Using Defeasible Logic Programming with Contextual Queries for Developing Recommender Servers

Mariano Tucat and **Alejandro J. García** and **Guillermo R. Simari**
Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering (DCIC)
Universidad Nacional del Sur, Bahía Blanca, Argentina
{mt, ajg, grs}@cs.uns.edu.ar

Abstract

In this work we introduce a defeasible logic programming recommender server that accepts different types of queries from client agents that can be distributed in remote hosts. We formalize new ways of querying recommender servers containing specific information or preferences, and creating a particular *context* for the queries. This special type of queries (called *contextual queries*) allows recommender servers to compute recommendations for any client using its preferences, and will be answered using an argumentative inference mechanism. We focus on a particular implementation of recommended systems that extends the integration of argumentation and recommender systems to a multi-agent setting. Our approach is based on a DeLP-server that can answer queries from agents in remote hosts. Since client agents can consult different domain specific recommender servers, then, multiple configurations of clients and servers can be defined.

Introduction

Recommender systems have become an important research area in AI over the last decade, and the demand for new approaches to intelligent product recommendation is increasing. In the last years, approaches that use argumentation for recommender systems applications have been proposed (Chesñevar, Maguitman, and Simari 2007; 2006). There, the integration of argumentation and recommender systems has been established and formalized, and in some of them Defeasible Logic Programming (DeLP) was proposed for knowledge representation.

In this paper we focus on a particular implementation of recommender systems called *Recommender Servers* that extends the integration of argumentation and recommender systems to a multi-agent setting. In our proposal, Recommender Servers are based on a new and more flexible implementation of DeLP called DeLP-Server (García et al. 2007). A DeLP-server can answer multiple queries received from several client agents that can be distributed in remote hosts; and each client agent can consult different domain specific DeLP-servers. Therefore, multiple configurations of clients and servers can be defined. To answer client queries, a DeLP-server uses public knowledge (stored in the proper server) together with individual knowledge that clients can send along with the query, creating a particular *context* for the query.

In this paper, we define different types of *Contextual Queries* that a Recommend Server can process. Contextual queries will allow client agents to inquire recommender servers sending individual knowledge that can represent preferences or private information. This contextual information will be used by Recommender Servers for generating arguments, and to obtain a warranted recommendation. As we will show below, since the context sent with a query represents private information, it can not be used by other agents.

Recommender Systems usually operate by creating a model of the users' preferences in order to guide them in a personalized way to interesting or useful objects in a large space of possible options (Resnick and Varian 1997; Konstan 2004). Many recommender systems attempt to anticipate the user's needs and are capable of providing assistance proactively, while in many situations the user explicitly posts a request for recommendations in the form of a query. A contextual query represents an alternative way a user, or an agent on behalf of a user, can use in order to request a recommendation.

A DeLP based Recommender Server that accepts different types of contextual queries, represents one alternative way of developing Recommender Systems using Content Based recommendation. In particular, we are interested in defining different alternatives for querying a recommender server in order to obtain the desired information. Considering that recommender systems are a special class of user support tools which particularly focus on user-dependent filtering and selection of relevant information, our contextual queries represent one alternative for specifying user preferences and asking recommender servers for specific information.

Thus, the contribution of this paper is to define Recommender Servers that extends argumentative recommender systems for multi-agent environments, and to define new types of contextual queries that can be used by this new type of servers for answering client agents that need a recommendation. The rest of the paper is structured as follows. The following section gives some preliminaries and introduces a working example that will be used in the rest of the paper. Then we formalize the new types of contextual queries, and

$recommend(X) -\!\!\prec available(X), match\_features(X), match\_location(X).$
$match\_features(X) -\!\!\prec large\_liv\_room, has\_large\_liv\_room(X).$
$match\_features(X) -\!\!\prec two\_bedrooms, has\_two\_bedrooms(X).$
$match\_location(X) -\!\!\prec \sim prefer\_suburbs(X), downtown(X).$
$match\_location(X) -\!\!\prec prefer\_suburbs(X), suburbs(X).$

