APE: An AnsProlog* Environment

Adrian Sureshkumar, Marina De Vos, Martin Brain, and John Fitch

Department of Computer Science
University of Bath
Bath, BA2 7AY, UK
{mdv,mjb,jpff}@cs.bath.ac.uk

Abstract. It has been recognised that better programming tools are required to support the logic programming paradigm of Answer Set Programming (ASP), especially when larger scale applications need to be developed. In order to meet this demand, the aspects of programming in ASP that require better support need to be investigated, and suitable tools to support them identified and implemented. In this paper we detail an exploratory development approach to implementing an Integrated Development Environment (IDE) for ASP, the AnsProlog* Programming Environment (APE). APE is implemented as a plug-in for the Eclipse platform. Given that an IDE is itself composed of a set of programming tools, this approach is used to identify a set of tool requirements for ASP, together with suggestions for improvements to existing tools and programming practices.

1 Introduction

Answer Set Programming (ASP) is a declarative programming paradigm with a semantics known as the answer set semantics [4]. It is declarative in that the programmer specifies *what* needs to be achieved, rather than *how* it should it be achieved. It therefore lends itself naturally to applications in the domain of artificial intelligence, such as plan generation and reasoning in agents.

ASP programs, which are written in the language of AnsProlog*, are composed of a set of facts together with a set of rules from which other facts can be derived. A set of consistent facts that can be derived from a program using the rules is known as an answer set of the program. The possible answer sets for an AnsProlog* input program are computed with a program called a solver. Current solvers include SMODELS[23, 27], DLV[9, 10], CLASP [13] and CMODELS [19].

A report by the Working group on Answer Set Programming (WASP) [26] acknowledges that better tools are required to support programming in this paradigm [22]. However in order to identify the aspects that require better support, and consequently develop the appropriate tools to support them, a better understanding of the programming process is needed.

The widespread use of programming tools in other paradigms is an indication of their value to the programmer. It is therefore important to investigate whether these tools could be applied to the domain of ASP and whether they would have the same impact as in other domains, in addition to identifying new tools to solve problems specific to ASP and improving programming practices.

[17] and [11] describe the situation in the 1980's, in which little progress had been made with respect to programming environments for logic programming. Indeed they observe that the environments of the time were restricted to imperative and functional languages. This is clearly no longer the case, given that a quick Internet search for *Prolog IDE* generates many pages of results for development tools. The same is true for other declarative approaches like SAT and CLP.

However, performing a similar search for ASP does not return any relevant results. In fact, at the time of writing the top search result on Google for the query *"Answer Set Programming" Integrated Development Environment* was the undergraduate project proposal that resulted in this paper, demonstrating that this is indeed one of the areas of tools for ASP that is underdeveloped. Thus it can be said that we find ourselves in a similar situation today with ASP, as [17, 11] did in the 1980's with logic programming in general.

The fact that several environments exist for the Prolog language indicate that the development of IDEs for logic programming languages can be achieved and warrants an investigation into whether this would also be possible for ASP.

2 Requirement Elicitation

In order to develop a set of requirements for the system, the people who have a vested interest in the successful development of the system needed to be identified - these are known as the stake-holders [25]. The primary stake-holders for the IDE, are ASP programmers and other members of the ASP community, as they stand to benefit from the improved tool support that the IDE could provide.

In order to gather the fundamental requirements for the IDE and a list of potential features, a questionnaire was developed and distributed by e-mail to members of the ASP community.

The questionnaire was designed to be short and consisted mainly of closed questions in order to minimise the time required by the participant to complete it, although a few open questions were included to allow further elaboration if required. From the 48 questionnaires, only 17 were returned although some of them were group responses instead of individual responses.

The experience of the participants in ASP development ranged from 1 to 10 years experience, with 4 years experience on average. Only 4 participants of the 16 that responded to the question had less than 3 years experience. This suggests that the participants have sufficient knowledge about the process of developing in ASP to provide valuable feedback on how this could be better supported. However it is also possible that through their years of using the current ASP development tools, the participants may be less aware of areas that need better support, as they have learned to work around them.

Supported Solvers: The first question on the questionnaire aimed to determine which tools were used by the participants.

The results of the questionnaire showed DLV and LPARSE/SMODELS to be the ASP tools most widely used by the participants, although this was not surprising given that they are arguably the most well known solver implementations. Although it had

a slightly lower response than the DLV solver, it was chosen to develop the IDE around the SMODELS solver and LPARSE front-end. Given that it is an open source product under the GNU General Public Licence (GPL) [12], whereas only binary builds are available for DLV, the possibility of code reuse was available. Beside the tools that had been suggested to the respondents, the questionnaire also identified five other tools that had not previously been considered:

CMODELS - "*an answer set solver that uses SAT solvers as search engines*" [19, 18].

