On the Application of the Answer Set Programming System DLV in Industry: a Report from the Field

Francesco Calimeri[1,2] and Francesco Ricca[1]

[1]Department of Mathematics, University of Calabria, Rende, Italy
{calimeri,ricca}@mat.unical.it
[2]DLVSystem s.r.l., Technest @Unical, Rende, Italy
calimeri@dlvsystem.com

Abstract

Answer Set Programming (ASP) is a declarative logic programming paradigm developed in the area of logic programming and nonmonotonic reasoning. Nowadays, the formal properties of ASP are well-understood, and several efficient ASP systems are available. Among them, DLV is one of the most relevant. DLV is the product of more than twelve years of research, and, notably, it is one of the first ASP systems effectively employed for developing applications at the industrial level. This paper presents some of the most valuable among such applications, and reports on the lessons we have learned on the field using DLV as a powerful tool in industry.

1 Introduction

Answer Set Programming (ASP) is a declarative approach to computer programming stemming roots in the area of nonmonotonic reasoning and logic programming [21, 22, 29, 32]. In its general form, allowing disjunction in rule heads [31] and nonmonotonic negation in rule bodies, ASP can represent every problem in the second level of the polynomial hierarchy [13]. The high knowledge modeling power of ASP, and the availability of a number of efficient ASP systems [9], have recently ignited the interest on this formalism. Indeed, the applications of ASP belong to several fields, including Artificial Intelligence [2, 4, 5, 18, 19, 33, 48], Information Integration [27, 30], Knowledge Management [3, 6, 24], Bioinformatics [12, 20, 34], etc. Moreover, in the last few years, ASP has been exploited also for the development of industrial-level applications [25, 35].

One of the first ASP systems employed for developing commercial applications [25] is DLV [28]. The DLV system is the product of more than twelve years of research and development, and it is nowadays one of the most relevant state-of-the art implementations of ASP [9]. The implementation of DLV started within a project funded by the Austrian Science Funds (FWF) and led by Nicola Leone at Vienna University of Technology (TU Vienna). At present,

DLV is the subject of an international cooperation between the University of Calabria and the TU Vienna. After the very first release, it has been significantly improved over and over in the years, both by enriching the language and optimizing the computational machinery [1, 7, 14, 15, 36, 39, 46, 47], thus making it suitable for developing real-world applications.

In the last few years, DLVSystem s.r.l. [16], a spin-off company of the University of Calabria, has started the industrial exploitation of DLV, which is now currently employed in many industrial applications [25, 41].

This paper reports on our *on-the-field* experience in developing industrial applications with DLV. In particular, after briefly describing system language and architecture, we focus on two relevant industrial applications, namely:

- a workforce management system [41] developed under request of a company operating in the international seaport of Gioia-Tauro, where DLV is used for computing optimal allocations of the available personnel in such a way that the right processing of the shoring cargo boats is guaranteed; and,
- IDUM [38], an e-tourism system that exploits ASP for finding the best matches between the offers from the tour operators and the requests of the tourists.

The paper mentions also some other applications and commercial products based on DLV, and eventually reports some lessons we have learned while developing the above-mentioned applications.

2 The DLV System

In this section, we introduce both the language and the architecture of DLV.

2.1 The Language of DLV

We describe in the following the language of the DLV system by means of examples, in order to provide the intuitive meaning of the main constructs. For further details, formal definitions, and discussions about language extensions, we refer the reader to the existing literature [8, 15, 28].

The language of DLV is based on the foundational work by Gelfond and Lifschitz [22], and the main construct is the rule, which is an expression of the form `Head :- Body.` where `Body` is a conjunction of literals and `Head` is a disjunction of atoms. Informally, a rule can be read as follows: "if `Body` is true, then `Head` is true". A rule without a body is called a *fact*, since it models an unconditional truth (for simplicity, the symbol `:-` is omitted), whereas a rule with an empty head, called *strong constraint*, is used to model a condition that must be false in any possible solution. A set of rules is called *program*. The semantics of a program is given by its *answer sets* [22]. A program can be used to model a problem to be solved: the answer sets of the program (which are computed by DLV) correspond one-to-one to the solutions of the modeled problem. Therefore, a program may have no answer set (if the problem has no solution), one (if the problem has a unique solution) or several (if the problem has more than one possible solutions).

As an example, consider the problem of automatically creating an assessment test from a given database of questions, where each question is identified by a unique string, covers a particular topic, and requires an estimated time to be answered. The input data about questions can be represented by means of a set of facts of type `question(q,topic,time)`; in addition, facts of the form `relatedTopic(topic)` specify the topics related to the subject of the test.

Suppose that we are in the case in which only four questions are given, represented by the facts: `question(q1,computerscience,8)`, `question(q2, computerscience,15)`, `question(q3, mathematics,15)`, and `question(q4, mathematics,25)`. Moreover, suppose that *computer science* is the only topic to be covered by the test, and therefore `relatedTopic(computerscience)` is also part of the input facts. The program consisting of these facts only has a unique answer set A_1, featuring exactly the five facts.

