Incorporating Defeasible Knowledge and Argumentative Reasoning in Web-based Forms

Sergio Alejandro Gómez[†], **Carlos Iván Chesñevar**[†‡], **Guillermo Ricardo 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: {sag,grs}@cs.uns.edu.ar

[‡]Artificial Intelligence Research Group – Department of Computer Science

Universitat de Lleida – Campus Cappont – C/Jaume II, 69 – E-25001 Lleida, SPAIN – EMAIL: cic@eps.udl.es

Abstract

The notion of forms as a way of organizing and presenting data has long been used since the beginning of the WWW. Web-based forms have evolved together with the development of new markup languages (*e.g.*, XML), in which it is possible to provide validation scripts as part of the form code in order to test whether the intended meaning of the form is correct. However, for the form designer, part of this intended meaning involves frequently other features which are not constraints themselves, but rather *attributes* emerging from the form, which provide plausible conclusions in the context of incomplete and potentially inconsistent information. As the value of such attributes may change in presence of new knowledge, we call them *defeasible attributes*. In this paper we propose extending traditional web-based forms to incorporate defeasible attributes as part of the knowledge that can be encoded in a form. The proposed extension allows the specification of scripts for reasoning about form fields using a defeasible knowledge base, expressed in terms of a Defeasible Logic Program.

1 Introduction and Motivations

The notion of form as a way of organizing and presenting data is a well-known structural abstraction for data collection, storage, and information retrieval. Forms are an important means to designing and developing user-oriented information systems, and have long been used since the very beginning of the World Wide Web. Web-based forms have evolved together with the development of new markup languages (*e.g.*, XML), in which it is possible to provide validation scripts as part of the form code in order to test whether the intensional meaning of the form is correct [Wu *et al.*, 2004].

Fulfilling the goals of the Semantic Web program [Berners-Lee *et al.*, 2001] requires having tools capable of dealing with the potential inconsistencies and incompleteness of web data sources. One particularly important application domain is e-commerce technologies, which typically demand validation of user data (*e.g.*, credit card numbers) against a set of criteria for determining if a given user is eligible for certain prospective transaction. Performing validations on field values allows to determine whether the intended meaning of such fields is coherent according to some criteria established by the form designer. Such validations usually consist of a number of hard-coded decision criteria as a portion of imperative code in a script language. However, in many cases there are some emerging features which can be inferred as part of the "intended meaning" of the form without being field values themselves. Thus, in the case of a bank loan application, the notion of "reliable client" may be inferred as plausible from knowing the annual income and banking records of a particular customer. Such features (or *attributes*) of the form are difficult to model in terms of pieces of imperative code, particularly in presence of incomplete and potentially contradictory information. The associated conclusions turn out to be *defeasible*, as they may change in the light of new information.

In this paper, we propose extending traditional web-based forms to incorporate additional attributes as part of the declarative knowledge that can be encoded in a form. As the value of such attributes may change in presence of new information, we call them *defeasible attributes*. The proposed extension allows the specification of scripts for handling defeasible attributes on the basis of a defeasible knowledge base associated with the form, expressed in terms of *Defeasible Logic Programming* (DeLP) [García and Simari, 2004], a particular formalization of defeasible argumentation [Chesñevar *et al.*, 2000] based on logic programming. We will show how this extension can be easily integrated with existing client-based approaches for handling forms, such as the use of JavaScript validation codes. The rest of this paper is structured as follows. In Section 2, we present the fundamentals of DeLP along with an example that will be used later for explaining our proposal. Section 3 describes generic issues about web-based forms as well as the importance of characterizing defeasible attributes. Section 4 introduces the notion of web-based form with defeasible attributes, or *d-forms*. We show how to encode d-forms using XDeLP, a script-like variant of DeLP oriented towards XHTML standards. Section 5 presents the notion of program redefinition. Section 6 discusses related work and finally Section 7 concludes.

