Answer-Set Programming Based Dynamic User Modeling for Recommender Systems

João Leite and Manoela Ilić

CENTRIA, Universidade Nova de Lisboa, Portugal

Abstract. In this paper 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.

1 Introduction

Recommender systems have become an essential tool for finding the needle in the haystack of the World Wide Web - the information or item one is searching for [22]. 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 [23].

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. [21,16]) 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.[3,6]) where descriptions of the content of items is matched against user profiles, employing techniques from the information retrieval field; knowledge-based recommendation (e.g. [9,14]) 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. [8,25]) 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 [10].

Recommender systems can also be categorised, according to the way they interact with the user [7], as being *single shot systems* where for each request the system make its interpretation of information and proposes recommendations to the user without taking in account any previous interaction, and *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 recommendation 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 behavior), 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 [5], making it a challenging task to include both sensibility to changes of taste and maintenance of permanent preferences. While implicit user modeling disburdens the user of providing direct feedback, explicit user modeling 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 [17].

In this paper we will concentrate on explicit user modeling 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 consider 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 recommendation system.

In this paper, we borrow concepts from the area of knowledge representation and non-monotonic reasoning, to set forth a proposal that aims at extending systems with the possibility of allowing users to specify their individual models and preferences, while taking into account the three claims above. Concretely, we adopt the paradigm of *Dynamic Logic Programming* [2,18,1] to suit our needs.

Dynamic Logic Programming (DLP) is an extension of Answer-set Programming (ASP) [15] 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 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 have been made in recent years. The existence of very efficient ASP solvers (e.g. DLV [19] and SMODELS[2] [20]) make it finally possible to investigate some serious applications in the area of e.g. data integration [11], data source selection [12] and also the Semantic Web [24].

According to *DLP*, as extension of ASP, knowledge is given by a series of theories, encoded as generalized logic programs[3] (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, the role of *DLP* is to employ 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, at the end of the sequence, newer rules leaving to *DLP* the task of ensuring that these rules are in force, and that previous ones are valid (by inertia) only so far as possible, i.e. that they are kept for as long as they are not in conflict with newly added ones, these always prevailing.

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; a well defined semantics; 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; easy connection with relational databases (ASP can also be seen as a query language, more expressive than SQL); support for explanations; amongst others.

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 recommendation system, enabling them to specify some policies that may not be captured by the existing recommendation system (e.g. preference for

[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 [4] and [26].

[2] Available at www.dlvsystem.com and www.tcs.hut.fi/Software/smodels/ resp.

[3] LPs with default and strong negation both in the body and head of rules.

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 recommendation system, a set of rules specified by the owner of the recommendation system and a sequence of rules specified by the user, for which we provide an expressive language. The modules are combined in a way such 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, keep the result of the initial recommendation module as much as possible in the final output, among others.

The remainder of this paper is organised as follows. In Sect. 1, for self-containment, we briefly recap the notion of *DLP*, establishing the language used in the paper. In Sect. 3 we define our framework and its semantics. In Sect. 4 we preset a brief illustrative example. In Sect. 5 we discuss some properties and conclude in Sect. 6.

2 Dynamic Logic Programming

For self containment, in this Appendix, we briefly recap the notion and semantics of Dynamic Logic Programming needed throughout. More motivation regarding all these notions can be found in [2,18].

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 $B(r) = \{L_1, \ldots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \ldots, L_n$. A **tautology** is a rule of the form $L \leftarrow Body$ with $L \in Body$. A **generalized logic program** (*GLP*) P, in $\mathcal{A}$, is a finite or infinite set of rules. If $H(r) = A$ (resp. $H(r) = not\, A$) then $not\, H(r) = not\, A$ (resp. $not\, H(r) = A$). If $H(r) = \neg A$, then $\neg H(r) = A$. By the **expanded generalized logic program** corresponding to the GLP P, denoted by $\mathbf{P}$, 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 $M \vDash L$, iff $L \in M$, and false otherwise. A default literal $not\, L$ is true in M, denoted by $M \vDash not\, L$, iff $L \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \vDash B$, iff each literal in B is true in M. An interpretation M of $\mathcal{A}$ is an **answer set** of a GLP P iff $M' = least\,(\mathbf{P} \cup \{not\, A \mid A \notin M\})$, where $M' = M \cup \{not\_A \mid A \notin M\}$, 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 $AS\,(P)$ denote the set of answer-sets of P.

