Datalog Development Tools
(Extended Abstract)

Onofrio Febbraro[1], Giovanni Grasso[2], Nicola Leone[3], Kristian Reale[3], and Francesco Ricca[3]

[1] DLVSystem s.r.l. - P.zza Vermicelli, Polo Tecnologico, 87036 Rende, Italy
febbraro@dlvsystem.com
[2] Oxford University, Department of Computer Science - , Oxford, UK
giovanni.grasso@cs.ox.ac.uk
[3] Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
{leone,reale,ricca}@mat.unical.it

1 Introduction

The recent successful application of Datalog in a number of advanced projects, has renewed the interest in Datalog-based systems for developing real-world applications. Indeed, in the last few years Datalog has been successifully applied in many different areas of computer science, including: Artificial Intelligence, Data Extraction, Information Integration and Knowledge Management. Interestingly, besides the scientific applications, Datalog-based systems were also applied for developing some industrial systems. Nonetheless, in order to boost the adoption of Datalog-based technologies in the scientific community and especially in industry, it is important to provide effective programming tools, which support the activities of researchers and implementors and simplify users' interactions with Datalog systems. Indeed, development frameworks and tools provide indispensable means for assisting and simplifying application development. For this reason, the most popular programming languages and also commercial off-of-the-shelf software products (e.g., DBMSs) are always complemented by Integrated Development Environments (IDE) and Software Development Kits (SDK).

We have dealt with this issue, and we designed development tools for the Datalog-based system DLV [1]. The language of DLV is an extension of Datalog allowing disjunction in rule heads, aggregate atoms, and both weak and strong constraints in rule bodies [1]. DLV is widely considered a state-of-the-art implementation of Disjunctive Datalog under the stable models semantics [2,3] (also called Answer Set Programming (ASP) [4]), and it is undergoing an industrialization process [5] conducted by a spin-off company named DLVSYSTEM s.r.l..

The development tools described in this paper are born from our experience in developing Datalog real-world applications (see. e.g., [6,7]) with DLV. In particular, while implementing Datalog-based applications, we recognized two basic needs: (i) the availability of an IDE supporting Datalog program development (as it is customary for languages like C++ or Java); and (ii) the strong need of integrating Datalog programs and solvers in the well-assessed software-development processes and platforms, which are tailored for imperative/object-oriented programming languages. Indeed, complex business-logic features can be developed with Datalog-based technologies at a lower (implementation) price than in traditional imperative languages, and there are several

© Springer-Verlag Berlin Heidelberg 2012

additional advantages from a Software Engineering viewpoint, in flexibility, readability, extensibility, ease of maintenance, etc. However, since Datalog is not a full general-purpose language, logic programs *must be embedded*, at some point, in systems components that are usually built by employing imperative/object-oriented programming languages, e.g., for developing visual user-interfaces. To respond to the above mentioned needs, we have developed two tools:

1. *ASPIDE* [8]: a complete IDE for disjunctive Datalog programs, 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, database access, solver execution configuration and output-handling; and
2. *JDLV* [9]: a plug-in for the Eclipse platform that implements $\mathcal{JASP}$, a hybrid language that transparently supports a bilateral interaction between disjunctive Datalog and Java. The Datalog program can access Java variables, and the results of the evaluation are automatically stored in Java objects, possibly populating Java collections, transparently. A key ingredient of $\mathcal{JASP}$ is the mapping between (collections of) Java objects and Datalog facts, which can be customized by following widely-adopted standards for Object-Relational Mapping (ORM).

These tools speed-up and empower the development of Datalog-based solutions and their integration in the well-assessed development processes and platforms, which are tailored for imperative/object-oriented programming languages.

2 *ASPIDE*

ASPIDE [8] supports the entire life-cycle of the development of Datalog-based applications, from (assisted) programs editing to application deployment. In the following we overview the main features that make *ASPIDE* one of the most comprehensive development environment for logic programming.[1]

Workspace organization. The system allows for organizing logic programs in projects à la Eclipse, which are collected in a special directory (called workspace). *ASPIDE* allows to manage logic programs in DLV syntax [1] and `ASPCore` [10]; other file types can be added by providing input plugins (see below).

Advanced text editor. *ASPIDE* features an editor tailored for logic programs that offers, besides the basic functionalities also *text coloring*, *automatic completion*, and program *refactorings*.

Outline navigation. *ASPIDE* creates an outline view which graphically represents program elements. 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).

