An Argument-Based Multi-Agent System for Information Integration

Marcela Capobianco and Guillermo R. Simari

Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur – Av. Alem 1253, (8000) Bahía Blanca Argentina
Email: {*mc,grs*}*@cs.uns.edu.ar*

Abstract. In this paper we address the problem of obtaining a consolidated view of the knowledge that a community of information agents possesses in the form of private, possibly large, databases. Each agent in the community has independent sources of information and each database could contain information that is potentially inconsistent and incomplete, both by itself and/or in conjunction with some of the others. These characteristics make the consolidation difficult by traditional means. The idea of obtaining a single view is to provide a way of querying the resulting knowledge in a skeptical manner, *i.e.*, receiving one answer that reflects the perception of the information community.
Agents using the proposed system will be able to access multiple sources of knowledge represented in the form of deductive databases as if they were accessing a single one. One application of this schema is a novel architecture for decision-support systems (DSS) that will combine database technologies, specifically federated databases, which we will cast as information agents, with an argumentation-based framework.

Categories and Subjects Descriptors: I.2.4 [Artificial Intelligence]: Knowledge Representation Formalisms and Methods—Representation languages, Representations (procedural and rule-based); H.1.0 [Models and Principles]: Systems and Information Theory—General systems theory.
General Terms: Algorithms, Design, Performance.
Keywords: Argumentation, Knowledge representation, Design languages for agent systems.

1 Introduction

Information systems, and the capability to obtain answers from them, play a key role in our society. In particular, data intensive applications are in constant demand and there is need for computing environments with much more intelligent capabilities than those present in today's Data-base Management Systems (DBMS). The expected requirements for these systems change every day: they constantly become more complex and more advanced features are demanded from them.

In this paper we address the problem of obtaining a consolidated view of the knowledge that a community of information agents possess in the form of private, possibly large, databases. Each agent in the community has independent sources of information and each database could contain information that is potentially inconsistent and incomplete, both by itself and/or in conjunction with some of the others. These characteristics make the consolidation difficult by traditional means. The idea of obtaining a single view is to provide a way of querying the resulting knowledge in a skeptical manner, *i.e.*, receiving one answer that reflects the perception of the information community.

Agents using the proposed system will be able to access multiple sources of knowledge represented in the form of deductive databases as if they were accessing a single one. One application of this schema is a novel architecture for decision-support systems (DSS) that will combine database technologies, specifically federated databases, which we will cast as information agents, with an argumentation-based framework.

Recently, there has been much progress in developing efficient techniques to store and retrieve data, and many satisfactory solutions have been found for the associated problems. However it remains an open problem how to understand and interpret large amounts of information. To do this we need specific formalisms that can perform complicated inferences, obtain the appropriate conclusions, and justify their results. We claim that these formalisms should also be able to access seamlessly databases distributed over a network.

In the field of deductive databases there has been a continued effort to produce an answer to this problem. Deductive databases store explicit and implicit information; explicit information is stored in the manner of a relational database and implicit information is recorded in the form of rules that enable inferences based on the stored data. These systems combine techniques and tools from relational databases with rule based formalisms. Hence, they are capable of handling large amounts of information and perform some sort of reasoning based on it. Nevertheless, these systems have certain limitations and shortcomings for knowledge representation and modeling commonsense reasoning, especially for managing incomplete and potentially contradictory information, as argued by several authors [17, 23, 16].

Argumentation frameworks [9, 19, 14] are an excellent starting point for building intelligent systems with interesting reasoning abilities. Research in argumentation has provided important results while striving to obtain tools for commonsense reasoning, and this prompted a new set of argument-based applications in diverse areas where knowledge representation issues play a major role [10, 5, 7].

We believe that deductive databases can be combined with argumentation formalisms to obtain interactive systems able to reason with large databases, even in the presence of incomplete and potentially contradictory information. This can be a significant advantage with respect to systems based on logic programming, such as datalog, that cannot deal with contradictory information.[1] In particular, this could be useful in contexts where information is obtained from multiple databases, and these databases may be contradictory among themselves.

The multi-agent system introduced here virtually integrates different databases into a common view; in that manner users of this system can query multiple databases as if they were a single one. This schema can be applied to obtain a novel system architecture for decision-support systems (DSS) that combines database technologies, specifically federated databases [18], with an argumentation based framework.

In our proposal we consider that information is obtained from a number of different heterogeneous database systems, each represented by a particular agent. The reasoning mechanisms, based on an argumentative engine, use this information to construct a consolidated global view of the database. This task is performed by the reasoning agent, that is based on a set of rules expressed in a particular argumentation framework. This agent can deal with incomplete and contradictory information and can also be personalized for any particular DSS in a relatively simple way.

We have also considered that one of the design objectives of interactive systems is that they can respond in a timely manner to users' queries. So far the main objection to the use of argumentation in interactive systems is their computational complexity. In previous work [6] we have addressed the issue of optimizing argumentation systems, where the optimization technique consisted in using a precompiled knowledge component as a tool to allow significant speed-ups in the inference process. We also apply this technique in our reasoning agent.

To understand the advantages of the proposed reasoning mechanism used in our multiagent system, consider a set of databases used by the employers responsible of drug administration, sales, and delivery in a given hospital. These databases contains information regarding drugs, patients, known allergies, and addictions. Suppose a deductive database system in the style of datalog is used to query this information to derive certain relations. In this setting, there is a rule establishing that a drug should be given to a patient if the patient has a prescription for this drug signed by a physician. There could also be a rule saying that the drug should not be sold if the prescription is signed by the patient. In this case, if Dr. Gregory House enters the clinic with a prescription signed by himself to get Vicodin, the employers could query the system to see if the drug should or should not be sold. If a traditional deductive database is used, in the style of datalog or another logic programming based system, this would give raise to a contradiction and the system would not be able to recommend a course of action. Using our system, an argument can be built backing that the medication should be sold, given that there is a prescription signed by a doctor that warrants it. However, an argument for not selling the drug could also be built considering that the doctor and the patient are the same person. Our

[1] Some extensions of datalog handle negation using CWA (see [8]), but these approaches do not allow negation in the head of the rules in the style of extended logic programming.

argument-based framework can then compare both arguments, decide that the second argument is preferred, and answer the query saying that the drug should no be sold. In addition, it can also explain the reasons that back its answer. Note that this kind of behavior cannot be obtained in Datalog-like systems.

The rest of this article is organized as follows. Section 2 sums up related work, Section 3 contains our proposal for the system architecture, and section 4 formally describes the reasoning module, a key component of this architecture. Section 5 presents a key optimization for the argumentation-based reasoning process, and Section 6 shows a realistic example of the system's mechanics. Finally, Section 7 states the conclusions.