Assessment creation amounts then to selecting a set of questions from the database, according to a given specification. In order to single out questions related to the subject of the test, one can write the rule:

```
relatedQuestion(Q) :- question(Q,Topic,Time), relatedTopic(Topic).
```

that can be read: "Q is a question related to the test if Q has a topic related to some of the subjects that have to be assessed". The addition of this rule to the input facts reported earlier yields the only answer set $A_2 = A_1 \cup \{relatedQuestion(q1), relatedQuestion(q2)\}$. Now, the following disjunctive rule can be used in order to determine all the possible subsets of related questions:

```
inTest(Q) v discard(Q) :- relatedQuestion(Q).
```

Intuitively, this rule can be read as: "if Q identifies a related question, then either Q is taken in the test, or Q is discarded." This rule causes the effect of associating each possible choice of related questions with an answer set of the program; indeed, the answer sets of the program $\mathcal{P}$ consisting of the above two rules and the input facts are:

$$A_3 = A_2 \cup \{discard(q1), discard(q2)\}, \quad A_4 = A_2 \cup \{inTest(q1), discard(q2)\},$$
$$A_5 = A_2 \cup \{discard(q1), inTest(q2)\}, \quad A_5 = A_2 \cup \{inTest(q1), inTest(q2)\}$$

corresponding to the four possible choices of questions $\{\}, \{q1\}, \{q2\}, \{q1, q2\}$. Note that the answer sets are minimal with respect to subset inclusion. Indeed, for each question Q there is no answer set in which both $inTest(Q)$ and $discard(Q)$ appear.

At this point, some strong constraints can be used to single out some solutions fulfilling a number of specification requirements. For instance, suppose that we are interested in tests containing only questions requiring less than 10 minutes to be answered. The following constraint models this requirement:

```
:- inTest(Q), question(Q,Topic,Time), Time >= 10.
```

Informally, the constraint can be read as follows: "discard tests having a question requiring less than 10 minutes to be answered." It is easy to see that the program resulting from the addition of this constraint to program $\mathcal{P}$ reported above features two answer sets only, namely A_3 and A_4.

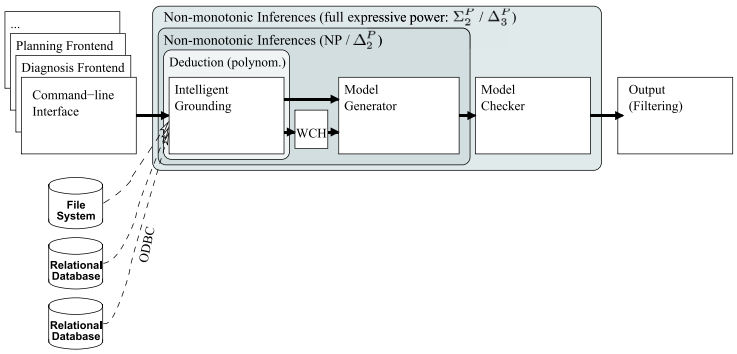

Figure 1: DLV– Overall System Architecture

In order to ease the modeling of a large number of practical cases, DLV supports aggregates [15], a construct similar to aggregation in the SQL language. At present, DLV supports five aggregate functions: `#sum`, `#count`, `#times`, `#max`, `#min`. In our running example, we might want to have a test solvable in an estimated time of less than 60 minutes. This can be achieved thanks to the following strong constraint, featuring an aggregate literal:

```
:- not #sum{Time,Q: inTest(Q), question(Q,Topic,Time)} < 60.
```

The aggregate sums up the estimated solution times of all questions in the test, and the constraint will eliminate all scenarios in which this sum is not less than 60.

2.2 Architecture

We briefly present the architecture of DLV (see Figure 1); for more insights we refer the reader to [28]. The DLV Core (the shaded part of the figure) is an efficient engine for computing answer sets, and has three layers, each of which is a powerful subsystem per se: The *Intelligent Grounder* (IG, also *Instantiator*) has the power of a deductive database system; the *Model Generator* (MG) is as powerful as a Satisfiability Checker; and the *Model Checker* (MC) verifies that a candidate interpretation model is an answer set. In addition to its kernel language described above, DLV provides a number of application front-ends. Each front-end maps its problem specific input into a DLV program, invokes the DLV kernel, and then post-processes any answer set returned, extracting from it the desired solution.

Upon startup, the DLV Core, or one of the front-ends, parses the input and transforms it into proper internal data structures; the input can be read from text files, but DLV also provides a bridge to relational databases through an ODBC interface. The *Intelligent Grounding* module then efficiently generates a ground instantiation of the input that has the same answer sets as the full program instantiation, but is much smaller in general. The heart of the computation is then performed by the *Model Generator* and the Model Checker. Roughly, the former produces some "candidate" answer sets, the stability of which is subsequently verified by the latter. Finally, once an answer set has been found, the control is returned to the front-end in use, if one; this performs

post-processing, and possibly invokes the *Model Generator* to look for further models.

