ASPIDE the Integrated Development Environment for ASP: Progress Report

Onofrio Febbraro
DLVSystem s.r.l.
P.zza Vermicelli, Polo Tecnologico
87036 Rende, Italy
febbraro@dlvsystem.com

Nicola Leone and **Kristian Reale**
Dipartimento di Matematica
Università della Calabria
87036 Rende, Italy
{leone,reale}@mat.unical.it

Francesco Ricca
Dipartimento di Matematica
Università della Calabria
87036 Rende, Italy
{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; and *ASPIDE* is the most comprehensive Integrated Development Environments (IDE) for ASP. *ASPIDE* supports the entire life-cycle of the development of ASP-based applications from (assisted) programs editing to application deployment. We summarize the salient features of the current version of *ASPIDE*, and describe the ones that we have recently improved and/or newly introduced for testing, database management, and extensibility via custom plug-ins.

Introduction

In the area of non-monotonic reasoning, according to (Baral 2003), one of the most prominent declarative programming paradigms is Answer Set Programming (ASP) (Lifschitz 1999). A computational problem is represented in ASP by a logic program (set of rules) whose answer sets (also called stable models (Gelfond and Lifschitz 1991)) correspond to problem's solutions, which can be, thus, effectively computed by an ASP solver (Lifschitz 1999). The language of ASP is expressive (Eiter, Gottlob, and Mannila 1997): it can deal with all problems in the second level of the polynomial hierarchy; and the availability of efficient implementations (Leone et al. 2006; Simons, Niemelä, and Soininen 2002; Gebser et al. 2007; Janhunen et al. 2006; Lierler 2005; Calimeri et al. 2011) made ASP an effective tool for developing advanced real-world applications of non-monotonic reasoning. At the time of this writing, the applications of ASP belong to several fields ranging from Artificial Intelligence (Calimeri et al. 2011; Balduccini et al. 2001; Baral and Gelfond 2000; Baral and Uyan 2001; Franconi et al. 2001; Nogueira et al. 2001) to Information Integration (Leone et al. 2005), and Knowledge Management (Baral 2003; Bardadym 1996; Grasso et al. 2009). Moreover, in the last few years, ASP has been employed for developing some industrial application (Ricca et al. 2010; 2011; Grasso et al. 2011).

In order to facilitate the design of ASP applications, some tools for ASP-program development were proposed that range from specialized editors (Perri et al. 2007; iGrom 2010; DLV!sual 2009; Oetsch et al. 2011; Ricca et al. 2009) to debuggers (Brain et al. 2007b; Brain and De Vos 2005; El-Khatib, Pontelli, and Son 2005; Oetsch et al. 2011; Brain et al. 2007a). In the last few years, the first Integrated Development Environments (IDE) (including several tools in the same framework) were presented (Sureshkumar et al. 2007; Oetsch, Pührer, and Tompits 2011a; Febbraro, Reale, and Ricca 2011). Among them, one of the most comprehensive[1] is *ASPIDE* (Febbraro, Reale, and Ricca 2011). *ASPIDE* supports the entire life-cycle of ASP development, from (assisted) program editing to application deployment. *ASPIDE* combines a cutting-edge *editing tool* with a collection of user-friendly *graphical tools* for program composition, debugging, testing, profiling, DBMS access, solver execution configuration and output-handling.

This paper reports on the progress made since the first system presentation of *ASPIDE* at LPNMR'11 (Febbraro, Reale, and Ricca 2011), and describes the new features available in the upcoming major release of March 2012.

System Features

In this section the functionalities provided by *ASPIDE* since its inception (and their improvements) are described first, then the new features are presented.

