ERASP – a system for enhancing recommendations using answer-set programming

Manoela Ilic, João Leite* and Martin Slota

CENTRIA and Departamento de Informática,
Faculdade de Ciências e Tecnologia,
Universidade Nova de Lisboa,
Quinta da Torre, 2829-516 Caparica, Portugal
E-mail: milic@uos.de
E-mail: jleite@di.fct.unl.pt
E-mail: martin.slota@gmail.com
*Corresponding author

Abstract: User modelling and personalisation are the key aspects of recommender systems in terms of recommendation quality. While being very efficient and designed to work with huge amounts of data, present recommender systems often lack the facility of user integration when it comes to feedback and direct user modelling. We propose the introduction of dynamic logic programming – an extension of answer set programming – in recommender systems, as a means for users to specify and update their models with the purpose of enhancing recommendations.

Keywords: dynamic logic programming; DLP; answer-set programming; ASP; user modelling; recommender systems; personalisation; reasoning-based intelligent systems.

Reference to this paper should be made as follows: Ilic, M., Leite, J. and Slota, M. (2009) 'ERASP – a system for enhancing recommendations using answer-set programming', *Int. J. Reasoning-based Intelligent Systems*, Vol. 1, Nos. 3/4, pp.147–163.

Biographical notes: Manoela Ilic is a Software Developer at Bendit GmbH Innovative Interfaces, Germany. She holds a BSc in Cognitive Science from the University of Osnabruck and MSc in Computational Logic from Universidade Nova de Lisboa in 2008. Currently she is designing and developing user interfaces for mobile devices.

João Leite is an Assistant Professor in Departamento de Informática at Universidade Nova de Lisboa and Researcher at CENTRIA – Centro de Inteligência Artificial. He completed his PhD from Universidade Nova de Lisboa in 2002. Since 1997, he has been researching in knowledge representation and reasoning and its application in multi-agent systems and recommender systems.

Martin Slota is a PhD student in Departamento de Informática at Universidade Nova de Lisboa and a Researcher at CENTRIA – Centro de Inteligência Artificial. He completed his MSc from Univerzita Komenského in Bratislava in 2007. His area of research is knowledge representation and reasoning.

1 Introduction

Recommender systems have become an essential tool for finding the needle in the haystack of the World Wide Web (WWW) – the information or item one is searching for (Resnick and Varian, 1997). Finding the desired item is a daunting task considering the amount of information that is present in the WWW and its databases, and almost every e-commerce application will provide the user with the help of a recommender system to suggest products or information that the user might want or need (Schafer et al., 2001).

Recommender systems are employed to recommend products in online stores, news articles in news subscription sites or financial services. Common techniques for selecting the right item for recommendation are: collaborative filtering (e.g., Resnick et al., 1994; Goldberg et al., 1992) where user ratings for objects are used to perform an inter-user comparison and then propose the best rated items; content-based recommendation (e.g., Balabanović and Shoham, 1997; Pazzani and Billsus, 2007) where descriptions of the content of items is matched against user profiles, employing techniques from the information retrieval field; knowledge-based recommendation (e.g., Burke, 2000; Felfernig and Kiener, 2005) where knowledge about the user, the objects and some distance measures between them, i.e., relationships between users and objects are used to infer the right selections; and, as always, hybrid

versions of these (e.g., Burke, 1999; Smyth and Cotter, 2001) where two or more of these techniques (collaborative filtering being usually one of them) are used to overcome their individual limitations. For further details on this subject the reader is referred to Burke (2002).

Recommender systems can also be categorised according to the way they interact with the user (Kelly and Bridge, 2006) as being either *single shot systems* where for each request the system makes its interpretation of information and proposes recommendations to the user without taking into account any previous interaction or *conversational systems* where recommendations are made on the basis of the current request of the user and some feedback provided as a response to previously proposed items.

The extent to which users find the recommendations satisfactory is, ultimately, the key feature of a recommender system and the accuracy of the user models that are employed is of significant importance to this goal. Such user models represent the user's taste and can be implicit (e.g., constructed from information about the user behaviour) or explicit (e.g., constructed from direct feedback or input by the user, like ratings). The accuracy of a user model greatly depends on how well short-term and long-term interests are represented (Billsus and Pazzani, 2000), making it a challenging task to include both sensibility to changes of taste and maintenance of permanent preferences. While implicit user modelling disburdens the user of providing direct feedback, explicit user modelling may be more confidence inspiring to the user since recommendations are based on a conscious assignment of preferences.

Though most of the recommender systems are very efficient from a large-scale perspective, the effort in user involvement and interaction is calling for more attention. Moreover, problems concerning trust and security in recommender systems could be approached with a better integration of the user and more control over the user model. For more details on the subject about security and manipulation in recommender systems the reader is referred to Lam et al. (2006).

In this paper, we will concentrate on explicit user modelling for recommender systems, guided by the following three claims:

1 Recommender systems should provide users with a way (language) to specify their models and preferences. This language should be expressive enough to allow for specifications that exceed the mere assignment of ratings to products. It should allow for the usage of existing concepts (e.g., product characteristics) as well as for the definition of new concepts (e.g., own qualitative classifications based on product characteristics). The language should allow for the specification of rules that use these concepts to define the policies regarding the recommendations made by the system. The language should also include some form of negation to allow for the specification of both positive as well as negative information.

2 Users should be able to update their models by specifying new rules. The system must take into account that users are not consistent along time, i.e., some newer rules may directly contradict previously specified ones, possibly representing an evolution of the user's tastes and needs, and these 'contradictions' should be dealt with by the system, relieving the user from any consistency requirements, always difficult to impose and often a discouraging factor.

3 Recommender systems should not depend solely on the model specified by the user. Other approaches and existing systems that do not require the user specification should be used as complement. Their outputs should be taken into consideration since they may already encode large amounts of data that should not be disregarded, and would be particularly useful in the absence of user specified knowledge. At the same time, the output of the recommender system should take into strict consideration the user specifications which, if violated, would turn the user away from the recommender system.

In this paper, we borrow concepts from the area of knowledge representation and non-monotonic reasoning to address these three issues. Specifically, we will employ the paradigm of *Dynamic Logic Programming* (Alferes et al., 2000, 2005; Leite, 2003) to set forth a proposal that aims at extending existing recommender systems with the possibility of allowing users to specify their individual models and preferences, while taking into account the three claims above.

Dynamic logic programming (DLP) is an extension of answer-set programming (ASP) (Gelfond and Lifschitz, 1990) introduced to deal with knowledge updates. ASP is a form of declarative programming that is similar in syntax to traditional logic programming and close in semantics to non-monotonic logic that is particularly suited for knowledge representation[1]. In contrast to Prolog, where proofs and substitutions are at its heart, the fundamental idea underlying ASP, due to its model-theoretic semantics, is to describe a problem declaratively in such a way that models of the description provide solutions to problems. Enormous progress concerning both the theoretical foundations of the approach and implementation issues has been made in recent years. The existence of very efficient ASP solvers [e.g., DLV[2] (Leone et al., 2006) and SMODELS[3] (Niemelä and Simons, 1997)] make it finally possible to investigate some serious applications in the area of e.g., data integration (Eiter, 2005), data source selection (Eiter et al., 2002) and also the semantic web (Schindlauer, 2006).

An extension of ASP, where knowledge is specified in a single theory, is *DLP* where knowledge is given by a series of theories, encoded as generalised logic programs[4] (or answer-set programs), each representing distinct states of the world. Different states, sequentially ordered, can represent different time periods, thus allowing DLP to represent knowledge undergoing successive updates. As

individual theories may comprise mutually contradictory as well as overlapping information, *DLP* employs the mutual relationships among different states to determine the declarative semantics for the combined theory comprised of all individual theories at each state. Intuitively, one can add newer rules to the end of the sequence and *DLP* automatically ensures that these rules are in force and that the older rules are kept for as long as they are not in conflict with the newly added ones.

DLP can provide an expressive framework for users to specify rules encoding their model, preferences and their updates, while enjoying several properties, some discussed below, such as:

- a simple extendable language, thus allowing for the specification of simple relations as well as the introductions of new concepts
- a well defined semantics, thus allowing for the formal study of properties of the system
- the possibility to use default negation to encode non-deterministic choice, thus generating more than one set of recommendations, facilitating diversity each time the system is invoked
- the combination of both strong and default negation to reason with the closed and open world assumptions, thus allowing for reasoning with incomplete information
- easy connection with relational databases (ASP can also be seen as a query language, more expressive than SQL), thus allowing easy integration with existing systems
- support for explanations, thus improving the interaction with the user.

Since we are providing users with a way to express themselves by means of rules, we can also provide the same rule-based language to the owners of the recommender system, enabling them to specify some policies that may not be captured by the existing recommender system (e.g., preference for recommending certain products). Even though this was not at the heart of our initial goal, it is an extra feature provided by our proposal, with no added cost.

In a nutshell, we want to propose a system, with a precise formal specification and semantics, composed of three modules, namely the output of an existing recommender system, a set of rules specified by the owner of the recommender system and a sequence of sets of rules specified by the user, for which we provide an expressive language. The modules are combined in such a way that they produce a set of recommendations that obeys certain properties such as obedience to the rules specified by the user, removal of contradictions specified by the user along time and keeping the result of the initial recommendation module as much as possible in the final output, among others.

The proposed system can work as an add-on for existing recommender systems. Owners of such systems should be able to plug in the application as a recommendation enhancer, offering users the possibility of explicit preference creation, for defining specific system rules or both. A rule-based language like *DLP* can empower owners of recommender systems with the necessary tools to employ marketing strategies with precision, while keeping the diversity of recommendations and following user's preferences. Moreover, it can be used as a query tool to extract information from the database using a more sophisticated language than for example SQL, allowing the system to be used on its own without the need of previously existing recommender systems.