2 Modelling Argumentation in DeLP

Defeasible argumentation has evolved in the last decade as a successful approach to formalize defeasible reasoning [Chesñevar *et al.*, 2000]. The growing success of

argumentation-based approaches has caused a rich cross-breeding with other disciplines, providing interesting results in different areas such as knowledge engineering, multiagent systems, and decision support systems, among others [Parsons *et al.*, 1998; Chesñevar *et al.*, 2000]. *Defeasible logic programming* (DeLP) [García and Simari, 2004] 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 [Gómez and Chesñevar, 2004], intelligent web search [Chesñevar and Maguitman, 2004b], knowledge management [Brena *et al.*, 2005] and natural language processing [Chesñevar and Maguitman, 2004a]. To make this paper self-contained, we will summarize next the fundamentals of DeLP.[1]

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.[2] *The set Δ of defeasible knowledge involves* defeasible rules *of the form $P \prec Q_1, \ldots, Q_k$, 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 *$P, Q_1, Q_2, \ldots$. 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 " $\prec$ " to denote defeasible rules. Both default and classical negation are allowed (denoted not and $\sim$, resp.). Syntactically, the symbol " $\prec$ " is all what distinguishes a *defeasible* rule $P \prec Q_1, \ldots Q_k$ 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.

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. Defeasible rules (12–13) 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 (14) says that a right profile is defined in terms of monthly income and country. Rule (15) establishes that usually all countries are trustworthy. Rule (16) says that a person P has a reasonable income if it is typically $\$300$ a month or higher. Rule (17) expresses that usually a person P who is not economically solvent does not have a reasonable income. Rules (18–19) say that usually PhD students are not solvent people unless they come from rich families. Finally, rule (20) says that people assessed by the bank with a family status "rich" are expected to be from rich families. Note that in this particular example we have $\mathsf{Pred}(\mathcal{P}_{bank}) = \{info/4,\ family\_record/2,\ req\_loan/2,\ credible/1,\ candidate/1,\ profile\_ok/1,\ trustctry/2,\ goodincome/1,\ solvent/1,\ richfamily/1\ \}$.*

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* 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 $\mathcal{P}$.*

[1] For an in-depth treatment, the interested reader is referred to [García and Simari, 2004].

[2] 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) $req\_loan(danae, 1000)$.

Facts (bank information):
(7) $family\_record(john, rich)$.
(8) $family\_record(ajax, unknown)$.
(9) $family\_record(danae, unknown)$.
(10) $credible(krakosia)$.
(11) $credible(greece)$.

Defeasible rules:
(12) $candidate(P) \prec profile\_ok(P)$.
(13) $candidate(P) \prec info(P, \_, \_, Income), req\_loan(P, Amount), Amount < Income * 10, trustctry(P, Ctry)$.
(14) $profile\_ok(P) \prec goodincome(P), trustctry(P, Ctry)$.
(15) $trustctry(P, Ctry) \prec info(P, Ctry, \_, \_), credible(Ctry)$.
(16) $goodincome(P) \prec info(P, \_, \_, Income), Income > 300$.
(17) $\sim goodincome(P) \prec \sim solvent(P)$.
(18) $\sim solvent(P) \prec info(P, \_, phdstudent, \_)$.
(19) $solvent(P) \prec info(\_, \_, phdstudent, \_), richfamily(P)$.
(20) $richfamily(P) \prec family\_record(P, rich)$.

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

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' [Simari and Loui, 1992] 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* 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* not Q_1 *in $\langle \mathcal{A}_2, Q_2 \rangle$.*

Generalized specificity [Simari and Loui, 1992] is typically used as a syntax-based criterion among conflicting arguments, preferring those arguments which are *more informed* or *more direct* [Simari and Loui, 1992; Stolzenburg *et al.*, 2003].[3] However, it must be remarked that other alternative partial orders could also be valid, such us defining argument comparison using rule priorities [García and Simari, 2004].

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:*[4]

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

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:

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

Using generalized specificity [Simari and Loui, 1992] as the preference criterion among conflicting arguments, the argument $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$ turns out to be a blocking *defeater for 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 $[\langle \mathcal{A}_0, Q_0 \rangle, \langle \mathcal{A}_1, Q_1 \rangle, \langle \mathcal{A}_2, Q_2 \rangle, \ldots, \langle \mathcal{A}_n, Q_n \rangle \ldots]$ 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 ini-

[3] 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).

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

tial 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 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

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

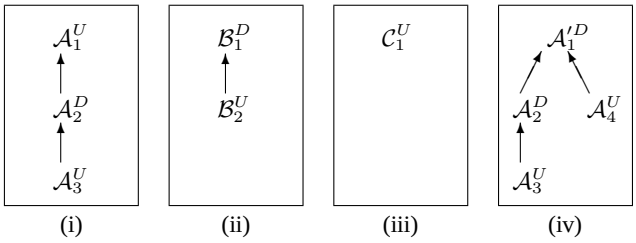

Figure 2: Dialectical trees for queries: (i) $candidate(john)$ wrt $\mathcal{P}_{bank}$; (ii) $candidate(ajax)$ wrt $\mathcal{P}_{bank}$; (iii) $candidate(danae)$ wrt $\mathcal{P}_{bank}$; and (iv) $candidate(john)$ wrt $\mathcal{P}_{bank} \triangleleft \mathcal{P}_{sec}$

Hence the associated dialectical tree for $candidate(ajax)$ has two nodes, with the root labelled as D-node (Fig. 2(i)). 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:[5]

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

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).