Features since LPNMR'11

Workspace organization. The system allows for organizing ASP programs in projects similar to Eclipse, which are collected in a special directory (called workspace). Complex applications can be organized in modules (or projects) collecting either different parts of an encoding or several equivalent encodings solving the same problem. *ASPIDE* manages: (i) ASP programs (ii) *TYP files* specifying a mapping between program predicates and database tables in the DLV^{DB} syntax (Terracina et al. 2008); (iii) *TEST files* defining unit tests; (iv) *TEXT files* that are opened in a standard text editor; (v) *PLUGIN files* associated to any file format handled by a user defined plug-in. Note that, *ASPIDE* plug-

[1] For an exhaustive comparison among the available tools for logic programming and ASP we refer the reader to (Febbraro, Reale, and Ricca 2011).

ins are one of the major new extensions of the IDE that are detailed in the next section.

Advanced text editor. ASPIDE features an editor tailored for ASP programs that offers, besides the basic functionalities, such as code line numbering, find/replace, undo/redo, copy/paste, also:

- *Text coloring*. The editor performs keyword outlining (such as " :– ") and dynamic highlighting of predicate names, variables, strings, and comments. Specific colors are also exploited for indicating predicates mapped to external databases or properly defined by annotations (more details about annotations are reported in the next section).
- *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. Suggestions exploit also program annotations (more details follow).
- *Refactoring*. The refactoring tool guides programs modification. For instance, variable renaming in a rule is done by considering bindings of variables, and predicate name changes are done selectively so that common side effects of find/replace are avoided by ensuring that variables/predicates/strings occurring in other expressions remain unchanged. Now refactoring can be done by selecting rules and applying some rewriting (possibly implemented in a user-defined plug-in, more details follow).

Outline navigation. ASPIDE creates a graphic outline of both programs and TYP files, which represents language statements. 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. The schema management of predicates was improved by adding support for attribute names, datatypes, and external source handling. An additional panel was added to shows the schema annotations.

Dependency graph. The system provides a graphical representation of several variants of the (non-ground) dependency graphs associated to the project.

Dynamic code checking and errors highlighting. Programs are parsed while writing, and arity and safety errors or possible warnings are immediately outlined without the need of saving files. Note that, the checker considers the entire project for errors and warnings, indicating e.g., that atoms with the same predicate name have different arity in several files. This global checking behavior in the current release is applied to any error condition.

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. A number of new quick fixes were added, and several existing ones were improved.

Code template. ASPIDE provides support for assisted writing of rules (guessing patterns, aggregates, etc.), as well as automated writing of entire subprograms (e.g., transitive closure rules) by exploding code templates.

Debugger and Profiler. The debugging interface calls the tool *spock* (Brain et al. 2007b); whereas profiling is done according to (Calimeri et al. 2009).

Configuration of the execution. The RunConfiguration Dialog allows one to set the solver/system executable, setup invocation options and input files. We have improved the usability of this dialog, and we have embedded a direct support of *IDPDraw* (Wittocx since 2009) for graphical visualization of the answer sets.

Presentation of the results. Execution results are presented to the user in a comfortable view combining tabular representation of predicates and a tree-like representation of answer sets. The result of the execution can be also saved in text files for subsequent analysis.

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 viceversa) thanks to a reverse-rengineering mechanism from text to graphical format. An important improvement was the introduction of quick fixes also in the graphical view.

Major Additions

We now present the most remarkable additions, including both novel and completely reengineered tools.

Unit Testing for ASP. In software engineering, the task of testing and validating programs is a crucial part of the lifecycle of software development process and a test conceived for verifying the behavior of a specific part of a program is called unit testing. In *ASPIDE* we have improved the testing feature by introducing a unit testing framework in the style of JUnit. Our testing language allows the developer to specify the rules by composing one or several units, specify one or more inputs and assert a number of conditions on both expected outputs and the expected behavior of subprograms. The sub-program is expected to compose a correct module (Splitting Set (Lifschitz and Turner 1994) or DLP-function (Janhunen et al. 2009)) on the entire program. The obtained test case specification can be run by exploiting an ASP solver, and the assertions are automatically verified by analyzing the output of the execution. A graphical tool helps the programmer in composing test cases and analyzing the execution results. For an exhaustive description the testing language and the graphical tool we refer the reader to (Febbraro et al. 2011).