$ap(ap1, available, suburbs, [has\_two\_bedrooms, cheap, subway, ...])$
$ap(ap2, available, downtown, [has\_two\_bedrooms, \sim subway, ...])$
$ap(ap3, available, suburbs, [large\_liv\_room, cheap, \sim subway, ...])$

$prefer\_suburbs(X) -\!\!\prec car.$
$prefer\_suburbs(X) -\!\!\prec \sim car, subway(X).$
$\sim prefer\_suburbs(\_) -\!\!\prec \sim car.$
$large\_liv\_room -\!\!\prec young\_couple.$
$\sim large\_liv\_room -\!\!\prec two\_bedrooms.$
$two\_bedrooms -\!\!\prec young\_couple, children.$
$two\_bedrooms -\!\!\prec young\_couple, professional.$

Figure 1: Part of the DeLP-program of the *re-server* of Example 1

introduce contextual interrogations. Finally, some conclusions and related work are presented.

DeLP Server

A Defeasible Logic Programming Server (DeLP-server) answers queries received from client agents that can be distributed in remote hosts. Public knowledge can be stored in the DeLP-server represented as a Defeasible Logic Program. To answer queries, the DeLP-server uses this public knowledge together with individual knowledge that clients might send, creating a particular *context* for the query. These *contextual queries* are answered using the defeasible argumentative analysis of DeLP (García and Simari 2004). Several DeLP-servers can be used simultaneously, each of them providing a different shared knowledge base. Thus, several agents can consult the same DeLP-server, and the same agent can consult several DeLP-servers. This approach does not impose any restriction over the type, architecture, or implementation language of the client agents.

In this model, both public knowledge stored in the server and contextual knowledge sent by the agents are used for answering queries, however, no permanent changes are made to the stored program. The temporal scope of the contextual information sent in a query is limited and it will disappear with the finalization of the process performed by the DeLP-server to answer that query.

In DeLP, knowledge is represented using facts, strict rules or defeasible rules. *Facts* are ground literals representing atomic information or the negation of atomic information using the strong negation "$\sim$". *Defeasible Rules* are denoted $L_0 -\!\!\prec L_1, \ldots, L_n$ and represent defeasible knowledge, *i.e.*, tentative information, where the *head* L_0 is a literal and the *body* $\{L_i\}_{i>0}$ is a set of literals. DeLP-servers consider a restricted form of programs that do not have strict rules. A restricted DeLP-program (*de.l.p.* for short) will be denoted (Π,Δ), distinguishing the set of facts Π (that must be non-contradictory), and the set of defeasible rules Δ.

Example 1. Consider a DeLP-server that implements a "Real Estate" recommender server (*re-server*). This *re-server* will have a database of apartments for rent (set of facts $ap/4$) and, using contextual information that it will receive from client agents, it will answer queries for recommending clients which apartment to rent. The *re-server* will have knowledge represented with defeasible rules that will be used for building arguments for and against each apartment. Figure 1 shows part of the DeLP-program that can be stored in the *re-server*. For instance, the defeasible rule $large\_liv\_room -\!\!\prec young\_couple$ represents the fact that usually young couples prefer apartments with large living rooms, and this rule may be used for building an argument for recommending a young couple a particular apartment (*e.g.*, $ap3$). However, if the recommendation is for a young couple that has children, then rule $two\_bedrooms -\!\!\prec young\_couple, children$ may be used for building an argument for recommending an apartment with two bedrooms (*e.g.*, , $ap1$); and since normally apartments with two bedrooms do not have large living rooms ($\sim large\_liv\_room -\!\!\prec two\_bedrooms$), then an argument against recommending "$ap3$" can be build. As it will be clear below, the *re-server* will recommend an apartment ap if a warrant for the literal $recommend(ap)$ exists, using both the program stored in the DeLP-server and the context sent by the client agent.

