# An Approach to Argumentative Reasoning Servers with Conditions based Preference Criteria

Juan Carlos Teze[1,2,3], Sebastián Gottifredi[1,3],
Alejandro J. García[1,3] and Guillermo R. Simari[1]

[1]Artificial Intelligence Research and Development Laboratory (LIDIA)
Department of Computer Science and Engineering (DCIC)
Universidad Nacional del Sur (UNS) - Alem 1253
(8000) Bahía Blanca, Buenos Aires, Argentina

[2]Agents and Intelligent Systems Area, Fac. of Management Sciences,
Universidad Nacional de Entre Ríos (UNER)
(3200) Concordia, Entre Ríos, Argentina

[3]Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
e-mail:{jct,sg,ajg,grs}@cs.uns.edu.ar

**Abstract.** Argumentation is a reasoning mechanism attractive due to its dialectical and non monotonic nature, and its properties of computational tractability. In dynamic domains where the agents deal with incomplete and contradictory information, an argument comparison criterion can be used to determine the accepted information. Argumentation systems with a single argument comparison criterion have been widely investigated; in some of these approaches, the comparison criterion is fixed while in others a criterion can be selected and replaced in a modular way. We present an argumentative server providing client agents with recommendations and giving the possibility of specifying which of the available argument comparison criteria will be used to answer a query; for that, we formalize a special type of contextual query which by the use of conditions allows the server to dynamically change the criterion providing a declarative way of representing users preferences.

## 1 Introduction

A defeasible argumentation system provides ways of confronting contradictory statements to determine whether some particular information can be accepted or warranted [9,1,6,7]. To obtain an answer, an argumentation process goes through a series of steps. A very important one is the comparison of conflicting arguments to decide which one prevails. The definition of a formal comparison criterion thus becomes a central problem in defeasible argumentation.

Argumentation systems using a single argument comparison criterion have been widely studied in the literature [13,16,18,3,17,9]. The comparison criterion represents a fundamental part of an argumentation system because the inferences

an agent can obtain from its knowledge will depend on the criterion being used. In some of these approaches, the comparison criterion is fixed while in others a criterion can be selected and replaced in a modular way. The main contribution is to provide a more declarative way of representing users preferences by means of a framework where several comparison criteria can be taken into account and the selection of one depend on the specific conditions that are satisfied. Next, we present an example that will serve two purposes: to motivate the main ideas of our proposal, and to serve as a running example.

**Example 1** *Consider an on_board_computer making recommendations to the user of the vehicle in which is installed, like suggesting a hotel. To give advice, the computer uses two types of knowledge: the user's particular preferences which are obtained before starting the travel; and particular information about the user's context, obtained dynamically during the travel. Besides this knowledge, the computer also needs certain criteria for making decisions to give recommendations. The computer could use some of the following criteria: security and comfort. However, the driver can restrict the criteria that the computer may use by means of conditions; for instance, before starting a journey the driver may tell the computer that if the road is blocked by striking workers, the computer has to consider the security criteria. In this situation, the restrictions are defined over the possible existence of specific knowledge stored in the computer and, depending on the chosen criteria, the user might receive contradictory recommendations.*

Recently, there have been important developments in AI regarding contextual and conditional preferences [5,2]. A particularly active area is focused on the association of conditions to users preferences [4,12]. In [4], a conditional preference network (*CP-nets*) is proposed. Like us, the authors present a model for representing and reasoning with the user preferences, where conditional preference expressions are permitted. In contrast to [4], our approach is defined over dynamic domains where the agents deal with incomplete and contradictory information. We formalize a special type of contextual query which by means of conditions allows the server to dynamically know what criterion to choose.

The study of Recommender Systems [14,15,11,8] has become important in AI over the last decade. We focus on a particular form of implementing recommender systems, called Recommender Servers that extends the integration of argumentation and recommender systems to a Multi-Agent System setting. Recommender Servers are based on an implementation of DeLP [9] called DeLP-Server [10]. In this paper we will introduce a defeasible logic programming recommender server which allows the clients to select through conditional expressions which criteria the server will use to answer their queries.

The rest of the paper is structured as follows. In Section 2 we will present the necessary background introducing basic definitions and some works that will be used in the rest of the paper; then, in Section 3 we will introduce the recommender server whose reasoning will be addressed by one of the comparison criteria indicated in the client query. To illustrate the formalism, in Section 4 we introduce an example in DeLP. Finally, in Section 5 we discuss related work and offer our conclusions and the possible directions for our future work.