Annotation management for ASP programs. Meta information are used in many programming languages to give more expressiveness to them for both programmers and compilers. For example, Java uses annotations to add meta-information to classes, methods, functions and so on. In *ASPIDE* we have introduced also annotations for ASP programs for indicating rule names, specify predicate schemas (name, arity, optional data-type), handle database connectivity and so on. Each annotation in *ASPIDE* is included within a comment (so that the solvers can ignore it) and starts by "@". Meta-information given trough annotations is exploited by the IDE for enriching the information displayed in the outlines and providing the user with smarter editing facilities

(such as auto-completion, test case composition, etc.).

Schema management and Interaction with databases. Interaction with external databases is useful in real-world applications, see e.g., (Leone et al. 2005; Grasso et al. 2011). The database management feature of *ASPIDE* has been redesigned to: (i) deal with a more sofisticated schema management; (ii) improve outline views; (iii) add support for *#import/#export* directives of DLV; (iv) support fully-graphical composition of TYP files (Terracina et al. 2008); and (v) extend support for DBMS types. Database oriented applications can be run by setting DLV^{DB} as solver in a run configuration. A data integration scenario (Leone et al. 2005) can be implemented by exploiting these features.

User-defined Plug-ins. In real-world applications input data is usually not encoded in ASP, and the results of a reasoning task specified by an ASP program is expected to be saved in an application-specific format. An important feature introduced in *ASPIDE* is the possibility to extend it with user defined plug-ins. Developers can create libraries for extending *ASPIDE* with: (i) new input formats, (ii) program rewritings, and even (iii) customizing the format of solver results. A rewriting plug-in may encode a procedure that can be applied to rules in the editor (e.g., disjunctive rule shifting can be applied on the fly by selecting rules in the editor and applying the mentioned rewriting).

Now consider a scenario where: input data is generated from a spreadsheet; some complex reasoning task has to be performed on it, and the output has to be loaded back on the spreadsheet for further processing. Thus, data has to be first transformed in a database of facts by applying a suitable knowledge representation. One might export data in CSV and apply a transformation script to obtain this; in turn, the programmer develops an ASP program for reasoning on the input data; finally, the results printed by an ASP solver might be converted back in CSV. An input plug-in can take care of the CSV input files that appear in *ASPIDE* as a logic program, and an output plug-in can handle the external conversion of the computed answer sets in CSV. An entire ASP-based application can, in this way, be developed and tested in *ASPIDE* with minimal (or no) need for external conversion tools.

An SDK distributed under LGPL allows to develop new plug-ins for extending *ASPIDE* with custom program rewritings, new input/output formats and application-specific features.

Implementation and Availability

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 and custom plug-ins). *ASPIDE* exploits: (i) the JGraph (`http://www.jgraph.com/`) library for the visual editor; (ii) the DLV Wrapper (Ricca 2003) for interacting with the solver; and, (iii) JDBC libraries for database connectivity. Currently, *ASPIDE* is able to load and store ASP programs in the syntax of the ASP system DLV (Leone et al. 2006), and supports the `ASPCore` language profile employed in the ASP System Competition 2011(Calimeri et al. 2011). *ASPIDE* is available for all the major operating systems, including Linux, Mac OS and Windows, and can be downloaded from the system website `http://www.mat.unical.it/ricca/aspide`.

Conclusion

Since its first presentation at LPNMR'11 in Vancouver, *ASPIDE* has improved in several respects. Many existing components were made more usable and stable; and new remarkable additions were developed: (i) the testing environment was completely reengineered to support a new unit testing language (Febbraro et al. 2011); (ii) the database management tool has been completely restructured to seamlessly integrate/generate TYP files and import directives; (iii) program annotations for ASP were introduced; and, finally, (iv) the support for extending *ASPIDE* by user-defined plug-ins has been integrated into the system.