The remainder of this paper is organised as follows. In Section 2 we briefly recap the notion of DLP, establishing the language used throughout. In Section 3 we define our framework and its semantics while in Section 4 we present a short illustrative example. Section 5 is devoted to the discussion of some properties of the proposed system and their proofs are presented. In Section 6 we describe the implementation and present some performance results. Further in Section 7, we discuss possible optimisations of our system, in Section 8 we compare our work to related systems and we conclude in Section 9.

2 Dynamic logic programming

For self containment, in this section we briefly recap the notion and semantics of DLP needed throughout. More motivation regarding all these notions can be found in Alferes et al. (2000) and Leite (2003).

Let $\mathcal{A}$ be a set of propositional atoms. An *objective literal* is either an atom A or a strongly negated atom $\neg A$. A *default literal* is an objective literal preceded by *not*. A *literal* is either an objective literal or a default literal. A *rule* r is an ordered pair $H(r) \leftarrow B(r)$ where $H(r)$ (dubbed the head of the rule) is a literal and $B(r)$ (dubbed the body of the rule) is a finite set of literals. A rule with $H(r) = L_0$ and $B(r) = \{L_1, ..., L_n\}$ will simply be written as

$$L_0 \leftarrow L_1, ..., L_n.$$

A *tautology* is a rule of the form $L \leftarrow Body$ with $L \in Body$. A *generalised logic program* (*GLP*) P over $\mathcal{A}$ is a finite or infinite set of rules. A *normal logic program* is a GLP that does not contain a default literal in the head of any rule. If $H(r) = A$ (resp. $H(r) = not\ A$), then $not\ H(r) = not\ A$ (resp. $not\ H(r) = A$). Similarly, if $H(r) = A$ (resp. $H(r) = \neg A$), then $\neg H(r) = \neg A$ (resp. $\neg H(r) = A$). By the *expanded GLP* corresponding to the GLP P, denoted by **P**, we mean the GLP obtained by augmenting P with a rule of the form $not\ \neg H(r) \leftarrow B(r)$ for every rule, in P, of the form $H(r) \leftarrow B(r)$, where $H(r)$ is an objective literal.

An *interpretation* M of $\mathcal{A}$ is a set of objective literals that is consistent, i.e., M does not contain both A and $\neg A$.

an objective literal L is true in M, denoted by $M \models L$, iff $L \in M$, and false otherwise. A default literal $not\ L$ is true in M, denoted by $M \models not\ L$, iff $L \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \models B$, iff each literal in B is true in M. An interpretation M of $\mathcal{A}$ is an *answer set* of a GLP P iff $M' = least\left(\mathbf{P} \cup \left\{not\ A \middle| A \notin M\right\}\right)$, where $M' = M \cup \left\{not\_A \middle| A \notin M\right\}$, A is an objective literal, and *least*(.) denotes the least model of the definite program obtained from the argument program, replacing every default literal *not A* by a new atom *not_A*. Let *AS*(*P*) denotes the set of answer-sets of *P*.

A dynamic logic program (*DLP*) is a sequence of GLPs. Let $\mathcal{P} = (P_1, ..., P_s)$ and $\mathcal{P}' = (P_1', ..., P_s')$ be DLPs. We use $\rho(\mathcal{P})$ to denote the multiset of all rules appearing in the programs $\mathbf{P}_1, ..., \mathbf{P}_s$, and $\mathcal{P} \cup \mathcal{P}'$ to denote $(P_1 \cup P_1', ..., P_s \cup P_s')$ and $(P_i', P_j', \mathcal{P})$ to denote $(P_i', P_j', P_1, ..., P_s)$. We can now set forth the definition of semantics, *based on causal rejection of rules*, for DLPs. We start by defining the notion of conflicting rules as follows: two rules r and r' are *conflicting*, denoted by $r \bowtie r'$, iff $H(r) = not\ H(r')$, used to accomplish the desired rejection of rules:

Definition 1 (rejected rules): .Let $\mathcal{P} = (P_1, ..., P_s)$ be a DLP and *M* an interpretation. We define:

$$Rej(\mathcal{P}, M) = \left\{r \mid r \in \mathbf{P}_i, \left(\exists r' \in \mathbf{P}_j\right)\left(i \le j, r \bowtie r', M \models B(r')\right)\right\}$$

We also need the following notion of default assumptions.

Definition 2 (default assumptions): Let $\mathcal{P} = (P_1, ..., P_s)$ be a DLP and *M* an interpretation. We define (where *A* is an objective literal):

$$Def(\mathcal{P}, M) = \left\{not\ A \mid \left(\nexists r \in \rho\right)\left((\mathcal{P}), H(r) = A, M \models B(r)\right)\right\}$$

We are now ready to define the semantics for DLPs based on the intuition that some interpretation is a model iff it obeys an equation based on the least model of the multi-set of all the rules in the (expanded) DLP, without the rejected rules, together with the set of default assumptions. The semantics is dubbed *(refined) dynamic stable model* (*RDSM*) *semantics.*

Definition 3 (semantics of DLP): Let $\mathcal{P} = (P_1, ..., P_s)$ be a DLP and *M* an interpretation. *M* is an RDSM of $\mathcal{P}$ iff

$$M' = least\left(\left[\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M)\right] \cup Def(\mathcal{P}, M)\right)$$

where $M', \rho(.)$ and *least*(.) are as before. Let $RDSM(\mathcal{P})$ denote the set of all RDSMs of $\mathcal{P}$.

3 Framework and semantics

In this section, we introduce our framework and its semantics.

Our goal is to take the strengths of DLP as a knowledge representation framework with the capabilities of allowing for the representation of updates, and put it at the service of the user and the company, while at the same time ensuring some degree of integration with the output of other recommendation modules, possibly based on distinct paradigms (e.g., statistical, etc.).

To make the presentation of our ideas simpler, we will make some assumptions and simplifications that, in our opinion, do not compromise our proposal and can be subsequently lifted (not in this paper though).

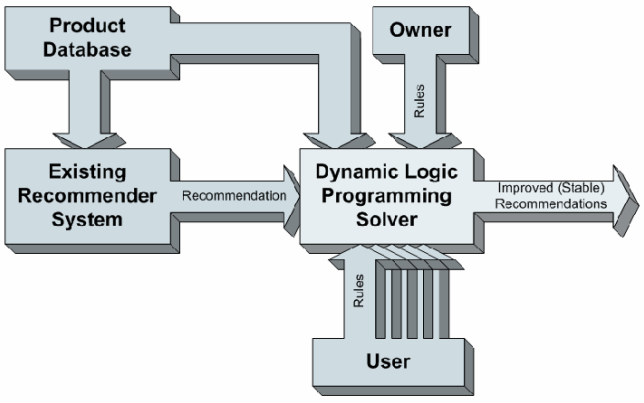
We start by assuming a layered system where the output of the existing recommendation module is simply used as the input to our system, and where no feedback to the initial module exists. We are aware that allowing for feedback from our system to the existing module could benefit its output, but such process would greatly depend on such existing module and we want to make things as general as possible, and concentrate on other aspects of the system. This takes us to the next assumption, that of the output of such existing module. We consider it to be an interpretation, i.e., a consistent set of objective literals representing the recommendations. For simplicity, we can assume that the language contains a reserved predicate *rec*/1 where the items are the argument terms of the predicate and the interpretation will contain atoms corresponding to the recommended items. For example, the interpretation

$$M = \{rec("Godfather"), rec("Underground"), rec("The\ Köln\ Concert"), rec("Paris, Texas"), rec("Takk")\}$$

encodes the recommendations for the films '*Godfather*', '*Underground*' and '*Paris, Texas*', as well as for the music albums '*The Köln Concert*' and '*Takk*'. It would be straightforward to extend this case to one where some value would be associated with each recommendation, e.g., using a predicate of the form *rec*(*item*, *value*). However, to get our point across, we will keep to the simplified version where the output of the existing module is simply a set of recommendations. An outline is given in Figure 1.

What we have then, is an initial interpretation representing the output of the initial module, which we dub the *initial model*, a GLP representing the owner's policy and a DLP representing the rules (and their evolution) specified by the user. The product database is, typically, a relational database[5] that can easily be represented by a set of facts in a logic program. For simplicity, without loss of generality, we assume such database to be a part of the GLP representing the owner's policy.

Figure 1 Outline of the system (see online version for colours)



To formalise this notion, we introduce the concept of dynamic recommender frame (DRF) defined as follows:

Definition 4 (dynamic recommender frame): Let $\mathcal{A}$ be a set of propositional atoms. A DRF, over $\mathcal{A}$, is a triple $\langle M, P_0, \mathcal{P} \rangle$ where M is an interpretation of $\mathcal{A}$, P_0 a GLP over $\mathcal{A}$, and $\mathcal{P}$ a DLP over $\mathcal{A}$.

The semantics of a DRF is given by the set of stable models of its transformation into a DLP. This transformation is based on a few underlying assumptions concerning the way these three modules should interact and be combined. In particular, we want the rules specified by the user to be the most relevant and be able to supersede both those issued by the owner and the recommendation issued by the existing module. This is a natural principle as users would not accept a recommender system that would explicitly violate their rules (e.g., recommend a horror movie when the user said that no horror movies should be recommended, just because the owner wants to push horror movies). This limits the impact of recommendations made by the initial module and the company to those not directly constrained by the user or to those for which the rules specified by the user allow for more than one alternative. The other principle concerns the relationship between the initial model and the policy specified by the owner. Here, we will opt for giving a prevailing role to the rules specified by the owner. The rational for this is also rather obvious and natural: the owner must be given the option/power to supersede the initial recommendation module (e.g., specify preference to specify products of a given brand over those of another because of higher profit margins), and it may be impossible to have such control directly inside the initial module (e.g., if it is a sub-symbolic system such as a neural network).