A **dynamic logic program** (DLP) is a sequence of generalized logic programs. Let $\mathcal{P} = (P_1, ..., P_s)$ and $\mathcal{P}' = (P'_1, ..., P'_n)$ 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 $\left(P'_i, P'_j, \mathcal{P}\right)$ to denote $\left(P'_i, P'_j, P_1, ..., P_n\right)$. We can now set forth the definition of a 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:*

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

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):*

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

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

Definition 3 (Semantics of DLP). *Let* $\mathcal{P} = (P_1, ..., \ P_s)$ *be a* DLP *and* M *an interpretation.* M *is a* (refined) dynamic stable model *of* $\mathcal{P}$ *iff*

$$M' = least\,(\rho\,(\mathcal{P}) - Rej\,(\mathcal{P}, M) \cup Def\,(\mathcal{P}, M))$$

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

3 Framework and 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).

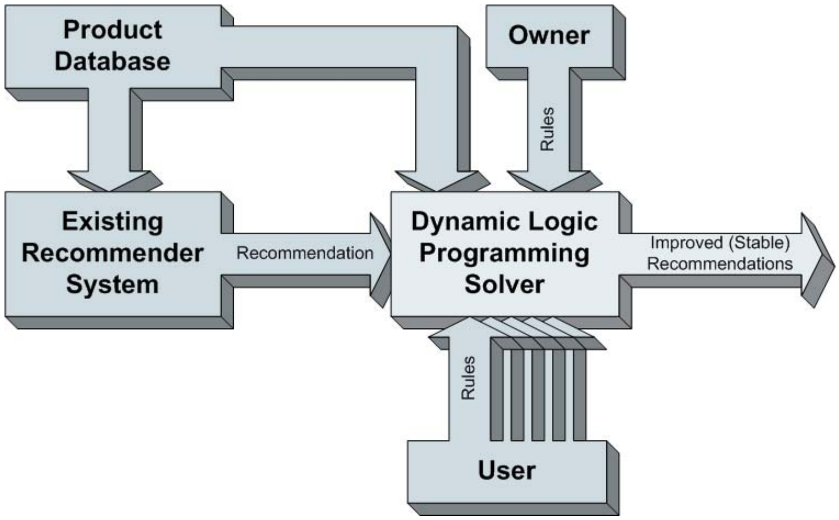
Product Database
Owner
Rules
Existing Recommender System
Recommendation
Dynamic Logic Programming Solver
Improved (Stable) Recommendations
Rules
User

Fig. 1. Outline of the system

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 of the form $rec/1$ where the items are the terms of the predicate, and the interpretation will contain those predicates corresponding to the recommended items. For example, the interpretation $M = \{rec(\text{“}Takk\text{”}), rec(\text{“}Underground\text{”}), rec(\text{“}The\ K\ddot{o}ln\ Concert\text{”}), rec(\text{“}Paris, Texas\text{”}), rec(\text{“}Godfather\text{”})\}$ encodes the recommendations for the films “*Godfather*”, “*Underground*” and “*Paris, Texas*”, as well as for the music albums “*Takk*” and “*The Köln Concert*”. 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 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 generalised logic program representing the owner's policy, and a dynamic logic program, representing the rules (and their evolution) specified by the user. The Product Database is, typically, a relational database[4] easily represented by a set of facts in a logic program. For simplicity, without loss of generality, we assume such database to be part of the generalised logic program representing the owner's policy. To formalise this notion, we introduce the concept of Dynamic Recommender Frame:

Definition 4 (Dynamic Recommender Frame). *Let $\mathcal{A}$ be a set of propositional atoms. A Dynamic Recommender Frame (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 generalised logic program over $\mathcal{A}$, and $\mathcal{P}$ a dynamic logic program over $\mathcal{A}$.*

The semantics of a Dynamic Recommender Frame is given by the set of stable models of its transformation into a Dynamic Logic Program. 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 recommendation 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 whose 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 rather natural: the owner must be given the option to supersede the initial recommendation module (e.g. 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 inside the initial module (e.g. a sub-symbolic system such as a neural network).

With these principles in mind, we first define a transformation from a Dynamic Recommender Frame into a Dynamic Logic Program:

Definition 5 (Υ). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P}\rangle$ be a Dynamic Recommender Frame. Let $\Upsilon(\mathcal{R})$ be 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), in turn followed by the sequence of user specifications ($\mathcal{P}$). 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 Dynamic Recommender Frame and M_R an interpretation. M_R is a stable*

[4] Recent developments have formalised an extension of Answer-Set Programming that allows for the interface with ontologies specified in Description Logics [13], making our work easily extensible to the case where product information is available in the Semantic Web, instead of a local relational database.

recommendation iff M_R is a dynamic stable model of $\Upsilon(\mathcal{R})$. Let $SR(\mathcal{R})$ denote the set of all stable recommendations.

