DR-BROKERING: A Semantic Brokering System

Grigoris Antoniou
Institute of Computer Science, FORTH, Greece
Department of Computer Science, University of Crete, Greece
antoniou@ics.forth.gr

Thomas Skylogiannis, Antonis Bikakis
Department of Computer Science, University of Crete, Greece
{dogjohn,bikakis}@csd.uoc.gr

Martin Doerr
Institute of Computer Science, FORTH, Greece

Nick Bassiliades
Department of Informatics, Aristotle University of Thessaloniki, Greece
nbassili@csd.auth.gr

Abstract. In this paper we study the brokering and matchmaking problem, that is, how a requester's requirements and preferences can be matched against a set of offerings collected by a broker. The proposed solution uses the Semantic Web standard of RDF to represent the offerings, and a deductive logical language for expressing the requirements and preferences. We motivate and explain the approach we propose, and report on a prototypical implementation exhibiting the described functionality in a multi-agent environment.

Keywords: Brokering, Rules, Ontologies, RDF Schema

1. Introduction

E-Commerce describes the revolution that is currently transforming the way business is conducted through the use of information technology, and in particular the World Wide Web. According to [22], in the 1st generation e-Commerce applications (current state), buyers and sellers are *humans* who typically browse through a catalogue of well-defined commodities (e.g. flights, books…) and make fixed price purchases usually by means of credit card transactions. Humans are in the loop at all stages of buying process something which is time consuming.

The 2nd generation of e-Commerce will be realized through the use of automated methods of information technology. Web users will be represented by *software agents*. According to [27], there is an increasing use of software agents for all the aspects of e-Commerce.

As software agents start to engage in e-commerce, new issues arise. Information must be organized in a way that is accessible by both humans and machines. Additionally, machines must be able to access, process and interpret the information

in the same way. This vision is consistent with the *Semantic Web* initiative [10], which enriches the current Web through the use of machine-processable information about the meaning (semantics) of information content. This way, the meaning of displayed information is accessible not only to humans, but becomes also accessible to software agents.

The key techniques of the Semantic Web are *semantic annotations (meta-data*), such that Web information carries its meaning on its sleeve, and *ontologies* which organize terms in a conceptualization of a domain, thus connecting semantic annotations with each other and serving as a basis for interoperability.

The focus of the present work is related to semantic-based electronic markets. Semantics-based electronic markets help both service providers and requesters to match their interests. The key operations in such markets are to:

(a) Identify appropriate services that satisfy user requirements.

(b) Select the best service based on the user's preferences.

How to address these questions using Semantic Web technology is the main focus of the present work. The three basic roles that we identify are the *service requester* (or buyer or consumer), the *service provider* (or seller), and the *broker*. The technical solution we provide is based on the following key ideas:

- The offerings or advertisements of service providers are represented in a Semantic Web language (RDF).
- These advertisements are semantically enriched by the use of a domain ontology (in RDF Schema).
- The advertisements are communicated to the broker by their providers.
- The requirements and preferences of a potential customer are represented in a logical language, based on rules and priorities.
- The logical representation of preferences and requirements are communicated to the broker by the requester.
- The broker matches the preferences against the set of available advertisements. The outcomes are communicated back to the requester.
- The broker maintains a repository to permanently store the advertisements.
- All the above operations take place in a multi-agent environment based on the peer to peer paradigm.

In the following we will elaborate on the problem, the technical solution, and an implemented system displaying the described functionality. The remainder of the paper is organized as follows: Section 2 reviews the most important Semantic Web technologies used in our solution. Section 3 describes our solution to the brokering problem, including a rationale for the chosen technologies. Section 4 illustrates the approach using a concrete example. Section 5 describes the technical details of the system that implements the solution. Section 6 reviews related work, and section 7 concludes the paper and poses future research directions.

2. An Overview of the Used Technologies

2.1 The Semantic Web Initiative

The aim of the Semantic Web initiative is to advance the state of the current Web through the use of semantics. More specifically, it proposes to use *semantic annotations* to describe the meaning of certain parts of Web information. For example, the Web site of a hotel could be suitably annotated to distinguish between hotel name, location, category, number of rooms, available services etc. Such meta-data could facilitate the automated processing of the information on the Web site, thus making it accessible to machines and not primarily to human users, as it is the case today.

However, the question arises as to how the semantic annotations of different Web sites can be combined, if everyone uses terminologies of their own. The solution lies in the organization of vocabularies in so-called *ontologies*. References to such shared vocabularies allow interoperability between different Web resources and applications. For example, an ontology of hotel classifications in a given country could be used to relate the rating of certain hotels. And a geographic ontology could be used to determine that Crete is a Greek island and Heraklion a city on Crete. Such information would be crucial to establish a connection between a requester looking for accommodation on a Greek island, and a hotel advertisement specifying Heraklion as the hotel location.

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. The layers that have reached sufficient maturity at present are:

- At the bottom layer we find *XML* [11], a language that lets one write structured Web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability.
- *RDF* [5] is a basic data model for writing simple statements about Web objects (resources). The RDF data model does not rely on XML, but RDF has an XML-based syntax. Therefore it is located on top of the XML layer.
- *RDF Schema* [11] provides modeling primitives for organizing Web objects into hierarchies. RDF Schema is based on RDF. RDF Schema can be viewed as a primitive language for writing ontologies.
- But there is a need for more powerful *ontology languages* that expand RDF Schema and allow the representation of more complex relationships between Web objects. Ontology languages, such as OWL, are built on the top of RDF and RDF Schema.

For an easy yet comprehensive introduction to the Semantic Web see [4]. In this paper we will make use of RDF to express semantic annotations of offerings, and RDF Schema for expressing ontologies.

