An ASP-based system for preference handling and planning.

Stefania Costantini, Giovanni De Gasperis, Niva Florio, and Claudia Zuppella

Dept. of Information Engineering, Computer Science and Mathematics,
University of L'Aquila, Via Vetoio, Coppito, L'Aquila
{stefania.costantini,giovanni.degasperis,niva.florio}@univaq.it
{claudia.zuppella}@hotmail.it

Abstract. Internet and mobile applications are becoming more and more helpful and widespread, while our daily lives are becoming increasingly busy and complicated. Preferences affect our actions, as well as those of intelligent agents. Answer Set Programming is a suitable framework for a decision making process which aims at supporting users by suitably planning their activities. Resourced Answer Set Programming (RASP) provides mechanisms for the optimization of answer sets according to preferences and resources. In this paper, an ASP-based system integrated with a mobile application able to plan the activities of a user is proposed, taking into account the context in which the user is located, the available resources, the geographical position and user's preferences.

Keywords: answer set programming, logic programming, preference handling, resource management, planning and scheduling

1 Introduction

Everyone's daily life is becoming increasingly complicated and busy, and preferences affect our daily actions and the way in which we try to achieve our goals. On the other side, the technology, and in particular intelligent agents, is potentially able to help us, since mobile applications and the web in particular are becoming more and more helpful and are available on affordable mobile devices. In this scenario, evaluation and adequate handling of user preferences (expressed either in an explicit or in an implicit way) is becoming increasingly useful. In fact, an adequate preference handling system can help the user in the organization of everyday life. In fact, preferences affect the way intelligent agents (including human) act, and their decision-making process.

Approaches concerning preferences in (constraint) logic programming and non-monotonic reasoning are widely studied (cf., e.g., [9], [11], [12] and [8]). More specifically, reasoning on preferences in relation to Answer Set Programming (ASP, cf., e.g., [2], [1], [10] and [13]) has been investigated (see among many [2], [4], [6] and [3]) These approaches introduce preferences either globally (e.g., [4]) or among rules (e.g., [14]). Particularly suitable to our purposes seems to be the Resourced Answer Set Programming (RASP), an extension of ASP that

supports declarative reasoning on consumption and production of resources and allows to model and plan preferences on these aspects in a very simple way (see [6], [5] and [7] for a comprehensive treatment about RASP).

In this paper, we illustrate the design of an integrated system able to plan activities of users, taking into account the context in which the user is located, the available resources, the geographical position and her/his preferences. This system, which is being implemented, is ASP- and RASP-based and can interact with the user through a mobile application. In Section 2 we briefly overview RASP, while in Section 3 the design of the system is described and in Section 4 we conclude.

2 RASP

Even resource production and consumption processes are connected with preferences: in fact, an agent may prefer to use some resources and not others in a given situation, or it may prefer to use available resources for obtaining a certain resource rather than others. o

Resourced Answer Set Programming (RASP) extends the ASP framework by introducing resources and preferences. With RASP we can easily specify available resources and the amount of resources needed to produce others: it supports reasoning about resource consumption and production, according to preferences. Resources are modelled by *amount-atoms* of the form $q\#a$ (q is the kind of resource and a its available quantity [1]). *Amount-atoms* are used in *r-facts* (RASP-facts) to represent the available quantities of resources. Thanks to *r-rules* (RASP-rules), a specific quantity of certain resources can be transformed into a specific quantity of another resource: *amount-atoms* in the body of a rule model the consumed resources, while in the head they model the resources produced by that rule. An *r-rule* can be fired several times thanks to the prefix $[N-M]$ (respectively the minimum and the maximum number of times a rule can be fired). In the following example (1), the rule can be fired from one to four times, producing from one to four portions of pasta according to available resources; to produce one portion of pasta with pesto the needed resources are 80 gr. of pasta and a little bottle of home made pesto, while to make that bottle of pesto we need some grams of many ingredients:

$$
\begin{aligned}
&[1-4]pasta\_with\_pesto\#1 \leftarrow pasta\#80, home\_made\_pesto\#1.\\
&home\_made\_pesto\#1 \leftarrow basil\#15, garlic\#1, pine\_nuts\#25,\\
&oil\#10, grated\_parmesan\#25. \qquad (1)\\
&basil\#300.\ pine\_nut\#250.\\
&garlic\#3.\ oil\#120.\ grated\_parmesan\#120.
\end{aligned}
$$

[1] For lack of space we do not consider management of quantities here, and in the example we just use integers.

