A First Approach to Combining Ontologies and Defeasible Argumentation for the Semantic Web[1]

Sergio Alejandro Gómez[†2] **Carlos Iván Chesñevar**[†‡] **Guillermo Ricardo Simari**[†2]

[†]Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)[3]
Depto. Cs. e Ing. de la Computación – Universidad Nacional del Sur
Alem 1253 (8000) Bahía Blanca - Argentina – Tel/Fax: (+54) 291-4595135/6
E-mail: {sag, cic, grs}@cs.uns.edu.ar

[‡]Grupo de Investigación en Inteligencia Artificial
Departament d'Informàtica — Universitat de Lleida
Campus Cappont – C/Jaume II, 69 – E-25001 Lleida, SPAIN
Tel/Fax: (+34) (973) 70 2764 / 2702 – E-mail: cic@eps.udl.es

Abstract

The Semantic Web is a project intended to create a universal medium for information exchange by giving semantics to the content of documents on the Web through the use of ontology definitions. Problems for modelling common-sense reasoning (such as reasoning with uncertainty or with incomplete and potentially inconsistent information) are also present when defining ontologies. In recent years, defeasible argumentation has succeeded as an approach to formalize such common-sense reasoning. Agents operating in multi-agent systems in the context of the Semantic Web need to interact with each other in order to achieve the goals stated by their users. In this paper we propose a XML-based language named *XDeLP* for ontology interchange among agents in the web.

Keywords: Defeasible argumentation, Semantic Web, ontology integration, eXtensible Markup Language, multiagent systems.

1 Introduction

Although the World Wide Web is a vast repository of information, its utility is restricted by limited facilities for searching and integrating different kinds of data, as search for queries is mostly *syntax-based* (*e.g.*, using keywords). The Semantic Web [1] has emerged as a project intended to create a universal medium for information exchange by giving *semantics* to the content of documents on the Web. A common way to provide semantics to documents on the web is through the use of *ontology definitions*. Common problems from common-sense reasoning (*e.g.*, reasoning with uncertainty or with incomplete and potentially inconsistent information) are present when defining ontologies. To cope with such problems, during the last decade defeasible argumentation has developed as a successful approach to formalize such common-sense reasoning [6, 16].

Agents are programs with power delegated by its users to act on their behalf [3]. Agents operating in multi-agent systems in the context of the Semantic Web usually need to interact with each other in order to achieve some goals. In the last years, several solutions and alternative representation languages have been explored in the literature to solve this problem (*e.g.*, [11, 13, 14]). In particular, the XML language [12] (acronym for eXtensible Markup Language) has resulted successful, providing an standardized framework with which a programmer can define his *own* markup language. Any

[1]*VI Workshop de Agentes y Sistemas Inteligentes. XI Congreso Argentino de Ciencias de la Computación (CACIC '2005)*. Concordia, Argentina.

[2]Partially funded by PICT 2002 Nro. 13096 (Agencia Nacional de Promoción Científica y Tecnológica).

[3]LIDIA is a member of IICyTI Instituto de Investigación en Ciencia y Tecnología Informática.

language based on XML consists of a set of *element types* which serves to define *types* of documents and are referred to as *Document Type Definitions* (DTDs).

In a previous paper [9], we have proposed an ontology algebra based on Defeasible Logic Programming (DeLP) [7]. In this paper we extend those preliminary results by defining *XDeLP*, an XML-based language for ontology interchange among agents within the web. By means of the namespace mechanism provided by XML, XDeLP can be suitable integrated with existing approaches in the realm of Semantic Web applications.

The rest of the paper is structured as follows. Section 2 introduces some fundamentals of DeLP by means of an example in the banking domain [10]. Next, Section 3 summarizes the concepts needed to understand both XML notation and XML DTD definitions. Section 4 introduces the syntax of the proposed XDeLP language. Finally, Section 5 concludes the paper.

2 Defeasible Logic Programming: Fundamentals

Defeasible argumentation has evolved in the last decade as a successful approach to formalize defeasible reasoning [6]. The growing success of argumentation-based approaches has caused a rich crossbreeding with other disciplines, providing interesting results in different areas such as knowledge engineering, multiagent systems, and decision support systems, among others [15, 6]. *Defeasible logic programming* (DeLP) [7] is a particular formalization of defeasible argumentation based on logic programming, which has proven to be particularly attractive in the context of real-world applications, such as clustering [8], intelligent web search [5], knowledge management [2], natural language processing [4], and web form-based applications [10]. To make this paper self-contained, we will summarize next the fundamentals of DeLP.[4]

2.1 Knowledge Representation in DeLP

Next we will introduce the basic definitions to represent knowledge in DeLP.