4 Short Illustrative Example

In this Section we present a small and very simplified example, with the purpose of illustrating some (very few) features of our proposal.

Let's consider an on-line dish and wine recommender, with some existing recommendation 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: $M = \{rec(d_1), rec(d_2), rec(w_1), rec(w_2), rec(d_7), rec(w_6)\}$ where d_i are dishes offered in a restaurant and w_i are wines. Then, the owner can define some further recommendation policies, e.g. that the system should:

– recommend at most one dish of a given restaurant, encoded by the rule (1):

$$not\, rec(X) \leftarrow restaurant(X,E), restaurant(Y,E), X \neq Y, rec(Y). \quad (1)$$

– non-deterministically, recommend at least one of dishes d_3 and d_4, and one of the wines w_6 and w_7 encoded by the rules (2 - 5)[5]:

$$\begin{array}{ll} rec(d_3) \leftarrow not\, rec(d_4). & rec(w_6) \leftarrow not\, rec(w_7). \\ rec(d_4) \leftarrow not\, rec(d_3). & rec(w_7) \leftarrow not\, rec(w_6). \end{array} \quad (2\text{-}5)$$

– always recommend the wine that goes with a recommended dish, and vice versa, in case such a relation exists (defined by the predicate $rel/2$), encoded by the rules (6,7):

$$\begin{array}{l} rec(Y) \leftarrow type(Y, wine), type(X, dish), rel(X,Y), rec(X). \\ rec(X) \leftarrow type(Y, wine), type(X, dish), rel(X,Y), rec(Y). \end{array} \quad (6\text{-}7)$$

The owner program P_0 contains the previous rules, together with the site's relational database where the relations $type/2$, $rel/2$, etc, are defined. Here, we consider $restaurant(d_2, rest_1)$, $restaurant(d_1, rest_1)$, $restaurant(d_5, rest_1)$, $rel(d_5, w_4)$, $taste(w_7,$ "buttery"$)$, $grapesort(w_4,$ "Chardonnay"$)$, $type(w_i, wine)$ and $type(d_i, dish)$. Without rules specified by the user, the frame has four stable recommendations resulting from the effect of the owner's rules on the initial model[6]:

$$\begin{array}{l} M_{R1} = \{rec(d_1), rec(d_3), rec(w_1), rec(w_2), rec(d_7), rec(w_6)\} \\ M_{R2} = \{rec(d_2), rec(d_3), rec(w_1), rec(w_2), rec(d_7), rec(w_6)\} \\ M_{R3} = \{rec(d_1), rec(d_4), rec(w_1), rec(w_2), rec(d_7), rec(w_6)\} \\ M_{R4} = \{rec(d_2), rec(d_4), rec(w_1), rec(w_2), rec(d_7), rec(w_6)\} \end{array}$$

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

[6] Here, we restrict the models to the propositions of the form $rec/1$.

When taking a closer look at the four stable recommendations, we observe that the first rule specified by the owner removed the recommendation for either d_1 or d_2, and the second and third rules introduced either $rec\,(d_3)$ or $rec\,(d_4)$. The combination of these generated four stable recommendations. Note that the fourth and fifth rules (for products w_6 and w_7) did not cause any change because $rec\,(w_6)$ belonged to the initial recommendation and the semantics tends to keep recommendations by inertia. The system would, e.g. non-deterministically, choose one of these stable recommendations to present to the user thus add diversity to the recommendation system which, otherwise, would only have one set of recommendations to present in case of consecutive equal requests.