3 Main Industrial Applications of the DLV System

In this section we describe two relevant industrial products having their kernel features implemented by exploiting DLV: the e-tourism system IDUM [38], and the workforce management system developed for a transshipment company operating in the international seaport of Gioia Tauro [41].

3.1 IDUM

A large share of the products in the travel and tourism market are nowadays traded on-line; however, even if the most common e-tourism portals and web sites allow the customer for itinerary arrangements and bookings, no one can guarantee the same feeling and customization of a "traditional" travel agency, where people interact with other people, and possibly know them for years. The need for a more "human" on-line selling process found a solution with the application of Artificial Intelligence techniques and the DLV system, that led to IDUM: *Internet Diventa UMana*.

IDUM has been developed within a technological transfer project funded by the Regione Calabria, Italy, and constitutes an e-tourism system strongly relying on knowledge; it is based on detailed representations of the domains and the users, and has been conceived in order to help both the employees and the customers of any travel agency, in the quest for the best deal while saving time. IDUM can be roughly seen as a "middleman" between offers coming from tour operators and requests coming from customers, and "mimics" the typical behaviors of a travel agent. The system can be accessed via a web-based user interface, thus allowing the agency for a dialog with the customer which actually starts from the very first phase. The system collects the specific needs of the customer (trip kind, preferred means, etc.) and her constraints (time, budget, etc.), and then builds the recommendations that are best suited for her, thanks to a number of reasoning modules implemented with DLV. Basically, the system features the following capabilities.

- Intelligent building of a knowledge base of offers. Indeed, the agency typically receives such offers as electronic leaflets that are barely, or by no means, structured; in additions, they are too many to be effectively useful without the help of some sort of automatic tools.
- Customer profiling. This is pursued collecting information both explicitly, by means of questionnaires, and implicitly, by analyzing the history of the interaction with the system.
- "Semantic" suggestions of products. The system predicts the customer's behavior while examining the offers, according to her profile.

The system allows people for a more "human" interaction with an intelligent guide, and greatly supports the travel agencies, that can easily cancel or reduce the effects of the typical "exploration phase": customers coming to agency in order to get preliminary information, often without purchasing a thing.

Some technical insight. As an example, we briefly talk here about the crucial task of performing a *customized* trip search, where the deductions made by an employee are "simulated" by a set of specifically devised logic programs.

In a typical scenario, the travel agent proposes a number of candidate offers to the customer, and then he has to understand her needs by interpreting her preferences. Such preferences depend on personal information (age, gender, marital status, lifestyle, budget, etc.), but also on her habits (e.g. she prefers going to the mountains, or she has already been in Italy) of the customer. This pieces of information are elicited by interviewing the customer, but most might be already known, while serving a returning customer. The seller has to understand *where* the customer wants to go; *when* she can/wishes to leave; how much time she will dedicate to his holiday; which is the preferred transportation mean (*how*); and, eventually, the available budget. However, the customer usually does not give such information directly, rather the seller has to exploit her knowledge, in order to infer it. This is exactly what the IDUM system does when a user starts a new search. Current needs are specified by filling an appropriate search form, while, notably, an appropriate tourism ontology models both the knowledge of the seller and the profile of customers; at the same time, an automatic extraction process continuously populates the ontology with new tourist offers. The system, by running a specifically devised reasoning module, combines the specified information with the one available in the knowledge base, and shows the packages that best fit the customer needs. For example, suppose that a customer specifies only the kind of holiday and the period; then, the following module[1] (we report next an excerpt of a simplified version) would take care of selecting the appropriate packages.

```
%detect possible and suggested places
possiblePlace(Place)    :- askFor(_,TripKind,_),
                           PlaceOffer(Place, TripKind).

suggestPlace(Place)     :- possiblePlace(Place), askFor(_,_,Period),
                           SuggestedPeriod(Place_1, Period),
                           not BadPeriod(Place_1, Period).

%select possible and alternative offers and suggestible packages
possibleOffer(Offer)    :- TouristOffer(Offer, Place,_), possiblePlace(Place).

alternativeOffer(Offer) :- TouristOffer(Offer, Place,_), suggestPlace(Place).

suggestOffer(Offer)     :- TouristOffer(Offer, Place, Mean),
                           suggestPlace(Place), askFor(Customer,_,_),
                           CustomerPrefersMean(Customer, Mean).
```

Intuitively, the first two rules select places matching the kind of holiday requested and places to be suggested because the corresponding offer matches the time of the year requester. The remaining three rules search in the available holiday packages the ones that offer an holiday that matches the original input (possible offer), or that constitute good alternatives to suggested places (alternativeOffer), or match the customer's preferred mean. Once such reasoning processes have been translated into DLV programs, they constitute a formal and at the same time executable specification, that can be easily adjusted and updated as needed.

