Discovery and Uncertainty in Semantic Web Services

Recuerda, F., & Robertson, D. (2005). Discovery and Uncertainty in Semantic Web Services. In Proceedings of the ISWC 2005 Workshop on Uncertainty Reasoning for the Semantic Web. CEUR Workshop Proceedings.

Discovery and Uncertainty in Semantic Web Services

Francisco Martín-Recuerda[1] and Dave Robertson[2]

[1] Digital Enterprise Research Institute (DERI), Leopold-Franzens Universität Innsbruck, Technikerstraße 21a, 6020 Innsbruck, Austria
francisco.martin-recuerda@deri.org
http://www.deri.at/
[2] University of Edinburgh, School of Informatics, Centre for Intelligence Systems and their applications, Appleton Tower, Crichton Street, Edinburgh, EH89LE, Scotland
dr@inf.ed.ac.uk
http://www.cisa.informatics.ed.ac.uk/

Abstract. Although Semantic Web service discovery has been extensively studied in the literature ([7], [12], [15] and [10]), we are far from achieving an effective, complete and automated discovery process. Using the incidence calculus [4], a truth-functional probabilistic calculus, and a lightweight brokering mechanism [17], the article explores the suitability of integrating probabilistic reasoning in Semantic Web services environments. We show how the combination of relaxation of the matching process and evaluation of web service capabilities based on a previous historical record of successful executions enables new possibilities in service discovery.

1 Introduction

Discovery composition, invocation and interoperation are the core pillars of the deployment of Semantic Web services [9]. Discovery has been extensively studied in the literature ([7], [21], [12] and [15]). In a recent effort, the authors of [10] have focused on providing a coherent and formal model for Semantic Web services discovery.

Roughly speaking, the relaxation of the matching process between a goal (a functional description of objectives that clients want to achieve using web services) and web services capabilities (functional descriptions of a service) has been based on the following set of matching notions [10]: (i) exact-match, a goal and matched web service capabilities are the same; (ii) plug-in-match, a goal is subsumed by matched web service capabilities; (iii) subsume-match, matched web service capabilities are subsumed by a goal; (iv) intersection-match, a goal and matched web service capabilities have some elements in common; and (v) disjoint-match, a goal and matched web service capabilities does not follow any of the previous definitions. Although matching notions relax the identification of target web services, in a future scenario in which thousands of services can potentially fulfill (or partially fulfill) the objectives described in a goal, a fine-grained classification of matching notions may be necessary for improving the degree of automation of the discovery process. One possible

approach is to identify a degree of matching inside of each matching notion. Thus, if we found one thousand web services that follow an intersection-match pattern, we need to distinguish which are the web services that are closer to the goal requested capability.

Brokers [5] bring another interesting approach to the problem of filtering the most promising web services. Brokers are intermediate systems between clients and service providers. They store web service capabilities and interfaces, execute matching processes for each goal that they have received, and manage the interaction between clients and selected web services. Thus after several interactions, brokers can gain valuable knowledge about which web services are providing a good service and which are not. A quality of service historical record can help in the identification of promising web services during the matching process executed in a broker.

Current Semantic Web services frameworks (e.g. OWL-S[1], WSMO[2] and Meteor-S[3]) use first order logic, description logics and logic programs to represent web service and goal capabilities and execute matching processes mostly based on subsumption checking or query-answering. In this article, we address the two problem areas raised above as part of a novel architecture for service matching, based on the incidence calculus. The incidence calculus [4] is a truth-functional probabilistic calculus in which the probabilities of composite formulae are computed from intersections and unions of the sets of worlds for which the atomic formulae hold true. Incidence Calculus can be easily integrated with other logic formalisms like propositional logic and logic programs and facilitate the implementation of a fine-grained matching mechanism based on probabilities and quality of service records.

The experiments were executed on a platform called F-X [17], a modular formal knowledge management system developed at University of Edinburgh. F-X has common roots with WSMO (both follows the main principles of UPML [3]), and can deal with WSMO/OWL-S ontologies and web services that fall into DLP fragment [8]. We will show how to specify service capabilities in F-Broker, and how incidence calculus can be nicely integrated

The paper is structured as follows: section 2 introduces semantic web services, F-X and Incidence Calculus. In section 3, the key implementation efforts are described, and testing results are discussed. Section 4 provides a short review of related work on probabilistic logic in the Semantic Web. Finally, conclusions and future work are included in section 5.

