An Argumentative Reasoning Service for Deliberative Agents

Alejandro J. García Nicolás D. Rotstein Mariano Tucat Guillermo R. Simari

Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Department of Computer Science and Engineering
Universidad Nacional del Sur, Bahía Blanca, Argentina.
{ajg, ndr, mt, grs}@cs.uns.edu.ar

Abstract. In this paper we propose a model that allows agents to deliberate using defeasible argumentation, to share knowledge with other agents, and to represent individual knowledge privately. We describe the design and implementation of a Defeasible Logic Programming Server that handles queries from several remote client agents. Queries will be answered using public knowledge stored in the Server and individual knowledge that client agents can send as part of a query, providing a particular context for it. The Server will answer these contextual queries using a defeasible argumentative analysis. Different types of contextual queries are presented and analyzed.

1 Introduction

Deliberative agents that take part of a Multi-agent System (MAS) usually reason by using two sources of knowledge: public knowledge they share with other agents, and individual or private knowledge that arise in part from their own perception of the environment. In this paper we propose a model that allows agents to deliberate using defeasible argumentation, to share knowledge with other agents, and to represent individual knowledge privately. We focus on the design and implementation of a client-server approach based on Defeasible Logic Programming that provides a knowledge representation formalism and a Defeasible Argumentation reasoning service. Thus, agents can reason with the mentioned formalism, using both private and shared knowledge, by means of this external service.

In our approach, a Defeasible Logic Programming Server (DeLP-server) will answer 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 will use this public knowledge together with individual knowledge that clients might send, creating a particular *context* for the query. These *contextual queries* will be answered

Partially supported by Universidad Nacional del Sur, CONICET (PIP 5050), and Agencia Nacional de Promoción Científica y Tecnológica.

using a defeasible argumentative analysis. 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. Our approach do not impose any restriction over the type, architecture, or implementation language of the client agents.

In our 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. Since agents are not restricted to consult a unique server, they may perform the same contextual query to different servers, and they may share different knowledge with other agents through different servers. Thus, several configurations of agents and servers can be established (statically or dynamically). For example, special-purpose DeLP-servers can be used, each of them representing particular shared knowledge of a specific domain. Thus, like in other approaches, shared knowledge will not be restricted to be in a unique repository and therefore, it can be structured in many ways.

2 DeLP Basis

In our approach, both the individual knowledge of an agent and the public knowledge loaded in a DeLP-Server are represented using a defeasible logic program. A brief description of Defeasible Logic Programming (DeLP) is included in this section –for a detailed presentation see [1]. DeLP is a formalism that combines logic programming and defeasible argumentation [2, 3]. 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 "$\sim$". *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. In both cases, the *head* L_0 is a literal and the *body* $\{L_i\}_{i>0}$ is a set of literals. In this paper we will consider a restricted form of program that do not have strict rules.

Definition 1. *A restricted defeasible logic program (DeLP-program for short) $\mathcal{P}$ is a set of facts and defeasible rules. When required, $\mathcal{P}$ is denoted (Π,Δ) distinguishing the subset Π of facts and the subset Δ of defeasible rules.*

Strong negation is allowed in the head of program rules, and hence may be used to represent contradictory knowledge. From a program (Π,Δ) contradictory literals could be derived, however, the set Π (which is used to represent non-defeasible information) must possess certain internal coherence. Therefore, Π has to be non-contradictory, *i.e.*, no pair of contradictory literals can be derived from Π. Given a literal L the complement with respect to strong negation will be denoted $\overline{L}$ (*i.e.*, $\overline{a}{=}{\sim}a$ and $\overline{{\sim}a}{=}a$). Adding facts to a DeLP-program can produce a contradictory set Π, producing a non-valid program. However, defeasible rules can be added without any restriction.

Observation 1 *Let (Π,Δ) be a DeLP-program, Δ_1 a set of defeasible rules and Π_1 a set of facts. The pair (Π,($\Delta \cup \Delta_1$)) is a valid DeLP-program, but ($\Pi \cup \Pi_1$, Δ) may not because $\Pi \cup \Pi_1$ can be a contradictory set. Nevertheless, if $\Pi \cup \Pi_1$ is a non-contradictory set then ($\Pi \cup \Pi_1$, $\Delta \cup \Delta_1$) is a valid DeLP-program.*

Definition 2 (DeLP-query). *A* DeLP-query *is a ground literal that DeLP will try to warrant. A query with at least one variable will be called* schematic query *and will represent the set of DeLP-queries that unify with the schematic one.*

To deal with contradictory and dynamic information, in DeLP, *arguments* for conflicting pieces of information are built and then compared to decide which one prevails. The prevailing argument provides a *warrant* for the information that it supports. In DeLP a query L is *warranted* from a program (Π,Δ) if a *non-defeated* argument $\mathcal{A}$ supporting L exists. An *argument* $\mathcal{A}$ for a literal L [1], denoted $\langle \mathcal{A}, L \rangle$, is a minimal set of defeasible rules $\mathcal{A} \subseteq \Delta$, such that $\mathcal{A} \cup \Pi$ is non-contradictory and there is a derivation for L from $\mathcal{A} \cup \Pi$.