We are constantly following suggestions and implementing the requests of the system's users, as well as the needs of the developers of DLVSYSTEM s.r.l. (a spin-off company of the University of Calabria developing ASP-based commercial products). Some of the future additions will include: (semi)automatic test case generation based on (Janhunen et al. 2010); a new graphical debugging tool supporting the methodology of (Oetsch, Pührer, and Tompits 2011b); and a library of optional plug-ins for supporting various input/output formats and program rewritings (including a plug-in for source-to-source transformation for seamless conversions from multiple ASP dialects handling all the major ASP systems).

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

Balduccini, M.; Gelfond, M.; Watson, R.; and Nogeira, M. 2001. The USA-Advisor: A Case Study in Answer Set Planning. In *LPNMR-01*, LNCS 2173, 439–442.

Baral, C., and Gelfond, M. 2000. Reasoning Agents in Dynamic Domains. In *Logic-Based Artificial Intelligence*. Kluwer. 257–279.

Baral, C., and Uyan, C. 2001. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In *LPNMR-01*, LNCS 2173, 186–199.

Baral, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving*. CUP.

Bardadym, V. A. 1996. Computer-Aided School and University Timetabling: The New Wave. In *PTAT'95*, LNCS 1153, 22–45.

Brain, M., and De Vos, M. 2005. Debugging Logic Programs under the Answer Set Semantics. In *ASP'05*.

Brain, M.; Gebser, M.; Pührer, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2007a. Debugging asp programs by means of asp. In *LPNMR'07*, LNCS 4483, 31–43. Tempe, Arizona:

Brain, M.; Gebser, M.; Pührer, J.; Schaub, T.; Tompits, H.; and Woltran, S. 2007b. That is Illogical Captain! The Debugging Support Tool spock for Answer-Set Programs: System Description. In *SEA 07*, 71–85.

Calimeri, F.; Leone, N.; Ricca, F.; and Veltri, P. 2009. A Visual Tracer for DLV. In *Proc. of SEA'09*.

Calimeri, F.; Ianni, G.; Ricca, F.; Alviano, M.; Bria, A.; Catalano, G.; Cozza, S.; Faber, W.; Febbraro, O.; Leone, N.; Manna, M.; Martello, A.; Panetta, C.; Perri, S.; Reale, K.; Santoro, M. C.; Sirianni, M.; Terracina, G.; and Veltri, P. 2011. The third answer set programming competition: Preliminary report of the system competition track. In *LPNMR*, LNCS 6645, 388–403.

DLV!sual. 2009. DLV!sual homepage. `http://thp.io/2009/dlvisual`.

Eiter, T.; Gottlob, G.; and Mannila, H. 1997. Disjunctive Datalog. *ACM TODS* 22(3):364–418.

El-Khatib, O.; Pontelli, E.; and Son, T. C. 2005. Justification and debugging of answer set programs in ASP. In *Proc. of Automated Debugging*. California, USA: ACM.

Febbraro, O.; Leone, N.; Reale, K.; and Ricca, F. 2011. Unit testing in aspide. *CoRR* abs/1108.5434.

Febbraro, O.; Reale, K.; and Ricca, F. 2011. ASPIDE: Integrated Development Environment for Answer Set Programming. In *LPNMR 2011*, LNCS 6645, 317–330.

Franconi, E.; Palma, A. L.; Leone, N.; Perri, S.; and Scarcello, F. 2001. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In *LPAR 2001*, LNCS 2250, 561–578.

Gebser, M.; Kaufmann, B.; Neumann, A.; and Schaub, T. 2007. Conflict-driven answer set solving. In *IJCAI 2007*, 386–392.

Gelfond, M., and Lifschitz, V. 1991. Classical Negation in Logic Programs and Disjunctive Databases. *NGC* 9:365–385.

Grasso, G.; Iiritano, S.; Leone, N.; and Ricca, F. 2009. Some DLV Applications for Knowledge Management. In *LPNMR 2009*, LNCS 5753, 591–597.