2.2 Rules and Priorities on the Semantic Web

At present, the highest layer that has reached sufficient maturity is the ontology layer in the form of the description logic based language OWL [21]. The next step in the development of the Semantic Web will be the logic and proof layers, and *rule systems* appear to lie in the mainstream of such activities. Moreover, rule systems can

also be utilized in ontology languages. So, in general rule systems can play a twofold role in the Semantic Web initiative: (a) they can serve as extensions of, or alternatives to, description logic based ontology languages; and (b) they can be used to develop declarative systems on top of (using) ontologies. Reasons why rule systems are expected to play a key role in the further development of the Semantic Web include the following:

- Seen as subsets of predicate logic, monotonic rule systems (Horn logic) and description logics are orthogonal; thus rules provide additional expressive power to ontology languages.
- Efficient reasoning support exists to support rule languages.
- Rules are well known in practice, and are reasonably well integrated in mainstream information technology, such as knowledge bases, etc.

Possible interactions between description logics and monotonic rule systems were studied in [20]. However, these works don't exploit features such as negation, rules with exceptions and conflicting rules. Based on that work and on previous work on hybrid reasoning [25] it appears that the best one can do at present is to take the intersection of the expressive power of Horn logic and description logics; one way to view this intersection is the Horn-definable subset of OWL.

In our work we follow a different approach, by adding rules "on top" of web ontologies. However, as it is argued in [7], putting rules and description logics together poses many problems, and may be overkill, both computationally and linguistically. Another possibility is to start with RDF/RDFS, and extend it by adding rules; this approach is adopted in the present work. Furthermore, we make use of a feature called *conflicts among rules*. Generally speaking, the main sources of such conflicts are:

- Default inheritance within ontologies.
- Ontology merging.
- Rules with exceptions as a natural representation of business rules.
- Reasoning with incomplete information.

Basics of Defeasible Logics

Defeasible reasoning is a simple rule-based approach to reasoning with incomplete and inconsistent information. It can represent facts, rules, and priorities among rules. This reasoning family comprises defeasible logics [2] and Courteous Logic Programs [18]. The main advantage of this approach is the combination of two desirable features: enhanced representational capabilities allowing one to reason with incomplete and contradictory information, coupled with low computational complexity compared to mainstream nonmonotonic reasoning. The main features of this approach are:

- Defeasible logics are rule-based, without disjunction.
- Classical negation is used in the heads and bodies of rules, but negation-as-failure is not used in the object language (it can easily be simulated, if necessary [3]).
- Rules may support conflicting conclusions.
- The logics are skeptical in the sense that conflicting rules do not fire. Thus consistency is preserved.

- Priorities on rules may be used to resolve some conflicts among rules.
- The logics take a pragmatic view and have low computational complexity.

There are two kinds of rules (fuller versions of defeasible logics include also defeaters): *Strict rules* are denoted by $A \rightarrow p$, and are interpreted in the classical sense: whenever the premises are indisputable then so is the conclusion. An example of a strict rule is "*Professors are faculty members*". Written formally: $professor(X) \rightarrow faculty(X)$. Inference from strict rules only is called *definite inference*. Strict rules are intended to define relationships that are definitional in nature, for example ontological knowledge.

Defeasible rules are denoted by $A \Rightarrow p$, and can be defeated by contrary evidence. An example of such a rule is $faculty(X) \Rightarrow tenured(X)$ which reads as follows: "*Professors are typically tenured*".

A *superiority relation* on R is an acyclic relation > on R (that is, the transitive closure of > is irreflexive). When $r_1 > r_2$, then r_1 is called *superior* to r_2, and r_2 *inferior* to r_1. This expresses that r_1 may override r_2. For example, given the defeasible rules

$$r: professor(X) => tenured(X)$$
$$r': visiting(X) => \neg tenured(X)$$

which contradict one another, no conclusive decision can be made about whether a visiting professor is tenured. But if we introduce a superiority relation > with r' > r, then we can indeed conclude that he/she cannot be tenured.

A formal definition of the proof theory is found in [2]. A model theoretic semantics is found in [29], and argumentation semantics is discussed in [15].

3. Proposed Solution

3.1 Agent Discovery and Service Providing Architectures

Agent discovery is a way of advertising, managing and finding information about agents' services and capabilities. We can distinguish two different categories of agent discovery mechanisms, centralized and distributed.

When it comes to centralized solutions for agent discovery, or "middle agents" according to [38], three different kinds of agents prevail. They are called *matchmakers* or Yellow Pages Services, *facilitators* and *brokers* respectively. We borrow the next two figures (Figure 1, Figure 2) from their work.

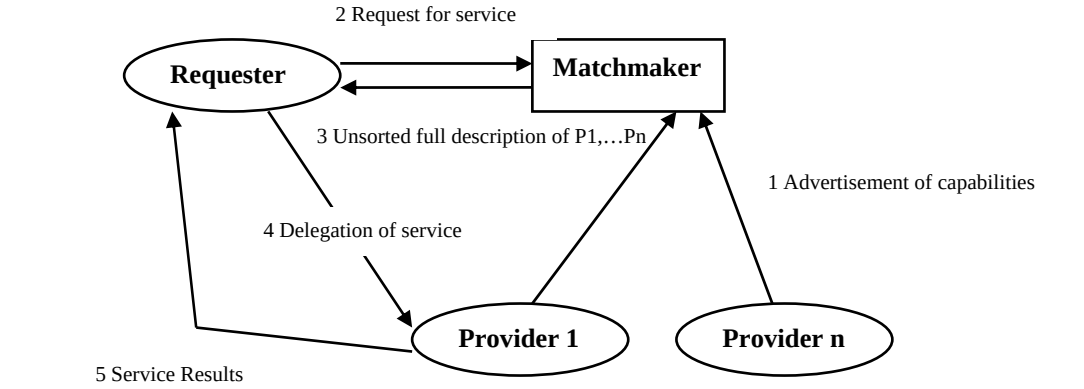

Figure 1. Matchmaker Architecture

When it comes to matchmakers, different service providers advertise their capabilities (1) and the matchmaker puts them into a repository. When the matchmaker is asked for a particular service by a service requester (2), it returns information about all the available service providers (3). It now depends on the requester which provider it will choose (4) for the required service. Lastly, the provider serves the request and returns the results (5). It is assumed that the "address" of a matchmaker is well-known.