Example 1. Consider the DeLP-program $\mathcal{P}_1 = (\Pi_1, \Delta_1)$, where $\Pi_1 = \{q, s, t\}$ and $\Delta_1 = \{(r \multimap q)(\sim r \multimap q, s)(r \multimap s)(\sim r \multimap t)(\sim a \multimap q)(a \multimap s)\}$. From $\mathcal{P}_1$ the following arguments can be built:
$\langle \mathcal{R}_1, \sim r \rangle = \langle \{\sim r \multimap t\}, \sim r \rangle$ $\langle \mathcal{R}_2, r \rangle = \langle \{r \multimap q\}, r \rangle$ $\langle \mathcal{R}_3, r \rangle = \langle \{r \multimap s\}, r \rangle$
$\langle \mathcal{R}_4, \sim r \rangle = \langle \{\sim r \multimap q, s\}, \sim r \rangle$ $\langle \mathcal{A}_1, \sim a \rangle = \langle \{\sim a \multimap q\}, \sim a \rangle$ $\langle \mathcal{A}_2, a \rangle = \langle \{a \multimap s\}, a \rangle$

To establish if $\langle \mathcal{A}, L \rangle$ is a non-defeated argument, *defeaters* for $\langle \mathcal{A}, L \rangle$ are considered, *i.e.*, counter-arguments that by some criterion are preferred to $\langle \mathcal{A}, L \rangle$. It is important to 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* [4], a criterion that favors two aspects of an argument: it prefers (1) a *more precise* argument (*i.e.*, with greater information content) or (2) a *more concise* argument (*i.e.*, with less use of rules). Using this criterion in Example 1, $\langle \mathcal{R}_4, \sim r \rangle$ is preferred to $\langle \mathcal{R}_2, r \rangle$ (more precise).

A defeater $\mathcal{D}$ for an argument $\mathcal{A}$ can be *proper* ($\mathcal{D}$ is preferred to $\mathcal{A}$) or *blocking* (same strength). A defeater can attack the conclusion of another argument or an inner point of it. Since defeaters are arguments, there may exist defeaters for them, and defeaters for these defeaters, and so on. Thus, a sequence of arguments called *argumentation line* [1] can arise. Clearly, for a particular argument there might be more than one defeater. Therefore, many argumentation lines could arise from one argument, leading to a tree structure called *dialectical tree* [1]. In a dialectical tree (see Fig. 1), every node (except the root) is a defeater of its parent, and leaves are non-defeated arguments. A dialectical tree provides a structure for considering all the possible acceptable argumentation lines that can be generated. In a dialectical tree every node can be marked as *defeated* or *undefeated*: leaves are marked as undefeated nodes, and inner nodes are marked defeated when there is at least a child marked undefeated, or are marked undefeated when all its children are marked defeated. Figure 1 shows three different marked dialectical trees, where white triangles represent undefeated nodes, black triangles defeated ones, and arrows the defeat relation.

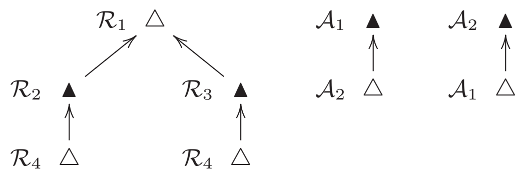

Fig. 1. Dialectical trees of Ex. 1

Definition 3 (Warranting a DeLP-query). *A DeLP-query Q is warranted from a DeLP-program $\mathcal{P}$ if there exists an argument $\mathcal{A}$ supporting Q such that $\mathcal{A}$ is the root of a dialectical tree and the root is marked as undefeated.*

Definition 4 (Answer for a DeLP-query). *The answer for a query Q from a DeLP-program $\mathcal{P}$ is either:* YES*, if Q is warranted from $\mathcal{P}$;* NO*, if the complement of Q is warranted from $\mathcal{P}$;* UNDECIDED*, if neither Q nor its complement are warranted from $\mathcal{P}$; or* UNKNOWN*, if Q is not in the language of the program $\mathcal{P}$.*

Consider again Ex. 1. From $\mathcal{P}_1$ the answer for $\sim r$ is YES whereas for r is NO (Fig. 1 shows the dialectical tree that provides a warrant for $\sim r$). The answer for a is UNDECIDED and the answer for $\sim a$ is also UNDECIDED (observe that $\mathcal{A}_1$ and $\mathcal{A}_2$ block each other). Finally, the answer for z is UNKNOWN.

3 Contextual Queries

As stated above, several DeLP-Servers can be used in a MAS, and each of them provides a defeasible argumentation reasoning service for other agents. Public knowledge (represented as a DeLP-program) can be stored in each DeLP-Server, and will be used for answering DeLP-queries. For example, if the query $\sim r$ is sent to a DeLP-Server where the program (Π_1, Δ_1) of Ex. 1 is stored, then the answer YES will be returned.

Besides public knowledge, agents may have their own private knowledge. Therefore, the proposed model not only allows agents to perform queries to the program stored in a DeLP-Server, but also permits the inclusion in the query of private pieces of information related to the agent's particular context. This type of query will be called *contextual query* and will be introduced next.