2 Preliminaries

Commonly in a Virtual Travel Agency scenario, customers require services in terms of goals (for instance, "*I want to book the cheapest flight and hotel available. The destination is Galway and I want to go on the 4th of November and back to San Francisco on the 9th of November"*). Airline companies and hotels provide services

[1] http://www.daml.org/services/owl-s/
[2] http://www.wsmo.org/
[3] http://lsdis.cs.uga.edu/projects/meteor-s/

("*to book a flight please provides: origin, destination, departure date, return date, valid passport id and credit-card*"). The broker is the *virtual travel agency* that stores service descriptions related with hotel and flights booking and attend requests from customers. We will show in this section how F-X can become in an efficient virtual travel agency for representing, storing and matching services. First we will introduce F-Broker, the broker component, and then we will describe incidence calculus and how this formalism can be integrated in F-Broker to improve its matching capabilities.

2.1 F-Broker

F-Broker [17] is an automated broker mechanism of F-X with the responsibility to identify the assemblies of knowledge components appropriate to a task we wish to achieve. This information is specified using F-Comp. In a multi-agent environment, agents advertise their *competences* (or capabilities, defined in the knowledge components they contain) simply by sending these to F-Broker, which records the competences and the agents who claim to be able to supply them.

When other agent sends a query, the broker processes it, and constructs an internal description, *brokerable structure*, based in the competences that previously it recorded which describes how the query might be answered. In the final stage the broker translates its brokerable structure into a sequence of *performative statements* describing the messages that will be necessary to establish a communication with the agents that can attend the query. The broker manages the communication between agents (request and providers) sending and receiving messages which the appropriate information to response the query [17].

[17] describes how capabilities and related brokerable structures are represented in previous versions of F-X. Four forms of capability, *C*, each of which is implemented within the expression `cap(K, C)` , denoting that the agent named `K` can deliver capability, `C` in at least one instance or, if not, will signal failure. Valid options for `C` are [17]:

- A **unit goal** of the form $P(A_1,\ldots,A_n)$, where P is a predicate name and $A_1,\ldots, A_n$ are its arguments.
- A **conjunctive goal** of the form $(C1\wedge\ldots\wedge Cm)$, where each C_i is a unit goal or a set expression.
- A **set expression** of the form `setof(X,C,S)` , where `C` is either a unit goal or a conjunctive goal; `X` is a tuple of variables appearing in `C`; and `S` is a set of instances of those tuples which satisfy `C`.
- A **conditional goal** of the form $C_c \leftarrow C_p$, where C_c is a unit goal which the agent, *K*, will attempt to satisfy (but will not guarantee to satisfy) if the condition, C_p , is satisfied. C_p is either a unit goal or a conjunctive goal.

Although for simplicity, we will use this version of the capability language, in later versions of FX, capabilities are represented following the next pattern:

```
service(Agent, Uri, Ontology, [Service1:-Preconditions1, Inputs1, Outputs1,Externals1], [...],..., [...]).
```

A simple brokerable structure has the form c(K, C), where K is the name of the agent which should be able to deliver the capability and C is a description of the sources of the capability. C can be in any of the following forms [17]:

- A capability available directly from K.
- A term of the form `c(K, dq(Q,QC))`, where `Q` is a capability obtainable from K conditional on its other capabilities and `QC` describes how these capabilities are obtained.
- A term of the form `c(K, pdq(Q,QC,QP))`, where `Q` is a capability obtainable from `K` conditional on its other capabilities and on capabilities external to `K`, and `QC` and `QP` describe how these internal and external capabilities (respectively) are obtained.
- A term of the form `c(conj, co(CQ1,CQ2))`, where `CQ1` and `CQ2` are two capability structures which must jointly be satisfied.
- A term of the form `c(K, cn(Q, G, c(K1,Q1)))`, where `K1` is the name of an agent different from `K` which allows capability structure `Q` to be delivered in combination with capability structure `Q1` provided that the correspondence constraints given by `G` are satisfiable.