## 2 Preliminary Background

In [10], an extended implementation of DeLP, called DeLP-server, has been presented; this system provides an argumentative reasoning service for multi-agent systems. A DeLP-server is a stand-alone application that stores a DeLP-program that is used to answer client queries. To answer queries, the DeLP-server will use the public knowledge stored in it as a Defeasible Logic Program, complementing this with individual knowledge a client agent might send, thus creating a particular scenario for the query (see Fig. 1). This information modifying the public knowledge stored in the DeLP-server is called context, and denoted $\mathcal{C}o$.

In DeLP, knowledge is represented using facts, strict and defeasible rules. *Facts* are ground literals representing atomic information, or the negation of atomic information using the strong negation "$\sim$". An overlined literal will denote the complement of that literal with respect to strong negation, *i.e.,* $\overline{L}$ is $\sim L$, and $\overline{\sim L}$ is $L$. *Strict Rules* are denoted $L_0 \leftarrow L_1, \ldots, L_n$ and represent firm information, whereas *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. A Defeasible Logic Program $\mathcal{P}$ (DeLP-program for short) is a set of facts, strict rules and defeasible rules.

**Example 2** *Continuing with Ex. 1, let $\mathcal{P}_l$ be a DeLP-program that models the information stored inside the on_board_computer and $\mathcal{P}_c$ be a DeLP-program representing the private pieces of information related to a driver particular context:*

$$\mathcal{P}_l = \left\{ \begin{array}{l} nearby\_lodging \prec included\_AC. \\ \sim nearby\_lodging \prec dangerous\_area. \\ road\_blocked \leftarrow workers\_on\_strike. \\ dangerous\_area \leftarrow zone\_lots\_of\_thefts. \\ downtown \leftarrow zone\_lots\_of\_restaurants. \end{array} \right\} \quad \mathcal{P}_c = \left\{ \begin{array}{l} zone\_lots\_of\_thefts. \\ zone\_lots\_of\_restaurants. \\ included\_AC. \\ workers\_on\_strike. \end{array} \right\}$$

Given a DeLP-program $\mathcal{P}$, a derivation for a literal $L$ from $\mathcal{P}$ is called 'defeasible', because there may exist information that contradicts $L$.

**Definition 1 (Defeasible/Strict Derivation)** *[9] Let $\mathcal{P}$ be a DeLP-program and $L$ a ground literal. A defeasible derivation of $L$ from $\mathcal{P}$, denoted $\mathcal{P} \mid\sim L$, is a finite sequence of ground literals $L_1, L_2, \ldots, L_n = L$, where each literal $L_i$ is in the sequence because:*

*(a) $L_i$ is a fact in $\mathcal{P}$, or*
*(b) there exists a rule $R_i$ in $\mathcal{P}$ (strict or defeasible) with head $L_i$ and body $B_1, B_2, \ldots, B_k$ and every literal of the body is an element $L_j$ of the sequence appearing before $L_i$ $(j < i)$.*

*We will say that $L$ has a* strict derivation *from $\mathcal{P}$, denoted $\mathcal{P} \vdash L$, if either $L$ is a fact or all the rules used for obtaining the sequence $L_1, L_2, \ldots, L_n$ are strict rules.*

Two literals are contradictory if they are complementary. Let $\mathcal{P}$ be a DeLP-program, the program $\mathcal{P}$ is coherent iff there are no strict derivations for two contradictory literals from $\mathcal{P}$. The set of literals supported by strict derivations is assumed to be non-contradictory, since these derivations are based on strict rules, which cannot be defeated.

A query is a literal $Q$, and the set of all possible queries will be denoted $\mathbb{Q}$. In [10], several contextual queries were defined, these types of queries allow the inclusion of private pieces of information related to the agents's particular context that will be taken into consideration when computing the answers.

**Definition 2 (Contextual query)** *[10]*
*Given a DeLP-program $\mathcal{P}$, a contextual query for $\mathcal{P}$ is a pair [Ls, Q] where Ls is a non-contradictory set of literals, and Q is a query.*

Three operators for DeLP-programs were introduced in [10] to consider different ways in which the clients' specific information is used when computing answers; these proposed operators will temporally modify the public knowledge stored in the server just for obtaining the answer. Here, our research is not focussed in these contextual operators, and we will use the union operator $\cup$ as a simple context handling operator.

