ASPIDE: Integrated Development Environment for Answer Set Programming

Onofrio Febbraro[1], Kristian Reale[2], and Francesco Ricca[2]

[1] DLVSystem s.r.l. - P.zza Vermicelli, Polo Tecnologico, 87036 Rende, Italy
febbraro@dlvsystem.com
[2] Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
{reale,ricca}@mat.unical.it

Abstract. Answer Set Programming (ASP) is a truly-declarative programming paradigm proposed in the area of non-monotonic reasoning and logic programming. In the last few years, several tools for ASP-program development have been proposed, including (more or less advanced) editors and debuggers. However, ASP still lacks an Integrated Development Environment (IDE) supporting the entire life-cycle of ASP development, from (assisted) programs editing to application deployment. In this paper we present *ASPIDE*, a comprehensive IDE for ASP, integrating a cutting-edge *editing tool* (featuring dynamic syntax highlighting, on-line syntax correction, autocompletion, code-templates, quick-fixes, refactoring, etc.) with a collection of user-friendly *graphical tools* for program composition, debugging, profiling, database access, solver execution configuration and output-handling.

1 Introduction

Answer Set Programming (ASP) [1] is a declarative programming paradigm which has been proposed in the area of non-monotonic reasoning and logic programming. The idea of ASP is to represent a given computational problem by a logic program whose answer sets correspond to solutions, and then use a solver to find such a solution [2]. The language of ASP is very expressive [3]. Furthermore, the availability of some efficient ASP systems [4,5,6,7,8,9,10,11,12,13,14,15] made ASP a powerful tool for developing advanced applications. ASP applications belong to several fields, from Artificial Intelligence [14,16,17,18,19,20,21] to Information Integration [22], and Knowledge Management [23,24,25]. These applications of ASP have confirmed, on the one hand, the viability of the exploitation in real application settings and, very recently, stimulated some interest also in industry [26]. On the other hand, they have evidenced the lack of effective development environments capable of supporting the programmers in managing large and complex projects [27]. It is nowadays recognized [27] that this may discourage the usage of the ASP programming paradigm, even if it could provide the needed reasoning capabilities at a lower (implementation) price than traditional imperative languages. Note also that, the most diffused programming languages always come with the support of SDKs featuring a rich set of tools that significantly simplify both programming and maintenance tasks.

J. Delgrande and W. Faber (Eds.): LPNMR 2011, LNAI 6645, pp. 317–330, 2011.

In order to facilitate the design of ASP applications, some tools for ASP-program development were proposed in the last few years (consider, for instance, the aim of SEA workshop series [28,29]), including editors [30,31] and debuggers [32,33,34,35,36]. However, ASP still lacks an Integrated Development Environment (IDE) supporting the entire life-cycle of ASP development, from (assisted) programs editing to application deployment.

This paper provides a contribution in this setting. In fact, it presents *ASPIDE*, a comprehensive IDE for ASP, which integrates a cutting-edge *editing tool* (featuring dynamic syntax highlighting, on-line syntax correction, autocompletion, code-templates, quick-fixes, refactoring, etc.) with a collection of user-friendly *graphical tools* for program composition, debugging, profiling, DBMS access, solver execution configuration and output-handling. Currently, the system is able to load and store ASP programs in the syntax of the ASP system DLV [4], and supports the `ASPCore` language profile employed in the ASP System Competition 2011[37]. Data base management is compliant with DLV^{DB} [38] language directives.

A comprehensive feature-wise comparison with existing environments for developing logic programs is also reported in this paper, which shows that *ASPIDE* represents a step forward in the present state of the art of tools for ASP programs development.

2 Main Features of *ASPIDE*

The main functionalities provided by *ASPIDE* are first summarized below, and then described in more detail. In particular, *ASPIDE* provides the following features:

- *Workspace management*. The system allows to organize ASP programs in projects à la Eclipse, which are collected in a special directory (called workspace).
- *Advanced text editor*. The editing of ASP files is simplified by an advanced text editor, which provides several functionalities: from simple text coloring to auto completion of predicates and variables names.
- *Outline navigation*. *ASPIDE* creates an outline view which graphically represents program elements.
- *Dependency graph*. The system provides a graphical representation of the dependency graph of a program.
- *Dynamic code checking and errors highlighting*. Syntax errors and relevant conditions (like safety) are checked *while typing programs*: portions of code containing errors or warnings are immediately highlighted.
- *Quick fix*. The system suggests quick fixes to reported errors or warnings, and applies them (on request) by automatically changing the affected part of code.
- *Dynamic code templates*. Writing of repeated programming patterns (like transitive closure or disjunctive rules for guessing the search space) is assisted by advanced auto-completion generating rules or part of them.
- *Test suite*. The user can define test suites on the line with the principle of unit testing diffused in the development of software with iterative languages.
- *Debugger and Profiler*. Semantic errors detection as well as code optimization can be done by exploiting graphic tools.

