EMERALD: A Multi-Agent System for Knowledge-based Reasoning Interoperability in the Semantic Web

Kalliopi Kravari, Efstratios Kontopoulos and Nick Bassiliades,

Dept. of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece
{kkravari, skontopo, nbassili}@csd.auth.gr

Abstract. The Semantic Web aims at augmenting the WWW with meaning, assisting people and machines in comprehending Web content and better satisfying their requests. Intelligent agents are considered to be greatly favored by Semantic Web technologies, because of the interoperability the latter will achieve. One of the main problems in agent interoperation is the great variety in reasoning formalisms, as agents do not necessarily share a common rule or logic formalism. This paper reports on the implementation of EMERALD, a knowledge-based framework for interoperating intelligent agents in the Semantic Web. More specifically, a multi-agent system was developed on top of JADE, featuring trusted, third party reasoning services, a reusable agent prototype for knowledge-customizable agent behavior, as well as a reputation mechanism for ensuring trust in the framework. Finally, a use case scenario is presented that illustrates the viability of the proposed framework.

Keywords: semantic web, intelligent agents, multi-agent system, reasoning.

1 Introduction

The *Semantic Web* (*SW*) is a rapidly evolving extension of the WWW that derives from Sir Tim Berners-Lee's vision of a universal medium for data, information and knowledge exchange. The SW aims at augmenting Web content with meaning (i.e. semantics), making it possible for people and machines to comprehend the available information and better satisfy their requests. Until now, the fundamental SW technologies (content representation, ontologies) have been established and researchers are currently focusing their attention on logic and proofs.

Intelligent agents (*IAs*) are considered the most prominent means towards realizing the SW vision [1]. Via the use of IAs, programs are extended to perform tasks more efficiently and with less human intervention. The gradual integration of *multi-agent systems* (*MAS*) with SW technology will affect the use of the Web in the future; the next generation of the Web will consist of groups of intercommunicating agents traversing it and performing complex actions on behalf of their users. Intelligent agents are considered to be greatly favored by SW technologies, because of the interoperability the latter will achieve.

Nevertheless, a critical issue is now raised: agent interoperation is overwhelmed from the variety of representation and reasoning technologies. On the other hand, agents do not necessarily share a common rule or logic formalism. In fact, it will often be the case that two or more intercommunicating agents will 'understand' different (rule) languages. On the other hand, it would be unrealistic to attempt imposing specific logic formalisms in a rapidly changing world like the Web.

We propose a novel, more viable approach, which involves trusted, third-party reasoning services that will infer knowledge from an agent's rule base and verify the results. More specifically, this paper reports on the implementation of *EMERALD*, a framework for interoperating, knowledge-based IAs in the SW. A JADE MAS was extended with reasoning capabilities, provided as agents. Furthermore, the framework features a generic, reusable agent prototype for *knowledge-customizable agents* (*KC-Agents*), consisted of an agent model (*KC Model*), a yellow pages service (*Advanced Yellow Pages Service*) and several external Java methods (*Basic Java Library*). Also, since the notion of trust is vital here, a reputation mechanism was integrated in the framework. Finally, the paper presents a use case scenario that illustrates the usability of the framework and the integration of all the technologies involved.

The rest of the paper is structured as follows: Section 2 gives a brief overview of EMERALD, followed by descriptions of its various components. More specifically, the featured reasoning services are presented, as well as the knowledge-customizable agent prototype (KC-Agents). Since trust is essential in a framework like EMERALD, section 5 presents the deployed reputation mechanism. Finally, section 6 illustrates an apartment renting use case scenario, which better displays the potential of the framework. The paper is concluded with references to related work and conclusions, as well as directions for future improvements.

2 Framework Overview

As mentioned in the introduction, EMERALD is a common framework for interoperating knowledge-based intelligent agents in the SW. The motivation behind our work was to leverage the weaknesses in agent intercommunication outlined above and attempt to deploy trusted, third-party reasoning services instead. EMERALD is developed on top of JADE [2], the popular MAS Java framework.