3 Web-based Forms: From HTML to XForms

The notion of form has been a central structural abstraction for data collection, storage, and retrieval in information management systems. Forms provide a standard way of allowing the Web user to send information back to the server by means of different technologies to verify and validate data (*e.g.*, CGI scripting). A number of programming technologies were developed, enabling the creation of interactive Web applications which outperformed static Web pages. The growing popularity of e-commerce technologies as well as the envisioning of the Semantic Web motivated the specification of sophisticated standards for web-based forms, notably XForms [Dubinko *et al.*, 2003].

As we stated in the introduction, in this paper we extend the traditional approach to web-based forms by including defeasible reasoning capabilities encoded in DeLP. In order to

[5] 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, $trustctry(danae, greece))$; $(trustctry(danae, greece) \prec info(danae, greece, \_, \_),$ $credible(greece))\}$.

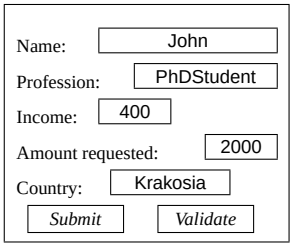

Figure 3: Form view for the loan application

do this, we will first provide a rather generic definition that captures the notion of form schema and form instance, which will prove useful for presenting our approach.

Definition 4 (Form Schema. Form Instance) *A* form schema *is a 2-uple $\mathcal{F}$=$\langle F, T\rangle$, where $F = [f_1, f_2, \ldots, f_n]$ is a list of* form fields *and $T = [T_1, \ldots, T_n]$ is a list of* types *(each of them consisting of a set of values).Given a form schema $\mathcal{F}$=$\langle F, T\rangle$ defined as above, a* form instance *based on $\mathcal{F}$ with value V (denoted $\mathcal{F}_V$) is a 2-uple $\mathcal{F}_V = \langle F, V\rangle$, where $V = [v_1, \ldots, v_n]$ is a list of values such that every $v_i \in T_i$ is the associated value for $f_i \in F$.*

Example 4 *Let F = [name, $profession$, $income$, $amountreq$, country] and T = [string, string, real, integer, string], where string, real, and integer are type names with the usual meaning. Then $\mathcal{F} = \langle F, T\rangle$ is a form schema. Let V = [john, phdstudent, 400, 2000, krakosia]. Then $\mathcal{F}_V = \langle F, V\rangle$ is a form instance based on $\mathcal{F}$.*

Figure 3 shows the typical graphical appearance of a web-based form according to the form schema given in Ex. 4. Note that control actions associated with the form (*e.g.*, submit, clear, etc.) are not considered in Def. 4.

In spite of the evolution of web-based form technologies, most form designers perform validation of form fields by enforcing constraints (*e.g.*, numeric ranges) encoded as pieces of imperative code in a scripting language (*e.g.*, JavaScript). Thus, validation of data is done client-side, and the form data is finally processed by a program located in a remote server (usually accessing some sort of database). However, in many cases there are some emerging features which can be inferred as part of the "intended meaning" of the form without being field values themselves. Thus, in the case of a bank loan application discussed in the previous sections, a concept like *reliable client*, modelled on the basis of the field values for a particular customer, could prove useful for the form designer in order to codify decision making issues associated with form processing. To identify every relevant attributes needed to infer a concept like "reliable customer" using only imperative code may be a difficult task, as in complex situations such conclusions are defeasible (particularly in presence of incomplete and potentially inconsistent information). Forms can be suitably extended to formalize such situations on the basis of DeLP by means of so-called *defeasible attributes*, as we will see in the next section.

4 Forms with Defeasible Attributes

In this section, we will outline an approach to extending traditional web-based forms to incorporate defeasible knowledge expressed in terms of a defeasible logic program, characterizing the notion of forms with defeasible attributes.