[1] Encoding adapted from [38].

In order to provide a concrete idea on the system behavior, we mention here a result reported in [38], where system performances have been assessed in a concrete usage scenario. Given a corpus of 755 tourism leaflets received by a travel agency, corresponding to 10285 offers in total, a logic program that computes the offers matching customer's preferred destination and budget requires less than two seconds on the average to be evaluated on a standard desktop hardware.

The success of IDUM has been confirmed by the flattering appreciation of the *Touring Club Italiano*, the largest Italian organization of travel agents: the president spoke at the system presentation conference, held at the University of Calabria. Major Italian and international operators shown interest, and proposed to offer their own data in order to foster testings on a larger industrial base. At the time of writing, commercial negotiations are in progress.

3.2 Team-Building at Gioia Tauro Seaport

The Gioia Tauro international seaport is the biggest industrial hub on the Mediterranean Sea. The seaport mainly works on transhipment, which consists of transferring a load from an incoming ship to other ships that will head to the final destination. This activity requires the companies working in the seaport to meet many, usually very strong, constraints that are enforced by the transportation companies and cost high non-compliance penalties. In addition, processing the load of a ship is quite a complex job: goods must be managed, processed, stored and sent again. This requires a working team consisting of employees able to play different and very specialized roles, depending on the kind of ship and load. The traffic volume requires to manage a complex shift organization, thus making the task of building adequate teams even more difficult. Each team must fulfill many conditions: some constraints are enforced by the employment contracts (such as the maximum amount of worked hours in a week, for instance), by equity criteria (guarantee a proper workload distribution among employees, especially in case of heavy or dangerous jobs), or simply by the skills required for a particular task. Upon request of the company ICO BLG, that manages logistics within the seaport, a DLV-based "team-building" system has been implemented, that is able to automatically generate proper working groups starting from the description of the required resources and the personnel availability. The system can both build the teams from scratch, and automatically complete partial allocations in case of roles fixed in advance for some employees. Moreover, a friendly user interface allows one to manually modify each team, while automatically checks that the constraints remain fulfilled after each change. In case of errors, the system highlights the reasons and conveniently provide some suggestions to overcome the problems.

Some technical insight. We report below a simplified version of the kernel part of the employed ASP program.[2]

```
(r)       assign(Em,Sh,Sk) v nAssign(Em,Sh,Sk) :- skill(Em,Sk),
               metaPlan(Sh,Sk,_,D), not absent(Em),
               not manuallyExcluded(Em), workedHours(Em,Wh), Wh+D <= 36.
```

[2]Encoding adapted from [23].

```
(c1)      :- metaPlan(Sh,Sk,EmpNum,_),
               #count{ Em : assign(Em,Sh,Sk) } != EmpNum.

(c2)      :- assign(Em,Sh,Sk1), assign(Em,Sh,Sk2), Sk1 != Sk2.

(c3)      :- wstats(Em1,Sk,_,LastTime1), wstats(Em2,Sk,_,LastTime2),
               LastTime1 > LastTime2, assign(Em1,Sh,Sk),
               not assign(Em2,Sh,Sk).

(c4)      :- workedHours(Em1,Wh1), workedHours(Em2,Wh2), threshold(Tr),
               Wh1+Tr < Wh2, assign(Em1,Sh,Sk), not assign(Em2,Sh,Sk).

(r_aux)   workedHours(Em,Wh) :- skill(Em,_),
               #count{ H, Em : wstats(Em,_,H,_) } = Wh.
```

Predicate	**Description**
`skill(employee, skill)`	Employees and their skills
`metaPlan(shift, skill, needed-Employees, duration)`	Meta-plan specification
`wstat(employee, skill, hours, lastTime)`	Weekly statistics. For each employee both the number of worked hours per skill and the last allocation date are specified
`absent(employee)`	Absent employees
`manuallyExcluded(employee)`	Employees excluded by a management decision

Table 1: Predicates from Team-Building description in Section 3.2

The input predicates are described in Table 1. Following the guess&check programming methodology [28], the disjunctive rule `(r)` generates the search space by guessing the assignment of a number of available employees to the shift in the appropriate roles. Absent or manually excluded employees, together with employees exceeding the maximum number of weekly working hours, are automatically discarded. Then, admissible solutions are selected by means of constraints: `(c_1)` discards assignments with an wrong number of employees in some skill; `(c_2)` avoids that an employee covers two roles in the same shift; `(c_3)` implement the tournament of roles; and `(c_4)` guarantees a fair distribution of the workload; `(r_aux)` computes the total number of worked hours per employee.

In order to provide a concrete image on the behavior of the system, the result of an experimental analysis are reported in [41]. In order to give an idea, the management of ICO BLG dedicated several hours per day to manually schedule the next day; whereas the system required a few seconds to generate a shift plan, and in some minutes we could simulate a complete allocation covering an entire month.

