to give useful feedback. However for this project there is a mixture giving a variety of views enabling requirements to be gathered from both experienced and inexperienced ASP users.

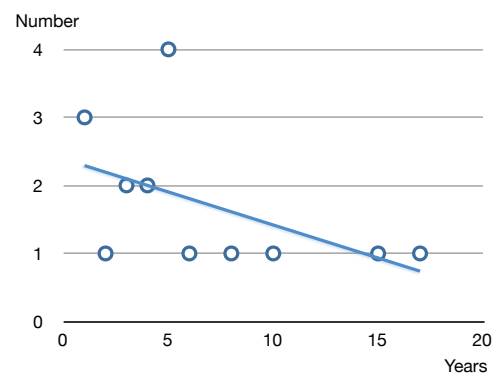


Figure 3.1: Experience With ASP

### 3.2.2 Current Editor

The main fundamental part of ASP development is the writing of the ASP program. For a better understanding of currently used tools, a question was asked to find out which text editor the respondents preferred to use. The question was designed to see whether any particular editor is the preference and if so what features does it have that could be incorporated into APE.

From the answers received it can be seen that the most popular choices are Emacs and

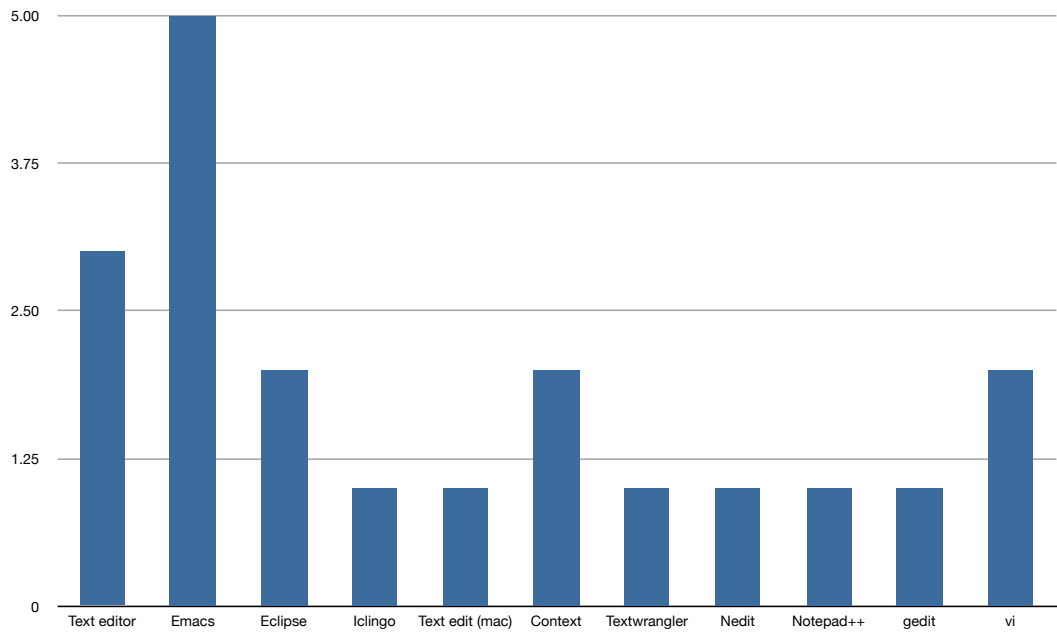


Figure 3.2: Editors Currently Used For ASP Development

Vi(m). Some people answered with the response text editor before the question was changed, as explained in Section 3.1, page 20. Some people specified that they use Eclipse, not as part of an IDE just as a text editor. The responses show that the market is there for APE to penetrate as no one questioned mentioned using an existing IDE. To ensure that people move across to using APE it will need to be publicised as well as being user friendly. User interface design is discussed in the literature survey (Section 2.8, page 17). Ensuring that these principles are met will ensure that the migration across to APE will be as seamless as possible.

### 3.2.3 APE

The next question asked users outright if they had heard of APE. Of the responses received only 45% had heard of APE, with no one actively using it. The reasons given for this were:

- APE is not compatible with the latest version of Eclipse
- Not enough features are provided
- It is hard to use

Firstly, 55% had not heard of APE leading to the conclusion that the work must not have been widely publicised following the work done by Sureshkumar. However following the responses received this is perhaps a good thing as one response was that APE does not work correctly with Eclipse. This may just be a single user issue or it may be part of a wider problem. If the latter it is a good job that not many people have used it as their view of the product would be skewed, with users having negative connotations towards APE.

Following other responses APE will need to support the features that users want. As it stands APE is a bare bones IDE and will not be suitable for all users. However by the end of the project it will have been extended making it suitable for more users. If these features are not provided then quite simply users will not use the IDE.

### Testing

One of the issues raised with the current version of APE is that it is incompatible with the latest version of Eclipse. Features that should work do not. Therefore a requirement to do regression testing was added. As discussed in the literature survey (see Section 2.7, page 16) SWTBot can provide a easy to use interface for this. By having regression tests in place it will enable errors to be picked up when new versions of Eclipse are released. Additionally it will allow future developers to implement new features, ensuring that previous working features have not been broken because of their changes.

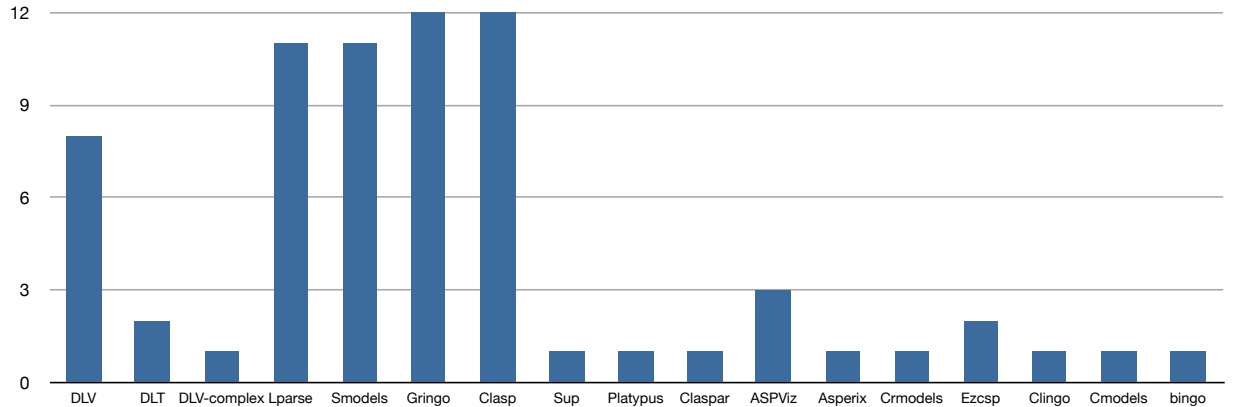


Figure 3.3: ASP Tools Currently Used

### 3.2.4 ASP Tools

As discussed in the literature survey many ASP tools exist that are not currently present in APE. However, too many tools exist to add all of them into APE given the scope of the project. A question was asked asking users what tools they actively use in their ASP development. By gauging their responses it enabled the common tools to be identified and integrated into APE, rather than spending time adding tools that would not be used by any users.

The results were not that surprising with the main better known tools having the market share of use. The most popular tools of the users questioned were:

1. Gringo
2. clasp
3. LPARSE
4. SMODELS
5. DLV
6. ASPViz

LPARSE and SMODELS are already present in APE so no further work needs to be done for them. Gringo and clasp are the most popular grounder and solver combination meaning that support will definitely need to be added. The specifics of these tools can be found discussed in Section 2.5, page 11. DLV is another tool that could be added however

depending upon time constraints may not. 17% of the respondents used ASPViz so like DLV it will be added into APE if time constraints will allow.

In addition to the popular tools listed above, it was shown that users make use of other tools that had not been considered in the literature survey. Responses showed that users make use of the following programs:

**DLV-Complex**<sup>1</sup> extends DLV by means of functions. It is a powerful system supporting functions, sets, lists as well as libraries.

**ASPeRIX**<sup>2</sup> Grounding is performed on the fly meaning no pre-grounding processing is performed. Only ASPeRIX is needed for both grounding and solving.

**EZCSP**<sup>3</sup> is an inference engine that allows computing extended answer sets of ASP programs.

**Platypus**<sup>4</sup> is an extensible distributed platform for answer set programming.

**Claspar**<sup>5</sup> is a parallelised version of clasp.

**Clingo**<sup>5</sup> combines both Gringo and clasp. Takes the input as the input language of Gringo and outputs in the output language of clasp.

**Bingo**<sup>6</sup> is still under development. It is an experimental bottom-up grounder based on semi-naive evaluation.

Although it would be nice if APE supported every tool listed above it would be infeasible given the timescales of the project. It was decided that support should definitely be added for the most popular, Gringo and clasp. For the other tools it was decided to not add support for them at this time within APE. This was to allow implementation to focus more on specific IDE features as discussed in Section 3.2.5, page 26.

## Framework

Gringo and clasp are the main tools that will be added into APE following user feedback. However in future, integration of other tools should be made as seamless as possible. To allow for this a framework will be developed at the same time as integrating Gringo and clasp. By ensuring a suitable framework is in place it will enable future developers of APE to rapidly add in new programs without having to worry about low level implementation details. The framework will handle the difficult coding for the developer.

---

<sup>1</sup><http://www.mat.unical.it/dlv-complex>

<sup>2</sup><http://www.info.univ-angers.fr/pub/claire/asperix/>

<sup>3</sup><http://krlab.cs.ttu.edu/~marcy/ezcsp/index.html>

<sup>4</sup><http://www.cs.uni-potsdam.de/platypus/>

<sup>5</sup><http://potassco.sourceforge.net/>

<sup>6</sup><http://potassco.sourceforge.net/labs.html>

<b>Feature</b>	<b>Count</b>
Predicate completion	9
Conversion between solvers	10
Debugging	7
Integration of dlv	2
Visualisation	7
Integration of Gringo/clasp	5
Replacement of a rule by its grounding	2

Table 3.1: Most Desired Features

### 3.2.5 Features

Although the most popular tools in use by the ASP community have been identified this does not necessarily correspond to the features that possible users may want to see in the IDE. An effective part of requirements gathering is ensuring that the system matches the needs of the user. In order to do this users were asked to pick their three most desired features for an IDE for ASP. The results can be seen in Table 3.1.

From the results it is clear that the most desired features are being able to convert between different solvers as well as having predicate completion. After this the features most desired are debugging as well as visualisation of the program being developed. Both debugging and visualisation would be implemented through the means of external tools, specifically SPOCK and ASPViz. Both of these are explained in the literature survey in Section 2.5, page 11. As a framework is being developed to enable ease of future implementation it was decided to not add SPOCK or ASPViz initially. Instead they could be added if time suffices. After that the most desired feature is integrating Gringo and clasp into APE. This is understandable given that they are the most widely used tools as seen in Section 3.2.4, page 24.

#### Conversion Between Solvers

As stated above the most desired feature from respondents is the ability to convert between different solvers. The answer did not mean a true solver but rather a grounder and solver pair. Such examples of grounders are LPARSE and Gringo. Although Gringo extends the language of LPARSE differences between the two are apparent. Examples of which are discussed in Section 2.6, page 16.

As LPARSE and Gringo are the most common tools in use then it made sense that the type of conversion supported should be between the two grounders. A program written in the syntax of LPARSE should be able to be converted to Gringo and ran through clasp returning the same answer set as if the original had been ran through SMODELs. Likewise the opposite should occur with a Gringo syntactically correct program being converted to



a LPARSE accepted program giving the same computed answer sets through both solvers, clasp and SMODELS.

A huge advantage to the user is the time that this conversion could save. If it cannot be done automatically the user has to manually perform the cumbersome job of rewriting the program this time in the syntax that will be recognised. Having this done automatically by a few clicks or a keyboard shortcut speeds up the user's development process.

### **Predicate Completion**

Providing predicate completion would reduce the time taken to input the program. For ease of understanding the program, predicates often follow a naming convention. By having a descriptive predicate name it saves the programmer having to explain what it does through other means such as commenting. Having predicate completion would save the programmer having to type the full name each time. By only typing the first letter and seeing a list of proposals, input time can be reduced. In addition it can reduce the number of errors caused by mistyping predicate names later on.

A predicate view could also be provided to the programmer to allow them to see an outline of the program and all possible actions they can take from previous predicates defined. The view would allow the developer to see all the previously declared predicates as well as the constants, variables and rules in the program. The view could provide an additional way of auto completion for the user. The user could double click on some text in the view in order to insert the text into the editor document at the cursors position.

### **Other Features**

Space was left on the questionnaire to allow users to highlight new features that may not have been covered by other aspects of the questionnaire. The only response received was for the integration of bingo/caspd to be added to APE. As this was just the one response it was decided that it was not worth investigating bingo or caspd as other features were more of a priority.

## **3.3 Summary**

The requirements gathering has helped to give a fuller insight into what potential users of APE wish to see from the IDE. Non-functional requirements have been identified such as which tools APE should support. In addition improvements and features have been suggested and explored in more detail. Requirements have been considered from a developers point of view, planning for the future. Examples include having a framework so future developers can add in additional tools easily. In addition maintainability is also considered by constructing a suite of regression tests that can be run for future releases of APE.

A full requirements specification can be found in Appendix C, page 85. However as documented above only specific features will be considered for further development of APE. This is because of the time constraints faced when completing an undergraduate dissertation. The requirements to be considered, thus the scope of the project given the time scale can be summarised as follows:

- Support for Gringo and clasp
- Auto predicate completion
- Conversion between LPARSE and Gringo
- Conversion between Gringo and LPARSE
- Framework to add new features
- Regression testing ability
- User friendly

# Chapter 4

## Design

Following the requirements process the next step was the design stage. The design stage allows the requirements to be thought about in more detail, assessing the best way to implement these. This chapter documents the rationale behind these decisions and the choices made prior to implementing the system.

### 4.1 Architecture

APE as it currently stands is implemented as an Eclipse plug-in. The reasoning behind this decision is documented in [Sur06]. In this document Sureshkumar justifies and gives examples of many of the advantages of choosing Eclipse. Some given are:

- Basic functionality already provided by Eclipse
- Multi-platform support as Eclipse is written in Java
- Proven history of successful plug-ins built on Eclipse architecture

As much previous work had been done by Sureshkumar changing the architecture was not even considered. By changing the architecture it would have meant time was spent repeating and re-implementing the work done by Sureshkumar albeit using different tools. Given the timescale, no new features would have been added providing no benefit to the overall goal of developing a suitable IDE for users of ASP.

#### 4.1.1 Eclipse Plug-ins

In order to understand how APE fits together the plug-in architecture of Eclipse must be understood. Rivieres and Beaton [RB06] describe the Eclipse platform in great detail. Apart from a small kernel called the platform runtime, all of Eclipse's functionality is found

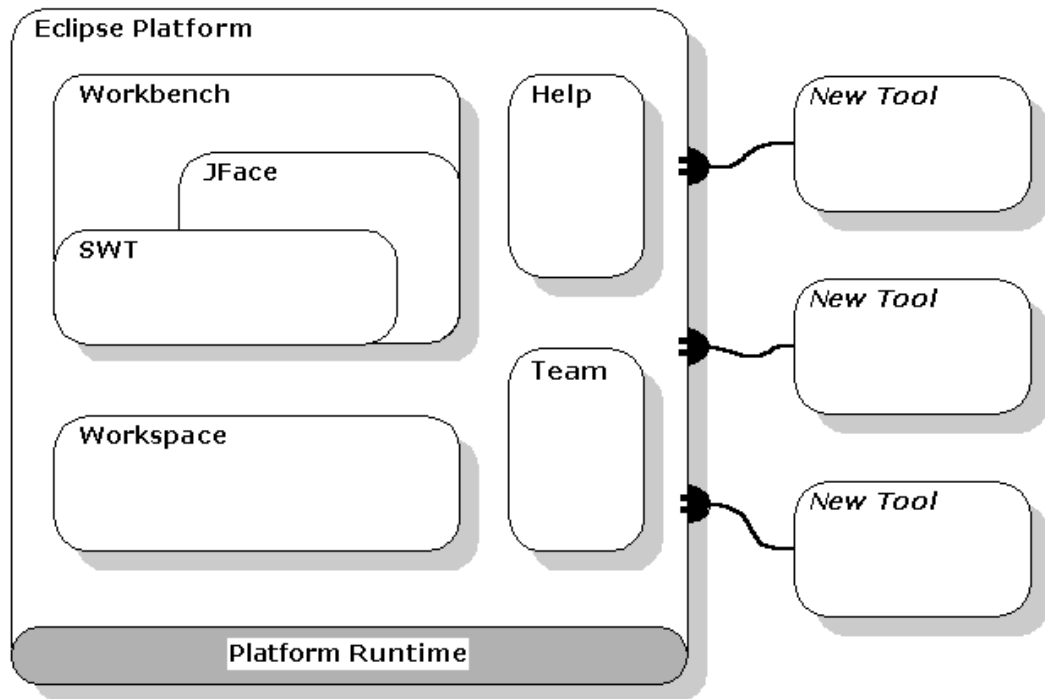


Figure 4.1: Eclipse Architecture As Seen In [RB06]

located inside plug-ins. The structure can be seen in Figure 4.1. Multiple plug-ins interact with Eclipse's core seven plug-ins. The platform runtime then brings them altogether to be the application known as Eclipse.

A typical Eclipse plug-in consists of:

- Java code in a Java ARchive (JAR)
- Read-only files
- Resources such as images
- A manifest

The manifest is the way a plug-in declares how it interconnects to other plug-ins. This is managed through extension points and extensions. Plug-ins can implement behaviour by extending from an extension point. For example by extending the extension point `org.eclipse.ui.preferencePages` each plug-in can contribute its own preferences. A manifest is represented by two files `MANIFEST.MF` which describes the runtime dependencies

of the plug-in. The other is `plugin.xml` which lists the extensions and extension points that the plug-in implements.

Upon startup this all combines together to provide a usable application. The platform runtime discovers all plug-ins and builds an in memory plug-in registry after reading all the plug-in's manifests. Each plug-in is lazy, in that it is only activated when it is needed in order to reduce the memory footprint of Eclipse.

All of this development is made easy through the use of the Plug-in Development Environment (PDE)<sup>1</sup> which itself is an Eclipse plug-in to aid the development of Eclipse plug-ins.

#### 4.1.2 APE

As discussed by Rivieres and Beaton [RB06] complex tools are usually split across multiple plug-ins. Sureshkumar followed this guidance splitting APE into seven plug-ins. As suggested by Clayberg and Rubel in [CR08] they have been broken down into User Interface (UI) and core plug-ins. The UI plug-ins are for the user interface whereas the core plug-ins are designed to operate in a headless environment (one without a UI).

As discussed by Sureshkumar in [Sur06], APE consists of:

- **uk.ac.bath.cs.asp.ide** - Logging
- **uk.ac.bath.cs.asp.ide.ui** - ASP perspective and syntax colouring
- **uk.ac.bath.cs.asp.ide.dependencygraphs** - Dependency graph model and dependency graph user interface
- **uk.ac.bath.cs.asp.ide.lparse** - LPARSE preferences, LPARSE parser, document model and LPARSE launcher
- **uk.ac.bath.cs.asp.ide.lparse.ui** - LPARSE editor, LPARSE preference page and LPARSE launch configuration user interface
- **uk.ac.bath.cs.asp.ide.smodels** - SMODELS preferences and SMODELS launcher
- **uk.ac.bath.cs.asp.ide.smodels.ui** - SMODELS preference page and SMODELS launch configuration user interface

A problem with this approach is that if APE continues to grow then it will become unmanageable. If every new feature is going to have a core and UI plug-in then the number of plug-ins increases very quickly. Another problem is that boundaries are blurred across plug-ins. Taking the LPARSE plug-ins as an example, a couple of problems can be seen upon first observation. In order to access the preferences in the UI plug-in part, the name

---

<sup>1</sup><http://www.eclipse.org/pde/>

of the LPARSE plug-in is hard coded. This means that it is very fragile if any refactoring occurs such as renaming a package or plug-in name then APE will break.

Secondly a problem can be seen from a developers point of view. LPARSE is an external program meaning that arguments need to be passed to it. These arguments need to be found for the user in the launch configuration interface. In addition they need to be found in the launcher found in the LPARSE plug-in. As a result these arguments are repeated in both the interface and the launcher. This is not maintainable and if LPARSE changes then changes have to be made in two locations.

As part of the new framework to be developed (see Section 4.2, page 33) this will be resolved, however it will not be suitable to have separate core and UI components. Therefore the original APE plug-ins will be refactored leaving four from the original seven. The resulting plug-ins will be a combination, resulting in:

- **uk.ac.bath.cs.asp.ide**
- **uk.ac.bath.cs.asp.ide.dependencygraphs**
- **uk.ac.bath.cs.asp.ide.lparse**
- **uk.ac.bath.cs.asp.ide.smodels**

This refactoring provides a clearer structure as well as enabling the launcher and interface to share the same components.

Of course this dissertation is aiming to provide new and usable features for the potential users. Therefore to keep the same pattern three new plug-ins will be added to the feature of APE. These will be:

- **uk.ac.bath.cs.asp.ide.gringo**
- **uk.ac.bath.cs.asp.ide.clasp**
- **uk.ac.bath.cs.asp.ide.testing**

## **Installation**

All the plug-ins documented above combine together to create a feature. A feature provides the branding of the plug-ins; bringing together a group of related plug-ins that install and get updated together. Of course this is the whole project known as AnsProlog\* Programming Environment (APE). The feature project already exists and can be found under the name **uk.ac.bath.cs.asp.ide.feature**. At the moment this feature is installed by dragging and dropping the JARs into the plug-in directory located inside the Eclipse install folder. This is not user friendly and is something that will be changed in order to ensure APE is as easy to install as possible.

A new update site project called **uk.ac.bath.cs.asp.ide.update.site** will be created wrapping around the feature to allow users to install the plug-in by connecting to a website that will install or update APE. In addition the feature will be able to periodically keep checking upon the website to inform the user when an update is available. This will alert users to new features that may be available in the future. This is a much easier method than having to download, then drag and drop files each time a user wishes to install a new update.

## 4.2 Developer Framework

A new framework is going to be developed that will allow future tools to be integrated into APE much more easily. In order to do this existing tools were analysed to understand how they were implemented. This allowed identification of shared components that could be reused.

APE currently consists of two external tools, LPARSE and SMOBELS. Both tools within APE enable the launching of an external tool. This is exactly what will need to be done when Gringo and clasp are added into APE. However future tools may be added so it made sense to develop a framework to allow future features to be added quickly as well as being maintainable.

The implementations of LPARSE and SMOBELS were analysed to find the similarities that exist between the tools. The main things found were:

- Each program has program arguments (e.g. `lparse -help`)
- A launch configuration delegate
- Launch configuration interface
- Launch shortcut

The framework was designed so that these similarities were supported by it. Choices needed to be made such as how are the clients (e.g. Gringo, LPARSE) going to instantiate it? How are clients going to be able to share information? Are interfaces going to be helpful? Is there code to re-use?

It was decided that the easiest method would be to allow clients to subclass key classes in the framework in order to instantiate. Each client can be ran as a module keeping it away from depending upon others. Modularity plays a key part in safe software allowing for better testing, as well as developers understanding the structure better. Having these classes abstract means only clients will be able to implement them. A choice then had to be made about whether to provide clients with interfaces within the framework or whether to restrict them to an abstract class.

Bloch [Blo08] states that interfaces should be preferred to abstract classes. “Interfaces enable safe, powerful functionality enhancements [Blo08].” Abstract classes however leave

the programmer with no alternative other than inheritance, leading to more fragile and less powerful classes. However Bloch does state that it is “far easier to evolve an abstract class than an interface .” As the framework is new and likely to evolve it was decided that for this project abstract classes would be favoured, with any methods that the client needing to implement being declared abstract in the abstract class. Additionally interfaces can be added later as the framework develops if there is a justifiable reason for the client needing them.

### 4.3 Conversion

The most desired feature from the requirements gathering was the ability for users to automatically switch between different solvers. Given the scope of the project the requirements narrowed it down to converting from Gringo to LPARSE and vice versa. As discussed in Section 2.6, page 16 differences exist between the input languages between Gringo and LPARSE.

To solve this problem there are two possible solutions that stand out. The first possible option is to parse the LPARSE program into the internal model representation within APE. This then enables easy checking of the program to spot any incompatibilities between the model and what is acceptable Gringo syntax. However, a problem occurs with this method when converting back the other way from Gringo to LPARSE. No functionality exists to convert to internal Gringo representation. This of course poses a problem as this model cannot then be analysed to spot incompatible LPARSE syntax.

The alternative method is to use regular expressions to scan the file and to spot the incompatibilities. Action can then be taken to convert from Gringo to LPARSE. This is the approach that will be taken in order to ensure consistency between both methods added to APE.

### 4.4 Predicate Completion

A new feature to be implemented, learnt through the requirements process is that users would like automatic predicate completion. This is a common feature found in many IDEs and makes the programmer’s life a lot easier. The most appropriate choice to implement this would be to have it as a keyboard shortcut. To keep it consistent with the Eclipse Java content assist processor, the control key and space bar should be pressed to pop up a small dialog box inline, showing all the possible suggestions to the user.

Upon pressing the shortcut the possible predicates are displayed for the user to choose from. The possibilities could be facts, constants, variables or predicate names. The choices for the user to choose from are two pronged. The first choices are based on the letters typed and then the possible choices from previous declarations. Secondly the choices are based on what the user has typed as being the full predicate name. The choices then presented



to the user are the possible constants or variables.

Additionally it was decided that a predicate view should be implemented into APE. This was designed to be implemented as an Eclipse standard view window. A view window is a panel in Eclipse that can be docked on any side toolbar. Alternatively it can be minimised or even expanded. By implementing it like this it allows the user control over the view, deciding how and when they want to see it.

## 4.5 Testing

Testing is a fundamental process to any software development project. Testing is the process to ensure that the system works as expected with the requirements of the system implemented correctly. Different approaches are taken in a project depending on the methodology of the project. People such as Proulx [Pro09] prefer an agile approach with test driven design favoured, reasoning that higher quality code is produced. However as this dissertation is following a waterfall style development process then such agile methods are not applicable. Instead a higher level view of the testing process is considered before deciding on the most appropriate tests.

Three different views of a system can be considered when tests are performed:

**Black box** The system is treated as an external entity (black box) where no internal representation of the system is known.

**White box** The tests have access to the internal data representations and assumptions made in the system.

**Grey box** A hybrid of the other two approaches. The internal representation is known but the tests are performed at a black box level.