- *Configuration of the execution*. This feature allows to configure input programs and execution options.
- *Presentation of results*. The output of the program (either answer sets, or query results) are visualized in a tabular representation or in a text-based console.
- *Visual Editor*. The users can *draw* logic programs by exploiting a full graphical environment that offers a QBE-like tool for building logic rules. The user can switch, every time he needs, from the text editor to the visual one (and vice-versa) thanks to a reverse-rengineering mechanism from text to graphical format.
- *Interaction with databases.* Interaction with external databases is made easy by a fully graphical import/export tool that automatically generates mappings by following the DLV^{DB} Typ files specifications [38]. Text editing of Typ mappings is also assisted by syntax coloring and auto-completion.

In the following, we describe in more detail the above mentioned functionalities.

Workspace organization. The system allows for organizing ASP programs in projects à la Eclipse. This facilitates the development of complex applications by organizing modules (or projects) in a space where either different parts of an encoding or several equivalent encodings solving the same problem are stored. In particular, *ASPIDE* allows to manage: (i) *DLV files* containing ASP programs in both DLV syntax [4] and `ASPCore` [37]; (ii) *TYP files* specifying a mapping between program predicates and database tables in DLV^{DB} syntax; (iii) *TEST files* containing a set of directives conceived for defining unit tests for the created logic programs.

Advanced text editor. The presence of an editor that provides a set of advanced features is indispensable for a good development environment. In particular, besides the core functionality that basic text editors offer (like, code line numbering, find/replace, undo/redo, copy/paste, etc.), *ASPIDE* offers others advanced functionalities:

- *Text coloring*. *ASPIDE* exploits different colors for outlining key words (like " :– ") predicate names, variables, strings, comments etc. Predicates involved in database mappings are also specifically marked.
- *Automatic completion*. The system is able to complete (on request) predicate names, as well as variable names. Predicate names are both learned while writing, and extracted from the files belonging to the same project; variables are suggested by taking into account the rule we are currently writing. This helps while developing either an alternative encoding for the same problem (input/intermediate predicate names are ready to be suggested after the first file is completed) or when the same solution is divided in several files. When several possible alternatives for completion are available the system shows a pop-up dialog.
- *Refactoring*. The refactoring tool allows to modify in a guided way, among others, predicate names and variables. For instance, variable renaming in a rule is done by considering bindings of variables, so that common side effects of find/replace are avoided by ensuring that variables/predicates/strings occurring in other expressions remain unchanged.

Outline navigation. ASPIDE creates a graphic outline of both programs and Typ files, which graphically represents language statements. Regarding the programs, the outline is a tree representation of each rule where in the first level of the tree there are the head atoms, while the second level corresponds to the bodies. Each item in the outline can be used to quickly access the corresponding line of code (a very useful feature when dealing with long files), and also provides a graphical support for building rules in the graphical editor (see [39]).

Dependency graph. The system provides a graphical representation of the (non-ground) dependency graph associated to the project. Several variants of the dependency graph are supported depending on whether both positive and negative dependencies are considered. The graph of strongly connected components (playing an important role in the instantiation of the program) can be displayed also.

Dynamic code checking and errors highlighting. Programs are parsed while writing, and both errors or possible warnings are immediately outlined without the need of saving files. In particular, syntax errors as well as mismatching predicate arities and safety conditions are checked. Note that, the checker considers the entire project, and warns the user by indicating e.g., that atoms with the same predicate name have different arity in several files. This condition is usually revealed only when programs divided in multiple files are run together.

Quick fix. The system suggests quick fixes to reported errors or warnings, and applies them (on request) by automatically changing the affected part of code. This can be activated by clicking on the line of code which contains an error/warning and choosing from a popup window the desired fix among several suggestions, e.g., safety problems can be fixed by correcting variable names or by projecting out "unsafe" variables through an auxiliary rule (which will be automatically added).

Code template. ASPIDE provides support for automated writing of parts of rules (guessing patterns, aggregates, etc.), as well as automated writing of entire subprograms (e.g., transitive closure rules) by exploding code templates. Note that, templates can be also user defined by writing DLT [40] files.

Test suite. A very important phase in the development of software is testing. In *ASPIDE* we have defined a syntax for writing and running tests in the style of JUnit. One can write assertions regarding the number of answer sets or the presence/absence of an atom in the results of a test.

Debugger and Profiler. We have embedded in *ASPIDE* the debugging tool *spock* [32]. To this end we have worked on the adaptation of spock for dealing with the syntax of the DLV system. Moreover, we developed a graphical user interface that wraps the above mentioned tool. Regarding the profiler, we have fully embedded the graphical interface presented in [41].