Facilitators operate in a slightly different way. Initially, providers advertise their capabilities (1). After requesters have located a facilitator they pass on their preferences along with the delegation of a service (2). The facilitator, in turn, picks one of the providers to delegate the requested service (3). The provider then returns the result (4) and the facilitator returns it to the requester (5).

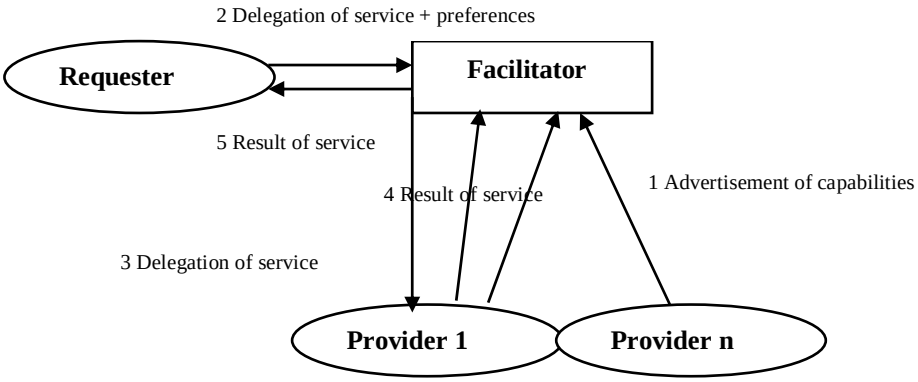

Figure 2. Facilitator-Broker Architecture

A variation of this architecture could be that the middle agent itself performs the serving of a request using services and information from other agents in conjunction with his own services. In the latter case the middle agents is called "broker". We use this variation for our implementation. However, we would like to stress that our technology can easily be adapted to realize any of the above architectures; we have chosen to implement the broker architecture to demonstrate the feasibility of the overall approach.

3.2 General Approach

The three basic roles that we identify in our brokering system are the *Service Requester* (or the Buyer), the *Service Provider* (or Seller), and the *Broker*. Another agent, called Directory Facilitator (D.F.) plays a secondary role and is the yellow pages service (or Matchmaker), which agents use to find each other and register what protocols they use, what ontologies they use, etc. The technical solution we provide is based on the following key ideas:

- Service requesters, service providers and brokers are represented by software agents that run on the JADE multi-agent platform.
- The requirements of the service requester are represented in defeasible logic, using rules and priorities. These requirements include both indispensable requirements that must be met for a service to be acceptable (for example, air-

conditioning is required), and soft requirements (preferences) that can be used to select among the potentially acceptable offerings. These requirements are communicated to the broker agent by the requester agent. This communication initiates a brokering activity.

- The offerings or advertisements are represented in a certain semi-structured format using the Semantic Web standard language RDF for describing Web resources. The provider agents communicate the offerings to the broker agent.
- The terminology shared by providers, requesters and brokers is organized in an ontology using the Semantic Web standard of RDF Schema.
- The broker is also a software agent and has special knowledge both for the declarative language and the advertisement format. It also has the ability to perform semantic checks to the information it receives.
- When the broker receives a request it matches the request to the advertisements by running the request specification against the available offerings, making use of information provided by the shared ontology, as required. Then the requester's preferences are applied to select the most suitable offering(s) which are then presented to the requester.
- For the persistent storage of advertisements, an RDF repository, and particularly ICS-FORTH RDF Suite [1], is used.

3.3 Description of Offerings

The offerings are described in RDF, the standard Semantic Web language for representing factual statements. This choice (a) supports interoperability among agents and applications, and (b) facilitates the easy publication, collection and combination in decentralized dynamic settings. The offerings are enriched through reference to shared ontologies, e.g. of the tourism domain or geographical terms. The benefits of ontologies for matching requester requirements to offerings were stated previously. We assume that this ontology is expressed in RDF Schema, a simple ontology language based on RDF. We have chosen this language over the use of OWL because at present it is not clear how the deductive capabilities of OWL and rule systems can be combined; it is one of the main research lines in the Semantic Web community. We could certainly use most features of OWL Lite, given that they can be expressed using rules [20].

3.4 Description of Requests and Preferences

The requirements and preferences of the requester are described in a logical language. Before choosing one or several languages for the specification of requests it is important to establish a set of criteria that such languages need to satisfy. The criteria presented below are inspired from those formulated by [23] in the context of techniques for information modeling. They encompass several well-known principles of language design.

Firstly, a language for specifying requirements and preferences needs to be *formal*, in the sense that its syntax and its semantics should be precisely defined. This ensures that the requirements and preferences can be interpreted unambiguously (both by machines and humans) and that they are both *predictable* and *explainable*.

Secondly, the language should be *conceptual*. This, following the well-known *Conceptualization Principle* of [17], effectively means that it should allow its users to focus only and exclusively on aspects related to requirements, without having to deal with any aspects related to their realization or implementation. Examples of conceptually irrelevant aspects in the context that we consider are: physical data organization and access, platform heterogeneity (e.g. message-passing formats), and book-keeping (e.g. message queue management).

Thirdly, in order to ease the interpretation of strategies and to facilitate their documentation, the language should be *comprehensible*. Comprehensibility can be achieved by offering a graphical representation, by ensuring that the formal and intuitive meanings are as much in line as possible, and by offering structuring mechanisms (e.g. decomposition). These structuring mechanisms often lead to *modularity*, which in our setting means that a slight modification to a strategy should concern only a specific part of its specification. Closely related to its comprehensibility, the language that we aim should be suitable, that is, it should offer concepts close to those involved in requirements and preferences.

As we are interested in the automation of the brokering process, the requirements description language should be *executable*, and its execution should exhibit acceptable performances even for complex requirements (i.e. the execution performances should be *scalable*). Finally, the language that we aim should be sufficiently *expressive*, that is, it should be able to precisely capture a wide spectrum of requirements.

