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

Sergio Alejandro Gómez ̆ャ Carlos Iván Chesñevar ̆ャ Guillermo Ricardo Simari

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
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.

### 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 \(\Pi\) and \(\Delta\) stand for sets of strict and defeasible knowledge, resp. The set \(\Pi\) 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 \(\Delta\) of defeasible knowledge involves defeasible rules of the form \(P \rightarrow Q_1, \ldots, Q_k\), 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 \(P, Q_1, Q_2, \ldots\). 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 “\(\rightarrow\)” to denote defeasible rules. Both default and classical negation are allowed (denoted \(\text{not}\) and \(\sim\), resp.). Syntactically, the symbol “\(\sim\)” is all what distinguishes a defeasible rule \(P \rightarrow Q_1, \ldots, Q_k\) 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 \text{name}(\text{\(P\)}) and \text{arity}(\text{\(P\)}) will denote the predicate name and arity, resp. We will write \(\text{Pred}(\mathcal{P})\) 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 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).

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

Strict rules:
(22) candidate(P) ← bankmanager(P).

Figure 1: Defeasible logic program $P_{bank}$ 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 $P_{bank}$ 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 trustworthy 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 $\text{Pred}(P_{bank}) = \{ \text{info/4, family_record/2, req_loan/2, credible/1, candidate/1, profile_ok/1, trustctry/2, goodincome/1, solvent/1, richfamily/1, bankmanager/1} \}$. 
2.2 Argument, Counterargument, and Defeat in DeLP

Deriving literals in DeLP results in the construction of arguments. An argument $A$ is a (possibly empty) set of ground defeasible rules that together with the set $\Pi$ 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 $A$ for a query $Q$, denoted $\langle A, Q \rangle$, is a subset of ground instances of defeasible rules in $\mathcal{P}$, such that:

1. there exists a defeasible derivation for $Q$ from $\Pi \cup A$;
2. $\Pi \cup A$ is non-contradictory (i.e., $\Pi \cup A$ does not entail two complementary literals $P$ and $\sim P$ (or $P$ and not $P$)), and,
3. $A$ 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 $\Pi \cup A'$).

An argument $\langle A_1, Q_1 \rangle$ is a sub-argument of another argument $\langle A_2, Q_2 \rangle$ if $A_1 \subseteq A_2$. Given a DeLP program $\mathcal{P}$, $\text{Args}(\mathcal{P})$ 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 $\text{no}_P$. 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 $A$ whenever $\Pi \cup A$ 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 $\langle A_1, Q_1 \rangle$ is a counterargument for an argument $\langle A_2, Q_2 \rangle$ iff

- **Subargument attack**: there is a subargument $\langle A, Q \rangle$ of $\langle A_2, Q_2 \rangle$ (called disagreement subargument) such that the set $\Pi \cup \{Q_1, Q\}$ is contradictory, or
- **Default negation attack**: a literal not $Q_1$ is present in the body of some rule in $A_2$.

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

An argument $\langle A_1, Q_1 \rangle$ is a proper defeater for an argument $\langle A_2, Q_2 \rangle$ if $\langle A_1, Q_1 \rangle$ counterargues $\langle A_2, Q_2 \rangle$ with a disagreement subargument $\langle A, Q \rangle$ (subargument attack) and $\langle A_1, Q_1 \rangle$ is strictly preferred over $\langle A, Q \rangle$ wrt $\preceq$.

An argument $\langle A_1, Q_1 \rangle$ is a blocking defeater for an argument $\langle A_2, Q_2 \rangle$ if $\langle A_1, Q_1 \rangle$ counterargues $\langle A_2, Q_2 \rangle$ and one of the following situations holds: (a) There is a disagreement subargument $\langle A, Q \rangle$ for $\langle A_2, Q_2 \rangle$, and $\langle A_1, Q_1 \rangle$ and $\langle A, Q \rangle$ are unrelated to each other wrt $\preceq$; or (b) $\langle A_1, Q_1 \rangle$ is a default negation attack on some literal not $Q_1$ in $\langle A_2, Q_2 \rangle$.

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]. However, it must be remarked that other alternative partial orders could also be valid, such us defining argument comparison using rule priorities [7].