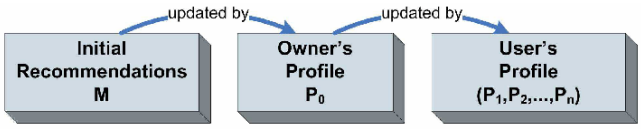
With these principles in mind, we first define a transformation from a DRF into a DLP:

Definition 5 (Υ)*:* Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then $\Upsilon(\mathcal{R})$ is the DLP $(P_M, P_0, \mathcal{P})$ where $P_M = \{A \leftarrow : A \in M\}$.

Intuitively, we construct a DLP where the initial knowledge is the program obtained from the initial model. Such initial program is followed (updated with) the owner's policy specification (P_0), which is then followed by the sequence of specifications provided by the user $(\mathcal{P})$. This is illustrated in Figure 2. We define the semantics of a DRF as follows:

Definition 6 (stable recommendation semantics): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF and $M_{\mathcal{R}}$ an interpretation. $M_{\mathcal{R}}$ is a stable recommendation iff $M_{\mathcal{R}}$ is a dynamic stable model of $\Upsilon(\mathcal{R})$. Let $SR(\mathcal{R})$ denote the set of all stable recommendations.

Figure 2 The $\Upsilon(\mathcal{R})$ operator (see online version for colours)



4 Illustrative example

In this section, we present an example with the purpose of illustrating some features of our proposal.

Let us consider an online movie recommender, with some existing recommender system based on statistical analysis performed over the years. We consider the output of such module to be a set of recommended items, that, for our purposes, we will consider constant throughout this example. Let the interpretation M represent such output, i.e., the initial model, and be:

$$M = \{rec(497), rec(527), rec(551), rec(589), rec(1249), rec(1267), rec(1580), rec(1608), rec(1912), rec(2396)\}$$

The properties of those recommended movies can be seen in Table 1.

Table 1 Initial recommendations

ID	*Title*	*Year*	*Genres*
497	Much Ado About nothing	1993	Comedy, romance
527	Schindler's List	1993	Drama, war
551	Nightmare Before Christmas, The	1993	Children's, comedy
589	Terminator 2: Judgment Day	1991	Action, sci-fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-noir, thriller
1580	Men in Black	1997	Action, adventure
1608	Air Force One	1997	Action, thriller
1912	Out of Sight	1998	Action, crime
2396	Shakespeare in Love	1998	Comedy, romance

The owner can, on top on this, define some further recommendation policies, such as:

- The system should, non-deterministically, recommend at least one of movies 12 and 15[6]:

$$rec(12) \leftarrow not\ rec(15). \quad (1)$$

$$rec(15) \leftarrow not\ rec(12). \quad (2)$$

- The system should always recommend the movies with the genre *Musical* from the year 1998 if there are initial recommendations of movies with the genre *Romance* from the same year:

$$\begin{aligned} rec(X) \leftarrow\ & genre(X,"Musical"), year(X,1998), \\ & rec(Y), year(Y,1998), \\ & genre(Y,"Romance"). \end{aligned} \quad (3)$$

These previous rules will be contained by the owner program P_0. Without any rules specified by the user, the frame $\langle M, P_0, (\ ) \rangle$ has two stable recommendations resulting from the effect of the owner's rules on top of the initial model, namely[7]:

$$\begin{aligned} M_{R1} = \{ & rec(12), rec(497), rec(527), rec(551), \\ & rec(589), rec(1249), rec(1267), rec(1580), \\ & rec(1608), rec(1856), rec(1912), rec(2394), \\ & rec(2396)\} \end{aligned}$$

$$\begin{aligned} M_{R2} = \{ & rec(15), rec(497), rec(527), rec(551), \\ & rec(589), rec(1249), rec(1267), rec(1580), \\ & rec(1608), rec(1856), rec(1912), rec(2394), \\ & rec(2396)\} \end{aligned}$$

The properties of the movies of these models can be seen in Tables 2 and 3[8].

Table 2 Movies of model M_{R1}

ID	*Title*	*Year*	*Genres*
12	Dracula: Dead and Loving	1995	Comedy, horror
497	Much Ado About Nothing	1993	Comedy, romance
527	Schindler's List	1993	Drama, war
551	Nightmare Before Christmas, The	1993	Children's, comedy
589	Terminator 2: Judgment Day	1991	Action, sci-fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-noir, thriller
1580	Men in Black	1997	Action, adventure
1608	Air Force One	1997	Action, thriller
1856	Kurt & Courtney	1998	Documentary, musical
1912	Out of Sight	1998	Action, crime
2394	Prince of Egypt, The	1998	Animation, musical
2396	Shakespeare in Love	1998	Comedy, romance

Table 3 Movies of model M_{R2}

ID	*Title*	*Year*	*Genres*
15	Cutthroat Island	1995	Action, adventure
497	Much Ado About Nothing	1993	Comedy, romance
527	Schindler's List	1993	Drama, war
551	Nightmare Before Christmas, The	1993	Children's, comedy
589	Terminator 2: Judgment Day	1991	Action, sci-fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-noir, thriller
1580	Men in Black	1997	Action, adventure
1608	Air Force One	1997	Action, thriller
1856	Kurt & Courtney	1998	Documentary, musical
1912	Out of Sight	1998	Action, crime
2394	Prince of Egypt, The	1998	Animation, musical
2396	Shakespeare in Love	1998	Comedy, romance

When taking a closer look at the two stable recommendations, we can observe that the first rule specified by the owner introduced either movie 12 or movie 15. The second rule introduced the movie 2394 since there was an initial recommendation for a *Romance* from year 1998. The recommender system would, e.g., randomly, choose one of these stable recommendations to present to the user and thus adding diversity to the recommender system which, otherwise, would only have one set of recommendations to present in case of consecutive equal requests.

Let us now consider the user. Initially, the user specifies that he does not want any recommendations of *Animation* movies:

$$not\ rec(X) \leftarrow genre(X,"Animation"). \quad (4)$$

This rule alone will override all the initial recommendations for *Animations* and also all the rules of the owner specifying those recommendations such as rule (3). In this case, the only *Animation* rejected will be the one introduced by the owner policy. If $\mathcal{P}_1 = (P_1)$, where P_1 contains rule (4), the two stable recommendations of the frame $\langle M, P_0, \mathcal{P}_1 \rangle$ are:

$$\begin{aligned} M_{R3} = \{ & rec(12), rec(497), rec(527), rec(551), \\ & rec(589), rec(1249), rec(1267), rec(1580), \\ & rec(1608), rec(1856), rec(1912), rec(2396)\} \end{aligned}$$

$$\begin{aligned} M_{R4} = \{ & rec(15), rec(497), rec(527), rec(551), \\ & rec(589), rec(1249), rec(1267), rec(1580), \\ & rec(1608), rec(1856), rec(1912), rec(2396)\} \end{aligned}$$

As one can see, the recommendation for the movie 2394, *The Prince of Egypt*, is not anymore present in the two

models. Later on, the user decides to obtain some recommendations for some specific movies. For this, he decides to define the concept of what good movies are. Initially, he considers a good movie to be one with the genres *Action*, *Adventure* and *Fantasy*. So he writes the following rule:

$$\begin{aligned} good(X) \leftarrow\ & genre(X,"Action"),\\ & genre(X,"Adventure"),\\ & genre(X,"Fantasy"). \end{aligned} \tag{5}$$

Furthermore, the user wants to obtain at least one recommendation for a good movie[9]:

$$\begin{aligned} rec(X) &\leftarrow good(X), not\ n\_rec(X).\\ n\_rec(X) &\leftarrow good(X), not\ rec(X).\\ rec\_at\_least\_one &\leftarrow good(X), rec(X).\\ &\leftarrow not\ rec\_at\_least\_one \end{aligned} \tag{6}$$

If $\mathcal{P}_2 = (P_1, P_2)$, where P_2 contains the rules (5) and (6), the stable recommendations of the frame $\langle M, P_0, \mathcal{P}_2 \rangle$ are:

$$\begin{aligned} M_{R5} = \{ & rec(12), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396)\} \end{aligned}$$

$$\begin{aligned} M_{R6} = \{ & rec(15), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396)\} \end{aligned}$$

Since there is only one movie with those three genres in the database, we still have two models, both of them with a newly introduced recommendation for the movie 558, *The Pagemaster*. The user comes back at a later stage and refines the notion of a good movie:

$$P_3 : good(X) \leftarrow genre(X,"War"), year(X, 2000). \tag{7}$$

The rule on getting at least one recommendation of a good movie is still in force. So, with $\mathcal{P}_3 = (P_1, P_2, P_3)$, the stable recommendations of the frame $\langle M, P_0, \mathcal{P}_3 \rangle$ are:

$$\begin{aligned} M_{R7} = \{ & rec(12), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396)\} \end{aligned}$$

$$\begin{aligned} M_{R8} = \{ & rec(15), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396)\} \end{aligned}$$

$$\begin{aligned} M_{R9} = \{ & rec(12), rec(497), rec(527), rec(551),\\ & rec(589), rec(1249), rec(1267), rec(1580),\\ & rec(1608), rec(1856), rec(1912), rec(2396),\\ & rec(3746)\} \end{aligned}$$

$$\begin{aligned} M_{R10} = \{ & rec(15), rec(497), rec(527), rec(551),\\ & rec(589), rec(1249), rec(1267), rec(1580),\\ & rec(1608), rec(1856), rec(1912), rec(2396),\\ & rec(3746)\} \end{aligned}$$

$$\begin{aligned} M_{R11} = \{ & rec(12), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396), rec(3746)\} \end{aligned}$$

$$\begin{aligned} M_{R12} = \{ & rec(15), rec(497), rec(527), rec(551),\\ & rec(558), rec(589), rec(1249), rec(1267),\\ & rec(1580), rec(1608), rec(1856), rec(1912),\\ & rec(2396), rec(3746)\} \end{aligned}$$

Table 4 Movies of model M_{R11}

ID	*Title*	*Year*	*Genres*
12	Dracula: Dead and Loving It	1995	Comedy, horror
497	Much Ado About Nothing	1993	Comedy, romance
527	Schindler's List	1993	Drama, war
551	Nightmare Before Christmas, The	1993	Children's, comedy
558	Pagemaster, The	1994	Action, adventure, fantasy
589	Terminator 2: Judgment Day	1991	Action, sci-fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-noir, thriller
1580	Men in Black	1997	Action, adventure
1608	Air Force One	1997	Action, thriller
1856	Kurt & Courtney	1998	Documentary, musical
1912	Out of Sight	1998	Action, crime
2396	Shakespeare in Love	1998	Comedy, romance
3746	Butterfly	2000	Drama, war

The movie properties of the model M_{R11} can be seen in Table 4. One can see that in all models there is at least one good movie recommendation. There is again just one movie in the database that fulfills the rule of having genre *war* and being from year 2000 [rule (7)]. The old rule (5) defining the concept of a good movie is still in effect because it is not in conflict with the new one. This results in six models, each one containing at least one of the two good movies and either movie 12 or 15.