2 Integrating DBMS and Reasoning Systems

Previous work on the integration of databases and reasoning systems has almost been restricted to coupling Prolog interpreters and relational databases. These approaches were motivated in the declarative nature of logic programming languages and the data-handling capabilities of database systems. Several researchers have built intelligent database systems coupling Prolog and a relational DBMS or extending Prolog with database facilities [8]. These works were motivated by the fact that Prolog attracted attention in the 80's for its ability to support rapid development of complex applications. Besides, Prolog is based on Horn Clauses that are close relatives of database query languages and its language is more powerful than SQL [24].

Brodie and Jarke [4] envisioned several years ago that large scale data processing would require more efficient and more intelligent access to databases. He proposed the integration of logic programming and databases to meet future requirements. First, he identified two different approaches for coupling a Prolog interpreter and a Relational DBMS, which are usually called "tight coupling' and "loose coupling". In the tight coupling approach the Prolog interpreter and the Relational DBMS are strongly integrated. For example, the Prolog interpreter can directly use low level functionalities of the DBMS, like relation management in secondary memory, and relation access via indexes [12]. In contrast, in the loose coupling approach, the Relational DBMS is called by the Prolog interpreter at the top level, that acts like a standard user. It sends Relational queries to the DBMS, and the corresponding answers are treated as ground clauses by the interpreter.

Brodie and Jarke also identified four basic architectural frameworks for combining Prolog and a database system:

- Loose coupling of an existing Prolog implementation to an existing relational database system;
- Extending Prolog to include some facilities of the relational database system;
- Extending an existing relational database to include some features of Prolog;
- Tightly integrating logic programming techniques with those of relational database systems.

They recommend the fourth alternative (tight integration), based on the belief that a special purposed language for large scale knowledge base systems would best address issues regarding performance, knowledge representation and software engineering. They also put forward a number of issues concerning the best division of tasks between logic programming and a DBMS.

Zaniolo [25] proposed an approach to intelligent databases based on deductive databases. He advocates for elevating database programming to the more abstract level of rules and knowledge base programming to create an environment more supportive of the new wave of database applications. To achieve these goals the *LDL*/*LDL+* project was developed. During the project a new logic-based language was designed along with the definition of its formal semantics, new implementation techniques were developed for the efficient support of rule-based logic languages, and it was successfully used in a wide range of application domains. The system supported an open architecture and SQL schema from external databases could be incorporated into the *LDL* program seamlessly.

In the following section we present the system architecture for our proposal. We believe that argumentation can offer a new perspective into the problem of reasoning with large databases, giving more expressive power to the reasoning component, making it able to decide even in the presence of uncertain and/or contradictory information. This addresses a limitation that was present in each of the deductive database systems considered in this section.

3 System Architecture

In this section we present an architectural pattern for our multiagent system that can be applied to design information-based applications where a certain level of intelligence is required. Such applications will be engineered for contexts where: (1) information is uncertain and heterogeneous, (2) handling of great volume of data flows is needed, and (3) data may be incomplete, vague, or contradictory. These applications are also expected to integrate multiple information systems such as databases, knowledge bases, source systems, etc.

Our system architecture is presented in Figure 1. The architecture is modular and is independent of any particular domain or application. We have used a layered architectural style, where every layer provides a series of services to the one above. The first of our layers concerns data and knowledge acquisition. This layer will receive heterogeneous sources of data and will extract and transform this data into the formats required of the particular application. It can work with diverse sources, such as laboratory data, different types of sensors, knowledge bases, etc.

The received data will be formatted to comply with the relational models provided by a group of federated databases that share a common export schema. In our system, each one of the databases is represented by an agent. The common export schema will be the result of a negotiation process among these agents. The union of the views of these databases will generate a key element of our framework, the extensional database that contains the information needed for the

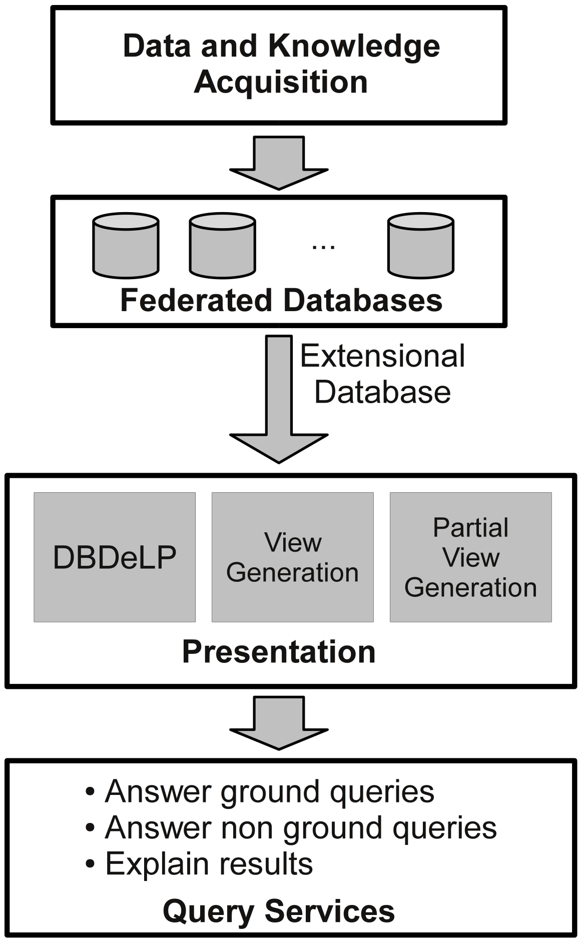

Fig. 1. Proposed architectural pattern

reasoning module. The extensional database also will provide the data elements with a certainty degree that depends of the credibility of the data source from where it was obtained.

We have chosen to use a multi-source perspective for the characterization of data quality [3]. In this case, the quality of data can be evaluated by comparison with the quality of other homologous data (i.e., data from different information sources which represent the same reality but may have contradictory values). The approaches usually adopted to reconcile heterogeneity between values of data are: (1) to prefer the values of the most reliable sources, (2) to mention the source ID for each value, or (3) to store quality meta-data with the data.

For our proposed architecture, we have used the second approach. In multi-source databases, each attribute of a multiple source element has multiple values with the ID of their source and their associated *Quality of Expertise*, which is represented as meta-data associated with each value, such as a given certainty degree. This degree may be obtained weighting the plausibility of the data value, its accuracy, the credibility of its source, and the freshness of the data.