If we want to clearly express which resource we prefer to use, we have to introduce *p-lists* (i.e. preference-lists), where the leftmost element of the list has high priority. For example, we can state that when we cook pasta with pesto we prefer to use home made pesto rather than pesto from supermarket:

$$[1-4]pasta\_with\_pesto\#1 \leftarrow \\ pasta\#80, (home\_made\_pesto\#1 > supermarket\_pesto\#1). \tag{2}$$

RASP provides also two kinds of conditional preference lists (*cp-list*): $pref\_when$ and *only_when* lists. Suppose that one prefers normal pasta instead of gluten free pasta, but not in case of *allergy*: the *only_when* condition is false and the *cp-list* is ignored. And suppose that when one has guests, one prefers to use home made pesto instead of supermarket pesto: if the condition *has_guest* does not hold, the *cp-list* becomes simply a disjunction.

$$[1-4]pasta\_with\_pesto\#1 \leftarrow \\ (pasta\#80 > gluten\_free\_pasta\#80\ only\_when\ notallergy), \\ (home\_made\_pesto\#1 > supermarket\_pesto\#1\ pref\_when\ has\_guest). \tag{3}$$

3 Framework Design

The design of an ASP-based integrated system for preference and resource management and planning is proposed here as a concrete application in real-world contexts. The main purpose of this system is to simplify our daily life, and for this reason the ASP-based system is integrated with web services to reach users everywhere, in every moment and situation via inexpensive mobile devices, typically smartphones. It is important to notice that the system is able to handle even conditional preferences and priorities among preferences. Furthermore, starting from the user's geographic location, the system optimizes the preferred answer sets according to her/his objectives, whether they are declared in an explicit or implicit way.

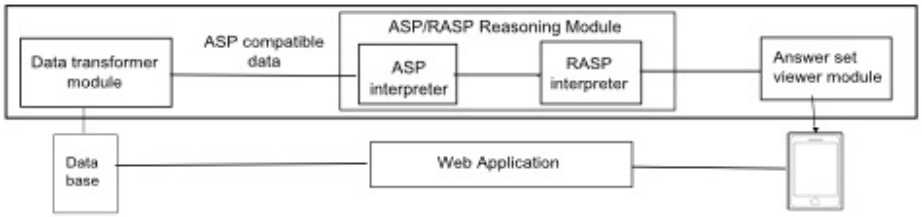

Fig. 1. The framework

Resources managed by the system are al least time and money, but the end-user can define others, through a web application interface, so to ensure a simple customization and personalization of the system. The framework is composed of different parts (Fig. 1): a database, a web application, an answer set viewer and the core of the system. In the database, the data needed by the system to make inferences are stored. It is a NoSQL database (http://nosql-database.org/), useful when we have to deal with a huge quantity of data that do not require a relational model. NoSQL is the *de facto* standard for mobile applications, and has bee chosen in view of efficiency and scalability, since the reasoning part is performed by the ASP module and thus no complex queries are needed. Data extracted from the database are processed by the data-transformer module, that transforms query results into an ASP-compatible format. At first, a pre-processing is needed: the system extracts the activities to plan and the preferences from the database, and then constructs suitable RASP rules and ASP facts and rules (needed to describe available resources and constraints). At this point, the files built in the previous step are processed by the ASP/RASP reasoning module, that consists of two parts: the ASP interpreter processes the data and grounds the program; the RASP interpreter processes the ground program and produces the preferred answer sets. Finally, these answer sets are transformed into a format understandable by a non academic user: the interaction between end-users and the system is made through the web application, via a smartphone. The end interface also performs user monitoring and profiling, feeding the system with new data, so as to elicit user needs, habits and preferences.