In the next update, the user decides to write a rule stating that he does not want any more recommendations for *Adventure* movies:

$$P_4 : not\ rec(X) \leftarrow genre(X, "Adventure"). \quad (8)$$

Let $\mathcal{P}_4 = (P_1, P_2, P_3, P_4)$. Then the only stable recommendation of the frame $\langle M, P_0, \mathcal{P}_4 \rangle$ is:

$$M_{R13} = \{rec(12), rec(497), rec(527), rec(551), rec(589), rec(1249), rec(1267), rec(1608), rec(1856), rec(1912), rec(2396), rec(3746)\}$$

The movie 15 cannot be recommended anymore since it is of genre *Adventure*. The same holds for the movie introduced by rule (5): the new rule (8) causes a direct contradiction with that rule for a good movie. The semantics based on the causal rejection of rules deals with this issue. The movies of that model can be seen in Table 5.

Table 5 Movies of model M_{R13}

ID	*Title*	*Year*	*Genres*
12	Dracula: Dead and Loving It	1995	Comedy, horror
497	Much Ado about Nothing	1993	Comedy, romance
527	Schindler's List	1993	Drama, war
551	Nightmare Before Christmas, The	1993	Children's, comedy
589	Terminator 2: Judgment Day	1991	Action, sci-fi
1249	Nikita	1990	Thriller
1267	Manchurian Candidate, The	1962	Film-noir, thriller
1608	Air Force One	1997	Action, thriller
1856	Kurt & Courtney	1998	Documentary, musical
1912	Out of Sight	1998	Action, crime
2396	Shakespeare in Love	1998	Comedy, romance
3746	Butterfly	2000	Drama, war

Worth noting is that for each recommendation there is an explanation based on user rules, or on owner rules if not overridden by those of the user or, if there is nothing overriding it, on an initial recommendation. This and other properties are explored in the next section.

5 Properties

One of the good features of using a formal system is that it makes it possible to state and prove some of its properties, allowing for a better understanding of its behaviour. In this section we discuss some properties of the stable recommendation semantics and provide their proofs.

We start with conservation, a property that states that if the recommendation of the initial module is a dynamic stable model of the DLP consisting of the owner rules followed by the user DLP, then the initial recommendation is a stable recommendation. This ensures that the semantics will keep the results of the initial module if they agree with the specification of the owner and user. Formally:

Proposition 1 (conservation): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then

$$M \in RDSM((P_0, \mathcal{P})) \Rightarrow SR(\mathcal{R}) \supseteq \{M\}$$

Proof: Let $\mathcal{P}_1 = (P_0, \mathcal{P})$, M be an interpretation such that $M \in RDSM(\mathcal{P}_1), P_M = \{A \leftarrow \mid A \in M\}$ and $\mathcal{P}_2 = \Upsilon(\mathcal{R}) = (P_M, P_0, \mathcal{P})$. Following definition 6, $SR(\mathcal{R}) \supseteq \{M\}$ holds iff M is a stable recommendation which in turn holds iff M is a dynamic stable model of $\mathcal{P}_2$. Hence we need to show that $M \in RDSM(\mathcal{P}_2)$, or, more formally:

$$M' = least([\rho(\mathcal{P}_2) \setminus Rej(\mathcal{P}_2, M)] \cup Def(\mathcal{P}_2, M)) \quad (9)$$

Since M belongs to $RDSM(\mathcal{P}_1)$, we have

$$M' = least([\rho(\mathcal{P}_1) \setminus Rej(\mathcal{P}_1, M)] \cup Def(\mathcal{P}_1, M)) \quad (10)$$

As can be seen from definition 2, adding more rules to a DLP can only reduce the set of default assumptions. If $not\ A \in Def(\mathcal{P}_1, M)$ for some $A \in M$, then the right hand side of (10) would contain $not\ A$ while the left hand side wouldn't. So for any $A \in M$ we have $not\ A \notin Def(\mathcal{P}_1, M)$. Hence adding the rules in P_M to $\mathcal{P}_1$, obtaining $\mathcal{P}_2$, will not eliminate any default assumptions. More formally,

$$Def(\mathcal{P}_1, M) = Def(\mathcal{P}_2, M) \quad (11)$$

Apart from that, we know that the set $\rho(\mathcal{P}_2) \setminus Rej(\mathcal{P}_2, M)$ is a superset of $\rho(\mathcal{P}_1) \setminus Rej(\mathcal{P}_1, M)$ because $\rho(\mathcal{P}_2) \supseteq \rho(\mathcal{P}_1)$ and the rules in P_M cannot cause rejection of other rules. Moreover, the extra rules in the former set all belong to P_M. So

$$(\rho(\mathcal{P}_2) \setminus Rej(\mathcal{P}_2, M)) \setminus P_M = \rho(\mathcal{P}_1) \setminus Rej(\mathcal{P}_1, M) \quad (12)$$

From (12), (11) and (10) we can conclude that

$$least\left(\left[\left(\rho(\mathcal{P}_2)\setminus Rej(\mathcal{P}_2,M)\right)\setminus P_M\right]\cup Def(\mathcal{P}_2,M)\right)$$
$$= least\left(\left[\rho(\mathcal{P}_1)\setminus Rej(\mathcal{P}_1,M)\right]\cup Def(\mathcal{P}_1,M)\right)$$
$$= M'$$

Furthermore, adding the rules from P_M to the set $\left[\left(\rho(\mathcal{P}_2)\setminus Rej(\mathcal{P}_2,M)\right)\setminus P_M\right]\cup Def(\mathcal{P}_2,M)$ will not change the result of the *least*(.) operator because each atom $A\in M$ is inferred anyway. Hence (9) also holds.

The property of conservation is very valuable for owners of recommender systems since it guarantees the presence of each initial recommendation that does not conflict with any of the specifications.

It is also important to establish the relationship between the stable recommendation semantics and DLP for the case where there is no initial recommendation module. This ensures the proper transfer of all properties of DLP and ASP, namely expressiveness results, to our system. It also ensures that the system can be used without having any initial recommendations and even when no owner rules are present.

We will need the following Lemma:

Lemma 2: Let $P_1,P_2,...,P_n$ be GLPs, $0\le i\le n$, $\mathcal{P}=(P_1,P_2,...,P_n)$ and $\mathcal{Q}=(P_1,P_2,...,P_i,\phi,P_{i+1},...,P_n)$. Then

$$RDSM(\mathcal{P})=RDSM(\mathcal{Q})$$

Proof: M is a dynamic stable model of $\mathcal{P}$ iff

$$M'=least\left(\left[\rho(\mathcal{P})\setminus Rej(\mathcal{P},M)\right]\cup Def(\mathcal{P},M)\right) \qquad (13)$$

We can immediately see that

$$\rho(\mathcal{Q})=P_1\cup P_2\cup\ldots\cup P_i\cup\phi\cup P_{i+1}\cup\ldots\cup P_n$$
$$=P_1\cup P_2\cup\ldots\cup P_n=\rho(\mathcal{P})$$

Furthermore, since there are no rules in ϕ, it does not affect the set of rejected rules $Rej(\mathcal{P},M)$ and we obtain $Rej(\mathcal{Q},M)=Rej(\mathcal{P},M)$. Similarly, the empty program does not affect the set of default assumptions, so $Def(\mathcal{Q},M)=Def(\mathcal{P},M)$. Hence

$$least\left(\left[\rho(\mathcal{Q})\setminus Rej(\mathcal{Q},M)\right]\cup Def(\mathcal{Q},M)\right)$$
$$=least\left(\left[\rho(\mathcal{P})\setminus Rej(\mathcal{P},M)\right]\cup Def(\mathcal{P},M)\right)$$

Together with (13) this implies that *M* is a dynamic stable model of $\mathcal{P}$ iff it is a dynamic stable model of $\mathcal{Q}$. Hence $RDSM(\mathcal{P})=RDSM(\mathcal{Q})$.

Proposition 3 (generalisation of DLP): Let P_0 be a GLP and $\mathcal{P}$ a DLP. Then

$$SR(\langle\phi,\phi,\mathcal{P}\rangle)=RDSM(\mathcal{P})$$
$$SR(\langle\phi,P_0,\mathcal{P}\rangle)=RDSM((P_0,\mathcal{P}))$$

Proof: According to definition 6 we have that

$$SR(\langle\phi,\phi,\mathcal{P}\rangle)=RDSM((\phi,\phi,\mathcal{P}))$$
$$SR(\langle\phi,P_0,\mathcal{P}\rangle)=RDSM((\phi,P_0,\mathcal{P}))$$

By applying Lemma 2 (with $i=0$) three times we obtain:

$$RDSM((\phi,\phi,\mathcal{P}))=RDSM((\phi,\mathcal{P})=RDSM(\mathcal{P}))$$
$$RDSM((\phi,P_0,\mathcal{P}))=RDSM((P_0,\mathcal{P}))$$

Hence the proposition holds.