Let's now consider the user. Initially, being only looking for dish recommendations, the user states that she doesn't want any recommendations for wines, encoded by the rule (8): $not\,rec\,(X) \leftarrow type\,(X, wine)$ .This rule alone will override all the initial recommendations for wines and also all the rules of the owner specifying wine recommendations such as rules 4 and 5. If $\mathcal{P}_1 = (P_1)$, where P_1 contains rule 8, the four stable recommendations of the frame $\langle M, P_0, \mathcal{P}_1 \rangle$ are:

$$\begin{array}{ll} M_{R5} = \{rec\,(d_1)\,, rec\,(d_3)\,, rec\,(d_7)\} & M_{R7} = \{rec\,(d_1)\,, rec\,(d_4)\,, rec\,(d_7)\} \\ M_{R6} = \{rec\,(d_2)\,, rec\,(d_3)\,, rec\,(d_7)\} & M_{R8} = \{rec\,(d_2)\,, rec\,(d_4)\,, rec\,(d_7)\} \end{array}$$

Later on, the user decides to get some recommendations for wines. For this, she decides to define the concept of what good wines are. Initially, she considers a good wine to be one with the grape "'Chardonnay"' or one with a buttery taste. She writes the following two rules (9 and 10):

$$\begin{array}{l} good\,(X) \leftarrow type\,(X, wine)\,, grapesort\,(X, \text{“Chardonnay”})\,. \\ good\,(X) \leftarrow type\,(X, wine)\,, taste\,(X, \text{“buttery”})\,. \end{array} \qquad (9\text{-}10)$$

Furthermore, she decides that she wants to get at least one recommendation for a good item. She writes the rules (11-14)[7]:

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

If $\mathcal{P}_2 = (P_1, P_2)$, where P_2 contains rule 9-14, the stable recommendations of the frame $\langle M, P_0, \mathcal{P}_2 \rangle$ are:

$$\begin{array}{l} M_{R9} = \{rec\,(w_4)\,, rec\,(d_5)\,, rec\,(d_3)\,, rec\,(d_7)\} \\ M_{R10} = \{rec\,(w_7)\,, rec\,(d_1)\,, rec\,(d_4)\,, rec\,(d_7)\} \\ M_{R11} = \{rec\,(w_7)\,, rec\,(d_2)\,, rec\,(d_4)\,, rec\,(d_7)\} \\ M_{R12} = \{rec\,(w_4)\,, rec\,(w_7)\,, rec\,(d_5)\,, rec\,(d_3)\,, rec\,(d_7)\} \end{array}$$

[7] These rules are a classic construct of ASP. The first two rules state that each good item X is either recommended or *n_rec*. Then, the third rule makes the proposition *rec_at_least_one* true if at least one good item is recommended. The fourth rule, an integrity constraint, eliminates all models where *rec_at_least_one* is not true. The recommendation system would have special syntactical shortcuts for these kind of specifications, since we cannot expect the user to write this kind of rules.

Note that all four sets have at least one recommendation for a good wine. Furthermore, M_{R9} and M_{R12} have, besides the recommendation for w_4, also the recommendation for d_5, as specified by rule 7 since they are related. Also note that rules 11 and 12 cause a direct contradiction with rule 8 for good wines. The semantics based on the causal rejection of rules deals with this issue.

Also worth noting is that, for each recommendation, there is an explanation based on user rules, on owner rules, or on the initial recommendation (as is the case of product d_7). This and other properties are explored in the next Section.

5 Properties

The use of formal semantics enables a rigorous account of its behaviour. In this Section we discuss some properties of the Stable Recommendation Semantics.

We start with conservation, stating that if the initial recommendation is a dynamic stable model of the DLP consisting of the owner rules followed by the user DLP, then it is a stable recommendation. This ensures that the semantics will keep the results of the initial module if they are agreed by owner and user.

Proposition 1 (Conservation). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF*[8]*. Then,*

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

We now establish the relationship between the stable recommendation semantics and DLP for the case of no existing recommendation module, ensuring the transfer of all properties of DLP, and ASP, namely expressiveness results.

Proposition 2 (Generalisation of DLP). *Let P_0 be a generalised logic program and $\mathcal{P}$ a dynamic logic program. Then,*

$$SR\left(\langle \emptyset, \emptyset, \mathcal{P} \rangle\right) = RDSM\left(\mathcal{P}\right) \;\; SR\left(\langle \emptyset, P_0, \mathcal{P} \rangle\right) = RDSM\left((P_0, \mathcal{P})\right)$$

An important issue in recommendation systems is that of explaining the output to the user (and the owner). 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 recommendations. 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, i.e. there is always an explanation.