Given a query posed by a client, a *broker* tries to find all the possible ways in which agents which have advertised their capabilities might be contacted in order to satisfy that query. It is necessary a formal representation of this sort of combination of capabilities, for which we use what we call a brokerage structure, of the form `c(K, C)`, where `K` is the name of the agent which should be able to deliver the capability and `C` is a description of the sources of the capability. `C` can be in any of the following forms [17][4]:

```
broker(Q,c(K,Q))
  ←cap(K,Q).
broker(Q, c(K, dq(Q,QC)))
  ←cap(K, (Q←C)) ∧
    broker(C,QC).
broker(Q, c(K1,pdq(Q,QC,QP)))
  ←p_cap(K1, (Q←C), P) ∧
    broker(C,QC) ∧
    e_broker(P,K1,QP).
broker((Q1,Q2), c(conj,
co(CQ1,CQ2)))
  ←broker(Q1,CQ1) ∧
    broker(Q2,CQ2).
broker(Q2, c(K2, cn(Q2, G,
c(K1,BQ))))
  ←corr(K1,Q1,K2,Q2,G) ∧
    Broker(Q1, c(K1,BQ)).
```

```
e_broker(Q, Kn, c(K,Q))
  ←cap(K,Q) ∧ not(K=Kn).
e_broker(Q, Kn, c(K, dq(Q,QC)))
  ←cap(K, (Q←C)) ∧ not(K=Kn) ∧
    broker(C,QC).
e_broker(Q, Kn, c(K1,
pdq(Q,QC,QP)))
  ←p_cap(K1, (Q←C), P) ∧
    not(K1=Kn) ∧ broker(C,QC) ∧
    e_broker(P,K1,QP).
e_broker((Q1,Q2), Kn, c(conj,
co(CQ1, CQ2)))
  ←e_broker(Q1,Kn,CQ1) ∧
    e_broker(Q2,Kn,CQ2).
e_broker(Q2, Kn, c(Kn, cn(Q2, G,
c(K1,BQ))))
  ←corr(K1,Q1,Kn,Q2,G) ∧
    broker(Q1, c(K1,BQ)).
```

[4] "corr" represents a correspondence, the equivalent of a bridge in UPML [3].

2.3 Incidence Calculus

Bundy [4] demonstrated that purely numeric probabilistic formalism can derive into contradictory results during the calculation of an uncertainty measure of complex formula. The key result of his analysis is that in general $P(A \wedge B) \bullet P(A) * P(B)$.

Incidence Calculus [4] reviews the notions of probability theory and introduces an important novelty: "*the probability of a sentence is based on a sample space of elements. Each element defines a situation in a possible world where a sentence can be true or false. The sample space,* $\top$*, contains an exhaustive and disjoint set of elements that for computational reasons should be finite*".

The incidence of a sentence A, i(A), is the subset of W in which sentence A is true. The dependence or independence of two sentences, A and B, is defined by the amount of common points of the result of the intersection between their incidences, $i(A) \cap i(B)$.

The axioms of Incidence Calculus [4] associate a set of theoretic function with each connective, propositional constant and quantifier of Predicate (Propositional) Logic so that the incidence of a complex sentence can be calculated from the incidences of its sub-sentences. The probabilities of composite formulae are computed from intersections and unions of the sets of worlds for which the atomic formulae hold true. Bundy called the resulting system Predicate (Propositional) Incidence Logic [4]:

$i(\top) = \{\}$ $\quad$ $i(\bot) = \{\}$

$i(A) = i(A)$ $\quad$ $i(\neg A) = i(\top) \backslash i(A)$

$i(A \wedge B) = i(A) \cap i(B)$ $\quad$ $i(A \vee B) = i(A) \cup i(B)$

$i(A \rightarrow B) = i(\neg A \vee B) = (i(\top) \backslash i(A)) \cup i(B)$

Thus, probabilities are calculated in the following way [4]:

$P(\top) = |i(\top)| = 1$ $\quad$ $P(\bot) = |i(\bot)| = 0$

$P(A) = |i(A)| / |i(\top)|$ $\quad$ $P(\neg A) = 1 - |i(A)| / |i(\top)|$

$P(A \wedge B) = |i(A) \cap i(B)| / |i(\top)|$

$P(A \vee B) = (|i(A) \cup i(B)| - |i(A) \cap i(B)|) / |i(\top)|$

$P(A|B) = |i(A) \cap i(B)| / |i(B)|$

As an illustration, consider the following set of incidences describing the weather of a given week adopted from [4]:

Suppose there are two propositions, P={rainy, windy} and seven possible worlds, $\top$ ={sunday, monday, tuesday, wednesday, thursday, friday, saturday}. Suppose that each possible world is equally probable (i.e. 1/7), and we learn that rainy is true in four possible worlds (friday, saturday, sunday and monday) and windy is true in three possible worlds (Monday, wednesday and Friday). Therefore, we can derivate the following incidence sets [4]:

i(rainy) = {friday, saturday, sunday, monday}

i(windy)= {monday,wednesday, friday}

$i(windy \wedge rainy)$= {monday, friday}

Moreover, we can calculate their probabilities in the following way:

P(rainy) = |i(rainy)| / |i(⊤)|=4/7
P(windy) = |i(windy)| / |i(⊤)|=3/7
P(windy∧rainy)= | i(windy)∩i(rainy)| / |i(⊤)|=2/7

2.3 Travel Agency example, writing capabilities in F-Broker

For simplicity we will use the capability language of an earlier version of F-Broker presented in [17]. We extend the capability language to store in a list the number of incidences in which each atomic capability was execute successfully (a client used this service for a given goal). Initially the set of incidences is empty and after several computations the broker is populating the sets of incidences according with the results in the requests attended. For our traveling scenario capabilities, we can model the services related with an airline company in the following way:

```
n_requests = [1,2,3,4,5, … , 320].

p_capability(airline_aa, ((book_flight(Person, Flight, Ori-
gin, Destination, DepartureDate, ArrivalDate, PurchaseOrder,
Price, Currency, PaymentMethod) :- flight(Flight, Origin,
Destination, DepartureDate, ArrivalDate, Price, Currency)),
pay_order(Person, Nationality, PurchaseOrder, Price, Cur-
rency, PaymentMethod))).

capability(airline_aa, (flight(Flight, Origin, Destination,
DepartureDate, ArrivalDate, Price, Currency), [3,4,5, … ,
301]).

capability(airline_ib, (flight(Flight, Origin, Destination,
DepartureDate, ArrivalDate, Price, Currency), [1,2, … ,
319]).

capability(airline_ba, (flight(Flight, Origin, Destination,
DepartureDate, ArrivalDate, Price, Currency) [6,7, … , 318).

p_capability(financial_vs, pay_order(Person, PurchaseOrder,
Price, Currency, PaymentMethod):-
has_money(Person,Price,Currency, PaymentMethod),
has_passport(Person, Nationality))).

capability(financial_vs,, has_money(Person,Price,Currency,
PaymentMethod), [2,3,4, … , 315]).

capability(financial_ms,, has_money(Person,Price,Currency,
PaymentMethod), [5,6 … , 320]).

capability(financial_amex,, has_money(Person,Price,Currency,
PaymentMethod)[100,105, …, 255]).

capability(police, has_passport(Person, Nationality), [3,4,5,
… , 301]).
```

...

3 Implementation and Results

We present a set of extensions in F-X to allow the system to deal with many OWL-S service profiles, take advantage of a probabilistic mechanism based on Incidence Calculus and relax the matching process.

3.1 From Description Logics to Description Logic Programs.

One of the objectives of the implementation was to test F-Broker with real examples of Semantic Web Services descriptions and also to integrate it in an industrial standard in order to find possible business applications. Many web services are annotated using DAML-S Service Profile descriptions. So we thought that it could be a good idea to provide a translator that semi-automatically converts services descriptions from DAML-S into F-Broker Service Description Language (SDL). One of the difficulties is how to translate DL logical statements into Prolog statements.

Description Logic Programs (DLP)[8], is an expressive fragment of the intersection of Description Logics (DL) [2] and Logic Programs (LP) [13]. An important result of the development of this formalism is DLP-fusion, a bidirectional translation of premises and inferences from DLP fragment of DL to LP, and vice versa from DLP fragment of LP to DL that allows Prolog to describe on expressive subset of DL. The implementation of DLP-Fusion in Prolog is straightforward [14] and with this translator F-Broker is able to import and export knowledge represented using Description Logics.

3.2 Extending matching algorithm

This section describes the necessary extensions to the matching algorithm of F-Broker in order to incorporate subsumption reasoning, matching notions (exact, plug-in, subsume, intersection and disjoint), a fine-grained degree of matching for some of these matching notions, and finally a evaluation algorithm based on historical records. We follow a bottom-up approach in which any new functionality is tested before we continue with the implementations of new refinements.