Configuration of the execution. ASPIDE provides a form for configuring and managing solver execution. A run configuration allows to set the solver/system executable, setup invocation options and input files. Moreover, *ASPIDE* supports a graphical tool

for composition of workflow executions (e.g., combining several solver/system calls), which are transparently compiled in perl scripts.

Presentation of the results. The ASP solvers output results in textual form. For making comfortable the browsing of results, *ASPIDE* visualizes answer sets by combining tabular representation of predicates with a comfortable tree-like representation for handling multiple answer sets. The result of the execution can be also saved in text files for subsequent analysis.

Visual editor. *ASPIDE* offers also the possibility to draw programs using a fully graphical environment by embedding and extending the *Visual Editor* tool which offers a QBE-like approach; it allows, for example, to create graphical bindings of variables using "joins" between predicates. For a detailed explanation on how the Visual Editor works, see [39]. An important feature offered by *ASPIDE* is *reverse engineering* that allows to switch between textual and visual representation of programs.

Interaction with databases. Interaction with external databases is useful in several applications (e.g., [22,26,19]). *ASPIDE* allows to access external databases by exploiting a graphical tool connecting to DBMSs via JDBC. Imported sources are emphasized also in the program editor by exploiting a specific color. Database oriented applications can be run by setting DLV^{DB} as solver in a run configuration. A data integration scenario [22] can be implemented by exploiting this feature.

3 Interface Overview and Implementation

The system interface of *ASPIDE* is depicted in Figure 1, where the main components are outlined in different numbered zones. In the upper part of the interface (zone 1) a toolbar allows the user to call the most common operations of the system (from left to right: save files, undo/redo, copy & paste, find & replace, switch between visual to text editor, run the solver/profiler/debugger). In the center of the interface there is the main editing area (zone 4), organized in a multi-tabbed panel possibly collecting several open files. The left part of the interface is dedicated to the explorer panel (zone 2), and to the error console (zone 3). The explorer panel lists projects and files included in the workspace, while the error console organizes errors and warnings according to the project and files where they are localized. On the right, there are the outline panel (zone 5) and the sources panel (zone 6). The first shows an outline of the currently edited file, while the latter reports a list of the database sources which might be mapped to some predicate of the current project. The one shown in Figure 1 is the standard appearance of the system, which can be however modified, since panels can be moved as the user likes. A comprehensive description of *ASPIDE* is available in the online manual published in the system web site `http://www.mat.unical.it/ricca/aspide`.

ASPIDE is written in Java by following the Model View Controller (MVC) pattern. A *core* module manages, by means of suitable data structures, projects, files content, system status (e.g., error lists, active connections to DBMSs etc.), and external component management (e.g., interaction with solver/debugger/profiler). Any update to the information managed by *ASPIDE* is obtained by invoking methods of the core, while, *view*

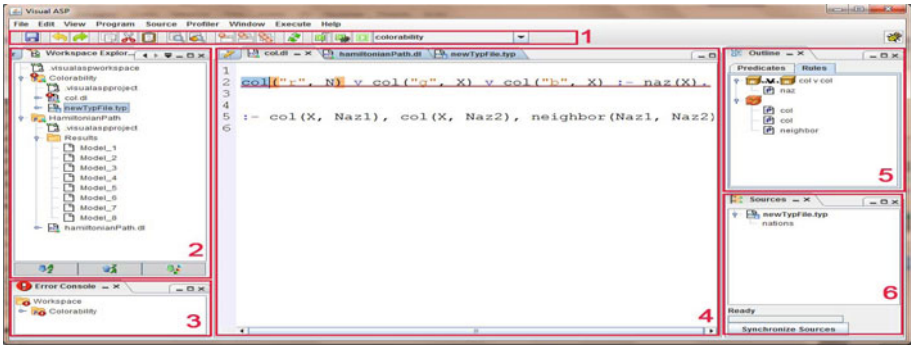

Fig. 1. The *ASPIDE* graphical user interface

modules (graphically implemented by interface panels) are notified by proper events in case of changes in the system status. *ASPIDE* exploits: (i) the JGraph (`http://www.jgraph.com/`) library in the visual editor, in the dependency graph and in the workflow modules; (ii) the DLV Wrapper [42] for interacting with DLV; and, (iii) JDBC libraries for database connectivity. Debugging and profiling are implemented by wrapping the tool *spock* [32], and the *DLV profiler* [41], respectively. Visual editing is handled by including an extended version of the *VisualASP* tool [39].