Testing is not a short cycle and many different levels of testing exist in order to fully understand the system. Some of the main stages in the cycle are:

### Unit Testing

Unit testing is the first stage of testing. Szeder [Sze09] says the benefit is that it allows developers to detect bugs early on in the development process. All individual components are tested leading onto integration testing. One such unit testing framework that exists for Java is JUnit. As described by Wick et al. in [WSW05], JUnit has a number of fundamental concepts. Each test case is a Java class that consists of one or more test methods. Each method tests some particular aspect of the Java class being tested. Multiple test cases can be structured so that they are part of a test suite.

Problems are faced when unit testing is performed. User interfaces are not easily testable as well as mistakes made in the construction of the actual tests. A common pitfall faced is when tests are too simple meaning complex paths throughout the system are not tested.

## **Integration Testing**

Following on from unit testing it is common for integration testing to be performed. As all the individual components of the system have been tested they then need to be tested so that the interaction between components is correct. The interfaces that make up the communication between components have to be tested to ensure that the design is met.

## **Regression Testing**

Another stage of the test cycle is regression testing. These tests are usually ran after integration tests. The tests aim to find errors in the system after changes have been made to the previous version of the system. Rothermel et al. [REM<sup>+</sup>04] gives a definition as “Regression testing is an expensive testing process used to revalidate software as it evolves”.

### **4.5.1 APE Tests**

Potential users of the system have already been put off from using APE due to incompatibilities encountered with APE (see Section 3.2.3, page 23). Having tests in place would have identified this. Due to the limited time it would not be suitable to devise tests for all three test cycles above. Instead a hybrid approach of unit testing and regression testing was adopted.

Implementing any testing strategy comes with an associated development cost. For a large project this cost can be thought of as a temporary short term cost especially when the application will evolve. The time cost spent will be recuperated in the long term when the system is large and tests will pick up many errors. For a smaller development project the cost is much harder to recover. The project may not evolve with the time spent devising tests not being recovered in the long term. This was a consideration for this project. Given the limited timescale in place then the time spent devising tests may be better adding new features. Likewise not having enough test coverage may mean that the APE is full of bugs frustrating users. Therefore the project aimed to strike a balance between good tests and the time spent developing them.

As mentioned above the right mix of testing had to be implemented. The unit tests devised were carefully designed so that they were not too simple and covered enough of the code base to be useful. The unit tests have been designed so that they are decouple from one another. No test depends upon another. Unfortunately tools such as JUnit are not designed to test user interfaces. However an open source tool SWTBot built on JUnit is available to use. The testing is performed at the grey box level where the internal

data representations are known but performed from the perspective of a potential user of the system. SWTBot provides this functionality automating the user interface testing by clicking through simulating a user.

Regression testing will be performed by comparing the expected output of the program and the actual output. For each tool in APE the expected output will be stored. SWTBot can then be used so that they are used with the same input files. The output from each tool can then be stored with a check occurring with the expected and output lines read line by line for equality.

As the system grows and tests are added the time taken for the whole test suite to run will increase. In future it may be more suitable for only a subset of tests to run. When one aspect of the system is changed then the tests for that area will be ran. However a full test run of all tests will be ran regularly as well. To be able to do this the tests have to be designed so that they can be ran independently from one another. Therefore an architecture will be implemented to perform common tasks that all tests may have to share.

## Chapter 5

# Detailed Design and Implementation

The previous chapter identified ways in which the system should be designed in order for the requirements as outlined in Section 3 to be successfully implemented. This chapter documents how these have been implemented, as well as the further design choices faced along the way.

### 5.1 Approach

APE is licensed under the General Public License<sup>1</sup> (GPL). This type of licence explicitly declares that users can freely run, copy, change, develop and improve the software. Modifying and improving APE will not infringe upon this licence. APE will continue to be licensed under the GPL to allow future developers the same freedom.

The version of APE produced by Sureshkumar was hosted on the Knowledge Representation and Reasoning (KRR) website<sup>2</sup> at the university. As with all software projects the first stage was to add the existing code into a version control system, making use of the advantages that this provides.

Ruparelia describes in [Rup10] that one main advantage of version control is that it enables developers to keep historical versions of source code and project files that are under development, with the ability to retrieve past versions. A variety of choices exist with two of the most common being concurrent versions system (CVS) and subversion (SVN). As Ruparelia explains, distributed version control systems such as GIT are becoming increasingly popular in the open source community. SVN revolves around one repository with each client connecting to that one repository. GIT on the other hand is a repository with each client having its own repository, each with a user. It is decentralised meaning people

---

<sup>1</sup><http://www.gnu.org/licenses/gpl.html>

<sup>2</sup>[http://krr.cs.bath.ac.uk/index.php/Main\\_Page](http://krr.cs.bath.ac.uk/index.php/Main_Page)

can track their own edits locally without having to push things to an external server.

As APE in future may have multiple users contributing it was decided that GIT would be the best form of version control. By hosting it on Github<sup>3</sup> it allows people to clone and work on their repository before merging the changes back in. Github has the advantage of allowing SVN access to the GIT repository<sup>4</sup>. This made it an easy choice as people have the choice of using SVN or GIT for developing APE.

## 5.2 Improvements

As documented before (see Section 3.2.3, page 23 and Section 4.5.1, page 36) users initially trying APE had experienced difficulties getting it to function with the latest version of Eclipse. Trying the existing code as supplied by Sureshkumar revealed the problems. When trying to launch LPARSE or SMOBELS through a launch shortcut in APE, nothing would appear to happen. A launch shortcut allows the user to run the program by right clicking in the editor or right clicking on a file and launching it. Sureshkumar had only implemented the latter method, no code was in place to allow launching within the editor. Further, no error message informed users that it had not been implemented and no exception was thrown, leading users wondering what was wrong. This problem was fixed by changing LPARSE and SMOBELS to use the new framework launch shortcut as discussed in Section 5.4.3, page 46.

Other problems encountered were problems when the program location of LPARSE and/or SMOBELS were not set. The program is launched as an external process so the file path to that program has to be specified. This is set within the preferences inside APE. If it is not set then APE cannot launch the program. However again, no error message is given if the program path has not been set and the program is launched leading to APE hanging.

When inspecting the code it was clear why no error message was given. Taking the case of LPARSE, the code had been split into two projects, `uk.ac.bath.cs.asp.ide.lparse` and `uk.ac.bath.cs.asp.ide.lparse.ui` corresponding to the core and user interface components of LPARSE. Informing the user that the program path has not been set is only applicable when the program is launched. However the launch delegate which launches the program is located within the core plug-in, meaning there is no access to the user interface components. As a result no error message can be presented to the user, at least not in a nice graphical manner.

To solve this the changes to the structure as documented in Section 4.1.2, page 31 were made. As `uk.ac.bath.cs.asp.ide.lparse` will have the correct user interface dependencies a message box is now presented to the user whenever the program path has not been set.

Other improvements made to APE were to create a specific project and file creation wizard

---

<sup>3</sup><https://github.com/>

<sup>4</sup>Repository can be found at `git@github.com:robibbotson/APE.git`

for AnsProlog\* projects. Previously a general project and text file had to be created for any development within APE. This was confusing to users as time was spent looking for the APE wizard which did not exist. Therefore, one of the first things implemented for this dissertation was to implement the wizard as can be seen in Appendices E.1, page 88 and E.2, page 89. The new file wizard automatically appends a ‘.lp’ extension to the filename if one has not been specified by the user. By having these wizards it ensures greater clarity for users of APE.

### 5.3 Installation

In order to facilitate easy installation for users, an update site was created. This allows the user to connect to a website<sup>5</sup> within Eclipse to install the plug-in. An example showing the installation wizard can be seen in Appendix E.3, page 90.

Previously, installation required the user to do the majority of the work. The user was responsible for placing the correct Java ARchives (JARs) into Eclipse’s plug-in folder. This method is tiresome and can lead to errors if done incorrectly. With the new method the chances of incorrect installation of APE are reduced. Future updates are easily installed. Once installed Eclipse periodically checks the update sites of installed plug-ins to see if a new version is available. If one is the user is informed, enabling them to install the updated version.

The update site stores a `site.xml` file which tells Eclipse about the feature being installed. The update site also has a `features` and `plugins` which stores the JAR files. These are copied across to the user’s computer during the installation.

### 5.4 Framework

This dissertation did not set out to add every possible ASP tool into APE. Instead it aimed to differentiate from Sureshkumar’s work by providing an architecture for future developers to add tools and programs quickly.

To do this the common aspects from the programs currently implemented within APE were analysed. As each plug-in implements extension points, the common extension points used in the implementation of LPARSE and SMODELS were inspected.

It was found both plug-ins implemented the following:

**org.eclipse.ui.preferencePages** The preference page allows the user to specify any specific preferences for the program. One preference that is necessary in all programs is the program location path, enabling the external tool to be launched within APE.

---

<sup>5</sup>Website can currently be found at <http://ape.robibbotson.co.uk/updates>

**org.eclipse.debug.core.launchConfigurationTypes** When a program launches it has a launch configuration type. This has to be unique as it acts as an id for that programs launch. In addition a delegate is specified that performs the launch for the specific launch type.

**org.eclipse.debug.ui.launchConfigurationTabGroups** For a program to be launched the configuration has to be specified by the user. This extension point allows the program configuration to be specified within the `Run Configurations...` dialog, found within Eclipse.

**org.eclipse.debug.ui.launchConfigurationTypeImages** For the launch configuration type an image can be associated. This is shown in the `Run Configurations...` dialog as well as when the program is launched through a shortcut.

**org.eclipse.debug.ui.launchShortcuts** A launch shortcut allows the program to be launched within the editor by right clicking. Alternatively the file can be right clicked in the navigator pane and launched.

The above extensions are what a typical external program project implements within APE. It was decided to leave the preference page and image out of the framework due to being extremely simple to implement. Instead the other three extensions were concentrated on to produce a framework that allows for simple implementation.

### 5.4.1 Launch Configuration Tab Group

#### Background

The external program is typically ran in APE through the `Run Configurations...` interface. In this dialog the user picks the external program that they wish to run. The user is then presented with a range of tabs which can be clicked through, building the launch configuration. All the options picked on the tabs make up the launch configuration. The process of building the launch configuration from the tabs can be seen in Figure 5.1. The launch plug-in tab group implements the Eclipse extension point. In the tab group instances of all the tabs are constructed. The tab group is the holder for all of them defining the name that the user sees in the `Run Configurations...` dialog. When the user clicks 'Run' then this launch configuration is passed to the delegate which builds up a command line before launching the program in a separate process.

Common tabs present that are found in all APE external programs are the `input` and `common` tabs. The input tab allows the user to pick the files that the program will run on. The common tab meanwhile allows the user to specify common Eclipse options. For example standard output can be redirected, as well as choosing whether the launch should be a foreground or background process. The program can also implement its own tabs if necessary. A program may allow users to specify its arguments on a tab. For example Gringo has the argument '-ifixed' allowing the user to fix the number of incremental steps

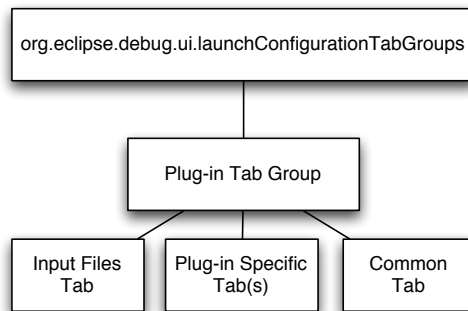


Figure 5.1: Tab Group Structure

to a number. On the tab page a checkbox will allow the user to choose that option before specifying a number for the number of incremental steps.

### Implementation

The scenario set out above is common with every external program having program arguments that the user can specify. Each argument can be thought as having two components. Firstly there is the actual option that the program understands followed by the human readable description, describing what the option does.

In the case of APE the arguments of a program are used in two places: the delegate and the tab which allows the user to specify the arguments. Previously in the cases of LPARSE and SMOBELS a list of arguments were hardcoded in both the delegate and tab. If any were added or changed in future then it would have had to be done in two places; not very maintainable code.

The framework set out to solve this problem. It was decided to allow the user to specify a list of program arguments in one file. From this the tab group and delegate would be produced. If any were added or changed in future then updating the one list would be reflected in both the tab interface and delegate.

For each program argument the user may choose to enable the option. This is represented in the user interface tab as a checkbox. Specifically in the code it is a `Button` with type of `SWT.Check`. However dependencies occur with arguments, for example if the user enables a option they may need to enter a number or type some text to be more specific. The framework handles this possibility allowing dependencies to be added and still automatically generated for the user interface.

To understand the whole process the algorithm first needs to be defined:

1. Use reflection to obtain all public fields in the argument list class



2. For each field retrieve the annotation checking it is of type `CommandLineSwitch`
3. Get the type of the field which should be of type `Button`
4. Generate the control from the type with the label being the description and option obtained from the annotation
5. Check the annotation to see if the additional flag is true
6. If true obtain the additional fields stored in `AbstractArgumentConstants`
7. For the additional field generate the appropriate control based on the type, placing it next to the control created in step 4.

From the algorithm it can be seen that two new concepts have been introduced into the framework, the annotation `CommandLineSwitch` (see Appendix H.1.6, page 113) and the abstract class `AbstractArgumentConstants` (see Appendix H.1.1, page 102).

The annotation is used on all the fields declared in the argument list class. Each field is annotated with the annotation so the option, description and additional flag is known when the fields are obtained through reflection.

Reflection being the ability to modify or examine programs running in the Java virtual machine. As the user interface is being generated at runtime this method has to be used. Bloch [Blo08] explains the problems of using reflection; compile time checking is eliminated as well as the code for it being clumsy and verbose. The Java tutorial page [Sun10] explains a further disadvantage is the increase in performance overhead. Bloch [Blo08] explains that ‘you can obtain many of the benefits of reflection while incurring few of its costs by using it only in a very limited form.’ Creating instances of the class via reflection and then accessing them normally using their interface or superclass is the method explained. This is exactly what occurs in the framework. The interface is the annotation `CommandLineSwitch` which provides access to the objects. From that compile time checking can occur as the annotation is the object being referred to. Due to this it was decided that reflection was an appropriate form to use for the project, especially considering the disadvantages are reduced by following the advice set out by Bloch.

The type of the field is used to know what control the interface should generate. Typically it will be a `Button` creating a checkbox for each option. However the type is more critical in the additional dependent fields. As the user may be needed to enter a number or text the type could be `Spinner` or `Text`. However the additional dependent fields are not declared public. Instead they are private members. However a mapping takes place linking the public field with its additional members. This is managed by the `AbstractArgumentConstants` class. By the argument list file extending this class a mapping is provided allowing mapping of fields. The additional flag on the annotation tells the interface when generating the code to look in the map to generate the additional controls.

Listing 5.1: Example Gringo Argument Annotation

```

@CommandLineSwitch(option = "--ifixed", description = "Fix number of
    incremental steps to <num>", additional = true)
public static final Button ATTR_GRINGO_FIX_NUMBER = null;

@SuppressWarnings("unused")
private static final Spinner ATTR_GRINGO_FIX_NUMBER_NUM = null;

```

A full example can be seen in Appendix H.2.1, page 114. However a short code snippet showing the key fundamentals is shown below:

The result of the generation for the whole of the Gringo arguments can be seen in Figure 5.2.

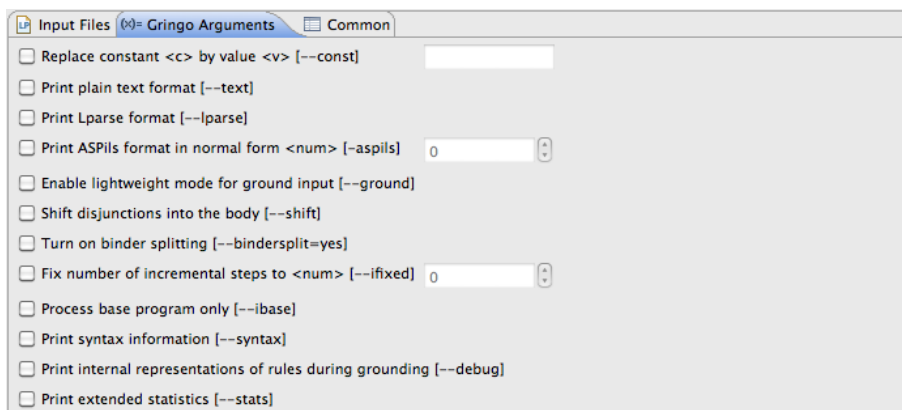


Figure 5.2: Single File Arguments

The generation code can be found in `AbstractSingleFileArgumentsTab` (see Appendix H.1.5, page 109). This is the class that performs the algorithm set out above producing the result seen in Figure 5.2 from a list of fields with annotations. For programs with many arguments the above method was not suitable. Although it will generate all arguments fine it would produce a huge list that would require a lot of scrolling for the user. It was deemed that this was not a viable option to the user so an additional abstract class `AbstractMultipleFileArgumentsTab` (see Appendix H.1.4, page 108) was created. This extends from the `AbstractSingleFileArgumentsTab` class but allows the user to specify multiple argument input files. Each files arguments are generated into collapsable panels, one for each file input meaning that it is much more user friendly to the user. An example of this in action can be seen in Figure 5.3.

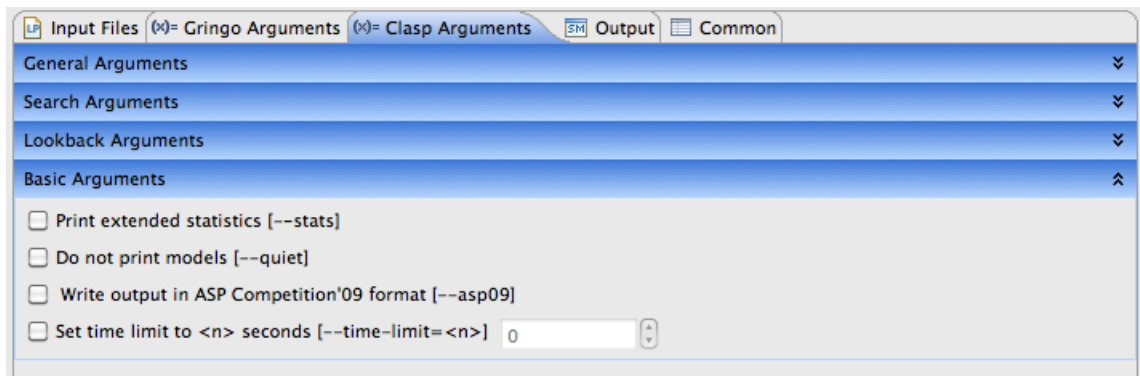


Figure 5.3: Multiple File Arguments

## Summary

A launch configuration tab is the main way a user will launch a program. A framework has been developed allowing users to specify a list of program arguments with each one annotated with the annotation `CommandLineSwitch`. Dependencies between arguments which require more information from the user are managed by `AbstractArgumentConstants` as the argument list class extends from it.

The launch configuration consists of a tab group managing multiple tabs. Two of the tabs will always nearly be the `input` and `common` tabs. However for the external program argument tab the tab can extend from `AbstractSingleFileArgumentsTab` or `AbstractMultipleFileArgumentsTab`. The user can then specify the argument list file with the whole user interface being generated at runtime. It saves having to write hundreds of lines of code managing the layout of controls as the framework generates it all at runtime for the user. As a result very little code has to be written to generate the argument tab. An example is the `GringoArgumentsTab` (see Appendix H.2.2, page 115).

### 5.4.2 Launch Configuration Type

Each launch configuration has an associated launch delegate that facilitates the program running. When the users runs the program the launch configuration is passed to the delegate which constructs the correct command line from the launch configuration as defined in the launch configuration tab.

For the delegate to be built correctly it must be aware of the elements in the launch configuration and what they correspond to. To share information between the tab and the delegate a new abstract class `AbstractLaunchConfigurationDelegate` (see Appendix H.1.2, page 103) is provided. The developer can extend from this passing in the argument list file. When the launch configuration is passed from the tab, the delegate knows all about the arguments, obtaining the list of possibilities through reflection again. Dependencies are

figured out with the command line having the correct information following the additional flag on the annotation.

The framework allows a much cleaner extendable interface with the argument list being reused. By using the `AbstractLaunchConfigurationDelegate` in combination with `AbstractArgumentConstants` and `CommandLineSwitch` the programmer can significantly cut down the amount of code needed to implement the tool into APE.

### 5.4.3 Launch Shortcuts

A launch shortcut allows users to launch the program quicker than going through the `Run Configurations...` dialog. When a shortcut is clicked a default launch configuration is created or re-used if it already exists. Two methods of launching are possible:

1. Right clicking in the editor and launching from the context menu
2. Finding the file you wish to launch in the navigator, right clicking and launching from there

Ultimately both methods follow the same code path but the information at the start is gathered slightly differently, such as how to know which file it needs to be launched for.

Between tools the process is exactly the same with every program launching in the same manner, that is passing it to its delegate. The framework allows developers to extend `AbstractLaunchShortcut` (see Appendix H.1.3, page 106). It means that a launch shortcut for each program can be implemented with one line in the constructor of the class. This line calling the super class with the configuration type id. An example in action can be seen in `GringoLaunchShortcut` (see Appendix H.2.5, page 118).

## 5.5 Gringo and Clasp

A requirement of the project was the addition of the grounder Gringo and solver clasp. Both of these tools were added into APE making use of the framework above. Each program had a new plug-in project created `uk.ac.bath.cs.asp.ide.gringo` and `uk.ac.bath.cs.asp.ide.clasp`.

### 5.5.1 Gringo Editor Problems

A main feature of any IDE is syntax checking as programs are typed. Any errors that are found are highlighted or underlined indicating to the user where the error lies. This process is called syntax analysis. This is carried out by parsing the program. The input file is broken up into a stream of tokens (lexical analysis) with the parser trying to match these tokens against the program specification known as the grammar of the program.

In order to provide syntax checking and error highlighting within the IDE this process has to occur whilst the user is typing their program. In an Eclipse editor this process is managed by a reconciler. A reconciler is a background process that updates the underlying model of the program. It can be activated after every key stroke, when the user stops typing or when the user saves the file depending upon the performance implications. When developing a LPARSE file the reconciler is working in the background updating the underlying model, enabling errors to be indicated to the user whilst they are typing.

The LPARSE parser is written for the tool Yet another compiler-compiler (Yacc). ‘Yacc provides a general tool for describing the input to a computer program’ [Joh]. Yacc is a parser that calls the lexical analyser to compare against the rules (the grammar). LPARSE uses the lexical analyser tool, Lex to provide the input token stream. As LPARSE is licensed under the GPL, Sureshkumar was able to extract the grammar from the source code of LPARSE.

Both Yacc and Lex output their programs in C code. However programs exist to perform the same functionality but output in Java code instead. Sureshkumar made use of the tool JFlex<sup>6</sup> to do this. This allowed an equivalent Java parser to be produced that is now used within APE. A Java object model is built up as the user types their program allowing syntax error checking to be performed.

For this project it was hoped that a similar approach could be adopted to provide the same functionality, however this time for the grounder Gringo. However the parser for Gringo is not written in Yacc or Lex. Instead it is a C++ program that uses the r2c parser<sup>7</sup>. This made the same approach used for LPARSE unsuitable. One approach to solve it would be to learn the Java Native Interface (JNI) so that the Java Virtual Machine (JVM) could interact with the Gringo code. ‘JNI allows Java applications to call native methods’ [Blo08]. Bloch states that the JNI approach has serious disadvantages as shown below:

- JNI is platform specific
- Native languages are not safe, exposed to memory corruption
- Decreases performance as there is a fixed cost of going into and out of native code
- Require ‘glue’ code, that is difficult to read and tedious to write
- One small bug can corrupt the entire application

Due to these problems outlined it was deemed that using the JNI would not be an appropriate approach for the project. Further it was decided that the time spent converting the r2c grammar into a compatible yacc one would be better spent implementing other features of the project. Instead it was decided that it would be suitable for Gringo to make use of the LPARSE editor as Gringo is just an extension on the LPARSE language. Although there are

---

<sup>6</sup><http://jflex.de/>

<sup>7</sup><http://www.rforge.net/r2c/>

incompatibilities, conversion between the two can take place (see Section 5.6, page 48). In addition an option was added into the `Source` menu so that the user could disable syntax checking if they knew the program was correct. Although not an ideal solution it was the one considered viable given the time constraints and the time that would be taken learning C++ (r2c) or JNI.

## 5.6 Conversion

One of the main features requested was for conversion between different solvers. Two input languages are now present in APE. It made sense to give users the option to convert between them especially given the syntax highlighting problem outlined above. Differences between the languages have been taken from the Gringo user manual [GKK<sup>+</sup>] as discussed in Section 2.6, page 16.

The differences are not a definitive list as stated in the Gringo manual [GKK<sup>+</sup>]. However they are the ones well known about, meaning that it should be suitable for this project to convert the more common cases. As future work is done on APE further conversions could be added, as the differences between the languages become better known.

The user can choose for the conversion to occur in two ways whilst viewing the document in the editor:

**Source Menu** Both conversion methods are available in the `Source` menu as can be seen in Figure 5.4. This is in the same menu as being able to toggle comments for the editor. It was decided to locate the conversion method options in the same menu to ensure consistency for the user. Any options they can perform in the editor can be performed from the `Source` menu.

**Keyboard Shortcut** Alternatively keyboard shortcuts can be used to convert the program. The specific shortcuts can be seen by the user to the right of the options in the `Source` menu as seen in Figure 5.4. The shortcut to convert from Gringo to LPARSE is `Control/Command + Shift + S`. Originally the shortcut was designed to be `Control/Command + Shift + L` to stand for converting to LPARSE. However it was found that it was already defined in Eclipse. Therefore `Control/Command + Shift + S` to stand for converting to SMOBELS the solver associated with LPARSE. Likewise a shortcut exists to convert from LPARSE to Gringo. This shortcut is `Control/Command + Shift + G` with the same principle being applied, of converting to a Gringo program.

### 5.6.1 Process

An abstract class `AbstractProgramConverter` (see Appendix H.3.1, page 120) was created in order to provide the core functionality when converting between the programs. This class

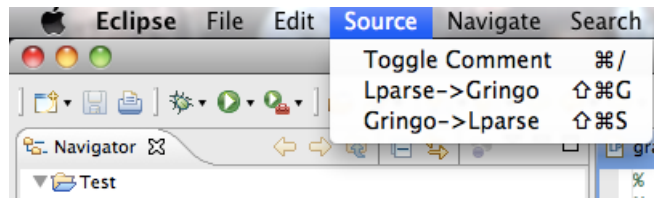


Figure 5.4: Conversion Options

provides methods so that each line in the editor's document can be identified. Additionally methods exist so that text can be replaced as well as providing a way to get a regular expression matcher for a given `String`. The conversions implemented work in all cases implemented by using regular expressions to check for matches that need converting. The process is a cycle with the regular expression being matched, the replacement being added and so on until no more matches can take place. Performing the conversion one by one means that any convoluted matches can be found with the replacement being inserted into the correct place. Performing all conversions at once would not work. When a conversion takes place the location in the document for the converted to be reinserted is known. However if other conversions took place then the place in the document would change. However the converter does not know this and so would insert it into the wrong location.