The most prominent advantage of a solution based on logics with respect to other approaches, in this application, is given by the purely declarative nature of the specification language. Indeed, the use of DLV allowed to obtain in

very short time a first set of logic rules, which constitutes a logic program able to provide admissible solutions, and then to refine and calibrate it according to problem specifications while directly interacting with the interested people. The resulting logic program is not only very powerful, but also extremely flexible: to change its behavior it is enough to modify, add, or delete the constituting rules by editing simple text files, and checking correctness in "real-time".

4 Other Applications and Industrial Products

We briefly report next on the role of DLV in the development of a number of other applications, as well as some specialized products for knowledge management, information extraction, and content management.

Other applications. The European Commission funded a project on Information Integration, which produced a sophisticated and efficient data integration system, called INFOMIX, which uses DLV at its computational core [27]. The powerful mechanisms for database interoperability, together with magic sets [10, 14] and other database optimization techniques, which are implemented in DLV, make DLV very well-suited for handling information integration tasks. And DLV (in INFOMIX) was succesfully employed to develop in an integration system for the information system of the University of Rome "La Sapienza". The DLV system has been experimented also with an application for Census Data Repair [18], in which errors in census data are identified and eventually repaired.

DLV has been employed at CERN, the European Laboratory for Particle Physics, for an advanced deductive database application that involves complex knowledge manipulation on large-sized databases.

The Polish company Rodan Systems S.A. has exploited DLV in a tool for the detection of price manipulations and unauthorized use of confidential information, which is used by the Polish Securities and Exchange Commission.

In the area of self-healing Web Services, moreover, DLV is exploited for implementing the computation of minimum cardinality diagnoses [19].

In [26] a complete on-line exam taking portal has been described, called EXAM. The system allows teachers and students to be assisted in the whole process of assessment test building, exam taking, and test correction. The system exploits ASP for automatically generating assessment tests based on user defined constraints: a teacher is made able to build up an assessment test template; her preferences are then translated into a logic specification executable by DLV.

DLV is the core of a system that helps the citizens of the Regione Calabria, Italy while moving within the transportation system (including public and private companies). The user can access a web portal and ask the automatic building of a complete itineray, freely choosing any locations as starting and destination points. The system is actually very accurate: it tells the user where and when to take the train/bus, where to get off and change, and how long the trip will last; it even gives precise directions for the distance to be covered on foot.

Knowledge Management Products. Three among the main industrial products offered by the Exeura s.r.l. company (a spin-off of the University of Calabria) are based on DLV, namely: OntoDLV [40, 42], OLEX [11, 45], and HiLεX [44, 43]. OntoDLV [40, 42] is an ontology management and reasoning system; indeed, OntoDLV implements a powerful logic-based ontology representation language, called OntoDLP, which is an extension of (disjunctive) ASP with all the main ontology constructs including classes, inheritance, relations, and axioms. Using OntoDLV, domain experts can create, modify, store, navigate, and query ontologies thanks to a user-friendly visual environment; at the same time, application developers can easily implement knowledge-intensive applications embedding OntoDLP specifications. The OntoLog Enterprise Categorizer System (OLEX) [11, 45] is a corporate classification system supporting the entire content classification life-cycle, including document storage and organization, ontology construction, pre-processing and classification. OLEX exploits a reasoning-based approach to text classification exploiting ASP as categorization rule language. Logic rules provide a natural and powerful way to encode how document contents may relate to ontology concepts. HiLεX [44, 43] is an advanced system for ontology-based information extraction from semi-structured and unstructured documents. HiLεX is based on OntoDLP for describing ontologies, since this language perfectly fits the definition of semantic extraction rules. It is worth mentioning that the extraction module employed in the IDUM system was developed by using HiLεX.

5 Learned Lessons

The strong efforts spent by the scientific community made ASP a powerful tool for problem solving and knowledge representation and reasoning; the availability of a number of efficient and solid systems stimulated the interest in developing ASP-based applications and solutions. The exploitation of ASP for developing real-world, industrial-level applications has started in the last few years; and the DLV system has been one of the first successfully exploited ASP systems in this respect, so far.

In this work we have reported on the applications of DLV for solving real-world problems, mainly focusing on industrial-level applications. The experience we have gained developing real-world DLV-based applications, has confirmed on the one hand, the viability of the industrial exploitation of ASP; on the other hand, it has brought into light some practical problems.