Not only does this property show the possible usage of the system as a query tool, but one can also think of the system as secured against a possible failure of the initial recommendation mechanism. The owner could specify rules that, in any case, would output a set of recommendations.

A very important issue in recommender systems is that of providing the user (and the owner) with explanations regarding the recommendations made. The fact that the stable recommendation semantics is well defined already provides a formal basis to support its results. However, we can state stronger results concerning the justification for the existence and absence of a particular recommendation. If a recommendation belongs to a stable recommendation, then either there is a user rule supporting it or there is an owner rule supporting it and no user rule overriding it or it is in the output of the initial module and no rules are overriding it. In other words, there is always an explanation for recommendations.

Proposition 4 (positive supportiveness): Let $\mathcal{R}=\langle M,P_0,\mathcal{P}\rangle$ be a DRF and $A\in M_{\mathcal{R}}\in SR(\mathcal{R})$. Then one of these three cases must occur:

a $\exists r\in\rho(\mathcal{P}):H(r)=A,M_{\mathcal{R}}\vDash B(r)$

b $\exists r\in P_0:H(r)=A,M_{\mathcal{R}}\vDash B(r)\ and$
$\nexists r\in\rho(\mathcal{P}):H(r)=not\ A,M_{\mathcal{R}}\vDash B(r)$

c $A\in M\ and$
$\nexists r\in\rho((P_0,\mathcal{P})):H(r)=not\ A,M_{\mathcal{R}}\vDash B(r)$

Proof: Let $\mathcal{Q}=(P_M,P_0,\mathcal{P})$. Since we know that $A\in M_{\mathcal{R}}\in SR(\mathcal{R})$, we also know that $A\in M_{\mathcal{R}}\in RDSM(\mathcal{Q})$. Hence there must be a rule *r* in the set

$$\left[\rho(\mathcal{Q})\setminus Rej(\mathcal{Q},M_{\mathcal{R}})\right]\cup Def(\mathcal{Q},M_{\mathcal{R}})$$

such that $H(r)=A$ and $M_{\mathcal{R}}\vDash B(r)$. Since the set $Def(\mathcal{Q},M_{\mathcal{R}})$ does not contain any rules with positive literals in their heads, we have

$$r\in\rho(\mathcal{Q})\setminus Rej(\mathcal{Q},M_{\mathcal{R}}) \qquad (14)$$

Let's assume (b) and (c) do not hold. We consider three cases:

1 If $r \in \rho(\mathcal{P})$, then (a) holds.

2 If $r \notin \rho(\mathcal{P})$ and $r \in P_0$, then, since (b) does not hold, we can conclude that there exists a rule $r' \in \rho(\mathcal{P})$ such that $H(r') = not\ A$ and $M_{\mathcal{R}} \models B(r')$. So r' rejects r and hence $r \notin \rho(\mathcal{Q}) \setminus Rej(\mathcal{Q}, M_{\mathcal{R}})$, which is in conflict with (14).

3 If $r \notin \rho(\mathcal{P})$ and $r \notin P_0$ and $r \in P_M$, then r is a fact $(A \leftarrow)$ and $A \in M$. Since (c) does not hold, we have that there exists a rule $r' \in \rho((P_0, \mathcal{P}))$ such that $H(r') = not\ A$ and $M_{\mathcal{R}} \models B(r')$. This means that r' rejects r and hence $r \notin \rho(\mathcal{Q}) \setminus Rej(\mathcal{Q}, M_{\mathcal{R}})$. This is in conflict with (14).

So, for each item in the set of final recommendations presented to the user, we exactly know why it is there. Likewise, for the absence of recommendations, if a recommendation belongs to the output of the initial system and is not part of a stable recommendation, then there must be an owner or user rule overriding it.

Proposition 5 (negative supportiveness): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF, $M_{\mathcal{R}} \in SR(\mathcal{R})$ be a stable recommendation and A be an atom such that $A \in M$ and $A \notin M_{\mathcal{R}}$ Then there exists some rule $r \in \rho((P_0, \mathcal{P}))$ such that $H(r) = not\ A$ and $M_{\mathcal{R}} \models B(r)$.

Proof: From $A \notin M_{\mathcal{R}}$ it follows that every rule $r' \in \rho((P_M, P_0, \mathcal{P}))$ such that $H(r') = A$ and $M_{\mathcal{R}} \models B(r')$ must be rejected by some other rule in $\rho((P_0, \mathcal{P}))$. Further, from $A \in M$ it follows that there is a rule r_A such that $H(r_A) = A, B(r_A) = \phi$ and $r_A \in P_M \subseteq \rho((P_M, P_0, \mathcal{P}))$. Hence the rule r_A must be rejected by some other rule from $\rho((P_0, \mathcal{P}))$. More formally, there exists a rule $r \in \rho((P_0, \mathcal{P}))$ such that $H(r) = not\ A$ and $M_{\mathcal{R}} \models B(r)$.

As with all logic programming based semantics, it is important to ensure that tautological (irrelevant) rules do not cause any effect.

Proposition 6 (immunity to tautologies): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF, $E_0, \ldots, E_s$ sets of tautologies, $\mathcal{E} = (E_1, \ldots, E_s)$ and $\mathcal{R}' = \langle M, P_0 \cup E_0, \mathcal{P} \cup \mathcal{E} \rangle$. Then

$$SR(\mathcal{R}) = SR(\mathcal{R}')$$

Proof: Proposition 3.5 from Alferes et al. (2005): Let *Sem* be a semantics for generalised programs, *P* a GLP and τ a tautology. If *Sem* complies with the refined extension principle then $Sem(P \cup \tau) \subseteq Sem(P)$.

Proposition 3.6. from Alferes et al. (2005): Let *P* be any given GLP, $P \cup E$ be a refined extension of *P* and *M* a stable model of $P \cup E$. Then *M* is also a stable model of *P*.

The addition to tautologies to a GLP does not introduce new stable models nor eliminate stable models.

Theorem 4.3. from Alferes et al. (2005): Let $\mathcal{P}$ be any DLP and $\mathcal{E}$ a sequence of sets of tautologies. *M* is a refined dynamic stable model of $\mathcal{P}$ iff *M* is a refined dynamic stable model of $\mathcal{P} \cup \mathcal{E}$. So, $RDSM((P_M, P_0 \cup E_0, \mathcal{P} \cup \mathcal{E})) = RDSM(P_M, P_0, \mathcal{P})$ and $SR(\mathcal{R}) = SR(\mathcal{R}')$ (by definition 6).

Going back to the property of conservation, stating that when the output of the existing module is a dynamic stable model of the DLP $(P_0, \mathcal{P})$, then it is a stable recommendation, it could be desirable to have a stronger result, namely that the semantics obeys a notion of strong conservation according to which it would be *the only* stable recommendation. It turns out that the stable recommendation semantics does not obey such property:

Proposition 7 (strong conservation): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then

$$M \in RDSM((P_0, \mathcal{P})) \not\Rightarrow SR(\mathcal{R}) = \{M\}$$

This may, however, be also seen as an advantage of the system. The owner and user policy may introduce interesting new sets of recommendations, facilitating diversity and non-determinism. Consider the following simple example: Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF where $M = \{a\}, P_0 = \{not\ a \leftarrow\}, \mathcal{P} = (P_1)$ and

$$P_1: \quad a \leftarrow not\ b.$$
$$\quad\ b \leftarrow not\ a.$$

$\mathcal{R}$ has two stable recommendations: $\{a\}$ and $\{b\}$. Here, the owner rule overrides the initial recommendation and the user rules introduce two possibilities, one of the being identical to the initial recommendation $\{a\}$. But given that the owner rules should prevail over the initial recommendations, one might even expect the stable set $\{b\}$ to be preferred to $\{a\}$, not the other way around. So the strong conservation property may not always be desirable.

But for case that some specific system needs to obey the strong conservation property, we also introduce an alternative semantics that only keeps the models closest to the initial recommendation. In order to define it, we first need to introduce the notion of distance between interpretations:

Definition 7 (distance between interpretations): Let $\mathcal{A}$ be a set of propositional atoms and M, M_1 and M_2 interpretations of $\mathcal{A}$. We say that M_1 is closer to M than M_2, denoted by $M_1 \sqsubset_M M_2$ iff

$$(M_1 \setminus M \cup M \setminus M_1) \subset (M_2 \setminus M \cup M \setminus M_2)$$

Now we can define the minimal change semantics and prove that it obeys the strong conservation property:

Definition 8 (minimal change SR semantics): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF and $M_{\mathcal{R}}$ an interpretation. $M_{\mathcal{R}}$ is a minimal change stable recommendation iff $M_{\mathcal{R}} \in SR(\mathcal{R})$ and $\nexists M'_{\mathcal{R}} \in RDSM(\Upsilon(\mathcal{R})) : M'_{\mathcal{R}} \sqsubset_M M_{\mathcal{R}}$. $SR^m(\mathcal{R})$ denotes the set of all minimal change stable recommendations.

Proposition 8 (strong conservation): Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then

$$M \in RDSM((P_0, \mathcal{P})) \Rightarrow SR^m(\mathcal{R}) = \{M\}$$

Proof: From definitions 8 and 7 it follows that M is the only stable recommendation, since $M \in RDSM((P_0, \mathcal{P}))$ and $M \sqsubset_M M'$ for all other stable recommendations M' (there cannot be a model closer to M than M itself).