DLV-EX - An implementation for the DLV system of "*Answer Set Programming with External Predicates (ASP-EX), a framework aimed at enabling ASP to deal with external sources of computation*" [6, 7].

CR-MODELS - The inference engine for CR-Prolog, "*an extension of A-Prolog by consistency restoring rules with preferences*" [3, 16].

ASSAT - A system that computes "*answer sets of a logic program by using SAT solvers*" [21, 20].

ASET-SOLVER - A solver for ASET-Prolog, "*an extension of A-Prolog that adds to the language sets of terms and functions from these terms to natural numbers*" [15, 14].

Given the range of tools used by members of the community, it would clearly not be viable to attempt to provide support for every one of these, at least in the initial version. Equally, it would clearly be impractical for users of these tools to develop a program from within the IDE, but solve it, say, from the command line. Consequently, this could limit the user base of the system. Given this, it was clear that the IDE would need to provide some sort of extension mechanism (e.g. plug-ins) or the ability to run external commands from within the IDE. This would allow others to integrate other tools into the environment if required. Indeed it was commented that "*it would be nice to have a plugin system that will enable it to be extended to other AnsProlog inference systems*" and to have "*the possibility to choose which solver one wants to use*".

Target Platform: The second question in the questionnaire aimed to identify the operating systems used by the community for ASP development.

The most widely used operating system for ASP development by participants of the survey was clearly Linux. Thus this was chosen as the target platform for the system. However as other platforms were in use, and indeed some participants used these exclusively (e.g. Windows), a platform independent solution is clearly desirable.

The target platform for the IDE is also constrained by the supported platforms of any tools to be integrated. However, given that LPARSE and SMODELS are available in source form, and builds of DLV are available for Linux, Free-BSD, MacOS X and Windows this should not have a great impact on the IDE. If other tools were to be integrated that only supported specific platforms, the availability of a version for the desired platform could be arranged with the tool developer.

Potential Features: In order to determine some potential features for the IDE, a local brainstorming session was conducted.

Syntax highlighting could help the programmer to more easily distinguish between different elements of the program code. For example, by highlighting all keywords in a given colour it would immediately become apparent to the programmer if they at-

tempted to use a keyword as a constant name. This error may otherwise not have been discovered until the program was run through the solver or grounder.

Providing the automatic completion of predicates (or terms) that had already been defined in the program would reduce the time taken to input the program. It would also help to reduce errors in the code caused by mistyping a name, not only by reducing the amount of typing that occurs, but equally through the lack of an expected completion indicating to the programmer that a mistake had been made.

Although the name of a predicate should be descriptive, it may not always be possible to achieve this without making the name long and cumbersome to type. Therefore it would be convenient to be able to associate a textual description with each predicate giving a more accurate definition of its meaning. However as there is currently no syntax to support this in DLV or LPARSE, this would have to be encoded within comments. If this feature was shown to be a success the inclusion of a special syntax for this could be requested from the solver developers.

Version control tools, such as the Concurrent Versions System (CVS), are often used when developing software to maintain a history of revisions of source files and facilitate several developers working together on a project. Integrating such tools into the IDE would facilitate their use within ASP development and eliminate the need to switch to an external program to interact with them.

The ability to divide a program into multiple files is important as it allows a core set of rules to be used in more than one program. For example, a set of rules encoding a problem could be defined in one file and sets of facts representing inputs to the problem in several other files. Given that input from multiple files is supported by the same grounder, this should also be supported by an IDE.

An ASP program can be represented in terms of a dependency graph [4], which shows how the truth value of a predicate depends on the truth or falsity of other predicates. Providing a graph representation of programs as part of the IDE would convey this information easily, rather than having to manually extract it from the source code.

It can be difficult to find the source of errors in programs, and as discussed by [5] this is compounded by the fact that it is difficult to determine whether an ASP program is behaving correctly. Whereas a procedural program may crash or throw an exception when an error is encountered, this is not the case for an ASP program. It would therefore be useful to include some form of debugging tools in the IDE to support this process.

Constantly switching between different tools can limit the productivity of the programmer. This frequently occurs when programming, for example switching between the editor to write a program, and to the command line in order to run it. Therefore, integrating the running of the LPARSE and SMODELS tools and the editor into the same environment would remove this need.

Validation: An important aspect of the requirements engineering process is to verify that the requirements that have been gathered for a system "*actually define the system that the customer wants*" [28]. In order to understand how the features proposed at Bath would be viewed by the wider community, an informal review of the requirements was performed by presenting the list of potential features on the questionnaire. Participants were asked to rate their desire for a particular feature on a unipolar scale, with 0 being least useful and 10 being the most useful.