4 Usage Example

In the following we report an usage example by describing how to exploit *ASPIDE* for encoding the maximum clique problem. The maximum clique is a classical *NP-complete* problem in graph theory requiring to find a clique (a complete subgraph) with the largest number of vertices in an undirected graph. Suppose that the graph G is specified by using facts over predicates *node* (unary) and *edge* (binary), then the following program solves the problem:

```
% Guess the clique
r1: inClique(X1) v outClique(X1) :- node(X1).
% Order edges in order to reduce checks
r2: uedge(X1,X2) :- edge(X1,X2), X1 < X2.
r3: uedge(X2,X1) :- edge(X1,X2), X2 < X1.
% Ensure property.
r4: :- inClique(X1), outClique(X2), not uedge(X1,X2), X1 < X2.
r5: :~ outClique(X2).
```

The disjunctive rule ($\mathtt{r}_1$) guesses a subset S of the nodes to be in the clique, while the rest of the program checks whether S constitutes a clique, and the weak constraint maximizes the size of S. Here, an auxiliary predicate *uedge* exploits an ordering for reducing the time spent in checking.

Suppose that we have already created a new file named *clique.dl*, which is open in the text editing area. To write the disjunctive rule $\mathtt{r}_1$, we exploit the code template *guess*.

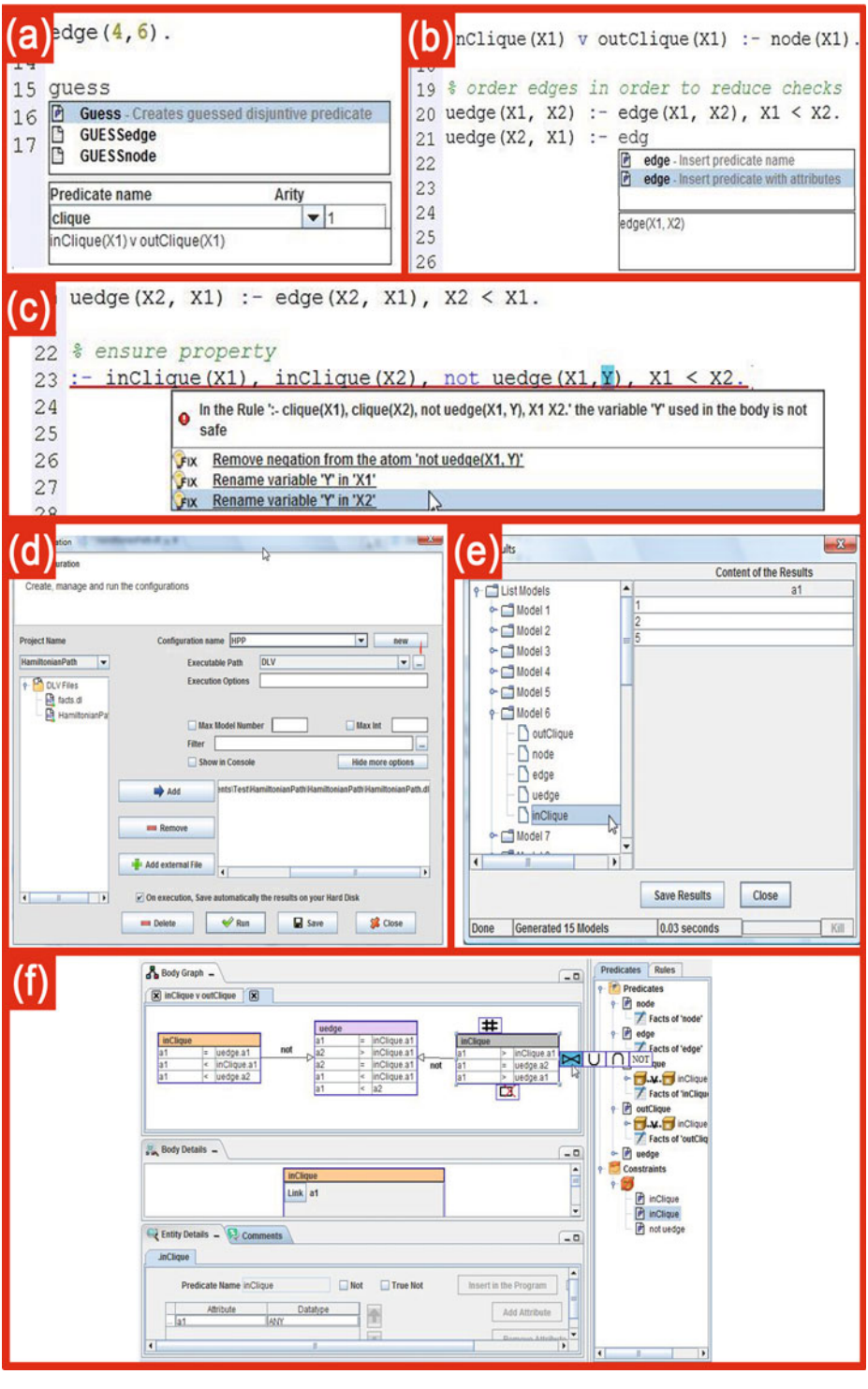

Fig. 2. Using *ASPIDE*

We type the key word *guess* and press CTRL-Space: the text editor opens a popup window where we indicate *clique*; *ASPIDE* auto-composes the head of the disjunctive rule (a preview is displayed on the bottom of the popup, see Fig. 2(a)). After that, we write the predicate *node* in the body. Rules $\mathtt{r}_2$, $\mathtt{r}_3$, $\mathtt{r}_4$ and $\mathtt{r}_5$ are easily written by exploiting the auto-completion (see Fig. 2(b)): atoms are composed by typing only the initial part of the predicate names, and variables are automatically added.