### 5.6.2 Gringo to Lparse

Two converters have been implemented to convert from Gringo to LPARSE. These are `RemoveRestrictedNames` and `PoolingGringoToLparseConverter`. Both of these extend `AbstractProgramConverter` using regular expression matching to find lines that need converting within the editor.

#### Remove Restricted Names

Symbolic names such as **and**, **plus** and **eq** are not valid LPARSE syntax. The converter has a list of restricted names that are not valid in LPARSE. This list is iterated through with each item being checked to see any are present within the document. This is done using the regular expression `"\\s name [ \\(\\.]"` where name is the restricted name from the list.

The regular expression specified checks for the restricted name and checks that it is not a substring of some other word by checking for whitespace before and after it. Upon finding a match an underscore is appended to the start of the word. The matching process then continues to find anymore matches of the name in the editor. When no more matches can be found the next item in the restricted name list is checked until the end of the list is reached and no more matches have been found.

## Pooling Converter

Having a pooled term such as `p(X,Y ; X,Z)` expands to `p(X,Y,Z)`, `p(X,X,Z)` in Gringo. However for LPARSE the same term would stand for `p(X,Y)`, `p(X,Z)`. As this conversion is going from Gringo to LPARSE the conversion finds the pooled terms within the editor document and expands them into the format of `p(X,Y,Z)`, `p(X,X,Z)`. Expanding the terms like this would mean the same output would be given whether running through the solver clasp or SMOELS.

To identify pooled terms the regular expression below is used.

```
[a-zA-Z]+\(\( )*[A-Z]+( )*([,;]( )*[A-Z]+( )*)+\)
```

This checks for the following:

1. At least one letter, followed by an opening bracket
2. Possible spaces, followed by a variable (capital letter), followed by possible spaces
3. Comma or semi-colon followed by possible spaces
4. Another variable, followed by spaces
5. Item 3 and 4 has to occur at least once
6. Closing bracket

The converter locates a match in the document using the regular expression. Each expanded term is then inserted into the document with the whole original line it was on in order to keep the context. For example `p(X,Y; X,Z) :- male(X), female(Y;Z)` is initially expanded to:

```
p(X,Z,Y) :- male(X), female(Y;Z).  
p(X,Z,X) :- male(X), female(Y;Z).
```

However, as this contains multiple pooled terms the process carries on until there are no more matches. After all matches have been found and expanded the following is left by the converter:

```
p(X,Z,Y) :- male(X), female(Y).  
p(X,Z,Y) :- male(X), female(Z).  
p(X,Z,X) :- male(X), female(Y).  
p(X,Z,X) :- male(X), female(Z).
```



### 5.6.3 Lparse to Gringo

Two converters have also been implemented for the conversion of LPARSE to Gringo. These are `AggregateLparseToGringoConverter` and `PoolingLparseToGringoConverter`. Again both extend from `AbstractProgramConverter` using regular expression matching to find lines that need converting within the editor.

#### Aggregate

LPARSE treats  $2p(c), p(c)$  as  $2[p(c)=1, p(c)=1]$  whereas Gringo treats it as  $2p(c)$ . The same process as pooling occurs with expansion occurring so both solvers understand it.

The regular expression used to find matches is specified below.

```
\\d+[{}]( )*\\w+( )*\\((\\w+\\)( )*(, ( )*\\w+( )*\\((\\w+\\)( )*[]]
```

This checks for the following:

1. A digit followed by { followed by optional space
2. A word followed by optional space followed by an opening bracket
3. A word followed by a closing bracket and optional space
4. Optionally, a comma followed by the same as steps 2 + 3
5. It finishes with }

#### Pooling Converter

As explained earlier (see Section 5.6.2, page 50) LPARSE expands a term such as  $p(X, Y ; X, Z)$  to  $p(X, Y), p(X, Z)$ . Gringo on the other hand treats it as  $p(X, Y, Z), p(X, X, Z)$ . As this conversion is going from LPARSE to Gringo the converter expands the term as LPARSE would. The regular expression used to identify matches is:

```
[a-zA-Z]+\\(( )*[A-Z]+( )*(, ( )*[A-Z]+( )*)*(; ( )*[A-Z]+( )*(, ( )*[A-Z]+( )*)*)+\\)
```

This checks for the following:

1. At least one letter followed by an opening bracket
2. Possible spaces followed by a variable (capital letter) followed by possible spaces
3. Possible comma followed by possible spaces followed by variable followed with spaces

4. Semi colon followed by spaces followed by variable followed by spaces
5. Same as step 3
6. Closing bracket

Using the same example as before (see Section 5.6.2, page 50) the line `p(X,Y; X,Z) :- male(X), female(Y;Z)` would be expanded to the following using this converter:

```
p(X,Y) :- male(X), female(Y).
p(X,Y) :- male(X), female(Z).
p(X,Z) :- male(X), female(Y).
p(X,Z) :- male(X), female(Z).
```

## 5.7 Auto Predicate Completion

A feature requested by users was the ability to have suggestions presented to them whilst typing their program. Using a content assister would allow for quicker development. The Eclipse plug-in developer guide [Ecl09] describes the implementation of content assist as “allowing context sensitive completion upon the user request.” Popup windows show the user the possible choices to complete a phrase with the user then able to select a choice for insertion into the document. Contextual popups provide the user with proposals relevant to the current document cursor position.

For this project it was decided that contextual popups would be implemented to provide the user with relevant choices to the position in the document. To do this a content assist processor `LparseContentAssistProcessor` was added to the LPARSE editor. Suggestions are presented to LPARSE and Gringo programs as both make use of the LPARSE editor. To keep it consistent with other development tools within Eclipse the keyboard shortcut (Ctrl + space) was assigned to activate the content assistant. Whenever the user wishes for proposals the user can press the shortcut to be presented with suggested options. Keeping the same shortcut allows developers with experience in other tools to be able to apply them within APE.

As discussed in Section 2.2.1, page 5, ASP programs can consist of different components: atoms, variables, constants and rules. Each of these have a specific content generator responsible for generating the `ICompletionProposal` that makes up what the user sees.

Contextual proposing is used meaning proposals are based on the last word typed by the user. Only proposals starting with the last word typed are presented to the user. If the user has not typed any word then all proposals are shown to the user. Two forms of proposal can be presented:

**The last word typed is based on previous predicate names** Proposals presented should be previous predicate names with possible constant and variable options.

**The last word typed is the new predicate name** Proposals presented should be the last word typed with possible constant and variable options.

An example of this can be seen in Figure 5.5. Previous predicates are suggested first in the suggestions as it is the most likely reason for users using predicate completion.

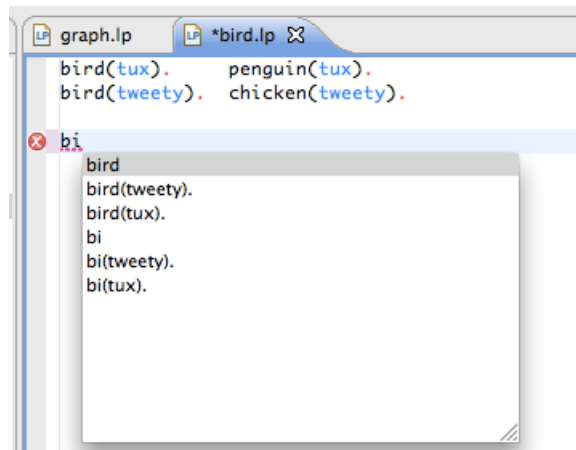


Figure 5.5: Predicate Completion Ordering

As mentioned above suggestions do not just consist of predicate names. Instead possible constant declarations or variable rules are suggested as well. This can visually be seen in Figure 5.6(a). Proposals are suggested in the following order:

**Predicate name** The first proposal is the the predicate name

**Numeric constants** Numeric proposals consist of the predicate name followed by a number in brackets (e.g. `age(Rob, 2)`)

**Symbolic constants** Symbolic proposals consist of the predicate name followed by a word in brackets (e.g. `bird(tweety)` in Figure 5.6(a))

**Variable proposal** Variable proposals consist of two forms. Variables can be defined in the head or body of a rule. If in the head of the rule then the suggestion should include `:-` whilst the body should end with a possible `.`. Examples of both types of proposals can be seen in Figure 5.6(a) and 5.6(b).

### 5.7.1 Predicate View

To further aid predicate completion for the user it was decided to implement a predicate view that represents the current document in the editor. A view is a collapsible component in the Eclipse workbench that presents information to the user.

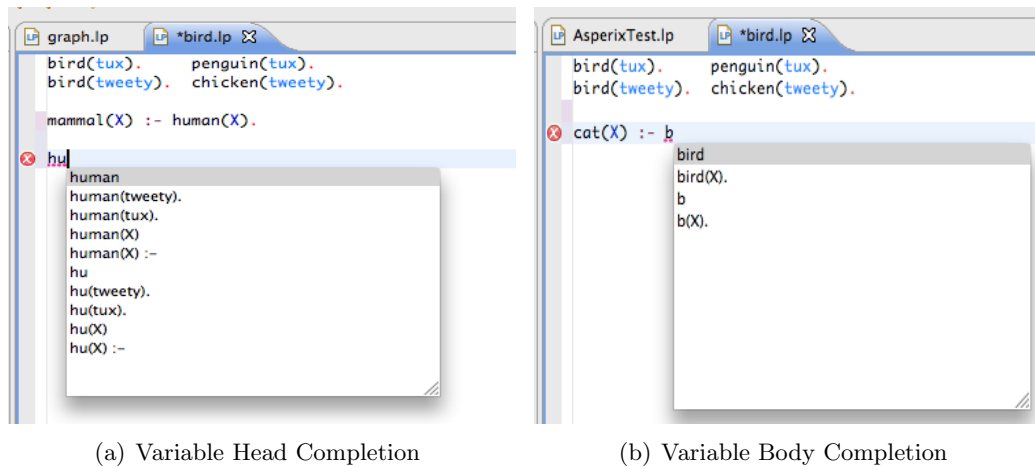


Figure 5.6: Predicate Completion Proposals

The view implemented, followed the model, view, controller (MVC) design pattern. The model represents the state of the data being presented to the user. The view presents this model to the user graphically with the controller providing the interaction between the two. For the predicate view the model was an array of `PredicateViewItem` objects with the view being a standard SWT `TreeView` control. The controlling actions are performed in the `PredicateView` class (see Appendix H.4.8, page 137).

Like the content assist processor the view presents suggestions based on the last word typed by the user. It shows all the previously defined predicates as well as suggestions from the last word typed. The suggestions presented use the same `IProposalGenerator` generators to give suggestions. If the user wishes to add a suggestion to the editor then they can double click on an item in the tree which will insert it into the document at the last location.

An example of the view can be seen in Figure 5.7. This corresponds to the same file shown in Figure 5.5. The comparison between the two is clear.

Again the view shows both forms of predicate proposal as explained in Section 5.7, page 52. However various options and filters are presented to the user to allow more fine grain control compared to the content assist. The options presently available can be seen in Figure 5.8.

**Do not suggest from previous predicates** As the name suggests any previous predicates are not shown in the view. Only suggestions from what the user has last typed are.

**Do not show from current** Only suggestions from previous predicates are suggested. No suggestions are presented based on the word last typed by the user.

**Auto-expand** Like the name suggests, auto expansion of the tree can be controlled. By

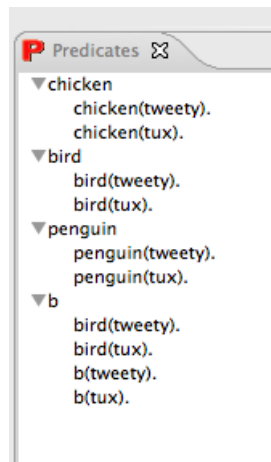


Figure 5.7: Predicate View

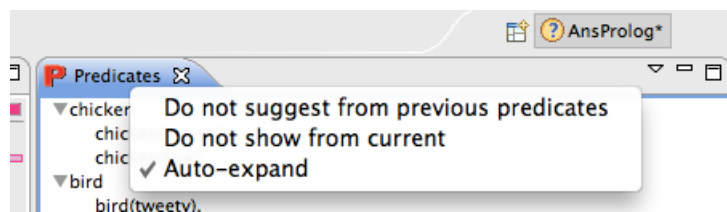


Figure 5.8: Predicate View Options

default it is turned on. However the user can turn it off and any future items added to the tree will not be expanded. Instead just the predicate name will be visible with the option to expand to see all possible suggestions for that predicate.

## 5.8 Summary

This chapter has documented the features and improvements made to APE throughout the course of the dissertation. Improvements have been made with a new file and project wizard being created to be more helpful to users. Additional improvements have been made with the existing tools inside APE being fixed so they can be launched through a shortcut. Further, error reporting has been improved with refactoring taking place to ensure that the right classes can access the user interface components.

New features have been added with conversion between program languages now available. This will save the user having to find the incompatibilities themselves and spend time rewriting their program. Predicate completion is now available to enable faster development on programs for users. This can be activated through the new view or through the

traditional Eclipse content assister. Additionally a new framework has been developed, which new programs being implemented into APE can subclass. The framework minimises the amount of coding needed by the developer to add the program into APE. This has been taken advantage of with the programs Gringo and clasp added making use of this.

Further, the project has added a automated testing strategy to minimise the chance of errors occurring as APE is developed in the future. This is fully documented in the next chapter.

## Chapter 6

# Testing

Part of any software project is the testing stage. Clayberg and Rubel [CR08] state that tests are necessary to ensure that a product continues to function correctly over multiple releases. Sureshkumar's testing was limited to manually testing the system which Clayberg and Rubel [CR08] say is acceptable as long as it is just one release. However as APE is being extended as well as being developed in the future with many possible releases, then the advice given by Clayberg and Rubel in [CR08] of having automated and manual tests being in place to prevent regressions was headed. Indeed it was decided that a robust testing policy should be in place, requiring a suite of tests to be run before a check-in is approved into the repository in the future. This chapter outlines the tests in place to identify errors that may be present within APE. If errors were found whilst testing they were debugged and fixed, ensuring that the test passes.

In the case of this project it was decided to have two strands of testing:

**System testing** Automated and manual tests of the system comprising of unit and regression tests following the advice set out by Clayberg and Rubel.

**Requirements** Test that the requirements as set out in section have been implemented correctly.

### 6.1 Test Environment

#### 6.1.1 Automatic Tests

As discussed previously in Section 2.7, page 16 and Section 4.5.1, page 36 automatic testing will be handled by SWTBot. SWTBot provides APIs that hide the complexity involved with SWT and Eclipse. SWTBot tests run in the UI thread simulating the actions of users. Tests for SWTBot have the same annotation features as JUnit4<sup>1</sup>. Tests to be ran

---

<sup>1</sup><http://www.junit.org/>

are annotated with the annotation `@Test`. Static setup and tear down can be performed with the annotations `@BeforeClass` and `@AfterClass`. For specific tests, setup and tear down can be done with the annotations `@Before` and `@After`.

Tests have no guaranteed order of execution resulting in tests being independent of one another. If tests depended upon one another it would cause a catalog of issues when the execution order changes. A benefit of this approach is that each test can be isolated when failing, being reran when fixed.

### 6.1.2 Manual Tests

Whilst developing features for APE the plug-in development environment (PDE) allowed individual aspects of APE to be tested. A new instance of Eclipse can be launched along with the plug-in under development, simulating the configuration of a normal Eclipse installation. This configuration includes the Java Runtime Environment as well as the set of other plug-ins available to the installation. In addition the Eclipse instance launched through the PDE can be started in debug mode, allowing defects to be easily located.

## 6.2 Location

A decision had to be made about where the tests within APE were to be stored. A couple of options were available:

**Existing projects** One option was to store the tests under each project in APE. For example tests for LPARSE would live in `uk.ac.bath.cs.asp.ide.lparse`

**New project** Another option was to create a separate plug-in project called `uk.ac.bath.cs.asp.ide.testing` to hold tests for all the projects within APE.

It was decided the second option was the best course of action. By having all tests within one package it allows future developers to quickly locate them, as well as being able to run all tests in one click. This made more sense, the alternative would have been running through the tests in each project individually.

The structure of the new project can be seen in Figure 6.1.

SWTBot tests refer to the Java classes comprising of the tests for APE. However as can be seen the project also holds other key information. The `MANIFEST.MF` file is needed for the plug-in to interact correctly. Two folders, `results` and `tests` are also present. This is where the regression input and expected result files are stored, as explained in Section 6.3.2, page 61.



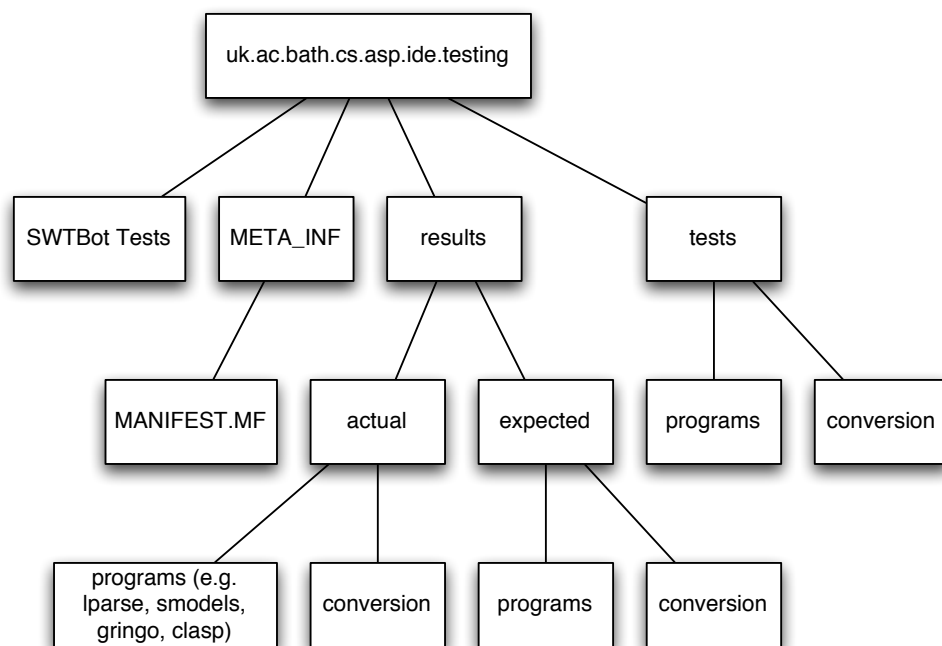


Figure 6.1: Testing Plug-in Structure

## 6.3 Automatic Testing

Automatic testing is performed by SWTBot. It runs from a users perspective opening a new version of Eclipse running through all the tests. A demonstration of the tests being run for APE can be seen in Appendix F.1.1, page 91.

### 6.3.1 Project Tests

Table 6.1 shows the tests devised to ensure that project creation and deletion works correctly. A key part of developing ASP programs is being able to store all development files within a project.

Test Case	Expected Result
Project creation	A new AnsProlog* project is created with the specified name using the <b>New</b> menu option.
Project deletion	The project is selected in the file navigator and the option to delete is clicked. The project is deleted successfully.

Table 6.1: Project Creation & Deletion Tests

### 6.3.2 External Program Testing

The main functionality of APE comes from the external programs that are able to be launched within APE. The four programs present in APE at the end of this dissertation project are:

- LPARSE
- SMODELS
- Gringo
- clasp

All of these programs share the same the functionality:

1. File creation and deletion for the program.
2. Can create and delete a launch configuration.
3. Can close the launch configuration.
4. Can run a launch configuration.
5. Can access all the tabs in the launch configuration dialog.

6. Can input dummy data into the launch configuration.

The test cases performed by SWTBot can be seen in Table 6.2.

<b>Test Case</b>	<b>Expected Result</b>
File creation	A new file is created through the <b>New</b> file menu wizard. The file is successfully created without a problem.
File deletion	The file created is highlighted in the file navigator view. The option to delete the file is clicked with the file being successfully removed from the system.
Program launch configuration	The <b>Run Configurations...</b> menu option is chosen. The specific program is selected in the list with the launch configuration options opening successfully.
Launch configuration closed	The same as 3, with the option to close the launch configuration clicked. The dialog box closes without error.
Launch configuration runs	The same as 3, with the option to run the launch configuration clicked. The program runs without error.
Launch configuration tab check	The same as 3. Each tab in the launch configuration is clicked, no errors occur when checking each one.
Launch configuration data check	The same as 6. When each tab is opened, each checkbox is checked and unchecked with numbers and text being entered in the relevant controls.

Table 6.2: Program Tests

Although the tests in Table 6.2 test the program functions correctly within APE, it does not test that the output produced is correct. To do this regression tests were added for each program ensuring the output produced is consistent across APE versions.

### Regression Tests

Regression tests currently exist for each program implemented within APE. These are LPARSE, SMODELS, Gringo and clasp. Each program has an associated set of test input files. These are stored under the `tests` directory (see Section 6.2, page 58) in a subfolder, specific to each program.

The expected results that the programs should output are stored in the `expected` folder under the `results` directory within the testing project. The program output when ran through APE can then be compared to the expected result to ensure it is correct. When the program is ran, the output from the program is redirected from standard output to a file. The file it is written to is stored under the `results` folder in a directory called `actual`.

The tests are ran automatically with SWTBot providing the functionality to do this. The tests are ran just like a user would run them with all the user interface steps being per-

formed.

The following algorithm takes place when running the tests:

1. For each test file in the test directory
2. Create a new AnsProlog\* project
3. Create a new AnsProlog\* input file
4. Open the editor for this file
5. Read the test into a buffer and copy into the editor
6. Save the editor
7. Open the `Run Configurations...` dialog
8. Create a new launch configuration setting up all necessary options for the program
9. Redirect output to the `actual` directory
10. Launch the program
11. Repeat the process for the other tests in the test directory

Test output for each test will be stored in the `actual` directory. The next stage performed is comparison between this output and the expected output in the `expected` directory. Files to be compared have the same name in both directories. Each file is read into a buffer line by line. Each line between the two files is compared. If they are identical the next line is read, if not then the output is not equal and therefore an error is identified. When all lines have been read without error for all tests it shows that the program is running within APE as expected.

### 6.3.3 Conversion Between Solvers

A feature added to APE was the ability to convert between the languages of LPARSE and Gringo. Again automatic tests using SWTBot were devised to ensure conversion can be tested. The same process as the regression tests above was used. Files to be converted are stored in a `conversion` folder under the `test` directory within the testing project. The expected output of these files when ran through the conversion process is stored in a `conversion` folder under the `expected` results folder.

The following algorithm then takes place:

1. For each conversion test in the test directory
2. Create a new AnsProlog\* project

3. Create a new `AnsProlog*` input file
4. Open the editor for this file
5. Read the test into a buffer and set the text in the editor to be this
6. Select the conversion method from the `Source` menu
7. Write the text in the editor to a file stored in the `actual` directory.

Following the tests the expected and actual output is compared to ensure that it is correct. If any differences are found then this will fail the test allowing the developer running the tests to look into the problem in more depth. Future tests can be added with the developer adding extra files in the `test` directory as well as the expected output in the `expected` directory.

### 6.3.4 Predicate View

The majority of the predicate tests have been performed manually due to time constraints limiting the amount of time to write tests. However a simple SWTBot test to check the view is present and opens in APE as can be seen in Table 6.3.

Test Case	Expected Result
Predicate view	The predicate view is selected. It opens and is visible.

Table 6.3: Predicate View Tests

## 6.4 Manual Testing

For this project it was not possible to use SWTBot to test every aspect of APE. This was down to limitations of SWTBot as well as time constraints limiting the time available to spend writing tests. Instead manual testing was used to complement the testing process.

### 6.4.1 Preference Dialog

Each external program that can be run within APE has its own preference page. On this page, the external program location is set so the program can be launched. However the preference dialog opening sequence is different on different operating systems. For Windows it is opened by clicking `Window->Preferences` whilst on Mac OS X it is opened by clicking `Eclipse->Preferences`. A decision was made to not spend time writing versions of tests for specific operating systems. Instead the preferences were tested manually with the test cases seen in Table 6.4.

<b>Test Case</b>	<b>Expected Result</b>
Program location set	The program is selected in the preference page. A location path is stored for the program without error.
Program location saved	The preference dialog is reopened. The path stored is the same one as saved from the test above.
Program run with path	A launch configuration is made and ran with the path being set in the preferences. The program runs without error.
Program run without path	A launch configuration is made and ran without the path being set in the preferences. A message dialog should inform the user that the program cannot run as no program location path has been set.
Syntax highlighting	Changes made to the syntax highlighting are reflected in the visible editor.

Table 6.4: Preference Tests

### 6.4.2 Platform Compatibility & Installation

Part of the reason for developing APE as an Eclipse plug-in originally was so it could be run on any platform due to running on Java. Although it should be cross-platform compatible, tests were still carried out ensuring no incompatibilities within APE were encountered.

<b>Test Case</b>	<b>Expected Result</b>
Installation	Installing from the update site causes no problems with all features of APE present
Uninstallation	APE is removed from the system correctly with no trace left

Table 6.5: Installation Tests

APE was installed through the update site onto the following operating systems:

- Windows XP
- Windows 7
- Mac OS X (10.6)
- Linux (Ubuntu 9.10)

Following the installation on each operating system the corresponding external programs were installed on the host system. After that the automatic test suite was ran as well as the

manual tests documented in this section. With no problems being encountered it provided assurance that APE is platform independent.

### 6.4.3 Predicate View & Predicate Auto Completion

A feature not automatically tested is predicate completion. No automatic tests were constructed due to time constraints within the project. However manual tests were still performed in order to make sure the feature worked as expected. When developing an ASP program two possible ways are presented to the user: through the editor and through the predicate view. Both forms were tested as seen in Table 6.6 and Table 6.7.

In order to test both of these features a common set up occurred. A new AnsProlog\* project was created with a new LPARSE file being created. Testing then occurred with the input files found in Appendix G, page 93.

<b>Test Case</b>	<b>Expected Result</b>
Ctrl+space pressed when cursor is positioned at end of Appendix G.2	The suggestions presented to the user should be a combination of all the previously defined predicates in the file. In addition suggestions should be made from the predicate has just typed.
Ctrl+space pressed when cursor is positioned at end of Appendix G.3	Suggestions suggested to the user should be all the relevant previous predicates defined in the file. No duplication should occur with suggestions from the predicate just typed by the user.
Ctrl+space pressed when cursor is positioned at end of Appendix G.4	Suggestions presented to the user should show the possibilities with the variable defined previously in the list.

Table 6.6: Predicate Completion Tests

The expected results of these tests can visually be seen in Appendix G.5, page 97.