Grasso, G.; Leone, N.; Manna, M.; and Ricca, F. 2011. *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of M. Gelfond*, LNCS 6565

iGrom. 2010. iGrom on sourceforge. `http://igrom.sourceforge.net/`.

Janhunen, T.; Niemelä, I.; Seipel, D.; Simons, P.; and You, J.-H. 2006. Unfolding Partiality and Disjunctions in Stable Model Semantics. *ACM TOCL* 7(1):1–37.

Janhunen, T.; Oikarinen, E.; Tompits, H.; and Woltran, S. 2009. Modularity aspects of disjunctive stable models. *J. Artif. Intell. Res. (JAIR)* 35:813–857.

Janhunen, T.; Niemelä, I.; Oetsch, J.; Pührer, J.; and Tompits, H. 2010. On testing answer-set programs. In *ECAI 2010*, 951–956. IOS Press.

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.; and Terracina, G. 2005. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *SIGMOD 2005*, 915–917. : ACM Press.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV System for Knowledge Representation and Reasoning. *ACM TOCL* 7(3):499–562.

Lierler, Y. 2005. Disjunctive Answer Set Programming via Satisfiability. In *LPNMR'05*, LNCS 3662, 447–451.

Lifschitz, V., and Turner, H. 1994. Splitting a Logic Program. In *ICLP'94*, 23–37. : MIT Press.

Lifschitz, V. 1999. Answer Set Planning. In *ICLP'99*, 23–37. Las Cruces, New Mexico, USA:

Nogueira, M.; Balduccini, M.; Gelfond, M.; Watson, R.; and Barry, M. 2001. An A-Prolog Decision Support System for the Space Shuttle. In *PADL 2001*, LNCS 1990, 169–183.

Oetsch, J.; Pührer, J.; Seidl, M.; Tompits, H.; and Zwickl, P. 2011. Videas: A development tool for answer-set programs based on model-driven engineering technology. In *LPNMR*, 382–387.

Oetsch, J.; Pührer, J.; and Tompits, H. 2011a. The sealion has landed: An ide for answer-set programming—preliminary report. In *INAP2011/WLP2011*, volume abs/1109.3989.

Oetsch, J.; Pührer, J.; and Tompits, H. 2011b. Stepping through an answer-set program. In *LPNMR*, 134–147.

Perri, S.; Ricca, F.; Terracina, G.; Cianni, D.; and Veltri, P. 2007. An integrated graphic tool for developing and testing DLV programs. In *SEA 07*, 86–100.

Ricca, F.; Gallucci, L.; Schindlauer, R.; Dell'Armi, T.; Grasso, G.; and Leone, N. 2009. OntoDLV: an ASP-based system for enterprise ontologies. *JLC*.

Ricca, F.; Dimasi, A.; Grasso, G.; Ielpa, S. M.; Iiritano, S.; Manna, M.; and Leone, N. 2010. A Logic-Based System for e-Tourism. *FI* 105((1–2)):35–55.

Ricca, F.; Grasso, G.; Alviano, M.; Manna, M.; Lio, V.; Iiritano, S.; and Leone, N. 2011. Team-building with Answer Set Programming in the Gioia-Tauro Seaport. *TPLP*.

Ricca, F. 2003. The DLV Java Wrapper. In *ASP'03*, 305–316. Online at `http://CEUR-WS.org/Vol-78/`.

Simons, P.; Niemelä, I.; and Soininen, T. 2002. Extending and Implementing the Stable Model Semantics. *AI* 138:181–234.

Sureshkumar, A.; Vos, M. D.; Brain, M.; and Fitch, J. 2007. APE: An AnsProlog* Environment. In *SEA 07*, 101–115.

Terracina, G.; Leone, N.; Lio, V.; and Panetta, C. 2008. Experimenting with recursive queries in database and logic programming systems. *TPLP* 8:129–165.

Wittocx, J. since 2009. IDPDraw, a tool used for visualizing answer sets. `http://dtai.cs.kuleuven.be/krr/software/visualisation`.