Below, in Fig. 1, the graphical representation of the client/server model proposed in [10], is shown depicting a client agent sending a contextual query, and the main components of a DeLP-server.

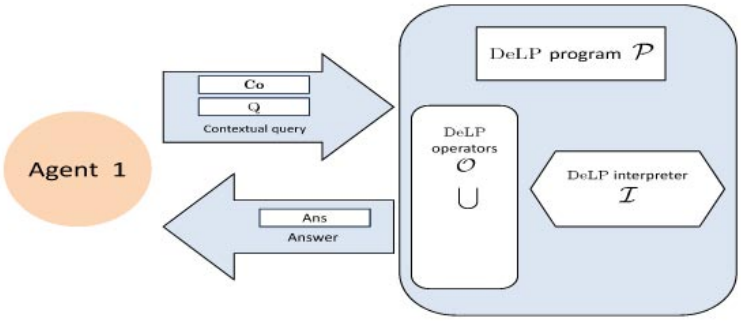


Fig. 1: Answer for a contextual query.

When reasoning with contradictory and dynamic information, the DeLP system builds arguments from the program. An argument $\mathcal{A}$ for a literal $L$ is a minimal, non contradictory set of defeasible rules such that together with the programs' strict knowledge allows the derivation of $L$ that will also be called the "conclusion" supported by $\mathcal{A}$, and for a given program the set of all possible arguments will be denoted as *Args*.

Given an argument $\mathcal{A}_1$ that is in conflict with other argument $\mathcal{A}_2$, to decide which one prevails, the two arguments must be compared using some criterion. Existing argumentation systems use a fixed comparison criterion embedded in the system.

## 3 Conditions based reasoning for the dynamic selection of criteria

As we have said, given two arguments $\mathcal{A}_1$ and $\mathcal{A}_2$ in conflict it is necessary to use a criterion to decide which argument prevails and if $\mathcal{A}_1$ is the prevailing argument, $\mathcal{A}_1$ is said to be a defeater of $\mathcal{A}_2$. If $\mathcal{A}_2$ is preferred to $\mathcal{A}_1$, then $\mathcal{A}_1$ will not defeat $\mathcal{A}_2$, and $\mathcal{A}_2$ prevails. If neither argument is preferred over the other, a blocking situation occurs and both arguments become defeated [9]. We will denote a preference criterion over arguments with $\mathsf{C}$ and a set of preference criteria $\mathcal{S} = \{\mathsf{C}_1, \mathsf{C}_2, \ldots, \mathsf{C}_n\}$. For our running example, we could assume the two criteria $\mathsf{C}_{comfort}$, that favors "comfort", and $\mathsf{C}_{security}$, that favors "security".

**Definition 3 (Preference criterion)** *Let Args be a set of arguments. A Preference criterion is a function* $\mathsf{C} : Args \times Args \longrightarrow \{\bot, \top\}$*, obtaining* $\top$ *when the first argument is preferred over the second, and* $\bot$ *otherwise.*

A contextual query includes the client's own information, and that information is used by the server to compute an answer. However, if in a client's query the server has to use a particular criterion, it will be necessary to change the structure of the contextual query to include them expanding it by including an expression that indicates to the server how to solve the query; to this end, a conditional expression will be added to the query expression.

A server will answer a query considering the preference criteria indicated by the client in a conditional-preference expression, or *cp-exp* for short; Definition 4 introduces the *cp-exps*. A *cp-exp* will be either a preference criterion, or *cp-exp* will be a guard $\mathcal{G}$ followed by two expressions $\mathcal{E}_1$ and $\mathcal{E}_2$ as explained below.