Definition 5 (Contextual query). *Given a DeLP-program $\mathcal{P}$=$(\Pi$,$\Delta)$, a contextual query for $\mathcal{P}$ is a pair $[\Phi$,$Q]$ where Q is a DeLP-query, and Φ is a non-contradictory set of literals such that $\Pi \cup \Phi$ is a non-contradictory set.*

Definition 6 (Warrant for a contextual query). *A contextual query $[\Phi$,$Q]$ is warranted from the DeLP-program $(\Pi$,$\Delta)$ if Q is warranted from $(\Pi\cup\Phi$,$\Delta)$.*

Note that Def. 5 requires $\Pi \cup \Phi$ to be non-contradictory due to the problem addressed in Obs. 1. The effect of performing a contextual query $[\Phi, Q]$ to a DeLP-program $(\Pi$,$\Delta)$ (Def. 6) will be to add all the literals in Φ to $(\Pi$,$\Delta)$ building (temporarily) a new program $(\Pi\cup\Phi$,$\Delta)$, which will be used for returning the appropriate answer. Note that although some new information not present in $(\Pi$,$\Delta)$ may be considered for warranting a contextual query $[\Phi$,$Q]$, this process will not affect the content of the stored program $(\Pi$,$\Delta)$ for future queries.

Observation 2 *A DeLP-query is a particular case of a contextual query* $[\Phi,Q]$ *where* $\Phi = \emptyset$*. Given* $[\emptyset,Q]$ *and a program* (Π,Δ)*, it is easy to show that* $[\emptyset,Q]$ *is warranted from* (Π,Δ)*, iff* Q *is warranted from* (Π,Δ)*.*

Since contextual queries are performed following a client/server model, the sender of a query (client) may not know the content of the program in a Server. Therefore, it may not be possible for a client to know in advance whether $\Pi \cup \Phi$ is non-contradictory. Hence, we will extend the notion of answer (Def. 4) to return INVALID and the subset of Φ that produces the contradictory set.

Definition 7 (Answer for a contextual query). *An answer for a contextual query* $[\Phi,Q]$ *from a DeLP-program* (Π,Δ) *is a pair* (Ans,S)*, where* $Ans \in \{$YES, NO, UNDECIDED, UNKNOWN, INVALID$\}$ *and* S *is a set of literals. If* $\Pi \cup \Phi$ *is contradictory, then* $Ans =$ INVALID *and* $S = \{L \in \Pi \cup \Phi \mid \overline{L} \in \Pi \cup \Phi\}$ *. Otherwise, the answer will be* (YES,$\{\}$) *if* Q *is warranted from* $(\Pi \cup \Phi,\Delta)$*;* (NO,$\{\}$) *if* $\overline{Q}$ *is warranted from* $(\Pi \cup \Phi,\Delta)$*;* (UNDECIDED,$\{\}$) *if neither* Q *nor* $\overline{Q}$ *are warranted from* $(\Pi \cup \Phi,\Delta)$*; and* (UNKNOWN,$\{\}$)*, if* Q *is not in the language of* $(\Pi \cup \Phi,\Delta)$*.*

Example 2. Consider the DeLP-program $\mathcal{P}_2 = (\Pi_2, \Delta_2)$, where:
$\Pi_2 = \{r\}$ and $\Delta_2 = \{(m \prec c), (\sim m \prec r), (c \prec s), (\sim c \prec s, p)\}$.
Here, the letters m, r, c, s and p are the abbreviations of *use_metro*, *rush_hour*, *closed_roads*, *snow* and *snowplow* respectively. From $\mathcal{P}_2$, the answer for the DeLP-query c is UNDECIDED, whereas the answer for the contextual query $[\{s\}, c]$ is (YES,$\{\}$) (s activates an argument for c). Note that the answer for $[\{s,p\}, c]$ is (NO,$\{\}$) because now p activates a proper defeater. Observe that from $\mathcal{P}_2$, the answer for $\sim m$ is YES, however, the answer for the contextual query $[\{s\}, \sim m]$ is (UNDECIDED,$\{\}$) and the answer for $[\{s,p\}, \sim m]$ is (YES,$\{\}$). Finally, the answer for $[\{\sim m, p, \sim r\}, r]$ is (INVALID,$\{$r$\}$). Since these contextual queries do not actually make a change on program $\mathcal{P}_2$, after performing them, the DeLP-query $\sim m$ will still remain warranted from $\mathcal{P}_2$ and c will not be warranted from $\mathcal{P}_2$.

Proposition 1. *Given a DeLP-program* (Π,Δ) *and a contextual query* $[\Phi,Q]$*, if* $Q \in \Pi \cup \Phi$ *then the answer for* $[\Phi,Q]$ *is* (YES,$\{\}$)*. If* $\overline{Q} \in \Pi \cup \Phi$ *then the answer for* $[\Phi,Q]$ *is* (NO,$\{\}$)*.* Proof: By the proposition 5.1 given in [1], if $Q \in \Pi \cup \Phi$ then Q is warranted from $(\Pi \cup \Phi,\Delta)$. Hence, by Definition 7, the answer for $[\Phi,Q]$ is (YES,$\{\}$). By the same cited proposition, if $\overline{Q} \in \Pi \cup \Phi$ then $\overline{Q}$ is warranted from $(\Pi \cup \Phi,\Delta)$ and hence, the answer for $[\Phi,Q]$ is (NO,$\{\}$).