Definition 1 (DeLP program $\mathcal{P}$) *A defeasible logic program (delp) is a set $\mathcal{P} = (\Pi, \Delta)$ of Horn-like clauses, where Π and Δ stand for sets of* strict *and* defeasible *knowledge, resp. The set Π of strict knowledge involves* strict rules *of the form $P \leftarrow Q_1, \ldots, Q_k$ and* facts *(strict rules with empty body), and it is assumed to be* non-contradictory.[5] *The set Δ of defeasible knowledge involves* defeasible rules *of the form $P \mathrel{-\!\!\prec} Q_1, \ldots, Q_k$, which stands for "$Q_1, \ldots Q_k$* provide a tentative reason to believe *P." Strict and defeasible rules in DeLP are defined in terms of* literals *P, Q_1, Q_2, ... A literal is an atom or the strict negation ($\sim$) of an atom.*

The underlying logical language is that of extended logic programming, enriched with a special symbol " $\mathrel{-\!\!\prec}$ " to denote defeasible rules. Both default and classical negation are allowed (denoted not and $\sim$, resp.). Syntactically, the symbol " $\mathrel{-\!\!\prec}$ " is all what distinguishes a *defeasible* rule $P \mathrel{-\!\!\prec} Q_1, \ldots, Q_k$ from a *strict* (non-defeasible) rule $P \leftarrow Q_1, \ldots, Q_k$. DeLP rules are thus Horn-like clauses to be thought of as *inference rules* rather than implications in the object language. Analogously as in traditional logic programming, the *definition* of a predicate P in $\mathcal{P}$, denoted $P^{\mathcal{P}}$, is given by the set of all those (strict and defeasible) rules with head P and arity n in $\mathcal{P}$. If P is a predicate in $\mathcal{P}$, then $name(P)$ and $arity(P)$ will denote the predicate name and arity, resp. We will write $\mathsf{Pred}(\mathcal{P})$ to denote the set of all predicate names defined in a program $\mathcal{P}$.

Next we will present an example in the banking domain which will be used to illustrate our proposal.

[4] For an in-depth treatment, the interested reader is referred to [7].

[5] Contradiction stands for deriving two complementary literals wrt strict negation (P and $\sim P$) or default negation (P and $\mathsf{not}\ P$).

Facts (user-provided information):
(1) $info(john, krakosia, phdstudent, 400)$.
(2) $info(ajax, greece, phdstudent, 350)$.
(3) $info(danae, greece, none, 10000)$.
(4) $req_loan(john, 2000)$.
(5) $req_loan(ajax, 4500)$.
(6) $req_loan(danae, 1000)$.

Facts (bank information):
(7) $family_record(john, rich)$.
(8) $family_record(ajax, unknown)$.
(9) $family_record(danae, unknown)$.
(10) $credible(krakosia)$.
(11) $credible(greece)$.
(12) $bankmanager(peter)$.

Defeasible rules:
(13) $candidate(P) \prec profile_ok(P)$.
(14) $candidate(P) \prec$
$info(P, _, _, Income), req_loan(P, Amount),$
$Amount < Income * 10, trustctry(P, Ctry)$.
(15) $profile_ok(P) \prec goodincome(P), trustctry(P, Ctry)$.
(16) $trustctry(P, Ctry) \prec info(P, Ctry, _, _), credible(Ctry)$.
(17) $goodincome(P) \prec info(P, _, _, Income), Income > 300$.
(18) $\sim goodincome(P) \prec \ \sim solvent(P)$.
(19) $\sim solvent(P) \prec info(P, _, phdstudent, _)$.
(20) $solvent(P) \prec info(_, _, phdstudent, _), richfamily(P)$.
(21) $richfamily(P) \prec family_record(P, rich)$.

Strict rules:
(22) $candidate(P) \leftarrow bankmanager(P)$.

Figure 1: Defeasible logic program $\mathcal{P}_{bank}$with bank criteria for granting a loan application

Example 1 *An international bank keeps track of its clients in order to determine whether to concede loans. For every client the bank keeps name, country of origin, profession, average income per month, and family status of the client. The account manager of the bank has a number of criteria for conceding loans. Loans are given if the person has a reasonable "profile," according to his personal records. Figure 1 shows a DeLP program $\mathcal{P}_{bank}$ for assessing the status of such a loan application.*

Facts (1–3) of the form $info(Name, Country, Profession, IncomePerMonth)$ describe information about the customers—fact (1) says that John is a PhD student from a country named Krakosia and has an average income of $400 *a month; fact (2) says that Ajax is also a PhD student but from Greece and has an average income of* $350 *a month, and fact (3) says that Danae is from Greece with an income of* $10,000 *a month and with no information regarding her profession. Facts (4–6) describe how much money has been requested by each customer to the bank, whereas facts (7–9) summarize the family records of the customers. Facts (10–11) establish that Krakosia and Greece are considered as trustworthhy countries by the bank authorities. Fact (12) says that Peter is one of the bank managers.*

Defeasible rules (13–14) express that a person P is candidate for a loan usually if the person P has the right profile or if the requested loan is reasonable for the income in 10 months and P comes from a trustworthy country. Rule (15) says that a right profile is defined in terms of monthly income and country. Rule (16) establishes that usually all countries are trustworthy. Rule (17) says that a person P has a reasonable income if it is typically $300 *a month or higher. Rule (18) expresses that usually a person P who is not economically solvent does not have a reasonable income. Rules (19–20) say that usually PhD students are not solvent people unless they come from rich families. Besides, rule (21) says that people assessed by the bank with a family status "rich" are expected to be from rich families. Finally, rule (22) says that bank managers are without no doubt candidates for loans. Note that in this particular example we have* Pred*($\mathcal{P}_{bank}$)* $=$ $\{info/4$, $family_record/2$, $req_loan/2$, $credible/1$, $candidate/1$, $profile_ok/1$, $trustctry/2$, $goodincome/1$, $solvent/1$, $richfamily/1$, *bankmanager/1* $\}$.