---

6When using generalized specificity as the comparison criterion between arguments, the argument $\langle \{a \leftarrow b, c\}, a \rangle$ is preferred over the argument $\langle \{\sim a \leftarrow b\}, \sim a \rangle$ as it is considered more informed (i.e., it relies on more premises). However, the argument $\langle \{\sim a \leftarrow b\}, \sim a \rangle$ is preferred over $\langle \{(a \leftarrow b); (b \leftarrow c)\}, a \rangle$ 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 $A$ supporting the defeasible conclusion that John is a candidate for a loan, i.e., $\langle A_1, \text{candidate}(john) \rangle$, where:\footnote{For the sake of clarity, we use parentheses to enclose defeasible rules in arguments, separated by semicolons, i.e. $A = \langle \text{rule}_1 \rangle ; \langle \text{rule}_2 \rangle ; \ldots ; \langle \text{rule}_k \rangle$.}

\[
A_1 = \{ \langle \text{candidate}(john) \sim \text{profile}_{ok}(john) \rangle; \\
(\text{profile}_{ok}(john) \sim \text{goodincome}(john), \text{trustctry}(john, krakosia)); \\
(\text{trustctry}(john, krakosia) \sim \text{info}(john, krakosia, \_\_), \text{credible}(krakosia)); \\
(\text{goodincome}(john) \sim \text{info}(john, \_\_, 400), 400 > 300) \};
\]

Another argument $\langle A_2, \sim \text{goodincome}(john) \rangle$ can be derived from $P_{\text{bank}}$, supporting the conclusion that John does not have a reasonable income, with:

\[
A_2 = \{ \langle \sim \text{goodincome}(john) \rangle \sim \sim \text{solvent}(john); \langle \sim \text{solvent}(john) \rangle \sim \text{info}(john, \_\_, \text{phdstudent}, \_\_) \}.
\]

Using generalized specificity [17] as the preference criterion among conflicting arguments, it turns out that the argument $\langle A_2, \sim \text{goodincome}(john) \rangle$ is a blocking defeater for the argument $\langle A_1, \text{candidate}(john) \rangle$.

2.3 Computing Warrant through Dialectical Analysis

An argumentation line starting in an argument $\langle A_0, Q_0 \rangle$ (denoted $\lambda^{(A_0, Q_0)}$) is a sequence $[\langle A_0, Q_0 \rangle, \langle A_1, Q_1 \rangle, \langle A_2, Q_2 \rangle, \ldots, \langle A_n, Q_n \rangle \ldots]$ 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 $\langle A_i, Q_i \rangle$ is a defeater for the previous argument $\langle A_{i-1}, Q_{i-1} \rangle$ 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 $P$ and an initial argument $\langle A_0, Q_0 \rangle$, the set of all acceptable argumentation lines starting in $\langle A_0, Q_0 \rangle$ accounts for a whole dialectical analysis for $\langle A_0, Q_0 \rangle$ (i.e., all possible dialogues about $\langle A_0, Q_0 \rangle$ between proponent and opponent), formalized as a dialectical tree.

Nodes in a dialectical tree $T_{(A_0, Q_0)}$ 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 $T_{(A_0, Q_0)}$ 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 $\langle A_0, Q_0 \rangle$ is ultimately accepted as valid (or warranted) wrt a DeLP program $P$ iff the root of its associated dialectical tree $T_{(A_0, Q_0)}$ is labelled as U-node.

Given a DeLP program $P$, solving a query $Q$ wrt $P$ 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 $P$.
2. No: accounts for believing $\sim Q$ iff there is at least one warranted argument supporting $\sim Q$ on the basis of $P$.
3. Undecided: neither $Q$ nor $\sim Q$ are warranted wrt $P$.
4. Unknown: $Q$ does not belong to the signature of $P$.

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

Example 3 Consider the query $\text{candidate}(john)$ solved wrt the program $P_{\text{bank}}$(Fig. 1). As shown in Example 1, this query would start a search for arguments supporting $\text{candidate}(john)$, and argument $\langle A_1, \text{candidate}(john) \rangle$ will be found. In order to determine whether this argument is warranted, its
Figure 2: Dialectical trees for queries: (i) candidate(john) wrt $P_{bank}$; (ii) candidate/ajax) wrt $P_{bank}$, (iii) candidate(danae) wrt $P_{bank}$, and (iv) candidate(peter) wrt $P_{bank}$

dialectical tree will be computed: as shown in Example 1, there is only one (blocking) defeater for $\langle A_1, \text{candidate(john)} \rangle$, namely $\langle A_2, \sim \text{goodincome(john)} \rangle$. This defeater, on its turn, has another (proper) defeater $\langle A_3, \text{solvent(john)} \rangle$, with $A_3 = \{ (\text{solvent(john)} \leftarrow \text{info(john, _, phdstudent, _)}, \text{richfamily(john)}); (\text{richfamily(john)} \leftarrow \text{family_record(john, rich)}) \}$ 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 $\langle B_1, \text{candidate/ajax) \rangle$, which is defeated by $\langle B_2, \sim \text{goodincome/ajax) \rangle$, with

$$B_1 = \{(\text{candidate/ajax) \leftarrow \text{profile.ok/ajax)})
\text{profile.ok/ajax) \leftarrow \text{goodincome/ajax), trustctry/ajax, greece)})
\text{trustctry/ajax, greece) \leftarrow \text{info/ajax, greece, _, _)}, \text{credible(greece)})
\text{goodincome/ajax) \leftarrow \text{info/ajax, _, 350), 350 > 300)}\}$$

$$B_2 = \{(\sim \text{goodincome/ajax) \leftarrow \sim \text{solvent/ajax)})
\sim \text{solvent/ajax) \leftarrow \text{info/ajax, _, phdstudent, _)}\}$$