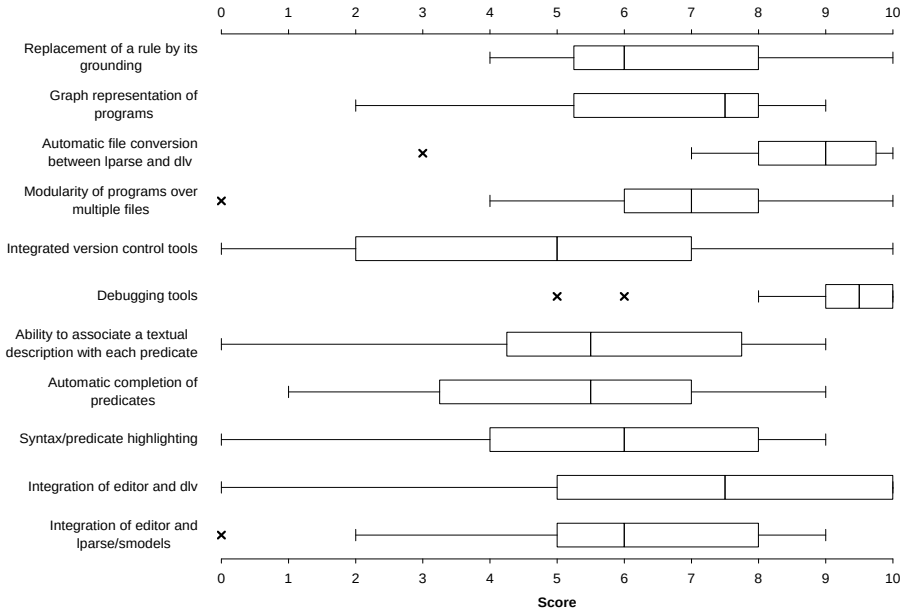

Fig. 1. Score of Suggested IDE Features

The results of this questionnaire have been presented as a box-and-whisker plot (Figure 1), in order to show the spread in the responses for each feature. The plot shows the upper and lower scores for each response (whiskers), together with the median score and interquartile range (box). Any outliers have been indicated with a cross.

From the list of features proposed on the questionnaire, it was clear that debugging tools were the most desired by the respondents, given that the majority gave this a score between 9 and 10. This would therefore be a core component of an IDE for ASP. At present a number of groups are working on debugging techniques for ASP [5, 31, 24]. Given the early stages of this research, it was not deemed viable to specifically consider debugging as part of this initial IDE. However we look forward to integrate or support their work in the future.

Another popular choice was the automatic conversion between files in the LPARSE and DLV formats. However, as the version of the IDE produced will only support the LPARSE language, this feature will not be implemented.

Although there was a large spread in the responses for integrating an editor with the solver, it was generally desired as most responses rated it at 5 and above. Moreover, this is an essential component of an IDE as the programmer needs to be able to edit the program and then run it through the solver. The replacement of a rule by its grounding, graph representation of programs and modularity of programs over multiple files, also appeared to be popular with the respondents as the interquartile range for each fell between a score of 5 and 8.

Furthermore, the remaining features all received a median score of at least 5 demonstrating some support for them, even if there was a wide spread in the scores that they were given. As none of the features in the list was shown to be very unpopular, in the way that debugging tools were shown to be very popular, it would appear that they are all judged to be of some potential by the respondents and should therefore all be explored further.

Suggested Features: As discussed by [25], it is important to include as large a number of representatives from each stakeholder group as possible in the data gathering process, in order to avoid getting a narrow view of the requirements. Therefore in order to integrate the views of the wider community into the list of potential features, respondents to the questionnaire were asked to suggest any other features that could be included. We now consider these features.

One request was made to incorporate the statical analysis of program tightness into the IDE. This syntactic condition on a program is also known as positive order consistency [2]. If a program can be shown to be tight, then for that program the answer set semantics are equivalent to another semantics known as the completion semantics. In this case a satisfiability solver can be used to determine the answer sets of a program, rather than an answer set solver such as SMODELS. Including this analysis as part of the IDE could be used for indicating whether this type of solver could be used on a given program.

It was also requested to provide support for make files. Given that a program could potentially be split over several files, some of which may have already been grounded, a build script could be used to automate the process of grounding any files that had changed since last being grounded and then running the program through a solver. The IDE would therefore need to provide support for this functionality.

In addition to this was the request to support scripts to filter the input to and output from the solvers. Providing support for scripts to perform this would permit the transformation of data from some source into a program that would be accepted by the solver, and accordingly the output from the solver to be transformed into a more useable form.