3.1 Generalized Contextual Queries

In a contextual query $[\Phi,Q]$, imposing that $\Pi \cup \Phi$ must be a non-contradictory set may be in some cases too restrictive. Although the agent will be informed in case of an invalid query (Definition 7), the agent should have to reformulate the query in order to obtain an answer. Therefore, it would be useful to have a more general type of query that, without imposing the mentioned restriction, resolves the problem of a contradictory set before the query is processed.

In this section we propose three generalized types of contextual queries, each one with a different policy for handling the mentioned problem. In the first one (called non-prioritized contextual query), given a query $[\Phi,Q]$ and a program (Π,Δ), if $\Pi \cup \Phi$ is contradictory, then the DeLP-Server will assign less priority to the knowledge of the agent (Φ) than to the one in the Server (Π). Hence, those literals from Φ that are problematic will be ignored when answering that query. In contrast, in the third one (called prioritized contextual query), when $\Pi \cup \Phi$ is contradictory, more priority will be assigned to the literals in Φ than those in Π. Therefore, the literals in Π that produce the contradiction will be temporarily ignored for answering that query. The second generalized type (called restrictive contextual query) has the effect of temporarily ignore literals from Π while processing the query. In this case, no policy is needed since no literals are added to Π and a contradiction can never occur. The symbols "$*$", "$+$", "$-$" will be used to distinguish each type of query.

Definition 8 (Non-prioritized contextual query).
Given a DeLP-program $\mathcal{P}$, a non-prioritized contextual query for $\mathcal{P}$ is denoted $[\Phi^*,Q]$, where Q is a DeLP-query and Φ^* is a non-contradictory set of literals.

Definition 9 (Warrant for a non-prioritized contextual query).
A query $[\Phi^*,Q]$ is warranted from (Π,Δ) if Q is warranted from $((\Pi \otimes \Phi^*),\Delta)$. Let $C(\Pi) = \{\overline{L} \text{ if } L \in \Pi\}$, then $(\Pi \otimes \Phi^*) = (\Phi^* \setminus C(\Pi)) \cup \Pi$.

In contrast with Def. 5, in a non-prioritized contextual query $[\Phi^*,Q]$ the set $\Pi \cup \Phi^*$ can be contradictory. However, the set Φ^* is required to be non-contradictory as a minimal coherence principle. This type of query is useful for agents that need to assign a preference to the information stored in the Server. The operator $\otimes$ resembles a non-prioritized merge operator that assigns a preference to the elements of Π when merging Π and Φ. Thus, the effect of performing a non-prioritized contextual query $[\Phi^*,Q]$ to a program (Π,Δ) will be to temporarily consider in (Π,Δ) only those literals from Φ^* that are not problematic. Thus, if $\Pi \cup \Phi^*$ is contradictory, those literals from Φ^* that produce contradictions will be ignored by the server. The subset of ignored literals will be returned with the answer to inform that they have not been used for the dialectical analysis. If $[\Phi^*,Q]$ returns $(\cdot,\{\})$, all the literals of Φ were considered.

Definition 10 (Answer for a non-prioritized contextual query).
The answer for $[\Phi^*,Q]$ from (Π,Δ) is a pair (Ans,S), where $Ans \in \{$YES, NO, UNDECIDED, UNKNOWN, INVALID$\}$ is computed as shown in Def. 7, and the set $S \subseteq \Phi^*$ will contain those literals that have been ignored by the DeLP-Server while processing Q, (i.e., $L \in S$ if $L \in \Phi^*$ and $\overline{L} \in \Pi$). If Φ^* is contradictory, then $Ans =$ INVALID and $S = \{L \in \Phi^* \mid \overline{L} \in \Phi^*\}$.

Example 3. Consider the DeLP-program $\mathcal{P}_3 = (\Pi_3, \Delta_3)$, where $\Pi_3 = \{r, p, \sim f\}$ and $\Delta_3 = \{(\sim m \prec r), (m \prec c, r), (t \prec \sim c)\}$. The letters m, r, c, p, f and t are the abbreviations of *use_metro*, *rush_hour*, *closed_roads*, *park_closed*, *fog* and *use_taxi* respectively. From $\mathcal{P}_3$ the answer for the query $\sim m$ is YES whereas for the non-prioritized contextual query $[\{c\}^*, {\sim}m]$ the answer is (NO,$\{\}$). Observe that the query $[\{{\sim}r\}^*, {\sim}m]$ returns the answer (YES,$\{{\sim}r\}$), since $r \in \Pi$ is preferred. The DeLP-query t has the answer NO, whereas $[\{{\sim}c, r, {\sim}p, f\}^*, t]$ returns the answer (YES,$\{{\sim}p, f\}$). Note that $[\{{\sim}c, {\sim}t\}^*, t]$ returns the answer (NO,$\{\}$) because $\{{\sim}c, {\sim}t\} \cup \Pi_3$ is non-contradictory and hence, the literal ${\sim}t$ is included for answering the query. Although there exists a defeasible derivation for t with ${\sim}c$ and $(t \prec {\sim}c)$, if ${\sim}t$ is present as a fact, in DeLP there is no argument for t.