The federated database layer provides the extensional database to the presentation layer. The extensional database can be computed on demand and is not necessarily stored in a physical component. The presentation layer contains the services related with the reasoning process and its optimization. This is the core of our proposal and will be described later on in Sections 4 and 5. The reasoning agent that generated the consolidated view is part of this layer. It contains the set of rules that encode the specific knowledge of the application. These rules will be used by the argumentation-based inference engine. The presentation layer also commands the generation of the extensional database, and the selection if it is going to be done on demand (following a lazy approach) or if it has to be computed completely. It can also generate a partial view of the system according to these rules, resulting in an optimization mechanism. This partial view depends only on the set of rules and must be changed accordingly if changes on the rules are produced. Finally, the query services layer is composed by an interactive agent that receives user queries, provides answers, and can also explain the reasons backing these answers.

4 The DB_DeLP Argumentation Framework

In this section we formally define the argumentation system that is used by the reasoning agent in the Query Services Layer of the proposed system architecture. Here we detail the semantics and proof theory of the framework and we also show some practical examples. A simplified view of our system would describe it as a deductive database whose inference engine is based on a specialization of the DeLP language [15]. This particular framework will be known as *Database Defeasible Logic Programming* (DB_DeLP). Formally, DB_DeLP is a language for knowledge representation and reasoning that uses *defeasible argumentation* to decide between contradictory conclusions through a *dialectical analysis*. DB_DeLP also incorporates uncertainty management, taking elements from Possibilistic Defeasible Logic Programming (P_DeLP) [2, 1], an extension of DeLP in which the elements of the language have the form (ϕ, α), where ϕ is a DeLP clause or fact and α expresses a lower bond for the certainty of ϕ in terms of a necessity measure. Conceptually, our deductive database consists of an extensional database **EDB**, an intensional database **IDB**, and a set of integrity constrains **IC**. In what follows, we formally define these elements.

The language of DB_DeLP follows a logic programming style. Standard logic programming concepts (such as signature, variables, functions, *etc.*) are defined in the usual way. Literals are atoms that may be preceded by the symbol "∼" denoting *strict* negation, as in extended logic programming.

Definition 1. [Literal–Weighted Literal] *Let Σ be a signature, then every atom A of Σ is a positive literal, while every negated atom $\sim A$ is a negative literal. A* literal *of Σ is a positive literal or a negative literal. A* certainty weighted literal*, or simply a weighted literal, is a pair (L, α) where L is a literal and $\alpha \in [0, 1]$ expresses a lower bound for the certainty of L in terms of a necessity measure.*

The extensional database **EDB** is composed by a set of certainty weighted literals, according to the export schema of the federated database that is part of our architecture. Conceptually, it accounts for the union of the views of every particular database that belongs to the federation [18]. When implementing the system, this set of ground literals may not be physically stored in any place, and may simply be obtained on demand when information about a particular literal is needed.

The certainty degree associated with every literal is assigned by the federated database layer that assigns a particular degree to every data source according to its plausibility. The resulting extensional database is not necessarily consistent, in the sense that a literal and its complement w.r.t. strong negation may both be present, with different or the same certainty degrees. In this case, the system decides according to a given criterion which fact will prevail and which one will be removed from the view.

`species(X,Y)`	`age(X,Y)`
(`species(simba,lion)`, 0.6)	(`age(simba,young)`, 0.65)
(`species(mufasa,lion)`, 0.7)	(`age(mufasa,old)`, 0.7)
(`species(grace,lion)`, 0.6)	(`age(grace,adult)`, 0.8)
(`species(grace,leopard)`, 0.4)	(`age(dumbo,baby)`, 0.8)
...	...

Fig. 2. An Extensional Database in DB_DeLP

The intensional part of a DB_DeLP database is formed by a set of *defeasible rules* and *integrity constraints* Defeasible rules provide a way of performing tentative reasoning as in other argumentation formalisms [9, 19, 14].

Definition 2. [Defeasible Rule] *A* defeasible rule *expresses a tentative, weighted, relation between a literal L_0 and a finite set of literals $\{L_1, L_2, \ldots, L_k\}$. It has the form $(L_0 \prec L_1, L_2, \ldots, L_k, \alpha)$ where $\alpha \in [0, 1]$ expresses a lower bound for the certainty of the rule in terms of a necessity measure.*

In previously defined argumentation systems, the meaning of defeasible rules $L_0 \prec L_1, L_2, \ldots, L_k$ was understood as *"$L_1, L_2, \ldots, L_k$ provide tentative reasons to believe in L_0"* [22], but these rules did not have an associated certainty degree. In contrast, DB_DeLP adds the certainty degree, that expresses how strong is the connection between the premises and the conclusion. A defeasible rule with a certainty degree 1 will model a strong rule. Figures 2 and 3 show an extensional and an intensional database in our formalism.

```
(feline(X) —< species(X,lion),1)
(climbs_tree(X) —< feline(X),0.65)
(~climbs_tree(X) —< species(X,lion),0.70)
(climbs_tree(X) —< species(X,lion), age(X,young).,0.75)
(~climbs_tree(X) —< sick(X),0.45)
```

Fig. 3. An Intensional Database in DB_DeLP

Note that DB_DeLP programs are *range-restricted*, a common condition for deductive databases: a program is said to be range-restricted if and only if every variable that appears in the head of the clause also appears in its body. This implies that all the facts in the program must be ground (cannot contain variables). These programs can be interpreted more efficiently since full unification is not needed, only matching that is significantly more efficient. Nevertheless, the reason for this decision comes from a semantic standpoint, given that a defeasible reason in which there is no connection between the head and the body has no clear meaning; the range restriction ensures this connection.

Integrity constraints are rules of the form $L \leftarrow L_0, L_1, \ldots, L_n$ where L is a literal, and $L_0, L_1, \ldots, L_k$ is a non-empty finite set of literals, These rules are used to compute the *derived negations* as follows. For the extensional and intensional databases regarding felines, consider that the set of integrity constraints is composed by {`~leopard(X)` ← `lion(X)`, `~lion(X)` ← `leopard(X)`} and the negations { (`~species(grace,lion)`, 0.4), (`~species(grace,leopard)`, 0.6)} are then added to the extensional database. The certainty degree of the added rule is calculated as the minimum of the certainty degree of the literals that are present in the body of the integrity constraint rule used to obtain it. Note that a conflict may arise with information received from other knowledge bases, since we may want to add a literal and its complement may be already present in the extensional database. Then the system will decide according to a given criterion which fact will prevail and which one will be removed from the view. A standard criterion in this case would be using the plausibility of the source, the certainty degree of the literals, or a combination of both. Databases in DB_DeLP, for short called *defeasible databases*, can also include built-in predicates as needed along with their corresponding axioms.