4.1 Integrating Forms with DeLP

Given a form instance $\mathcal{F}_V$, the notion of *emerging facts* from $\mathcal{F}_V$ captures the knowledge present in field values as DeLP facts, introducing new predicate names associated with those field names in a form $\mathcal{F}$.

Definition 5 (Emerging facts $facts(\mathcal{F}_V)$) *Let $\mathcal{F}$= $\langle F, T\rangle$ be a form schema, with $F = [f_1, \ldots, f_n]$, and let $\mathcal{F}_V$ be a form instance. We define the set $facts(\mathcal{F}_V)$ of* emerging facts *from $\mathcal{F}_V$ as $facts(\mathcal{F}_V)$ = $\{f_1(\mathcal{F}, v_1)$, $f_2(\mathcal{F}, v_2)$, …, $f_n(\mathcal{F}, v_n)\}$.*

Example 5 *Given the form instance $\mathcal{F}_V$ in Example 4, the corresponding set $facts(\mathcal{F}_V)$ of emerging facts is $\{name(\mathcal{F}, john)$, $profession(\mathcal{F}$, $phdstudent)$, $income(\mathcal{F}, 400)$, $amountreq(\mathcal{F}, 2000)$, $country(\mathcal{F}$, $krakosia)\}$.*

Next we will show how field values can be integrated with an arbitrary DeLP program $\mathcal{P}$, characterizing so-called *d-forms*. Formally:

Definition 6 (Form schema with defeasible attributes. D-form instance) *Let $\mathcal{F}$ = $\langle F, T\rangle$ be a form schema, and $\mathcal{P} = (\Pi, \Delta)$ a DeLP program. A* form schema with defeasible attributes *(or* d-form schema*) $\mathcal{D}$ is a 2-uple $\langle \mathcal{F}, \mathcal{P}\rangle$. If V is a set of values for the form $\mathcal{F}$, a* d-form instance *$\mathcal{D}_V$ is the 2-uple $\langle \mathcal{F}_V, \mathcal{P}\rangle$. The set of* defeasible attributes *for $\mathcal{D}_V$ is defined as the set of predicates* $\mathsf{Pred}(\Pi \cup facts(\mathcal{F}_V), \Delta)$.

Given a d-form instance $\langle \mathcal{F}_V, \mathcal{P}\rangle$, the above definition aims at identifying features or attributes encoded by the form designer as predicates in the program $\mathcal{P}$. Such attributes are defeasible, as their associated value will be determined by DeLP queries solved wrt the DeLP program $(\Pi \cup facts(\mathcal{F}_V), \Delta)$. Hence, changing the field values in the form $\mathcal{F}$ or changing the underlying DeLP program $\mathcal{P}$ will result in changing the value for these attributes. Defeasible attributes will represent relevant features for the form designer, whose value depends on both the DeLP program encoding relevant domain knowledge and the particular field values for a given form instance.

Example 6 *Let $\mathcal{F} = \langle F, T\rangle$ be the form given in Example 4, and consider the program $\mathcal{P}_{bank}$' = $\mathcal{P}_{bank}\backslash\{(1), \ldots, (6)\}$ $\cup$ $\{$ $info(N, C, P, I) \leftarrow name(\mathcal{F}, N)$, $country(\mathcal{F}, C)$, $profession(\mathcal{F}, P)$, $income(\mathcal{F}, I)\}$ $\cup$ $\{$ $req\_loan(N, A) \leftarrow name(\mathcal{F}, N)$, $amountreq(\mathcal{F}, A)$ $\}$; i.e., the program given in Fig. 1 excluding user-provided information, as well as two additional strict rules linking the form schema $\mathcal{F}$ with the DeLP program rules.*

Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{P}_{bank}{}'\rangle$ be a d-form. According to the DeLP program $\mathcal{P}_{bank}{}'$, one defeasible attribute in $\mathcal{D}$ is $candidate/1 \in \mathsf{Pred}(\mathcal{P}_{bank}{}')$. Suppose now that three users John, Ajax, and Danae fill in this d-form as described in Example 4. For every user a particular d-form instance

$\mathcal{D}_{user}$ would be obtained. Thus, when analyzing, for example, the query "$candidate(john)$", the d-form instance $\mathcal{D}_{john}$ would involve providing all his particular user details, which will be present as emerging facts. Along with the two additional strict rules given above, reasoning from $\mathcal{P}_{bank}{}' \cup facts(\mathcal{D}_{john})$ would result in the dialectical analysis shown in Example 3, determining that $candidate(john)$ is warranted. The same applies for the other two users with respect to $candidate(ajax)$ and $candidate(danae)$, resp.