Proposition 2. *Given a DeLP-program (Π,Δ) and a non-prioritized contextual query $[\Phi^*,Q]$ it holds that:*
1. *if $Q \in \Pi$ then the answer for $[\Phi^*,Q]$ will always be* (YES,S).
2. *if $Q \in \Phi^*$ and $\overline{Q} \notin \Pi$ then the answer for $[\Phi^*,Q]$ will always be* (YES,S).
3. *if $\overline{Q} \in \Pi$ then the answer for $[\Phi^*,Q]$ will always be* (NO,S).
4. *if $\overline{Q} \in \Phi^*$ and $Q \notin \Pi$ will always be for $[\Phi^*,Q]$ will be always* (NO,S).

(Proof is omitted due to space restrictions)

Observation 3 *A contextual query $[\Phi,Q]$ is a particular case of a non-prioritized contextual query $[\Phi^*,Q]$ where $(\Pi \cup \Phi^*)$ is a non-contradictory set.*

The next type of contextual query will allow an agent to temporarily ignore some literals from the program stored in a DeLP-Server with the purpose of warranting Q. Since no literals will be added to the program in the server, then no restriction over the literals included in the query is needed.

Definition 11 (Restrictive contextual query).
Given a DeLP-program $\mathcal{P}=(\Pi,\Delta)$, a restrictive contextual query for $\mathcal{P}$ is denoted $[\Phi^-,Q]$ where Q is a DeLP-query and Φ^- is an arbitrary set of literals.

Definition 12 (Warrant for a restrictive contextual query). *The query $[\Phi^-,Q]$ is warranted from (Π,Δ) if Q is warranted from $((\Pi \setminus \Phi^-),\Delta)$.*

Definition 13 (Answer for a restrictive contextual query).
*The answer $[\Phi^-,Q]$ from (Π,Δ) is a pair (Ans,S), where $Ans \in \{$*YES, NO, UNDECIDED, UNKNOWN*$\}$ is computed as shown in Def. 7, and $S = \Phi^- \cap \Pi$, i.e., S will contain those literals that have been effectively ignored by the DeLP-Server.*

Example 4. Consider $\mathcal{P}_3$ from Ex. 3. The answer for the restrictive contextual query $[\{r\}^-, {\sim}m]$ will be (UNDECIDED,{r}). Observe that a query $[\Phi^-,Q]$ might include in Φ^- literals that are not in Π. This superfluous literals will have not effect to the query, and the agent will know them because they will not be returned in the answer. For instance, the answer for $[\{r, z\}^-, {\sim}m]$ is (UNDECIDED,{r}) .

Observation 4 *Given a restrictive contextual query $[\Phi^-,Q]$ to a DeLP-program (Π,Δ), in the particular case that $(\Pi \cap \Phi^-)$ is empty, then the restrictive contextual query behaves like a contextual query.*

As stated above, the type of contextual query we propose next will assign a preference to the information sent by the agent. Thus, if the Server has a program (Π,Δ) and the query includes a set Φ such that $(\Pi \cup \Phi)$ is a contradictory set, then, some literals from Π will be ignored for answering the query.

Definition 14 (Prioritized contextual query).
Given a DeLP-program $\mathcal{P}$, a prioritized contextual query for $\mathcal{P}$ is denoted $[\Phi^+,Q]$ where Q is a DeLP-query and Φ^+ is a non-contradictory set of literals.

Definition 15 (Warrant for a prioritized contextual query).
The query $[\Phi^+,Q]$ is warranted from (Π,Δ) if Q is warranted from $((\Pi\oplus\Phi^+),\Delta)$. Let $C(\Phi^+) = \{\overline{L} \text{ if } L \in \Phi^+\}$, then $(\Pi \oplus \Phi^+)= (\Pi \setminus C(\Phi^+)) \cup \Phi^+$.

The operator $\oplus$ resembles a prioritized merge operator. It removes (temporarily) from Π the complement of the literals that belongs to Φ^+ and then (temporarily) adds Φ^+. Both operations (the subtraction and then the addition) are made in the same "transaction". As remarked above, all of these changes to the set Π are local to the warrant process of the contextual query, and they will not affect other queries nor make permanent changes to Π.

Definition 16 (Answer for a prioritized contextual query).
The answer for $[\Phi^+,Q]$ from (Π,Δ) is a pair (Ans,S). The first element $Ans \in$ {YES, NO, UNDECIDED, UNKNOWN, INVALID} is computed as shown in Def. 7, and the set $S \subseteq \Pi$ will contain those literals that have been ignored by the DeLP-Server while processing Q (i.e., $L \in S$ if $L \in \Pi$ and $\overline{L} \in \Phi^+$). If Φ^+ is contradictory, then $Ans=$ INVALID and $S=\{L \in \Phi^+ \mid \overline{L} \in \Phi^+ \}$.

Example 5. Consider again $\mathcal{P}_3$ of Ex. 3. From (Π_3, Δ_3) the answer for the query $[\{c\}^+, {\sim}m]$ is (NO,$\{\}$). In contrast with the Ex. 3, the answer for $[\{{\sim}r, c\}^+, {\sim}m]$ is (UNDECIDED,$\{r\}$), because ${\sim}m$ is warranted from $(\{p, {\sim}f, {\sim}r\}, \Delta_3)$ since ${\sim}r$ is preferred over $r \in \Pi$. The answer for $[\{{\sim}c, r, {\sim}p, f\}^+, t]$ is (YES,$\{p, {\sim}f\}$).