Now, Suppose that a safety error is revealed in rule $\mathtt{r}_4$ (see Fig. 2(c)) since we write variable *Y* instead of *X2* in the negated literal *uedge*; then, the text editor promptly signals the error with a rule highlighting. By double-clicking on the wrong rule, the system will open a popup window reporting the error message and a list of possible quick fixes. With a simple click on the third proposal, variable *Y* is renamed in *X2*, and the problem is easily detected and fixed. Now, we can execute the program by setting a run configuration. To open the run configuration window (Fig. 2(d)), we select *Show Run Configuration* from the menu *Execute*. Using that window we choose the program file *clique.dl* and select DLV as solver. Finally we click on the *Run* button and a user-friendly window shows the results (see Fig. 2(e)).

Note that, *ASPIDE* supports many different ways of creating and modifying logic programs and files. For instance, the same encoding can be graphically composed by exploiting the visual editor (see Fig. 2(f)). For respecting the space constraints, we reported only one of the possible combinations of commands and shortcuts that can be exploited for writing the encoding of the considered problem. The reader can try *ASPIDE* by downloading it from the system website `http://www.mat.unical.it/ricca/aspide`.

5 Related Work

Different tools for developing ASP programs have been proposed up to now, including editors and debuggers [32,33,34]. Moreover, people from the logic programming community, and especially Prolog programmers are already exploiting tools for assisted program development. Some support to the development of logic programs is also present in environments conceived for logic-based ontology languages, which, besides graphical ontology development, also allow for writing logic programs (e.g., to reason on top of the knowledge base). To the best of our knowledge the systems which are closer to our are the following:

- *OntoDLV* [44], an ASP-based system for ontology management and reasoning on top of ontologies.
- *OntoStudio* (`http://www.ontoprise.de`), a commercial modeling environment for the creation and maintenance of ontologies; it also allows for writing logic programs according with to F-Logic molecules syntax.
- *DES* (`http://www.fdi.ucm.es/profesor/fernan/des/`): a deductive database system that supports both Datalog and SQL; it supports querying, debugging and testing of Datalog programs, and supports several DBMSs.
- *DLV!sual* (`http://thp.io/2009/dlvisual`): a GUI frontend for DLV, which allows for developing programs as well as graphically-browsing the answer sets;

- *Visual DLV* [30]: a graphical integrated environment for developing DLV programs; it helps the programmers during the development phases, supports the interaction with external DBMS and features a naïve debugging tool;
- *Digg* (`http://www.ezul.net/2010/09/gui-for-dlv.html`): a simple Java application conceived for learning ASP and experimenting with DLV;
- *APE* [31]: an Eclipse plug-in that allows users for writing ASP programs in the lparse/gringo language;
- *J-Prolog*(`http://www.trix.homepage.t-online.de/JPrologEditor`): an IDE for the Prolog language, written in Java.
- *ProDT* (`http://prodevtools.sourceforge.net`): an Eclipse plugin for developing Prolog projects.
- *PDT* (`http://roots.iai.uni-bonn.de/research/pdt`): an Eclipse plugin for developing Prolog projects.
- *ProClipse* [45]): an Eclipse plugin for developing Prolog projects;
- *Amzi! Prolog* (`http://www.amzi.com/products/prolog_products.htm`: an Eclipse plugin for developing Prolog projects.

People interested in programming, in general, wish to have, in those systems, a set of editing features (which are already available for a long time in development tools for imperative languages) like syntax coloring, syntax highlighting, code completion, error management, quick fix, etc. Debugging, profiling and testing tools, as well as the capability of organizing programs in projects are fundamental for assisting the development of complex applications. Moreover, the declarative nature of logic programming languages makes them good candidates for developing tools which allow for writing programs in a fully graphical way. In Table 1 we compare *ASPIDE* and the above listed tools by separately considering general desirable features. A check symbol indicates that a system provides (in a more or less sophisticated way) a feature. We first note that *ASPIDE* is the most complete proposal, followed by the commercial product *OntoStudio*; and, if we restrict our attention to competing systems tailored for ASP, APE follows *ASPIDE*.