Proposition 3 (Positive Supportiveness). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF and $A \in M_R \in SR\left(\mathcal{R}\right)$. Then:*

$\exists r \in \rho\left(\mathcal{P}\right) : H\left(r\right) = A, M_R \vDash B\left(r\right)$ *or*
$\exists r' \in P_0 : H\left(r'\right) = A, M_R \vDash B\left(r'\right) \wedge \nexists r \in \rho\left(\mathcal{P}\right) : H\left(r\right) = not\, A, M_R \vDash B\left(r\right)$ *or*
$A \in M \wedge \nexists r \in \rho\left((P_0, \mathcal{P})\right) : H\left(r\right) = not\, A, M_R \vDash B\left(r\right)$

[8] Lack of space prevents us from presenting proofs for the propositions.

Likewise for 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 a rule overriding it (either from the user or from the owner).

Proposition 4 (Negative Supportiveness). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF and $A \notin M_R \in SR(\mathcal{R})$ and $A \in M$. Then,*

$$\exists r \in \rho((P_0, \mathcal{P})) : H(r) = not\, A, M_R \vDash B(r)$$

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 obey 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 5 (Strong Conservation). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then,*

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

Likewise, if we could establish a distance measure between sets of recommendations, we could argue for a semantics that would only keep those stable recommendations that are closer to the output of the existing module, i.e., impose minimal change. If such distance measure is given by:

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*

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

The stable recommendation semantics does not obey minimal change:

Proposition 6 (Minimal Change). *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then,*

$$M_1, M_2 \in RDSM((P_0, \mathcal{P})) \wedge M_1 \sqsubset_M M_2 \not\Rightarrow M_2 \notin SR(\mathcal{R})$$

This is not necessarily a bad property as there are cases where one may argue for stable recommendations that involve non-minimal change i.e., the owner/ user rules introduce other alternatives, even though the initial model is one of them. Consider a very simple example with four products, w_1, w_2, w_3 and w_4 and where the initial set of recommendations is $M = \{rec(w_1), rec(w_3)\}$. The product database contains $property(a, w_2)$ and $property(b, w_1)$ encoding the fact that product w_2 (resp. w_1) has property a (resp b). Let us now consider that the owner of the system specifies the following policy:

$$rec(w_2) \leftarrow rec(w_1). \qquad rec(w_4) \leftarrow rec(w_3), not\, recom\_prop(a).$$
$$recom\_prop(P) \leftarrow rec(W), property(P, W).$$

encoding that if product w_1 is recommended then so should product w_2 and that if product w_3 is being recommended and no product with property a is being recommended, then product w_4 should be recommended. Let us now assume that the user specifies that she doesn't want to receive recommendations for products with property b if no product with property a is being recommended. This can be encoded as: $not\, rec\,(W) \leftarrow not\, recom\_prop\,(a)\,, property(b, W)$.With this scenario, there are two stable recommendations, namely

$$M_1 = \{rec\,(w_1)\,, rec\,(w_2)\,, rec\,(w_3)\} \qquad M_2 = \{rec\,(w_3)\,, rec\,(w_4)\}$$

where $M_1 \sqsubset_M M_2$, but where it may be argued that M_2 should also be accepted since it obeys the specified rules, provides both positive as well as negative explanations, and adds to the diversity of the possible recommendations. However, if desired, such stable recommendations can be eliminated and we can define a refined semantics that obeys such properties. Formally:

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

This new semantics obeys both strong conservation and minimal change. As for the remaining properties, the minimal change stable recommendation semantics obeys conservation and positive and negative supportiveness. As expected, it no longer generalises Dynamic Logic Programming as it is well known that DLP accepts non-minimal dynamic stable models that would be eliminated.

6 Conclusions and Future Work

In most large-scale recommender 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 by means of explicit user modeling. In this paper we proposed a system that can be seen as a complemental module for existing recommender systems, allowing for an improvement of results by means of explicit user modeling. The proposed system can thus 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.

Our future work will concentrate on integrating the module in a trial application in order to investigate its functionality and effectiveness. Such a system should contain a user interface for an easier specification and handling of the rules, as well as a justification option for recommender results. Moreover, a suitable way of connecting to external recommender systems should be included.

Subsequently, 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 been investigated in the context of ASP [12,11], with promising results. We will use such results 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, however, that our proposal already encodes several important concepts that bring an added value to existing recommender systems.