In DeLP a query Q is *warranted* from a program $\mathcal{P}$ if a *non-defeated* argument $\langle \mathcal{A}, Q \rangle$ supporting Q exists. To establish whether an argument $\langle \mathcal{A}, Q \rangle$ is a non-defeated argument, *defeaters* for $\langle \mathcal{A}, Q \rangle$ are considered, *i.e.*, counter-arguments that by some criterion are preferred to $\langle \mathcal{A}, Q \rangle$. Note that in DeLP the argument comparison criterion is modular and thus, the most appropriate criterion for the domain that is being represented can be selected. In the examples in this paper we will use *generalized specificity* (Stolzenburg et al. 2003). For a detailed presentation of DeLP see (García and Simari 2004).

In (García et al. 2007), several *contextual queries* were defined. These types of queries allow the inclusion of private pieces of information related to the agent's particular context to be taken into consideration at the moment of computing the answers. There, a *Combined Contextual Query* includes all the other contextual queries defined in (García et al. 2007), and allows the client agent to assign priority either to the knowledge at the server or to the one sent by it as context. In the next section we will define a new type of contextual query called *Regular Contextual Query* that is a simplified version of the Combined Contextual Query (defined in (García et al. 2007)) and assigns priority to the knowledge (context) sent by the agent.

Contextual Queries

In this section we propose a particular client/server interaction that allows client agents to interact with a recommender server sending queries with contextual information. Our goal is to allow the clients to provide specific contextual information to be used in the resolution of the queries, and also to efficiently cover different ways of querying the server. Therefore, we will introduce different types of contextual queries.

As will be shown below, each type of query provides a different kind of interaction. However, they all have in common that although both, the knowledge stored in the server and the contextual knowledge sent by the agents, are used for answering queries, no permanent changes are made to the stored program. That is, the temporal scope of the contextual information sent in a query is limited and it will disappear with the finalization of the process performed by the server to answer that query.

The simplest type, called *regular contextual query*, will be introduced first. Using a regular contextual query a client agent sends a server a query, a set of facts and a set of defeasible rules representing specific information the agent wants to add for the computation of the answer by the server. The definition of regular contextual query follows:

Definition 1 (Regular Contextual Query)**.** A regular contextual query for a *de.l.p.* (Π_S,Δ_S) is a triple [Π^+,Δ,Q] where the pair (Π^+,Δ) is a *de.l.p.* and Q is a DeLP-query.

As already mentioned, the regular contextual query assigns priority to the knowledge sent by the agent. Therefore, following the terminology used in (García et al. 2007), and as can be seen in the previous definition, we use the superscript $+$ to denote that the set of facts Π^+ has priority over the information stored in the server (see Definition 2).

Example 2. For instance, consider the *re-server* introduced in Example 1 and suppose a client agent representing a young couple wants to obtain a recommendation of whether an apartment $ap1$ fulfils their requirements. Considering that they are both professionals and prefer an apartment far from downtown only if it is cheap, the agent may submit the following regular contextual query Cq_1 = [Π_1^+,Δ_1,Q_1], where

$\Pi_1^+ = \{young\_couple, professionals\}$
$\Delta_1 = \{prefer\_suburbs(A) \prec cheap(A)\}$
$Q_1 = recommend(ap1)$.

The context of Cq_1 contains a set of facts Π_1^+ representing the client information: *young couple* and *professionals*. Note that Cq_1 also involves the set Δ_1 that contains a defeasible rule representing the fact that the client prefers an apartment in the suburbs only if it is cheap.

Example 3. Consider again the *re-server* introduced in Example 1 and suppose that a different young couple with one child and having no car want to know whether the *re-server* recommends the apartments $ap1$ and $ap3$. In this case, the agent representing them may submit two regular contextual queries, Cq_2 = [Π_2^+,Δ_2,Q_2] and Cq_3 = [Π_2^+,Δ_2,Q_3], where

$\Pi_2^+ = \{young\_couple, children, \sim car\}$
$\Delta_2 = \{\}$
$Q_2 = recommend(ap1)$.
$Q_3 = recommend(ap3)$.