**Definition 4 (Conditional-preference expression)** *Given the criterion* $\mathsf{C} \in \mathcal{S}$ *and a set of literals* $\mathcal{G}$*. An expression* $\mathcal{E}$ *is a cp-exp iff:*

*i)* $\mathcal{E} = \mathsf{C}$*, or*

*ii)* $\mathcal{E} = [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2]$ *where* $\mathcal{E}_1$ *and* $\mathcal{E}_2$ *are cp-exps.*

*In case* $(i)$*, the preference criterion* $\mathsf{C}$ *is applied, while in case* $(ii)$ *the guard* $\mathcal{G}$ *is evaluated, if* $\mathcal{G}$ *verifies then* $\mathcal{E}_1$ *is applied, otherwise* $\mathcal{E}_2$ *is applied.*

As noted, a *cp-exp* may be associated to restrictive conditions. Let $\mathcal{P}'$ be a DeLP-program, to evaluate the set of literals $\mathcal{G}$ we will assume a function $eval(\mathcal{G}, \mathcal{P}')$ such that its range is $\{\top, \bot\}$, obtaining $\top$ iff for each literal $L \in \mathcal{G}$ there exist a strict derivation from $\mathcal{P}'$, *i.e.,* $\mathcal{P}' \vdash L$, $\bot$ otherwise.

We assume that the client agent may want that the criterion to be used be the one available by default already configured in the server. This criterion is denoted with the constant *"default"*. Therefore, if *cp-exp* $\mathcal{E} = \emptyset$ we assume that the used criterion will be the server default criterion. With a conditional-preference expression we are interested in providing a more declarative way of representing users preferences.

**Example 3** *Consider the criteria* $\mathsf{C}_{comfort}$ *and* $\mathsf{C}_{security}$ *stated above. The criteria to establish whether a nearby lodging is recommended may be obtained by means of some of the following expressions:*

$$\mathcal{E}_1 = [\{downtown\} : \mathsf{C}_{comfort}, \mathsf{C}_{security}]$$
$$\mathcal{E}_2 = [\{dangerous\_area\} : [\{road\_blocked\} : \mathsf{C}_{security}, \mathsf{C}_{comfort}], default]$$

*The expression* $\mathcal{E}_1$ *represents the following conditional-preference expression: "If there is a strict derivation for downtown then use the comparison criterion preferring comfort, otherwise use the criterion favoring security". The expression* $\mathcal{E}_2$ *represents a nested condition: "If there is a strict derivation for dangerous_area then the expression* $[\{road\_blocked\} : \mathsf{C}_{security}, \mathsf{C}_{comfort}]$ *is evaluated, otherwise it uses the default criterion. The way of evaluating* $[\{road\_blocked\} : \mathsf{C}_{security}, \mathsf{C}_{comfort}]$ *is similar to the expression* $\mathcal{E}_1$*".*

The example above shows how simple expressions such as $\mathcal{E}_1$ and more complex expressions such as $\mathcal{E}_2$ could be built. As we will show in the following section, this example is of special interest since it will serve to show the different results two queries using these expressions will produce.

Thus, the client agents could indicate how their queries have to be solved by the server. For that reason, the *cp-exp* denoting the conditions the server has to consider to select a criterion, are included in the queries. This new type of contextual query will be called *conditional-preference based query.*

**Definition 5 (Conditional-preference based query)** *A conditional-preference based query* CQ *is a tuple* $[\mathcal{C}o, \mathcal{E}, Q]$*, where* $\mathcal{C}o$ *is a particular context for* CQ*,* $\mathcal{E}$ *is a cp-exp, and* $Q$ *is a query.*

It is important to mention that *CQ* is an extension of the contextual query introduced in [10]. We refer the interested reader to [10] for details on those queries.

**Example 4** *Going back to Example 2 and considering Example 3. Given the query "nearby_lodging", two conditional-preference based queries can be built:*

$$[\mathcal{P}_c, \mathcal{E}_1, nearby\_lodging]$$
$$[\mathcal{P}_c, \mathcal{E}_2, nearby\_lodging]$$

Given a *CQ*, we said that the set of criteria belonging to a *cp-exp* is the set of valid criteria, denoted $\mathbb{C}$, for that particular query. Following Example 4, the fact of having the same query but with distinct valid criteria set makes possible to vary the criterion used for solving each query.

As defined next, a DeLP-interpreter will be represented, in general, as a function such that given a program, a preference criterion and a query, it returns the corresponding answer.

**Definition 6 (DeLP-interpreter)** *Let* $\mathbb{P}$ *be the set of coherent DeLP-programs,* $\mathbb{C}$ *be a set of valid criteria and* $\mathbb{Q}$ *be the set of possible queries. A DeLP-interpreter is a function* $\mathcal{I} : \mathbb{P} \times \mathbb{C} \times \mathbb{Q} \longrightarrow \mathbb{R}$*, where* $\mathbb{R}$ *is the set of possible answers, i.e.,* $\mathbb{R} = \{$NO, YES, UNDECIDED, UNKNOWN$\}$.