Another key feature that was suggested by one participant was automatic syntax checking. Highlighting syntax errors in the editor as they are typed, would make the error immediately evident to the programmer and prompt them to make a correction. This would eliminate the overhead of running the program through the solver before the error would be discovered, and potentially doing this multiple times to locate and correct all of the errors.

Given the range of solvers used for ASP, it was suggested that the IDE should allow the user to choose which solver they want to use when running the program. However as we have restricted the initial version of the system to supporting the LPARSE and SMODELS tools, this feature will not be considered. Related to this was the ability to provide benchmarks for the different solvers in the system, such as the time taken to run the solver. This feature would allow the user to compare different solvers and potentially choose the one most suited to their specific program.

The value of generating the dependency graph for a program has already been considered, however this was reiterated with requests to display the components of these graphs (such as the atoms) and the dependencies between them.

Features Supported by Version 1 of APE This initial requirements gathering phase has helped to identify some of the non-functional requirements of the system, such as the platform on which it must operate and the tools that it must support, together with a list of potential features. For the initial release of an IDE, we decided on the following options:

- Support for LPARSE and SMODELS tools
- Multi-platform support
- Syntax highlighting
- Automatic syntax checking
- Integrated version control tools
- Multiple file support
- Display of program dependency graph
- Integration of editor and LPARSE
- Integrated build script support

The selection is wide enough so that the IDE can be evaluated in a significant manner, yet restricted enough for users to make comments and suggestions. Given that LPARSE is used as a grounder for many other solvers than SMODELS, the IDE can also be used for the development for these solvers. With a minor change this solver can also be called directly from the IDE.

3 Eclipse

To develop our IDE, we have opted for the Eclipse platform. Some of the people we questioned are already familiar with the platform, it already provides a number of programming tools and it can be easily extended in incremental steps.

The Eclipse platform, described as "*an IDE for anything, and for nothing in particular*" [8], is surrounded by an industry buzz according to [34]. The author identifies several reasons for this success including being free, given that equivalent IDEs can cost more than $1,000, and supporting multiple platforms including Windows, MacOS, Solaris and Linux (Red Hat & SuSE). The platform provides a lot of generic functionality and "*is built on a mechanism for discovering, integrating, and running modules called plug-ins*" [8]. Moreover, the licence terms allow third-party developers to charge for any extensions that they produce [34], which clearly provides an incentive for developers of commercial and open-source tools to use this platform. It is however criticised by some for having an excess of features, which could be overwhelming for inexperienced users.

Plug-ins typically consist of Java code contained in a JAR (Java Archive) file, together with resources and a manifest file [8]. The development of plug-ins is facilitated by the provision of an IDE in Eclipse - the Plug-in Development Environment (PDE). The manifest file is an XML file which defines a set of extension points, which other plug-ins may extend, together with its extensions - how it is extending the extension

point defined by another plug-in. This could clearly be an advantage in an IDE for ASP, by allowing developers to integrate their own solvers into the framework provided. The best known plug-in for Eclipse is probably the Java Development Tooling (JDT) included in the main distribution together with the platform and PDE - although the platform is also available separately. [34] observes that this is probably why Eclipse is viewed by many as simply a Java IDE, rather than a framework to host IDEs and other tools.

4 APE Version 1.0

In this section we have a closer look at how these features are incorporated in our IDE. By opting for Eclipse as our base model, we automatically obtain that our system is platform independent, provided that the we do not use any platform specific package to implement the various tools. Furthermore, the use of Eclipse gave us access to integrated version control tools and integrated build script support. Given its modular approach it provides the necessary support for integrating additional solvers into the IDE.

4.1 System Overview

The final system consists of 6 plug-ins: the core (doing the background work), the general user interface, one to run SMODELS and one for LPARSE, the user interface for both programs and one to generate dependency graphs.

Figures 2, 3 and 4 give three different screen shots of the final system. They demonstrate that IDE has four parts (if not closed). The left shows the working directory. The middle is the actual editor with all open files in the workbench and one file active. The right shows different views of the active program like the syntactic outline or the dependency graph. The bottom part shows either the console with the answer sets of the program if SMODELS is called or the errors/warnings the system has detected in the active program. Note that the layout can be customised by the Eclipse user if this is not what is wanted.

The system is licensed under the GNU GPL [12] and available at `http://krr.cs.bath.ac.uk/index.php/APE`. More information about APE can be found in [29] and on the above webpage.

4.2 Syntax Highlighting

In order to have syntax highlighting, or colouring as it called in Eclipse, that was suitable for ASP, it was not sufficient to extend one of the already available syntax colouring tools. Unfortunately, the highlighting of tokens such as constants and functions could not be achieved without additional parsing of the source file.