For answering a regular contextual query [Π^+,Δ,Q] the server will try to warrant the literal Q using both the stored program and the received context. The formal definitions follow:

Definition 2 (Warrant for a Regular Contextual Query)**.** The regular contextual query [Π^+,Δ,Q] is warranted from (Π_S,Δ_S) if Q is warranted from ($\Pi_S \oplus \Pi^+$,$\Delta_S \cup \Delta$). Let Co(Π^+) = {$\overline{L}$ if $L \in \Pi^+$}, then ($\Pi_S \oplus \Pi^+$)= ($\Pi_S \setminus$ Co(Π^+)) $\cup\ \Pi^+$.

In Definition 2, the operator $\oplus$ is used for obtaining a non-contradictory set of facts preferring the information received in the contextual query. In (García et al. 2007), different operators are defined (prioritized, non-prioritized and restrictive), however, in this work we will use only the one that assigns priority to the information send by the agent. Using other operators is subject of future work.

Definition 3 (Answer for a Regular Contextual Query)**.** The answer for [Π^+,Δ,Q] from $\mathcal{P}$=(Π_S,Δ_S) can be YES, NO, UNDECIDED or UNKNOWN. In the case that [Π^+,Δ,Q] is warranted from $\mathcal{P}$, then the answer is YES, if [Π^+,Δ,$\overline{Q}$] is warranted from $\mathcal{P}$, then the answer is NO, if neither [Π^+,Δ,Q] nor [Π^+,Δ,$\overline{Q}$] are warranted from $\mathcal{P}$ then the answer is UNDECIDED. Finally, if Q is not in the language of ($\Pi_S \oplus \Pi^+$,$\Delta_S \cup \Delta$), then the answer is UNKNOWN.

Example 4. Consider the *re-server* introduced in Example 1. The answer for the query Cq_1 of Example 2 is YES. In the case of Example 3, the answer of the *re-server* for the query Cq_2 is YES, whereas the answer for Cq_3 is NO.

As already mentioned, for computing the answer of a regular contextual query [Π^+,Δ,Q] the context (Π^+,Δ) temporarily modifies the program stored in the server. Note that this modification affects neither the answer of other regular contextual queries that may be processed in parallel with the computation of [Π^+,Δ,Q] nor other queries that the server may receive later (from the same or a different agent).

In this model, a client can execute several queries with different contexts, each one through a specific regular contextual query. Nevertheless, this alternative will require the exchange of a great number of messages and could be prohibitive in some domains. Therefore, we will define a new type of query that can contain a sequence of regular contextual queries. Although the effect of this new query will be the same compared with the execution of a sequence of individual regular contextual queries, the advantage of this new query is an efficient use of the message exchange infrastructure. A multiple contextual query is defined as an extended version of a regular contextual query.

Definition 4 (Multiple Contextual Query)**.** A multiple contextual query for a *de.l.p.* $\mathcal{P}$ is the sequence

$\langle[\Pi_1^+,\Delta_1,Q_1], [\Pi_2^+,\Delta_2,Q_2], \ldots, [\Pi_n^+,\Delta_n,Q_n]\rangle$, in which each $[\Pi_i^+,\Delta_i,Q_i]$, $1 \leq i \leq n$, is a regular contextual query.

Example 5. Following with the *re-server* example, a client agent may want to know whether this *re-server* recommends the apartments $ap1$ and $ap3$, considering that it acts on behalf of a *young couple* of *professionals* having *no car*. Since they are planning to buy a *car*, they may also be interested in the recommendation of the *re-server* in the case they have a car. In this case, they will also prefer an apartment in the suburbs only if it is cheap. Thus, this client agent may ask the following multiple contextual query: $Cq_4 = \langle[\Pi_4^+,\Delta_4,Q_4], [\Pi_5^+,\Delta_5,Q_5], [\Pi_6^+,\Delta_6,Q_6], [\Pi_7^+,\Delta_7,Q_7]\rangle$, where

$\Pi_4^+ = \{young\_couple, professionals, {\sim}car\}$
$\Delta_4 = \{\}$
$Q_4 = recommend(ap1)$.

$\Pi_5^+ = \{young\_couple, professionals, {\sim}car\}$
$\Delta_5 = \{\}$
$Q_5 = recommend(ap3)$.

$\Pi_6^+ = \{young\_couple, professionals, car\}$
$\Delta_6 = \{prefer\_suburbs(Ap) \prec cheap(Ap)\}$
$Q_6 = recommend(ap1)$.