2.2 Argument, Counterargument, and Defeat in DeLP

Deriving literals in DeLP results in the construction of *arguments*. An argument $\mathcal{A}$ is a (possibly empty) set of ground defeasible rules that together with the set Π provide a logical proof for a given literal Q, satisfying the additional requirements of *non-contradiction* and *minimality*. Formally:

Definition 2 (Argument) *Given a DeLP program $\mathcal{P}$, an* argument $\mathcal{A}$ *for a query Q, denoted $\langle \mathcal{A}, Q\rangle$, is a subset of ground instances of defeasible rules in $\mathcal{P}$, such that:*

1. *there exists a* defeasible derivation *for Q from $\Pi \cup \mathcal{A}$;*
2. *$\Pi \cup \mathcal{A}$ is non-contradictory (*i.e., *$\Pi \cup \mathcal{A}$ does not entail two complementary literals P and $\sim P$ (or P and* $\mathsf{not}\, P$*)), and,*
3. *$\mathcal{A}$ is minimal with respect to set inclusion (*i.e., *there is no $A' \subseteq A$ such that there exists a defeasible derivation for Q from $\Pi \cup \mathcal{A}'$).*

An argument $\langle \mathcal{A}_1, Q_1\rangle$ is a sub-argument *of another argument $\langle \mathcal{A}_2, Q_2\rangle$ if $\mathcal{A}_1 \subseteq \mathcal{A}_2$. Given a DeLP program $\mathcal{P}$, $Args(\mathcal{P})$ denotes the set of all possible arguments that can be derived from $\mathcal{P}$.*

The notion of defeasible derivation corresponds to the usual query-driven SLD derivation used in logic programming, performed by backward chaining on both strict and defeasible rules; in this context a negated literal $\sim P$ is treated just as a new predicate name no_P. Minimality imposes a kind of 'Occam's razor principle' [17] on argument construction. The non-contradiction requirement forbids the use of (ground instances of) defeasible rules in an argument $\mathcal{A}$ whenever $\Pi \cup \mathcal{A}$ entails two complementary literals. It should be noted that non-contradiction captures the two usual approaches to negation in logic programming (*viz.*, default negation and classical negation), both of which are present in DeLP and related to the notion of counterargument, as shown next.

Definition 3 (Counterargument. Defeat) *An argument $\langle \mathcal{A}_1, Q_1\rangle$ is a* counterargument *for an argument $\langle \mathcal{A}_2, Q_2\rangle$ iff*

- ***Subargument attack**: there is an subargument $\langle \mathcal{A}, Q\rangle$ of $\langle \mathcal{A}_2, Q_2\rangle$ (called* disagreement subargument*) such that the set $\Pi \cup \{Q_1, Q\}$ is contradictory, or*
- ***Default negation attack**: a literal* $\mathsf{not}\, Q_1$ *is present in the body of some rule in $\mathcal{A}_2$.*

We will assume a preference criterion *on conflicting arguments defined as a partial order $\preceq \subseteq Args(\mathcal{P}) \times Args(\mathcal{P})$. We distinguish between* proper *and* blocking defeaters *as a refinement of the notion of counterargument as follows:*

An argument $\langle \mathcal{A}_1, Q_1\rangle$ is a proper defeater *for an argument $\langle \mathcal{A}_2, Q_2\rangle$ if $\langle \mathcal{A}_1, Q_1\rangle$ counterargues $\langle \mathcal{A}_2, Q_2\rangle$ with a disagreement subargument $\langle \mathcal{A}, Q\rangle$ (subargument attack) and $\langle \mathcal{A}_1, Q_1\rangle$ is strictly preferred over $\langle \mathcal{A}, Q\rangle$ wrt $\preceq$.*

An argument $\langle \mathcal{A}_1, Q_1\rangle$ is a blocking defeater *for an argument $\langle \mathcal{A}_2, Q_2\rangle$ if $\langle \mathcal{A}_1, Q_1\rangle$ counterargues $\langle \mathcal{A}_2, Q_2\rangle$ and one of the following situations holds: (a) There is a disagreement subargument $\langle \mathcal{A}, Q\rangle$ for $\langle \mathcal{A}_2, Q_2\rangle$, and $\langle \mathcal{A}_1, Q_1\rangle$ and $\langle \mathcal{A}, Q\rangle$ are unrelated to each other wrt $\preceq$; or (b) $\langle \mathcal{A}_1, Q_1\rangle$ is a default negation attack on some literal* $\mathsf{not}\, Q_1$ *in $\langle \mathcal{A}_2, Q_2\rangle$.*

Generalized specificity [17] is typically used as a syntax-based criterion among conflicting arguments, preferring those arguments which are *more informed* or *more direct* [17, 18].[6] However, it must be remarked that other alternative partial orders could also be valid, such us defining argument comparison using rule priorities [7].

[6]When using generalized specificity as the comparison criterion between arguments, the argument $\langle \{a \prec b, c\}, a\rangle$ is preferred over the argument $\langle \{\sim a \prec b\}, \sim a\rangle$ as it is considered *more informed* (*i.e.*, it relies on more premises). However, the argument $\langle \{\sim a \prec b\}, \sim a\rangle$ is preferred over $\langle \{(a \prec b); (b \prec c\}), a\rangle$ as it is regarded as *more direct* (*i.e.*, it is a shorter derivation).