As for the remaining properties, the minimal change stable recommendation semantics obeys conservation, positive and negative supportiveness, and immunity to tautologies. As expected, it no longer generalises DLP as it is well known that DLP accepts non-minimal dynamic stable models that would be eliminated e.g., in the case of an empty output of the initial module.

6 Implementation and results

6.1 Description of the implementation

The system was implemented as an online application[10] using a PHP-based initial collaborative filtering recommender system[11]. Figures 3 and 4 present two screenshots of ERASP. The product database consists of the complete MovieLens[12] dataset of 3883 movies plus their genre and year information. After the user of the system rates some movies and receives some initial recommendations, he can edit his preferences in form of a DLP using an interface that also provides some automatised rule creation. Together with the initial recommendations, the facts from the product database and the owner specifications, the user program gets processed by an *Smodels*-based solver, where the following steps occur:

1 First, the whole input gets parsed, an object representation of the corresponding logic programming rules is created and a DLP is formed from it.

2 We use *Smodels* as our external answer set solver and it can only work with variable-free programs. So the next step is to produce an equivalent DLP without variables. This process is called grounding and is mainly performed by the program *Lparse*. But since *Lparse* can only ground normal logic programs, we first need to transform our DLP into a normal logic program. Intuitively, this involves remembering which program of the DLP each rule belongs to and getting rid of default negation in the heads. The former is achieved by adding an extra atom to the body of every rule, while the latter is achieved by deleting all default *not*s from the program and prepending *p_* or *n_* to the name of every atom, indicating whether it was a positive or negative literal. Apart from this, we need to prevent *Lparse* from throwing away instantiations of rules just because the formerly negative literals in their bodies, now transformed into positive atoms with *n_* prepended, cannot be derived by the program. This is achieved by adding a fact for every such atom.

3 After the grounded DLP $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ is parsed and an object representation is created, it gets transformed into an equivalent normal logic program $\mathcal{P}^R$. This transformation is described in detail in Banti et al. (2005). The stable models of $\mathcal{P}^R$ directly correspond to the dynamic stable models of $\mathcal{P}$.

4 The transformed program is again passed to *Lparse* in order to be given to *Smodels* with the according input format.

5 *Smodels* then computes all the stable models and our program writes the recommendations for each model into an SQL database.

Figure 3 ERASP user model interface (see online version for colours)

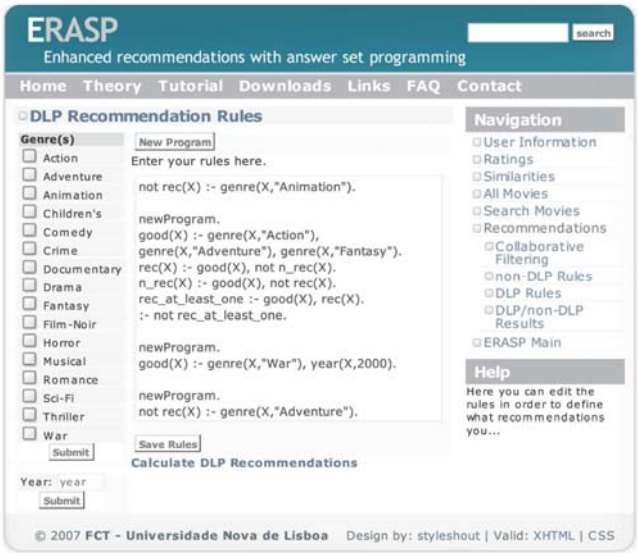
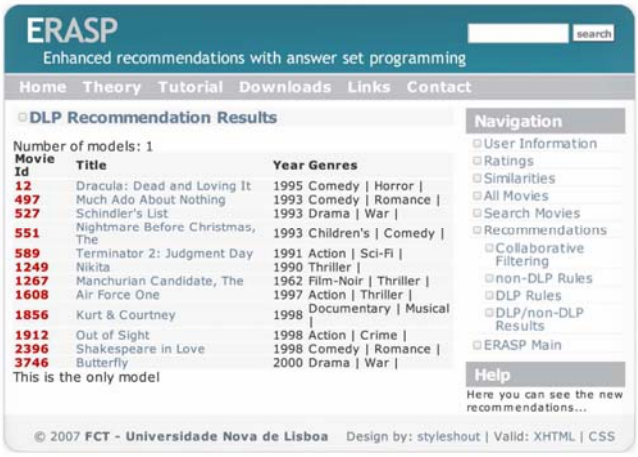


Figure 4 ERASP recommendation interface (see online version for colours)



6.2 Benchmarks

For the performance testing of the system in terms of time, we ran each of the DLPs of the example in Section 4. The programs were run 100 times and an average value for each of the steps was calculated. Six different input databases were used in order to investigate the importance of the size of the input:

- *all:* 3883 movies with all concepts (17406 atoms)
- *no title:* 3883 movies with all concepts except *title*(*ID, Title*) (9640 atoms)
- *1000:* 1000 movies with all concepts (4428 atoms)
- *1000 no title:* 1000 movies with all concepts except *title*(*ID, Title*) (3440 atoms)
- *500:* 500 movies with all concepts (2249 atoms)
- *500 no title:* 500 movies with all concepts except *title*(*ID, Title*) (1752 atoms).

The concepts in the complete database are *title*(*ID, Title*), *genre*(*ID, GenreName*) and *year*(*ID, Year*). We chose to exclude the concept *title*(*ID, Title*) in some of the testing databases, since it seems to be the concept that would be used most rarely. Recommender systems usually recommend items that the user does not know about and employing a concept that pinpoints the item itself can be seen as redundant.

In the following, we will present the performance results of each program from the example in Section 4. The machine used for testing has the following specifications: Intel Pentium D 3.4 Ghz (2 Mb cache) processor and 1 Gbyte of RAM.

As one can see, the times in seconds for parsing (PA), grounding (GR), transformation (TR) and for the stable models computation (SM) are included. The total time (Total) is the sum of the previous times. Clearly, decreasing input size means less time, as can also be seen by looking at the results of the remaining programs. It is important to note that we are testing with the owner policy P_0, which is updated by the user programs.

Table 6 Performance results of program $\mathcal{P}_1$

Input	*PA*	*GR*	*TR*	*SM*	*Total*
All	0.1953	1.5956	0.4985	1.2886	3.5779
No title	0.1507	1.1553	0.3050	0.9187	2.5296
1000	0.0946	0.4676	0.1312	0.3075	1.0010
1000 no title	0.0812	0.3285	0.1344	0.2184	0.7625
500	0.0756	0.2737	0.0879	0.2066	0.6438
500 no title	0.0603	0.2108	0.0679	0.1266	0.4656

Table 7 Performance results of program $\mathcal{P}_2$

Input	*PA*	*GR*	*TR*	*SM*	*Total*
All	0.1962	1.5960	0.5026	1.2939	3.5887
No title	0.1496	1.1588	0.3103	0.9223	2.5411
1000	0.0957	0.4699	0.1332	0.3095	1.0083
1000 no title	0.0844	0.3300	0.1370	0.2195	0.7709
500	0.0734	0.2742	0.0889	0.2087	0.6453
500 no title	0.0598	0.2176	0.0690	0.1279	0.4743

Table 8 Performance results of program $\mathcal{P}_3$

Input	*PA*	*GR*	*TR*	*SM*	*Total*
All	0.1985	1.5977	0.5023	1.3123	3.6108
No title	0.1492	1.1540	0.3190	0.9634	2.5856
1000	0.0956	0.4749	0.1332	0.3262	1.0300
1000 no title	0.0831	0.3320	0.1365	0.2351	0.7866
500	0.0761	0.2806	0.0893	0.2251	0.6711
500 no title	0.0603	0.2172	0.0692	0.1424	0.4891

Table 9 Performance results of program $\mathcal{P}_4$

Input	*PA*	*GR*	*TR*	*SM*	*Total*
All	0.1966	1.6389	0.5042	1.3201	3.6597
No title	0.1503	1.1813	0.4069	0.8588	2.5973
1000	0.0960	0.4760	0.1352	0.3101	1.0173
1000 no title	0.0845	0.3385	0.1381	0.2226	0.7837
500	0.0743	0.2777	0.0903	0.2101	0.6524
500 no title	0.0598	0.2164	0.0705	0.1253	0.4720

While the parsing step is linear, the grounding can be exponential in worst case, depending on the number of atoms in the input, the number of rules and the different variables appearing in them. More specifically, the instantiation of a rule is exponential in the number of different variables occurring in that rule. The upper bound of number of rules created by the transformation depends on the total number of rules and the initial language. The transformation is linear and the worst case upper bound is $2m + l$ rules after the transformation, where m is the number of rules in $\rho(\mathcal{P})$ and l is the number of input predicates.[13]

The computation of (dynamic) stable models can take a lot of time, since the task of finding a stable model is NP-hard. This means that in the worst case it is extremely

difficult to find a solution. The brute force way to compute stable models is to generate all subsets of the set of atoms of the whole program (which includes almost all of the input in our case) and then test for each subset whether it satisfies the condition of being a stable model. This is the most costly way of finding stable models and the algorithms that are employed have the main task to decrease the search space. *Smodels* employs a bottom-up backtracking search with a pruning method that can be very efficient. For more on that topic the reader is referred to Niemelä and Simons (1997). The main idea is to decrease the search space as soon as some evidence can be found that this part of the space is not needed anymore. This makes it also obvious how the size of the grounded and hence the transformed program influences the task of finding stable models, as can be seen in the test results.