References

1. Alferes, J.J., Banti, F., Brogi, A., Leite, J.A.: The refined extension principle for semantics of dynamic logic programming. Studia Logica 79(1) (2005)
2. Alferes, J.J., Leite, J.A., Pereira, L.M., Przymusinska, H., Przymusinski, T.: Dynamic updates of non-monotonic knowledge bases. Journal of Logic Programming 45(1-3) (2000)
3. Balabanović, M., Shoham, Y.: Fab: content-based, collaborative recommendation. Communications of the ACM 40(3), 66–72 (1997)
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
5. Billsus, D., Pazzani, M.J.: User modeling for adaptive news access. User Model. User-Adapt. Interact 10(2-3), 147–180 (2000)
6. Billsus, D., Pazzani, M.J.: Content-based recommendation systems. In: The Adaptive Web. LNCS, vol. 4321, pp. 325–341. Springer, Heidelberg (2007)
7. Bridge, D., Kelly, J.P.: Diversity-enhanced conversational collaborative recommendations. In: Creaney, N. (ed.) Procs. of the Sixteenth Irish Conference on Artificial Intelligence and Cognitive Science, University of Ulster, pp. 29–38 (2005)
8. Burke, R.D.: Integrating knowledge-based and collaborative-filtering recommender systems. In: AAAI Workshop on AI in Electronic Commerce, pp. 69–72. AAAI, Stanford, California, USA (1999)
9. Burke, R.D.: Knowledge-based recommender systems. In: Kent, A., Dekker, M. (eds.) Encyclopedia of Library and Information Systems, vol. 69, ch. suppl. 32 (2000)
10. Burke, R.D.: Hybrid recommender systems: Survey and experiments. User Model. User-Adapt. Interact 12(4), 331–370 (2002)
11. Eiter, T.: Data integration and answer set programming. In: Baral, C., Greco, G., Leone, N., Terracina, G. (eds.) LPNMR 2005. LNCS (LNAI), vol. 3662, pp. 13–25. Springer, Heidelberg (2005)

12. Eiter, T., Fink, M., Sabbatini, G., Tompits, H.: A generic approach for knowledge-based information-site selection. In: KR, pp. 459–469 (2002)
13. Eiter, T., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. In: KR 2004. Procs. of 9th Int. Conference on Principles of Knowledge Representation and Reasoning, pp. 141–151. AAAI Press, Stanford, California, USA (2004)
14. Felfernig, A., Kiener, A.: Knowledge-based interactive selling of financial services with FSAdvisor, pp. 1475–1482. AAAI, Stanford, California, USA (2005)
15. Gelfond, M., Lifschitz, V.: Logic programs with classical negation. In: Procs. of ICLP 1990, MIT Press, Cambridge (1990)
16. Goldberg, D., Nichols, D., Oki, B.M., Terry, D.: Using collaborative filtering to weave an information tapestry. Communications of the ACM 35(12), 61–70 (1992) Special Issue on Information Filtering
17. Lam, S.K., Frankowski, D., Riedl, J.: Do you trust your recommendations? An exploration of security and privacy issues in recommender systems. In: Procs. of the 2006 Int. Conference on Emerging Trends in Information and Communication Security, Freiburg, Germany, ETRICS (2006)
18. Leite, J.A.: Evolving Knowledge Bases. IOS Press, Amsterdam (2003)
19. Leone, N., Pfeifer, G., Faber, W., Calimeri, F., Dell'Armi, T., Eiter, T., Gottlob, G., Ianni, G., Ielpa, G., Koch, S.P.C., Polleres, A.: The dlv system. In: Flesca, S., Greco, S., Leone, N., Ianni, G. (eds.) JELIA 2002. LNCS (LNAI), vol. 2424, Springer, Heidelberg (2002)
20. Niemelä, I., Simons, P.: Smodels: An implementation of the stable model and well-founded semantics for normal LP. In: Fuhrbach, U., Dix, J., Nerode, A. (eds.) LPNMR 1997. LNCS, vol. 1265, Springer, Heidelberg (1997)
21. Resnick, P., Iacovou, N., Suchak, M., Bergstorm, P., Riedl, J.: GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In: Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work, Chapel Hill, North Carolina, pp. 175–186. ACM Press, New York (1994)
22. Resnick, P., Varian, H.R.: Recommender systems. Communications of the ACM 40(3), 56–58 (1997)
23. Schafer, J.B., Konstan, J.A., Riedl, J.: E-commerce recommendation applications. Data Min. Knowl. Discov. 5(1/2), 115–153 (2001)
24. Schindlauer, R.: Answer-Set Programming for the Semantic Web. Phd thesis, Vienna University of Technology, Austria (December 2006)
25. Smyth, B., Cotter, P.: Personalized electronic program guides for digital TV. AI Magazine 22(2), 89–98 (2001)
26. Working group on Answer-Set Programming. `http://wasp.unime.it`