Example 2 *Consider the DeLP program shown in Example 1. There exists an argument $\mathcal{A}$ supporting the defeasible conclusion that John is a candidate for a loan,* i.e.*, $\langle \mathcal{A}_1, candidate(john) \rangle$, where:*[7]

$$\begin{aligned}\mathcal{A}_1 = \{ & (candidate(john) \prec profile_ok(john)); \\ & (profile_ok(john) \prec goodincome(john), trustctry(john, krakosia)); \\ & (trustctry(john, krakosia) \prec info(john, krakosia, _, _), credible(krakosia)); \\ & (goodincome(john) \prec info(john, _, _, 400), 400 > 300)\};\end{aligned}$$

Another argument $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$ can be derived from $\mathcal{P}_{bank}$, supporting the conclusion that $John$ does not have a reasonable income, with:

$$\mathcal{A}_2 = \{(\sim goodincome(john) \prec \sim solvent(john)); (\sim solvent(john) \prec info(john, _, phdstudent, _)\}.$$

Using generalized specificity [17] as the preference criterion among conflicting arguments, it turns out that the argument $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$ is a blocking *defeater for the argument $\langle \mathcal{A}_1, candidate(john) \rangle$.*

2.3 Computing Warrant through Dialectical Analysis

An *argumentation line* starting in an argument $\langle \mathcal{A}_0, Q_0 \rangle$ (denoted $\lambda^{\langle \mathcal{A}_0, Q_0 \rangle}$) is a sequence $[\langle \mathcal{A}_0, Q_0 \rangle$, $\langle \mathcal{A}_1, Q_1 \rangle$, $\langle \mathcal{A}_2, Q_2 \rangle$, ..., $\langle \mathcal{A}_n, Q_n \rangle \ldots]$ that can be thought of as an exchange of arguments between two parties, a *proponent* (evenly-indexed arguments) and an *opponent* (oddly-indexed arguments). Each $\langle \mathcal{A}_i, Q_i \rangle$ is a defeater for the previous argument $\langle \mathcal{A}_{i-1}, Q_{i-1} \rangle$ in the sequence, $i > 0$. In order to avoid *fallacious* reasoning, dialectics imposes additional constraints on such an argument exchange to be considered rationally *acceptable*. Given a DeLP program $\mathcal{P}$ and an initial argument $\langle \mathcal{A}_0, Q_0 \rangle$, the set of all acceptable argumentation lines starting in $\langle \mathcal{A}_0, Q_0 \rangle$ accounts for a whole dialectical analysis for $\langle \mathcal{A}_0, Q_0 \rangle$ (*i.e.*, all possible dialogues about $\langle \mathcal{A}_0, Q_0 \rangle$ between proponent and opponent), formalized as a *dialectical tree*.

Nodes in a dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ can be marked as *undefeated* and *defeated* nodes (U-nodes and D-nodes, resp.). A dialectical tree will be marked as an AND-OR tree: all leaves in $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ will be marked U-nodes (as they have no defeaters), and every inner node is to be marked as *D-node* iff it has at least one U-node as a child, and as *U-node* otherwise. An argument $\langle \mathcal{A}_0, Q_0 \rangle$ is ultimately accepted as valid (or *warranted*) wrt a DeLP program $\mathcal{P}$ iff the root of its associated dialectical tree $\mathcal{T}_{\langle \mathcal{A}_0, Q_0 \rangle}$ is labelled as *U-node*.

Given a DeLP program $\mathcal{P}$, solving a query Q wrt $\mathcal{P}$ accounts for determining whether Q is supported by (at least) one warranted argument. Different doxastic attitudes can be distinguished as follows:

1. Yes: accounts for believing Q iff there is at least one warranted argument supporting Q on the basis of $\mathcal{P}$.
2. No: accounts for believing $\sim Q$ iff there is at least one warranted argument supporting $\sim Q$ on the basis of $\mathcal{P}$.
3. $Undecided$: neither Q nor $\sim Q$ are warranted wrt $\mathcal{P}$.
4. $Unknown$: Q does not belong to the signature of $\mathcal{P}$.

Thus, according to DeLP semantics, given a program $\mathcal{P}$, solving a query Q —for any $Q \in \mathsf{Pred}(\mathcal{P})$— will result in a value belonging to the set Ans= $\{Yes, No, Undecided, Unknown\}$.

Example 3 *Consider the query $candidate(john)$ solved wrt the program $\mathcal{P}_{bank}$(Fig. 1). As shown in Example 1, this query would start a search for arguments supporting $candidate(john)$, and argument $\langle \mathcal{A}_1, candidate(john) \rangle$ will be found. In order to determine whether this argument is warranted, its*

[7] For the sake of clarity, we use parentheses to enclose defeasible rules in arguments, separated by semicolons, *i.e.* $\mathcal{A} = \{(rule_1)\,;(rule_2)\,;\ldots;(rule_k)\}$.