The P_DeLP language [11], which presented the novel idea of mixing argumentation and possibilistic logic, is based on Possibilistic Gödel Logic or PGL [2, 1], which is able to model both uncertainty and fuzziness and allows for a partial matching mechanism between fuzzy propositional variables. In DB_DeLP, for simplicity reasons, we will avoid fuzzy propositions, and hence it will be based on the necessity-valued classical Possibilistic logic [13]. As a consequence, possibilistic models are defined by possibility distributions on the set of classical interpretations, and the proof theory for our formulas, written $\mid\!\sim$, is defined by derivation based on the following instance of the Generalized Modus Ponens rule (GMP): $(L_0 \prec L_1 \wedge \cdots \wedge L_k, \gamma), (L_1, \beta_1), \ldots, (L_k, \beta_k) \vdash (L_0, \min(\gamma, \beta_1, \ldots, \beta_k))$, which is a particular instance of the well-known possibilistic resolution rule, and which provides the *non-fuzzy* fragment of DB_DeLP with a complete calculus for determining the maximum degree of possibilistic entailment for weighted literals. Literals in the extensional database are the base case of the derivation sequence; for every literal Q in **EDB** with a certainty degree α it holds that (Q, α) can be derived from the corresponding program.

A query presented to a DB_DeLP database is a a ground literal Q which must be supported by an *argument*. Deduction in DB_DeLP is argumentation-based, thus a derivation is not enough to endorse a particular fact, and queries must be supported by arguments. In the following definition *instances(**IDB**)* accounts for any set of ground instances of the rules in **IDB**, replacing free variables for ground literals in the usual way.

Definition 3. [Argument]–[Subargument] *Let $\mathcal{DB} = (\boldsymbol{EDB}, \boldsymbol{IDB}, \boldsymbol{IC})$ be a defeasible database, $\mathcal{A} \subseteq$ instances(**IDB**) is an* argument *for a goal Q with necessity degree $\alpha > 0$, denoted as $\langle \mathcal{A}, Q, \alpha \rangle$, iff:*

1. $\Psi \cup \mathcal{A} \mathrel{|\!\sim} (Q, \alpha)$,
2. $\Psi \cup \mathcal{A}$ *is non contradictory, and*
3. *there is no $\mathcal{A}_1 \subset \mathcal{A}$ such that $\Psi \cup \mathcal{A}_1 \mathrel{|\!\sim} (Q, \beta)$, $\beta > 0$.*

An argument $\langle \mathcal{A}, Q, \alpha \rangle$ is a subargument *of $\langle \mathcal{B}, R, \beta \rangle$ iff $\mathcal{A} \subseteq \mathcal{B}$.*

Arguments in DB_DeLP can attack each other; this situation is captured by the notion of *counterargument.* An argument $\langle \mathcal{A}_1, Q_1, \alpha \rangle$ *counter-argues* an argument $\langle \mathcal{A}_2, Q_2, \beta \rangle$ at a literal Q if and only if there is a sub-argument $\langle \mathcal{A}, Q, \gamma \rangle$ of $\langle \mathcal{A}_2, Q_2, \beta \rangle$, (called *disagreement subargument*), such that Q_1 and Q are complementary literals. Defeat among arguments is defined combining the counterargument relation and a preference criterion "$\preceq$". This criterion is defined on the basis of the necessity measures associated with arguments.

Definition 4. [Preference criterion $\succeq$][11] *Let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ be a counterargument for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$. We will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is* preferred *over $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ (denoted $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle \succeq \langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$) iff $\alpha_1 \geq \alpha_2$. If it is the case that $\alpha_1 > \alpha_2$, then we will say that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is* strictly preferred *over $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$, denoted $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle \succ \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$. Otherwise, if $\alpha_1 = \alpha_2$ we will say that both arguments are* equi-preferred*, denoted $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle \approx \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$.*

Definition 5. [Defeat][11] *Let $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ and $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ be two arguments built from a program $\mathcal{P}$. Then $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$* defeats *$\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ (or equivalently $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ is a* defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$) iff (1) Argument $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ counter-argues argument $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$ with disagreement subargument $\langle \mathcal{A}, Q, \alpha \rangle$; and (2) Either it is true that $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle \succ \langle \mathcal{A}, Q, \alpha \rangle$, in which case $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ will be called a* proper defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$, or $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle \approx \langle \mathcal{A}, Q, \alpha \rangle$, in which case $\langle \mathcal{A}_1, Q_1, \alpha_1 \rangle$ will be called a* blocking defeater *for $\langle \mathcal{A}_2, Q_2, \alpha_2 \rangle$.*

As in most argumentation systems [9, 20], DB_DeLP relies on an exhaustive dialectical analysis which allows to determine if a given argument is *ultimately* undefeated (or *warranted*) w.r.t. a program $\mathcal{P}$. An *argumentation line* starting with an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ is a sequence $[\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle, \langle \mathcal{A}_1, Q_1, \alpha_1 \rangle, \ldots, \langle \mathcal{A}_n, Q_n, \alpha_n \rangle, \ldots]$ that can be thought of as an exchange of arguments between two parties, a *proponent* (even-numbered arguments) and an *opponent* (odd-numbered arguments).

Given a program $\mathcal{P}$ and an argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$, the set of all acceptable argumentation lines starting with $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ accounts for a whole dialectical analysis for $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ (i.e. all possible dialogs rooted in$\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$), formalized as a *dialectical tree* and denoted $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$. Nodes in a dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ can be marked as *undefeated* or *defeated* nodes (U-nodes and D-nodes, resp.). A dialectical tree will be marked as an AND-OR tree: all leaves in $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ will be marked as U-nodes (as they have no defeaters), and every inner node is to be marked as a *D-node* iff it has at least one U-node as a child, and as a *U-node* otherwise. An argument $\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle$ is ultimately accepted as valid (or *warranted*) iff the root of $\mathcal{T}_{\langle \mathcal{A}_0, Q_0, \alpha_0 \rangle}$ is labeled as a *U-node.*

Definition 6. [Warrant][11] *Given a database $\mathcal{DB}$, and a literal Q, Q is warranted w.r.t. $\mathcal{DB}$ iff there exists a warranted argument $\langle \mathcal{A}, Q, \alpha \rangle$ that can be built from $\mathcal{P}$.*

Example 1. Suppose the system has to solve the query *climbs(simba).* Then argument

$$\mathcal{A}_2 = \{(\textit{climbs(simba)} \prec \textit{feline(simba)}, 0.65), (\textit{feline(simba)} \prec \textit{species(simba,lion)}, 1)\}$$

must be built. This argument has a certainty degree of 0.6, taking into account the certainty degree of the literals on which the deduction is founded.