Note that, execution results are reported by most systems only in a textual form (only *ASPIDE*, OntoDLV and OntoStudio offer a graphical view of them in intuitive tables), the outline of the program is often missing, and the execution of systems/solvers is not handled in an effective way. Moreover, only *ASPIDE*, OntoStudio and APE show the

dependency graphs in a graphical way, and the detection of errors during the editing phase, as well as (some form of) debugging and testing are offered by a few systems. Interaction with databases, often required by applications, is supported by only 5 systems out of 13 (data integration only by two).

Conversely, text editing is supported by all systems, and in order to provide a more precise picture regarding this central feature we have deepened the analysis by considering, also, more advanced editing functionalities and support for project management (see Table 1). Also in this case, *ASPIDE* is the system offering more features. Surprisingly, *OntoStudio* lacks advanced text editing features, which are conversely provided by the more mature environments for Prolog. It is worth noting that, systems based on the Eclipse platform, which eases the development of text editors, provide quite a number of editing and project management features (see, e.g., APE). In general, many

Table 1. Logic programming tools comparison

General Features	ASPIDE	OntoDLV	OntoStudio	DES	DLVSUAL	VISUAL DLV	Digg	APE	J-Prolog	ProDT	PDT	ProClipse	Amzi! Prolog
Comparing General Features													
Text Editor	V	V	V	V	V	V	V	V	V	V	V	V	V
Visual Editor	V	V	V										
Project Management	V		V	V				V		V	V	V	V
File Content Outline	V	V	V			V		V		V	V	V	V
Test Suite	V		V	V						V			
Integration with databases	V		V	V		V						V	
Debugger	V		V	V		V			V	V	V		V
Command line				V					V	V	V	V	V
Visual Dependency Graph	V		V					V					
Profiler/Tracer	V			V					V	V	V		
Global Error Console	V	V	V			V		V		V	V	V	V
Textual Result Visualization	V	V		V	V	V	V	V	V	V	V	V	V
User Friendly Result Visualization	V	V	V										
Visual Workflow Definition	V												
Detect Error on editing	V		V			V				V	V	V	V
Dynamic Layout	V		V					V		V	V	V	V
Refactor variables and predicates	V		V										
Configuration of the execution	V		V	V	V	V		V		V	V		V
Data Integration	V		V										
Datatype Management	V	V	V	V		V							
Comparing Text Editor Features													
Text Coloring	V			V	V			V	V	V	V	V	V
Parenthesis pair highlighter	V				V			V		V	V	V	
Token pair highlighter	V				V		V	V		V	V	V	
Undo/Redo	V				V			V	V	V	V	V	V
Syntactic Error/Warning highlighter	V							V		V	V	V	
Semantic Error/Warning highlighter	V									V	V	V	
Find/Replace	V			V	V					V	V	V	V
Quick Fix	V									V	V	V	
Auto completion	V					V		V		V	V	V	V
Code Templates	V					V				V	V		
Dynamic Code Template Definition	V										V		
Code Annotation	V												
Automatic Code Indentation	V												
Text Hover (quick info of predicates)												V	
Code Documentation (like JavaDoc)	V											V	
Comparing Project Management Features													
Multi Project	V		V					V		V	V	V	V
Multi File	V		V	V	V	V	V	V	V	V	V	V	V
Sub Folder Organization	V		V					V		V	V	V	V

existing tools provide syntax coloring, but few systems for ASP support code completion and quick fix of errors. It is quite strange that very basic features like undo/redo and find/replace are not supported by all systems. A particular mention merits the Source-to-Source transformation feature, which allows for translating one language to another. Actually, only *OntoStudio* and *DES* support this feature to translate, respectively, from an ontology language to an other one (e.g., F-Logic and RDF) and from Datalog to SQL. Although every considered system supports a textual editor, only *ASPIDE*, OntoDLV and OntoStudio offer a complete graphical editing environment for writing programs. Thus, in Table 2 we report only the systems allowing for graphic composition

Table 2. Visual editing tools: features and language constructs

Comparing Visual Editor Features																	
	Building of Rules	Building of Queries	Editing of facts/instances	QBE/Diagram like style	Collapsing predicates	Join Attributes	Templates	Disjunction	Aggregates	Built-ins	Constraint	Weak Constraint	True negation	Negation as failure	Basic arithmetic function	Error Management	Reverse engineering
ASPIDE	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
OntoDLV		V	V	V		V			V					V			V
OntoStudio	V	V	V	V		V	V		V	V	V			V	V	V	V

of programs. In this case we consider also supported language constructs. Focusing on ASP-based systems *ASPIDE* easily beats *OntoDLV*, which supports only queries. *OntoStudio*, which supports a different logic language, clearly misses many ASP-specific constructs, but provides a rich environment that supports ontology constructs.

