A Defeasible Reasoning Web Service

Nicolás D. Rotstein Fernando M. Sagui Alejandro G. Stankevicius
Alejandro J. García Guillermo R. Simari

Artificial Intelligence Research and Development Laboratory
Department of Computer Science and Engineering
Universidad Nacional del Sur,
Bahía Blanca - Buenos Aires - ARGENTINA
e-mail: {ndr, fms, ags, ajg, grs}@cs.uns.edu.ar

ABSTRACT

In this article we outline a research line whose main goal is to give access to the software agents inhabiting the Web to a powerful inference service built around Defeasible Logic Programming (DeLP), a formalism that combines features of Logic Programming with Defeasible Argumentation. To do so, the web service prototype we proposed is the next logical step to take in order to allow agents to draw conclusions through DeLP within the Semantic Web.

Keywords: web services, defeasible reasoning, dialectical argumentation, semantic web.

1 INTRODUCTION

The World Wide Web Consortium defines Web Services as *"software systems designed to support interoperable machine-to-machine interaction over a network"* [5]. It has an interface that is described in a machine–processable format so other systems can interact with it in a manner prescribed by its interface. Agents can use web services to communicate with other agents or with software applications written in different programming languages and running on various platforms over the Internet. This interoperability is due to the use of open standards and protocols. Data can be transported using common protocols, such as HTTP, FTP, SMTP, *etc.*, that conforms the Web Service Protocol Stack.

Defeasible Logic Programming (DeLP) [2] is a formalism that combines features of Logic Programming with Defeasible Argumentation. DeLP provides the possibility of representing tentative information in the form of *weak rules*, and strong knowledge in the form of *strict rules*. A defeasible argumentation inference mechanism is used for warranting the entailed conclusions. Queries are supported by arguments, and the inference mechanism provides an answer according to the outcome of the warrant process. This process yields one of four possible answers: YES, NO, UNDECIDED, and UNKNOWN. An online implementation of DeLP can be found at: `http://lidia.cs.uns.edu.ar/delp`.

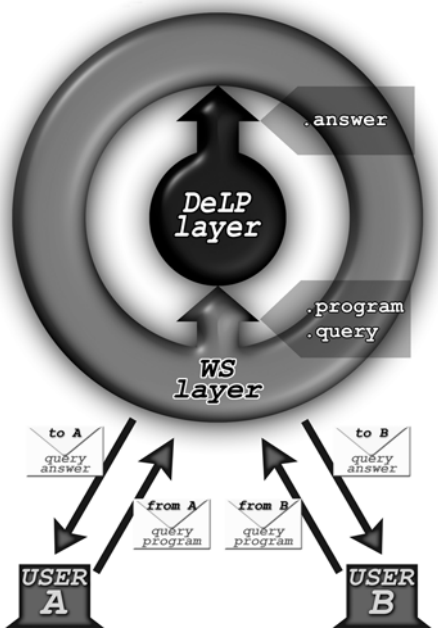

Figure 1: DeLP and WS layers

The web service prototype introduced in this article is the next logical step to take in order to allow agents to draw conclusions through DeLP over the Semantic Web [1]. Currently, our prototype is based on a layered approach (briefly sketched in Figure 1, further explained in Section 2.2), where clients interact with the Web Service Layer, which, in turn, queries the DeLP Layer to serve each client. This layered design allows us to update the system in a modular fashion, keeping apart the different components.

Partially supported by Agencia Nacional de Promoción Científica y Tecnológica (PICT 2002 Nro. 13096, PICT 2003 Nro. 15043, PAV 2003 Nro. 076), CONICET (*Consejo Nacional de Investigaciones Científicas y Técnicas de la República Argentina*), and CIC.

In what follows, we address (i) how to enhance the interactions among layers, and (ii) outline the requirements of a hypothetical client in an ideal scenario. This scenario is deemed *ideal* in the sense that the client can expect to receive information about the argumentation process that is not currently available in our implementation.

2 ARCHITECTURE

Briefly stated, our main goal is to give access to the software agents inhabiting the web to a powerful inference service built around DeLP. This research line requires that we define how these agents get access to DeLP's inference engine. We start by considering an ideal scenario within which a hypothetical client interact with this service, and then in section 2.2 we present what has been implemented so far. To avoid confusions, both 'client' and 'agent' will be understood as the same (*i.e.,* the entity making use of our service).

2.1 Ideal scenario

In an ideal scenario, the client would expect to have a completely transparent interaction with it, sending requests with a minimum of information, and receiving replies with a maximum of information (regarding the outcome of the process performed). Clearly, the client has to be able to establish a session with the service (that will maintain the state of execution), in order to start interacting with it. Then, the client should be able to provide (i) queries, (ii) knowledge bases, and (iii) commands.

A *request* involve the evaluation of a query against a knowledge base (*i.e.,* a defeasible logic program (*de.l.p.*)), no matter whether the latter is in possession of the agent or not. Having several agents turning over their own knowledge bases to the Web Service ought to be handled with security in mind. Naturally, these secure transfers must be completely transparent to the client.