$\Pi_7^+ = \{young\_couple, professionals, car\}$
$\Delta_7 = \{prefer\_suburbs(Ap) \prec cheap(Ap)\}$
$Q_7 = recommend(ap3)$.

The answer for a multiple contextual query is a sequence of answers, each of them corresponding to a *Contextual Query* in the sequence.

Definition 5 (Answer for a Multiple Contextual Query)**.** An answer for a multiple contextual query $\langle[\Pi_1^+,\Delta_1,Q_1], [\Pi_2^+,\Delta_2,Q_2], \ldots, [\Pi_n^+,\Delta_n,Q_n]\rangle$ from a *de.l.p.* $\mathcal{P}$ is a sequence $\langle Ans_1, Ans_2, \ldots, Ans_n\rangle$, where each Ans_i, $1 \leq i \leq n$, is the answer for the corresponding regular contextual query $[\Pi_i^+,\Delta_i,Q_i]$ of the sequence.

Example 6. Consider the *re-server* introduced in Example 1. Then, the answer for the query Cq_4 of Example 5 is $\langle$YES,NO,YES,NO$\rangle$

It is important to mention that the order in which the queries are solved by the server does not affect the answers obtained. That is, the queries are independent of each other, since the modifications made to the server program to take into account the context of each query do not affect subsequent queries. Therefore, each individual regular contextual query may be executed in parallel.

Clearly, only two messages are exchanged and the computation is done in the server side. In the particular case that a client has to execute various queries with the same context, it may execute several regular contextual queries, or it may execute a multiple contextual query, repeating the context. Observe that this alternative will require the same repeated modifications of the server program. Therefore, we define a new query (called contextual query sequence), as a particular case of the multiple contextual query, consisting of a sequence of queries with only one context.

Definition 6 (Contextual Query Sequence)**.** A contextual query sequence for a *de.l.p.* $\mathcal{P}$ is a triple $[\Pi^+,\Delta,Qs]$ where the pair (Π^+,Δ) is a *de.l.p.* and Qs is a sequence $\langle Q_1, Q_2, \ldots, Q_n\rangle$ in which each Q_i, $1 \leq i \leq n$, is a DeLP-query.

Example 7. Consider again the *re-server* example. In this case, the client agent may need to know whether the *re-server* recommends the apartments `ap1`, `ap2` and `ap3`, considering that it represents a *young couple* of *professionals*, and they prefer an apartment in the suburbs only if it is cheap. Therefore, this client agent may ask the contextual query sequence $Cq_5 = [\Pi_8^+,\Delta_8,Qs]$ where:

$\Pi_8^+ = \{young\_couple, professionals, {\sim}car\}$
$\Delta_8 = \{prefer\_suburbs(A) \prec cheap(A)\}$
$Qs = \langle recommend(ap1),$
$recommend(ap2),$
$recommend(ap3)\rangle$.

In this new query, the modifications needed to consider the context of a contextual query sequence can be made only once to the program at the server, and then, these changes are undone after all the queries of the sequence have been answered. Therefore, this new definition allows an efficient implementation of a server answering queries with equal contexts, avoiding not only an unnecessary overload of the message exchange infrastructure, but also preventing a reiterative modification of the server.

The answer for a contextual query sequence will include a sequence of answers, each of them corresponding to a query in the sequence of queries. Each answer will be given by the server using its program and considering the modifications needed to take the context included in the contextual query sequence into account.

Definition 7 (Answer for a Contextual Query Sequence)**.** Let $[\Pi^+,\Delta,Qs]$ be a contextual query sequence where Q_s is the sequence $\langle Q_1, Q_2, \ldots, Q_n\rangle$. An answer for $[\Pi^+,\Delta,Qs]$ from a *de.l.p.* $\mathcal{P}$ is a sequence $\langle Ans_1, Ans_2, \ldots, Ans_n\rangle$, where each Ans_i $1 \leq i \leq n$ is the answer of the regular contextual query $[\Pi^+,\Delta,Q_i]$ from $\mathcal{P}$.

