Dynamic Knowledge Based User Modeling for Recommender Systems

João Leite[1] and **Martina Babini**[2]

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 are programs that help the user in navigating large volumes of information available, attempting to provide a solution to the user's needs or suggesting items that the user may like. To chose or discard existing items, recommender systems adopt many different approaches. Most common techniques used to select the right suggestions are: collaborative filtering (e.g. [11, 9]) 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]) where measures of similarity between item's content are used to find the right selection, employing techniques from the information retrieval field; knowledge-based recommendation (e.g. [14, 16]) where knowledge about the user, the objects, and some distance measures between them, are used to infer the right selections i.e. relationships between users and objects; and, as always, hybrid versions of these (e.g. [5, 15]) 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 [7, 12, 13].

Recommender systems can also be categorised, according to the way they interact with the user [6], 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 such systems, and the accuracy of the user models that are employed by these systems is of key importance to this goal. Such user models are often implicit, inferred from past experience. The use of explicit models of both the products and user has been previously identified as opportune to improve the quality of recommendations, namely addressing the *early rater* and the *sparse ratings* problems experienced by many systems[16].

In this paper we will concentrate on user modeling for recommender systems, with the following in mind:

- 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 for of negation to allow for the specification of both positive as well as negative information.

- 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.

- 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 for users to specify their individual models and preferences, while taking into account the previously mentioned. Concretely, we will adapt the paradigm of *Dynamic Logic Programming* [2, 10, 1] to suit our needs.

Dynamic Logic Programming (DLP) is an extension of Answer-set Programming (ASP) [8] 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[3]. 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 and SMODELS [4]) make it finally possible to

[1] New University of Lisbon, Portugal, email: jleite@di.fct.unl.pt
[2] Università di Bologna, Italy, email: martina.babini@gmail.com

[3] 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 [17].

[4] http://www.dlvsystem.com/ and http://www.tcs.hut.fi/Software/smodels/

investigate some serious applications.

According to *DLP*, the extension of ASP we use, knowledge is given by a series of theories, encoded as generalized logic programs[5] (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.

Dynamic Logic Programming 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 recommending certain products).

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. 2 we briefly recap the notion of Dynamic Logic Programming, establishing the language used throughout. 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 conlude in Sect. 6.

2 Dynamic Logic Programming

For self containment, in this Section, we briefly recap the notion and semantics of Dynamic Logic Programming needed throughout. More motivation regarding all these notions can be found in [2, 10].

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:*

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

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 (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 Dynamic Logic Programming) *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

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

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

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 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. 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.

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.

To formalise this notion, we introduce the concept of Dynamic Recommender Frame defined as follows:

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). 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 user. Here, we will opt for giving a prevailing role to the rules specified by the owner. The rational for this is 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).

We 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), which is then followed by the sequence of specifications provided by the user. And 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 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 store selling books and dvd's, 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 products, 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:

$$M = \{rec(b_1), rec(b_2), rec(d_1), rec(d_2), rec(b_7), rec(d_6)\}$$

where b_i,and d_i are products. The owner can, on top on this, define some further recommendation policies, such as:

- the system should recommend at most one product of a given editor, encoded by the rule (1):

$$not\, rec(X) \leftarrow editor(X, E), editor(Y, E), X \neq Y, rec(Y).$$

- the system should, non-deterministically, recommend at least one of products b_3 and b_4, and one of products d_6 and d_7 encoded by the rules (2 - 5)[6]:

$$rec(b_3) \leftarrow not\, rec(b_4). \qquad rec(b_4) \leftarrow not\, rec(b_3).$$
$$rec(d_6) \leftarrow not\, rec(d_7). \qquad rec(d_7) \leftarrow not\, rec(d_6).$$

- the system should always recommend the film of a recommended book, and vice versa, in case it exists (defined by the predicate $rel/2$), encoded by the rules (6,7):

$$rec(Y) \leftarrow type(Y, dvd), type(X, book), rel(X, Y), rec(X).$$
$$rec(X) \leftarrow type(Y, dvd), type(X, book), rel(X, Y), rec(Y).$$

The owner program P_0 contains the previous rules, together with the store's relational database where the relations $brand/2$, $type/2$, $related/2$, etc, are defined. In this example, we consider the following propositions to be true: $type(b_i, book)$, $type(d_i, dvd)$, $editor(b_1, ed_1)$, $actor(d_4, \text{“Al Pacino”})$, $editor(b_5, ed_1)$,

[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.