Table 10 All test results ordered by input and program

Input	$\mathcal{P}$	*PA*	*GR*	*TR*	*SM*	*Total*
All	1	0.1953	1.5956	0.4985	1.2886	3.5779
All	2	0.1962	1.5960	0.5026	1.2939	3.5887
All	3	0.1985	1.5977	0.5023	1.3123	3.6108
All	4	0.1966	1.6389	0.5042	1.3201	3.6597
No title	1	0.1507	1.1553	0.3050	0.9187	2.5296
No title	2	0.1496	1.1588	0.3103	0.9223	2.5411
No title	3	0.1492	1.1540	0.3190	0.9634	2.5856
No title	4	0.1503	1.1813	0.4069	0.8588	2.5973
1000	1	0.0946	0.4676	0.1312	0.3075	1.0010
1000	2	0.0957	0.4699	0.1332	0.3095	1.0083
1000	3	0.0956	0.4749	0.1332	0.3262	1.0300
1000	4	0.0960	0.4760	0.1352	0.3101	1.0173
1000 no title	1	0.0812	0.3285	0.1344	0.2184	0.7625
1000 no title	2	0.0844	0.3300	0.1370	0.2195	0.7709
1000 no title	3	0.0831	0.3320	0.1365	0.2351	0.7866
1000 no title	4	0.0845	0.3385	0.1381	0.2226	0.7837
500	1	0.0756	0.2737	0.0879	0.2066	0.6438
500	2	0.0734	0.2742	0.0889	0.2087	0.6453
500	3	0.0761	0.2806	0.0893	0.2251	0.6711
500	4	0.0743	0.2777	0.0903	0.2101	0.6524
500 no title	1	0.0603	0.2108	0.0679	0.1266	0.4656
500 no title	2	0.0598	0.2176	0.0690	0.1279	0.4743
500 no title	3	0.0603	0.2172	0.0692	0.1424	0.4891
500 no title	4	0.0598	0.2164	0.0705	0.1253	0.4720

How methods of search space pruning can positively influence the time it takes to compute stable models can be seen when looking at programs $\mathcal{P}_3$ and $\mathcal{P}_4$. From the input size of 1000 movies, one can notice that the stable models computation of program $\mathcal{P}_4$ takes less time than of $\mathcal{P}_3$. The program update rules out the occurrence of certain atoms in the stable models of the programs and this is used to minimise the search space as soon as possible. This is also reflected in the number of stable models which is six in the case of $\mathcal{P}_3$ and only one in the case of $\mathcal{P}_4$.

While we have not so fast results for a big input database, the times for programs with a smaller input can be more than six times faster.

7 Optimisations

7.1 Performance

One of the major drawbacks of the system is its performance when dealing with a too large input database. One way to improve performance drastically is to decrease the size of the input. With an iteration mechanism, stable models could be computed in a step-wise manner, offering the user already some results, while processing the following input iteratively. The input database could as well be iteratively fed to the solver by first choosing the items that are ranked highest after some criteria of the initial recommender system, like e.g., the rating of the nearest neighbours in a collaborative filtering recommender system. This would, of course, at best be an approximation since the stable models cannot be computed incrementally. For every incrementation step of the input the models can change. The number *n* of items chosen in each step could be dependent on the demands of the domain. For example, if recommendations have to be presented very quickly as advertising products, *n* would be set very low, while *n* could be higher if the domain is one where the recommendation time is secondary, like e.g., in the area of financial advisory (Felfernig and Kiener, 2005).

Another way to improve the system in terms of speed would be to select the input according to the rules. Analysing the rules for appearing concepts could provide a mechanism for choosing the according input where e.g., the concepts that are not used are absent. Similar to our tests, where we used an input database where the concept of *title*(*ID, Title*) was missing, the system could have several databases prepared, where each one would be used with an according set of rules.

7.2 User interface

An often criticised point of the system is the task of the user to write rules. It was shown that such kind of interaction can be burdening for the user (Claypool et al., 2001) and one might assume that the idea of recommender systems is to automatically provide recommendations without any need for direct user interaction. To tackle this issue, we first need to consider the following motivations:

1 the system could be used as an optional add-on for a user of an existing recommender system

2 the system could be used by experts of specific domains that demand higher accuracy and more important, reliability of recommendations

3 the system could be employed directly by owners of recommender systems to define rules concerning policies.

Obviously, the rule-based character of the system does not appeal to every user that searches for products or information, but it might be appealing for those who are interested in the properties and advantages of a rule-based system, like in the cases mentioned above.

But there are also possibilities of making rule creation easier for the user, not demanding that he has to learn answer-set programming. This would mean that a way has to be found that avoids the requirement of knowing how to program and that avoids the usage of a simplification mechanisms (like e.g., choosing predefined concepts). There has been work on transforming natural language sentences into logic programs (Fuchs and Schwitter, 1995) and creating natural language interfaces for databases (Androutsopoulos et al., 1995). It would be a challenging and interesting way to complement a recommender system with rule-based user models. This could preserve expressibility to a major extent and provide a more user-friendly way of defining rules.

Further, rules could be created by suggestion or learning. Imagining a system that is started by experts that understand logic programming and that know how to program rules with high expressibility, we could provide new users that are not familiar with answer set programming, with suggestions. Experts can tag their rules, an idea seen very much established in *Amazon* where items are tagged. This tagging can provide some basic entry for users that do not want to or do not know how to deal with rules. While experts could tag each others rules, novice users could search for tags and receive tag suggestions. Of course, it would take some time to start up such a system, but it is well known that many recommender systems need such start up time as well, e.g., where items have to be rated, like it is the case in a collaborative filtering recommender system. The power of tagging, which is also seen as providing meaning, is widely exploited in the Web 2.0[14] paradigm. There has also been significant work on learning rules by induction (e.g., Aitken, 2002), which could be another way to make the system more user friendly by suggesting learned rules.

8 Related work

In our system we use a rule-based approach for modelling knowledge about the user, and hence it can be categorised as a knowledge-based system. The architecture is not considered to be a stand-alone system, but a complement to existing recommender systems. The functionality of the idea includes the use of an (external) input module that offers valuable recommendations, such as e.g., a collaborative filtering recommender system. A complemented system would therefore form a cascade hybrid recommender system (Burke, 2002), using the output of one system to feed our module which then optimises the results. While e.g., in Burke (1999) collaborative filtering is used to improve the recommendations of the knowledge-based system, we propose the opposite direction. A further difference to the majority of recommender systems is the use of an explicit user model. Though not being fully comparable to most of the traditional recommender systems, there are related approaches that we consider similar in some characteristics.

For example, in Chesñevar and Maguitman (2004) defeasible logic programming *(DeLP)* is employed in a website recommender system. The system *ArgueNet* is proposed where the user can formulate his preferences in the form of a defeasible logic program. *DeLP* is a defeasible argumentation formalism that is based on logic programming and the underlying logical language is that of extended logic programming. Classical and default negation are allowed. The *ArgueNet* system classifies relevant search results according to the preferences of the user. While our system receives results from a recommender system, *ArgueNet* requests information from a search engine. In both approaches, a form of logic programming is used to define the user model. In the *DLP* approach though, the user can update his model according to changes in preferences without worrying if the proposed specification is free of contradiction or tautological information. In the *DeLP* approach it is assumed that all the defined rules are consistent and no updates in the sense of *DLP* are possible. Both approaches are offering something that is desirable in most recommender systems, and that is the intrinsic ability to provide justifications. While *ArgueNet* is a recommender system on its own, our approach falls into the category of a complementary system. Since there are already efficient and data rich recommender systems that improved over the years, their recommender capability should be harvested. The main purpose of the DLP system is to improve the results and to add a user controlled personalisation to existing systems.

Another system that can be considered related from a user interaction point of view is the novel type of hybrid recommender system presented in Schafer et al. (2002), dubbed *meta-recommender system*. The idea behind the approach is to integrate recommendations from more than one source with *diverse* information, while giving the users the possibility to modify their requirements. This kind of user control is somehow similar to our proposal of defining a user model, and not usual for traditional recommender systems. The meta-recommender system *MetaLens* regards the results of an existing recommender module and allows the user to specify weights for certain fixed criteria. While this system helps to find interconnected information on-the-fly, our proposal is more focused on an elaborated user model that includes long-term and short-term properties, as well as an expressive modelling language. Our system only receives information from one source and therefore does not include a data integration layer like the meta-recommender system *MetaLens*. In the following dynamic version, *DynamicLens* (Schafer, 2005), a dynamic query interface is introduced, making it easier for the user to observe the importance of certain requirements and giving

him more control over the results. Both systems create, among others, an on-top refinement of already existing recommendations and also provide a high degree of interactive user personalisation.

Our system can be summarised as a complementary module for existing recommender systems allowing for an improvement of results by means of explicit user modelling. In most of the large-scale systems, performance and effectiveness are of high importance and therefore, implicit user model creation is essential. That might lead to imperfect results which can be refined using the DLP idea of letting the user define a more specific user model.

9 Conclusions and future work

In this paper we proposed what can be seen as part of a knowledge based (or hybrid) recommender system, with several characteristics, namely:

- allowing user personalisation with complex and expressive rules, improving the quality of recommendations
- allowing for interaction with the user by means of updates to those rules, automatically removing inconsistencies
- taking into account the output of other recommendation modules
- allowing for customisation by the owner of the system
- providing a semantics with multiple recommendation sets, facilitating diversity and non-determinism in recommendations
- enjoying a formal, well defined semantics which supports justifications
- enjoying all the other formal properties mentioned in the previous section, and many more inherited from DLP and ASP such as the expressiveness of the language and the efficient implementations.

Inspired by the idea of a meta-recommender system that integrates diverse information, we will consider a more developed architecture that receives input from more than one source. Data integration and information source selection has already been a trial application for ASP and in our system design this will be regarded. The main motivation is to move away from the idea of a recommender system being limited to one product domain and to embrace the integration of interconnected information.

We believe that our proposal encodes some important concepts that can bring an added value to existing systems.

Acknowledgements

We would like to thank Paulo Lopes for providing dedicated hardware for running the benchmark tests, for his reliable technical support and also for all his advices.