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

$$C_1 = \{(\text{candidate(danae) \leftarrow \text{profile.ok(danae)})
\text{profile.ok(danae) \leftarrow \text{goodincome(danae), trustctry(danae, greece)})
\text{trustctry(greece) \leftarrow \text{info(danae, greece, _)}, \text{credible(greece)})
\text{goodincome(danae) \leftarrow \text{info(danae, _, 100000), 10000 > 300)}\}$$

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 $D_1 = \emptyset \leftarrow \langle \emptyset, \text{candidate(peter)} \rangle$— 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).

8Note that there is also a second argument without defeaters supporting the query candidate(danae), namely $\langle C_2, \text{candidate(danae)} \rangle$, with $C_2 = \{ (\text{candidate(danae) \leftarrow \text{info(danae, _, 10000), req_loan(danae, 1000), 1000 < 10000*10, trustctry(danae, greece)})
\text{trustctry(danae, greece) \leftarrow \text{info(danae, greece, _), credible(greece)})\}$.}
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 $\Delta$ 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 they 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 `banana` element.

XML allows element types to be declared that can contain other elements. The expression `<!ELEMENT books (book)+>` declares that a person has a name, an address, and optionally a phone number. Zero or more occurrences can be specified as in `<!ELEMENT books (book)*>`, one or more occurrences 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 \( P_{\text{bank}} \):

\[
\text{<delp id="PBank" version="1.0">}
\]

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.:

\[
\text{<comment>Program for defining criteria for granting}
\]

\[
\text{a loan application.</comment>}
\]

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 \( \text{info(Name, Country, Profession, Income)} \) the DTD representation would be as follows:

\[
\text{<def-atom name="info" arity="4">}
\]

\[
\text{<def-arg pos="1" param="Name" type="string" />}
\]

\[
\text{<def-arg pos="2" param="Country" type="string" />}
\]

\[
\text{<def-arg pos="3" param="Profession" type="string" />}
\]

\[
\text{<def-arg pos="4" param="Income" type="float" />}
\]

\[
</def-atom>
\]

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:

\[
!ELEMENT definitions (def-language, def-rules)
\]

\[
!ELEMENT def-language (def-atom)*
\]

\[
!ELEMENT def-atom (def-arg)*
\]

\[
!ATTLIST def-atom name CDATA #REQUIRED arity CDATA #REQUIRED
\]

\[
!ELEMENT def-arg EMPTY
\]

\[
!ATTLIST def-arg pos CDATA #REQUIRED param CDATA #REQUIRED
\]

\[
!ATTLIST def-arg type CDATA #REQUIRED
\]

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(\( P \))):

\[
\text{<def-rule id="22" type="strict">}
\]

\[
\text{<def-head name="candidate" negated="no">}
\]

\[
\text{<arg pos="1" param="P" type="string" />}
\]

\[
</def-head>
\]

\[
\text{<def-body>}
\]

\[
\text{<def-body-atom name="bankmanager" negated="no">}
\]

\[
\text{<arg pos="1" value="P" />}
\]

\[
</def-body-atom>
\]

\[
</def-body>
\]

\[
</def-rule>
\]
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. The DTD definitions follow:

```xml
<!ELEMENT def-rules (def-rule)*>  
<!ATTLIST def-rule id CDATA #REQUIRED type (defeasible|strict) #REQUIRED>
<!ELEMENT def-rule (comment?, def-head, def-body)>  
<!ATTLIST def-head name CDATA #REQUIRED negated (no|classical) "no">  
<!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.

```xml
<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 $L_P$ for a given XDeLP program $P$ 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:

```xml
<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) ← info(john, krakosia, , , ), credible(karakosia),
```

we write:

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

---

Note that this corresponds to the “underscore” anonymous variable present in Prolog.
The DTD definitions follow:

```xml
<!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 \( \langle A, H \rangle \) will be represented by a fact instance \( H \) and a set \( A \) of rule instances:

```xml
<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:

```xml
<!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.
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