$rel\,(b_5, d_4)$, $editor\,(b_2, ed_1)$, $theme\,(d_7,$ "Gangsters"). Without any rules specified by the user, the frame has four stable recommendations resulting from the effect of the owner's rules on top of the initial model, namely[7]:

$$M_{R1} = \{rec\,(b_1), rec\,(b_3), rec\,(d_1), rec\,(d_2), rec\,(b_7), rec\,(d_6)\}$$
$$M_{R2} = \{rec\,(b_2), rec\,(b_3), rec\,(d_1), rec\,(d_2), rec\,(b_7), rec\,(d_6)\}$$
$$M_{R3} = \{rec\,(b_1), rec\,(b_4), rec\,(d_1), rec\,(d_2), rec\,(b_7), rec\,(d_6)\}$$
$$M_{R4} = \{rec\,(b_2), rec\,(b_4), rec\,(d_1), rec\,(d_2), rec\,(b_7), rec\,(d_6)\}$$

When taking a closer look at the four stable recommendations, we can observe that the first rule specified by the owner removed the recommendation for either b_1 or b_2, and the second and third rules introduced either $rec\,(b_3)$ or $rec\,(b_4)$. The combination of these generated four stable recommendations. Note that the fourth and fifth rules (for products d_6 and d_7) did not cause any change because $rec\,(d_6)$ belonged to the initial recommendation and the semantics tends to keep the recommendations by inertia. The recommendation system would, non-deterministically, choose one of these stable recommendations to present to the user.

Let's now consider the user. Initially, being only looking for books, the user states that she doesn't want any recommendations for dvd's, encoded by the rule (8): $not\, rec\,(X) \leftarrow type\,(X, dvd)$ .This rule alone will override all the initial recommendations for dvd's and also all the rules of the owner specifying dvd 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:

$$M_{R5} = \{rec\,(b_1), rec\,(b_3), rec\,(b_7)\}$$
$$M_{R6} = \{rec\,(b_2), rec\,(b_3), rec\,(b_7)\}$$
$$M_{R7} = \{rec\,(b_1), rec\,(b_4), rec\,(b_7)\}$$
$$M_{R8} = \{rec\,(b_2), rec\,(b_4), rec\,(b_7)\}$$

Later on, the user decides to get some recommendations for dvd's. For this, she decides to define the concept of what good dvd's are. Initially, she considers a good dvd to be one with Al Pacino or one about gangsters. She writes the following two rules (9 and 10):

$$good\,(X) \leftarrow type\,(X, dvd), actor\,(X, \text{"Al Pacino"}).$$
$$good\,(X) \leftarrow type\,(X, dvd), theme\,(X, \text{"Gangsters"}).$$

Furthermore, she decides that she wants to get at least one recommendation for a good product. She writes the rules (11-14)[8]:

$$rec\,(X) \leftarrow good\,(X), not\, n\_rec\,(X).$$
$$n\_rec\,(X) \leftarrow good\,(X), not\, rec\,(X).$$
$$rec\_at\_least\_one \leftarrow good\,(X), rec\,(X).$$
$$\leftarrow not\, rec\_at\_least\_one$$

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:

$$M_{R9} = \{rec\,(d_4), rec\,(b_5), rec\,(b_3), rec\,(b_7)\}$$
$$M_{R10} = \{rec\,(d_7), rec\,(b_1), rec\,(b_4), rec\,(b_7)\}$$
$$M_{R11} = \{rec\,(d_7), rec\,(b_2), rec\,(b_4), rec\,(b_7)\}$$
$$M_{R12} = \{rec\,(d_4), rec\,(d_7), rec\,(b_5), rec\,(b_3), rec\,(b_7)\}$$

Note that all four stable recommendations have at least one recommendation for a good dvd. Furthermore, M_{R9} and M_{R12} have, besides the recommendation for d_4, also the recommendation for b_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 dvd's. 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 , or on owner rules if not overridden by those of the user or, if there is nothing overriding it, on the initial recommendation (as is the case of product b_7). This and other properties are explored in the next Section.

5 Properties

In this Section we discuss some properties of the Stable Recommendation Semantics. We start with conservation, stating that if the recommendation of the existing 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.

Proposition 1 (Conservation) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P}\rangle$ be a DRF*[9]*. Then, $M \in RDSM\,((P_0, \mathcal{P})) \Rightarrow SR\,(\mathcal{R}) \supseteq \{M\}$.*

It is also important to establish the relationship between the stable recommendation semantics and DLP for the case where there is no existing recommendation module. This ensures the proper transfer of all properties of DLP, and ASP, to our system.

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

A very important issue in recommendation 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 have overridden it. In other words, there is always an explanation for recommendations.