Subsumption reasoning. A Meta-interpreter for a language is an interpreter for the language written in the language itself [20]. Meta-interpreters are powerful tools that were widely used for implementing the inference engines of many expert systems. Using these features the programmers can modify the behaviour of the interpreter of the language. Goal reduction is the best known and most widely used meta–interpreter that in Prolog is called *Vanilla* [20]. *Vanilla* does not support *subsumption*. So, the first step during the implementation process was the integration of substitution of vanilla meta-interpreter by the simple subsumption meta-interpreter. The integra-

tion of the subsumption mechanism with the brokering algorithm is very simple. It is only to add a clause subs in any of the brokerable predicates that compound the brokering algorithm for subsumption checking of terms:

```
brokerable(Q, c(S,Q)) :-
capability(S, Q1),
subs(Q1,Q).
```

Matching notions. The algorithm that evaluates the degree of matching basically compares two lists of terms that belong to a web service capability and a goal, verifies the number of common and no common terms, determines the appropriate notion of matching following the previous classification and returns a value with the notion of matching identified. According to the view described in [10], abstract services and goals are both represented as sets of objects during the service discovery step. Thus, the calculation of the notion of match can be naturally calculated using incidence calculus. The implementation is also simple. We substitute the subsumption clause in the brokerable predicates implemented before for a new clause that call a new algorithm that evaluates and return the notion of match between a capability and goal:

```
brokerable(Q, c(S,Q,Nmatch)) :-
capability(S, Q1),
matchingnotion(Q1,Q,Nmatch),
Nmatch<>"disjoint".
```

Instead of carrying out strings like "disjoint" or "exact", it should be interesting to carry numeric values that can be reused for the calculation of a joint probability of several composed services.

Degree of matching notion. The previous algorithm can be improved by using a degree of matching that qualified the goodness of the matching notion identified. To do this, we include a new return variable in the matchingnotion predicate with the value that the incidence calculus algorithm calculates during the evaluation of common terms between capability and goal.

```
brokerable(Q, c(S,Q,Nmatch, Dmatch)) :-
capability(S, Q1),
matchingnotion(Q1,Q,Nmatch, Dmatch),
Nmatch<>"disjoint".
```

Evaluation of historical records. The proposal described in the current section focus the evaluation of the brokerable structures according to an historical record of previous goals. Associated with any atomic service capability there is a list of successful previous goals. This notion of a set of points (previous goals) fits perfectly with the probabilistic mechanism Incidence Calculus introduced in the previous section. In this case, the implementation requires the modification the atomic capabilities that have to maintain a list of values:

```
brokerable(Q, c(S, Q, L)) :-
capability(S, Q1, L),
```

A predicate called *evaluate* finds all the possible broker structures that can satisfied a request and evaluate the different structures according with the information of the history record. During the interaction with the client, the broker should modify the set of previous request of the service that successfully attend the demand of the client:

```
|?- evaluate(time(T), L).
L = [c(sd,time( A),[1,2]),2/4] ?
yes
```

3.3 Discussion

The extended version of F-Broker was tested with a modified version of the ecologic knowledge base [19] and slightly adapted versions of several web services examples from DAML[5], Mindswap[6] and Carnegie-Mellon[7]. [14] shows that the use of incidence calculus does not make significantly worse the performances of the broker with respect to the original version of F-Broker, and the relaxation of the matching process and the filtering of services based on a list of previous experiences of goals improve the matching abilities of the matching algorithm.

In [11] the use of incidence calculus was tested with a more advanced version of F-Broker that includes a lightweight coordination calculus (LCC) [16], a method for specifying agent interaction protocols. Lambert and Robertson use incidence calculus for the evaluation of services based on an historical record. The use of incidence calculus clearly helps to identify most promising services and thus satisfied client goals more efficiently.

[14] identified an important limitation of the use of incidence calculus to evaluate web services based on an historical record of previous goals. This is the incapacity of the system to handle the changes that the environment undergoes in a specific periods of time. For instance, the provider of a service with a large and excellent history record can fall. Any request of the clients that asks for this service will be processed by the broker and the answer will include the service that the provider cannot supply. After many requests another service could overcome the re-cord of the unavailable service, but before this moment the broker will try to execute the wrong service.

4 Related Work

The use of probabilistic logic in the context of the Semantic Web has not been explored in detail. Even the inventor of the Semantic Web, Sir Tim Berners-Lee, mentioned during the dev day lunchtime session at WWW2004 conference[8], that the Se-

[5] http:// www .daml.org/services/examples.html
[6] http://www.mindswap.org/2002/services/
[7] http://www. daml.ri.cmu.edu/ont/TaskModeler/TMont-index.html# Request Realtor1
[8] http://esw.w3.org/mt/esw/archives/000055.html

mantic Web stack does not need a representation of uncertainty. The first serious attempt to incorporate probabilistic reasoning in the Semantic Web was done with P-SHOQ[18]. Unfortunately, this work was not taken into consideration by the Semantic Web Community. A detailed description of an early version of this work can be found in my master thesis, "*Dealing with uncertainty in semantic web services*" [14]. This work was the first attempt to incorporate incidence calculus in a broker for semantic web services. [11] based on this previous experience incorporates the use of incidence calculus in an advance version of F-Broker that includes a lightweight coordination calculus (LCC) [16], a method for specifying agent interaction protocols.