Example 8. Consider the *re-server* introduced in Example 1. Then, the answer for the contextual query sequence Cq_5 of Example 7 is $\langle$YES,YES,NO$\rangle$

Depending on the implementation, the sequence of answers generated by the server will be buffered at the server side and sent to the inquirer in only one message, or they may be sent as soon as they are obtained, requiring the exchange of one message per query. There exists a trade off between the number of messages exchanged and response time.

So far, we have defined three different kinds of queries. The first and simpler one, the regular contextual query allows the clients to execute one query with a specific context. In the case of the multiple contextual query, it allows the execution of several regular contextual queries grouped in just one message, providing an efficient use of the network. Therefore, this type of query provides the client the alternative of presenting multiple scenarios in the same query, obtaining specific answers for all of them. Finally, the contextual query sequence allows an efficient execution of several queries with the same context, since the server is only modified once to answer all the queries.

Contextual Interrogation

There exist situations in which a client needs to execute several queries with different contexts (*e.g.*, $[\Pi_a^+,\Delta_a,Q_a]$, $[\Pi_b^+,\Delta_b,Q_b]$, $[\Pi_c^+,\Delta_c,Q_c]$). In order to reduce message exchange, it may use the multiple contextual query $\langle[\Pi_a^+,\Delta_a,Q_a],[\Pi_b^+,\Delta_b,Q_b],[\Pi_c^+,\Delta_c,Q_c]\rangle$. Nevertheless, if the contexts of the queries overlap almost entirely, the client has to repeat great part of them and this alternative may turn into an inefficient one. For example if Π_a^+ has 50 facts and $\Pi_b^+ = \Pi_a^+ \cup \{f_{51}\}$ then the set Π_a^+ is sent twice and also the server has to add and remove the same 50 facts twice. In order to both, reduce traffic congestion and server processing, we will introduce the concept of *contextual interrogation* where the context modifications made at the server by the queries may hold for subsequent queries of the same interrogation. Using this new type of query, in our previous example, Π_b^+ may be just $\{f_{51}\}$, avoiding the repeated transmission and modification at the server required by Π_a^+.

Next, we define a contextual interrogation as a sequence of special regular contextual queries. As it will become clear below, this new kind of query is a generalization of the three ones defined in the previous section and it will allow incremental context modifications.

Definition 8 (contextual interrogation)**.** A contextual interrogation for a *de.l.p.* $\mathcal{P}$ is the sequence $\langle Cq_1, Cq_2, \ldots, Cq_n\rangle$, in which each Cq_i ($1 \leq i \leq n$) is one of:

- Starting Contextual Query, noted $[\Pi_i^+,\Delta_i,Q_i)$
- Continued Contextual Query, noted (Π_i^+,Δ_i,Q_i)
- Closing Contextual Query, noted $(\Pi_i^+,\Delta_i,Q_i]$
- Regular Contextual Query, noted $[\Pi_i^+,\Delta_i,Q_i]$

Note that in Definition 8 the symbols " [", "] ", " (" and ") " specify how the server has to process each part of the contextual interrogation. The symbol ")" specifies that changes made to the program at the server will be kept for subsequent queries from the same contextual interrogation, whereas the symbol "]" means that all the temporary changes that were made by queries in the same contextual interrogation will not hold for subsequent queries. The symbol "[" specifies that previous changes will not be considered (*i.e.*, the query will behave as if it were the first in the contextual interrogation).

In a contextual interrogation, a starting contextual query $[\Pi_i^+,\Delta_i,Q_i)$ will be answered by the server using the original program, however the modifications made to the program at the server will be kept for subsequent queries from the same contextual interrogation. The continued contextual query (Π_i^+,Δ_i,Q_i) is answered by the server using the program already modified by previous queries. In the case of the closing contextual query $(\Pi_i^+,\Delta_i,Q_i]$, it allows the client to obtain an answer considering the program already modified, but returning the program at the server to its original state after the query have been answered. Note that, by original state we mean the state of the program before it was first modified by the contextual interrogation. Finally, a regular contextual query $[\Pi_i^+,\Delta_i,Q_i]$ will be answered by the server using the original program and it will return the program at the server to its original state once the query have been answered.