<b>Test Case</b>	<b>Expected Result</b>
Predicate view, cursor at end of listings G.2, G.3 and G.4, filters off	All suggestions should be presented to the user. The suggestions should be expanded so the predicate name is the node of the tree with all the possibilities being children. The possibilities should include options from previous as well as predicates just typed. The suggestions should have all the previous defined constants and variables as possibilities.
Predicate view, cursor at end of listings G.2, G.3 and G.4, no auto-expansion filter on	The results should be the same as the first test. However the children in the tree view should not be visible. Instead it should just be the name of predicates.
Predicate view, cursor at end of listings G.2, G.3 and G.4, current predicate filter on	The suggestions should be the same as above. However no previously defined predicates should be suggested. Instead suggestions should be made for the predicate the user has just typed. The suggestions should suggest constants and variables.
Predicate view, cursor at end of listings G.2, G.3 and G.4, previous predicate filter on	Same as above apart from only previous predicate suggestions should be visible.
Double clicking	Double clicking on any item in the tree should insert the predicate into the editor at the cursor position.

Table 6.7: Predicate View Tests



## 6.5 Requirements

Gupta and Bhatia [GB10] describe the requirements testing process as “ensuring that the design and code fully meet those requirements.” The process is aimed at prevention rather than detection of defects. Gupta and Bhatia carry on to say that requirements are not often tested as they are ambiguous or incomplete. However in the case of this project it is felt that the requirements checking would be beneficial. In the following section the main requirements as seen in Section 3 are discussed describing how they have been checked for completeness with the tests in place.

### 6.5.1 Framework To Add New Features

One of the main features added into APE is the ability to quickly define an external program to be launched within APE. The new framework allows users to extend from a variety of abstract classes to implement the tool. If anything specific needs to be implemented then subclassing will allow it to be done.

A key benefit provided by the framework is the speed at which future tools will be able to be added into APE. In order to validate this claim then a test was performed seeing how long it takes for a new tool to be added into APE. The test run can be seen in Appendix F.2, page 92. The external program added into APE was ASPeRIX, a program that performs the grounding and solving in one step. As the results show in Appendix F.2 the whole process of adding the tool took under twenty minutes. The test added in the new tool, allowing the program’s location to be specified in the preference as well as being able to run from the `Run Configurations...` dialog. No shortcut or image was added however this is simple and would only take a few minutes for a good developer. The test showed how no complicated code was needed, with the framework handling all the GUI interaction and the process of launching the program. Of course this test is slightly unfair as it was performed by someone with knowledge of APE, however for a new developer the learning curve should not be too steep and should not cause a big rise in time taken.

### 6.5.2 Support for Gringo and Clasp

Both Gringo and Clasp have been implemented making use of the framework described above. The user now has the choice of developing their program for LPARSE/SMODELS or Gringo/clasp. Unit and regression tests are in place to ensure that the programs will continue to work in future versions of APE.

### 6.5.3 Auto Predicate Completion

Predicate completion has been implemented meaning that this requirement has been met. The user has the choice of using the inbuilt Eclipse content assister or using the new

predicate view which gives an outline of all the predicates defined in the program. Manual testing has been performed to ensure that these features work correctly. The tests and results can be seen in Appendix G, page 93.

#### **6.5.4 Conversion**

Conversion between the languages of LPARSE and Gringo has been added into APE. The process works both ways with tests in place to ensure regressions do not occur. This ability will allow the user to make use of convert into either language if they need to use a different solver.

#### **6.5.5 Regression Testing Ability**

This chapter has documented the various tests that have been performed following the development of APE. Unit tests as well as regression testing has been performed. This is automated through the tool SWTBot. A whole test run showing the speed that SWTBot runs at can be seen in Appendix F.1.1, page 91. If features were not able to be tested through SWTBot then they were tested manually.

Regression tests compare the output from previous versions of APE (expected output) to the output of the developer's build of APE. If all tests pass it means the output is correct, in the sense it is the same between versions of APE.

#### **6.5.6 User Friendly**

In order to achieve acceptance then APE has been designed to be as usable as possible. Most of the GUI design is handled by the Eclipse framework, however other features have been thought of that enable the user experience to be as pleasant as possible. Examples are the new file and project wizard for AnsProlog\* files and projects.

Additionally installation and updating of APE has been made easier for users. Users can now install from an update site which will tell the user when updates are available to be installed. This saves the user having to manually install by dragging JARs into specific folders on their computer. Further, usability has been improved by refactoring the projects. All have access to user interface components which means meaningful error messages can be presented to the user. An example of this is when a program location has not been set.

## Chapter 7

# Conclusions

The aim of the dissertation was to extend upon the work done by Sureshkumar. Research has been performed asking potential users for the features they desire. Additionally the project has aimed to make future development for APE easier. This chapter documents what has been achieved during the course of the dissertation, the significance of the work performed as well as any future improvements to the project identified.

### 7.1 Project Summary

The scope of the project was identified early on in the project. Given the limited time and resources available, the requirements elicitation process was limited to the distribution of an online questionnaire to potential users. This was used to collate the thoughts of the ASP community. With more resources available other requirements gathering processes could have been considered. For example having face to face consultations or demonstration systems would have provided a greater understanding of how APE could be used.

The response rate from the questionnaire was 37%. Although low, seventeen responses were received giving indication that enough people were interested to make the project worthwhile. These responses enabled a set of requirements to be drawn up as seen in Appendix C. Improvements that could be made to the previous version of APE were identified. Responses indicated that APE did not work due to new versions of Eclipse causing incompatibilities. Further, users who had heard of APE did not use it, as APE did not provide the features they needed.

APE is implemented as a set of plug-in projects for the Eclipse platform. This methodology allows APE to interact with other Eclipse plug-ins to use features such as source control and build script support. In addition as Eclipse is a Java program, the Java Virtual Machine (JVM) provides multi-platform support meaning APE can be run on any operating system where the JVM can be ran. The dissertation did not consider changing this implementation method as otherwise the project would have been repeating Sureshkumar's work albeit in

a different way. Instead the dissertation extended upon the previous work and provided more functionality into APE:

- New framework to facilitate easy adding of external programs into APE
  - Automatic program argument generation tab
  - Simple delegate generation for program arguments
  - Simple shortcut for launching the program
- Launching Gringo and clasp using the new framework
- Automatic unit and regression testing framework
- New AnsProlog\* file wizard
- Conversion between the language of LPARSE and Gringo
- Automatic predicate completion
  - Content assist within the LPARSE editor
  - Predicate view showing the program outline and predicated predicates
- Easy installation and updating with the new update site

One thing noted from the requirements gathering phase was the wide range of ASP programs and tools in use by a variety of users. Given the time constraints of the project it was infeasible to even consider implementing every program that may be useful to a potential user of APE. Instead it was decided to implement a framework so that future developers can quickly add a new program if there is enough demand. As the result in Appendix F.2, page 92 shows, a new program can be added and launched within APE in less than twenty minutes.

The previous version of APE suffered from not having a good test strategy when changes were made. Manual tests were ran to check the main features functioned correctly. Problems occurred when Eclipse was updated, with no specific tests in place the problem could not be quickly identified. Therefore a testing strategy was devised to perform automatic and manual testing, making use of SWTBot to provide this functionality. A common framework was devised so that future tests can extend and reuse common components.

The installation of APE has been made easier with potential users now able to install using the update manager located within Eclipse. This allows the user to specify a website that holds the plug-in. The installation is then performed automatically with the user not having to do anything other than specify the update site. Previously the user had to drag the JAR files into the correct directory in order for APE to be usable within Eclipse. The new process simplifies this greatly making it much more user friendly.

## 7.2 Limitations

Given the nature of the project not every feature implemented went smoothly. Conversion between solvers does not convert every incompatibility found between the languages of LPARSE and Gringo. The conversion methods are taken from the Gringo and clasp user manual [GKK<sup>+</sup>]. The manual [GKK<sup>+</sup>] itself says that not all possible incompatibilities between Gringo and LPARSE are known. APE does not give a guarantee that conversion will work in every possible case due to these unknown cases. However the more common conversions between pooled terms, aggregates and restricted variable names are converted successfully within APE.

Currently Gringo programs use the LPARSE editor. A underlying LPARSE model of the program being written is used, rather than a Gringo model. Problems with this could be incorrect syntax highlighting. Another issue could be syntax errors being identified at incorrect locations. Program errors could be shown when they shouldn't be, alternatively errors may not be shown when they should. Although the option is provided to the user to disable error checking, it does defeat the objective of the IDE. Alternatively the user can convert the syntax but problems exist with that as well, as discussed above.

This problem is caused by no suitable parser being implemented inside of APE for Gringo. Although the grammar is known for Gringo it would have meant converting the program from C++ into a Java compatible parser. This would have been a time consuming process. This was not considered an option given the very small chance of the problem occurring. If the user knows the program is correct then the syntax checking can be turned off.

## 7.3 Further Work

Further work could firstly be to look at the limitations outlined above. As more incompatibilities between LPARSE and Gringo are discovered then these could be added into APE. Additionally if any new grounder with a different input language is added to APE then this should be added as a new conversion method. One such example would be if DLV was added to APE then conversion between that and LPARSE could be performed. Once it is in LPARSE then it could be converted into Gringo compatible syntax. The idea is that there is one language that is the 'base' that can then be converted into any language.

Work could then be done writing a compatible Java parser for Gringo. At the moment any Gringo programs are using the LPARSE editor meaning that the program is being built into a LPARSE representation. Although the languages are similar differences do occur meaning that syntax errors could be missed or wrongly identified by using the wrong editor. This can be solved by writing a Java parser to allow parsing of Gringo programs whilst the user is typing.

Furthermore, work could be done improving the syntax error messaging presented to users. When a syntax error is detected within the editor the problem area is underlined in red.

When hovering over the area with their mouse the user is presented with a message informing them of the problem. Currently every error message says ‘syntax error’ which although indicating a problem at the location does not provide anything meaningful for the user. Error messages such as ‘unknown constant or missing rule head’ could be used to help the user identify problems quicker.

In this project two new external programs Gringo and clasp have been added. These programs make use of the new framework that has been implemented to facilitate easy integration of programs into APE. However the existing tools LPARSE and SMODELS were not fully migrated over to make use of the new framework. Aspects were such as being able to launch each program through the launch shortcut. However the delegate and program argument user interface code was not. Therefore further work could look at migrating this over to use the same methodology as Gringo and clasp. By doing this it would allow more maintainable code with each program using the same code path. With each program using the same implementation it would enable future developers to understand the code base which can only mean better designed code.

Further, the framework can be used to add any new ASP programs in the future. A key part of the IDE is the benefit of running external programs within it. The framework facilitates this implementation. Any new program can easily be added, to be run within APE. However the programs to be added should be driven from user feedback. Adding every possible tool could make the interface clunky and unusable. Instead the most desired programs should be added into APE. This feedback could be gathered from interview sessions, discussions and even more questionnaires. However interviews and observation sessions could be the most useful as they allow the most interaction between the interviewer and interviewee.

Testing has also been added into APE. Automatic testing is performed by the tool SWTBot. This runs as if it is the user clicking through all the options like the user would, in order to perform the test. These tests perform both unit and regression tests. This helps to identify individual components as well as checking between versions of APE. However, given the time constraints in the project not all features found within APE could have tests written for them. Instead manual tests complemented the automatic ones. Work could be done to ensure that the automatic tests are written for all features to give good test code coverage.

Work can also be done to change the way that SWTBot is ran. Currently tests are ran within Eclipse. However SWTBot has the feature so that it can be run from the command line, also known as headless mode. The tests could be ran like that rather than having to have a version of Eclipse open. Additionally this means that the tests could be scheduled. For example a server could be set up so that each night it pulls down the latest source code from the GIT repository. When that has completed the tests then run on the latest version. An email is then sent to interested parties to inform them of the result. This means that any bad code checked into the repository would be identified and fixed as soon as possible.

Future work could also look at the expansion of the framework to be more general purpose, perhaps a separate project. Currently it is embedded within APE, with it providing a mechanism for user interface code to be generated. The user specifies a list of arguments

that are automatically turned into user interface code at runtime through reflection. The framework handles the layout and positioning of these elements reducing the need for the developer to think about them. This could prove beneficial in other applications as it would reduce the time the developer has to spend developing an interface. The viability of this approach could be researched to see if the advantages could be justified in other applications other than APE.

## 7.4 Summary

This dissertation has succeeded in setting out what it aimed to do. APE has been extended providing an IDE that has the features most desired by the users. A viable IDE is now available for users to choose to develop their ASP programs within. Without APE, users would still have to use multiple programs to perform their development. Considerations have taken place to consider the usability through the use of an update site as well as providing error messages when preferences have been set incorrectly. Additionally if desired, external programs can quickly be added to APE. APE is now at the stage where it is ready to be used by the ASP community. Feedback gathered from these users will drive development in the future.

# Bibliography

- [Apt03] Krzysztof Apt. The logic programming paradigm and prolog. In John Mitchell, editor, *Concepts in Programming Languages*, pages 475–508, Cambridge, UK, 2003. Cambridge University Press.
- [Bar03] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK, 2003.
- [BD05] Martin Brain and Marina De Vos. Debugging logic programs under the answer set semantics. In Marina De Vos and Alessandro Provetti, editors, *Answer Set Programming: Advances in Theory and Implementation*, pages 142 – 152. Research Press International, 2005.
- [BGP<sup>+</sup>07] Martin Brain, Martin Gebser, Jörg Pührer, Torsten Schaub, Hans Tompits, and Stefan Woltran. Debugging ASP Programs by Means of ASP. In Chitta Baral, Gerhard Brewka, and John Schlipf, editors, *Proceedings of the 9th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR'07), Tempe, AZ, USA*, volume 4483 of *Lecture Notes in Artificial Intelligence*, pages 31–43. Springer, 2007.
- [Blo08] Joshua Bloch. *Effective Java (2nd Edition)*. Prentice Hall, 2008.
- [CDVBP08] Owen Cliffe, Marina De Vos, Martin Brain, and Julian Padget. Aspviz: Declarative visualisation and animation using answer set programming. In *ICLP '08: Proceedings of the 24th International Conference on Logic Programming*, pages 724–728, Berlin, Heidelberg, 2008. Springer-Verlag.
- [CR93] Alain Colmerauer and Philippe Roussel. The birth of prolog. In *HOPL-II: The second ACM SIGPLAN conference on History of programming languages*, pages 37–52, New York, NY, USA, 1993. ACM Press.
- [CR08] Eric Clayberg and Dan Rubel. *Eclipse Plug-ins (3rd Edition)*. Addison-Wesley Professional, 2008.
- [DMS84] Norman M. Delisle, David E. Menicosy, and Mayer D. Schwartz. Viewing a programming environment as a single tool. In *SDE 1: Proceedings of the first ACM SIGSOFT/SIGPLAN software engineering symposium on Practical*



- software development environments*, pages 49–56, New York, NY, USA, 1984. ACM Press.
- [DV09] Marina De Vos. Answer set programming lecture slides, December 2009. Available from <http://www.cs.bath.ac.uk/~jap/CM30174/Slides/asp.pdf>.
- [Ecl09] Eclipse. Platform plug-in developer guide, 2009. Available from <http://help.eclipse.org/help32/index.jsp>.
- [FGP<sup>+</sup>85] Nissim Francez, Shalom Goldenberg, Ron Y. Pinter, Michael Tiomkin, and Shalom Tsur. An environment for logic programming. In *Proceedings of the ACM SIGPLAN 85 symposium on Language issues in programming environments*, pages 179–190, New York, NY, USA, 1985. ACM Press.
- [FP] W. Faber and G. Pfeifer. DLV homepage. <http://www.dbai.tuwien.ac.at/proj/dlv/>.
- [GB10] Amit Gupta and Rajesh Bhatia. Testing functional requirements using b model specifications. *SIGSOFT Softw. Eng. Notes*, 35(2):1–7, 2010.
- [Gee05] David Geer. Eclipse becomes the dominant java ide. *Computer*, 38(7):16–18, 2005.
- [GG85] Michael R. Genesereth and Matthew L. Ginsberg. Logic programming. *Communications of the ACM*, 28(9):933–941, 1985.
- [GKK<sup>+</sup>] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Sven Thiele. A users guide to gringo, clasp, clingo, and iclingo. [http://sourceforge.net/projects/potassco/files/User\\_s%20Guide%20%28gringo\\_clasp\\_%2B%29/2008-11-09/guide.pdf/download](http://sourceforge.net/projects/potassco/files/User_s%20Guide%20%28gringo_clasp_%2B%29/2008-11-09/guide.pdf/download) Accessed on 09/12/2009.
- [GKO<sup>+</sup>09] Martin Gebser, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Sven Thiele. On the input language of asp grounder gringo. In *LPNMR '09: Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 502–508, Berlin, Heidelberg, 2009. Springer-Verlag.
- [GKS09] Martin Gebser, Benjamin Kaufmann, and Torsten Schaub. The conflict-driven answer set solver clasp. In *LPNMR '09: Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 509–514, Berlin, Heidelberg, 2009. Springer-Verlag.
- [GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the Fifth International Conference on Logic Programming*, pages 1070–1080, Cambridge, Massachusetts, 1988. The MIT Press.

- [GL93] Joseph Goguen and Charlotte Linde. Techniques for requirements elicitation. In *Requirements Engineering*, pages 152–164. IEEE, 1993.
- [GST07] Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In *LPNMR*, pages 266–271, 2007.
- [Joh] Stephen C. Johnson. Yacc - yet another compiler-compiler. Available from <http://dinosaur.compilertools.net/yacc/index.html>.
- [KO85] Henryk Jan Komorowski and Shigeo Omori. A model and an implementation of a logic programming environment. In *Proceedings of the ACM SIGPLAN 85 symposium on Language issues in programming environments*, pages 191–198, New York, NY, USA, 1985. ACM Press.
- [Lia] Yulia Lierler. CMODELS homepage. <http://www.cs.utexas.edu/~tag/cmodels/>.
- [Lieb] Yulia Lierler. SUP homepage. <http://www.cs.utexas.edu/users/tag/sup/>.
- [Lif08] Vladimir Lifschitz. What is answer set programming? In *AAAI’08: Proceedings of the 23rd national conference on Artificial intelligence*, pages 1594–1597. AAAI Press, 2008.
- [LPF<sup>+</sup>06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlvsystem for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7(3):499–562, 2006.
- [Nou05] Dana Nourie. Getting started with an integrated development environment (ide), 2005. <http://java.sun.com/developer/technicalArticles/tools/intro.html>.
- [Pro09] Viera K. Proulx. Test-driven design for introductory oo programming. In *SIGCSE ’09: Proceedings of the 40th ACM technical symposium on Computer science education*, pages 138–142, New York, NY, USA, 2009. ACM.
- [PRS02] Jenny Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
- [RB06] Jim des Rivieres and Wayne Beaton. Eclipse platform technical overview, 2006. Available from <http://www.eclipse.org/articles/Whitepaper-Platform-3.1/eclipse-platform-whitepaper.html>.
- [Rei96] Steven P. Reiss. Software tools and environments. *ACM Computing Surveys*, 28(1):281–284, 1996.
- [REM<sup>+</sup>04] Gregg Rothermel, Sebastian Elbaum, Alexey G. Malishevsky, Praveen Kallakuri, and Xuemei Qiu. On test suite composition and cost-effective regression testing. *ACM Trans. Softw. Eng. Methodol.*, 13(3):277–331, 2004.

- [Ric] Francesco Ricca. A java wrapper for dlvs. Available from <http://www.mat.unical.it/wrapper/papers/asp03.pdf>.
- [Rua04] Janet M. Ruane. *Essentials of Research Methods: A Guide to Social Science Research*. Wiley-Blackwell, 2004.
- [Rup10] Nayan B. Ruparelia. The history of version control. *SIGSOFT Softw. Eng. Notes*, 35(1):5–9, 2010.
- [See03] Donn M. Seeley. Coding smart: People vs. tools. *Queue*, 1(6):33–40, 2003.
- [SJ90] Lucy Suchman and Brigitte Jordan. Interactional troubles in face-to-face survey interviews. *Journal of the American Statistical Association*, 85(409):232–241, 1990.
- [Som06a] Ian Sommerville. *Software Engineering: (Update) (8th Edition)*. Addison Wesley, 2006.
- [Som06b] Ian Sommerville. *Software Engineering: (Update) (8th Edition)*, page 126. Addison Wesley, 2006.
- [Som06c] Ian Sommerville. *Software Engineering: (Update) (8th Edition)*, chapter 16, pages 383–385. Addison Wesley, 2006.
- [Sun10] Sun. The java tutorials, 2010. Available from <http://java.sun.com/docs/books/tutorial/reflect/index.html>.
- [Sur06] A. Sureshkumar. Ansprolog\* programming environment (ape): Investigating software tools for answer set programming through the implementation of an integrated development environment. B.Sc. Dissertation, Department of Computer Science, University of Bath, June 2006.
- [Syr] Tommi Syrjänen. *Lparse 1.0 User's Manual*. Available from <http://www.tcs.hut.fi/Software/smodels/lparse.ps>.
- [Syr06] Tommi Syrjänen. Debugging inconsistent answer set programs. In Jürgen Dix and Anthony Hunter, editors, *Proceedings of the 11th Workshop on Non-monotonic Reasoning (NMR)*, number Ifl-06-04 in Ifl Technical Report Series, 2006. Available from <http://cig.in.tu-clausthal.de/NMR06/>.
- [Sze09] Gábor Szeder. Unit testing for multi-threaded java programs. In *PADTAD '09: Proceedings of the 7th Workshop on Parallel and Distributed Systems*, pages 1–8, New York, NY, USA, 2009. ACM.
- [Wah99] Nancy J. Wahl. An overview of regression testing. *SIGSOFT Softw. Eng. Notes*, 24(1):69–73, 1999.
- [WC01] Leslie B. Wilson and Robert G. Clark. Prolog. In *Comparative Programming Languages*, pages 343–344, Harlow, England, 2001. Addison-Wesley Publishers Ltd.

- [WSW05] Michael Wick, Daniel Stevenson, and Paul Wagner. Using testing and junit across the curriculum. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education*, pages 236–240, New York, NY, USA, 2005. ACM.

All links in this dissertation were accessible on the 5th May 2010.

# Appendix A

## Questionnaire

As found on <http://www.robibbotson.co.uk/questionnaire.php>

---

1) Which editor do you currently use to develop ASP programs?

2) How many years have you been developing ASP programs?

3) Which ASP tools do you currently use?

- dlv
- dlt
- lparse/smodels
- lparse/cmodels
- lparse/sup
- gringo/clasp
- ASPViz
- SPOCK
- Other

4) Are you aware of APE?

- Yes
- No

**(Only if Yes to 4)**

5) Do you use APE?

- Yes
- No

**(Only if No to 5)**

6) What is your reason for not using APE?

- Do not understand how to use it
- Does not support tools I use (please specify the missing tools in the 'other' box below)
- Missing features (please specify the missing features in the 'other' box below)
- Other

**(To all)**

7) Which of the following features would you like in an IDE for ASP (please choose a maximum of three)

- Visualisation of program
- Automatic completion of predicates
- Integration of dlv
- Integration of gringo/clasp
- Conversion between different solvers
- Replacement of a rule by its grounding
- Debugging (please specify the debugging features you would like in the 'other' box below)
- Other

8) Any further comments about an IDE for ASP?

## Appendix B

# Questionnaire Results

Which editor do you currently use to develop ASP programs?

Editor Name	Count
Text Editor	3
Emacs	5
Eclipse	2
lClingo	1
Text Edit	1
Context	2
TextWrangler	1
Nedit	1
Notepad++	1
gedit	1
vi	2

How many years have you been developing ASP programs?

Years Developing	Count
1	3
2	1
3	2
4	2
5	4
6	1
8	1
10	1
15	1
17	1

**Which ASP tools do you currently use?**

<b>Tool</b>	<b>Count</b>
dlv	8
dlt	2
lparse/smodels	11
lparse/cmodels	1
lparse/sup	1
gringo/clasp	12
ASPViz	3
SPOCK	1
Other	7

Other:

- dlv-complex
- asperix
- ezesp
- platypus
- claspar
- bingo
- clingo

**Are you aware of APE?**

<b>Yes</b>	6
<b>No</b>	11

**Do you use APE?**

<b>Yes</b>	0
<b>No</b>	6

**What is your reason for not using APE?**

Other:

- Have never tried it
- Does not offer enough features



<b>Reason</b>	<b>Count</b>
Do not understand how to use it	0
Does not support tools I use (please specify the missing tools in the 'other' box below)	0
Missing features (please specify the missing features in the 'other' box below)	1
Other	4

- Does not work with eclipse
- Habitual emacs user

**Which of the following features would you like in an IDE for ASP?**

<b>Feature</b>	<b>Count</b>
Visualisation of program	7
Automatic completion of predicates	9
Integration of dlv	2
Integration of gringo/clasp	5
Conversion between different solvers	10
Replacement of a rule by its grounding	2
Debugging (please specify the debugging features you would like in the 'other' box below)	7
Other	1

Other:

- Integration of bingo/caspd

**Any further comments about an IDE for ASP?**

- Converting to DLV syntax would be very good!
- Editing facilities in an "eclipse" style
- I guess visualization of programs refers to an intuitive visualization of answer sets so that an immediate feeling of what comes out is available.
- I have seen a feature in many procedural IDEs that I like. Clicking a (-) next to the function name hides the body of the function(only shows the name), clicking the (+) shows the entire function. For ASP, (-) only shows the head of a rule. (+) shows the entire rule.

- At least in my case, the use of an IDE would depend on whether it supports new / experimental / non-mainstream solvers. I am especially thinking of whether automatic completion, color highlighting or syntax checking might break in a bad way (crashes or the IDE refusing to save or run the program) if one writes programs in some extended syntax, that the IDE does not know about. If all that happens is minor issues, e.g. the color highlighting does not look good, or syntax checking needs to be disabled, then I would be ok with it.
- This sounds like an excellent idea, could definitely help make developing ASP programs easier.
- Syntax highlighting, highlights unrestricted (first-order) variables
- Support for modularization (sections, files), size estimate for ground program (atoms, clauses), recognizing of obviously wrong clauses (no ground atoms, ...)
- To support Answer Set Programming, you need to understand both the paradigm and the methodology / methodologies used. Once you understand the methodology you can support it.
- Automatic syntax checking (depending on target system), hints on possible problems (configurable), aspviz is a nice idea but somewhat difficult to use, visualization should be simplified

# Appendix C

## Requirements Specification

This chapter documents the requirements for the project. Further discussion can be found in Section 3, page 19.