Next, the system looks for the defeaters. The only defeater is:

$$\langle \mathcal{A}_4, {\sim}\textit{climbs(simba)}, 0.6 \rangle, \mathcal{A}_4 = \{({\sim}\textit{climbs(simba)} \prec \textit{species(simba,lion)}, 0.75)\}$$

But this argument is in turn defeated by $\langle \mathcal{A}_3, \textit{climbs(simba)}, 0.6 \rangle$,

$$\mathcal{A}_3 = \{(\textit{climbs(simba)} \prec \textit{species(simba,lion),age(simba,young)}, 0.75)\}$$

Thus, *climbs(simba)* is warranted.

5 Optimization of DB_DeLP's Dialectical Process

To obtain faster query processing in the DB_DeLP system we integrate pre-compiled knowledge to avoid the construction of arguments which were already computed. The approach follows the proposal presented in [6] where the pre-compiled knowledge component is required to: (1) minimize the number of stored arguments in the pre-compiled base of arguments (for instance, using one structure to represent the set of arguments that use the same defeasible rules); and (2) maintain independence from the observations that may change with new perception in order to avoid modifying also the pre-compiled knowledge when new observations are incorporated.

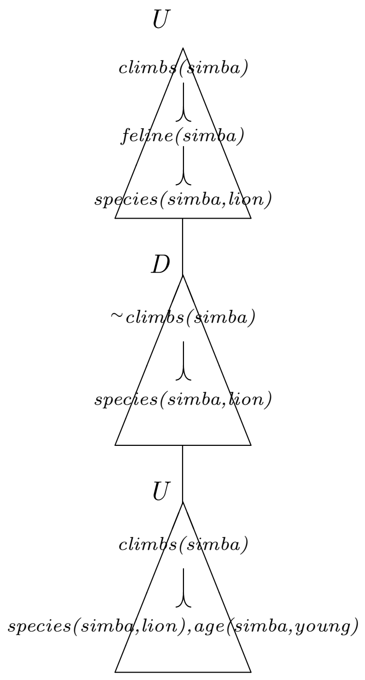

Fig. 4. Dialectical tree from Example 1

Considering these requirements, we define a database structure called *dialectical graph*, which will keep a record of all possible *arguments* in an DB_DeLP database $\mathcal{DB}$ (by means of a special structure named potential argument) as well as the counterargument relation among them. Potential arguments, originally defined in [6], contain non-grounded defeasible rules, thus depending only on the set of rules in the **IDB**, *i.e.*, they are independent from the extensional database.

Potential arguments have been can be thought as schemata that sum up arguments that are obtained using *different* instances of the *same* defeasible rules. Recording every generated argument could result in storing many arguments which are structurally identical, only differing on the constants being used to build the corresponding derivations. Thus, a potential argument stands for several arguments which use the same defeasible rules. Attack relations among potential arguments can be also captured, and in some cases even defeat can be pre-compiled. In what follows we introduce the formal definitions, adapted from [6] to fit the DB_DeLP system.

Definition 7. [Weighted Potential Argument] *Let* ***IDB*** *be an intensional database. A subset* A *of* ***IDB*** *is a* potential argument *for a literal* Q *with an upper bound* γ *for its certainty degree, noted as* $\langle\langle \mathsf{A}, Q, \gamma\rangle\rangle$ *if there exists a non-contradictory set of weighted literals* Φ *and an instance* $\mathcal{A}$ *that is obtained by finding an instance for every rule in* A*, such that* $\langle \mathcal{A}, Q, \alpha\rangle$ *is an argument w.r.t. the database with* Φ *as its extensional database and* ***IDB*** *as its intensional database* ($\alpha \leq \gamma$) *and there is no instance* $\langle \mathcal{B}, Q, \beta\rangle$ *of* A *such that* $\beta > \gamma$.

Definition 7 does not specify how to obtain the set of potential arguments from a given database. The interested reader may consult [6] for a constructive definition and its associated algorithm. The calculation of the upper bound γ deserves special mention, since the algorithm in [6] was devised for a different system, without uncertainty management. This element will be used later on to speedup the extraction of the dialectical tree from the dialectical graph for a given query. To calculate γ for a potential argument A we simply choose the lower certainty degree of the defeasible rules present in A.

The nodes of the dialectical graph are the potential arguments. The arcs of our graph are obtained by calculating the counterargument relation among the nodes previously obtained. To do this, we extend the concept of counter-argument for potential arguments. A potential argument $\langle\langle \mathsf{A}_1, Q_1, \alpha\rangle\rangle$ *counter-argues* $\langle\langle \mathsf{A}_2, Q_2, \beta\rangle\rangle$ at a literal Q if and only if there is a non-empty potential sub-argument $\langle\langle \mathsf{A}, Q, \gamma\rangle\rangle$ of $\langle\langle \mathsf{A}_2, Q_2, \beta\rangle\rangle$ such that Q_1 and Q are contradictory literals.[2] Note that potential counter-arguments may or may not result in a real conflict between the instances (arguments) associated with the corresponding potential arguments. In some cases instances of these arguments cannot co-exist in any scenario (*e.g.*, consider two potential arguments based on contradictory observations). Now we can finally define the concept of dialectical graph:

Definition 8. [Dialectical Graph] *Let $\mathcal{DB} = (\boldsymbol{EDB}, \boldsymbol{IDB}, \boldsymbol{IC})$ be a defeasible database. The* dialectical graph *of $\boldsymbol{IDB}$, denoted as $\mathcal{G}_{\boldsymbol{IDB}}$, is a pair*

$$(\mathrm{PotArgs}(\boldsymbol{IDB}), C)$$

such that:

1. $\mathrm{PotArgs}(\boldsymbol{IDB})$ *is the set* $\{\langle\langle A_1, Q_1, \alpha_1\rangle\rangle, \ldots, \langle\langle A_k, Q_k, \alpha_k\rangle\rangle\}$ *of all the potential arguments that can be built from* $\boldsymbol{IDB}$*;*
2. $\mathcal{C}$ *is the counterargument relation over the elements of* $\mathrm{PotArgs}(\boldsymbol{IDB})$.