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 from the text editor to the visual one (and vice-versa) thanks to a reverse-rengineering mechanism from text to graphical format.

[1] For an exhaustive comparison among the available tools for logic programming see [8].

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

Dynamic code checking and errors highlighting. Programs are parsed while writing, and both errors or possible warnings are immediately outlined.

Quick fixes and Code templates. The system suggests quick fixes to reported errors or warnings, and provides support for assisted writing of rule patterns (guessing patterns, etc.) by means of code templates that can be instantiated while writing.

Debugger and Profiler. *ASPIDE* embeds the debugging tool *spock* and the DLV Profiler [11].

Unit Testing. The testing feature consists on a unit testing framework for logic programs in the style of JUnit. For an exhaustive description the testing language and the graphical testing tool of *ASPIDE* we refer the reader to [12].

Annotation Management. *ASPIDE* supports annotations for indicating meta information of programs like rule names, predicate name, arity, etc. Meta-information given through annotations is exploited for auto-completion, test case composition, etc.

Schema Management and Interaction with Databases. *ASPIDE* simplifies access to external databases by a graphical tool connecting to DBMSs via JDBC. The database management of *ASPIDE* supports both DLV with ODBC interface and DLV^{DB} [13].

Configuration of the execution and Presentation of the Results. The RunConfiguration Dialog allows one to setup a DLV invocation; whereas the results are presented to the user in a comfortable tabular representation and they can be also saved in text files for subsequent analysis.

User-defined Plugins. *ASPIDE* can be extended with user defined plugins for handling: (i) new input formats, (ii) program rewritings, and even (iii) customizing the visualization/format of results. An input plugin can take care of input files that appear in *ASPIDE* as a logic program, and an output plugin can handle the external conversion of the computed results. A rewriting plugin may encode a procedure that can be applied to rules in the editor.

Availability. *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`.

3 JDLV

We overview in the following a new programming framework blending logic programming with Java and its implementation as plug-in for the Eclipse platform [14], called *JDLV*. *JDLV* is based on $\mathcal{JASP}$ [9], a hybrid language that transparently supports a bilateral interaction between (disjunctive) Datalog and Java. A key ingredient of $\mathcal{JASP}$ is the mapping between (collections of) Java objects and ASP facts. $\mathcal{JASP}$ shares with Object-Relational Mapping (ORM) frameworks, such as Hibernate and TopLink, the structural issues of the *impedance mismatch* [15,16] problem. In $\mathcal{JASP}$, Java Objects are mapped to logic facts (and vice versa) by adopting a structural mapping strategy similar to the one employed by ORM tools for retrieving/saving persistent objects from/to relational databases. $\mathcal{JASP}$ supports both a default mapping strategy, which fits the most common programmers' requirements, and custom ORM specifications, which comply with the Java Persistence API (JPA) [17], to perfectly suit enterprise application development standards. In this section, we present $\mathcal{JASP}$ by exploiting an example. We refer the reader to [9] for a complete description of the language and its semantics.

Integrating Disjunctive Datalog with Java. The $\mathcal{JASP}$ code is very natural and intuitive for a programmer skilled in both Disjunctive Datalog and Java; we introduce it by exploiting an example program, in which a monolithic block of plain Datalog code (called *module*) is embedded in a Java class, which is executed "in-place", i.e., the solving process is triggered at the end of the module specification. Consider the following $\mathcal{JASP}$ code:

```
1 class GraphUtil {
   public static Set<Colored> compute3Coloring(Set<Arc> arcs,
3                                                Set<String> nodes ){
    Set<Colored> res = new HashSet<Colored>();
5   <# in=arcs::arc, nodes::node out=res::col
       col(X,red) v col(X,green) v col(X,blue) :- node(X).
7      :- col(X,C), col(Y,C), arc(X,Y).   #>
    if_no_answerset { res = null; }
9   return res; }}
```