### C.1 Functional

1. The system should provide a easy way to create an AnsProlog\* project
2. The system should provide a easy way for creating a new AnsProlog\* file
3. The system should provide a source file editor
  - (a) The system should provide an editor for the grounders LPARSE and Gringo.
  - (b) The editor should provide syntax highlighting
  - (c) It should identify keywords, comments, variables, functions, atoms, negated atoms, end of line operator and the ‘:-’ characters signifying a rule.
4. The editor should provide syntax checking
  - (a) The editor should underline syntax errors
  - (b) The editor should explain the error
5. The editor should indent source code
6. The editor should provide block commenting
7. The system should be able to convert between different programs
  - (a) The system should convert from the language of LPARSE to Gringo
  - (b) The system should convert from the language of Gringo to LPARSE

8. The system should provide a way for auto completion
  - (a) The system should provide auto-completion of predicates from within the editor by using the standard Eclipse shortcut of **Control+space**
  - (b) The system should provide a new view showing all the predicates in the program giving an outline
9. The system should provide helpful error notifications when the program location has not been set
10. The system should provide a framework
  - (a) The framework should allow automatic generation of a user interface to configure solver arguments
  - (b) The framework should allow automatic generation of the delegate that runs the solver
  - (c) The framework should allow automatic generation of launch shortcuts that the solver can use
11. The system should be to filter the input to a solver
12. The system should filter output
13. The system should be able to save the program output
14. The system should provide debugging tools
15. The system provide version control tools

## **C.2 Non-functional**

1. The system should have a framework in place to allow easy integration of existing ASP tools
  - (a) The system should integrate the tools Gringo/clasp into APE
2. The system should run on multiple platforms
3. The system should run on the platform that the existing tools run on
4. The system should have tests in place to identify errors
  - (a) The tests should identify errors when a new version of Eclipse is released
  - (b) The tests should identify any errors that may occur when new features are added to APE

## Appendix D

# Installing APE

There are a variety of ways that the version of APE completed for this project can be downloaded and installed onto your system. As it is written in Java it is cross platform meaning that there are no specific operating system instructions.

### D.1 Update Site

In Eclipse navigate to the **Help** menu bar then select **Install New Software**. A dialog box as seen in E.3 on page 90 will be presented. Enter the update site into the **Work with:** text box. The update site is <http://ape.robibbotson.co.uk/updates>. The AnsProlog\* Programming Environment can then be selected and installed onto the system following the prompts.

When Eclipse has restarted the AnsProlog\* perspective can be activated by going to the **Window** menu bar. Then **Open Perspective** selecting **AnsProlog\***. APE is then ready to be used.

### D.2 Supplied CD or Website

Alternatively APE can be installed from the code supplied on the CD folder `update-site` or from the website <http://ape.robibbotson.co.uk/update-site>. Copy or download this to your computer storing in a local folder. This folder now contains all the same code found on the remote update site documented above. In Eclipse the **Install New Software** wizard can be directed to point to a local update site. Point it to the local folder and install it like above.

Alternatively the `src` folder found on the supplied CD or website can be compiled to build the correct JARs. These can then be copied across to the Eclipse plug-in directory. However this method of installation is not favoured and one of the above methods should be used.

# Appendix E

## Implementation Screenshots

In order to not break up the flow of the main dissertation body this chapter contains the screenshots referred to in chapter 5.

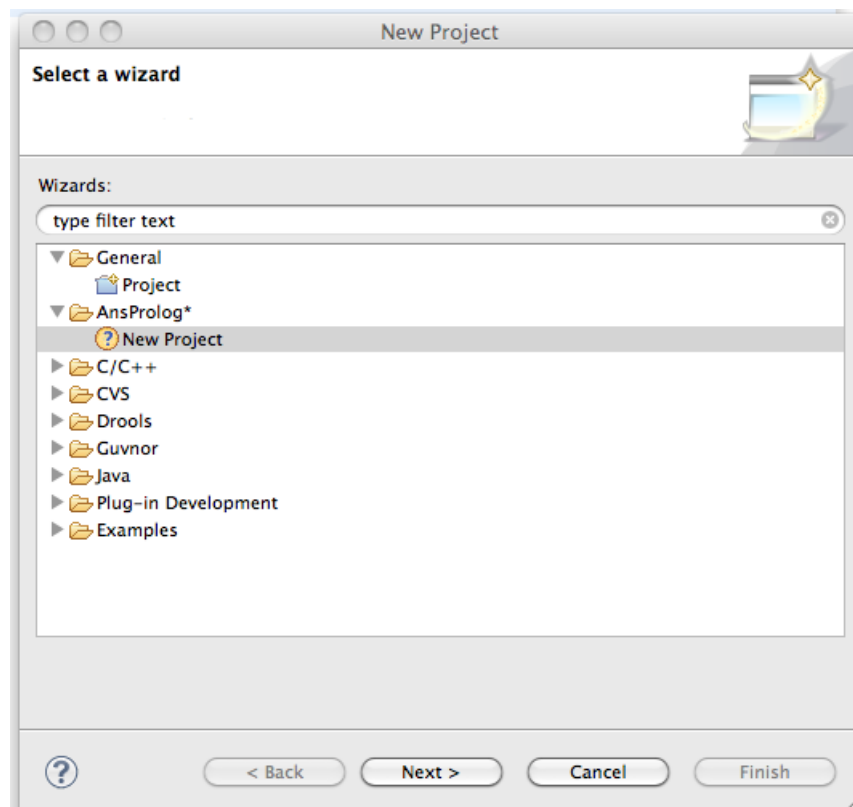


Figure E.1: New AnsProlog\* Project

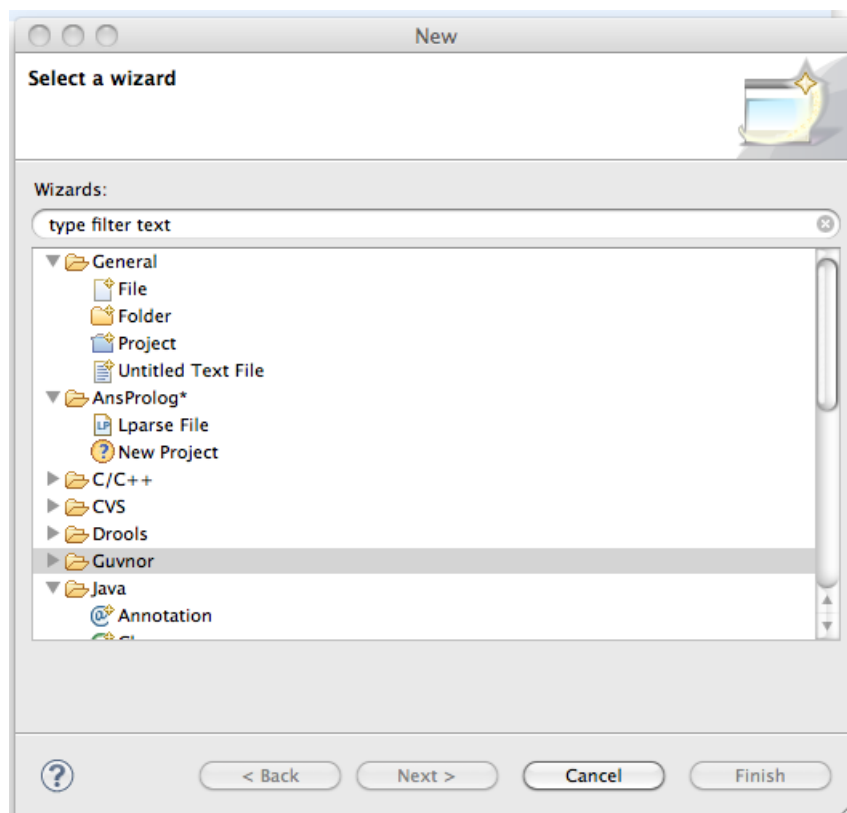


Figure E.2: New LPARSE File

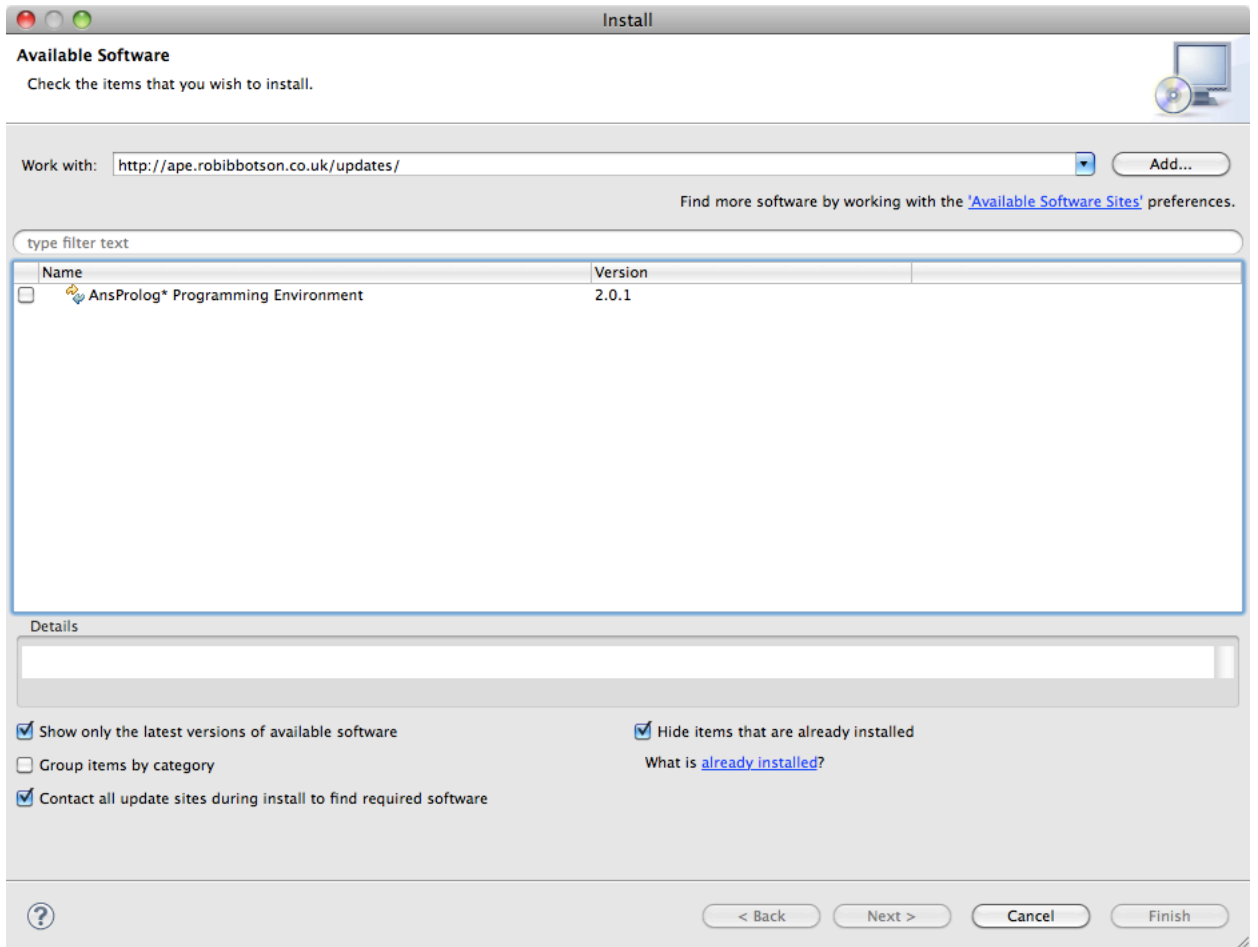


Figure E.3: Installing APE



# Appendix F

## Testing

A new plug-in project `uk.ac.bath.cs.asp.ide.testing` has been created containing all the test code. In order to run the tests please download the source code, either from the `src` folder on the supplied cd, from the website <http://ape.robibbotson.co.uk> or from the GIT repository found at `git@github.com:robibbotson/APE.git`

The testing project has a dependency on the Eclipse plug-in SWTBot found at <http://www.eclipse.org/swtbot/>. This can be installed into Eclipse using the update manager and update site <http://download.eclipse.org/technology/swtbot/galileo/dev-build/update-site/>.

### F.1 Running The Tests

When the source code has been inserted into the Eclipse workspace the tests can be run using SWTBot. The automatic tests are all contained within the plug-in project `uk.ac.bath.cs.asp.ide.testing`. To run first go to `Run Configurations...` and then select a new SWTBot launch configuration. Select the testing package and click `Run`. A new version of Eclipse will pop up to run all the tests.

#### F.1.1 Screencast

Actions speak louder than words so a screencast has been provided showing the tests in action. This video of this can be found on the website <http://ape.robibbotson.co.uk/screencasts> or in the folder `screencasts` on the supplied CD.

## **F.2 New Feature Test**

One of the main features added in the project was a framework with the ability to quickly add a new external program into APE. This was put to the test with the program ASPeRIX being added into APE. ASPeRIX is a grounder and solver combined into one. A screencast showing the ease of the process has been produced. Again this can be found on the website <http://ape.robibbotson.co.uk/screencasts> or in the folder `screencasts` on the supplied CD.

# Appendix G

## Predicate Completion Tests

### G.1 Base File

This is the file that all other tests will be based from. It sets up the program ready to be extended later on.

#### G.1.1 File: puzzle1\_base.lp

```
% puzzle1.lp — Knights and Knaves
% author: Tommi Syrj nen
%
% Raymond Smullyan presents the following puzzle in his book 'Forever
% Undecided '.
%
% The Island of Knights and Knaves has two types of inhabitants:
% knights, who always tell the truth, and knaves, who always lie.
%
% One day, three inhabitants (A, B, and C) of the island met a foreign
% tourist and gave the following information about themselves:
%
% 1. A said that B and C are both knights.
% 2. B said that A is a knave and C is a knight.
%
% What types are A, B, and C?
%
% There are three persons in this puzzle
person(a; b; c).

% Each person is either knight or knave, but not both.
1 { knight(P), knave(P) } 1 :- person(P).

% Rest of the rules model the hints.

%% Hint 1:

% If A tells the truth, both B and C are knights
2 { knight(b), knight(c) } 2 :- knight(a).
```

```

% If A lies, it is not possible that they are both knights
:- knave(a), knight(b), knight(c).

%% Hint 2:

% If B tells the truth, A is a knave and C is a knight
2 { knave(a), knight(c) } 2 :- knight(b).

% If B lies, it is not possible that A is knave and C is knight
:- knave(b), knave(a), knight(c).

```

## G.2 New Predicate Declaration

### G.2.1 File: puzzle1\_new.lp

```

% puzzle1.lp — Knights and Knaves
% author: Tommi Syrj nen
%
% Raymond Smullyan presents the following puzzle in his book 'Forever
% Undecided'.
%
% The Island of Knights and Knaves has two types of inhabitants:
% knights, who always tell the truth, and knaves, who always lie.
%
% One day, three inhabitants (A, B, and C) of the island met a foreign
% tourist and gave the following information about themselves:
%
% 1. A said that B and C are both knights.
% 2. B said that A is a knave and C is a knight.
%
% What types are A, B, and C?
%

% There are three persons in this puzzle
person(a; b; c).

% Each person is either knight or knave, but not both.
1 { knight(P), knave(P) } 1 :- person(P).

% Rest of the rules model the hints.

%% Hint 1:

% If A tells the truth, both B and C are knights
2 { knight(b), knight(c) } 2 :- knight(a).

% If A lies, it is not possible that they are both knights
:- knave(a), knight(b), knight(c).

%% Hint 2:

% If B tells the truth, A is a knave and C is a knight
2 { knave(a), knight(c) } 2 :- knight(b).

% If B lies, it is not possible that A is knave and C is knight
:- knave(b), knave(a), knight(c).

kni

```

## G.3 Previous Predicate Declaration

### G.3.1 File: puzzle1\_previous.lp

```
% puzzle1.lp — Knights and Knaves
% author: Tommi Syrj nen
%
% Raymond Smullyan presents the following puzzle in his book 'Forever
% Undecided'.
%
% The Island of Knights and Knaves has two types of inhabitants:
% knights, who always tell the truth, and knaves, who always lie.
%
% One day, three inhabitants (A, B, and C) of the island met a foreign
% tourist and gave the following information about themselves:
%
% 1. A said that B and C are both knights.
% 2. B said that A is a knave and C is a knight.
%
% What types are A, B, and C?
%

% There are three persons in this puzzle
person(a; b; c).

% Each person is either knight or knave, but not both.
1 { knight(P), knave(P) } 1 :- person(P).

% Rest of the rules model the hints.

%% Hint 1:

% If A tells the truth, both B and C are knights
2 { knight(b), knight(c) } 2 :- knight(a).

% If A lies, it is not possible that they are both knights
:- knave(a), knight(b), knight(c).

%% Hint 2:

% If B tells the truth, A is a knave and C is a knight
2 { knave(a), knight(c) } 2 :- knight(b).

% If B lies, it is not possible that A is knave and C is knight
:- knave(b), knave(a), knight(c).

knight
```

## G.4 Variable Predicate Declaration

### G.4.1 File: puzzle1\_variable.lp

```
% puzzle1.lp — Knights and Knaves
% author: Tommi Syrj nen
%
% Raymond Smullyan presents the following puzzle in his book 'Forever
% Undecided'.
%
```

```

% The Island of Knights and Knaves has two types of inhabitants:
% knights, who always tell the truth, and knaves, who always lie.
%
% One day, three inhabitants (A, B, and C) of the island met a foreign
% tourist and gave the following information about themselves:
%
% 1. A said that B and C are both knights.
% 2. B said that A is a knave and C is a knight.
%
% What types are A, B, and C?
%

% There are three persons in this puzzle
person(a; b; c).

% Each person is either knight or knave, but not both.
1 { knight(P), knave(P) } 1 :- person(P).

% Rest of the rules model the hints.

%% Hint 1:

% If A tells the truth, both B and C are knights
2 { knight(b), knight(c) } 2 :- knight(a).

% If A lies, it is not possible that they are both knights
:- knave(a), knight(b), knight(c).

%% Hint 2:

% If B tells the truth, A is a knave and C is a knight
2 { knave(a), knight(c) } 2 :- knight(b).

% If B lies, it is not possible that A is knave and C is knight
:- knave(b), knave(a), knight(c).

knight(X) :- p

```

## G.5 Expected Results

### G.5.1 New Predicate Declaration

#### Predicate View

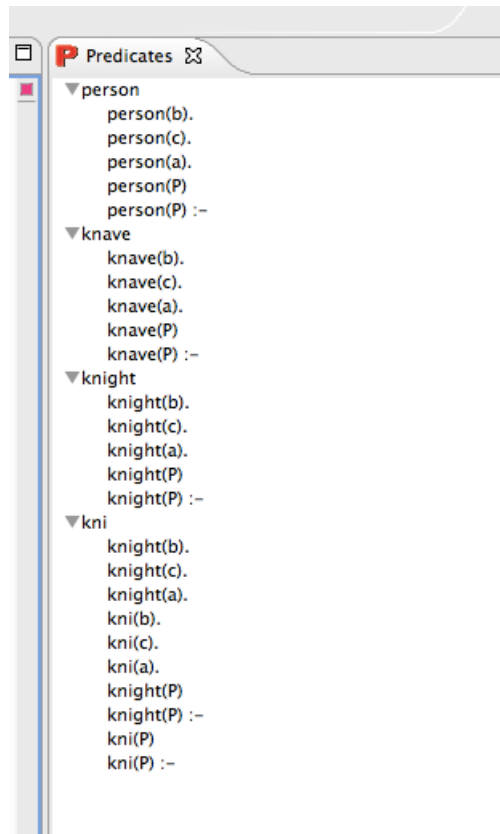


Figure G.1: Predicate View Suggestions

## Auto Completion



Figure G.2: Auto Completion Suggestions

## G.5.2 Previous Predicate Declaration

### Predicate View

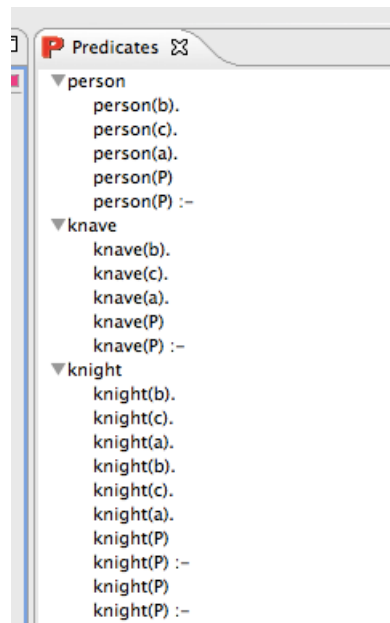


Figure G.3: Predicate View Suggestions



### Auto Completion

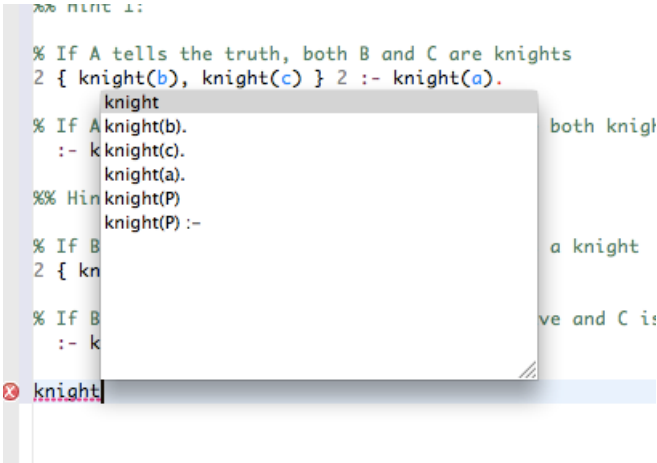


Figure G.4: Auto Completion Suggestions

### G.5.3 Variable Predicate Declaration

#### Predicate View

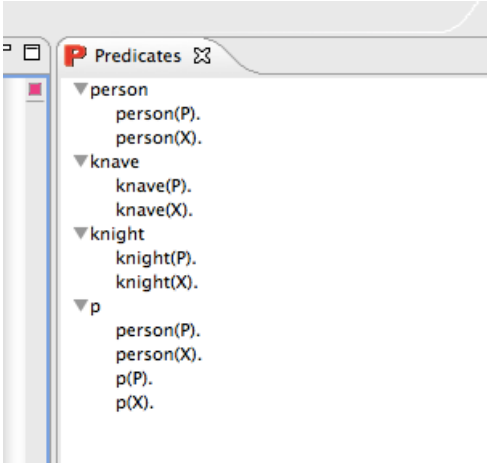


Figure G.5: Predicate View Suggestions

## Auto Completion

```
% If A tells the truth, both B and C are knights
2 { knight(b), knight(c) } 2 :- knight(a).
% If A lies, i
:- knave(a), person(X).
%% Hint 2:
% If B tells t
2 { knave(a),
% If B lies, i
:- knave(b),
knight(X) :- p
```

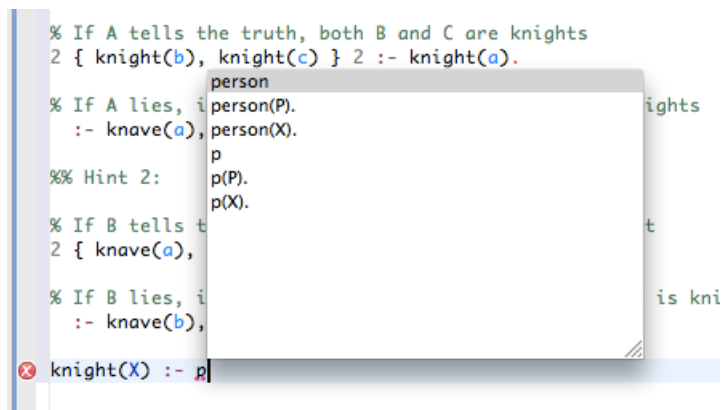


Figure G.6: Auto Completion Suggestions

# Appendix H

## Code

Due to the large size, it was not suitable to include all source code from APE in this chapter. Instead key aspects are included that highlight the points made in the dissertation. The full source code is available on the supplied CD in the folder `src`. Alternatively the code can be accessed from the GIT repository found at `git@github.com:robibbotson/APE.git` or from the website at <http://ape.robibbotson.co.uk/src>.

## H.1 Framework

The following classes make up the framework that has been produced. Programs making use of the framework extend from these classes.