4.2 Characterizing D-forms as DeLP Scripts in XML

In order to use d-forms in the context of web applications, we propose codifying a defeasible logic program as a web document. For doing this, we have defined *XDeLP*, a scripting language that combines features from markup languages and DeLP. XDeLP supports the representation of defeasible knowledge bases by aumenting XHTML with tags that allow to represent defeasible logic programs. XDeLP can be embedded directly in XHTML documents or used in XML documents. This decision provides several advantages as noted by [Heflin *et al.*, 2003] in the context of SHOE—(1) web authors are more confortable with XML syntax as there are many commercial applications to edit it, (2) its knowledge contents can be used in other XML aware applications, and (3) the XSLT style sheet standard [Clark, 1999] can be used to render it for human consumption.

In XDeLP syntax, a defeasible knowledge base or defeasible logic program appears between the tags <*delp id="..." version="...*> and </*delp*> and is identified by the combination of *id* and *version*. A defeasible knowledge base can define facts, strict rules, and defeasible rules by including special tags for these purposes. Figure 4 shows rules for defining a schema for facts of the form $req\_loan(Name, Amount)$, where $req\_loan(john, 2000)$ is a particular instance. Also, it shows a schema definition for the rule (15) of Ex. 1, and a ground instance of this rule that says that Krakosia is not a trustworthy democracy because it is at war.

As mentioned before, programmers usually validate form data by attaching some imperative JavaScript code to buttons. Our proposal for forms with defeasible attributes involves defininig a XML-based tag language for codifying DeLP programs defeasible knowledge base attached to a d-form, as an integrated part in a web-based form, integrating the DeLP inference engine to a web browser and extending the JavaScript programming language with primitives for invoking the DeLP engine. The architecture for the approach is depicted in Fig. 5. The extension to JavaScript consists of primitives for calling the DeLP engine services. This is implemented through specialized built-in boolean functions like $warranted(formid, h)$ that determine *e.g.* if there exists a warranted literal h wrt form $formid$. Similar functions are implemented for other possible values for defeasible attributes (*e.g.*, $undecided$). Next we show an example of how the proposed approach works in a JavaScript client-side script.

Example 7 *Suppose $\mathcal{P}$ is a d-form as described in Ex. 4, which in turn is written in XForms and has $form1$ as its identifier. Then, a JavaScript programmer would be capable of writing the following code embedded in a handler function for the* Validate *button such as in:*

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

<!-- EXAMPLE OF ATOM DEFINITION: req_loan(Name, Amount)-->
<def-atom name="req_loan" arity="2">
   <def-arg pos="1" param="Name" type="string" />
   <def-arg pos="2" param="Amount" type="float" />
</def-atom>
...
<!-- EXAMPLE OF FACT: req_loan(john, 2000) -->
<fact-instance negated ="no" name="req_loan">
    <arg pos="1" value="john" />
    <arg pos="2" value="2000" />
</fact>
...
<!-- EXAMPLE OF DEFEASIBLE RULE:
 A democracy at war usually is not a credible country -->
<def-drule id="15">
    <def-head name="credible" negated="yes">
      <arg pos="1" param="Ctry" type="string" />
    </def-head>
    <def-body>
       <def-body-atom negated="no" name="country">
          <arg pos="1" param="Ctry" type="string" />
          <arg pos="2" param="Status" type="string"
               value="democracy" />
       </def-body-atom>
       <def-body-atom negated="no" name="country">
          <arg pos="1" param="Ctry" type="string" />
          <arg pos="2" param="Status" type="string"
               value="atwar" />
       </def-body-atom>
    </def-body>
</def-drule>

<!-- EXAMPLE OF GROUND RULE: Krakosia is not a trustworthy
     democracy because it is at war. -->
<drule-instance id="15">
   <subst param="Ctry" value="krakosia" />
</drule-instance>
</delp>
```

Figure 4: XML syntax for XDeLP