To solve this problem, we adapted the parser and scanner from LPARSE to work with Java and reused this in the IDE. Using the same specification also ensures that the IDE's parser accepts the same programs as the LPARSE tool itself.

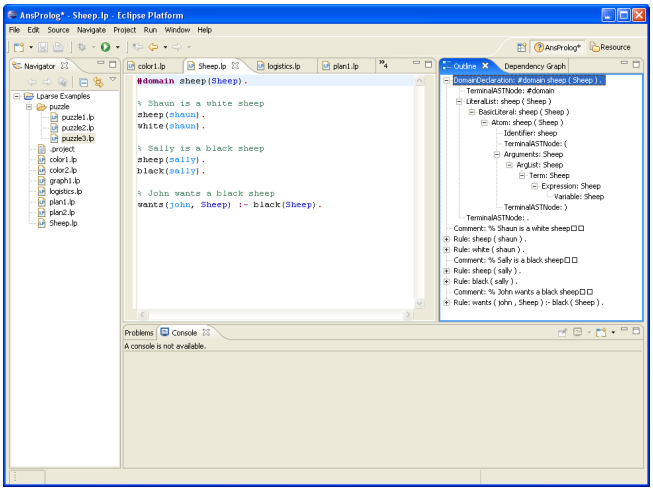

Fig. 2. IDE in outline mode

The scanner used in the LPARSE tool performed the analysis of a single program combined from all the input files specified on the command line. However for a file open in the LPARSE editor, it is not known with which other files it would be used or whether it would even be used with any other files. Therefore when programs are split over multiple files, some way of defining how these files are aggregated to form a single program would be necessary to perform an analysis of the entire program. We decided to leave this for future work and limit ourselves to only analysing the file which was currently open in the editor.

Thus parsing of the LPARSE source files is not only necessary for the more detailed syntax highlighting, but for any other tools that need to perform an analysis of the program. This includes highlighting of errors and warnings in the editor, computation of dependency graphs, auto-completion and analysis of program tightness. Given this is needed, we have opted for a data integration approach in which the source code is only parsed once and stored in a shared data structure that could be used by several tools.

The parser generates a data structure very similar to the one displayed in the outline view of Figure 2 which can then be used for assigning different colours to the various components.

Whenever a change is made to the source file, the entire document is re-parsed and a data structure is generated. However this one data structure is shared amongst all the features of the IDE such as syntax colouring and checking and graphs – in order that the file does not have to be re parsed for each feature. An improvement would be to do this incrementally.

The user can change the colour assignment of each of the individual components of the program using the ASPSyntaxColouring Dialog, as shown in Figure 5.

4.3 Automatic Syntax Checking

A syntax error occurs "*when the string of input tokens is not a sentence in the language*" [1]. In order that as many syntax errors as possible can be reported to the programmer

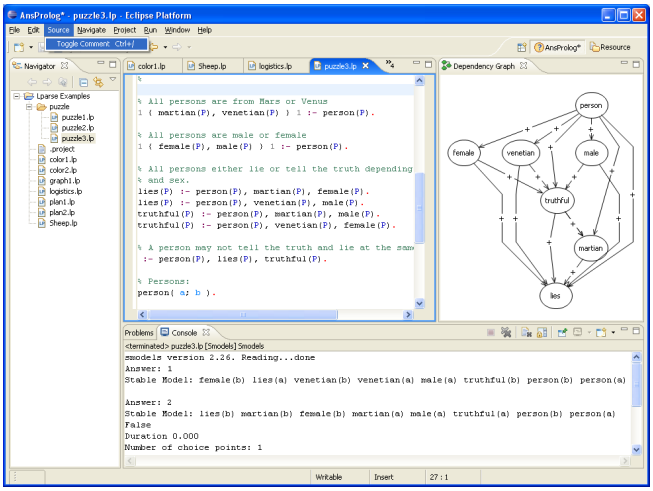

Fig. 3. IDE in dependency graph mode

in a single pass through the program, the parser should be able to recover from the discovery of an error and continue to discover other potential errors [33].

An LPARSE program consists of rules, statements and declarations each separated by a full stop (.)[30]. Therefore after encountering an error the parser can skip over the tokens until this token is encountered. Indeed this is the method of recovery used in the original parser, and has been maintained for the IDE.

In Eclipse, marker objects can be used to attach annotations to workspace resources, with these annotations being stored in the workspace meta-data rather than by modifying the existing file. The Eclipse text editor automatically highlights errors and warnings in a file, if problem markers representing them are attached to the resource. These are also displayed in the Eclipse problems view.Figure 4 on page 111. When the source file is modified, the problem markers are replaced with a new set generated from the information in the shared data structure.