### H.1.1 File: AbstractArgumentConstants.java

```
102 /*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.launch;

import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

import sun.reflect.Reflection;
import uk.ac.bath.cs.asp.ide.ASPPlugin;
```

```
/**
 * Abstract class to allow clients extending it to have
 * an easy mechanism to add
 * program arguments. The client should use it in
 * conjunction with the @CommandLineSwitch
 * annotation to have a list of program arguments. This
 * class allows the
 * additional arguments to be added, the ones that depend
 * upon others. For
 * example a radio button when selected may then need a
 * text box for the user to
 * enter the specifics. This class is to manage these
 * dependencies.
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractArgumentConstants
{
    private Map<String, Field> additionalArguments;

    public AbstractArgumentConstants()
    {
        additionalArguments = new HashMap<String, Field>
            >();
    }

    public Map<String, Field> getAdditionalArguments()
    {
        return additionalArguments;
    }

    public void addAdditionalArguments(String key, String
        value)
    {
        additionalArguments.put(getField(key).getName(),
            getField(value));
    }

    protected Field getField(String fieldName)
    {
        Field field = null;
        try
        {

```

```

        field = Reflection.getCallerClass(3).
            getDeclaredField(fieldName);
    }
    catch (SecurityException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    catch (NoSuchFieldException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    return field;
}
}
}

```

## H.1.2 File: AbstractLaunchConfigurationDelegate.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor , Boston , MA 02110-1301 , USA.
 *****/
package uk.ac.bath.cs.asp.ide.launch;

```

```

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.lang.reflect.Field;
import java.util.LinkedList;
import java.util.List;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.core.runtime.IProgressMonitor;
import org.eclipse.debug.core.ILaunch;
import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.model.
    ILaunchConfigurationDelegate;
import org.eclipse.debug.core.model.RuntimeProcess;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Display;
import org.eclipse.swt.widgets.MessageBox;
import org.eclipse.swt.widgets.Spinner;
import org.eclipse.swt.widgets.Text;

import uk.ac.bath.cs.asp.ide.ASPPlugin;

/**
 * An abstract class for allowing clients implementing it
 * an easy way to launch
 * a program. To be used in conjunction with
 * @CommandLineSwitch to allow it to
 * get all the program arguments to launch it.
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractLaunchConfigurationDelegate
    implements ILaunchConfigurationDelegate
{
    @Override
    public void launch(ILaunchConfiguration configuration
        , String mode, ILaunch launch, IProgressMonitor
        monitor)
        throws CoreException
    {

```

```

    final ProcessBuilder processBuilder = launch(
        configuration);
    if (processBuilder != null)
    {
        Process process = null;
        try
        {
            process = processBuilder.start();
        }
        catch (IOException e)
        {
            ASPPlugin.getLogger().logException(e);
        }
        new RuntimeProcess(launch, process, mode,
            null);
    }
}

public ProcessBuilder launch(ILaunchConfiguration
    configuration)
{
    ProcessBuilder builder = null;
    List<String> commandLine = new LinkedList<String>
        >();
    String programLocation = getProgramLocation();
    if (programLocation != null)
    {
        commandLine.add(programLocation);
        commandLine.addAll(getProgramArguments(
            configuration));
        builder = new ProcessBuilder(commandLine);
    }
    return builder;
}

public abstract String getProgramLocation();

public abstract List<String> getProgramArguments(
    ILaunchConfiguration configuration);

protected List<String> getFieldValue(
    ILaunchConfiguration configuration, Field field,
    AbstractArgumentConstants constants)
{
    List<String> arguments = new LinkedList<String>()
        ;
    if (field.getType() == Button.class)
    {
        arguments.addAll(getButtonValues(
            configuration, field, constants));
    }
    if (field.getType() == Text.class)
    {
        arguments.addAll(getTextValues(configuration,
            field, constants));
    }
    if (field.getType() == Spinner.class)
    {
        arguments.addAll(getSpinnerValues(
            configuration, field, constants));
    }
    return arguments;
}

private List<String> getButtonValues(
    ILaunchConfiguration configuration, Field field,
    AbstractArgumentConstants constants)
{
    List<String> arguments = new LinkedList<String>()
        ;
    boolean attribute = false;
    try
    {
        attribute = configuration.getAttribute(field.
            getName(), false);
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    if (attribute)
    {
        CommandLineSwitch annotation = field.
            getAnnotation(CommandLineSwitch.class);
        arguments.add(annotation.option());
        if (annotation.additional())
        {
            Field additionalField = constants.
                getAdditionalArguments().get(field.

```

```

        getName());
        arguments.addAll(getFieldValue(
            configuration, additionalField,
            constants));
    }
}
return arguments;
}

protected List<String> getTextValues(
    ILaunchConfiguration configuration, Field field,
    AbstractArgumentConstants constants)
{
    List<String> arguments = new LinkedList<String>()
        ;
    String attribute = null;
    try
    {
        attribute = configuration.getAttribute(field.
            getName(), "");
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    if (attribute != null && !attribute.equals(""))
    {
        arguments.add(attribute);
    }
    return arguments;
}

protected List<String> getSpinnerValues(
    ILaunchConfiguration configuration, Field field,
    AbstractArgumentConstants constants)
{
    List<String> arguments = new LinkedList<String>()
        ;
    int attribute = -999;
    try
    {
        attribute = configuration.getAttribute(field.
            getName(), -999);
    }
    catch (CoreException e)
        {
            ASPPlugin.getLogger().logException(e);
        }
    if (attribute != -999)
    {
        arguments.add(Integer.toString(attribute));
    }
    return arguments;
}

/**
 * Helper method to pipe output from one process to
 * another.
 *
 * @param output
 *         The process to read the output from
 *
 * @param input
 *         The process to write the input to
 */
protected void pipe(Process output, Process input)
{
    BufferedReader reader = new BufferedReader(new
        InputStreamReader(output.getInputStream()));
    BufferedWriter writer = new BufferedWriter(new
        OutputStreamWriter(input.getOutputStream()));
    String line = "";
    try
    {
        while ((line = reader.readLine()) != null)
        {
            writer.write(line);
            writer.newLine();
        }
        writer.flush();
        reader.close();
        writer.close();
    }
    catch (IOException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
}

@SuppressWarnings("unchecked")

```

```

protected List<String> getInputFiles(
    ILaunchConfiguration configuration)
{
    List<String> files = null;
    try
    {
        files = configuration.getAttribute(
            LaunchConstants.ATTR_INPUT_FILES, new
            LinkedList<String>());
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    return files;
}

protected void programNotSetErrorMessage(final String
    program)
{
    // Have to do on ui thread
    final Display display = Display.getDefault();
    display.syncExec(new Runnable()
    {
        public void run()
        {
            MessageBox box = new MessageBox(display.
                getActiveShell(), SWT.OK);
            box.setText("Problem");
            box.setMessage("The program location for
                " + program
                + " has not been set. Please set
                in the preferences dialog.");
            box.open();
        }
    });
}
}

```

### H.1.3 File: AbstractLaunchShortcut.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
   the Eclipse platform

```

```

 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
   and/or modify it under the
 * terms of the GNU General Public License as published
   by the Free Software
 * Foundation; either version 2 of the License, or (at
   your option) any later version.
 *
 * This program is distributed in the hope that it will
   be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
   License for more details.
 *
 * You should have received a copy of the GNU General
   Public License along with this
 * program; if not, write to the Free Software Foundation
   , Inc. , 51 Franklin Street ,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.launch;

import java.io.File;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

import org.eclipse.core.resources.IFile;
import org.eclipse.core.resources.IResource;
import org.eclipse.core.runtime.CoreException;
import org.eclipse.debug.core.DebugPlugin;
import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.ILaunchConfigurationType;
import org.eclipse.debug.core.ILaunchConfigurationWorkingCopy;
import org.eclipse.debug.core.ILaunchManager;
import org.eclipse.debug.ui.ILaunchShortcut;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.ui.IEditorInput;
import org.eclipse.ui.IEditorPart;

import uk.ac.bath.cs.asp.ide.ASPPlugin;

```



```

/**
 * Abstract class to allow clients implementing an easy
 * way to launch a program.
 * To be used with {@link
 *   AbstractLaunchConfigurationDelegate} to launch the
 * program. This shortcut allows launching from the
 * editor or navigator view.
 *
 * @author Rob Ibbotson
 */
public class AbstractLaunchShortcut implements
    ILaunchShortcut
{
    protected final ILaunchManager manager =
        DebugPlugin.getDefault().getLaunchManager();
    protected ILaunchConfigurationType type = null;

    public AbstractLaunchShortcut(String type)
    {
        this.type = manager.getLaunchConfigurationType(
            type);
    }

    @Override
    public void launch(ISelection selection, String mode)
    {
        if (selection instanceof IStructuredSelection)
        {
            final List<String> inputFiles =
                getInputFilesFromSelection((
                    IStructuredSelection) selection);
            launch(inputFiles);
        }
    }

    @Override
    public void launch(IEditorPart editor, String mode)
    {
        IEditorInput editorInput = editor.getEditorInput
            ();
        IResource adapter = (IResource) editorInput.
            getAdapter(IResource.class);
        String workspaceLocation = adapter.getWorkspace()
            .getRoot().getLocation().toOSString();

```

```

        String fileName = adapter.getFullPath().
            toOSString();
        String fullPath = workspaceLocation + fileName;
        launch(Arrays.asList(fullPath));
    }

    private void launch(List<String> inputFiles)
    {
        if (inputFiles.size() > 0)
        {
            ILaunchConfigurationWorkingCopy workingCopy =
                getWorkingCopy(manager, type, inputFiles
                    );
            workingCopy = (workingCopy == null) ?
                makeNewWorkingCopy(inputFiles, type) :
                workingCopy;
            try
            {
                workingCopy.launch(ILaunchManager.
                    RUNMODE, null);
            }
            catch (CoreException e)
            {
                ASPPlugin.getLogger().logException(e);
            }
        }
    }

    private List<String> getInputFilesFromSelection(
        IStructuredSelection selection)
    {
        final List<String> inputFiles = new LinkedList<
            String>();
        for (Object o : selection.toArray())
        {
            if (o instanceof IFile)
            {
                final IFile file = (IFile) o;
                final String filePath = file.getLocation
                    ().toFile().toString();
                inputFiles.add(filePath);
            }
        }
        return inputFiles;
    }
}

```

```

@SuppressWarnings("unchecked")
private ILaunchConfigurationWorkingCopy
    getWorkingCopy(ILaunchManager manager,
        ILaunchConfigurationType type,
        Object inputFiles)
{
    ILaunchConfigurationWorkingCopy workingCopy =
        null;
    try
    {
        for (ILaunchConfiguration config : manager.
            getLaunchConfigurations(type))
        {
            List<String> configInputFiles = config.
                getAttribute(LaunchConstants.
                    ATTR_INPUT_FILES,
                    new LinkedList<String>());
            if (inputFiles.equals(configInputFiles))
            {
                workingCopy = config.getWorkingCopy()
                    ;
                break;
            }
        }
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    return workingCopy;
}

private ILaunchConfigurationWorkingCopy
    makeNewWorkingCopy(List<String> inputFiles,
        ILaunchConfigurationType type)
{
    File file = new File(inputFiles.get(0).toString()
        );
    ILaunchConfigurationWorkingCopy workingCopy =
        null;
    try
    {
        workingCopy = type.newInstance(null, file.
            getName());
    }

```

```

        workingCopy.setAttribute(LaunchConstants.
            ATTR_INPUT_FILES, inputFiles);
        workingCopy.doSave();
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    return workingCopy;
}
}

```

#### H.1.4 File: AbstractMultipleFileArgumentsTab.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.launch;

import java.util.LinkedHashMap;
import java.util.Map;

```

```

import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.
    ILaunchConfigurationWorkingCopy;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.ExpandBar;
import org.eclipse.swt.widgets.ExpandItem;

/**
 * Gui abstract control to show the program argumnets
 *   specified
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractMultipleFileArgumentsTab
    extends AbstractSingleFileArgumentsTab
{
    protected Map<String, AbstractArgumentConstants>
        fileList = new LinkedHashMap<String,
            AbstractArgumentConstants>();

    private int
        expandItemNumber = 0;

    @Override
    public abstract String getName();

    @Override
    public abstract void setDefaults(
        ILaunchConfigurationWorkingCopy configuration);

    @Override
    public void createControl(Composite parent)
    {
        ExpandBar bar = new ExpandBar(parent, SWT.
            V_SCROLL);
        for (String name : fileList.keySet())
        {
            super.setArgumentConstantFile(fileList.get(
                name));
            Composite composite = super.
                createCompostiteControl(bar);
            createNextExpandItem(bar, composite, name);
        }
        setControl(bar);
    }

```

```

    }

    @Override
    public void initializeFrom(ILaunchConfiguration
        configuration)
    {
        for (String name : fileList.keySet())
        {
            super.setArgumentConstantFile(fileList.get(
                name));
            super.initializeFrom(configuration);
        }
    }

    private ExpandItem createNextExpandItem(ExpandBar bar
        , Composite composite, String text)
    {
        ExpandItem item = new ExpandItem(bar, SWT.NONE,
            expandItemNumber);
        item.setText(text);
        item.setHeight(composite.computeSize(SWT.DEFAULT,
            SWT.DEFAULT).y);
        item.setControl(composite);
        if (expandItemNumber == 0)
        {
            item.setExpanded(true);
        }
        expandItemNumber++;
        return item;
    }
}

```

## H.1.5 File: AbstractSingleFileArgumentsTab.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 *   the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 *   and/or modify it under the
 * terms of the GNU General Public License as published
 *   by the Free Software

```

```

* Foundation; either version 2 of the License, or (at
  your option) any later version.
*
* This program is distributed in the hope that it will
  be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public
  License for more details.
*
* You should have received a copy of the GNU General
  Public License along with this
* program; if not, write to the Free Software Foundation
  , Inc., 51 Franklin Street,
* Fifth Floor, Boston, MA 02110-1301, USA.
*****/
package uk.ac.bath.cs.asp.ide.launch;

import java.lang.reflect.Field;
import java.util.HashMap;
import java.util.Map;

import org.eclipse.core.runtime.CoreException;
import org.eclipse.debug.core.ILaunchConfiguration;
import org.eclipse.debug.core.ILaunchConfigurationWorkingCopy;
import org.eclipse.debug.ui.AbstractLaunchConfigurationTab;
import org.eclipse.swt.SWT;
import org.eclipse.swt.events.ModifyEvent;
import org.eclipse.swt.events.ModifyListener;
import org.eclipse.swt.events.SelectionAdapter;
import org.eclipse.swt.events.SelectionEvent;
import org.eclipse.swt.layout.GridData;
import org.eclipse.swt.layout.GridLayout;
import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.swt.widgets.Control;
import org.eclipse.swt.widgets.Spinner;
import org.eclipse.swt.widgets.Text;

import uk.ac.bath.cs.asp.ide.ASPPlugin;

/**

```

```

* Abstract gui control to represent all the program
  arguments defined as
* constants with the {@link CommandLineSwitch}
  annotation.
*
* @author Rob Ibbotson
*/
public abstract class AbstractSingleFileArgumentsTab
  extends AbstractLaunchConfigurationTab
{
  private AbstractArgumentConstants
    argumentConstantFile = null;
  protected Map<String, Control> controlMapping
    = new HashMap<String, Control>();
  private ModifyListener modifyListener
    = new ModifyListener()
    {
      @Override
      public
        void
        modifyText
        (
          ModifyEvent
          e
          )
      {
        updateLaunchCo
          ()
          ;
      }
    };

  @Override
  public abstract String getName();

```

```

@Override
public abstract void setDefaults(
    ILaunchConfigurationWorkingCopy configuration);

@Override
public void createControl(Composite parent)
{
    setControl(createCompostiteControl(parent));
}

protected Composite createCompostiteControl(Composite
    parent)
{
    Composite composite = new Composite(parent, SWT.
        NONE);
    composite.setLayout(new GridLayout(2, false));
    for (Field field : argumentConstantFile.getClass
        ().getFields())
    {
        createField(composite, field);
    }
    return composite;
}

private Control createField(Composite composite,
    Field field)
{
    Control control = null;
    Control additional = null;
    CommandLineSwitch annotation = field.
        getAnnotation(CommandLineSwitch.class);

    if (annotation != null && annotation.additional()
        )
    {
        Field additionalField = argumentConstantFile.
            getAdditionalArguments().get(field.
                getName());
        additional = createField(composite,
            additionalField);
    }

    if (field.getType() == Button.class)
    {
        control = createButton(composite, annotation,
            additional);
    }
    if (field.getType() == Spinner.class)
    {
        control = createSpinner(composite);
    }
    if (field.getType() == Text.class)
    {
        control = createText(composite);
    }

    if (additional == null)
    {
        control.setLayoutData(new GridData(SWT.
            BEGINNING, SWT.BEGINNING, false, false,
            2, 1));
    }
    else
    {
        control.moveAbove(additional);
        control.setLayoutData(new GridData(SWT.
            BEGINNING, SWT.BEGINNING, false, false,
            1, 1));
        GridData additionalGrid = new GridData(SWT.
            BEGINNING, SWT.BEGINNING, true, false, 1,
            1);
        additionalGrid.minimumWidth = 100;
        additional.setLayoutData(additionalGrid);
    }
    controlMapping.put(field.getName(), control);
    return control;
}

private Control createButton(Composite composite,
    CommandLineSwitch annotation, final Control
    additional)
{
    final Button button = new Button(composite, SWT.
        CHECK);
    button.setText(annotation.description() + " [" +
        annotation.option() + "]");
    button.addSelectionListener(new SelectionAdapter
        ())

```

```

    {
        @Override
        public void widgetSelected(SelectionEvent e)
        {
            if (additional != null)
            {
                additional.setEnabled(button.
                    getSelection());
            }
            updateLaunchConfigurationDialog();
        }
    });
    return button;
}

private Control createSpinner(Composite composite)
{
    Spinner spinner = new Spinner(composite, SWT.
        BORDER);
    spinner.addModifyListener(modifyListener);
    return spinner;
}

private Control createText(Composite composite)
{
    Text text = new Text(composite, SWT.BORDER);
    text.addModifyListener(modifyListener);
    return text;
}

@Override
public void initializeFrom(ILaunchConfiguration
    configuration)
{
    for (Field field : argumentConstantFile.getClass
        ().getFields())
    {
        Button button = (Button) initializeField(
            configuration, field);
        // Now see if we have additional and do them
        CommandLineSwitch annotation = field.
            getAnnotation(CommandLineSwitch.class);
        if (annotation.additional())
        {
            Field additionalField =
                argumentConstantFile.
                    getAdditionalArguments().get(field.
                        getName());
            Control additionalControl =
                initializeField(configuration,
                    additionalField);
            additionalControl.setEnabled(button.
                getSelection());
        }
    }
}

private Control initializeField(ILaunchConfiguration
    configuration, Field field)
{
    Control control = controlMapping.get(field.
        getName());
    try
    {
        if (control instanceof Button)
        {
            Button button = (Button) control;
            button.setSelection(configuration.
                getAttribute(field.getName(), false));
        }
        if (control instanceof Spinner)
        {
            Spinner spinner = (Spinner) control;
            spinner.setSelection(configuration.
                getAttribute(field.getName(), 0));
        }
        if (control instanceof Text)
        {
            Text text = (Text) control;
            text.setText(configuration.getAttribute(
                field.getName(), ""));
        }
    }
    catch (CoreException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    return control;
}

```

```

    }

    @Override
    public void performApply(
        ILaunchConfigurationWorkingCopy configuration)
    {
        for (String attributeName : controlMapping.keySet(
            ))
        {
            Control control = controlMapping.get(
                attributeName);
            if (control instanceof Button)
            {
                Button button = (Button) control;
                configuration.setAttribute(attributeName,
                    button.getSelection());
            }
            if (control instanceof Spinner)
            {
                Spinner spinner = (Spinner) control;
                configuration.setAttribute(attributeName,
                    spinner.getSelection());
            }
            if (control instanceof Text)
            {
                Text text = (Text) control;
                configuration.setAttribute(attributeName,
                    text.getText());
            }
        }
    }

    protected void setArgumentConstantFile(
        AbstractArgumentConstants constantFile)
    {
        argumentConstantFile = constantFile;
    }
}

```

### H.1.6 File: CommandLineSwitch.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform

```

```

 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.launch;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Allows a nice way of representing the program argument
 * . The option is what
 * the program sees. The description is for the gui and
 * what the user sees.
 * Additional defines whether any further fields depend
 * upon this one.
 *
 * @author Rob Ibbotson
 */
@Retention(RetentionPolicy.RUNTIME)
@Target( { ElementType.FIELD })
public @interface CommandLineSwitch
{
    String option() default "";
}

```

```

    String description() default "";
    boolean additional() default false;
}

```

## H.2 Gringo

This section shows some of the classes that comprise of the Gringo integration inside of APE. These classes show how the framework is used. Clasp, although not included inside this chapter follows a similar methodology.

### H.2.1 File: GringoProgramArgumentConstants.java

114

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.

```

```

*****/
package uk.ac.bath.cs.asp.ide.gringo.launch.constants;

import org.eclipse.swt.widgets.Button;
import org.eclipse.swt.widgets.Spinner;
import org.eclipse.swt.widgets.Text;

import uk.ac.bath.cs.asp.ide.launch.
    AbstractArgumentConstants;
import uk.ac.bath.cs.asp.ide.launch.CommandLineSwitch;

/**
 * Class that lists all the of the arguments that the
 * gringo program accepts.
 * The description is what the user sees with the option
 * being what is actually
 * passed to gringo. Dependencies are added in the
 * constructor
 *
 * @author Rob Ibbotson
 */
public class GringoProgramArgumentConstants extends
    AbstractArgumentConstants
{
    @CommandLineSwitch(option = "--const", description =
        "Replace constant <c> by value <v>", additional =
        true)
    public static final Button
        ATTR_GRINGO_REPLACE_CONST = null;
    @CommandLineSwitch(option = "--text", description =
        "Print plain text format")
    public static final Button
        ATTR_GRINGO_PLAIN_TEXT_FORMAT = null;
    @CommandLineSwitch(option = "--lparse", description =
        "Print Lparse format")
    public static final Button
        ATTR_GRINGO_LPARSE_FORMAT = null;
    @CommandLineSwitch(option = "--aspils", description =
        "Print ASPils format in normal form <num>",
        additional = true)
    public static final Button ATTR_GRINGO_ASPILS
        = null;
    @CommandLineSwitch(option = "--ground", description =
        "Enable lightweight mode for ground input")

```



```

public static final Button ATTR_GRINGO_LIGHT_GROUND
    = null;
@CommandLineSwitch(option = "--shift", description =
    "Shift disjunctions into the body")
public static final Button
    ATTR_GRINGO_SHIFT_DISJUNCTIONS = null;
@CommandLineSwitch(option = "--bindersplit=yes",
    description = "Turn on binder splitting")
public static final Button ATTR_GRINGO_BINDER_SPLIT
    = null;
@CommandLineSwitch(option = "--ifixed", description =
    "Fix number of incremental steps to <num>",
    additional = true)
public static final Button ATTR_GRINGO_FIX_NUMBER
    = null;
@CommandLineSwitch(option = "--ibase", description =
    "Process base program only")
public static final Button ATTR_GRINGO_BASE_ONLY
    = null;
@CommandLineSwitch(option = "--syntax", description =
    "Print syntax information")
public static final Button ATTR_GRINGO_SYNTAX
    = null;
@CommandLineSwitch(option = "--debug", description =
    "Print internal representations of rules during
    grounding")
public static final Button ATTR_GRINGO_DEBUG
    = null;
@CommandLineSwitch(option = "--stats", description =
    "Print extended statistics")
public static final Button ATTR_GRINGO_STATS
    = null;

// Additional info needed for fields
@SuppressWarnings("unused")
private static final Text
    ATTR_GRINGO_REPLACE_CONST_TEXT = null;
@SuppressWarnings("unused")
private static final Spinner ATTR_GRINGO_ASPILS_NUM
    = null;
@SuppressWarnings("unused")
private static final Spinner
    ATTR_GRINGO_FIX_NUMBER_NUM = null;

public GringoProgramArgumentConstants()
    {
        super();
        super.addAdditionalArguments("
            ATTR_GRINGO_REPLACE_CONST", "
            ATTR_GRINGO_REPLACE_CONST.TEXT");
        super.addAdditionalArguments("ATTR_GRINGO_ASPILS"
            , "ATTR_GRINGO_ASPILS_NUM");
        super.addAdditionalArguments("
            ATTR_GRINGO_FIX_NUMBER", "
            ATTR_GRINGO_FIX_NUMBER_NUM");
    }
}

```

## H.2.2 File: GringoArgumentsTab.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.gringo.launch;

```

```

import org.eclipse.debug.core.
    ILaunchConfigurationWorkingCopy;
import org.eclipse.debug.ui.DebugUITools;
import org.eclipse.debug.ui.IDebugUIConstants;
import org.eclipse.swt.graphics.Image;
import org.eclipse.swt.widgets.Composite;

import uk.ac.bath.cs.asp.ide.gringo.launch.constants.
    GringoProgramArgumentConstants;
import uk.ac.bath.cs.asp.ide.launch.
    AbstractSingleFileArgumentsTab;

/**
 * The arguments tab for gringo in the run configuration
 *
 * @author Rob Ibbotson
 */
public class GringoArgumentsTab extends
    AbstractSingleFileArgumentsTab
{
    private Image image;

    @Override
    public void createControl(Composite parent)
    {
        super.setArgumentConstantFile(new
            GringoProgramArgumentConstants());
        super.createControl(parent);
    }

    @Override
    public String getName()
    {
        return "Gringo Arguments";
    }

    @Override
    public void setDefaults(
        ILaunchConfigurationWorkingCopy configuration)
    {
    }

    @Override
    public Image getImage()
    {

```

```

        if (image == null)
        {
            image = DebugUITools.getImage(
                IDebugUIConstants.IMG_VIEW_VARIABLES);
        }
        return image;
    }
}

```

### H.2.3 File: GringoLaunchConfigurationDelegate.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.gringo.launch;

import java.lang.reflect.Field;
import java.util.LinkedList;
import java.util.List;

```

```

import org.eclipse.core.runtime.preferences.
    IEclipsePreferences;
import org.eclipse.core.runtime.preferences.InstanceScope
    ;
import org.eclipse.debug.core.ILaunchConfiguration;

import uk.ac.bath.cs.asp.ide.gringo.GringoPlugin;
import uk.ac.bath.cs.asp.ide.gringo.launch.constants.
    GringoProgramArgumentConstants;
import uk.ac.bath.cs.asp.ide.gringo.preferences.
    GringoPreferenceConstants;
import uk.ac.bath.cs.asp.ide.launch.
    AbstractLaunchConfigurationDelegate;

/**
 * The delegate that runs gringo. Gets all the arguments
 * and what the user chose
 * from the run configuration then builds the command
 * line before running in a
 * separate process.
 *
 * @author Rob Ibbotson
 */
public class GringoLaunchConfigurationDelegate extends
    AbstractLaunchConfigurationDelegate
{
    @Override
    public List<String> getProgramArguments(
        ILaunchConfiguration configuration)
    {
        List<String> arguments = new LinkedList<String>()
            ;
        // add the input files
        arguments.addAll(getInputFiles(configuration));

        GringoProgramArgumentConstants gringoConstants =
            new GringoProgramArgumentConstants();

        for (Field field : gringoConstants.getClass().
            getFields())
        {
            arguments.addAll(getFieldValue(configuration,
                field, gringoConstants));
        }
        return arguments;
    }
}

```

```

    }

    @Override
    public String getProgramLocation()
    {
        IEclipsePreferences node = new InstanceScope().
            getNode(GringoPlugin.PLUGIN_ID);
        final String program = node.get(
            GringoPreferenceConstants.GRINGO_PROGRAM_PATH
            , null);
        if (program == null)
        {
            super.programNotSetErrorMessage("gringo");
        }
        return program;
    }
}

```

## H.2.4 File: GringoLaunchConfigurationTabGroup.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,

```

```

    * Fifth Floor, Boston, MA 02110-1301, USA.
    *****/
package uk.ac.bath.cs.asp.ide.gringo.launch;

import org.eclipse.debug.ui.
    AbstractLaunchConfigurationTabGroup;
import org.eclipse.debug.ui.CommonTab;
import org.eclipse.debug.ui.ILaunchConfigurationDialog;
import org.eclipse.debug.ui.ILaunchConfigurationTab;

import uk.ac.bath.cs.asp.ide.launch.
    InputFilesConfigurationTab;

/**
 * The tabs in the gringo run configuration
 *
 * @author Rob Ibbotson
 */
public class GringoLaunchConfigurationTabGroup extends
    AbstractLaunchConfigurationTabGroup
{
    @Override
    public void createTabs(ILaunchConfigurationDialog
        dialog, String mode)
    {
        ILaunchConfigurationTab[] tabs = new
            ILaunchConfigurationTab[] { new
                InputFilesConfigurationTab(),
                new GringoArgumentsTab(), new CommonTab()
            };
        setTabs(tabs);
    }
}

```

## H.2.5 File: GringoLaunchShortcut.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *

```

```

 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.gringo.launch;

```

```

import uk.ac.bath.cs.asp.ide.launch.
    AbstractLaunchShortcut;

/**
 * Launch shortcut for gringo so it can be launched from
 * the editor by right
 * clicking
 *
 * @author Rob Ibbotson
 */
public class GringoLaunchShortcut extends
    AbstractLaunchShortcut
{
    public GringoLaunchShortcut()
    {
        super("uk.ac.bath.cs.asp.ide.gringo.launch.
            GringoLaunchConfigurationType");
    }
}

```

## H.2.6 File: GringoPreferenceConstants.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.gringo.preferences;

/**
 * Constant for the plug-in preferences. Stored in the
 * preference store under
 * this id.
 *
 * @author Rob Ibbotson
 */
public class GringoPreferenceConstants
{
    public static final String GRINGO_PROGRAM_PATH = "
        gringoProgramPathPreference";
}

```

## H.2.7 File: GringoPreferencePage.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.gringo.preferences;

import org.eclipse.jface.preference.
    FieldEditorPreferencePage;
import org.eclipse.jface.preference.FileFieldEditor;
import org.eclipse.ui.IWorkbench;
import org.eclipse.ui.IWorkbenchPreferencePage;

import uk.ac.bath.cs.asp.ide.gringo.GringoPlugin;

/**
 * Gringo preference page for the program location
 *
 * @author Rob Ibbotson
 */
public class GringoPreferencePage extends
    FieldEditorPreferencePage implements
    IWorkbenchPreferencePage
{

```

```

public GringoPreferencePage()
{
    super(GRID);
    setPreferenceStore(GringoPlugin.getDefault().
        getPreferenceStore());
}

public void createFieldEditors()
{
    addField(new FileFieldEditor(
        GringoPreferenceConstants.GRINGO_PROGRAM_PATH
        , "Program Location",
        getFieldEditorParent()));
}

public void init(IWorkbench workbench)
{
}
}

```

## H.3 Conversion

This section shows some of the classes responsible for the conversion between the programs. It provides the functionality to convert from LPARSE to Gringo and Gringo to LPARSE.