6 Conclusion

This paper presents *ASPIDE*, a comprehensive IDE for ASP, combining several tools for supporting the entire life-cycle of logic programs development. A key feature of *ASPIDE* is its rich set of assisted program composition features, like dynamic syntax highlighting, on-line syntax correction, autocompletion, code-templates, quick-fixes, refactoring, visual editing, program browsing etc. Moreover, *ASPIDE* integrates several graphic tools *tailored for ASP*, including: fully-graphical program composition, debugging, profiling, project management, DBMS access, execution configuration, and user-friendly output-handling. Comparing several softwares for developing logic programs, *ASPIDE* is one of the most complete solutions available in the present state of the art.

As far as future work is concerned, we plan to extend *ASPIDE* by improving / introducing additional dynamic editing instruments, and graphic tools. In particular, we plan to enrich the interface with source-to-source transformation (e.g., for supporting seamless conversions from multiple ASP dialects); and, to develop/extend the input/output interfaces for handling multiple ASP solvers. Moreover, we are improving the testing tool by including more advanced approaches such as [43]. We are currently using *ASPIDE* in a logic programming course of the University of Calabria to assess the applicability of the system for teaching ASP.

Acknowledgments. This work has been partially supported by the Calabrian Region under PIA (Pacchetti Integrati di Agevolazione industria, artigianato e servizi) project DLVSYSTEM approved in BURC n. 20 parte III del 15/05/2009 - DR n. 7373 del 06/05/2009.

References

1. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC 9, 365–385 (1991)
2. Lifschitz, V.: Answer Set Planning. In: ICLP 1999, pp. 23–37 (1999)
3. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS 22(3), 364–418 (1997)
4. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM TOCL 7(3), 499–562 (2006)
5. Simons, P.: Smodels Homepage (since 1996), `http://www.tcs.hut.fi/Software/smodels/`
6. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. AI 138, 181–234 (2002)
7. Zhao, Y.: ASSAT homepage (since 2002), `http://assat.cs.ust.hk/`
8. Lin, F., Zhao, Y.: ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers. In: AAAI 2002, Edmonton, Alberta, Canada. AAAI Press / MIT Press (2002)
9. Babovich, Y., Maratea, M.: Cmodels-2: Sat-based answer sets solver enhanced to non-tight programs (2003), `http://www.cs.utexas.edu/users/tag/cmodels.html`
10. Gebser, M., Kaufmann, B., Neumann, A., Schaub, T.: Conflict-driven answer set solving. In: IJCAI 2007, pp. 386–392 (2007)
11. Janhunen, T., Niemelä, I., Seipel, D., Simons, P., You, J.H.: Unfolding Partiality and Disjunctions in Stable Model Semantics. ACM TOCL 7(1), 1–37 (2006)
12. Lierler, Y.: Disjunctive Answer Set Programming via Satisfiability. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 447–451. Springer, Heidelberg (2005)
13. Drescher, C., Gebser, M., Grote, T., Kaufmann, B., König, A., Ostrowski, M., Schaub, T.: Conflict-Driven Disjunctive Answer Set Solving. In: Proceedings of the Eleventh International Conference on Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia, pp. 422–432. AAAI Press, Menlo Park (2008)
14. Gebser, M., Liu, L., Namasivayam, G., Neumann, A., Schaub, T., Truszczyński, M.: The first answer set programming system competition. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 3–17. Springer, Heidelberg (2007)
15. Denecker, M., Vennekens, J., Bond, S., Gebser, M., Truszczyński, M.: The Second Answer Set Programming Competition. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 637–654. Springer, Heidelberg (2009)
16. Balduccini, M., Gelfond, M., Watson, R., Nogueira, M.: The USA-Advisor: A Case Study in Answer Set Planning. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 439–442. Springer, Heidelberg (2001)
17. Baral, C., Gelfond, M.: Reasoning Agents in Dynamic Domains. In: Logic-Based Artificial Intelligence, pp. 257–279. Kluwer, Dordrecht (2000)
18. Baral, C., Uyan, C.: Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, pp. 186–199. Springer, Heidelberg (2001)
19. Friedrich, G., Ivanchenko, V.: Diagnosis from first principles for workflow executions. Tech. Rep., `http://proserver3-iwas.uni-klu.ac.at/download_area/Technical-Reports/technical_report_2008_02.pdf`
20. Franconi, E., Palma, A.L., Leone, N., Perri, S.: Census Data Repair: A Challenging Application of Disjunctive Logic Programming. In: Nieuwenhuis, R., Voronkov, A. (eds.) LPAR 2001. LNCS (LNAI), vol. 2250, pp. 561–578. Springer, Heidelberg (2001)
21. Nogueira, M., Balduccini, M., Gelfond, M., Watson, R., Barry, M.: An A-Prolog Decision Support System for the Space Shuttle. In: Ramakrishnan, I.V. (ed.) PADL 2001. LNCS, vol. 1990, pp. 169–183. Springer, Heidelberg (2001)