Proposition 3. *Given a DeLP-program (Π,Δ) and a prioritized contextual query $[\Phi^+,Q]$ it holds that:*
1. *if $Q \in \Phi^+$ then the answer for $[\Phi^+,Q]$ will always be* (YES,S).
2. *if $Q \in \Pi$ and $\overline{Q} \notin \Phi^+$ then the answer for $[\Phi^+,Q]$ will always be* (YES,S).
3. *if $\overline{Q} \in \Phi^+$ then the answer for $[\Phi^+,Q]$ will always be* (NO,S).
4. *if $\overline{Q} \in \Pi$ and $Q \notin \Phi^+$ then the answer for $[\Phi^+,Q]$ will always be* (NO,S).

The proof is analogous to Proposition 2 and is not included due to space reasons.

Observation 5 *The effect of considering a negated literal ${\sim}L$ to answer a query differs from the effect of removing its complement L (see Ex. 6).*

Example 6. Consider $\Pi_6 = \{x\}$ and $\Delta_6 = \{(a \mathrel{-\!\!\prec} x), (b \mathrel{-\!\!\prec} {\sim}x)\}$. From (Π_6, Δ_6) the answer for the DeLP-query a is YES, and the answer for b is UNDECIDED. Observe that the answer for $[\{\text{x}\}^-,a]$ is (UNDECIDED,$\{\}$) –since there are no arguments for nor against a– and the answer for $[\{\text{x}\}^-,b]$ is also (UNDECIDED,$\{\}$). Nevertheless, the answer for $[\{{\sim}\text{x}\}^+,a]$ is (UNDECIDED,$\{\text{x}\}$), and the answer for $[\{{\sim}\text{x}\}^+,b]$ is (YES,$\{\text{x}\}$). Observe finally that the answer for $[\{{\sim}\text{x}\}^*,a]$ is (YES,$\{{\sim}\text{x}\}$), and the answer for $[\{{\sim}\text{x}\}^*,b]$ is (UNDECIDED,$\{{\sim}\text{x}\}$).

Observation 6 *A contextual query $[\Phi,Q]$ is a particular case of a prioritized contextual query $[\Phi^+,Q]$ where $(\Pi\cup\Phi^+)$ is a non-contradictory set.*

3.2 Extended and combined contextual queries

Extending contextual queries to consider not only facts but also defeasible rules is straightforward, because (as stated in Observation 1) if (Π,Δ) is a DeLP-program then $(\Pi,(\Delta \cup \Delta_1))$ is also a DeLP-program for any set of defeasible rules Δ_1. The next example shows why this kind of queries are useful.

Example 7. Let $\Pi_7 = \{f, h\}$ and $\Delta_7 = \{(a \mathrel{-\!\!\prec} g), (\sim\!d \mathrel{-\!\!\prec} h, g)\}$, from (Π_7, Δ_7) the answer for the DeLP-query a is UNDECIDED, and the answer for $\sim\!d$ is UNDECIDED. Consider an agent that has the DeLP-program $(\{g\}, \{d \mathrel{-\!\!\prec} a\})$ as part of its private knowledge. From (Π_7, Δ_7) the answer for $[\{g\},a]$ will be (YES,$\{\}$). Then, since the agent has the defeasible rule $(d \mathrel{-\!\!\prec} a)$ it could assume that if a is warranted then d will be warranted. However, from $(\Pi_7 \cup \{g\}, \Delta_7 \cup \{d \mathrel{-\!\!\prec} a\})$ the answer for d is NO, because there is a proper defeater that attacks d.

Definition 17 (Extended contextual query). *Given a DeLP-program (Π,Δ), an extended contextual query for (Π,Δ) is a pair $[\mathcal{P},Q]$ where Q is a DeLP-query and $\mathcal{P}$=(Φ,Δ_1) is a DeLP-program such that $\Pi\cup\Phi$ is a non-contradictory set.*

Definition 18 (Warrant for an extended contextual query). *The query $[(\Phi,\Delta_1),Q]$ is warranted from (Π,Δ) if Q is warranted from $(\Pi\cup\Phi,\ \Delta\cup\Delta_1)$.*

The definition of answer for $[(\Phi,\Delta_1),Q]$ will be the same as Def. 7 where Δ is replaced by $\Delta\cup\Delta_1$. Thus, a contextual query $[\Phi,Q]$ is a particular case of an extended contextual query $[(\Phi,\{\}),Q]$, where the set of defeasible rules is empty.

Extending non-prioritized, restrictive and prioritized contextual queries to include defeasible rules is straightforward. The corresponding definitions will not be included because they are analogous to the previous ones.

Although several types of contextual queries have been defined, there are some modifications to the program in the server that cannot be performed (see Ex. 8). Therefore, we propose a new type of query that is a generalization of the previous ones: the *combined contextual query.* This type of query will allow an application of successive changes to a DeLP-program.