```
<script language="JavaScript">
function validate()
{
 if( form1.warranted(p, candidate(form1.name.value)) )
    alert( "The requested loan will be probably conceded."
           + "We will contact you in a week." );
 else
    alert( "Your case will be analyzed and " +
           "we will contact you in a month." );
}
</script>
```

5 Redefining DeLP Programs

As stated in Section 2, a DeLP program $\mathcal{P}$ (Def. 1) can also be thought of as a set of predicate definitions, *i.e.* $\mathcal{P} =_{def} \{P_1^{\mathcal{P}}, \ldots, P_k^{\mathcal{P}}\}$. Thus, in our example concerning bank loans, the program $\mathcal{P}_{bank}$ (Fig. 1) provides the definition of a number of predicates ($candidate$, $trustctry$, etc.). This alternative conceptualization will allow us to define the notion of *redefinition*. A redefinition of a program $\mathcal{P}_1$ wrt another program $\mathcal{P}_2$ is a new DeLP program $\mathcal{P}$ that includes all predicate definitions in $\mathcal{P}_1$ and $\mathcal{P}_2$, except for those predicates in $\mathcal{P}_1$ which are also defined in $\mathcal{P}_2$. Formally:

Definition 7 (Redefinition) *Let $\mathcal{P}_1, \mathcal{P}_2$ be two DeLP programs, such that $\mathcal{P}_1$ defines the predicates R_1, R_2, ..., R_n,*

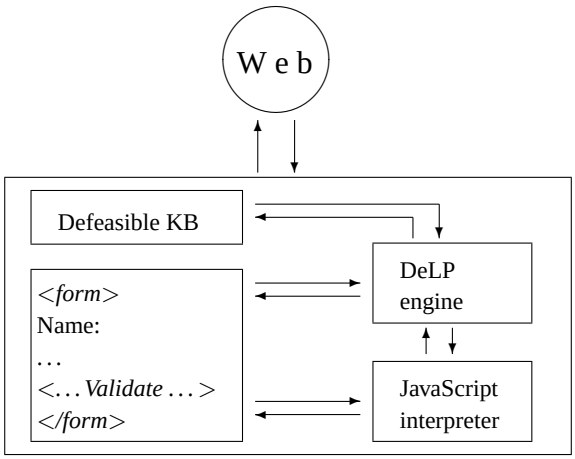

Figure 5: A framework for embedding the DeLP inference engine in a browser application

and $\mathcal{P}_2$ defines the predicates $S_1, \ldots, S_m$. The redefinition *of $\mathcal{P}_1$ wrt $\mathcal{P}_2$, denoted $\mathcal{P}_1 \triangleleft \mathcal{P}_2$, is a new program $\mathcal{P}'$ defined as follows:*
$\mathcal{P}' = \mathcal{P}_1 \triangleleft \mathcal{P}_2 =_{def} \{R_1^{\mathcal{P}_1}, \ldots, R_n^{\mathcal{P}_1}\} \cup \{S_1^{\mathcal{P}_2}, \ldots, S_m^{\mathcal{P}_2}\} \setminus \{ R_i^{\mathcal{P}_1} \mid \exists S_j^{\mathcal{P}_2}$ *in $\mathcal{P}_2$, with* $name(R_i) = name(S_j)$, *and* $arity(R_i)$=$arity(S_j)\}$.

Redefining a DeLP program basically involves providing new predicate definitions, which supersede already existing ones (if any). Let us suppose that the bank gets a number of basic criteria from the Homeland Security Office (HSO) concerning how to assess trustworthiness of countries. Such criteria could be encoded by HSO programmers in DeLP as shown in Fig. 6. Assuming that the bank wants to merge this knowledge base with theirs, the resulting redefined program $\mathcal{P}_{bank} \triangleleft \mathcal{P}_{sec}$ would consider a more detailed analysis for countries, as factors such as political system, political situation, etc. would be taken into account when conceding loans, as shown in the following example.

Example 8 *Consider the DeLP programs $\mathcal{P}_{bank}$=$\{(1), \ldots, (20)\}$ and $\mathcal{P}_{sec}$=$\{(1'), \ldots, (6')\}$ from Fig. 1 and 6, resp. Computing $\mathcal{P}_{bank} \triangleleft \mathcal{P}_{sec}$ gives as a result a new DeLP program $\mathcal{P}'$ = {(1), ..., (20)} $\cup$ {(1'), ..., (6')} $\setminus$ {(10),(11)}, in which the definition of* $credible$ *provided by $\mathcal{P}_{bank}$ is replaced by the new definition given in $\mathcal{P}_{sec}$. Solving the query "*$candidate(john)$*" wrt $\mathcal{P}'$ involves a search for arguments similar to the one perfomed in Example 3: an argument $\langle \mathcal{A}_1', candidate(john) \rangle$ supports the query* $candidate(john)$,[6] *with*

$\mathcal{A}_1'$ = { $(candidate(john) \prec profile\_ok(john))$;
$(profile\_ok(john) \prec goodincome(john),$
$trustctry(john, krakosia))$;
$(trustctry(krakosia) \prec info(john, krakosia, \_, \_),$
$credible(krakosia))$;
$(credible(krakosia) \prec country(krakosia, democracy))$;
$(goodincome(john) \prec info(john, \_, \_, 400), 400 > 300)$ }.

[6]Note that argument $\langle \mathcal{A}_1', candidate(john) \rangle$ involves defeasible information about Krakosia coming from $\mathcal{P}_{sec}$, in contrast with the original argument $\langle \mathcal{A}_1, candidate(john) \rangle$.

$(1')$ $country(greece, democracy)$
$(2')$ $country(krakosia, democracy)$
$(3')$ $country(krakosia, atwar)$
$(4')$ $credible(Ctry) \prec country(Ctry, democracy)$.