22. Leone, N., Gottlob, G., Rosati, R., Eiter, T., Faber, W., Fink, M., Greco, G., Ianni, G., Kałka, E., Lembo, D., Lenzerini, M., Lio, V., Nowicki, B., Ruzzi, M., Staniszkis, W., Terracina, G.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: SIGMOD 2005, Baltimore, Maryland, USA, pp. 915–917. ACM Press, New York (2005)
23. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. In: CUP (2003)
24. Bardadym, V.A.: Computer-Aided School and University Timetabling: The New Wave. In: Burke, E.K., Ross, P. (eds.) PATAT 1995. LNCS, vol. 1153, pp. 22–45. Springer, Heidelberg (1996)
25. Grasso, G., Iiritano, S., Leone, N., Ricca, F.: Some DLV Applications for Knowledge Management. In: Erdem, E., Lin, F., Schaub, T. (eds.) LPNMR 2009. LNCS, vol. 5753, pp. 591–597. Springer, Heidelberg (2009)
26. Grasso, G., Leone, N., Manna, M., Ricca, F.: Gelfond Festschrift. LNCS, vol. 6565. Springer, Heidelberg (2010)
27. Dovier, A., Erdem, E.: Report on application session @lpnmr09 (2009), `http://www.cs.nmsu.edu/ALP/2010/03/report-on-application-session-lpnmr09/`
28. De Vos, M., Schaub, T. (eds.): SEA 2007: Software Engineering for Answer Set Programming, vol. 281. CEUR (2007), `http://CEUR-WS.org/Vol-281/`
29. De Vos, M., Schaub, T. (eds.): SEA 2009: Software Engineering for Answer Set Programming, vol. 546. CEUR (2009), `http://CEUR-WS.org/Vol-546/`
30. Perri, S., Ricca, F., Terracina, G., Cianni, D., Veltri, P.: An integrated graphic tool for developing and testing DLV programs. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA 2007), pp. 86–100 (2007)
31. Sureshkumar, A., Vos, M.D., Brain, M., Fitch, J.: APE: An AnsProlog* Environment. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA 2007), pp. 101–115 (2007)
32. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: That is Illogical Captain! The Debugging Support Tool spock for Answer-Set Programs: System Description. In: Proceedings of the Workshop on Software Engineering for Answer Set Programming (SEA 2007), pp. 71–85 (2007)
33. Brain, M., De Vos, M.: Debugging Logic Programs under the Answer Set Semantics. In: Proceedings ASP 2005 - Answer Set Programming: Advances in Theory and Implementation, Bath, UK (2005)
34. El-Khatib, O., Pontelli, E., Son, T.C.: Justification and debugging of answer set programs in ASP. In: Proceedings of the Sixth International Workshop on Automated Debugging. ACM, New York (2005)
35. Oetsch, J., Pührer, J., Tompits, H.: Catching the ouroboros: On debugging non-ground answer-set programs. In: Proc. of the ICLP 2010 (2010)
36. Brain, M., Gebser, M., Pührer, J., Schaub, T., Tompits, H., Woltran, S.: Debugging asp programs by means of asp. In: Baral, C., Brewka, G., Schlipf, J. (eds.) LPNMR 2007. LNCS (LNAI), vol. 4483, pp. 31–43. Springer, Heidelberg (2007)
37. Calimeri, F., Ianni, G., Ricca, F.: The third answer set programming system competition (since 2011), `https://www.mat.unical.it/aspcomp2011/`
38. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. TPLP 8, 129–165 (2008)
39. Febbraro, O., Reale, K., Ricca, F.: A Visual Interface for Drawing ASP Programs. In: Proc. of CILC 2010, Rende, CS, Italy (2010)
40. Calimeri, F., Ianni, G.: Template programs for Disjunctive Logic Programming: An operational semantics. AI Communications 19(3), 193–206 (2006)
41. Calimeri, F., Leone, N., Ricca, F., Veltri, P.: A Visual Tracer for DLV. In: Proc. of SEA 2009, Potsdam, Germany (2009)

42. Ricca, F.: The DLV Java Wrapper. In: ASP 2003, Messina, Italy, pp. 305–316 (2003), http://CEUR-WS.org/Vol-78/
43. Janhunen, T., Niemelä, I., Oetsch, J., Pührer, J., Tompits, H.: On testing answer-set programs. In: Proceeding of the 2010 conference on ECAI 2010: 19th European Conference on Artificial Intelligence, pp. 951–956. IOS Press, Amsterdam (2010)
44. Ricca, F., Gallucci, L., Schindlauer, R., Dell'Armi, T., Grasso, G., Leone, N.: OntoDLV: an ASP-based system for enterprise ontologies. Journal of Logic and Computation (2009)
45. Bendisposto, J., Endrijautzki, I., Leuschel, M., Schneider, D.: A Semantics-Aware Editing Environment for Prolog in Eclipse. In: Proc. of WLPE 2008 (2008)