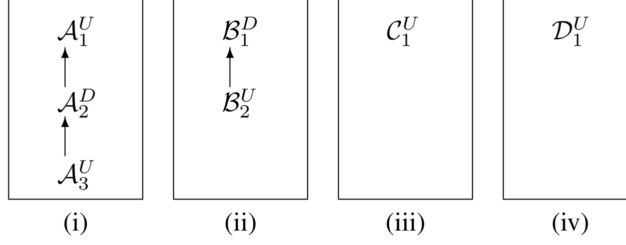

Figure 2: Dialectical trees for queries: (i) $candidate(john)$ wrt $\mathcal{P}_{bank}$; (ii) $candidate(ajax)$ wrt $\mathcal{P}_{bank}$, (iii) $candidate(danae)$ wrt $\mathcal{P}_{bank}$, and (iv) $candidate(peter)$ wrt $\mathcal{P}_{bank}$

dialectical tree will be computed: as shown in Example 1, there is only one (blocking) defeater for $\langle \mathcal{A}_1, candidate(john)\rangle$, namely. $\langle \mathcal{A}_2, \sim goodincome(john)\rangle$. This defeater, on its turn, has another (proper) defeater $\langle \mathcal{A}_3, solvent(john)\rangle$, with $\mathcal{A}_3$ = { $(solvent(john) \prec info(john, _, phdstudent, _)$, $richfamily(john))$; $(richfamily(john) \prec family_record(john, rich))$ } The resulting (marked) dialectical tree is depicted in Fig. 2(i). As the root note of the resulting dialectical tree is a U-node, the answer to $candidate(john)$ is Yes.

Consider now the query $candidate(ajax)$. As in John's case, we can find the argument $\langle \mathcal{B}_1$, $candidate(ajax)\rangle$, which is defeated by $\langle \mathcal{B}_2, \sim goodincome(ajax)\rangle$, with

$$\begin{aligned}
\mathcal{B}_1 \;=\; & \{(candidate(ajax) \prec profile_ok(ajax));\\
& (profile_ok(ajax) \prec goodincome(ajax), trustctry(ajax, greece));\\
& (trustctry(ajax, greece) \prec info(ajax, greece, _, _), credible(greece));\\
& (goodincome(ajax) \prec info(ajax, _, _, 350), 350 > 300)\};\\
\mathcal{B}_2 \;=\; & \{(\sim goodincome(ajax) \prec \; \sim solvent(ajax));\\
& (\sim solvent(ajax) \prec info(ajax, _, phdstudent, _)\};
\end{aligned}$$

Hence the associated dialectical tree for $candidate(ajax)$ has two nodes, with the root labelled as D-node (Fig. 2(ii)). The original argument for $candidate(ajax)$ is therefore not warranted. Finally consider the query $candidate(danae)$. There is an argument without defeaters (and hence warranted) for this query, as Danae has the right profile for the bank:[8]

$$\begin{aligned}
\mathcal{C}_1 \;=\; & \{(candidate(danae) \prec profile_ok(danae));\\
& (profile_ok(danae) \prec goodincome(danae), trustctry(danae, greece));\\
& (trustctry(greece) \prec info(danae, greece, _, _), credible(greece));\\
& (goodincome(danae) \prec info(danae, _, _, 10000), 10000 > 300)\}
\end{aligned}$$

None of these arguments has defeaters. Following the same reasoning as above, both of them are warranted. The resulting dialectical tree will have a unique node, as depicted in Fig. 2(iii).

Finally, there exists an empty argument $\mathcal{D}_1 = \emptyset$ —$\langle \emptyset, candidate(peter)\rangle$— supporting that Peter is a candidate for a loan as he is one of the bank managers. As the support of this argument is made up of strict derivations, it has no defeaters [7], so it is also warranted. The resulting dialectical tree is depicted in Fig. 2(iv).

[8]Note that there is also a second argument without defeaters supporting the query $candidate(danae)$, namely $\langle \mathcal{C}_2, candidate(danae)\rangle$, with $\mathcal{C}_2 = \{$ $(candidate(danae) \prec info(danae, _, _, 10000), req_loan(danae,$ 1000),1000<10000*10, $trustctry(danae, greece))$; $(trustctry(danae, greece) \prec info(danae,$ $greece, _, _)$, $credible(greece))\}$.

2.4 Argument-based Ontologies for the Semantic Web

An *ontology* is a specification of a conceptualization. In computer science, ontologies establish a joint terminology between members of a community of interest. These members can be human or automated agents. In the context of the semantic web, an OWL ontology [14] is just a collection of information, generally information about classes and properties.

In a recent work [9], we presented an approach to define ontologies based on DeLP—the so-called *d-ontologies*. In that approach, an ontology will be associated with a DeLP program representing knowledge, in which facts and strict rules are distinguished. More formally:

Definition 4 (*d-Ontology*) *A d-Ontology is a DeLP program* $\mathcal{P} = (K_P \cup K_G, \Delta)$ *where* K_P *stands for particular knowledge (facts about individuals),* K_G *stands for general knowledge (strict rules about relations holding among individuals), and* Δ *stands for defeasible knowledge (defeasible rules).*

3 Fundamentals of XML

As stated in the introduction, XML [12] (acronym for eXtensible Markup Language) is a computer language for describing information. Any language based on XML consists of a set of *element types* which serves to define *types* of documents and are referred to as *Document Type Definitions* (DTDs). We briefly summarize here the concepts needed to understand DTD notation.

3.1 XML Syntax

The eXtended Markup Language is accepted as *the* emerging standard for data interchange on the Web. XML allows authors to create their own markup (*e.g.*, <AUTHOR>). *Well-formed* XML documents are documents that meet the constraints in the XML specification. *Valid* XML documents are documents that are well formed and additionally meet all of the constraints specified in the DTD.