5 Conclusions and Future Work

The relaxation of the matching process and the evaluation web service capabilities based on a previous historical record of successful executions show the feasibility of the use of probabilistic logic in Semantic Web services. Uncertainty is present in functional aspects of Web Services like discovery, composition, interoperation, mediation, monitoring and compensation [1]. In this paper, we focused only in discovery, and in [14], composition is also studied.

Incidence calculus was an excellent choice because its simplicity, rigor and compatibility with other classical logic formalisms. F-Broker provides an excellent test platform for the evaluation of incidence calculus in semantic web services. Although simple, F-Broker provides all basic functionality of a broker and allows the composition of web services capabilities and the execution of services based on an elementary vocabulary inspired in KQML. The code is very compact and clean, and new extensions are easily to include.

Future work will concentrate in the migration of the test platform to more realistic scenarios and the evaluation of other probabilistic logic formalism that combines logic programming with description logics.

Acknowledgements

This work has been partially supported by the SFI (Science Funds Ireland) under the DERI-Lion project, and the European Commission under the project Knowledge Web.

References

1. S. Arroyo and D. Fensel (2004). The Semantic Web Service Usage Process. No published.
2. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider. The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2003.

3. V. R. Benjamins, D. Fensel, E. Motta, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga. The Unified Problem-solving Method Development Language UPML, February 1999. Esprit Project 27169 IBROW 3 (An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web.
4. A. Bundy. Incidence Calculus. In Encyclopedia of Artificial Intelligence, pages 663–668. 1992.
5. K. Decker and K. Sycara. Middle-Agents for the Internet. In Proceedings of ICJCAI-97, January 1997.
6. T. Finin, Y. Labrou, and J. Mayfield. KQML as a Agent Communication Language. Software Agents, 1997. J.M. Bredshaw, AAAI Press/MIT Press.
7. J. Gonzalez-Castillo, D. Trastour, and C. Bartolini. Description logics for matchmaking of services. In KI-2001 Workshop on Applications of Description Logics, September 2001.
8. B. N. Grosof, I. Horrocks, R. Volz, and S. Decker. Description Logic Programs: Combining Logic Programs with Description Logics. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 48–57, 2003.
9. H. Haas and A. Brown (2004). Web Services Glossary. 2004. http://www.w3.org/TR/ws-gloss/
10. U. Keller, R. Lara, and A. Polleres (eds.). WSMO Web Service Discovery. Technical report, DERI, November 2004.
11. D. Lambert and D. Robertson. Matchmaking and Brokering Multi-Party Interactions Using Historical Performance Data. To appear in the Fourth International Joint Conference on Autonomous Agents and Multi Agent Systems, Utrecht 2005.
12. L. Li and I. Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. In WWW'03, Budapest, Hungary, May 2003.
13. J.W. Lloyd. Foundations of logic programming (second extended edition). Springer series in symbolic computation. Springer-Verlag, New York, 1987.
14. F. Martin-Recuerda. Dealing with uncertainty in Semantic Web services. MSc Thesis. University of Edinburgh. 2003.
15. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Service Capabilities. In ISWC, pages 333–347. Springer Verlag, 2002.
16. Robertson, D.: A lightweight method for coordination of agent oriented web services. In: Proceedings of the 2004 AAAI Spring Symposium on Semantic Web Services, California, USA (2004)
17. D. Robertson. F-X: A Formal Knowledge Management System. (unpublished), August 2001.
18. Thomas Lukasiewicz and Rosalba Giugno. P-SHOQ (Dn) : A Probabilistic Extension of SHOQ(Dn) for Probabilistic Ontologies in the Semantic Web. Technical report. Institut f¨ur Informations systeme, Technische Universität Wien,
April 2002. Technical Report Nr. 1843-02-06.
19. D. Robertson, A. Bundy, R. Muetzelfeldt, M. Haggith, and M Uschold. Eco-Logic: Logic-Based Approaches to Ecological Modeling. MIT Press (Logic Programming Series), 1991.
20. L. Sterling and E. Shapiro. The Art of Prolog: Advanced Programming Techniques, 2nd Edition. MIT Press, 1994.
21. K. Sycara, S. Widoff, M. Klusch, and J. Lu. LARKS: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. Autonomous Agents and Multi-Agent Systems, pages 173–203, 2002.