The original grammar for LPARSE contained code in the action for the constant declaration rule to warn the user if the constant that they were declaring had already been defined or used as a symbolic constant. It also contained rules for common mistakes made when entering constant declarations: using a variable name rather than identifier for the constant or missing the assignment operator. This provided a more specific error message than the general ‘parse error’ message would have, aiding the programmer to locate the problem more quickly. The action for these rules was therefore implemented to create a new problem object with the same message as provided in the original C code. This could be extended by investigating other common errors made when writing LPARSE programs, adding rules to support them, and returning a problem object with a more specific error message.

4.4 Integration of Editor, LPARSE and SMODELS

To eliminate the user overhead of switching between the editor and command-line to run a program, we have provided a plug-in to enable launching LPARSE and SMODELS

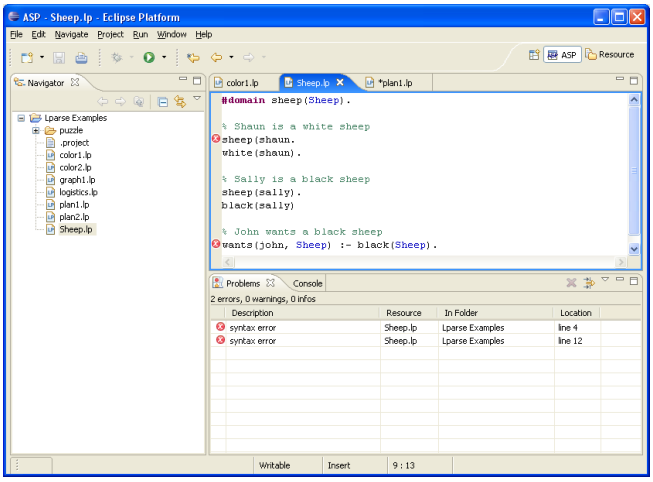

Fig. 4. Program with syntactical errors

from within the IDE. The respective dialog boxes allows the user to set all the flags and parameters that are available to the programs. Figure 6 shows a screen shot of a LPARSE dialog box where the user can opt for different run-time settings.

The output provided by SMODELS contains a lot of information which is not always required, or can be difficult to interpret easily. So being able to pass the output to another program or script for formatting can be welcome. Also, it might be important to save the answer sets of certain programs. To allow for this, we have added an extra tab to the SMODELS launch dialog that allows the SMODELS output to be piped to a different program. It will be the output of this other program that will be displayed in the console part of the IDE.

4.5 Dependency Graphs

The questionnaire demonstrated strong support for the display of dependency graphs, and so this feature was chosen to be implemented given the availability of the source file model.

[4] defines the dependency graph of a program to consist of:

- a set of vertices, such that each vertex corresponds to a predicate name.
- a set of edges, such that the edge from P_i to P_j is in the set if and only if there exists a rule in the program that has P_i in the head and P_j in the body. The edge is labelled with a + if P_j appears as a positive literal, with a - if it appears as a negative literal, or indeed with + and - if rules exist such that both cases are present.

The dependency graph functionality was defined in a separate plug-in (section 3). This allows other ASP tools to reuse the functionality.

In order to display the dependency graph in Eclipse a suitable library to support graph drawing had to be chosen. The criteria for this package were that it had to be platform independent and use the Eclipse graphical packages (Eclipse's SWT) rather than Swing. In the end we decided to use the Draw2D plug-in from the Eclipse Graphical Editing Framework (GEF) feature. support for drawing classes for modelling and

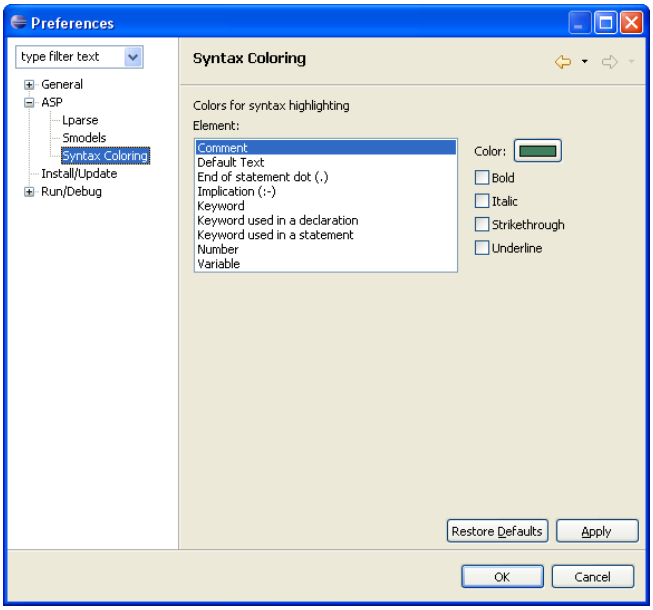

Fig. 5. ASPSyntaxColouring Dialog

To display the graph within the IDE, the `DependencyGraphView` was implemented, which given an `AnsPrologProgram` data model would build a dependency graph model using the Draw2D graph classes and display this in its `GraphViewer` (Figure 3 on page 110). The LPARSE editor was also updated to make sure that the dependency graph was updated every time the active source file was changed.