### H.3.1 File: AbstractProgramConverter.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *

```

```

 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import java.util.List;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.texteditor.ITextEditor;

import uk.ac.bath.cs.asp.ide.ASPPlugin;

/**
 * Provides many functions for the converters that
 * extends this
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractProgramConverter
{
    public abstract void convert(ITextEditor editor);

    protected Matcher getMatcher(String regex, String
        toBeSearched)
    {
        Pattern pattern = Pattern.compile(regex);
        return pattern.matcher(toBeSearched);
    }

    protected String getDocumentString(int offset, int
        end, IDocument document)
    {

```

```

String string = "";
try
{
    string = document.get(offset, end);
}
catch (BadLocationException e)
{
    ASPPlugin.getLogger().logException(e);
}
return string;
}

protected void replaceInDocument(int offset, int
length, String replacement, IDocument document)
{
    try
    {
        document.replace(offset, length, replacement)
            ;
    }
    catch (BadLocationException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
}

protected int getLineStart(IDocument document, int
offset) throws BadLocationException
{
    while (offset >= 0 && document.getChar(offset) !=
'\n')
    {
        offset--;
    }
    // Don't want to return \n want the one before
    // nor return -1 want 0
    return offset + 1;
}

protected int getLineEnd(IDocument document, int
offset) throws BadLocationException
{
    while (offset < document.getLength() && document.
getChar(offset) != '\n')
    {

```

```

        offset++;
    }
    return offset;
}

protected void replaceWithFull(List<String>
replacements, IDocument document, int startPos,
int endPos)
{
    StringBuffer buffer = new StringBuffer();
    int startOfLine = startPos;
    int endOfLine = endPos;
    String previous = "";
    String suffix = "";
    try
    {
        startOfLine = getLineStart(document, startPos
        );
        endOfLine = getLineEnd(document, endPos);
        previous = document.get(startOfLine, startPos
        - startOfLine);
        suffix = document.get(endPos, endOfLine -
        endPos);
    }
    catch (BadLocationException e)
    {
        ASPPlugin.getLogger().logException(e);
    }
    for (String replace : replacements)
    {
        String newString = previous + replace +
            suffix + System.getProperty("line.
            separator");
        buffer.append(newString);
    }
    replaceInDocument(startOfLine, endOfLine -
startOfLine, buffer.toString(), document);
}
}
}

```

### H.3.2 File: AbstractProgramConverterAction.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import org.eclipse.jface.action.Action;
import org.eclipse.ui.texteditor.ITextEditor;

/**
 * Abstract class to share common functions
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractProgramConverterAction
    extends Action
{
    protected ITextEditor editor;
    private AbstractProgramConverter[] converters;

    public AbstractProgramConverterAction(String name,
        AbstractProgramConverter[] converters)
    {
        super(name);

```

```

        setEnabled(false);
        this.converters = converters;
    }

    public void setEditor(ITextEditor editor)
    {
        this.editor = editor;
        setEnabled(editor != null);
    }

    @Override
    public void run()
    {
        super.run();
        for (AbstractProgramConverter conv : converters)
        {
            conv.convert(editor);
        }
    }
}

```

### H.3.3 File: AggregateLparseToGringoConverter.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *

```



```

* You should have received a copy of the GNU General
  Public License along with this
* program; if not, write to the Free Software Foundation
  , Inc., 51 Franklin Street,
* Fifth Floor, Boston, MA 02110-1301, USA.
*****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import java.util.regex.Matcher;

import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.texteditor.ITextEditor;

/**
 * This is used to convert from lparse to gringo.
 * Aggregation is done
 * differently as described in the gringo manual. This
 * class expands it out so
 * that it can be understood by both.
 *
 * @author Rob Ibbotson
 */
public class AggregateLparseToGringoConverter extends
  AbstractProgramConverter
{
  private static final String startRegex = "\\d+[{]"
  ;
  private static final String atom = "( )*\\(\\w+\\)"
  +( )*\\(\\w+\\)";
  private static final String optionalAtoms = "(, " +
  atom + ")*";
  private static final String endRegex = "[]]";

  @Override
  public void convert(ITextEditor editor)
  {
    IDocument document = editor.getDocumentProvider()
      .getDocument(editor.getEditorInput());
    Matcher matcher = getMatcher(getFullRegex(),
      getDocumentString(0, document.getLength(),
        document));
    while (matcher.find())
    {
      String replacement = buildReplacement(matcher
        .group());

```

```

      replaceInDocument(matcher.start(), matcher.
        group().length(), replacement, document);
      matcher = getMatcher(getFullRegex(),
        getDocumentString(0, document.getLength(),
          document));
    }
  }

  private String buildReplacement(String group)
  {
    StringBuffer newString = new StringBuffer();
    Matcher matcher = getMatcher(startRegex, group);
    String found = getStringFromMatcher(matcher);
    newString.append(found.substring(0, found.length()
      () - 1));
    newString.append(" ");

    matcher = getMatcher("[{].+[}]\"", group);
    found = getStringFromMatcher(matcher);
    matcher = getMatcher(atom, found);

    boolean notFirst = false;
    while (matcher.find())
    {
      if (notFirst)
      {
        newString.append(",");
      }
      newString.append(matcher.group());
      newString.append("=1");
      notFirst = true;
    }
    newString.append("]");
    return newString.toString();
  }

  private String getStringFromMatcher(Matcher matcher)
  {
    matcher.find();
    return matcher.group();
  }

  private String getFullRegex()
  {

```

```

        return startRegex + atom + "( )*" + optionalAtoms
        + endRegex;
    }
}

```

### H.3.4 File: PoolingGringoToLparseConverter.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;

import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.texteditor.ITextEditor;

/**

```

```

 * Used to convert from gringo to lparse. There is a
 * difference in pooling
 * between the two. p(X,Y;Z) in gringo is p(X,Z) p(X,Y)
 * whereas in lparse it is
 * p(X,Y) p(Z). This expands out the rules so that they
 * can then be understand
 * by both grounders in the same manner. This converter
 * converts from p(X,Y;Z)
 * to the form p(X,Y) p(X,Z).
 *
 * @author Rob Ibbotson
 */
public class PoolingGringoToLparseConverter extends
    AbstractProgramConverter
{
    private static final String startRegex = "[a-zA
-Z]+\\(";
    private static final String variable = "[A-Z
]+";
    private static final String variableSpaced = "( )*"
        + variable + "( )*";
    private static final String combinedVariable =
        variableSpaced + "[,;]" + variableSpaced + "+";
    private static final String endRegex = "\\)";

    @Override
    public void convert(ITextEditor editor)
    {
        IDocument document = editor.getDocumentProvider()
            .getDocument(editor.getEditorInput());
        Matcher matcher = getMatcher(fullRegex(),
            getDocumentString(0, document.getLength(),
            document));
        while (matcher.find())
        {
            List<String> replacement = buildReplacement(
                matcher.group());
            if (replacement.size() > 0)
            {
                replaceWithFull(replacement, document,
                    matcher.start(), matcher.end());
                matcher = getMatcher(fullRegex(),
                    getDocumentString(0, document.
                    getLength(), document));
            }
        }
    }
}

```

```

    }
}

private String fullRegex()
{
    return startRegex + combinedVariable + endRegex;
}

private List<String> buildReplacement(String group)
{
    List<String> newStrings = new ArrayList<String>()
        ;
    String start = getStart(group);

    // Going to bruce forsyth it
    List<String> constants = new ArrayList<String>();

    // Find the constant first
    constants.addAll(getRegex("\\(", " ", group));
    constants.addAll(getRegex(", ", " ", group));
    constants.addAll(getRegex(", ", "\\)", group));
    constants.addAll(getRegex("\\(", "\\)", group));

    List<String> arguments = new ArrayList<String>();
    arguments.addAll(getRegex("\\(", " ", group));
    arguments.addAll(getRegex(", ", " ", group));
    arguments.addAll(getRegex(", ", " ", group));
    arguments.addAll(getRegex(", ", " ", group));
    arguments.addAll(getRegex(", ", "\\)", group));

    // Now have every constant but with a different
    // arg each time
    String constantString = "";
    for (String constant : constants)
    {
        constantString += constant + ", ";
    }

    for (String arg : arguments)
    {
        String argString = constantString;
        argString += arg + " ";
        newStrings.add(start + argString);
    }
}

```

```

        return newStrings;
    }

private String getStart(String group)
{
    Matcher matcher = getMatcher(startRegex, group);
    matcher.find();
    return matcher.group();
}

private List<String> getRegex(String previous, String
    end, String group)
{
    List<String> found = new ArrayList<String>();
    String regex = previous + variableSpaced + end;
    Matcher matcher = getMatcher(regex, group);
    int start = 0;
    while (matcher.find(start))
    {
        String matched = matcher.group();
        matched = matched.substring(1, matched.length
            () - 1);
        found.add(matched);
        start = matcher.end() - 1;
    }
    return found;
}
}

```

### H.3.5 File: PoolingLparseToGringoConverter.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.

```

```

*
* This program is distributed in the hope that it will
* be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of
* MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public
* License for more details.
*
* You should have received a copy of the GNU General
* Public License along with this
* program; if not, write to the Free Software Foundation
* , Inc., 51 Franklin Street,
* Fifth Floor, Boston, MA 02110-1301, USA.
*****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import java.util.ArrayList;
import java.util.List;
import java.util.regex.Matcher;

import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.texteditor.ITextEditor;

/**
 * Used to convert from lparse to gringo. There is a
 * difference in pooling
 * between the two. p(X,Y;Z) in gringo is p(X,Z) p(X,Y)
 * whereas in lparse it is
 * p(X,Y) p(Z). This expands out the rules so that they
 * can then be understand
 * by both grounders in the same manner. This converter
 * converts from p(X,Y;Z)
 * to the form p(X,Y) p(Z).
 *
 * @author Rob Ibbotson
 */
public class PoolingLparseToGringoConverter extends
AbstractProgramConverter
{
    private static final String startRegex = "[a-zA
-Z]+\\\\";
    private static final String variable = "[A-Z
]+";
    private static final String variableSpaced = "( )*"
+ variable + "( )*";

```

```

private static final String combinedVariable =
variableSpaced + "(" + variableSpaced + ")*";
private static final String semiColon =
combinedVariable + ";" + combinedVariable + "+";
;
private static final String endRegex = "\\)";

@Override
public void convert(ITextEditor editor)
{
    IDocument document = editor.getDocumentProvider()
.getDocument(editor.getEditorInput());
    Matcher matcher = getMatcher(fullRegex(),
getDocumentString(0, document.getLength(),
document));
    while (matcher.find())
    {
        List<String> replacement = buildReplacement(
matcher.group());
        replaceWithFull(replacement, document,
matcher.start(), matcher.end());
        matcher = getMatcher(fullRegex(),
getDocumentString(0, document.getLength(),
document));
    }
}

private String fullRegex()
{
    return startRegex + semiColon + endRegex;
}

private List<String> buildReplacement(String group)
{
    List<String> newStrings = new ArrayList<String>()
;
    String start = getStart(group);
    Matcher matcher = getMatcher(combinedVariable +
";", group);
    String found = "";
    while (matcher.find())
    {
        found = matcher.group();
    }
}

```

```

        found = found.substring(0, found.length() -
            1).trim();
        newStrings.add(start + found + ")");
    }

    // For the final case different reg expression
    matcher = getMatcher(";"+ combinedVariable + "
        \\)", group);
    matcher.find();
    found = matcher.group();
    found = found.substring(1, found.length() - 1).
        trim();
    newStrings.add(start + found + ")");
    return newStrings;
}

private String getStart(String group)
{
    Matcher matcher = getMatcher(startRegex, group);
    matcher.find();
    return matcher.group();
}
}

```

### H.3.6 File: RemoveRestrictedNames.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.

```

```

 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.conversion;

import java.util.regex.Matcher;

import org.eclipse.jface.text.IDocument;
import org.eclipse.ui.texteditor.ITextEditor;

/**
 * Used when converting from gringo to lparse. Lparse
 * does not accept the
 * restricted names below so this class simply appends a
 * - at the beginning of
 * each occurrence of the restricted names.
 *
 * @author Rob Ibbotson
 */
public class RemoveRestrictedNames extends
    AbstractProgramConverter
{
    private String [] restrictedNames = new String [] { "
        abs", "and", "assign", "bnot", "div", "eq", "ge",
        "gt", "le",
        "lt", "minus", "mod", "neq", "or", "plus", "
        times", "xor" };

    @Override
    public void convert(ITextEditor editor)
    {
        IDocument document = editor.getDocumentProvider()
            .getDocument(editor.getEditorInput());
        for (String regex : restrictedNames)
        {
            String fullRegex = "\\s" + regex + "[ \\(.)";
            Matcher matcher = getMatcher(fullRegex,
                getDocumentString(0, document.getLength()
                    , document));
            while (matcher.find())
            {

```



```

public List<ICompletionProposal>
    generatePredictedProposals(LparseSourceFile file ,
        String prefix , int offset ,
        boolean hasBodySeperator)
{
    List<ICompletionProposal> proposals = new
        ArrayList<ICompletionProposal>();
    proposals.addAll(makeProposals(super.
        getPrefixNamedAtoms(prefix , file) , prefix ,
        offset));
    return proposals;
}

```

```

@Override
public List<ICompletionProposal>
    generateCurrentProposals(LparseSourceFile file ,
        String prefix , int offset ,
        boolean hasBodySeperator)
{
    List<ICompletionProposal> proposals = new
        ArrayList<ICompletionProposal>();
    // Add itself
    if (!prefix.equals(""))
    {
        proposals.add(new CompletionProposal(prefix ,
            offset - prefix.length() , prefix.length()
            , prefix.length()
            + offset));
    }
    return proposals;
}

```

```

private List<ICompletionProposal> makeProposals(
    String[] symbols , String prefix , int offset)
{
    final List<ICompletionProposal> proposals = new
        ArrayList<ICompletionProposal>();
    for (String symbol : symbols)
    {
        // Don't add the prefix as will be added by
        // current proposals
        if (!symbol.equals(prefix))
        {
            final ICompletionProposal proposal = new
                CompletionProposal(symbol , offset -

```

```

        prefix.length() , prefix
        .length() , prefix.length() +
        offset);
        proposals.add(proposal);
    }
}
return proposals;
}
}

```

## H.4.2 File: IProposalGenerator.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor , Boston , MA 02110-1301 , USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions;

import java.util.List;

import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

```

```

import uk.ac.bath.cs.asp.ide.lparse.core.ast .
    LparseSourceFile;

/**
 * What proposal generators implements. The predicted are
 * from previously typed
 * predicates. The current are based on what the user has
 * just typed.
 *
 * @author Rob Ibbotson
 */
public interface IProposalGenerator
{
    public List<ICompletionProposal>
        generatePredictedProposals(LparseSourceFile file ,
            String prefix , int offset ,
            boolean hasBodySeperator);

    public List<ICompletionProposal>
        generateCurrentProposals(LparseSourceFile file ,
            String prefix , int offset ,
            boolean hasBodySeperator);
}

```

### H.4.3 File: LparseContentAssistProcessor.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A

```

```

 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor , Boston , MA 02110-1301 , USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor .
    actions;

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

import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.ITextViewer;
import org.eclipse.jface.text.contentassist .
    ICompletionProposal;
import org.eclipse.jface.text.contentassist .
    IContentAssistProcessor;
import org.eclipse.jface.text.contentassist .
    IContextInformation;
import org.eclipse.jface.text.contentassist .
    IContextInformationValidator;

import uk.ac.bath.cs.asp.ide.ASPPlugin;
import uk.ac.bath.cs.asp.ide.lparse.core.ast .LparseParser
    ;
import uk.ac.bath.cs.asp.ide.lparse.core.ast .
    LparseSourceFile;

/**
 * Provides the implementation for the auto completion
 * proposals for lparse
 * source files. The main setup for this is provided in
 * the
 * LparseSourceViewerConfiguration file .
 *
 * @author Rob Ibbotson
 */
public final class LparseContentAssistProcessor
    implements IContentAssistProcessor
{

```



```

private static List<IProposalGenerator> generators =
    new ArrayList<IProposalGenerator>();
static
{
    generators.add(new AtomProposalGenerator());
    generators.add(new
        NumericConstantProposalGenerator());
    generators.add(new SymbolProposalGenerator());
    generators.add(new VariableProposalGenerator());
}

public LparseContentAssistProcessor()
{
}

public ICompletionProposal[] computeProposals(
    IDocument document, int offset, List<
    IProposalGenerator> generators)
{
    final List<ICompletionProposal> proposals = new
        ArrayList<ICompletionProposal>();
    final LparseSourceFile sourceFile = new
        LparseParser().parse(document);
    final String prefix = lastWord(document, offset);
    final boolean hasBodySeperator =
        lineContainsBodySeperator(document, offset);
    for (IProposalGenerator generator : generators)
    {
        proposals.addAll(generator.
            generatePredictedProposals(sourceFile,
                prefix, offset, hasBodySeperator));
    }
    for (IProposalGenerator generator : generators)
    {
        proposals.addAll(generator.
            generateCurrentProposals(sourceFile,
                prefix, offset, hasBodySeperator));
    }
    return proposals.toArray(new ICompletionProposal[
        proposals.size()]);
}

@Override

```

```

public ICompletionProposal[]
    computeCompletionProposals(ITextViewer viewer,
        int offset)
{
    return computeProposals(viewer.getDocument(),
        offset, generators);
}

public static String lastWord(final IDocument
    document, final int offset)
{
    try
    {
        for (int n = offset - 1; n >= 0; n--)
        {
            final char c = document.getChar(n);
            if (!Character.isJavaIdentifierPart(c))
            {
                return document.get(n + 1, offset - n
                    - 1);
            }
        }
    }
    catch (BadLocationException e)
    {
        // Bad location return ""
        ASPPlugin.getLogger().logException(e);
    }
    return "";
}

public static boolean lineContainsBodySeperator(final
    IDocument document, int offset)
{
    try
    {
        for (int n = offset - 2; n > 0; n--)
        {
            final char c = document.getChar(n);
            final char c1 = document.getChar(n + 1);
            if (c == ':' && c1 == '-')
            {
                return true;
            }
        }
        if (c == '\n' || c1 == '\n')

```

```

        {
            return false;
        }
    }
}
catch (BadLocationException e)
{
    ASPPlugin.getLogger().logException(e);
}
return false;
}

@Override
public IContextInformation[]
    computeContextInformation(ITextViewer viewer, int
        offset)
{
    return null;
}

@Override
public char[]
    getCompletionProposalAutoActivationCharacters()
{
    return null;
}

@Override
public char[]
    getContextInformationAutoActivationCharacters()
{
    return null;
}

@Override
public IContextInformationValidator
    getContextInformationValidator()
{
    return null;
}

@Override
public String getErrorMessage()
{
    return null;
}

```

```

    }
}

```

#### H.4.4 File: NumericConstantProposalGenerator.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions;

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

import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;

```

```

/**
 * Generates content assist proposals from numeric
 * constants.
 *
 * @author Rob Ibbotson
 */
public class NumericConstantProposalGenerator extends
    ProposalGenerator
{
    @Override
    public List<ICompletionProposal>
        generatePredictedProposals(LparseSourceFile file ,
            String prefix , int offset ,
            boolean hasBodySeperator)
    {
        if (!hasBodySeperator)
        {
            return super.
                generateConstantPredictedProposals(super.
                    getPrefixNamedAtoms(prefix , file) , file
                    .getNumericConstantTable().getSymbols
                    () , prefix , offset);
        }
        return new ArrayList<ICompletionProposal>();
    }

    @Override
    public List<ICompletionProposal>
        generateCurrentProposals(LparseSourceFile file ,
            String prefix , int offset ,
            boolean hasBodySeperator)
    {
        if (!hasBodySeperator)
        {
            return super.generateConstantCurrentProposals
                (super.getPrefixNamedAtoms(prefix , file) ,
                    file
                    .getNumericConstantTable().getSymbols
                    () , prefix , offset);
        }
        return new ArrayList<ICompletionProposal>();
    }
}

```

## H.4.5 File: ProposalGenerator.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor , Boston , MA 02110-1301 , USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions;

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

import org.eclipse.jface.text.contentassist.
    CompletionProposal;
import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;

/**
 * Proposal generator class. Provides some common helpful
 * methods for the
 * content assist generators

```

```

*
* @author Rob Ibbotson
*/
public abstract class ProposalGenerator implements
    IProposalGenerator
{
    protected String [] getPrefixNamedAtoms(String prefix ,
        LparseSourceFile file)
    {
        final List<String> common = new ArrayList<String>
            >();
        for (String symbol : file.getAtomTable().
            getSymbols())
        {
            if (symbol.startsWith(prefix))
            {
                common.add(symbol);
            }
        }
        return common.toArray(new String[common.size()]);
    }

    protected List<ICompletionProposal>
        generateConstantPredictedProposals(String []
        prefixNamedAtoms, String [] symbols,
        String prefix, int offset)
    {
        List<ICompletionProposal> proposals = new
            ArrayList<ICompletionProposal>();

        for (String common : prefixNamedAtoms)
        {
            if (!common.equals(prefix))
            {
                proposals.addAll(makeFullBodyProposal(
                    symbols, prefix, common, offset));
            }
        }

        return proposals;
    }

    protected List<ICompletionProposal>
        generateConstantCurrentProposals(String []
        prefixNamedAtoms, String [] symbols,
        String prefix, int offset)
    {
        List<ICompletionProposal> proposals = new
            ArrayList<ICompletionProposal>();

        if (!prefix.equals(""))
        {
            proposals.addAll(makeFullBodyProposal(symbols
                , prefix, prefix, offset));
        }
        return proposals;
    }

    protected List<ICompletionProposal>
        makeFullHeadProposal(String [] symbols, String
        prefix, String replacement,
        int offset)
    {
        return makeFullProposal(symbols, prefix,
            replacement, offset, ")");
    }

    protected List<ICompletionProposal>
        makeFullBodyProposal(String [] symbols, String
        prefix, String replacement,
        int offset)
    {
        return makeFullProposal(symbols, prefix,
            replacement, offset, ").");
    }

    protected List<ICompletionProposal> makeFullProposal(
        String [] symbols, String prefix, String
        replacement,
        int offset, String suffix)
    {
        final List<ICompletionProposal> proposals = new
            ArrayList<ICompletionProposal>();
        for (String symbol : symbols)
        {
            String fullReplacement = replacement + "(" +
                symbol + suffix;
            int replacementOffset = offset - prefix.
                length();
            int replacementLength = prefix.length();

```

```

        int cursor = replacementOffset +
            replacementLength;
        proposals.add(new CompletionProposal(
            fullReplacement, replacementOffset,
            replacementLength, cursor));
    }
    return proposals;
}
}
}

```

#### H.4.6 File: SymbolProposalGenerator.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions;

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

```

```

import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;

/**
 * Proposals based on symbols
 *
 * @author Rob Ibbotson
 */
public class SymbolProposalGenerator extends
    ProposalGenerator
{
    @Override
    public List<ICompletionProposal>
        generatePredictedProposals(LparseSourceFile file,
            String prefix, int offset,
            boolean hasBodySeperator)
    {
        if (!hasBodySeperator)
        {
            return super.
                generateConstantPredictedProposals(super.
                    getPrefixNamedAtoms(prefix, file), file
                        .getSymbolicConstantTable().
                            getSymbols(), prefix, offset);
        }
        return new ArrayList<ICompletionProposal>();
    }

    @Override
    public List<ICompletionProposal>
        generateCurrentProposals(LparseSourceFile file,
            String prefix, int offset,
            boolean hasBodySeperator)
    {
        if (!hasBodySeperator)
        {
            return super.generateConstantCurrentProposals
                (super.getPrefixNamedAtoms(prefix, file),
                    file
                        .getSymbolicConstantTable().
                            getSymbols(), prefix, offset);
        }
    }
}

```

```

        }
        return new ArrayList<ICompletionProposal>();
    }
}

```

#### H.4.7 File: VariableProposalGenerator.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions;

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

import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;

/**

```

```

 * Proposals based on variables
 *
 * @author Rob Ibbotson
 */
public class VariableProposalGenerator extends
    ProposalGenerator
{
    @Override
    public List<ICompletionProposal>
        generatePredictedProposals(LparseSourceFile file ,
            String prefix , int offset ,
            boolean hasBodySeperator)
    {
        List<ICompletionProposal> proposals = new
            ArrayList<ICompletionProposal>();
        String [] symbols = file.getVariableTable().
            getSymbols();

        for (String common : super.getPrefixNamedAtoms(
            prefix , file))
        {
            if (!common.equals(prefix))
            {
                if (!hasBodySeperator)
                {
                    proposals.addAll(makeFullHeadProposal
                        (symbols , prefix , common , offset)
                    );
                    proposals.addAll(makeFullProposal(
                        symbols , prefix , common , offset ,
                        " :-" ));
                }
                else
                {
                    proposals.addAll(makeFullBodyProposal
                        (symbols , prefix , common , offset)
                    );
                }
            }
        }

        return proposals;
    }

    @Override

```

```

public List<ICompletionProposal>
    generateCurrentProposals(LparseSourceFile file ,
        String prefix , int offset ,
        boolean hasBodySeperator)
{
    List<ICompletionProposal> proposals = new
        ArrayList<ICompletionProposal>();
    String [] symbols = file.getVariableTable().
        getSymbols();
    if (!prefix.equals(""))
    {
        if (!hasBodySeperator)
        {
            proposals.addAll(makeFullHeadProposal(
                symbols , prefix , prefix , offset));
            proposals.addAll(makeFullProposal(symbols
                , prefix , prefix , offset , ") :-"));
        }
        else
        {
            proposals.addAll(makeFullBodyProposal(
                symbols , prefix , prefix , offset));
        }
    }
    return proposals;
}
}