We have chosen defeasible logic to represent requesters' requirements and preferences because it satisfies the above criteria. In particular,

- It is a formal language with well-understood meaning ([2] presents a proof theory, [29] its model semantics, and [15] its argumentation semantics), thus it is also predictable and explainable.
- It is designed to be executable; implementations are described in [30]. It is also scalable, as demonstrated in the same paper, where it was shown that 100,000 rules can be processed efficiently. This is so because the computational complexity of defeasible logic is low [28].
- It is expressive, as demonstrated by the use of rules in various areas of information technology. In fact, among the logical systems, it is rule-based systems that have been best integrated in mainstream IT.
- Finally, it is suitable for expressing requirements and preferences in our setting. This is so because it supports the natural representation of important features:
    - Rules with exceptions are a useful feature in our problem. For example, a general rule may specify acceptable offerings, while more specific rules may describe cases in which the general rule should not apply and the offering should not be accepted. We will elaborate on this point in the next section when we consider a concrete example.
    - Priorities are an integral part of defeasible logic, and are useful for expressing user preferences for selecting the most appropriate offerings from the set of the acceptable offerings.

4. A Brokered Trade Example

In this section we present a full example of using defeasible logic in a brokered trade application that takes place via an independent third party, the broker. The broker matches the buyer's requirements and the sellers' capabilities, and proposes a transaction when both parties can be satisfied by the trade. In our case, the concrete application (which has been adopted from [4]) is apartment renting and the landlord takes the role of the abstract seller.

Available apartments reside in an RDF document (Figure 4). The requirements of a potential renter, called e.g. Carlo, are shown in Figure 3. These requirements are expressed in defeasible logic as explained below, in a logic-like syntax. More specifically, the following predicates are used to describe properties of apartments:

- `size(x,y)`, where y is the size of apartment x (in m^2)
- `bedrooms(x,y)`, where apartment x has y bedrooms
- `price(x,y)`, where y is the price for x
- `floor(x,y)`, where apartment x is on the y-th floor
- `gardenSize(x,y)`, where apartment x has a garden of size y
- `lift(x)`, meaning that there is an elevator in the house of x
- `pets(x)`, meaning that pets are allowed in x
- `central(x)`, meaning that x is centrally located

1. *Carlos is looking for an apartment of at least 45m^2 with at least 2 bedrooms. If it is on the 3rd floor or higher, the house must have an elevator. Also, pet animals must be allowed.*
2. *Carlos is willing to pay $300 for a centrally located 45m^2 apartment, and $250 for a similar flat in the suburbs. In addition, he is willing to pay an extra $5 per m^2 for a larger apartment, and $2 per m^2 for a garden.*
3. *He is unable to pay more than $400 in total. If given the choice, he would go for the cheapest option. His 2nd priority is the presence of a garden; lowest priority is additional space.*

Figure 3. Verbal description of Carlo's (a potential renter) requirements.

```
<!DOCTYPE rdf:RDF [...
  <!ENTITY carlo "http://.../carlo.rdf#"> ]>
<rdf:RDF ... xmlns:carlo="&carlo;">
  <carlo:apartment rdf:about="&carlo;a1">
    <carlo:bedrooms rdf:datatype="&xsd;integer">1</carlo:bedrooms>
    <carlo:central>yes</carlo:central>
    <carlo:floor rdf:datatype="&xsd;integer">1</carlo:floor>
    <carlo:gardenSize rdf:datatype="&xsd;integer">0</carlo:gardenSize>
    <carlo:lift>no</carlo:lift>
    <carlo:name>a1</carlo:name>
    <carlo:pets>yes</carlo:pets>
    <carlo:price rdf:datatype="&xsd;integer">300</carlo:price>
    <carlo:size rdf:datatype="&xsd;integer">50</carlo:size>
  </carlo:apartment>
  ...
</rdf:RDF>
```

Figure 4. RDF document for available apartments

Also the following predicates are used:

- `acceptable(x)`, meaning that flat x satisfies Carlos's requirements
- `offer(x,y)`, meaning that Carlos is willing to pay $ y for flat x

Any apartment is a priori acceptable.

```
r1: => acceptable(X)
```

However, Y is unacceptable if one of Carlos's requirements is not met (exceptions to rule `r1`).

```
r2: bedrooms(X,Y), Y < 2 => ¬acceptable(X)
r3: size(X,Y), Y < 45 => ¬acceptable(X)
r4: ¬pets(X) => ¬acceptable(X)
r5: floor(X,Y), Y > 2, ¬lift(X) => ¬acceptable(X)
r6: price(X,Y), Y > 400 => ¬acceptable(X)
r2 > r1, r3 > r1, r4 > r1, r5 > r1, r6 > r1
```

The price Carlos is willing to pay for an apartment is calculated as follows:

```
r7: size(X,Y), Y ≥ 45, garden(X,Z), central(X) => offer(X, 300 + 2Z +
5(Y-45))
r8: size(X,Y), Y ≥ 45, garden(X,Z),¬central(X) => offer(X, 250 + 2Z +
5(Y-45))
```

An apartment is only acceptable if the amount Carlos is willing to pay is not less than the price specified by the landlord.

```
r9: offer(X,Y), price(X,Z), Y < Z => ¬acceptable(X)
r9 > r1
```

In addition to identifying the apartments acceptable to Carlos it is also possible to reduce the number further, even down to a single apartment, by taking further preferences into account. Carlos's preferences are based on price, garden size, and size, in that order, represented as follows:

```
r10: cheapest(X) => rent(X)
r11: cheapest(X), largestGarden(X) => rent(X)
r12: cheapest(X), largestGarden(X), largest(X) => rent(X)
r11 > r10, r12 > r10, r12 > r11
```

Since at most one apartment can be rented, literals `rent(X)` are conflicting. This is represented using conflict sets: `C(rent(x)) = {¬rent(x)} ∪ {rent(y) | y ≠ x}`

The prerequisites of these rules can be derived from the set of acceptable apartments using further rules. For example, `cheapest(X)` can be calculated by the following rule that makes use of negation as failure (operator `not`):

```
rc: acceptable(X), price(X,Z),
    not(acceptable(Y), Y ≠ X, price(Y,W), W < Z)
            => cheapest(X)
```

Similar rules exist for `largestGarden(X)` and `largest(X)`, as well.