4.6 Extra Features

During several intermediate validation and observation sessions, it was pointed out to us that it would have been nice to have the facility to have block comments and the short-cuts similar to the subjects favourite browsers, which was Emacs in this particular case. The latter was easy to accommodate as Eclipse has a set of built-in short-cut schemes, one of which is Emacs. The former did not take much effort either as a similar action had already been implemented as part of the JDT, which we were able to use as an example. The action was named 'Toggle Comment' in order to be consistent with the JDT, and was implemented to behave in the same way. The option was also added to the menu and a short-cut key was associated to it. It was again set to be the same as used in the JDT: `Ctrl + /` under the default configuration and `Ctrl + 7` under the Emacs configuration. However the use of `Ctrl + %` for the default configuration may have been more natural for the user, given that `%` is the single line comment character for LPARSE, rather than the `//` used in Java. In the end it was decided to keep the Eclipse default, in order not to confuse users who also use Eclipse for different languages. of both plug-ins

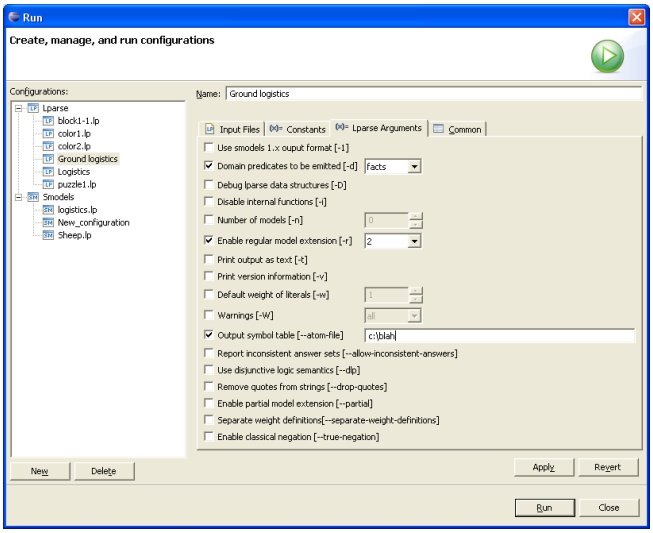

Fig. 6. LPARSE command-line options dialog box

5 Conclusions and Future Work

As far as we are aware, very little research has been taken place on software engineering for answer set programming. Apart from a graphical user interface for the SMODELS solver [32], perhaps the Emacs mode for LPARSE/SMODELS is the only one publicly available. The Emacs mode provides Emacs users with indentation, syntax highlighting and running LPARSE and SMODELS via commands [30].

Apart from providing an initial IDE for ASP, this paper provided the first set of requirements for ASP development tools. In a way the presented IDE is only the tip of the iceberg compared to programming tools available for traditional languages.

The questionnaire only reached a small proportion of the ASP field. In the future, a wider range of users needs to be considered for the evaluations of the system. This wider view would have allowed a better definition of the requirements of the IDE to be produced, by considering the needs of a more varied user base and any conflicts of interest between their different needs.

At present, the IDE has only been tested within the department. To evaluate it in a more scientific manner, it needs to be tested by a wider group including experienced ASP programmers and novices. Such an evaluation should bring to light the requirements from different user groups. Furthermore, observing people using the tool will also give more insight into programming techniques.

During the requirements analysis, another tool that was identified to be of potential use when programming in ASP was that of automatically indenting the code to facilitate maintaining a consistent, easy to read layout throughout the program. However, the layout that the tool should adhere to would first need to be defined. Therefore it is proposed that a study into coding styles for ASP should be undertaken in order to define

a common set of coding standards to improve the readability and maintainability of code.

In addition to the new ASP tools that were identified in the requirements elicitation process, an improvement to an existing tool was also identified. One requirement of the IDE that was raised throughout the elicitation process was for a tool to perform block commenting. This was due to the syntax of the LPARSE solver only supporting single line comments rendering the commenting of large blocks of code a tedious process. Although developing this tool supports the programmer, it is resolving the problem in the wrong place. It would be better to add Multi-line comments to the LPARSE syntax, in order that all ASP programmers could benefit from faster commenting, regardless of whether they use the IDE or not.