In the implementation which is being developed, the database and ASP/RASP parts are on a server, while the interface is on the mobile. In perspective (i.e., when a suitable deployment will be available) for the sake of scalability the ASP/RASP part can be moved on the mobile.

Let us provide an example of use. Assume that John wants to keep fit and hates gyms, but loves being outdoors and is very busy because of his new job: the system can produce a training program tailored for him in real time, that changes from time to time according to his preferences, resources, objectives and the place he actually is (the current user geographic and contextual location). The database has a catalogue of gym exercises, John's preferences (he hates gym, loves being outdoors, etc.), his available resources (the amount of his free time today, exercise equipment he has at home, parks nearby, and so on), resources he wants to be produced (e.g. loss of weight). If it is a sunny day, but John is very busy and has a lot of work to do: the system plans for him a run in the park near the office during his lunch break and exercises suitable for outdoor. However, if it is a rainy day and John had a rough day, the system plans for him training less hard exercises he can do while watching TV.

4 Concluding Remarks

The main purpose of the proposed framework is not to advance the state of the art of ASP and RASP approaches, but to realize an innovative application of logic

programming, by means of an effective integration with modern technologies affordable and understandable by everyone. The architecture includes in fact a suitable user interface, which is being designed so as to be understandable also by elder or impaired users, for which such a system can be particularly useful.

The novelty is the design of the framework itself; it aims at moving ASP and RASP outside the academic world, into the real world, and make them usable in concrete situations by ordinary people. In fact, we gave two small examples that can represent situations of our daily lives: in the first (see Section 1) the system can be a cooking teacher who offers us recipes suitable to our needs, while in the second (see Section 2) a personal trainer that offers custom workouts.

As mentioned, an implementation is under way, where the system will be able to automatically learn user preferences and objectives through machine learning mechanisms typical of proactive and adaptive agents.

References

1. Anger, C., Schaub, T., Truszczynski, M.: Aspargus-the Dagstuhl initiative. (2004)
2. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press. (2003)
3. Bienvenu, M., Lang, J., Wilson, N., et al.: From preference logics to preference languages, and back. Proc. KR 2010.(2010)
4. Brewka, G.: Complex preferences for answer set optimization. In: Principles of Knowledge Representation and Reasoning: Proc. of the Ninth International Conference (KR2004). 213–223. (2004)
5. Costantini, S., Formisano, A., Petturiti, D.: Extending and implementing RASP. Fundamenta Informaticae, 105(1), 1–33. (2010)
6. Costantini, S., Formisano, A.: Modeling preferences and conditional preferences on resource consumption and production in ASP. Journal of Algorithms, 64(1), 3–15. (2009)
7. Costantini, S., Formisano, A.: Answer set programming with resources. Journal of Logic and Computation, 20(2), 53–571. (2010)
8. Cui, B., Swift, T.: Preference logic grammars: fixed point semantics and application to data standardization. Artif. Int., 138(1), 117–147. (2002)
9. Domshlak, C., Hllermeier, E., Kaci, S., Prade, H.: Preferences in AI: An overview. Artif. Intell. 175(7-8), 1037–1052. (2011)
10. Gelfond, M.: Answer sets. Foundations of Artificial Intelligence, 3, 285–316. (2008)
11. Govindarajan, K., Jayaraman, B., Mantha, S.: Preference queries in deductive databases. New Generation Computing, 19 (1), 57–86. (2001)
12. Guo, H.F., Jayaraman, B.: Mode-directed preferences for logic programs. In: Proc. of the 2005 ACM symposium on Applied computing. ACM Press, 1414–1418. (2005)
13. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In: Logic Programming and Nonmonotonic Reasoning. Springer, 1. (2007)
14. Van Nieuwenborgh, D., Vermeir, D.: Preferred answer sets for ordered logic programs. Theory and Practice of Logic Programming, 6(2), 107–167. (2006)