As already mentioned, we propose a client/server interaction allowing the client agents to interact with a recommender server by sending conditional-preference based queries. Now, we formally present the concept of preference-based reasoning server.

**Definition 7 (Conditional-preference based reasoning server)** *A conditional-reference based reasoning server is a 4-tuple $\mathcal{CRS} = \langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathcal{S} \rangle$, where $\mathcal{I}$ is a DeLP-interpreter, $\mathcal{O}$ is a set of DeLP-operators, $\mathcal{P}$ is a DeLP-program and $\mathcal{S}$ is a set of preference criteria.*

A $\mathcal{CRS}$ can accept queries from several clients and an agent can consult several servers. However, a $\mathcal{CRS}$ will answer only the queries that include criteria the server recognizes.

Consider a DeLP-program $\mathcal{P}$ modified with the context $\mathcal{C}o$, and a valid criterion $\mathsf{C} \in \mathbb{C}$. To evaluate a *cp-exp* $\mathcal{E}$ we will use a function $cond(\mathcal{E}, \mathcal{P})$ such that its range is the set of valid criteria $\mathbb{C}$, defined as follows.

**Definition 8 (Condition Evaluation Function)** *Let $\mathbb{E}$ be the set of all possible cp-exp, $\mathbb{P}$ be the set of coherent DeLP-programs, and $\mathbb{C}$ be a set of valid criteria, then we define the function cond with the following signature*

$$cond : \mathbb{E} \times \mathbb{P} \longrightarrow \mathbb{C},$$

*and the evaluation of $\mathcal{E}$ in $\mathcal{P}$ is defined as:*

*i)* $cond(\mathcal{E}, \mathcal{P}) = \mathsf{C}$ *if* $\mathcal{E} = \mathsf{C}$, *or*

*ii)* $cond(\mathcal{E}, \mathcal{P}) = cond(\mathcal{E}_1, \mathcal{P})$ *if* $\mathcal{E} = [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2]$ *and* $eval(\mathcal{G}, \mathcal{P}) = \top$, *or*

*iii)* $cond(\mathcal{E}, \mathcal{P}) = cond(\mathcal{E}_2, \mathcal{P})$ *if* $\mathcal{E} = [\mathcal{G} : \mathcal{E}_1; \mathcal{E}_2]$ *and* $eval(\mathcal{G}, \mathcal{P}) = \bot$.

Queries are answered using public knowledge stored in the server, plus individual knowledge sent with the query, and one of the criteria that a client agent sends as part of a conditional-preference expression. The answer will be obtained by means of an argumentative inference mechanism.

**Definition 9 (Answer for a query)** *Let $\mathcal{CRS} = \langle \mathcal{I}, \mathcal{O}, \mathcal{P}, \mathcal{S} \rangle$ be a conditional-preference based reasoning service, $PQ = [\mathcal{C}o, \mathcal{E}, Q]$ be a conditional-preference based query for $\mathcal{CRS}$, $\mathcal{P}'$ be a program modified with the context $\mathcal{C}o$, i.e., $\mathcal{P}' = \mathcal{P} \cup \mathcal{C}o$, and $\mathsf{C}_i$ be the criterion obtained from evaluating the expression $\mathcal{E}$. An answer for $PQ$ from $\mathcal{CRS}$, denoted $Ans(\mathcal{CRS}, PQ)$, corresponds to the result of the function $\mathcal{I}(\mathcal{P}', \mathsf{C}_i, Q)$.*

## 4 Application example

In this section we will present a DeLP example showing how the answer to a query varies according to the criterion that results from evaluating a *cp-exp*. Let $\mathcal{P}_l$ and $\mathcal{P}_c$ be the DeLP-programs presented in Example 2, and consider the conditional-preference based queries from Example 4;

1. $[\mathcal{P}_c, \mathcal{E}_1, nearby\_lodging]$
2. $[\mathcal{P}_c, \mathcal{E}_2, nearby\_lodging]$

such that

$$\mathcal{E}_1 = [\{downtown\} : \mathsf{C}_{comfort}, \mathsf{C}_{security}]$$
$$\mathcal{E}_2 = [\{dangerous\_area\} : [\{road\_blocked\} : \mathsf{C}_{security}, \mathsf{C}_{comfort}], default]$$