$(5')$ $\sim credible(Ctry) \prec$
$country(Ctry, democracy)$,
$country(Ctry, atwar)$.
$(6')$ $\sim credible(Ctry) \prec$
$country(Ctry, democracy)$,
$country(Ctry, corruptgovt)$.

Figure 6: Defeasible logic program $\mathcal{P}_{sec}$ from the HSO

*As in Example 3, this argument is defeated by another argument $\langle \mathcal{A}_2, \sim goodincome(john) \rangle$, which on its turn is defeated by another argument $\langle \mathcal{A}_3, solvent(john) \rangle$. In all these arguments, however, the redefined program allows a fourth argument to be inferred, namely $\langle \mathcal{A}_4, \sim credible(krakosia) \rangle$ with $\mathcal{A}_4$=$\{ \sim credible(krakosia) \prec country(krakosia, democracy), country(krakosia, atwar)$ }, *which is a proper defeater for $\langle \mathcal{A}_1', candidate(john) \rangle$. As a result, the root of the dialectical tree for the query "$candidate(john)$" is marked as D-node, as shown in Fig. 2(iv).*

Note that redefining a program will usually result in providing *more specific* information associated with particular predicates. Thus, arguments in a program $\mathcal{P}_1$ which had a particular epistemic status (e.g. warranted) may no longer keep it in a redefined version $\mathcal{P}_1 \triangleleft \mathcal{P}_2$.

6 Implementation Issues and Related Work

Performing defeasible argumentation is a computationally complex task. An abstract machine for an efficient implementation of DeLP has been developed, based on an extension of the WAM (Warren's Abstract Machine) for Prolog [García and Simari, 2004]. On the basis of this abstract machine a Java-based integrated development environment was then implemented, which was used for our experiments as a prototype of the embedded DeLP engine in a web browser.

To the best of our knowledge there are no other works in the area of introducing defeasible knowledge in web-based forms as done in this paper. Recent research [Wu *et al.*, 2004] has been focused on developing a methodology for designing form-based decision support systems, which uses factoring and synthesis to process knowledge involved in forms. The resulting framework allows flexible creation and modification of computer-generated forms useful for decision making and suited for simplifying the process of report generation. However, even though this approach exploits the semantics of the knowledge involved in forms, it does not provide any connection with web-based systems nor with handling defeasible knowledge. In similar direction to our work, rule-based defeasible reasoning in the context of the Semantic Web has been motivated the development of alternative systems such as DR-DEVICE [Bassiliades *et al.*, 2004], which is capable of reasoning about RDF metadata over multiple Web sources using defeasible logic rules [Antoniou *et al.*, 2001; 2004]. In contrast with our approach, this system is implemented on top of CLIPS production rule system, whereas

our proposal relies on the computation of warrant performed by the DeLP inference engine using backward reasoning and depth-first search. Furthermore, comparison among rules in defeasible logic is performed on the basis of a superiority relationship, whereas our proposal relies on a modular comparison criterion among arguments. Besides, DeLP does not need to be supplied with defeater rules because the system will find all possible counterarguments automatically on the basis of the arguments it is able to build, and will decide on the defeat relation using the DeLP comparison criterion. Thus, a DeLP programmer does not need to encode exceptions explicitly.

7 Conclusions and Future Work

We have presented a novel argument-based approach for enriching traditional forms for web-based environments, which can be suitably adapted to existing markup language technologies like XHTML. As discussed in the introduction, our proposal involves providing the possibility of modelling inferences based on concepts which are part of the intended meaning of a form, which we have formalized as defeasible attributes.