The interaction between the client and our service is based on the following *requests*:

- **Query:** a term (possibly with unbound variables in it) which will be answered according to the current knowledge base.
- **Knowledge Base (KB):** it conveys the agent knowledge, and there might be more than one.
- **Command:** it modifies the current state of execution of the service.

Note that the service provided should be capable of *joining* several knowledge bases. This is useful, for instance, when the agent has two separated modules (*i.e., de.l.p.*'s), one containing codified background knowledge, and another module modelling perceptions. Since perceptions are likely to be dynamic, it makes sense to have them as a separate module. Hence, whenever the client poses a query, it has to send both modules, which should be automatically joined.

Regarding *commands*, agents can adjust several aspects of the inference being carried out by this web service:

- **Preference criteria**: the client might want to set which criteria defines the defeat relation being used.
- **Double-block check**: the client may set it on or off. This concerns a particular aspect of the warrant process.
- **Strict rules consistency check:**[1] the client can also define whether the strict rules set consistency check is required or not.
- **Remove knowledge base**: the client can request to remove a given knowledge base.

The Web Service maintains the state of execution of every agent connected with it to allow them to personalize its usage by sending commands to it. Without state, the service will have no memory, rendering commands useless.

For example, lets suppose that we have a software broker that helps us in the task of finding the best plane to travel between two given cities. The knowledge will be encoded in the form of a *de.l.p.*, stating reasons for and against choosing a certain flight. The broker will seek for the available flights between those cities, along with their description (stages, cost, and so forth). Then, according to its knowledge, a certain flight will be selected as the best choice. Since the reasons for and against choosing a certain flight must be weighed with respect to a preestablished preference criterion, it should also be capable of selecting that preference among a pool of argument preference criteria available on the web service. Briefly put, the software broker follows these four steps:

1. Establish the session with the web service.
2. Send both the background knowledge and the information about the available flights gathered.
3. Send a command establishing the desired preferences criterion (for instance, it may

[1]The argumentative formalism upon which DeLP is based requires the strict part of the program to be consistent. Whenever this condition does not hold, either the program is modified to accomplish it, or the answers computed by DeLP become unreliable.

prefer a cheap flight over one with many stages).

4. Query the service about what flight to take.

In this context, a *reply* involves not only getting the answer for the query just performed, but also getting that query grounded (if it had unbound variables), and the explanation for the answer. Note that this explanation consists of a set of dialectical trees: precisely, those built while generating the answer. The format of this set of dialectical trees must be standardized, to avoid agents having to *know* the way those trees were constructed. Lets assume that, in this ideal scenario, an ontology of dialectical trees (including the notions of arguments, attacks, and so forth) is available. Thus, the agent will be aware of what a dialectical tree is, what an argument is, what an attack is, *etc.*, as well as the relation among these. Therefore, with this information at hand, the agent will be able to reason about the explanations to the answers received, something quite useful to have when trying to understanding what is happening in the world and what is being warranted. Note that an agent could also infer the reason why something is not being warranted. These *replies* should carry the following information:

- **Answer:** either YES, NO, UNDECIDED, or UNKNOWN.
- **Ground Query:** the service provides, whenever possible, the instantiation of the unbound variables of the query.
- **Dialectical Trees:** the set of dialectical trees built; codified in the form of an agreed, public ontology.

To sum up, a typical agent may establish a connection with the Web Service, send a couple of knowledge bases and start querying over them. Later, the agent may decide to remove one of the knowledge bases and send a new one, set off the consistency check, perform another set of queries, and so on.

Finally, the agent might encounter some exceptional conditions while interacting with the service; namely, communication errors, lack of knowledge base consistency, *etc.* In order for the client to handle any problem that might arise, the web service should keep him informed about the occurrence of the following *errors*:

- **Communication error:** SOAP and/or XML encoding errors, or timeouts.
- **Knowledge Base Join error:** signalling an error while joining several knowledge bases.
- **Consistency error:** arising when a given program fails the consistency check.

Having these exceptional conditions categorized allows the clients to behave consequently. Sometimes, it will be capable of fixing the error (*e.g.,* if the program transferred is intrinsically inconsistent), or not (*e.g.,* if there was a broken communication link).

Regarding consistency, there are two possible sources of contradiction within an agent knowledge: (i) when the contradiction arises from the join of knowledge bases performed by the DeLP Layer, and (ii) when individual programs are inconsistent. When a program is inherently contradictory, it will be considered as a *Consistency* error; when a set of programs is joined into a new program that turns out to be contradictory, it will be considered as a *KB Join* error. Note that the inclusion of an inconsistent program in a join will trigger a Consistency error, which, in turn, will throw a KB Join error.

Should any of these exceptional situations happen, the client may expect further information about it (besides the type of error just encountered). Hence, the Web Service also provides the following information:

- **Type of error:** stating the type of error encountered.
- **Description:** where the web service provide any additional information the client may need to overcome this situation.