5. Brokering System Implementation

5.1 Multi-Agent Framework

The agent framework we used for the development of our system is JADE [9], [24]. JADE is an open-source middleware for the development of distributed multi-agent applications. It is Java-based and compliant with the FIPA specifications [14]. It provides libraries for agent discovery, communication and interaction, based on FIPA standards

From the functional point of view, JADE provides the basic services necessary to distributed peer-to-peer [34] applications in the fixed and mobile environment. JADE allows each agent to dynamically discover other agents and to communicate with them according to the peer-to-peer paradigm. From the application point of view, each agent is identified by a unique name and provides a set of services. It can register and

modify its services and/or search for agents providing given services, it can control its life cycle and, in particular, communicate with all other peers.

5.2 System Architecture and Modules

The architecture of the broker consists of five main modules: (a) reasoning module, (b) control module, (c) semantic and syntactic validator, (d) RDF Suite module, and (e) rule-query-RDF loader module. Reasoning and control modules consist of other sub-modules as one can see in Figure 5 which depicts the overall system architecture. The other three modules are stand-alone. Finally, the control module is responsible for the coordination of all the other modules.

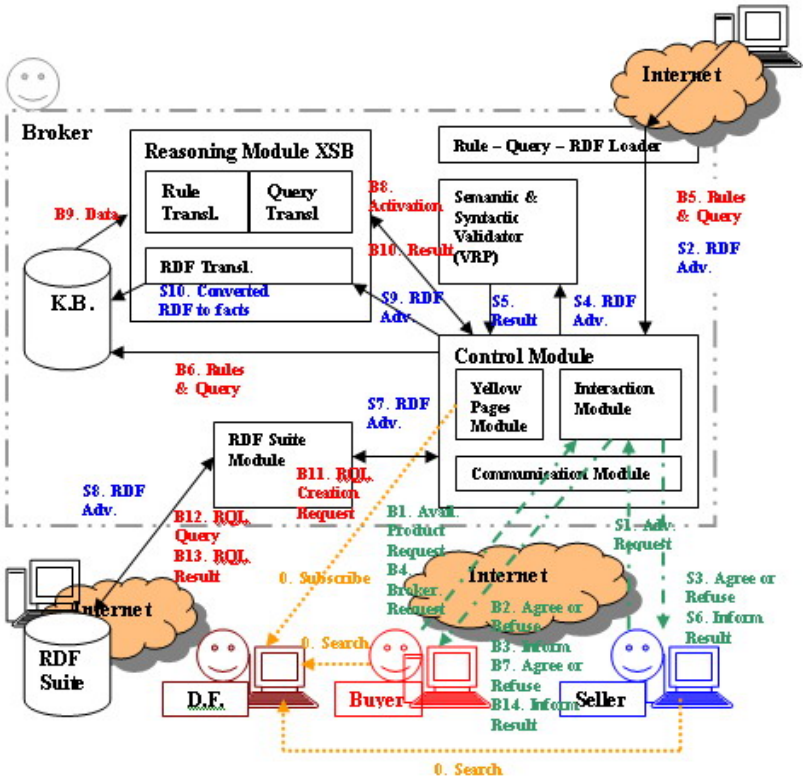

Figure 5. The Brokering System Architecture

RDF Translator

The role of the RDF translator is to transform the RDF statements into logical facts, and the RDFS statements into logical facts and rules. This transformation allows the RDF/S information to be processed by the rules provided by the Service Requester (representing the requester's requirements and preferences). For RDF data, the SWI-Prolog RDF parser is used to transform them into an intermediate format, representing triples as *rdf(Subject, Predicate, Object)*. Some additional processing (i) transforms the facts further into the format *Predicate(Subject, Object)*; (ii) cuts the namespaces and the "comment" elements of the RDF files, except for resources that refer to the RDF Schema, for which namespace information is retained.

In addition, for processing RDF Schema information, the following rules capturing the semantics of RDF Schema constructs are created:

```
A: C(X):- rdf:type(X,C).
```

```
B: C(X):- rdfs:subClassOf(Sc,C), Sc(X).
C: P(X,Y):- rdfs:subPropertyOf(Sp,P), Sp(X,Y).
D: D(X):- rdfs:domain(P,D), P(X,Z).
E: R(Z):- rdfs:range(P,R), P(X,Z).
```

Let us consider rule `B` that captures the meaning of the subclass relation of RDFS. A class `Sc` is subclass of a class `C` when all instances of `Sc` are also instances of `C`. Stated another way, if `X` is an instance of `Sc` then it is also instance of `C`. That is exactly what rule `B` says. All the above rules are created at compile-time, i.e. before the actual querying takes place. Therefore, although the above rules at first sight seem second-order because they contain variables in place of predicate names, they are actually first-order rules, i.e. predicate names are constant at run-time.

Semantic-Syntactic Validator

This module is an embedded version of [37], a parser for validating RDF descriptions. Upon receipt of an advertisement, the RDF description, which corresponds to that advertisement, is checked by this module. Among others, the tests performed are: class hierarchy loops, property hierarchy loops, domain/range of subproperties, source/target resources of properties and types of resources. This module is "called" by the control module and returns either the RDF description, in case the latter is error free, or an error message. For the implementation of this module we used the API of VRP.

Interaction and Communication Modules

The communication module is responsible for sensing the network and notifying the control module when an external event (e.g. a request message) occurs. In order to decide the course of action based on the incoming message's type, the broker agent extracts the message from the queue and examines its type, i.e. whether it is a "Broker Request", "Advertise Request" message etc. Accordingly it activates the interaction module. The interaction module consists of different interaction protocols that extend the standard FIPA Request interaction protocol. For the implementation of these modules, we used the API of JADE framework.