In both queries, the same DeLP-program $\mathcal{P}' = \mathcal{P}_l \cup \mathcal{P}_c$ is obtained. From the program $\mathcal{P}'$ two arguments can be built: the argument $\mathcal{A}$ in favor of recommending a nearby lodging.

$$\mathcal{A} = \{\ nearby\_lodging \prec included\_AC.\ \}$$

and the argument $\mathcal{B}$ in favor of not recommending a nearby lodging:

$$\mathcal{B} = \{\ \sim nearby\_lodging \prec dangerous\_area.\ \}$$

Clearly, $\mathcal{A}$ and $\mathcal{B}$ are in conflict. To determine which one prevails, we have to establish the argument comparison criterion to be used. Consider the first conditional-preference based query presented above:

$$[\mathcal{P}_c, \mathcal{E}_1, nearby\_lodging]$$

Due to the strict derivation of "*downtown*" from $\mathcal{P}'$, we obtain the criterion $\mathsf{C}_{comfort}$ as the result of the function "*cond*". We assume that $\mathsf{C}_{comfort}$ establishes that $\mathcal{A}$ is preferred to $\mathcal{B}$, since the argument $\mathcal{A}$ has the information that the bedrooms in the lodge have air conditioning, then $\mathcal{A}$ will defeat to $\mathcal{B}$. In DeLP a query $Q$ is warranted from a program $\mathcal{P}$ if there exists an undefeated argument $\mathcal{A}_1$ supporting $Q$. To establish whether the argument $\mathcal{A}$ is an undefeated argument, we will assume that the dialectical analysis proposed in [9] is performed. Since, this dialectical process assures that the conclusion *nearby_lodging* is warranted, then the answer for the query is YES.
Consider now the second conditional-preference based query presented above:

$$[\mathcal{P}_c, \mathcal{E}_2, nearby\_lodging]$$

after completing the whole argumentative process, the answer for *nearby_lodging* is NO, *i.e.,* the conclusion *nearby_lodging* is not warranted. In this case, the strict derivations of *dangerous_area* and *road_blocked* from $\mathcal{P}'$ are obtained, then the evaluation of $\mathcal{E}_2$ establish that the chosen criterion is $\mathsf{C}_{security}$. In order to obtain the answer for *nearby_lodging*, we assume that $\mathsf{C}_{security}$ determines that $\mathcal{B}$ is preferred to $\mathcal{A}$, since the argument $\mathcal{B}$ has the information that the vehicle is in a dangerous area. As $\mathcal{B}$ is a non-defeated argument, then the conclusion $\sim$*nearby_lodging* is warranted and the answer for the query is NO. For a detailed presentation of the dialectical analysis used for answer the two conditional-preference based queries introduced in this paper see [9].

As mentioned, in [10] the proposed server is configured to use a fixed comparison criterion embedded in the system. Thus, the answers to our example queries will be always solved using the same criterion. In contrast to [10], in the complete example we show that with our approach the same query with the same context but with different conditional-preference expressions can give different criteria, and possibly different answers. This was one of our goals.

## 5 Conclusions, related and future work

We presented a model that allows an argumentative reasoning server to dynamically select the argument comparison criterion to be used. For this, we formally defined the notion of conditional-preference expression, which is part of a new type of contextual query, called conditional-preference based query. We showed how these expressions are evaluated by means of a function, called "*cond*", which determines which criterion prevails for the query. DeLP was proposed as the knowledge representation and reasoning system, therefore the DeLP-interpreter is in charge of solving the queries. In Section 4 an example was presented where an agent performs two queries with the same context but with different conditional-preference expressions, getting different results. We showed how in the proposed model, argument comparison criteria are directly related to the inferences obtained by an agent.

Our approach was in part inspired by [10], where several servers can be created, and knowledge can be shared through them. Nevertheless, in contrast with us, they use a preference criteria embedded into the interpreter, *i.e.,* to answer a query, the server is configured to use the same specific criterion. In fact, we provide clients with the possibility of indicating to the server what criteria could use to compute the answer for a specific query.