```

#### H.4.8 File: PredicateView.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY

```

```

 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc. , 51 Franklin Street ,
 * Fifth Floor , Boston , MA 02110-1301 , USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.views;

import org.eclipse.jface.action.Action;
import org.eclipse.jface.action.IMenuManager;
import org.eclipse.jface.text.BadLocationException;
import org.eclipse.jface.text.IDocument;
import org.eclipse.jface.text.ITextSelection;
import org.eclipse.jface.viewers.DoubleClickEvent;
import org.eclipse.jface.viewers.IDoubleClickListener;
import org.eclipse.jface.viewers.ISelection;
import org.eclipse.jface.viewers.IStructuredSelection;
import org.eclipse.jface.viewers.TreeViewer;
import org.eclipse.jface.viewers.ViewerFilter;
import org.eclipse.swt.SWT;
import org.eclipse.swt.widgets.Composite;
import org.eclipse.ui.PlatformUI;
import org.eclipse.ui.part.ViewPart;

```

```

import uk.ac.bath.cs.asp.ide.ASPPlugin;
import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    LparseEditor;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.LparseContentAssistProcessor;

/**
 * A view showing the outline of predicates. These
 * predicates can be double
 * clicked to be inserted into the editor. It is an
 * alternative to content
 * assist with the possibilities all being represented in
 * a tree. The class makes

```

```

* use of the ContentAssistProcessors in order to get all
  the proposals easily
* Filtering is applied to allow the user to choose
  whether to show predicates
* from (a) previously typed e.g. a different line or (b)
  from what they have
* just typed.
*
* @author Rob Ibbotson
*/
public class PredicateView extends ViewPart
{
    public static final String PREDICATE_VIEW_ID = "uk.ac
        .bath.cs.asp.ide.lparse.views.PredicateView";

    private TreeViewer      viewer          = null;
    private Action          doubleClickAction = null;

    private Action          currentFilterAction;
    private Action          predictedFilterAction;
    private Action          autoExpandAction;
    private ViewerFilter    currentFilter    = new
        PredicateViewCurrentFilter();
    private ViewerFilter    predictedFilter  = new
        PredicateViewPredictedFilter();

    public PredicateView()
    {
    }

    public void createPartControl(Composite parent)
    {
        viewer = new TreeViewer(parent, SWT.SINGLE | SWT.
            V_SCROLL | SWT.H_SCROLL);
        viewer.setContentProvider(new
            PredicateViewContentProvider());
        viewer.setLabelProvider(new
            PredicateViewLabelProvider());
        viewer.expandAll();
        makeActions();
        createMenu();
        hookDoubleClickAction();
    }

    private void createMenu()
    {
        IMenuManager rootMenuManager = getViewSite().
            getActionBars().getMenuManager();
        rootMenuManager.add(currentFilterAction);
        rootMenuManager.add(predictedFilterAction);
        rootMenuManager.add(autoExpandAction);
    }

    private void makeActions()
    {
        makeDoubleClickAction();
        makeFilterActions();
        makeAutoExpandAction();
    }

    private void makeAutoExpandAction()
    {
        autoExpandAction = new Action("Auto-expand")
        {
            {
            };
            autoExpandAction.setChecked(true);
        }
    }

    private void makeFilterActions()
    {
        currentFilterAction = new Action("Do not suggest
            from previous predicates")
        {
            public void run()
            {
                updateFilter(currentFilterAction);
            }
        };
        currentFilterAction.setChecked(false);

        predictedFilterAction = new Action("Do not show
            from current")
        {
            public void run()
            {
                updateFilter(predictedFilterAction);
            }
        };
        predictedFilterAction.setChecked(false);
    }
}

```



```

private void updateFilter(Action action)
{
    if (action == currentFilterAction)
    {
        adjustFilterState(action.isChecked(),
            currentFilter);
    }

    if (action == predictedFilterAction)
    {
        adjustFilterState(action.isChecked(),
            predictedFilter);
    }
}

private void adjustFilterState(boolean checked,
    ViewerFilter filter)
{
    if (checked)
    {
        viewer.addFilter(filter);
    }
    else
    {
        viewer.removeFilter(filter);
    }
}

private void makeDoubleClickAction()
{
    doubleClickAction = new Action()
    {
        public void run()
        {
            ISelection selection = viewer.
                getSelection();
            Object obj = ((IStructuredSelection)
                selection).getFirstElement();
            try
            {
                String lastWord =
                    LparseContentAssistProcessor.
                        lastWord(getDocumentInEditor(),
                            getOffsetInEditor());
            }
            catch (BadLocationException e)
            {
                ASPPlugin.getLogger().logException(e);
            }
        }
    };
}

private void hookDoubleClickAction()
{
    viewer.addDoubleClickListener(new
        IDoubleClickListener()
    {
        public void doubleClick(DoubleClickEvent
            event)
        {
            doubleClickAction.run();
        }
    });
}

public void setFocus()
{
    viewer.getControl().setFocus();
}

public void modelUpdated(LparseSourceFile model)
{
    Object[] expandedElements = viewer.
        getExpandedElements();
    // Get the last word typed
    String lastWord = LparseContentAssistProcessor.
        lastWord(getDocumentInEditor(),
            getOffsetInEditor());
    boolean hasSeparator =
        LparseContentAssistProcessor.
            lineContainsBodySeparator(getDocumentInEditor()
                ,
                getOffsetInEditor());
}

```

```

viewer.setInput(PredicateViewItem.getInput(model,
    lastWord, getOffsetInEditor(), hasSeparator)
);

if (autoExpandAction.isChecked())
{
    viewer.expandAll();
}
else
{
    viewer.setExpandedElements(expandedElements);
}
}

private int getOffsetInEditor()
{
    LparseEditor activeEditor = (LparseEditor)
        PlatformUI.getWorkbench().
        getActiveWorkbenchWindow().getPages()[0]
        .getActiveEditor();
    ITextSelection textSelection = (ITextSelection)
        activeEditor.getSelectionProvider().
        getSelection();
    return textSelection.getOffset();
}

private IDocument getDocumentInEditor()
{
    LparseEditor activeEditor = (LparseEditor)
        PlatformUI.getWorkbench().
        getActiveWorkbenchWindow().getPages()[0]
        .getActiveEditor();
    return activeEditor.getDocumentProvider().
        getDocument(activeEditor.getEditorInput());
}
}

H.4.9 File: PredicateViewContentProvider.java

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 */

viewer.setInput(PredicateViewItem.getInput(model,
    lastWord, getOffsetInEditor(), hasSeparator)
);

if (autoExpandAction.isChecked())
{
    viewer.expandAll();
}
else
{
    viewer.setExpandedElements(expandedElements);
}
}

private int getOffsetInEditor()
{
    LparseEditor activeEditor = (LparseEditor)
        PlatformUI.getWorkbench().
        getActiveWorkbenchWindow().getPages()[0]
        .getActiveEditor();
    ITextSelection textSelection = (ITextSelection)
        activeEditor.getSelectionProvider().
        getSelection();
    return textSelection.getOffset();
}

private IDocument getDocumentInEditor()
{
    LparseEditor activeEditor = (LparseEditor)
        PlatformUI.getWorkbench().
        getActiveWorkbenchWindow().getPages()[0]
        .getActiveEditor();
    return activeEditor.getDocumentProvider().
        getDocument(activeEditor.getEditorInput());
}
}

package uk.ac.bath.cs.asp.ide.lparse.views;

import org.eclipse.jface.viewers.ArrayContentProvider;
import org.eclipse.jface.viewers.ITreeContentProvider;

/**
 * The content provider for the predicate view. Easy
 * parsing to a predicate view
 * item.
 *
 * @author Rob Ibbotson
 */
public class PredicateViewContentProvider extends
    ArrayContentProvider implements ITreeContentProvider
{
    @Override
    public Object[] getChildren(Object parentElement)
    {
        return ((PredicateViewItem) parentElement).
            getChildren();
    }
    @Override

```

```

    public Object getParent(Object element)
    {
        return ((PredicateViewItem) element).getParent();
    }

    @Override
    public boolean hasChildren(Object element)
    {
        return getChildren(element).length > 0;
    }
}

```

#### H.4.10 File: PredicateViewCurrentFilter.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.views;

import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.viewers.ViewerFilter;

```

```

/**
 * A filter for the predicate view. Only show suggestions
 * based on what the user
 * last typed.
 *
 * @author Rob Ibbotson
 */
public class PredicateViewCurrentFilter extends
    ViewerFilter
{
    @Override
    public boolean select(Viewer viewer, Object
        parentElement, Object element)
    {
        return !((PredicateViewItem) element).
            isPredictedProposal();
    }
}

```

#### H.4.11 File: PredicateViewItem.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *
 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this

```

```

    * program; if not, write to the Free Software Foundation
      , Inc., 51 Franklin Street,
    * Fifth Floor, Boston, MA 02110-1301, USA.
    *****/
package uk.ac.bath.cs.asp.ide.lparse.views;

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

import org.eclipse.jface.text.contentassist.
    CompletionProposal;
import org.eclipse.jface.text.contentassist.
    ICompletionProposal;

import uk.ac.bath.cs.asp.ide.lparse.core.ast.
    LparseSourceFile;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.AtomProposalGenerator;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.IProposalGenerator;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.NumericConstantProposalGenerator;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.SymbolProposalGenerator;
import uk.ac.bath.cs.asp.ide.lparse.internal.editor.
    actions.VariableProposalGenerator;

/**
 * Generates the proposals and stores in a model of {
 *   @link PredicateViewItem}
 *
 * @author Rob Ibbotson
 */
public class PredicateViewItem
{
    private static List<IProposalGenerator>
        nodeGenerators = new ArrayList<IProposalGenerator>
        >();
    private static List<IProposalGenerator>
        leafGenerators = new ArrayList<IProposalGenerator>
        >();

    static
    {
        nodeGenerators.add(new AtomProposalGenerator());

```

```

        leafGenerators.add(new
            NumericConstantProposalGenerator());
        leafGenerators.add(new SymbolProposalGenerator());
        ;
        leafGenerators.add(new VariableProposalGenerator
            ());
    }

    private String                name;
    private boolean               predictedProposal;
    private List<PredicateViewItem> children;
    private PredicateViewItem     parent;

    public PredicateViewItem(String name, List<
        PredicateViewItem> children, PredicateViewItem
        parent, boolean predicted)
    {
        this.name = name;
        this.children = children;
        this.parent = parent;
        this.predictedProposal = predicted;
    }

    public String getName()
    {
        return name;
    }

    public PredicateViewItem[] getChildren()
    {
        return children.toArray(new PredicateViewItem[
            children.size()]);
    }

    public PredicateViewItem getParent()
    {
        return parent;
    }

    public boolean isPredictedProposal()
    {
        return predictedProposal;
    }

```

```

@Override
public String toString()
{
    return getName();
}

@Override
public boolean equals(Object obj)
{
    if (this == obj)
    {
        return true;
    }
    if (!(obj instanceof PredicateViewItem))
    {
        return false;
    }
    PredicateViewItem pvi = (PredicateViewItem) obj;

    if (pvi.getName().equals(this.getName()))
    {
        return true;
    }

    return false;
}

@Override
public int hashCode()
{
    return this.getName().hashCode();
}

public static PredicateViewItem[] getInput(
    LparseSourceFile file, String prefix, int offset,
    boolean hasBodySeperator)
{
    List<PredicateViewItem> nodes = getHeadNodes(file
        , prefix, offset, hasBodySeperator);
    for (PredicateViewItem node : nodes)
    {
        for (IProposalGenerator gen : leafGenerators)
        {

```

```

        for (ICompletionProposal proposal : gen.
            generatePredictedProposals(file, "",
                offset, hasBodySeperator))
        {
            CompletionProposal completionProp = (
                CompletionProposal) proposal;
            if (completionProp.getDisplayString()
                .startsWith(node.name))
            {
                node.children.add(new
                    PredicateViewItem(
                        completionProp.
                            getDisplayString(),
                            new ArrayList<
                                PredicateViewItem>(),
                            node, true));
            }
        }
        for (ICompletionProposal proposal : gen
            .generateCurrentProposals(file,
                prefix, offset,
                hasBodySeperator))
        {
            CompletionProposal completionProp = (
                CompletionProposal) proposal;
            if (completionProp.getDisplayString()
                .startsWith(node.name))
            {
                node.children.add(new
                    PredicateViewItem(
                        completionProp.
                            getDisplayString(),
                            new ArrayList<
                                PredicateViewItem>(),
                            node, false));
            }
        }
    }
    return nodes.toArray(new PredicateViewItem[nodes.
        size()]);
}

```

```

private static List<PredicateViewItem> getHeadNodes(
    LparseSourceFile file, String prefix, int offset,
    boolean hasBodySeparator)
{
    List<PredicateViewItem> nodes = new ArrayList<
        PredicateViewItem>();
    for (IProposalGenerator gen : nodeGenerators)
    {
        for (ICompletionProposal proposal : gen.
            generatePredictedProposals(file, "",
                offset, hasBodySeparator))
        {
            CompletionProposal completionProp = (
                CompletionProposal) proposal;
            nodes.add(new PredicateViewItem(
                completionProp.getDisplayString(),
                new ArrayList<PredicateViewItem>(),
                null, true));
        }

        for (ICompletionProposal proposal : gen.
            generateCurrentProposals(file, prefix,
                offset, hasBodySeparator))
        {
            CompletionProposal completionProp = (
                CompletionProposal) proposal;
            nodes.add(new PredicateViewItem(
                completionProp.getDisplayString(),
                new ArrayList<PredicateViewItem>(),
                null, false));
        }
    }
    return nodes;
}
}
}

```

#### H.4.12 File: PredicateViewLabelProvider.java

```

/*****
 * APE: AnsProlog* Programming Environment plug-in for
 * the Eclipse platform
 * Copyright (C) 2010 Rob Ibbotson
 *

```

```

 * This program is free software; you can redistribute it
 * and/or modify it under the
 * terms of the GNU General Public License as published
 * by the Free Software
 * Foundation; either version 2 of the License, or (at
 * your option) any later version.
 *
 * This program is distributed in the hope that it will
 * be useful, but WITHOUT ANY
 * WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A
 * PARTICULAR PURPOSE. See the GNU General Public
 * License for more details.
 *
 * You should have received a copy of the GNU General
 * Public License along with this
 * program; if not, write to the Free Software Foundation
 * , Inc., 51 Franklin Street,
 * Fifth Floor, Boston, MA 02110-1301, USA.
 *****/
package uk.ac.bath.cs.asp.ide.lparse.views;

import org.eclipse.jface.viewers.LabelProvider;

/**
 * Label provider for predicate view
 *
 * @author Rob Ibbotson
 */
public class PredicateViewLabelProvider extends
    LabelProvider
{
}

```

#### H.4.13 File: PredicateViewPredictedFilter.java

```

package uk.ac.bath.cs.asp.ide.lparse.views;

import org.eclipse.jface.viewers.Viewer;
import org.eclipse.jface.viewers.ViewerFilter;

/**
 * A filter for the predicate view. Only show suggestions
 * based on previous.

```

```

*
* @author Rob Ibbotson
*/
public class PredicateViewPredictedFilter extends
    ViewerFilter
{
    @Override
    public boolean select(Viewer viewer, Object
        parentElement, Object element)
    {
        return ((PredicateViewItem) element).
            isPredictedProposal();
    }
}

```

## H.5 Testing

The code below shows the testing classes that most of the tests extend from. These are the abstract classes that provide most of the functionality.

### H.5.1 File: AbstractRegressionTest.java

```

package uk.ac.bath.cs.asp.ide.testing.base;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.eclipse.swtbot.eclipse.finder.widgets.
    SWTBotEclipseEditor;
import org.junit.Assert;

public abstract class AbstractRegressionTest extends
    CommonTests
{
    protected String testDirectory = null;
    protected String resultsDirectory = null;

```

```

    protected String expectedDirectory = null;

    public AbstractRegressionTest(String program)
    {
        super(program);
        testDirectory = "." + File.separator + "tests" +
            File.separator + program + File.separator;
        resultsDirectory = "." + File.separator + "
            results" + File.separator + "actual" + File.
            separator + program
            + File.separator;
        expectedDirectory = "." + File.separator + "
            results" + File.separator + "expected" + File
            .separator + program
            + File.separator;
    }

    protected void runTests() throws IOException
    {
        super.createNewProject();
        super.createNewFile();
        super.createNewConfiguration();
        String [] files = new File(testDirectory).list();
        setUpConfiguration();
        for (String file : files)
        {
            copyInputFile(file);
            setSaveFile(file);
            // Sleep so it has time to write the file
            BOT.sleep(2000);
        }
        super.deleteConfiguration();
        super.deleteFile();
        super.deleteProject();
    }

    protected void checkResults() throws IOException
    {
        String [] results = new File(resultsDirectory).
            list();
        String [] expected = new File(expectedDirectory).
            list();
        Assert.assertArrayEquals(expected, results);
        // Now go through and check the actual content is
        equal

```

```

for (String res : results)
{
    BufferedReader resultReader = new
        BufferedReader(new FileReader(new File(
            resultsDirectory + res)));
    BufferedReader expectedReader = new
        BufferedReader(new FileReader(new File(
            expectedDirectory + res)));
    String resultLine = "";
    String expectedLine = "";
    while (resultLine != null && expectedLine !=
        null)
    {
        resultLine = resultReader.readLine();
        expectedLine = expectedReader.readLine();
        Assert.assertEquals("File: " + res + ".
            Expected: " + expectedLine + ". Got:
            " + resultLine ,
            expectedLine , resultLine);
    }
    resultReader.close();
    expectedReader.close();
}
}

private void setUpConfiguration()
{
    super.getRunConfiguration();
    BOT.cTabItem("Input Files").activate();
    BOT.button("Add Resource...").click();
    BOT.shell("Resource Selection").activate();
    BOT.text(0).setText(testFileName);
    BOT.table(0).getTableItem(0).select();
    BOT.button("OK").click();

    BOT.cTabItem("Common").activate();
    BOT.checkBox("File:").click();
    BOT.button("Apply").click();
    BOT.button("Close").click();
}

protected void copyInputFile(String file) throws
    IOException
{
    File testFile = new File(testDirectory + file);

```

```

    BufferedReader reader = new BufferedReader(new
        FileReader(testFile));
    String currentLine = "";
    SWTBotEclipseEditor textEditor = BOT.activeEditor
        ().toTextEditor();
    StringBuffer buffer = new StringBuffer();
    while ((currentLine = reader.readLine()) != null)
    {
        buffer.append(currentLine);
        buffer.append(System.getProperty("line .
            separator"));
    }
    reader.close();
    textEditor.setText(buffer.toString());
    textEditor.save();
}

private void setSaveFile(String file) throws
    IOException
{
    String saveFileLocation = resultsDirectory + file
        + ".result";
    File result = new File(saveFileLocation);
    super.getRunConfiguration();
    BOT.cTabItem("Common").activate();
    BOT.text(3).setText(result.getCanonicalPath());
    BOT.checkBox("Launch in background").click();
    BOT.button("Apply").click();
    BOT.button("Run").click();
    BOT.viewByTitle("Console").show();
}
}

```

## H.5.2 File: AbstractTest.java

```

package uk.ac.bath.cs.asp.ide.testing.base;

import org.eclipse.swtbot.eclipse.finder.SWTWorkbenchBot;
import org.eclipse.swtbot.eclipse.finder.widgets.
    SWTBotView;
import org.eclipse.swtbot.swt.finder.utils.
    SWTBotPreferences;
import org.junit.AfterClass;
import org.junit.BeforeClass;

```



```

import uk.ac.bath.cs.asp.ide.clasp.preferences.
    ClaspPreferenceConstants;
import uk.ac.bath.cs.asp.ide.gringo.preferences.
    GringoPreferenceConstants;
import uk.ac.bath.cs.asp.ide.lparse.LparsePlugin;
import uk.ac.bath.cs.asp.ide.lparse.
    LparsePreferenceConstants;
import uk.ac.bath.cs.asp.ide.smodels.SmodelsPlugin;
import uk.ac.bath.cs.asp.ide.smodels.
    SmodelsPreferenceConstants;

/**
 * A very simple abstract test class that enables APE
 * tests to extend from this.
 * Each test extending it should have the annotation
 * @Test
 *
 * @author Rob Ibbotson
 */
public abstract class AbstractTest
{
    protected static SWTWorkbenchBot BOT;

    private static final String    LPARSE_LOCATION = "
        /usr/local/bin/lparse";
    private static final String    SMODELS_LOCATION = "
        /usr/local/bin/smodels";
    private static final String    GRINGO_LOCATION = "
        /usr/local/bin/gringo";
    private static final String    CLASP_LOCATION = "
        /usr/local/bin/clasp";

    protected String                testProjectName;
    protected String                testFileName;
    protected String                testConfigName;
    protected String                program;

    @BeforeClass
    public static void setUp()
    {
        SWTBotPreferences.PLAYBACK_DELAY = 5;
        BOT = new SWTWorkbenchBot();
        for (SWTBotView view : BOT.views())
        {
            if (view.getTitle().equals("Welcome"))
            {
                view.close();
            }
        }
        BOT.perspectiveByLabel("AnsProlog*").activate();
        setUpPreferences();
    }

    @SuppressWarnings("deprecation")
    private static void setUpPreferences()
    {
        LparsePlugin.getDefault().getPluginPreferences().
            setValue(LparsePreferenceConstants.
                LPARSE_PROGRAM_LOCATION,
                LPARSE_LOCATION);
        SmodelsPlugin.getDefault().getPluginPreferences().
            setValue(SmodelsPreferenceConstants.
                SMODELS_PROGRAM_LOCATION,
                SMODELS_LOCATION);
        uk.ac.bath.cs.asp.ide.gringo.GringoPlugin.
            getDefault().getPreferenceStore().setValue(
                GringoPreferenceConstants.
                    GRINGO_PROGRAM_PATH, GRINGO_LOCATION);
        ;
        uk.ac.bath.cs.asp.ide.clasp.ClaspPlugin.
            getDefault().getPreferenceStore().setValue(
                ClaspPreferenceConstants.
                    CLASP_PROGRAM_PATH, CLASP_LOCATION);
    }

    @AfterClass
    public static void afterClass()
    {
        BOT.sleep(1000);
    }

    public AbstractTest(String program)
    {
        this.program = program;
        testProjectName = "Test_Project";
        testFileName = "Test_File_" + program + ".lp";
        testConfigName = "Test_Configuration_" + program;
    }
}

```

```
}
```

### H.5.3 File: CommonTests.java

```
package uk.ac.bath.cs.asp.ide.testing.base;

import org.eclipse.swtbot.eclipse.finder.widgets.
    SWTBotView;

public class CommonTests extends AbstractTest
{
    public CommonTests(String program)
    {
        super(program);
    }

    protected void createNewProject ()
    {
        BOT.menu("File").menu("New").menu("Other...").
            click();
        BOT.shell("New").activate();
        BOT.tree().select("AnsProlog*").expandNode("
            AnsProlog*").select("New Project");
        BOT.button("Next >").click();
        BOT.textWithLabel("Project name:").setText(
            testProjectName);
        BOT.button("Finish").click();
    }

    protected void deleteProject ()
    {
        SWTBotView view = BOT.viewByTitle("Navigator");
        view.show();
        view.bot().tree().getTreeItem(testProjectName).
            select();
        BOT.menu("Edit").menu("Delete").click();
        BOT.shell("Delete Resources").activate();
        BOT.checkBox("Delete project contents on disk (
            cannot be undone)").select();
        BOT.button("OK").click();
    }

    protected void createNewFile ()
    {
```

```
        BOT.menu("File").menu("New").menu("Other...").
            click();
        BOT.shell("New").activate();
        BOT.tree().getTreeItem("AnsProlog*").expand().
            getNode("Lparse File").select();
        BOT.button("Next >").click();
        BOT.tree().getTreeItem(testProjectName).select();
        BOT.textWithLabel("File name:").setText(
            testFileName);
        BOT.button("Finish").click();
    }
```

```
protected void deleteFile ()
{
    SWTBotView view = BOT.viewByTitle("Navigator");
    view.show();
    view.bot().tree().getTreeItem(testProjectName).
        expand().getNode(testFileName).select();
    BOT.menu("Edit").menu("Delete").click();
    BOT.button("OK").click();
}
```

```
protected void canCreateAndDeleteFile ()
{
    createNewProject ();
    createNewFile ();
    deleteFile ();
    deleteProject ();
}
```

```
protected void getRunConfiguration ()
{
    BOT.menu("Run").menu("Run Configurations...").
        click();
    BOT.tree().getTreeItem(program).getNode(
        testConfigName).select();
}
```

```
protected void createNewConfiguration ()
{
    BOT.menu("Run").menu("Run Configurations...").
        click();
    BOT.shell("Run Configurations").activate();
    BOT.tree().getTreeItem(program).select();
}
```

```

        BOT.toolbarButtonWithTooltip("New launch
            configuration").click();
        BOT.text(1).setText(testConfigName);
        BOT.button("Apply").click();
        BOT.activeShell().close();
    }

    protected void deleteConfiguration()
    {
        BOT.menu("Run").menu("Run Configurations...").
            click();
        BOT.shell("Run Configurations").activate();
        BOT.tree().getTreeItem(program).expand().getNode(
            testConfigName).select().contextMenu("Delete")
            .click();
        BOT.button("Yes").click();
        BOT.activeShell().close();
    }

    protected void canCreateAndDeleteConfiguration()
    {
        createNewConfiguration();
        deleteConfiguration();
    }

    protected void closeConfiguration()
    {
        createNewConfiguration();
        getRunConfiguration();
        BOT.button("Close").click();
        deleteConfiguration();
    }

    // protected void runConfiguration()
    // {
        BOT.toolbarButtonWithTooltip("New launch
            configuration").click();
        BOT.text(1).setText(testConfigName);
        BOT.button("Apply").click();
        BOT.activeShell().close();
    }

    // createNewConfiguration();
    // getRunConfiguration();
    // BOT.button("Run").click();
    // deleteConfiguration();
    // }

    protected void accessCommonTabs(String... strings)
    {
        createNewConfiguration();
        getRunConfiguration();
        BOT.cTabItem("Input Files").activate();
        BOT.cTabItem("Common").activate();
        for (String str : strings)
        {
            BOT.cTabItem(str).activate();
        }
        BOT.activeShell().close();
        deleteConfiguration();
    }

    protected void testCheckBoxTabs(String tab, int
        number)
    {
        createNewConfiguration();
        getRunConfiguration();
        BOT.cTabItem(tab).activate();
        for (int i = 0; i < number; i++)
        {
            BOT.checkBox(i).click();
            BOT.checkBox(i).click();
        }
        BOT.activeShell().close();
        deleteConfiguration();
    }
}

```