ASP has confirmed on the field some of its strong points, particularly from the perspective of software engineering. Above all, we mention its suitability for fast prototyping and the very high flexibility it provides in the implementation of new requirement specifications. For instance, the key features that allowed the rapid development of an effective solution for Team-Building at Gioia Tauro Seaport were: (*i*) the possibility to design and implement complex reasoning modules at a lower (development) price than "traditional" imperative programming approaches, and (*ii*) the ease of modifying ready-to-execute logic specifications according to continuously changing requirements. The workforce management system was designed side by side with the management of the transshipment company; during this process we had to repeatedly change the model, and we were able to follow all the new directions by the customer,

who could immediately validate its requirements. Indeed, thanks to DLV, we were always able to modify and test the system in a relatively short time. We could hardly have achieved the same level of productivity by working with "traditional" imperative programming approaches: even in presence of a complex workforce model with a large number of constraints, the requirements have been elicited, rapidly transformed in executable specifications, and immediately tested and validated by the customer on a live working instance of the system. Moreover, the knowledge representation and reasoning feature of ASP allowed us to easily develop the customized search features of IDUM.

Nonetheless, the same experience allowed us also to discover what is still missing, or not yet completely adequate, in a software development process where ASP is employed. For instance, we noticed the lack of advanced development tools for ASP that are able to support programmers during the entire development life-cycle, from (assisted) programs editing to the management of large and complex projects. The most polular programming languages always come with SDKs featuring a rich set of tools that significantly simplify both programming and maintenance tasks; this observation led us to start the ASPIDE [17] project. Another important observation is that there is a strong need of integrating ASP technologies (i.e., ASP programs and solvers) in the well-assessed software-development processes and platforms. Indeed, since ASP is not a full general-purpose language, ASP 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. We already developed the DLV Java Wrapper API [37], but we are already working on much more powerful tools, and aiming at a more tight integration of DLV with industrial development tools and imperative languages.

To summarize, our experience confirms ASP can be effectively exploited in industry, but there are some issues that must be faced. In particular, a strong effort must be spent in order to make it easy to apply for developers, by integrating ASP in the most polular development environments. We do believe that a central role will be played by the future advancements in software engineering for ASP, and this is why we are actively involved in this mission on a number of fronts.

6 Acknowledgements

We would like to thank Nicola Leone, as the leader of the DLV team, for the useful directions, and, together with Exeura s.r.l. and DLVSystem s.r.l., for the chance to access the insights of all projects and applications herein cited.

References

[1] Mario Alviano, Wolfgang Faber, and Nicola Leone. Disjunctive asp with functions: Decidable queries and effective computation. *Theory and Practice of Logic Programming, 26th Int'l. Conference on Logic Programming (ICLP 2010) Special Issue*, 10(4–6):497–512, 2010.

[2] Marcello Balduccini, Michael Gelfond, Richard Watson, and Monica Nogeira. The USA-Advisor: A Case Study in Answer Set Planning. In Thomas

Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *LNCS*, pages 439–442. Springer, 2001.

[3] Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge University Press, 2003.

[4] Chitta Baral and Michael Gelfond. Reasoning Agents in Dynamic Domains. In Jack Minker, editor, *Logic-Based Artificial Intelligence*, pages 257–279. Kluwer Academic Publishers, 2000.

[5] Chitta Baral and Cenk Uyan. Declarative Specification and Solution of Combinatorial Auctions Using Logic Programming. In Thomas Eiter, Wolfgang Faber, and Mirosław Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2001)*, volume 2173 of *Lecture Notes in AI (LNAI)*, pages 186–199. Springer Verlag, 2001.

[6] Victor A. Bardadym. Computer-Aided School and University Timetabling: The New Wave. In Edmund Burke and Peter Ross, editors, *Practice and Theory of Automated Timetabling, First International Conference 1995*, volume 1153 of *LNCS*, pages 22–45. Springer, 1996.

[7] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Enhancing Disjunctive Datalog by Constraints. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):845–860, 2000.

[8] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. Computable Functions in ASP: Theory and Implementation. In *Proceedings of the 24th International Conference on Logic Programming (ICLP 2008)*, volume 5366 of *Lecture Notes in Computer Science*, pages 407–424, Udine, Italy, December 2008. Springer.

[9] Francesco Calimeri, Giovambattista Ianni, Francesco Ricca, Mario Alviano, Annamaria Bria, Gelsomina Catalano, Susanna Cozza, Wolfgang Faber, Onofrio Febbraro, Nicola Leone, Marco Manna, Alessandra Martello, Claudio Panetta, Simona Perri, Kristian Reale, Maria Carmela Santoro, Marco Sirianni, Giorgio Terracina, and Pierfrancesco Veltri. The Third Answer Set Programming Competition: Preliminary Report of the System Competition Track. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference (LPNMR 2011), Proceedings*, pages 388–403, Vancouver, Canada, May 2011.

[10] Chiara Cumbo, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Enhancing the magic-set method for disjunctive datalog programs. In *Proceedings of the the 20th International Conference on Logic Programming (ICLP 2004)*, volume 3132 of *Lecture Notes in Computer Science*, pages 371–385, 2004.