In [4], an approach where the preference is subjected to conditional dependence was proposed. A preference relation is defined as a total pre-order (a ranking) over some set of variables such that the preference over the values of one variable depends on the value of others. Their main contribution is a graphical representation of preferences that reflects conditional dependence and independence of preference statements under a *ceteris paribus* (all else being equal) interpretation. Similar to us, the authors present a model for representing and reasoning with the user preferences, where conditional preference expressions are permitted. In contrast with us, they provide a framework where the preferences are considered for decision making where the space of possible actions or decisions available to someone is fixed, with well-understood dynamics, conversely, in our framework the situation is different, *i.e.,* the selected application domains are dynamic and agents deal with incomplete and contradictory information; for that reason, our research is focused on argumentative systems.

As future work we are developing an implementation of a DeLP-server that can dynamically handle conditional based preference criteria. We are also studying the possibility of developing more powerful comparison criteria expressions

that allow more expressive combinations. Another extension will be to integrate our proposed framework with others argumentative systems similar to DeLP.

**Acknowledgements:** This work is partially supported by CONICET, Universidad Nacional de Entre Ríos (PID-UNER 7041), Universidad Nacional del Sur, SGCyT.

## References

1. Alsinet, T., Chesñevar, C.I., Godo, L., Simari, G.R.: A logic programming framework for possibilistic argumentation: Formalization and logical properties. Fuzzy Sets and Systems 159(10), 1208–1228 (2008)
2. Amgoud, L., Parsons, S.: Agent dialogues with conflicting preferences. In: ATAL. pp. 190–205 (2001)
3. Antoniou, G., Maher, M.J., Billington, D.: Defeasible logic versus logic programming without negation as failure. J. Log. Program. 42(1), 47–57 (2000)
4. Boutilier, C., Brafman, R.I., Hoos, H.H., Poole, D.: Reasoning with conditional ceteris paribus preference statements. In: Laskey, K.B., Prade, H. (eds.) UAI. pp. 71–80. Morgan Kaufmann (1999)
5. Boutilier, C.: Toward a logic for qualitative decision theory. In: KR. pp. 75–86 (1994)
6. Capobianco, M., Chesñevar, C.I., Simari, G.R.: Argumentation and the dynamics of warranted beliefs in changing environments. Autonomous Agents and Multi-Agent Systems 11(2), 127–151 (2005)
7. Capobianco, M., Simari, G.R.: A proposal for making argumentation computationally capable of handling large repositories of uncertain data. In: SUM. pp. 95–110 (2009)
8. Deagustini, C.A.D., Fulladoza Dalibón, S.E., Gottifredi, S., Falappa, M.A., Chesñevar, C.I., Simari, G.R.: Relational databases as a massive information source for defeasible argumentation. Knowledge-Based Systems (to appear) (2013)
9. García, A.J., Simari, G.R.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming (TPLP) 4, 95–138 (2004)
10. García, A.J., Rotstein, N.D., Tucat, M., Simari, G.R.: An argumentative reasoning service for deliberative agents. In: KSEM. pp. 128–139 (2007)
11. Konstan, J.A.: Introduction to recommender systems: Algorithms and evaluation. ACM Trans. Inf. Syst. 22(1), 1–4 (2004)
12. Li, M., Vo, Q.B., Kowalczyk, R.: Majority-rule-based preference aggregation on multi-attribute domains with cp-nets. In: AAMAS. pp. 659–666 (2011)
13. Loui, R.P.: Defeat among arguments: a system of defeasible inference. Computational Intelligence 3, 100–106 (1987)
14. Maher, M.J., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient defeasible reasoning systems. International Journal on Artificial Intelligence Tools 10(4), 483–501 (2001)
15. Resnick, P., Varian, H.R.: Recommender systems - introduction to the special section. Commun. ACM 40(3), 56–58 (1997)
16. Simari, G.R., Loui, R.P.: A mathematical treatment of defeasible reasoning and its implementation. Artificial Intelligence 53(2-3), 125–157 (1992)
17. Stolzenburg, F., García, A.J., Chesñevar, C.I., Simari, G.R.: Computing generalized specificity. J. of Applied Non-Classical Logics 13(1), 87–113 (2003)
18. Vreeswijk, G.: Abstract argumentation systems. Artificial Intelligence 90(1-2), 225–279 (1997)