Example 2. Consider the feline database previously presented; its dialectical graph is composed by:
(`feline(X)` $\prec$ `species(X,lion)`,1)
(`climbs_tree(X)` $\prec$ `feline(X)`,0.65)
(~`climbs_tree(X)` $\prec$ `species(X,lion)`,0.70)
(`climbs_tree(X)` $\prec$ `species(X,lion), age(X,young)`,0.75)
(~`climbs_tree(X)` $\prec$ `sick(X)`,0.45)

– $\langle\langle \mathsf{A}_1, \mathsf{climbs(X)}, 0.65\rangle\rangle$, $\mathsf{A}_1 = \{(\texttt{climbs(X)} \prec \texttt{feline(X)}, 0.65)\}$.
– $\langle\langle \mathsf{A}_2, \mathsf{climbs(X)}, 0.65\rangle\rangle$, $\mathsf{A}_2 = \{(\texttt{climbs(X)} \prec \texttt{feline(X)}, 0.65),$
$(\texttt{feline(X)} \prec \texttt{species(X,lion)}, 1)\}$.
– $\langle\langle \mathsf{A}_3, \mathsf{climbs(X)}, 0.75\rangle\rangle$,
$\mathsf{A}_3 = \{(\texttt{climbs(X)} \prec \texttt{species(X,lion), age(X,young)}, 0.75)\}$.

[2] Note that $P(X)$ and ~$P(X)$ are contradictory literals even though they are non-grounded. The same idea is applied to identify contradiction in potential arguments.

- $\langle\langle \mathsf{A}_4, \sim\mathsf{climbs(X)}, 0.75\rangle\rangle$, $\mathsf{A}_4 = \{(\sim\texttt{climbs(X)} \prec \texttt{species(X,lion)}, 0.75)\}$.
- $\langle\langle \mathsf{A}_5, \sim\mathsf{climbs(X)}, 0.45\rangle\rangle$, $\mathsf{A}_5 = \{(\sim\texttt{climbs(X)} \prec \texttt{sick(X)}, 0.45)\}$.
- $\langle\langle \mathsf{A}_6, \mathsf{feline(X)}, 1\rangle\rangle$, $\mathsf{A}_6 = \{(\texttt{feline(X)} \prec \texttt{species(X,lion)}, 1)\}$.
- $D_p = \{(\mathsf{A}_2, \mathsf{A}_4), (\mathsf{A}_4, \mathsf{A}_3)\}$
- $D_b = \{(\mathsf{A}_1, \mathsf{A}_4), (\mathsf{A}_4, \mathsf{A}_1), (\mathsf{A}_1, \mathsf{A}_5), (\mathsf{A}_5, \mathsf{A}_1), (\mathsf{A}_2, \mathsf{A}_5), (\mathsf{A}_5, \mathsf{A}_2), (\mathsf{A}_3, \mathsf{A}_5), (\mathsf{A}_5, \mathsf{A}_3)\}$.

The relations D_b and D_p can be depicted as shown in Figure 2, where blocking defeat is indicated with a double headed arrow.

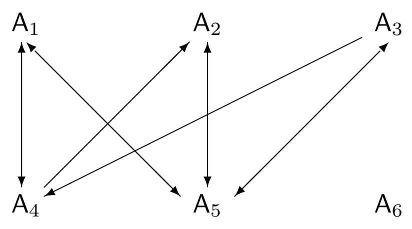

Fig. 5. Dialectical graph corresponding to Example 2.

Having defined the dialectical graph we can now use a specific graph traversing algorithm to extract a particular dialectical tree rooted in a given potential argument. The facts present in the **EDB** will be used as evidence to instantiate the potential arguments in the dialectical graph that depend on the intensional database **IDB**. This gives rise to the inference process of the system. This process starts when a new query is formulated. Consider the dialectical graph in Example 2 and suppose the set of facts in Figure 2 is present in the extensional database. Lets see how the system works when faced with the query *climbs(simba)*.

First, the set of potential arguments in the dialectical graph is searched to see if there exists an element whose conclusion can be instantiated to match the query. It finds $\langle\langle \mathsf{A}_2, \mathsf{climbs(X)}, 0.65\rangle\rangle$,

$$\mathsf{A}_2 = \{(\texttt{climbs(X)} \prec \texttt{feline(X)}, 0.65), (\texttt{feline(X)} \prec \texttt{species(X,lion)}, 1)\}$$

A_2 can be instantiated to

$$\mathcal{A}_2 = \{(\textit{climbs(simba)} \prec \textit{feline(simba)}, 0.65), (\textit{feline(simba)} \prec \textit{species(simba,lion)}, 1)\}$$

that has a certainty degree of 0.6 taking into account the certainty degree of the literals on which the deduction is founded.

Now, to see if *climbs(simba)* is inferred by the system from the intensional and the extensional database, we must check whether $\mathcal{A}_2$ can sustain its conclusion when confronted with its counterarguments. Using the links in the dialectical graph we find one defeater for $\mathcal{A}_2$, instantiating potential argument

$$\mathsf{A}_4 = \{(\sim\texttt{climbs(X)} \prec \texttt{species(X,lion)}, 0.75)\}$$

to

$$\mathcal{A}_4 = \{(\sim \textit{climbs(simba)} \prec \textit{species(simba,lion)}, 0.75)\}$$

The argument $\langle \mathcal{A}_4, \sim\textit{climbs(simba)}, 0.6\rangle$ is defeated by $\langle \mathcal{A}_3, \textit{climbs(simba)}, 0.6\rangle$ (an instantiation of $\langle\langle \mathsf{A}_3, \mathsf{climbs(X)}, 0.75\rangle\rangle$). Thus, *climbs(simba)* is warranted and we found the same dialectical tree that was found in example 1 with an optimized inference mechanism. Note that the links for the defeaters present in the dialectical graph are used to find the conflicts. This makes it easier to recover the tree from the dialectical graph of the framework.

The deductive database can be subject to constant changes as is the case with every real world database. The only restriction is that it must not be changed while a query is being solved. The dialectical graph is not affected by changes in the extensional database.

We present now a classic example in traditional deductive database systems based on logic programming, that usually causes problems with the semantics. In our case the system follows a cautious semantics, not deriving either $p(a)$ or $q(a)$.

Example 3. Consider a deductive database composed by:

- **EDB** $= \{(r, 0.6), (s, 0.6)\}$,
- **IDB** $= \{(\mathtt{p(X)} \prec \sim\mathtt{q(X)}, 0.8), (\mathtt{q(X)} \prec \sim\mathtt{p(X)}, 0.8)\}$

The dialectical graph $\mathbf{G}_{IDB}$ is composed by the two potential arguments:

- $\langle\langle \mathsf{A}_1, \mathsf{p(X)}, , \rangle\rangle$ $\mathsf{A}_1 = \{(\mathtt{p(X)} \prec \sim\mathtt{q(X)}, 0.8)\}$.
- $\langle\langle \mathsf{A}_2, \mathsf{q(X)}, , \rangle\rangle$ $\mathsf{A}_1 = \{(\mathtt{q(X)} \prec \sim\mathtt{p(X)}, 0.8)\}$.

and the defeat relation $D_b = \{(A_1, A_2), (A_2, A_1)\}$.