GraphUtil defines the method compute3Coloring(), that encompass a $\mathcal{JASP}$-module to computes a 3-coloring of the given graph. The parameters arcs and nodes are mapped to corresponding predicates (Line 5) `arc` and `node`, respectively, whereas the local variable res is mapped as output variable to the predicate `col` (Line 5). Intuitively, when compute3Coloring() is invoked, Java objects are transformed into logic facts, by applying a default ORM strategy. In this example, each string x in nodes is transformed in unary facts `node(x)`; similarly, each instance of Arc in the variable arcs produces a binary fact, e.g., `arc(from,to)`. These facts are input of the logic program, which is evaluated "in-place". If no 3-coloring exists, the variable res is set to **null** (Line 8); else, when the first result (called *answer set* [4]) is computed, for each fact `col` contained in the solution a new object of the class Colored is created and added to res, which, in turn, is returned by the method. Syntactically, $\mathcal{JASP}$ directly extends the syntax of Java by few new keywords (e.g.,« <# »,« #> »), in such a way that $\mathcal{JASP}$ module statements are allowed in Java *block statements*. Concerning the syntax allowed within modules, $\mathcal{JASP}$ is compliant with the language of DLV. The $\mathcal{JASP}$'s default ORM strategy uses compound keys, i.e., keys made of all basic attributes, and *embedded values*, for one to one associations. This choice naturally fits the usual way of representing information in Datalog, e.g., in the example, one fact models one node. Such mapping can be inverted to obtain Java objects from logic facts. Although, this strategy poses (a few) restrictions to Java specifications (e.g., such as non-recursive type definition, bean-like structure), based on our experience, it is sufficient to handle common use cases.

In the example above we have employed the basic syntax of $\mathcal{JASP}$. The language supports also other advanced features conceived for easing the development of complex applications, including: syntactic enhancements (e.g., named non-positional notation), incremental programs (to enable building programs throughout the application), access to Java variables (for accessign the Java environment), database table mappings (to directly access data stored in a DBMS), and complex mappings with JPA [17] annotations (for complex ORM). Indeed, $\mathcal{JASP}$ spouses the work done in the field ORM and supports the standard JPA Java annotations for defining how Java classes map to relations (logic predicates).

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. Giovanni Grasso has received funding from the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement DIADEM, no. 246858.

References

1. 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)
2. Eiter, T., Gottlob, G., Mannila, H.: Disjunctive Datalog. ACM TODS 22(3), 364–418 (1997)
3. Gelfond, M., Lifschitz, V.: Classical Negation in Logic Programs and Disjunctive Databases. NGC 9, 365–385 (1991)
4. Lifschitz, V.: Answer Set Planning. In: ICLP 1999, pp. 23–37 (1999)
5. Grasso, G., Leone, N., Manna, M., Ricca, F.: ASP at Work: Spin-off and Applications of the DLV System. In: Balduccini, M., Son, T.C. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning. LNCS, vol. 6565, pp. 432–451. Springer, Heidelberg (2011)
6. Ricca, F., Grasso, G., Alviano, M., Manna, M., Lio, V., Iiritano, S., Leone, N.: Team-building with answer set programming in the gioia-tauro seaport. TPLP 12(3), 361–381 (2012)
7. Ricca, F., Dimasi, A., Grasso, G., Ielpa, S.M., Iiritano, S., Manna, M., Leone, N.: A Logic-Based System for e-Tourism. FI 105(1–2), 35–55 (2010)
8. Febbraro, O., Reale, K., Ricca, F.: ASPIDE: Integrated Development Environment for Answer Set Programming. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 317–330. Springer, Heidelberg (2011)
9. Febbraro, O., Grasso, G., Leone, N., Ricca, F.: JASP: a framework for integrating Answer Set Programming with Java. In: Proc. of KR 2012. AAAI Press (2012)
10. 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., Veltri, P.: The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In: Delgrande, J.P., Faber, W. (eds.) LPNMR 2011. LNCS, vol. 6645, pp. 388–403. Springer, Heidelberg (2011)
11. Calimeri, F., Leone, N., Ricca, F., Veltri, P.: A Visual Tracer for DLV. In: Proc. of SEA 2009, Potsdam, Germany (2009)
12. Febbraro, O., Leone, N., Reale, K., Ricca, F.: Unit testing in aspide. CoRR abs/1108.5434 (2011)
13. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. TPLP 8, 129–165 (2008)
14. Eclipse: Eclipse (2001), `http://www.eclipse.org/`
15. Maier, D.: Representing database programs as objects. In: Advances in Database Programming Languages, pp. 377–386. ACM Press (1990)
16. Keller, A.M., Jensen, R., Agrawal, S.: Persistence software: Bridging object-oriented programming and relational databases. In: Proc. of ACM SIGMOD 1993, pp. 523–528. ACM Press (1993)
17. Oracle: JSR 317: JavaTM Persistence 2.0 (2009), `http://jcp.org/en/jsr/detail?id=317`