Each error type we mentioned can be triggered by several situations. The following table relates some types of error with short descriptions about the possible situations that cause them:

Error Type	**Description**
Communication	Network failure, Malformed message, Timeout
KB Join	A set of literals in contradiction
Consistency	Pairs of the form: *(Program, Set of literals in contradiction)*

Since errors are handled in a modular fashion, it is easy to add new error types and/or descriptions, as well as modify the existent ones.

After an error is triggered, it is up to the agent to deal with the problem. For instance, if a KB Join error arises, the agent could have a policy that turns the strict rules under conflict into defeasible rules. Sometimes, the problem has nothing to do with reasoning issues, like an error that occurred because of a timeout. Here, the agent cannot do anything but to wait and resend the message.

2.2 Current Prototype

The web service we have implemented is in its first stage of development. It is composed by two layers: the inner called DeLP Layer, and the outer Web Service Layer (shown above in Figure 1). Clients interact only with the Web Service (WS) Layer, and the interaction between layers (shown in Figure 2) is totally transparent to them.

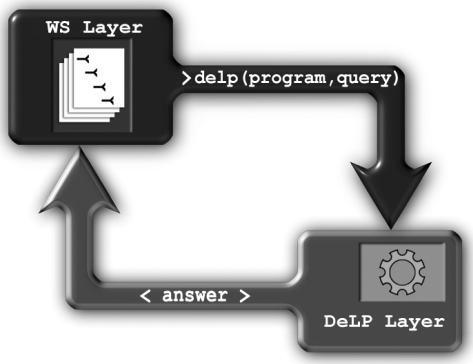

Figure 2: Interaction between DeLP and WS layers

The DeLP Layer is the reasoning core of the web service. It receives a single defeasible logic program and a query as input, and returns the answer for it. Whenever the agent wants to query a program, it has to send both program and query in the request to the web service. Program consistency issues and syntax checking are addressed by this layer. When an error occurs, it is communicated to the outer layer by reutilizing the answer and query response fields.

The DeLP Layer also generates the necessary information to draw the dialectical trees that support the argumentative process performed. This data is later used by an external application (or applet) to draw these trees.

The WS Layer is the interface between the reasoning module and the clients. In order to bring the reasoning module to the web, we advertise it as a web service. This layer receives queries and programs from agents and forwards them to the inner layer; afterwards, the latter sends the response to the outer layer, and the answer finally arrives back to the clients. The WS Layer handles the peer-to-peer communication with clients, and manages the individual environments for each of them. This layer provides a standard interface to the clients; hence, clients might be implemented in any language that supports interaction with web services via standard protocols (like SOAP[3], WSDL[6], UDDI[4]).

3 IMPROVEMENTS

A first prototype of the model overviewed in this work was implemented in order to have a running version of it. This is a first step towards the ideal scenario described in the previous section.

In this work, we presented both the current prototype and the scenario we want to achieve. To shorten the gap between them, we identified a series of improvements that ought to be implemented; some of them involve interaction between layers, and others, between the client and the web service. These improvements are summarized as follows:

- The current implementation does not maintain the agents state of execution which is needed to support file (knowledge base) transfers, commands and successive querying. We have to implement a robust session management model that allows keeping the state of every connected client.

- Security issues regarding the transport of knowledge bases were not addressed in the first versions of the system. Some security mechanisms must be implemented, given that the transfer of these files is essential to the functioning of the service.

- Our prototype does not support the transfer of more than one knowledge at once. We would like to drop this restriction, so to increase the service flexibility. Therefore, an updated version of the service should include the possibility of not only sending several knowledge bases at the same time, but also add the capability of joining them into a single *de.l.p.* in a transparent way.

- Regarding error management, the interaction between layers may be improved. The current prototype reutilizes the answer and query response fields to communicate the outer layer about the occurrence of an error. We wish to improve the error handling model, by separating the error output from the query–answer output. The objective is to achieve the type–description schema presented in the ideal scenario mentioned above.

- The scalability of the system ought be tested in the real world, and improved, if necessary. Replication and physical distribution of the layers are options that must be assessed.

4 SUMMARY & CONCLUSIONS

Web Services are a powerful tool to standardize and distribute different types of services over the Web. In this paper, we intend to bring the Defeasible Logic Programming reasoning to the Web. This a first step towards the integration of DeLP in the Semantic Web. By doing so, we initiating a new research line that takes this work as a starting point, aiming to develop higher level tools,

like ontology-based reasoners, mobile agents, and so forth.

References

[1] Berners-Lee, T., Hendler, J., and Lassila, O. The semantic web. *Scientific American 284*, 5 (2001), 35–43.

[2] García, A. J., and Simari, G. R. Defeasible logic programming: An argumentative approach. *Theory and Practice of Logic Programming 4*, 1 (2004), 95–138.

[3] SOAP. Simple object access protocol. `http://www.w3.org/TR/soap/`.

[4] UDDI. Universal description, discovery, and integration. `http://www.uddi.org/`.

[5] W3C. World wide web consortium. `http://www.w3.org/`.

[6] WSDL. Web services description language. `http://www.w3.org/TR/wsdl`.