RDF Suite Module

The RDF Suite module is responsible for all the actions related with the handling of the advertisements and the domain ontology. The most important functions of this module are:

- Initial upload of RDFS ontology and RDF instances into the RDF repository.
- Update of the RDF repository with RDF descriptions that are received from the service providers and correspond to product or service advertisements.
- Preparation of RQL queries and forwarding to the RDF Suite.
- Receipt of RQL queries' results.

The RDF Suite module implements a client socket, which connects to a server socket and passes an RQL query or retrieves the results. At this point we must say, that although RSSDB and VRP work well in MS Windows and their Java API can be easily used, there is a problem with RQL that operates only in UNIX. The server socket, which creates a UNIX pipe to RSSDB, solves this problem. For the implementation of this module we used the API of RSSDB and the API of Java for File Management and Networking.

Rule-Query-RDF Loader

The role of this module is to download the files, which contain the rules and the query of the user, in defeasible logic format. It also downloads the appropriate RDF descriptions, which correspond to service providers' advertisements. It also implements methods for file handling. For the implementation of this module we used the API of Java for File Management and the API for Networking.

Reasoning Module

The role of the Reasoning Module is to apply the queries to files, which contain the facts and the rules, and to evaluate the answer. When the Service Requester makes a query, the Reasoning Module compiles the files containing the facts and the rules, and applies the query to the compiled files. The answer of the query is sent to the Control Module of the system. The reasoning engine that we employed to implement this module is DR-Prolog [6]. This is a defeasible reasoning system that is based on the translation of defeasible theories into Prolog clauses, and is built on top of XSB Prolog.

Rule Parser & Translator

The Rule Parser is responsible for checking the validity of the defeasible rules, which are submitted by the Service Requester. The rules are considered to be valid, if they follow the standard syntax of defeasible logic, as described in [2]. If there are syntax errors, the system informs the user about these errors, and does not proceed to the translation. Otherwise, the parser creates a symbol table, which includes all the rules and priority information, and passes this table to the Translator.

The Rule Translator is responsible for transforming the rules submitted by the Service Requester using the syntax of defeasible logic, into Prolog rules that emulate the semantics of defeasible logic. The method we use for translating defeasible theories into logical programs is described in detail in [5].

The logical program that derives from this procedure will be later combined with the logical facts that represent the RDF triples, and will be used to evaluate the queries of the Service Requester.

Query Translator

In order to apply a query to the Prolog files, which contain the rules and the facts, it must be priorly transformed into a valid Prolog query. This task is performed by the Query Translator. There is a standard format for the queries that the Service Requester can make:

D x : which are the literals (atoms or their negation) x which are provable according to the rules provided by the Service Requester.

The literals 'x' represent the conclusions of the rules, which are submitted by the Service requester. 'x' can be for example of the form '*accept_hotel(X)*'. In this case a query of the form '*D accept_hotel(X)*', is intended to find those literals X satisfying the conclusion '*accept_hotel(X)*'.

5.3 System Interactions

We describe the sequence of actions, separately for the buyer (B label) and the seller (S label). Initially Buyer, Seller and Broker agents, subscribe to Directory Facilitator or D.F. agent. These actions are depicted by the dashed lines (step 0). They provide information such as the ontologies they are committed to, the interaction protocols they use, the content language they use etc.

A *seller* initializes an interaction by issuing an “Advertise” request (step S1). The broker extracts the field “RDFInfoAtWeb” from the received message and tries to download the corresponding advertisement from the web, which is an RDF description with information about the advertised product or service (step S2). If the document exists, broker informs the seller that he agrees to perform the requested action (step S3). Subsequently the broker checks the RDF advertisement semantically and syntactically, using the Semantic and Syntactic validator module (step S4). The result is returned to the control module of broker (step S5), and if the advertisement is valid, according to the domain ontology, seller is informed that the requested action was performed. Otherwise an error message is posted (step S6). Broker then performs a twofold action. He firstly feeds the RDF Suite module with the advertisement (step S7), which in turn stores it to the RDF Suite repository (step S8) and secondly sends the advertisement to the RDF translator module of the Inference engine to add it in the knowledge base (step S9). Finally the knowledge base is updated with the new facts which were previously extracted from the RDF advertisement (step S10).

A *buyer* sends an “Available Products” request to the broker (step B1). Broker informs buyer if he agrees or not to perform the action (step B2) and in turn he sends to buyer a message with the available categories of products (step B3). Buyer then issues a ”Brokering” request, for a particular category of products (step B4). Broker downloads from the web the Rules which capture the preferences of the buyer and are expressed in defeasible logic. He also downloads the submitted query which is also expressed in defeasible logic (step B5). Subsequently the rules and the query are stored in the knowledge base by the broker (step B6). Broker informs the buyer if he agrees or not to perform the requested action, according to the validity of the rules and the query (step B7). Broker in turn activates the reasoning module (step B8), which uses the stored data in the knowledge base (step B9) and performs the reasoning process. The result is the set of the ID’s of the matched RDF descriptions (step B10). Afterwards, the RDF Suite module of the broker creates dynamically an RQL query (step B11) which is passed to the RDF Suite repository for the retrieval of all the resources which correspond to the IDs of the result set (step B12). The result of the query is returned to the broker (step B13). Finally, the broker encapsulates the received information to an ACL message and sends it back to the buyer (step B14).

6. System Evaluation

In section 3 we described the main arguments in favour of our approach, and its distinctive features. In this section we concentrate on the performance evaluation of the reasoning module. In particular, we present the most significant results from the experimental evaluation of DR-Prolog, the reasoning engine at the heart of our system, which we conducted and originally presented in [6]. The experimental tests are defeasible theories, consisting of a varying number of facts, rules and superiority relations.

In DR-BROKERING, we assume that the facts derive from the translation of RDF documents that contain the available data, and the rules and superiority relations are contained in the defeasible theories imported by the user. Here, we focus only on those theories that contain a large number of facts, and a set of conflicting rules. These are the tree theories, *tree*(*n*,*k*), in which k^n facts and $\sum_{i=0}^{n-1} k^i$ conflicting rules form a *k*-branching tree of depth *n* in which every literal occurs *k* times (see [6] for more details).