[11] Chiara Cumbo, Salvatore Iiritano, and Pasquale Rullo. OLEX - A Reasoning-Based Text Classifier. In *Proceedings of Logics in Artificial Intelligence, 9th European Conference, (JELIA 2004), September 27-30, 2004*,

volume 3229 of *Lecture Notes in Computer Science*, pages 722–725, Lisbon, Portugal, 2004.

[12] Agostino Dovier. Recent constraint/logic programming based advances in the solution of the protein folding problem. *Intelligenza Artificiale*, 5(1):113–117, 2011.

[13] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive Datalog. *ACM Transactions on Database Systems*, 22(3):364–418, September 1997.

[14] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic Sets and their Application to Data Integration. *Journal of Computer and System Sciences*, 73(4):584–609, 2007.

[15] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Semantics and complexity of recursive aggregates in answer set programming. *Artificial Intelligence*, 175(1):278–298, 2011. Special Issue: John McCarthy's Legacy.

[16] Wolfgang Faber and Gerald Pfeifer. `DLV` homepage, since 1996. `http://www.dlvsystem.com/`.

[17] Onofrio Febbraro, Kristian Reale, and Francesco Ricca. Aspide: Integrated development environment for answer set programming. In *Logic Programming and Nonmonotonic Reasoning - 11th International Conference, LPNMR 2011, Vancouver, Canada, May 16-19, 2011. Proceedings*, volume 6645 of *Lecture Notes in Computer Science*, pages 317–330, 2011.

[18] Enrico Franconi, Antonio Laureti Palma, Nicola Leone, Simona Perri, and Francesco Scarcello. Census Data Repair: a Challenging Application of Disjunctive Logic Programming. In *Logic for Programming, Artificial Intelligence, and Reasoning, 8th International Conference (LPAR 2001)*, volume 2250 of *Lecture Notes in Computer Science*, pages 561–578. Springer, 2001.

[19] G. Friedrich and V. Ivanchenko. Diagnosis from first principles for workflow executions. Technical report, Alpen Adria University, Applied Informatics, Klagenfurt, Austria, 2008. `http://proserver3-iwas.uni-klu.ac.at/download\_area/Technical-Reports/t%echnical\_report\_2008\_02.pdf`.

[20] Martin Gebser, Torsten Schaub, Sven Thiele, and Philippe Veber. Detecting inconsistencies in large biological networks with answer set programming. *Theory and Practice of Logic Programming*, 11:323–360, 2011.

[21] Michael Gelfond and Vladimir Lifschitz. The Stable Model Semantics for Logic Programming. In *Logic Programming: Proceedings Fifth Intl Conference and Symposium*, pages 1070–1080, Cambridge, Mass., 1988. MIT Press.

[22] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[23] Giovanni Grasso, Salvatore Iiritano, Nicola Leone, Vincenzino Lio, Francesco Ricca, and Francesco Scalise. An ASP-Based System for Team-Building in the Gioia-Tauro Seaport. In Manuel Carro and Ricardo Peña, editors, *PADL*, volume 5937 of *Lecture Notes in Computer Science*, pages 40–42. Springer, 2010.

[24] Giovanni Grasso, Salvatore Iiritano, Nicola Leone, and Francesco Ricca. Some DLV Applications for Knowledge Management. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Proceedings of the 10th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2009)*, volume 5753 of *Lecture Notes in Computer Science*, pages 591–597. Springer, 2009.

[25] Giovanni Grasso, Nicola Leone, Marco Manna, and Francesco Ricca. Asp at work: Spin-off and applications of the dlv system. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*, volume 6565 of *LNCS*, pages 432–451. Springer, 2011.

[26] Giovambattista Ianni, Francesco Ricca, and Claudio Panetta. Specification of Assessment-Test Criteria through ASP Specification. In *Answer Set Programming: Advances in Theory and Implementation*, pages 293–302, Bath, UK, 2005. Research Press International, P.O. Box 144, Bristol BS 1YA.

[27] Nicola Leone, Georg Gottlob, Riccardo Rosati, Thomas Eiter, Wolfgang Faber, Michael Fink, Gianluigi Greco, Giovambattista Ianni, Edyta Kałka, Domenico Lembo, Maurizio Lenzerini, Vincenzino Lio, Bartosz Nowicki, Marco Ruzzi, Witold Staniszkis, and Giorgio Terracina. The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In *Proceedings of the 24th ACM SIGMOD International Conference on Management of Data (SIGMOD 2005)*, pages 915–917, Baltimore, Maryland, USA, June 2005. ACM Press.

[28] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV System for Knowledge Representation and Reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, July 2006.

[29] Victor W. Marek and Mirosław Truszczyński. Stable Models and an Alternative Logic Programming Paradigm. In Krzysztof R. Apt, V. Wiktor Marek, Mirosław Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm – A 25-Year Perspective*, pages 375–398. Springer Verlag, 1999.

[30] Mónica Caniupán Marileo and Leopoldo E. Bertossi. The consistency extractor system: Answer set programs for consistent query answering in databases. *Data & Knowledge Engineering*, 69(6):545–572, 2010.