Although APE is only the first version of an IDE for answer set programming, we are sure it already provides a number of tools that make it easier to write programs in the language. Initial trials support this belief. In the future we will be extending and improving the current set of available features. The first feature on the list is to incorporate debugging tools in the IDE.

References

1. A. W. Appel and M. Ginsburg. *Modern Compiler Implementation in C*. Press Syndicate of the University of Cambridge, 2004.
2. Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In C. Baral and M. Truszczynski, editors, *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning, NMR'2000*, 2000.
3. M. Balduccini. CR-MODELS homepage. `http://krlab.cs.ttu.edu/~marcy/crmodels/`.
4. C. Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, Cambridge, UK, 2003.
5. M. Brain and M. De Vos. Debugging logic programs under the answer set semantics. In M. De Vos and A. Provetti, editors, *Answer Set Programming: Advances in Theory and Implementation*, pages 142 – 152. Research Press International, 2005.
6. F. Calimeri and G. Ianni. External sources of computation for answer set solvers. *Lecture Notes in Computer Science*, 3662:105–118, 2005.
7. F. Calimeri, G. Ianni, and S. Cozza. `http://www.mat.unical.it/ianni/wiki/dlvex`.
8. Eclipse. Eclipse platform technical overview. 2003.
9. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In A. G. Cohn, L. Schubert, and S. C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
10. W. Faber and G. Pfeifer. DLV homepage. `http://www.dbai.tuwien.ac.at/proj/dlv/`.
11. N. Francez, S. Goldenberg, R. Y. Pinter, M. Tiomkin, and S. 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.
12. Free Software Foundation. Gnu general public license version 2, 1992.

13. M. Gebser, B. Kaufmann, A. Neumann, and T. Schaub. Conflict-driven answer set solving. In M. Veloso, editor, *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, pages 386–392. AAAI Press/The MIT Press, 2007. Available at http://www.ijcai.org/papers07/contents.php.
14. M. Heidt and V. S. Mellarkod. ASET homepage. `http://www.cs.ttu.edu/~mellarko/aset.html`.
15. M. L. Heidt. Developing an inference engine for aset-prolog. Master's thesis, University of Texas at El Paso, December 2001.
16. L. Kolvekal. Developing an inference engine for cr-prolog with preferences. Master's thesis, Texas Tech University, December 2004.
17. H. J. Komorowski and S. 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.
18. Y. Lierler. CMODELS homepage. `http://www.cs.utexas.edu/~tag/cmodels/`.
19. Y. Lierler and M. Maratea. Cmodels-2: Sat-based answer set solver enhanced to non-tight programs. *Lecture Notes in Computer Science*, 2923:346–350, 2004.
20. F. Lin and Y. Zhao. ASSAT homepage. `http://assat.cs.ust.hk/`.
21. F. Lin and Y. Zhao. Assat: computing answer sets of a logic program by sat solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.
22. I. Niemelä, editor. *WASP WP3 Report: Language Extensions and Software Engineering for ASP*. 2005.
23. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In J. Dix, U. Furbach, and A. Nerode, editors, *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
24. E. Pontelli and T. C. Son. *ustifications* for logic programs under answer set semantics. In S. Etalle and M. Truszczynski, editors, *ICLP*, volume 4079 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006.
25. J. Preece, Y. Rogers, and H. Sharp. *Interaction Design*. John Wiley & Sons, Inc., New York, NY, USA, 2002.
26. A. Provetti. Wasp homepage. `http://wasp.unime.it/`.
27. P. Simons. SMODELS homepage. `http://www.tcs.hut.fi/Software/smodels/`.
28. I. Sommerville. *Software Engineering*. Addison-Wesley Publishers Ltd., Harlow, England, 6th edition, 2001.
29. 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.
30. T. Syrjänen. *Lparse 1.0 User's Manual*.
31. T. Syrjänen. Debugging inconsistent answer set programs. In J. Dix and A. Hunter, editors, *Proceedings of the 11th Workshop on Nonmonotonic Reasoning (NMR)*, number Ifl-06-04 in Ifl Technical Report Series, 2006. Available from `http://cig.in.tu-clausthal.de/NMR06/`.
32. H. Takahashi. A GUI for Smodels. `http://www.baral.us/bookone/ansprolog/`, October 2004.
33. R. Wilhelm and D. Maurer. *Compiler Design*. Addison-Wesley Publishing Co. Inc., Wokingham, England, 1995.
34. A. Wolfe. Toolkit: Eclipse: A platform becomes an open-source woodstock. *Queue*, 1(8):14–16, 2003.