Suppose the system is faced with the query $p(a)$. The dialectical tree for this query is formed by argument $\langle \mathcal{A}_1, \sim\textit{q(a)}, 0.6\rangle$, $\mathcal{A}_1 = \{(\mathtt{p(a)} \prec \sim\mathtt{q(a)}, 0.6)\}$ that is in turn defeated by $\langle \mathcal{A}_2, \textit{q(a)}, 0.6\rangle$, $\mathcal{A}_1 = \{(\mathtt{q(a)} \prec \sim\mathtt{p(a)}, 0.6)\}$.

The situation with query q(a) is analogous and therefore the system cannot derive $p(a)$ nor $q(a)$.

Note that the DB_DeLP system can seamlessly treat this example without semantic or operational problems. Furthermore, there is no need for imposing additional restrictions, such as requiring predicate stratification. Traditional systems would enter a loop jumping from one rule to the other. This is prevented in DB_DeLP by the circularity condition imposed on argumentation lines of dialectical trees.[3] This condition does not allow the re-introduction of $\mathcal{A}_1$ as a defeater of $\mathcal{A}_2$ in the dialectical tree of the previous example.

[3] This condition was inherited from the original DeLP system, the interested reader may consult [15].

6 A Worked Example

In this section we present an example to illustrate the practical uses of defeasible databases. The example is based on a classical benchmark in deductive databases concerning data on prescriptions, physicians and patients [21]. The system is a DSS to help employees decide whether a given medication should be sold to a patient. The relation *prescription* (pres) means that there is a prescription for a given drug to be administered to a given patient. *Allergic* shows known allergic reactions in patients, *physician* lists where physicians work, *patient* lists insurance company and clinics to which a patient usually goes, and *psychiatrist* (psy) establishes that a physician is also a psychiatrist (see Figure 6).

`patients(patient_id,clinic,insurance)`
(`patients(456,new_line,hope)`, 0.6)
(`patients(587,delta,hope)`, 0.6)
(`patients(234,new_line,trust)`, 0.6)
(`patients(1211,delta,trust)`, 0.6)
(`patients(254,star,trust)`, 0.6)
...

`physicians(phy_id,clinic)`
(`physicians(432,star)`, 0.7)
(`physicians(54,delta)`, 0.7)
(`physicians(672,new_line)`, 0.7)
(`physicians(432,delta)`, 0.7)
...

`pres(note_id,patient_id,phy_id,drug,text)`
(`pres(23445,587,432,pen,text1)`, 0.6)
(`pres(23446,587,54,amoxicillin,text2)`, 0.6)
(`pres(23447,587,54,vicodin,text3)`, 0.6)
(`pres(23448,1211,54,morphine,text4)`, 0.6)
(`pres(23449,234,672,diazepam,text5)`, 0.6)
...

`allergic(patient_id,drug)`
(`allergic(587,pen)`, 0.7)
(`allergic(1211,pen)`, 0.6)
(`allergic(1211,morphine)`, 0.6)
...

`psy(phy_id)`
(`psy(672)`, 0.8)
(`psy(54)`, 0.8)
...

The intensional database is formed by the rules in Figure 6. The first rule says that a medication should be sold if there is prescription for it. The second rule says that it should not be sold if the physician is suspended and the third says that it should not be sold if the patient is allergic. The fourth rule concerns special drugs that have to be authorized before being sold and for that should have been prescribed by a psychiatrist. The fifth rule establishes the drug is not authorized when it is prescribed by a psychiatrist that has been suspended.

The dialectical graph contains arguments A, B, C, D, and E:

– $\langle\langle \mathsf{A}, \mathsf{sell(Patient,Drug)}, 0.65\rangle\rangle$,
  $\mathsf{A} = \{(\mathtt{sell(Patient,Drug)} \mathrel{-\!\!\!\prec} \mathtt{pres(X,Patient,Y,Drug)}, 0.65)\}$.

(`sell(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug)`,0.65)
(~`sell(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug),susp(Y)`,0.75)
(~`sell(Patient,Drug)` $\prec$ `allergic(Patient,Drug)`,0.95)
(`auth_pres(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug),psychiatrist(Y)`,0.6)
(~`auth_pres(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug),psy(Y),susp(Y)`,0.7)

- ⟪B, ~sell(Patient,Drug), 0.75⟫,
  B = {(~`sell(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug,Text),susp(Y)`, 0.75)}.
- ⟪C, ~sell(Patient,Drug), 0.95⟫,
  C = {(~`sell(Patient,Drug)` $\prec$ `allergic(Patient,Drug)`, 0.95)}.
- ⟪D, authorize_pres(Patient,Drug), 0.6⟫,
  D = {(`auth_pres(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug,Text),psy(Y)`, 0.6)}.
- ⟪E, ~authorize_pres(Patient,Drug), 0.7⟫,
  E = {(~`auth_pres(Patient,Drug)` $\prec$ `pres(X,Patient,Y,Drug,Text),psy(Y),`
  `susp(Y)`, 0.7)}.

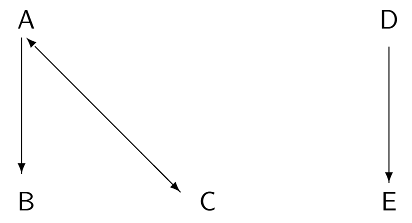

Fig. 6. Dialectical graph for clinical database.

Suppose the system is faced with a query for the fact sell(587,vicodin). It first finds a potential argument that can be instantiated to support this fact, so it selects A and instantiates it to:
$\mathcal{A}$ = {(`sell(787,vicodin)` $\prec$ `pres(23447,587,54,vicodin,text3)`, 0.6)}. Using the dialectical graph we can see that there are two links that connect A with its defeaters, so we can explore to see if an instance of B or C can be built to attack argument $\mathcal{A}$. Since this is not the case argument $\mathcal{A}$ is the only argument in the dialectical tree and the answer is yes.

Next, the system is faced with query sell(587,pen). The structure is similar to the previous case, but in this situation potential argument A is instantiated to

$$\mathcal{A} = \{(\texttt{sell(587,pen)} \prec \texttt{pres(23445,587,432,pen,text1)}, 0.6)\}$$

and following the links in the dialectical graph we find defeater B that can be instantiated to:
$\mathcal{B}$ = {(~`sell(587,pen)` $\prec$ `allergic(587,pen)`, 0.7)}. No more defeaters can be added to this dialectical tree so the answer to sell(587,pen) is no.