Example 8. Consider the DeLP-program (Π_8, Δ_8), where $\Pi_8 = \{a, b\}$ and $\Delta_8 = \{(z \mathrel{-\!\!\prec} y), (y \mathrel{-\!\!\prec} \sim\!b), (\sim\!y \mathrel{-\!\!\prec} a), (\sim\!z \mathrel{-\!\!\prec} \sim\!a)\}$ Here, the answer for $[\{\sim\!b\}^+,z]$ is (UNDECIDED,$\{b\}$) because the argument $\{(z \mathrel{-\!\!\prec} y), (y \mathrel{-\!\!\prec} \sim\!b)\ \}$ has $\{\sim\!y \mathrel{-\!\!\prec} a\}$ as a defeater. Suppose that an agent wants to add $\sim\!b$ but to simultaneously remove a in order to "deactivate" the mentioned defeater. If it uses first $[\{\sim\!b\}^+,z]$ and then $[\{a\}^-,z]$, the literal $\sim\!b$ will not be part of the program when the second query is processed. Inverting the order of the queries does not solve the problem because a will be present for the second query. On the other hand, if it submits $[\{\sim\!b, \sim\!a\}^+,z]$, the query will remove a but the literal $\sim\!a$ will be added (see Obs. 5). Therefore, the answer for $[\{\sim\!b, \sim\!a\}^+,z]$ will be (UNDECIDED,$\{b, a\}$) because the argument $\{\sim\!z \mathrel{-\!\!\prec} \sim\!a\}$ defeats $\{(z \mathrel{-\!\!\prec} y), (y \mathrel{-\!\!\prec} \sim\!b)\}$.

Definition 19 (Combined contextual query).
Given a DeLP-program $\mathcal{P}$=(Π,Δ), a combined contextual query for $\mathcal{P}$ is a pair $[(Seq,\ \Delta_1),Q]$ where Q is a DeLP-query, Δ_1 is a set of defeasible rules and Seq is a sequence of non-contradictory sets of literals, marked with $*$, $+$ or $-$.

Thus, a combined contextual query allows to apply several types of changes to a program. This resolves issues like the one mentioned in Ex. 8. For example, $[([\{a\}^-, \{\sim b\}^+], \{\}), z]$ will modify (Π_8, Δ_8) by removing a, and then adding $\sim b$ and removing b, before computing the DeLP-query z over the modified program.

The effect of performing a combined contextual query $[(Seq,\ \Delta_1), Q]$ to a DeLP-program (Π, Δ) will be to consider the changes defined in the sequence Seq, apply them in the given order, and then compute the answer for the DeLP-query Q from the modified program. The program will be modified according to Definitions 9, 12 and 14.

Example 9. Consider the program in Example 8, if we perform the combined query $[([\{a, \sim b\}^-, \{x\}^*, \{\sim b\}^+], \{a \prec x\}), z]$, then a and $\sim b$ will be removed from Π_8 ($\sim b \notin \Pi_8$, so its exclusion has no effect); next, x is added to Π_8 (since $\sim x \notin \Pi_8$); finally, $\sim b$ is added to Π_8 and, collaterally, b is dropped from Π_8. Therefore, the program over which query z will be performed is (Π_9, Δ_9), where $\Pi_9 = \{x, \sim b\}$ and $\Delta_9 = \{(z \prec y),\ (y \prec \sim b),\ (\sim y \prec a), (\sim z \prec \sim a),\ (a \prec x)\}$. Here, there is one argument for z: $\{(z \prec y),\ (y \prec \sim b)\}$, which is defeated by an argument for $\sim y$: $\{(\sim y \prec a),\ (a \prec x)\}$. Hence, the answer for $[([\{a, \sim b\}^-, \{x\}^*, \{\sim b\}^+], \{a \prec x\}), z]$ is (UNDECIDED,$\{a, b\}$).

4 Usage and Implementation

Several DeLP-Servers can be used simultaneously, each of them providing a different shared knowledge base. Hence, agents may perform the same contextual query to different servers, and they may share different knowledge with other agents through different servers. Thus, several configurations of agents and servers can be established (statically or dynamically). One possible configuration is to have a single DeLP-Server with all the public knowledge of the system loaded in it. This has the advantage of having the shared knowledge concentrated in one point, but it has the disadvantages of any centralized configuration. In a MAS with many client agents, several copies of the same DeLP-Server (providing the same knowledge) can be created, and each agent could select a different server for sending its queries. Having several copies of the same DeLP-Server is also useful in the case of time-consuming queries, because one agent may distribute or parallelize its own queries.

Since public knowledge will not be restricted to be in a unique repository, then it can be structured in many ways. For example, several servers with different shared knowledge can be created. Thus, special-purpose DeLP-Servers can be implemented, each of them representing knowledge of a specific domain. The MAS may then have several groups of agents with specific tasks, and each group can share specific knowledge through each special-purpose DeLP-Server.

Observe that, by using extended contextual queries, an agent may send a complete DeLP-program (Π, Δ) to a server in order to obtain answers for its queries. Thus, in a MAS it will be possible to have some DeLP-Servers with no public knowledge stored, which will receive pairs $[(\Pi, \Delta),\ Q]$ and will return the answer for Q from (Π,Δ). This particular kind of servers will act only as reasoning services without using public knowledge. As mentioned above, having more than one copy of this particular server without pre-loaded knowledge will be useful for one agent to distribute or parallelize several queries.