Martin Slota is partially supported by FCT Scholarship SFRH/BD/38214/2007.

References

Aitken, J.S. (2002) 'Learning information extraction rules: an inductive logic programming approach', in van Harmelen, F. (Ed.): *Proceedings of the 15th Eureopean Conference on Artificial Intelligence, ECAI'2002*, July 2002, Lyon, France, IOS Press, pp.355–359.

Alferes, J.J., Banti, F., Brogi, A. and Leite, J.A. (2005) 'The refined extension principle for semantics of dynamic logic programming', *Studia Logica*, Vol. 79, No. 1.

Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H. and Przymusinski, T. (2000) 'Dynamic updates of non-monotonic knowledge bases', *Journal of Logic Programming*, Vol. 45, Nos. 1–3.

Androutsopoulos, I., Ritchie, G. and Thanisch, P. (1995) 'Natural language interfaces to databases – an introduction', *Journal of Language Engineering*, Vol. 1, No. 1, pp.29–81.

Balabanović, M. and Shoham, Y. (1997) 'Fab: content-based, collaborative recommendation', *Communications of the ACM*, Vol. 40, No. 3, pp.66–72.

Banti, F., Alferes, J.J. and Brogi, A. (2005) 'Operational semantics for dyLPs', in Bento, C., Cardoso, A. and Dias, G. (Eds.): *Proceedings*, *Progress in Artificial Intelligence, 12th Portuguese Conference on Artificial Intelligence, EPIA 2005,* Covilhã, Portugal, 5–8 December, 2005*,* Vol. 3808 of Lecture Notes in Computer Science, Springer, pp.43–54.

Baral, C. (2003) *Knowledge Representation, Reasoning and Declarative Problem Solving*, Cambridge University Press.

Billsus, D. and Pazzani, M. (2000) 'User modeling for adaptive news access', *User Modeling and User-Adapted Interaction*, Vol. 10, Nos. 2–3, pp.147–180.

Burke, R.D. (1999) 'Integrating knowledge-based and collaborative-filtering recommender systems', in *AAAI Workshop on AI in Electronic Commerce*, pp.69–72, AAAI.

Burke, R.D. (2000) 'Knowledge-based recommender systems', in Kent, A. (Ed.): *Encyclopedia of Library and Information Systems*, Vol. 69, Chapter Supplement 32, M. Dekker.

Burke, R.D. (2002) 'Hybrid recommender systems: survey and experiments', *User Modeling and User-Adapted Interaction*, Vol. 12, No. 4, pp.331–370.

Chesñevar, C. and Maguitman, A. (2004) 'ArgueNet: an argument-based recommender system for solving web search queries', in *Procs. of the 2nd International IEEE Conference on Intelligent Systems*, Varna, Bulgaria, IEEE Press, pp.282–287.

Claypool, M., Le, P., Wased, M. and Brown, D. (2001) 'Implicit interest indicators', in *Proceedings of the 2001 International Conference on Intelligent User Interfaces*, pp.33–40.

Eiter, T. (2005) 'Data integration and answer set programming', in Baral, C., Greco, G., Leone, N. and Terracina, G. (Eds.): *Proceedings*, *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005,* Diamante, Italy, 5–8 September, 2005, Vol. 3662 of Lecture Notes in Computer Science, Springer, pp.13–25.

Eiter, T., Fink, M., Sabbatini, G. and Tompits, H. (2002) 'A generic approach to knowledge-based information-site selection', in Fensel, D., Giunchiglia, F., McGuinness, D.L. and Williams, M-A. (Eds): *Procs. of 8th Int. Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, Morgan Kaufmann, pp.459–469.

Eiter, T., Lukasiewicz, T., Schindlauer, R. and Tompits, H. (2004) 'Combining answer set programming with description logics for the semantic web', in Dubois, D., Welty, C.A. and Williams, M-A. (Eds): *Procs. of 9th Int. Conference on Principles of Knowledge Representation and Reasoning (KR'04)*, AAAI Press, pp.141–151.

Felfernig, A. and Kiener, A. (2005) 'Knowledge-based interactive selling of financial services with FSAdvisor', in Veloso, M.M. and Kambhampati, S. (Eds): *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, 9–13 July, 2005, Pittsburgh, Pennsylvania, USA, AAAI Press/The MIT Press, pp.1475–1482.

Fuchs, N.E. and Schwitter, R. (1995) *Specifying Logic Programs in Controlled Natural Language*, Technical Report ifi-95.17.

Gelfond, M. and Lifschitz, V. (1990) 'Logic programs with classical negation', in Warren, D. and Szeredi, P. (Eds.): *Proceedings of the 7th International Conference on Logic Programming*, Cambridge, MA, MIT Press, pp.579–597.

Goldberg, D., Nichols, D., Oki, B.M. and Terry, D. (1992) 'Using collaborative filtering to weave an information tapestry', *Communications of the ACM*, Vol. 35, No. 12, pp.61–70, Special issue on information filtering.

Kelly, J.P. and Bridge, D. (2006) 'Enhancing the diversity of conversational collaborative recommendations: a comparison', *Artificial Intelligence Review*, Vol. 25, No. 1, pp.79–95.

Lam, S.K., Frankowski, D. and Riedl, J. (2006) 'Do you trust your recommendations? An exploration of security and privacy issues in recommender systems', in Müller, G. (Ed.): *Proceedings*, *Emerging Trends in Information and Communication Security, International Conference, ETRICS 2006,* Freiburg, Germany, 6–9 June, 2006, Vol. 3995 of Lecture Notes in Computer Science, Springer, pp.14–29.

Leite, J.A. (2003) *Evolving Knowledge Bases*, IOS 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 Transactions on Computational Logic*, Vol. 7, No. 3, pp.499–562.

Niemelä, I. and Simons, P. (1997) 'Smodels – an implementation of the stable model and well-founded semantics for normal LP', in Dix, J., Furbach, U. and Nerode, A. (Eds.): *Proceedings*, *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97,* Dagstuhl Castle, Germany, 28–31 July, 1997, Vol. 1265 of Lecture Notes in Computer Science, Springer, pp.421–430.

Pazzani, M.J. and Billsus, D. (2007) 'Content-based recommendation systems', in Brusilovsky, P., Kobsa, A. and Nejdl, W. (Eds): *The Adaptive Web*, Vol. 4321 of Lecture Notes in Computer Science, Chapter 10, pp.325–341, Springer-Verlag.

Resnick, P. and Varian, H.R. (1997) 'Recommender systems', *Communications of the ACM*, Vol. 40, No. 3, pp.56–58.

Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P. and Riedl, J. (1994) 'GroupLens: an open architecture for collaborative filtering of netnews', in *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, . ACM, pp.175–186.

Schafer, J. (2005) 'Dynamiclens: a dynamic user-interface for a meta-recommendation system', in *Beyond Personalization 2005: A Workshop on the Next Stage of Recommender Systems Research*, ACM, pp.72–76.

Schafer, J.B., Konstan, J.A. and Riedl, J. (2001) 'E-commerce recommendation applications', *Data Mining and Knowledge Discovery*, Vol. 5, Nos. 1/2, pp.115–153.

Schafer, J.B., Konstan, J.A. and Riedl, J. (2002) 'Meta-recommendation systems: user-controlled integration of diverse recommendations', in Kalpakis, K., Goharian, N. and Grossmann, D. (Eds.): *Proceedings of the Eleventh International Conference on Information and Knowledge Management (CIKM-02)*, ACM Press, New York, pp.43–51.

Schindlauer, R. (2006) 'Answer-set programming for the semantic web', Phd thesis, Vienna University of Technology, Austria.

Smyth, B. and Cotter, P. (2001) 'Personalized electronic program guides for digital TV', *AI Magazine*, Vol. 22, No. 2, pp.89–98.

Vossen, G. and Hagemann, S. (2007) *Unleashing Web 2.0: From Concepts to Creativity*, Morgan Kaufmann.

Notes

1 The main difference between traditional logic programming (e.g., Prolog) and ASP is how negation as failure is interpreted. In traditional logic programming, negation-as-failure indicates the failure of a derivation; in ASP, it indicates the consistency of a literal. In contrast to Prolog, the semantics of ASP do not depend on a specific order of evaluation of the rules and of the atoms within each rule. For more on ASP, namely its semantics and applications, the reader is referred to Baral (2003).

2 Available at http://www.dlvsystem.com/.

3 Available at http://www.tcs.hut.fi/Software/smodels/.

4 LPs with default and strong negation both in the body and head of rules.

5 Recent redevelopments have formalised an extension of answer-set programming that allows for the interface with ontologies specified in Description Logics (Eiter et al., 2004), making our work easily extensible to the case where product information is available in the semantic web instead of a local relational database.

6 This encoding uses the classic even loop through negation which, in ASP, produces two models, each with one of the propositions belonging to it.

7 We restrict the listings of models to propositions of the form *rec*(*Id*).

8 IDs of newly introduced movies are written in italics.

9 These rules are, again, a classic construct of ASP. The first two rules state that each good item X is either recommended [*rec*(*X*)] or not recommended [*n_rec*(*X*)]. Then the third rule makes the proposition *rec_at_least_one* true if at least one good item is recommended. Finally, the fourth rule, an integrity constraint, eliminates all models where *rec_at_least_one* is not true. The actual recommender system would, like most answer-set solvers, have special syntactical shortcuts for this kind of specifications, since we cannot expect the user to write this kind of rules.

10 The prototype can be found at http://ssdi.di.fct.unl.pt/~a21451.

11 The collaborative filtering algorithm can be found at www.vogoo-api.com.

12 The MovieLens dataset can be found at http://www.grouplens.org/.

13 Banti et al. (2005) Theorem 2.

14 There are many references concerning this topic; the reader is referred to Vossen and Hagemann (2007) for an introduction.