In XML we have start tags (*e.g.*, <foo>), end tags (*e.g.*, </foo>), and empty tags (*e.g.*, <foo bar="baz"/>. Empty tags can have attributes (*e.g.*, bar) that take a value (*e.g.*, baz). Elements that contain some mixture of markup/character data must have matching start- and end-tags (*e.g.*, <country gov="democracy">Krakosia</country>).

3.2 Element Type Declarations

Element type declarations allow an XML application to constrain the elements that can occur in the document and to specify the order in which can occur. The expression <!ELEMENT foo EMPTY> declares foo elements to be empty elements. The expression <!ELEMENT foo (apple|orange|banana)> declares that the element foo can contain exactly one apple, orange, or one banana element.

XML allows element types to be declared that can contain other elements. The expression <!ELEMENT person (name, address, phone?)> declares that a person has a name, an address, and optionally a phone number. Zero or more occurrencies can be specified as in <!ELEMENT books (book)*>, one or more occurrencies as in <!ELEMENT books (book)+>. Character data is denoted by the keyword #PCDATA as in <!ELEMENT quotation #PCDATA>.

Attribute list declarations serve to specify the name, type, and optionally the default value of the attributes associated with an element. The expression <!ATTLIST foo bar CDATA #REQUIRED> means that the element foo has the attribute bar containing character data. The modifier #REQUIRED means that giving a value to the attribute is mandatory. Other modifiers such as #IMPLIED are possible meaning that the value for the attribute will be computed by an external application.

4 XDeLP: An XML-based Language for d-Ontology Interchange

We will present next a language named XDeLP for ontology interchange among intelligent agents on the Semantic Web. Our presentation will be based on the examples presented so far.

4.1 Defining Programs

First, XML commands should be provided to define DeLP programs. In our case study the program is named P_{bank}:

```
<delp id="PBank" version="1.0">
```

The delp tag is used to state that there is a program known as PBank. Besides, a version ID must be also added. Note that it may be desirable to annotate programs with comments, *e.g.*:

```
<comment>Program for defining criteria for granting
a loan application.</comment>
```

A program is composed by definitions of rule-schemas and declarations of rule and fact instances. The tag `</delp>` is a closing delimiter for the current definition of the program PBank. The corresponding DTD representation follows:

```
<!ELEMENT delp (comment?,definitions, declarations)>
<!ELEMENT comment PCDATA>
<!ATTLIST delp id CDATA #REQUIRED version CDATA #REQUIRED>
```

The definition of a program involves the specification of the allowed atoms (def-language part) and the rule schemas allowed. For example, to define an atom $info(Name, Country, Profession, Income)$ the DTD representation would be as follows:

```
<def-atom name="info" arity="4">
       <def-arg pos="1" param="Name" type="string" />
       <def-arg pos="2" param="Country" type="string" />
       <def-arg pos="3" param="Profession" type="string" />
       <def-arg pos="4" param="Income" type="float" />
</def-atom>
```

The atom name is info and has arity 4. Its four arguments are specified and a type for each argument is also given. The definition of atoms of arity 0 is also possible accounting for propositional DeLP programs. The corresponding part of the DTD follows:

```
<!ELEMENT definitions (def-language, def-rules)>
<!ELEMENT def-language (def-atom)*>
<!ELEMENT def-atom (def-arg)*>
<!ATTLIST def-atom name CDATA #REQUIRED arity CDATA #REQUIRED>
<!ELEMENT def-arg EMPTY>
<!ATTLIST def-arg pos CDATA #REQUIRED param CDATA #REQUIRED
       type CDATA #REQUIRED>
```

For the sake of example, the next lines describe an strict rule for expressing that a bank manager is *always* a candidate for granting a loan ($candidate(P) \leftarrow bankmanager(P)$):

```
<def-rule id="22" type="strict">
       <def-head name="candidate" negated="no">
              <arg pos="1" param="P" type="string" />
       </def-head>
       <def-body>
              <def-body-atom name="bankmanager" negated="no">
                     <arg pos="1" value="P" />
              </def-body-atom>
       </def-body>
</def-rule>
```

A rule has an ID and a type that can be either one of defeasible or strict, according to DeLP syntax rules. In particular, arguments (parameters) in literals can be anonymous by using the *dash* qualifier.[9] The DTD definitions follow:

```
<!ELEMENT def-rules (def-rule)*>
<!ATTLIST def-rule id CDATA #REQUIRED type (defeasible|strict) #REQUIRED>
<!ELEMENT def-rule (comment?,def-head, def-body)>
<!ELEMENT def-head (arg)*>
<!ATTLIST def-head name CDATA #REQUIRED negated (no|classical) "no">
<!ELEMENT arg EMPTY>
<!ATTLIST arg pos CDATA #REQUIRED value (CDATA|dash) #REQUIRED>
<!ELEMENT def-body (def-body-atom)+>
<!ELEMENT def-body-atom (arg)*>
<!ATTLIST def-body-atom name CDATA #REQUIRED
       negated (no|classical|default) "no">
```