We have shown that the use of an embedded DeLP interpreter on the client side allows the form designer to develop richer form schemas, in which the interaction of defeasible attributes is taken into account as part of the "behavior" of the form. Knowledge bases for forms are expressed in a declarative way, making easier to enrich a form by e.g. merging two existing knowledge bases. Implementing program redefinition as described in Section 5 is quite straightforward, and offers an attractive possibility for integrating defeasible knowledge bases from different sources (as $\mathcal{P}_{bank}$ and $\mathcal{P}_{sec}$). Clearly, additional ontological considerations (*e.g.* unique name assumption, etc.) are required for such merging operations; extending our formalization to handle such considerations is part of our current research work. The sample problem presented in this paper was encoded using a Java-based DeLP interpreter and solved successfully under the methodology we have described. However, our experiments regarding this approach only account as a "proof of concept" prototype, as we have not been able yet to carry out thorough evaluations in the context of real-world applications. Research in this direction is currently being pursued.

Acknowledgments

The authors would like to thank anonymous reviewers for their suggestions to improve the original version of this paper. 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

[Antoniou *et al.*, 2001] G. Antoniou, D. Billington, G. Governatori, and M. Maher. Representation results for defeasible logic. *ACM Trans. on Comp. Logic*, 2(2):255–287, 2001.

[Antoniou *et al.*, 2004] G. Antoniou, A. Bikakis, and G. Wagner. A system for nonmonotonic rules on the web. *LNCS 3323 (Proc. of RuleML2004)*, pages 23–26, 2004.

[Bassiliades *et al.*, 2004] N. Bassiliades, G. Antoniou, and I. Vlahavas. A defeasible logic reasoner for the semantic web. In *Proc. of the Workshop on Rules and Rule Markup Languages for the Semantic Web*, pages 49–64, 2004.

[Berners-Lee *et al.*, 2001] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scient. American*, 2001.

[Brena *et al.*, 2005] R. Brena, C. Chesñevar, and J. Aguirre. Argumentation-supported information distribution in a multiagent 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.

[Chesñevar and Maguitman, 2004a] C. Chesñevar and A. Maguitman. An Argumentative Approach to Assessing Natural Language Usage based on the Web Corpus. In *Proc. of the 16th ECAI Conf., Valencia, Spain*, pages 581–585, August 2004.

[Chesñevar and Maguitman, 2004b] C. Chesñevar and A. Maguitman. ARGUENET: An Argument-Based Recommender System for Solving Web Search Queries. In *Proc. of the 2nd IEEE Intl. IS-2004 Conference. Varna, Bulgaria*, pages 282–287, June 2004.

[Chesñevar *et al.*, 2000] C. Chesñevar, A. Maguitman, and R. Loui. Logical Models of Argument. *ACM Computing Surveys*, 32(4):337–383, December 2000.

[Clark, 1999] J. Clark. XSL Transformations (XSLT) Version 1.0. W3C Recommendation 16 Nov. 1999, 1999.

[Dubinko *et al.*, 2003] M. Dubinko, L. Klotz, R. Merrick, and T.V. Raman. XForms 1.0 - W3C Recomm. 14 Oct. 2003, 2003.

[García and Simari, 2004] A. García and G. Simari. Defeasible Logic Programming an Argumentative Approach. *Theory and Prac. of Logic Program.*, 4(1):95–138, 2004.

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

[Heflin *et al.*, 2003] J. Heflin, J. Hendler, and S. Luke. SHOE: A Blueprint for the Semantic Web. pages 29–63, 2003.

[Parsons *et al.*, 1998] S. Parsons, C. Sierrra, and N. Jennings. Agents that Reason and Negotiate by Arguing. *Journal of Logic and Computation*, 8:261–292, 1998.

[Simari and Loui, 1992] G. Simari and R. Loui. A Mathematical Treatment of Defeasible Reasoning and its Implementation. *Artificial Intelligence*, 53:125–157, 1992.

[Stolzenburg *et al.*, 2003] F. Stolzenburg, A. García, C. Chesñevar, and G. Simari. Computing Generalized Specificity. *J. of N.Classical Logics*, 13(1):87–113, 2003.

[Wu *et al.*, 2004] J. Wu, H. Doong, C. Lee, T. Hsia, and T. Liang. A methodology for designing form-based decision support systems. *Decision Support Systems*, (36):313–335, 2004.