Example 9. Considering our *re-server* example, a client may ask the following contextual interrogation: $Cq_6 = \langle[\Pi_9^+,\Delta_9,Q_9),(\Pi_{10}^+,\Delta_{10},Q_{10}),(\Pi_{11}^+,\Delta_{11},Q_{11}),(\Pi_{12}^+,\Delta_{12},Q_{12}]\rangle$, where

$\Pi_9^+ = \{young\_couple, professionals, \sim car\}$
$\Delta_9 = \{\}$
$Q_9 = recommend(ap1)$.

$\Pi_{10}^+ = \{\}$
$\Delta_{10} = \{\}$
$Q_{10} = recommend(ap3)$.

$\Pi_{11}^+ = \{car\}$
$\Delta_{11} = \{prefer\_suburbs(A) \prec cheap(A)\}$
$Q_{11} = recommend(ap1)$.

$\Pi_{12}^+ = \{\}$
$\Delta_{12} = \{\}$
$Q_{12} = recommend(ap3)$

This example shows another alternative for Cq_4 of example 5. However, this alternative avoids the repeated transmission of the facts $young\_couple$ and $professionals$ and their modification at the server.

The answer for a Contextual Interrogation is a sequence of answers for each specific regular contextual query, using not only the context given with the query, but also, depending on the type of query and the context of previous queries.

Definition 9 (Answer for a Contextual Interrogation)**.** An answer for a contextual interrogation $\langle Cq_1, Cq_2, \ldots, Cq_n\rangle$ from a *de.l.p.* $\mathcal{P}=(\Pi_S,\Delta_S)$ is a sequence of answers $\langle Ans_1, Ans_2, \ldots, Ans_n\rangle$. Each answer Ans_i will depend on both the type of Cq_i and those changes that have been accumulated by previous queries in the same interrogation. Here follows how Ans_i is computed for each type of query and also how this query will affect the accumulated program $\mathcal{P}_i$ for subsequent queries in the same interrogation. Observe that $\mathcal{P}_{i-1} = (\Pi_{i-1}^a, \Delta_{i-1}^a)$ is the *de.l.p.* with the contextual modifications accumulated for answering a previous query Cq_{i-1}.

- If $Cq_i = [\Pi_i^+,\Delta_i,Q_i)$ then
  Ans_i is the answer for $[\Pi_i^+,\Delta_i,Q_i]$ in $\mathcal{P}$,
  and $\mathcal{P}_i = (\Pi \oplus \Pi_i^+, \Delta \cup \Delta_i)$
- if $Cq_i = (\Pi_i^+,\Delta_i,Q_i)$ then
  Ans_i is the answer for $[\Pi_i^+,\Delta_i,Q_i]$ in $\mathcal{P}_{i-1}$,
  and $\mathcal{P}_i = (\Pi_{i-1}^a \oplus \Pi_i^+, \Delta_{i-1}^a \cup \Delta_i)$
- if $Cq_i = (\Pi_i^+,\Delta_i,Q_i]$ then
  Ans_i is the answer for $[\Pi_i^+,\Delta_i,Q_i]$ in $\mathcal{P}_{i-1}$,
  and $\mathcal{P}_i = \mathcal{P}$.
- if $Cq_i = [\Pi_i^+,\Delta_i,Q_i]$ then
  Ans_i is the answer for $[\Pi_i^+,\Delta_i,Q_i]$ in $\mathcal{P}$,
  and $\mathcal{P}_i = \mathcal{P}$.

Example 10. Consider the *re-server* introduced in Example 1. The answer for the contextual interrogation Cq_6 of Example 9 is $\langle$YES,NO,YES,NO$\rangle$. In this example, the first query of the sequence, a Starting Contextual Query, adds the fact $\sim car$. Then, the following query, a Continued Contextual Query, just queries whether the *re-server* recommends $ap3$, using the program at the server with the modifications made to answer the previous query. Next, another Continued Contextual Query sends the fact car as part of the context. Since the fact $\sim car$ is still part of the program at the server (for the current Contextual Interrogation), the addition of the fact car will turn our set of facts in a contradictory set. In this case, the operator $\oplus$ removes the fact at the server ($\sim car$), since it prefers the information sent as part of the context. Finally, the last query of the sequence, a Closing Contextual Query, queries whether the *re-server* recommends $ap3$.