As an example, we present next the encoding for rule (16) in Fig. 1.

```
<def-rule id="16" type="defeasible">
       <def-head name="trustctry" negated="no">
              <arg pos="1" param="P" type="string" />
              <arg pos="2" param="Ctry" type="string" />
       </def-head>
       <def-body>
              <def-body-atom name="info" negated="no">
                     <arg pos="1" value="P" />
                     <arg pos="2" value="Ctry" />
                     <arg pos="3" value="dash" />
                     <arg pos="4" value="dash" />
              </def-body-atom>
              <def-body-atom name="credible" negated="no">
                     <arg pos="1" value="Ctry" />
              </def-body-atom>
       </def-body>
</def-rule>
```

4.2 Defining Rules and Facts

Once a language $L_{\mathcal{P}}$ for a given XDeLP program $\mathcal{P}$ has been defined, the construction of arguments on the basis of this signature requires the instantiation of rules and facts. A program can have (ground) rules and facts instantiated as well. For example, the fact that John is PhDStudent from Krakosia and has an income of \$400 ($info(john, krakosia, phdstudent, 400)$) will be encoded as:

```
<fact-instance negated="no" type="fact" name="info">
       <arg pos="1" value="john"/>
       <arg pos="2" value="krakosia"/>
       <arg pos="3" value="phdstudent"/>
       <arg pos="4" value="400"/>
</fact-instance>
```

The attribute *negated* can take either one of two values—yes or no; taking value *no* by default. The attribute *type* can take either one of two values—*fact* and *assumption* accounting for the representation of defeasible facts (see [7]). Analogously, in order to represent the rule instance:

$$trustctry(john, krakosia) \prec info(john, krakosia, _, _), credible(krakosia),$$

we write:

```
<rule-instance id="16">
       <subst param="P" value="john" />
       <subst param="Ctry" value="krakosia" />
</rule-instance>
```

[9]Note that this corresponds to the “underscore” anonymous variable present in Prolog.

The DTD definitions follow:

```
<!ELEMENT declarations (rule-instances, fact-instances)>
<!ELEMENT rule-instances (rule-instance)*>
<!ATTLIST rule-instance id CDATA #REQUIRED>
<!ELEMENT rule-instance (subst)*>
<!ELEMENT subst EMPTY>
<!ATTLIST subst param CDATA #REQUIRED value CDATA #REQUIRED>
<!ELEMENT fact-instances (fact-instance)*>
<!ELEMENT fact-instance (arg)*>
<!ATTLIST fact-instance negated (yes|no) "no"
       type (fact|assumption) "fact" name CDATA #REQUIRED>
```

4.3 Defining Arguments and Argument Trees

The next several annotations describe how to represent arguments, argumentation lines, and dialectical trees in XDeLP. An argument $\langle \mathcal{A}, H \rangle$ will be represented by a fact instance H and a set $\mathcal{A}$ of rule instances:

```
<argument>
       <fact-instance>...Code for H...</fact-instance>
       <rule-instances>...Code for A...</rule-instances>
</argument>
```

A tag named *complete-argument* is also provided for representing all of the information concerning the derivation of arguments. A tag named *argument-derivation* is also provided in order to represent the tree supporting the derivation of an argument. Finally, tags for representing argument lines and argument trees are provided. Each node of an argument tree has an associated epistemic status that can take either one of two values—defeated or undefeated. Next we present the corresponding DTD definitions:

```
<!ELEMENT argument (fact-instance, rule-instances)>
<!ELEMENT complete-argument
       (fact-instance, rule-instances, fact-instances)>
<!ELEMENT argument-derivation (fact-instance, argument-derivation*)>
<!ELEMENT argument-line (argument)*>
<!ELEMENT argument-tree (argument, argument-tree*)>
<!ATTLIST argument-tree epistemic-status (defeated|undefeated) #IMPLIED>
```

A complete listing of the DTD for the XDeLP ontology interchange language presented in this section can be found in Fig. 3.

5 Conclusions

We have presented a novel argument-based language for ontology interchange among agents operating in the semantic web. As stated in the introduction, our proposal involves the definition of a XML-based language that allows to represent argumentation-based ontologies in a computer language. Besides, the proposed language can be suitable integrated with existing approaches in the realm of the Semantic Web applications.

Additional considerations regarding the translation between existing web ontology languages (such as DAML-ONT and OWL) and XDeLP need to be addressed in order to establish the respective expressive power of the language under study. Research in this direction is currently being pursued.

```
<!ELEMENT delp (comment?,definitions, declarations)>
<!ATTLIST delp id CDATA #REQUIRED version CDATA #REQUIRED>
<!ELEMENT comment PCDATA>

<!ELEMENT definitions (def-language, def-rules)>

<!ELEMENT def-language (def-atom)*>

<!ELEMENT def-atom (def-arg)*>
<!ATTLIST def-atom name CDATA #REQUIRED arity CDATA #REQUIRED>
<!ELEMENT def-arg EMPTY>
<!ATTLIST def-arg pos CDATA #REQUIRED param CDATA #REQUIRED
                  type  CDATA #REQUIRED>


<!ELEMENT def-rules (def-rule)*>
<!ATTLIST def-rule id CDATA #REQUIRED
                   type (defeasible|strict) #REQUIRED>
<!ELEMENT def-rule (comment?,def-head, def-body)>
<!ELEMENT def-head (arg)*>
<!ATTLIST def-head name CDATA #REQUIRED
                   negated (no|classical) "no">
<!ELEMENT arg EMPTY>
<!ATTLIST arg pos CDATA #REQUIRED value (CDATA|dash) #REQUIRED>

<!ELEMENT def-body (def-body-atom)+>
<!ELEMENT def-body-atom (arg)*>
<!ATTLIST def-body-atom name CDATA #REQUIRED
                        negated (no|classical|default) "no">

<!ELEMENT declarations (rule-instances, fact-instances)>

<!ELEMENT rule-instances (rule-instance)*>
<!ATTLIST rule-instance id CDATA #REQUIRED>
<!ELEMENT rule-instance (subst)*>
<!ELEMENT subst EMPTY>
<!ATTLIST subst param CDATA #REQUIRED value CDATA #REQUIRED>

<!ELEMENT fact-instances (fact-instance)*>
<!ELEMENT fact-instance (arg)*>
<!ATTLIST fact-instance negated (yes|no) "no"
          type (fact|assumption) "fact" name CDATA #REQUIRED>

<!ELEMENT argument (fact-instance, rule-instances)>
<!ELEMENT complete-argument
          (fact-instance, rule-instances, fact-instances)>

<!ELEMENT argument-derivation
          (fact-instance, argument-derivation*)>

<!ELEMENT argument-line (argument)*>

<!ELEMENT argument-tree (argument, argument-tree*)>
<!ATTLIST argument-tree epistemic-status
          (defeated|undefeated) #IMPLIED>
```