[31] Jack Minker. On Indefinite Data Bases and the Closed World Assumption. In Donald W. Loveland, editor, *Proceedings* 6^{th} *Conference on Automated Deduction (CADE '82)*, volume 138 of *Lecture Notes in Computer Science*, pages 292–308, New York, 1982. Springer.

[32] Ilkka Niemelä. Logic Programming with Stable Model Semantics as Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[33] Monica Nogueira, Marcello Balduccini, Michael Gelfond, Richard Watson, and Matthew Barry. An A-Prolog Decision Support System for the Space Shuttle. In I.V. Ramakrishnan, editor, *Practical Aspects of Declarative Languages, Third International Symposium (PADL 2001)*, volume 1990 of *Lecture Notes in Computer Science*, pages 169–183. Springer, 2001.

[34] Luigi Palopoli, Simona E. Rombo, and Giorgio Terracina. Flexible Pattern Discovery with (Extended) Disjunctive Logic Programming. In *Foundations of Intelligent Systems, 15th International Symposium, (ISMIS 2005), Proceedings*, pages 504–513, May 2005. `http://dx.doi.org/10.1007/11425274_52`.

[35] Alessandro Dal Palú and Paolo Torroni. 25 years of applications of logic programming in italy. In Agostino Dovier and Enrico Pontelli, editors, *25 Years GULP*, volume 6125 of *LNCS*, pages 300–328. Springer, 2010.

[36] Simona Perri, Francesco Scarcello, Gelsomina Catalano, and Nicola Leone. Enhancing DLV instantiator by backjumping techniques. *Annals of Mathematics and Artificial Intelligence*, 51(2–4):195–228, 2007.

[37] Francesco Ricca. The DLV Java Wrapper. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP03 - Answer Set Programming: Advances in Theory and Implementation*, pages 305–316, Messina, Italy, September 2003. `http://CEUR-WS.org/Vol-78/`.

[38] Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Maria Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone. A Logic-Based System for e-Tourism. *Fundamenta Informaticae*, 105(1-2):35–55, 2010.

[39] Francesco Ricca, Wolfgang Faber, and Nicola Leone. A Backjumping Technique for Disjunctive Logic Programming. *AI Communications – The European Journal on Artificial Intelligence*, 19(2):155–172, 2006.

[40] Francesco Ricca, Lorenzo Gallucci, Roman Schindlauer, Tina Dell'Armi, Giovanni Grasso, and Nicola Leone. OntoDLV: an ASP-based system for enterprise ontologies. *Journal of Logic and Computation*, 2009.

[41] Francesco Ricca, Giovanni Grasso, Mario Alviano, Marco Manna, Vincenzino Lio, Salvatore Iiritano, and Nicola Leone. Team-building with Answer Set Programming in the Gioia-Tauro Seaport. *Theory and Practice of Logic Programming*, 2011. http://dx.doi.org/10.1017/S147106841100007X.

[42] Francesco Ricca and Nicola Leone. Disjunctive Logic Programming with types and objects: The DLV$^+$ System. *Journal of Applied Logics*, 5(3):545–573, 2007.

[43] Massimo Ruffolo, Nicole Leone, Marco Manna, Domenico Saccà, and Antonio Zavatto. Exploiting ASP for Semantic Information Extraction. In Marina de Vos and Alessandro Provetti, editors, *Proceedings ASP05 - Answer Set Programming: Advances in Theory and Implementation*, pages 248–262, Bath, UK, July 2005.

[44] Massimo Ruffolo and Marco Manna. HiLeX: A System for Semantic Information Extraction from Web Documents. In Yannis Manolopoulos, Joaquim Filipe, Panos Constantopoulos, and José Cordeiro, editors, *ICEIS (Selected Papers)*, volume 3 of *Lecture Notes in Business Information Processing*, pages 194–209, 2008.

[45] Pasquale Rullo, Chiara Cumbo, and Veronica L. Policicchio. Learning rules with negation for text categorization. In Yookun Cho, Roger L. Wainwright, Hisham Haddad, Sung Y. Shin, and Yong Wan Koo, editors, *Proceedings of the 2007 ACM Symposium on Applied Computing (SAC), Seoul, Korea, March 11-15*, pages 409–416. ACM, 2007.

[46] Giorgio Terracina, Erika De Francesco, Claudio Panetta, and Nicola Leone. Enhancing a DLP system for advanced database applications. In *Proceedings of the International Conference on Web Reasoning and Rule Systems (RR 2008)*, Karlsruhe, Germany, 2008. Springer Verlag.

[47] Giorgio Terracina, Nicola Leone, Vincenzino Lio, and Claudio Panetta. Experimenting with recursive queries in database and logic programming systems. *Theory and Practice of Logic Programming*, 8:129–165, 2008.

[48] Wasp showcase. `http://www.kr.tuwien.ac.at/projects/WASP/showcase.html`.