This latter type of query includes all the queries defined in previous section and it also provides new alternatives for querying the servers. There exist several ways of implementing these contextual queries, and the different behaviors and opportunities that these queries provide will depend on the way in which they are implemented. However, the semantics of the queries will remain the same as well as their corresponding answers.

Conclusions and Related Work

In this work, we propose a particular implementation of recommender systems, called Recommender Servers, that uses argumentation for supporting its recommendations, and can be used in multi-agent scenarios. A Recommender Server (*re-server*) allows several client agents to ask for recommendations, and a client agent can consult several *re-server*s. Client agents can be implemented in any programming language.

In order to generate a warranted recommendation, a *re-server* computes arguments using both the knowledge stored in the server and also private knowledge that client agents can send as context with the queries. The knowledge stored in the server and the private knowledge sent by agents are restricted DeLP-programs, that consist of facts and defeasible rules. The way that the server integrates its proper knowledge with the information provided by client agents was defined as different types of contextual queries. These different types of contextual queries provide a declarative way of representing users' preferences.

We have defined four different kinds of queries. The first one, the regular contextual query allows the clients to execute one query with a specific context. Then, the multiple contextual query allows the execution of several regular contextual queries grouped in just one message, providing an efficient use of the network. In the case of the contextual query sequence, it allows an efficient execution of several queries with the same context, since the server is only modified once to answer all the queries. Finally, in order to reduce traffic congestion and server processing, we introduced the concept of *contextual interrogation* where the context modifications made at the server by the queries may hold for subsequent queries of the same interrogation.

In (Chesñevar, Maguitman, and Simari 2007) and (Chesñevar, Maguitman, and Simari 2006) the integration of argumentation and recommender systems has been established and formalized. There, DeLP was proposed for knowledge representation, and examples for different domains applications were shown. As in those frameworks, we use arguments and a dialectical analysis for warranting recommendations, however, in contrast with them, we focus on a particular framework that extends the integration of argumentation and recommender systems to a multi-agent setting. Since in our proposal, Recommender Servers are based on a new and more flexible implementation of DeLP called DeLP-Server (García et al. 2007), a Recommender Server can answer multiple contextual queries from agents that can be distributed in remote hosts; and each client agent can consult different domain specific DeLP-servers. Therefore, multiple configurations of clients and servers can be defined.

References

Chesñevar, C. I.; Maguitman, A. G.; and Simari, G. R. 2006. Argument-Based Critics and Recommenders: A Qualitative Perspective on User Support Systems. *Journal of Data and Knowledge Engineering* 59(2):293–319.

Chesñevar, C.; Maguitman, A.; and Simari, G. 2007. *Emerging Artificial Intelligence Applications in Computer Engineering*, volume 160 of *Frontiers in Artificial Intelligence and Applications*. IOS Press (Amsterdam, Netherlands). chapter Recommender Systems based on Argumentation, 53–70.

García, A., and Simari, G. 2004. Defeasible logic programming: An argumentative approach. *Theory Practice of Logic Programming* 4(1):95–138.

García, A.; Rotstein, N.; Tucat, M.; and Simari, G. 2007. An argumentative reasoning service for deliberative agents. In Zhang, Z., and Siekmann, J., eds., *LNAI 4798 Proceedings of the 2nd. International Conference on Knowledge Science, Engineering and Management (KSEM 2007)*, 128–139. Springer-Verlag.

Konstan, J. A. 2004. Introduction to recommender systems: Algorithms and evaluation. *ACM Trans. Inf. Syst* 22(1):1–4.

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

Stolzenburg, F.; García, A. J.; Chesñevar, C. I.; and Simari, G. R. 2003. Computing generalized specificity. *Journal of Applied Non-Classical Logics* 13(1):87–.