Figure 3: Complete listing of the DTD for the XDeLP language

Acknowledgments

This research was funded by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 No. 13.096), by CONICET (Argentina), by projects TIC2003-00950 and TIN2004-07933-C03-03 (MCyT, Spain) and by Ramón y Cajal Program (MCyT, Spain).

References

[1] BERNERS-LEE, T., HENDLER, J., AND LASSILA, O. The Semantic Web. *Scientific American* (17 May 2001).

[2] BRENA, R., CHESÑEVAR, C., AND AGUIRRE, J. Argumentation-supported information distribution in a multi-agent system for knowledge management. In *Proc. 2nd. Intl. Workshop on Argumentation in Multiagent Systems (ArgMAS). 4th Intl. AAMAS Conf., Utrecht, Holland (in press)* (July 2005).

[3] CHEONG, F. C. *Internet Agents: Spiders, Wanderers, Brokers, and Bots*. New Riders, 1995.

[4] CHESÑEVAR, C., AND MAGUITMAN, A. An Argumentative Approach to Assessing Natural Language Usage based on the Web Corpus. In *Proc. of the 16th ECAI Conf., Valencia, Spain* (Aug. 2004), pp. 581–585.

[5] CHESÑEVAR, C., AND MAGUITMAN, A. ARGUENET: An Argument-Based Recommender System for Solving Web Search Queries. In *Proc. of the 2nd IEEE Intl. IS-2004 Conference. Varna, Bulgaria* (June 2004), pp. 282–287.

[6] CHESÑEVAR, C., MAGUITMAN, A., AND LOUI, R. Logical Models of Argument. *ACM Computing Surveys 32*, 4 (Dec. 2000), 337–383.

[7] GARCÍA, A., AND SIMARI, G. Defeasible Logic Programming an Argumentative Approach. *Theory and Prac. of Logic Program. 4*, 1 (2004), 95–138.

[8] GÓMEZ, S., AND CHESÑEVAR, C. A Hybrid Approach to Pattern Classification Using Neural Networks and Defeasible Argumentation. In *Proc. of 17th Intl. FLAIRS Conference. Miami, Florida, USA* (May 2004), American Assoc. for Art. Intel., pp. 393–398.

[9] GÓMEZ, S. A., CHESÑEVAR, C. I., AND SIMARI, G. R. Embedding defeasible argumentation in the Semantic Web: an argument-based approach. *VII Workshop de Investigadores en Ciencias de la Computación* (2005), 153–157.

[10] GÓMEZ, S. A., CHESÑEVAR, C. I., AND SIMARI, G. R. Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms. *Workshop of Intelligent Techniques for Web Personalization (ITWP'05), International Joint Conference in Artificial Intelligence (IJCAI'05) (in press)* (2005).

[11] HUNHS, M. N., AND SINGH, M. P. *Readings in agents*. Morgan Kaufmann, 1998.

[12] MCGRATH, S. *XML by example. Building e-commerce applications*. Prentice Hall, 1998.

[13] MCGUINESS, D., FIKES, R., STEIN, L. A., AND HENDLER, J. DAML-ONT: An Ontology Language for the Semantic Web. In *Spinning the Semantic Web*, D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, Eds. The MIT Press, 2003, pp. 65–93.

[14] MCGUINESS, D. L., AND VAN HARMELEN, F. OWL Web Ontology Language Overview, 2004. http://www.w3.org/TR/owl-features/.

[15] PARSONS, S., SIERRRA, C., AND JENNINGS, N. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation 8* (1998), 261–292.

[16] PRAKKEN, H., AND VREESWIJK, G. Logical Systems for Defeasible Argumentation. In *Handbook of Philosophical Logic*, D. Gabbay and F.Guenther, Eds. Kluwer Academic Publishers, 2002, pp. 219–318.

[17] SIMARI, G., AND LOUI, R. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence 53* (1992), 125–157.

[18] STOLZENBURG, F., GARCÍA, A., CHESÑEVAR, C., AND SIMARI, G. Computing Generalized Specificity. *J. of N.Classical Logics 13*, 1 (2003), 87–113.