Now the query auth_pres(234,diazepam) is performed. In this case potential argument D is instantiated to:
$\mathcal{D} = \{$auth_pres(234,diazepam) $\prec$ pres(23449,234,672,diazepem,text5), psy(672)$, 0.6)\}$ and no defeater can be found for $\mathcal{D}$ thus the answer is yes.

Facts can be added to the database and also new tables can be created. Suppose we add a new table that contains a list of doctors that have been suspended due to legal issues. This table contains the fact (suspended(672)$, 0.8)$. If query authorize_pres(234,diazepam) is re-processed by the system the answer would now be no, given that a new argument:
$\mathcal{E} = \{($~auth_pres(234,diazepam) $\prec$ pres(23449,234,672,diazepam,text5), psychiatrist(672),suspended(672)$, 0.7)\}$. can be built by instantiating E, resulting in a defeater for $\mathcal{D}$. Thus, $\mathcal{D}$ is no longer warranted. Note how new tables and new facts can be added to the system without rebuilding the dialectical graph.

7 Conclusions and Future Work

In this work, we have defined a multi-agent system which virtually integrates different databases into a common view. We have also presented a layered architectural model that we have designed to develop practical applications for reasoning with data from multiple sources. This model provides a novel system architecture for decision-support systems (DSS) that combines database technologies with an argumentation based framework.

We have also defined an argumentation-based formalism that integrates uncertainty management and is specifically tailored for database integration. This formalism was also provided with an optimization mechanism based on pre-compiled knowledge. Using this mechanism, the argumentation system can comply with real time requirements needed to manage data and model reasoning over this data in dynamic environments.

Future work may be done in different directions. First, many important and interesting issues could be considered in the general framework of database theory or information integration theory, such as how integrity constraints affect this set-up, how our proposal relates to local/global views, or which connections could be established with database repairs. Second, we will integrate DB_DeLP with a massive data component to obtain experimental results regarding the system's behavior in such trying environment.

A related research line could also be obtained by extending the language of DB_DeLP to use it in practical systems, particularly to implement argumentation-based active databases and decision support systems backed by large repositories of data.

Acknowledgements. This research was partially supported by CONICET (Argentina), and by the *Universidad Nacional del Sur*. The authors would also like to thank the anonymous reviewers for their valuable comments and suggestions that helped improve the quality of this work.

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. Alsinet, T., Godo, L.: A complete calculus for possibilistic logic programming with fuzzy propositional variables. In: Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence (UAI-2000). pp. 1–10. ACM Press (2000)
3. Berti, L.: Quality and recommendation of multi-source data for assisting technological intelligence applications. In: Proc. of 10th International Conference on Database and Expert Systems Applications. pp. 282–291. AAAI, Italy (1999)
4. Brodie, M.L., Jarke, M.: On integrating logic programming and databases. In: Expert Database Workshop 1984. pp. 191–207 (1984)
5. Bryant, D., Krause, P.: An implementation of a lightweight argumentation engine for agent applications. Logics in Artificial Intelligence 4160(1), 469–472 (2006)
6. Capobianco, M., Chesñevar, C.I., Simari, G.R.: Argumentation and the dynamics of warranted beliefs in changing environments. Journal of Autonomous Agents and Multiagent Systems 11, 127–151 (2005)
7. Carbogim, D., Robertson, D., Lee, J.: Argument-based applications to knowledge engineering. The Knowledge Engineering Review 15(2), 119–149 (2000)
8. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. on Knowledge and Data Eng. 1(1) (1989)
9. Chesñevar, C.I., Maguitman, A.G., Loui, R.P.: Logical Models of Argument. ACM Computing Surveys 32(4), 337–383 (2000)
10. Chesñevar, C.I., Maguitman, A.G., Simari, G.R.: Argument-based critics and recommenders: A qualitative perspective on user support systems. Data & Knowledge Engineering 59(2), 293–319 (2006)
11. Chesñevar, C.I., Simari, G.R., Alsinet, T., Godo, L.: A logic programming framework for possibilistic argumentation with vague knowledge. In: Proc. of Uncertainty in Artificial Intelligence Conference (UAI 2004), Banff, Canada (2004)
12. Cuppens, F., Demolombe, R.: Cooperative answering: a method to provide intelligent access to databases. In: Proc. 2nd Conf. on Expert Database Systems. pp. 621–643 (1988)
13. Dubois, D., Lang, J., Prade, H.: Possibilistic logic. In: D.Gabbay, C.Hogger, J.Robinson (eds.) Handbook of Logic in Art. Int. and Logic Prog. (Nonmonotonic Reasoning and Uncertain Reasoning), pp. 439–513. Oxford Univ. Press (1994)
14. Dung, P.M.: On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning and Logic Programming and n-Person Games. Artificial Intelligence 77(2), 321–357 (1995)
15. García, A.J., Simari, G.R.: Defeasible Logic Programming: An Argumentative Approach. Theory and Practice of Logic Programming 4(1), 95–138 (2004)
16. Lakshmanan, L.V.S., Sadri, F.: Probabilistic deductive databases. In: Proc. of the Int. Logic Programming Symposium. pp. 254–268 (1994)
17. Lakshmanan, L.V., Shiri, N.: A parametric approach to deductive databases with uncertainty. Journal of Intelligent Information Systems 13(4), 554–570 (2001)
18. McLeod, D., Heimbigner, D.: A federated architecture for information management. ACM Transactions on Information Systems 3(3), 253–278 (1985)
19. Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In: Handbook of Philosophical Logic, vol. 4, pp. 219–318 (2002)

20. Prakken, H., Vreeswijk, G.: Logical systems for defeasible argumentation. In: Handbook of Philosophical Logic, vol. 4, pp. 219–318 (2002)
21. Quian, X.: Query folding. In: Proc. 12th Intl. Conf on Data Engineering. pp. 48–55 (1996)
22. Simari, G.R., Loui, R.P.: A Mathematical Treatment of Defeasible Reasoning and its Implementation. Artificial Intelligence 53(1–2), 125–157 (1992)
23. Subrahmanian, V.S.: Paraconsistent disjunctive deductive databases. Theorethical Computer Science 93(1), 115–141 (1992)
24. Zaniolo, C.: Prolog: A database query language for all seasons. In: Expert Database Workshop 1984. pp. 219–232 (1984)
25. Zaniolo, C.: Intelligent databases: Old challenges and new opportunities 3/4(1), 271–292 (1992)