It is important to note that, in our model, the public program stored in a DeLP-server is loaded upon server creation and it remains unaffected thereafter. That is, a client agent cannot make permanent changes to it. Our design choice differs from other approaches that use a public shared space for storing knowledge (as a blackboard or a tuple space) where every agent has permission to change it. If client agents in our model were allowed to make permanent changes to the public knowledge stored in a DeLP-server, then the model would collapse to a shared memory repository, and several issues would have to be addressed. For example, a permission model would have to be defined because one agent can change information that is needed by other agents, a policy for updating information that changes dynamically could be needed, etc.

The DeLP-Server was implemented as a stand-alone program that runs in a host and interacts with client agents using the TCP protocol. It supports the connection of several agents, which can be either in the same host, or in different ones. Furthermore, an agent may connect to several DeLP-Servers, and send queries to all of them. The DeLP-Server is implemented in SWI-PROLOG [5] and client agents can be written in any language that supports TCP connections.

5 Conclusions and Related Work

In this paper we propose a model that allows agents to deliberate using defeasible argumentation, to share knowledge with other agents, and to represent individual knowledge in a private manner. We have considered the design and implementation of a Defeasible Logic Programming Server that handles queries from several client agents distributed in remote hosts. Queries are answered using public knowledge stored in the Server and individual knowledge that client agents can send as part of a query, providing a particular context for it. Different types of contextual queries were presented and analyzed.

Our approach was in part inspired by Jinni/Bin-Prolog [6], where several servers can be created, and knowledge can be shared through them. Nevertheless, both approaches have several differences. Their knowledge representation language and their reasoning formalism is PROLOG, whereas we use DeLP. In contrast with us, they share knowledge through blackboards where agents can make permanent changes. Finally, we provide several types of contextual queries, whereas they use PROLOG queries and Linda operations.

In [7] a description of a light-weight Java-based argumentation engine is given that can be used to implement a non-monotonic reasoning component over the Internet or agent-based applications. In contrast with us, they provide an application programming interface (API) that exposes key methods to allow an agent or Internet application developer to access and manipulate the knowledge base to construct rules, specify and execute queries and analyze results. Therefore, their engine allows the agent or Internet application developer to change the program or knowledge base. This capability, in a scenario where several agents try to access to the same knowledge base requires that several issues have to be considered (*e.g.*, synchronization and access/update permissions). Another difference with our approach is that their engine is implemented in Java and presented as a self-contained component that can be integrated into applications. In contrast, a DeLP-Server is an independent stand-alone program that runs in a host and interact with agents using the TCP protocol. In [7], it is not clear what are the possible answers that an agent will obtain from the argumentation engine.

In [8], a query answering system employing a backward-chaining approach is introduced. This query answering system, Deimos, is a suite of tools that supports Defeasible Logic [9, 10]. Although related, Defeasible Logic differs from Defeasible Logic Programming (see [1] for a comparison of both approaches). Similar to us, the system is prepared to receive and compute queries using rules representing defeasible knowledge. In contrast with us, the query answering system is accessible through a command line interface and a CGI interface. Thus, it must be used by a human being, whereas, in our approach, a software agent or any kind of application is allowed to interact with the Server, including a human being through a proper program interface. Another difference with our approach is that in Deimos there is no notion of argument. In order to decide between two contradictory conclusions, Deimos compares only one pair of rules, whereas in DeLP the two arguments that support those conclusions are compared. Comparing only a pair of rules may be problematic as we show in [1].

References

1. García, A.J., Simari, G.R.: Defeasible logic programming: An argumentative approach. Theory and Practice of Logic Programming **4**(1) (2004) 95–138
2. Chesñevar, C.I., Maguitman, A.G., Loui, R.P.: Logical Models of Argument. ACM Computing Surveys **32**(4) (2000) 337–383
3. Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In: Handbook of Philosophical Logic, 2nd ed. Kluwer Academic Pub. (2000)
4. Stolzenburg, F., García, A.J., Chesñevar, C.I., Simari, G.R.: Computing generalized specificity. Journal of Aplied Non-Classical Logics **13**(1) (2003) 87–113
5. (SWI-Prolog) http://www.swi−prolog.org.
6. Tarau, P.: (Jinni 2002: A High Performance Java and .NET based Prolog for Object and Agent Oriented Internet Programming.) http://www.cs.unt.edu/∼tarau/.
7. Bryant, D., Krause, P.J., Vreeswijk, G.: Argue tuProlog: A Lightweight Argumentation Engine for Agent Applications. In: Proc. of 1st Int. Conference on Computational Models of Argument (COMMA06), IOS Press (2006) 27–32
8. Maher, M., Rock, A., Antoniou, G., Billington, D., Miller, T.: Efficient defeasible reasoning systems. Int. Journal on Artificial Intell. Tools **10**(4) (2001) 483–501
9. Nute, D.: Defeasible logic. In: Handbook of Logic in Artificial Intelligence and Logic Programming. Vol 3. Oxford University Press (1994) 355–395
10. Governatori, G., Maher, M., Antoniou, G., Billington, D.: Argumentation semantics for defeasible logic. Journal of Logic and Computation **14** (2004) 675–702