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

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

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).

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

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

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

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

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

Proposition 5 (Immunity to Tautologies) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF, $E_0, ..., E_s$ sets of tautologies, $\mathcal{E} = (E_1, ..., E_s)$ and $\mathcal{R}' = \langle M, P_0 \cup E_0, \mathcal{P} \cup \mathcal{E} \rangle$. Then, $SR(\mathcal{R}) = SR(\mathcal{R}')$.*

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 6 (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*

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

The stable rec. semantics does not obey minimal change:

Proposition 7 (Minimal Change) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. $M_1, M_2 \in RDSM((P_0, \mathcal{P})) \wedge M_1 \sqsubset_M M_2 \not\Rightarrow M_2 \notin SR(\mathcal{R})$.*

This is not necessarily a bad property as there are cases where one may argue for the validity of such 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. Lack of space prevents us from presenting examples of such. 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 Stable Recommendation Sem) *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 $M_R \in RDSM(\Upsilon(\mathcal{R}))$ 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.*

Regarding this new semantics, we have the following properties:

Proposition 8 (Strong Conservation) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. Then, $M \in RDSM((P_0, \mathcal{P})) \Rightarrow SR^m(\mathcal{R}) = \{M\}$.*

Proposition 9 (Minimal Change) *Let $\mathcal{R} = \langle M, P_0, \mathcal{P} \rangle$ be a DRF. $M_1, M_2 \in RDSM((P_0, \mathcal{P})) \wedge M_1 \sqsubset_M M_2 \Rightarrow M_2 \notin SR^m(\mathcal{R})$.*

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 Dynamic Logic Programming 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 Discussion and Conclusions

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 rules, improving the quality of recommendations;
- allowing for interaction with 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.

Lack of space prevents us from comparing with other, somehow related, systems. An extended version of this paper will include such comparisons. We believe, however, that our proposal encodes some important concepts that can bring an added value to existing systems.

REFERENCES

[1] J. J. Alferes, F. Banti, A. Brogi, and J. Leite, 'The refined extension principle for semantics of dynamic logic programming.', *Studia Logica*, **79**(1), (2005).

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

[3] M. Balabanović and Y. Shoham, 'Fab: content-based, collaborative recommendation', *Communications of the ACM*, **40**(3), 66–72, (1997).

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

[5] D. Billsus and M. J. Pazzani, 'User modeling for adaptive news access', *User Model. User-Adapt. Interact*, **10**(2-3), 147–180, (2000).

[6] D. Bridge and J. P. Kelly, 'Diversity-enhanced conversational collaborative recommendations', in *Procs. of the Sixteenth Irish Conference on Artificial Intelligence and Cognitive Science*, ed., N. Creaney, pp. 29–38. University of Ulster, (2005).

[7] R. D. Burke, 'Hybrid recommender systems: Survey and experiments', *User Model. User-Adapt. Interact*, **12**(4), 331–370, (2002).

[8] M. Gelfond and V. Lifschitz, 'Logic programs with classical negation', in *Procs. of ICLP'90*. MIT Press, (1990).

[9] D. Goldberg, D. Nichols, B. M Oki, and D. Terry, 'Using collaborative filtering to weave an information tapestry', *Communications of the ACM*, **35**(12), 61–70, (December 1992). Special Issue on Information Filtering.

[10] J. Leite, *Evolving Knowledge Bases*, IOS press, 2003.

[11] P. Resnick, N. Iacovou, M. Suchak, P. Bergstorm, and J. Riedl, 'GroupLens: An Open Architecture for Collaborative Filtering of Netnews', in *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, pp. 175–186, Chapel Hill, North Carolina, (1994). ACM.

[12] P. Resnick and H. R. Varian, 'Recommender systems', *Communications of the ACM*, **40**(3), 56–58, (1997).

[13] J. Ben Schafer, J. A. Konstan, and J. Riedl, 'E-commerce recommendation applications', *Data Min. Knowl. Discov*, **5**(1/2), 115–153, (2001).

[14] I. Schwab, A. Kobsa, and I. Koychev. Learning user interests through positive examples using content analysis and collaborative filtering, November 30 2001. Internal Memo, GMD, St. Augustin, Germany.

[15] B. Smyth and P. Cotter, 'Personalized electronic program guides for digital TV', *AI Magazine*, **22**(2), 89–98, (2005).

[16] B. Towle and C. Quinn. Knowledge based recommender systems using explicit user models, May 29 2000.

[17] Working group on Answer-Set Programming. http://wasp.unime.it.