In Table 1, we present the time (in CPU seconds) that DR-Prolog requires to conduct a proof for one of the literals supported by the rules of the theory. The overall "size" of the theory is defined as the sum of the number of facts, rules and literals in the bodies of all rules. The experiments are designed to execute all rules and literals of each test theory.

tree(n,k)	Size	Time
n = 6, k = 3	2185	0.22
n = 7, k = 3	6559	1.16
n = 8, k = 3	19681	9.61

Table 1. Execution times for tree theories

More details about the experimental evaluation of the performance of DR-Prolog, as well as a comparison with the performance of similar defeasible reasoning implementations can be found in [6].

7. Related Work

InfoSleuth [31] is an agent-based information discovery and retrieval system that adopts "broker agents" to perform the syntactic and semantic matchmaking. The broker uses a rule-based reasoning engine, implemented in LDL, to determine which agents have advertised services that match those requested in the query. The brokering is realized in two levels. Syntactic brokering is the process of matching request, on the basis of the syntax of incoming messages and used ACLs or Content Languages. Semantic brokering is the process of matching requests on the basis of the requested capabilities or offered services. An agent's knowledge is expressed independently of syntax, based on shared common service ontology.

Trastour et al. [36] enumerate the requirements for a language to express service descriptions in the context of a matchmaking service. They propose the use of RDF/RDFS for the matchmaking process. Each advertisement, either for service request, or for service offering is represented as an RDF resource. Properties from this resource characterize the type of requested or offered service. The advertisements are stored into a repository and the matching of advertisements is reduced to matching of RDF graphs. The authors implemented a matching algorithm.

Li and Horrocks [26] assess the requirements for a service description language and ontology, and argue that DAML+OIL and DAML-S common service ontology, fulfil these requirements. This argument is supported by their design and implementation of a prototype matchmaker which uses a description logic reasoner to match service advertisements and requests based on the semantics of ontology-based service descriptions. Similar is the work of [33]. They also use DAML-S to describe the advertisements along with the request and afterwards they use a matching algorithm.

Chen et al. [13] propose an architecture for an agent that, although not explicitly stated, could be used for semantic brokering. The iAgent they propose consists of inference, control and communication layer. As they say, a typical inference engine makes inference according to static facts and rules. As an alternative they propose that the facts are extracted from semantic mark-up documents that are written in

DAML+OIL. The fact translator module of the iAgent, converts all the DAML+OIL documents into prolog format. As a result, although DAML+OIL is description logic, which is not suitable for complex queries, iAgent finally uses a Horn-based logic engine (SWI-Prolog) for inferencing.

8. Conclusions and Future Work

In this paper we studied the brokering and matchmaking problem, that is, how a requester's requirements and preferences can be matched against a set of offerings collected by a broker. The proposed solution uses the Semantic Web standard of RDF to represent the offerings, and a deductive logical language for expressing the requirements and preferences. We motivated and explained the approach we propose, and reported on a prototypical implementation exhibiting the described functionality in a mulit-agent environment.

Our approach has obvious advantages compared to other approaches. Particularly, (a) we do not provide a fixed algorithm for brokering but it is the user who specifies the algorithm on the basis of its preferences. (b) The architecture we provide is higly reusable. The system can be applied in any domain only with the addition of a new ontology and new rules which capture the preferences. (c) Using JADE, we exploit the advantages of peer-to-peer systems (i.e. travel agencies and broker as peers) and also make use of FIPA standards for agent communication and discovery. (d) We use a higly expessive language for preferences specification with interesting features, such as conflicting rules and priorities of rules. (e) We use RDF for the expression of advertisements. This choice supports interoperability among agents and applications and facilitates the easy publication, collection and combination in decentralized dynamic settings. (f) We allow for permanent storing of advertisements with the use of the RDF Suite repository.

The main limitations of the current implementation are: (a) The advertisements cannot be removed automatically when they expire. (b) The syntax of the defeasible logic may appear too complex for many users, and should be supported by, possibly graphical, authoring tools.

The architecture we proposed is based on the assumption that all service advertisements have the same format, i.e. that there is a shared ontology. This is not a very unrealistic assumption, since many business communities have already conceded into common ontologies. However, even if different travel agents use different ontologies, ontology translation techniques could be used to map different ontologies onto the common ontology supported by the brokering service. The ontology translation service could be offered by the brokering service for several popular ontologies. However, if a service provider uses a minor and/or personal ontology, then it should be his/her responsibility to provide translation to the common ontology, should he/she wants to take advantage of the brokering services offered.

In the future we intend to extend the described work in various directions: (i) Add advertisement removal functionality, which will be activated, when the advertisement has expired. (ii) Implement graphical user interfaces for the integrated system. Someone will be able to load the files, which correspond to the rules of negotiation strategy and brokering preferences respectively, using a file manager. He will be also able to adjust negotiation protocol parameters and monitor the progress of the brokering and negotiation procedure. (iii) Integrate the current brokering system with the negotiation system proposed in [35]. In our current implementation, a service

requester agent is able to find potential products or services and potential service providers. We intend to extend our system to support direct involvement of the service requester in negotiation with the service provider for the resulted product, as soon as the brokering stage has been completed.

Finally, as a long-term goal we intend to extend the semantic brokering approach presented in this paper to brokering for general purpose semantic web services, providing matchmaking between Web Service advertisements and requests described in OWL-S. Of course, first of all we must carefully define which of the features of OWL are captured by defeasible logic and can be used, and which cannot.

References

[1] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis and K. Tolle (2001). The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In *Proc. 2nd International Workshop on the Semantic Web*, Hongkong, May 1, 2001.

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

[3] G. Antoniou, M. J. Maher and D. Billington (2000). Defeasible Logic versus Logic Programming without Negation as Failure. *Journal of Logic Programming* 41,1 (2000): 45 – 57.

[4] G. Antoniou and F. van Harmelen (2004). *A Semantic Web Primer*. MIT Press 2004.

[5] G. Antoniou, A. Bikakis (2004). "A System for Non-Monotonic Rules on the Web». Workshop on Rules and Rule Markup Languages for the Semantic Web (RuleML 2004), G. Antoniou, H. Boley (Ed.), Springer-Verlag, Hiroshima, Japan, 8 Nov. 2004.

[6] G. Antoniou and A. Bikakis (2006). DR-Prolog: A System for Defeasible Reasoning with Rules and Ontologies on the Semantic Web, *IEEE Transactions on Knowledge Data and Engineering* (accepted).

[7] G. Antoniou, G. Wagner, "Rules and Defeasible Reasoning on the Semantic Web", in *Proc. RuleML Workshop 2003*, Springer-Verlag, LNCS 2876, pp. 111–120, 2003.

[8] D. Beckett (2004). RDF/XML Syntax Specification, W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[9] F.Bellifemine, G Caire, A.Poggi, G. Rimassa (2003). JADE A White Paper. Telecom Italia EXP magazine Vol 3, No 3 September 2003.

[10] T. Berners-Lee (1999). *Weaving the Web*. Harper 1999.

[11] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler (2000). Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation, October 2000. Available at: http://www.w3.org/TR/2000/REC-xml-20001006.

[12] D. Brickley, R.V. Guha (2004). RDF Vocabulary Description Language 1.0: RDF Schema W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/2004/REC-rdf-schema-20040210/.

[13] Y-C. Chen, W-T. Hsu, P-H. Hung (2003). Towards Web Automation by Integrating Semantic Web and Web Services. In *Proc. 12th International WWW Conference.*

[14] FIPA. http://www.fipa.org.

[15] G. Governatori and M.J. Maher. An argumentation-theoretic characterization of defeasible logic. In Proceedings of the 14th European Conference on Artificial Intelligence}, Amsterdam, 2000. IOS Press.

[16] G. Governatori, M. Dumas, A. ter Hofstede and P. Oaks (2001). A formal approach to legal negotiation. In *Proc. ICAIL 2001*, 168-177.

[17] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and the Information Base*. Publ. nr. ISO/TC97/SC5/WG3-N695, ANSI, 11 West 42nd Street, New York, NY 10036, 1982.

[18] B. N. Grosof (1997). Prioritized conflict handing for logic programs. In *Proc. of the 1997 International Symposium on Logic Programming*, 197-211.

[19] B. N. Grosof, M. D. Gandhe and T. W. Finin: SweetJess: Translating DAMLRuleML to JESS. RuleML 2002. In: *Proc. International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*.

[20] B. N. Grosof, I. Horrocks, R. Volz and S. Decker (2003). Description Logic Programs: Combining Logic Programs with Description Logic". In: *Proc. 12th Intl. Conf. on the World Wide Web (WWW-2003)*, ACM Press.

[21] D.L. McGuinness , F. van Harmelen (2004). OWL Web Ontology Language Overview W3C Recommendation, February 2004. Available at: http://www.w3.org/TR/owl-features/.

[22] M. He, N.R. Jennings, and H-F. Leung (2003). On Agent-Mediated Electronic Commerce. *IEEE Transactions on Knowledge and Data Engineering* Vol. 15, No 4 July/August 2003.

[23] A.H.M. ter Hofstede. *Information Modelling in Data Intensive Domains*. PhD thesis, University of Nijmegen, Nijmegen, The Netherlands, 1993.

[24] JADE Project: http://jade.cselt.it/.

[25] A. Levy and M-C. Rousset (1998). Combining Horn rules and description logics in CARIN. *Artificial Intelligence* 104, 1-2 (1998):165 – 209.

[26] L. Li and I. Horrocks (2003). A Software Framework For Matchmaking Based on Semantic Web Technology. In *Proc. 12th International Conference on WWW*, ACM 2003.

[27] P. Maes, R.H. Guttman and A.G. Moukas (1999). Agents That Buy and Sell. *Communications of the ACM* Vol. 42, No. 3 March 1999.

[28] M.J. Maher (2001). Propositional Defeasible Logic has Linear Complexity. Theory and Practice of Logic Programming 1,6, p. 691-711.

[29] M.J. Maher (2002). A Model-Theoretic Semantics for Defeasible Logic", *Proc. Workshop on Paraconsistent Computational Logic*, 67 - 80, 2002.

[30] M.J. Maher, A. Rock, G. Antoniou, D. Billington and T. Miller (2001). Efficient Defeasible Reasoning Systems. *International Journal of Tools with Artificial Intelligence* 10,4 (2001): 483—501.

[31] M. Nodine, W. Bohrer, A. Hee Hiong Ngu (1998). Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth. In *Proc. 15th International Conference on Data Engineering*, IEEE Computer Society.

[32] D. Nute (1994). Defeasible logic. In *Handbook of logic in artificial intelligence and logic programming (vol. 3): nonmonotonic reasoning and uncertain reasoning*. Oxford University Press.

[33] M. Paolucci, T. Kawamura, T.R. Payne, K. Sycara (2002). Semantic Matching of Web Services Capabilities. In *Proc. 1st International Semantic Web Conference (ISWC-2002).*

[34] S. Saroiu, P. Krishna, Gummadi, S.D. Gribble (2002). A Measurement Study of Peer-to-Peer File Sharing Systems SPIE and ACM Multimedia - *Multimedia Computing and Networking (MMCN-2002).*

[35] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori (2005) "DR-NEGOTIATE-A System for Automated Agent Negotiation with Defeasible Logic-Based Strategies ". In proceedings of the IEEE international conference on e-Technology,e-Commerce and e-Service (EEE05). Hong Kong, China April 2005.

[36] D. Trastour, C. Bartolini, J. Gonzalez- Castillo (2001). A Semantic Web Approach to Service Description for Matchmaking of Services. HP Technical Report. August 2001.

[37] VRP. The ICS-FORTH Validating Rdf Parser - VRP (2004). Available at: http://139.91.183.30:9090/RDF/VRP/.

[38] H-C. Wong and K. Sycara (2000). A Taxonomy of Middle-agents for the Internet. In *Proc. 4th International Conference on Multi Agent Systems (ICMAS-2000).*