Fig. 1 illustrates a general overview of EMERALD: Each human user controls a single all-around agent. Agents can intercommunicate, but do not necessarily share a common rule/logic formalism; therefore, it is vital for them to find a way to exchange their position arguments seamlessly. Our approach does not rely on translation between rule formalisms but on exchanging the rule base results. The receiving agent uses an external reasoning service to grasp the semantics of the rulebase, i.e. the set of rule base conclusions. In EMERALD, reasoning services are "wrapped" by an agent interface, called the *Reasoner*, allowing other IAs to contact them via ACL messages. Reasoners are, in essence, agents offering reasoning services to the rest of the agent community. Currently, the framework features a variety of few but widely diverse Reasoners (see section 3), but the available array can be easily extended.

Moreover, agents are knowledge-customizable, meaning that they are not confined in having their logics and strategies/policies hard-wired. Instead, they can be either generic or customizable; each agent contains a rule base that describes its knowledge of the environment, its behaviour pattern as well as its strategy/policy. By altering the rule base, the agent's knowledge and/or behaviour will instantly be modified accordingly. Currently, EMERALD provides a knowledge-based agent module based on Jess language, but our goal is to provide a range of modules based on a variety of rule languages (i.e. Prolog/Prova, RuleML). The use case scenario presented later in this work (section 6) demonstrates this feature.

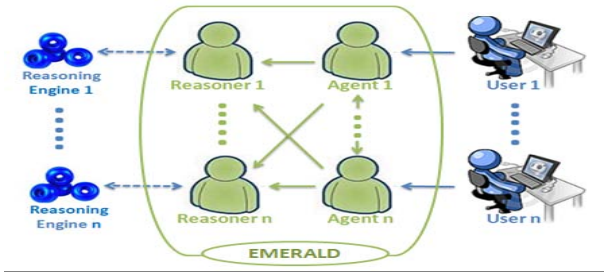

Fig. 1. Overview of the proposed framework.

Overall, the goal is to apply as many standards as possible (ACL, RuleML, RDF/S, OWL), in order to encourage the application and development of the framework. In practice, the SW serves as the framework infrastructure.

3 Reasoning Services

EMERALD integrates a number of reasoning engines that use various logics. The Reasoner, a reasoning service provided by a JADE IA, can call an associated reasoning engine in order to perform inference and provide results (Fig. 2). The procedure is straightforward: each Reasoner stands by for new requests (ACL messages with a "*REQUEST*" communication act) and as soon as it receives a valid request, it launches the associated reasoning engine and returns the results (ACL message with an "*INFORM*" communication act). Consequently, although Reasoners seem to be fully autonomous agents, actually they behave more like a web service.

Fig. 2. Input – Output of a Reasoner agent.

EMERALD currently implements a number of Reasoners that offer reasoning services in two major reasoning formalisms: deductive and defeasible reasoning. *Deductive reasoning* is based on classical logic arguments, where conclusions are proved to be valid when the premises of the argument (rule condition) are true. EMERALD implements two such reasoners, which are actually based on the logic programming paradigm, namely R-DEVICE [3] and Prova [4]. R-DEVICE is a deductive object-oriented knowledge base system for querying and reasoning about RDF metadata. R-DEVICE, transforms RDF triples into objects and uses a deductive rule language for querying and reasoning about them, in a forward-chaining Datalog fashion. Prova is a rule engine that supports distributed inference services, rule interchange and rule-based decision logic and combines declarative rules, ontologies and inference with dynamic object-oriented programming.

Defeasible reasoning [5] constitutes a simple rule-based approach for efficient reasoning with incomplete and inconsistent information. When compared to mainstream non-monotonic reasoning, the main advantages of defeasible reasoning are enhanced representational capabilities and low computational complexity. Currently, EMERALD supports two Reasoners that use defeasible reasoning, the *DR-Reasoner* and the *SPINdle-Reasoner* based on *DR-DEVICE* [6] and *SPINdle* [7], accordingly. *DR-Reasoner* was presented in [8]; *DR-DEVICE* accepts as input the address of a defeasible logic rule base, written in the OORuleML-like syntax. The rule base contains only rules; the facts for the rule program are contained in RDF documents, whose addresses are declared in the rule base. Finally, conclusions are exported as an RDF document. On the other hand, *SPINdle-Reasoner*, supports reasoning on both standard and modal defeasible logic. It accepts defeasible logic theories represented using XML or plain text (with pre-defined syntax), processes them and finally exports the results via XML.

4 KC-Agents: The prototype

EMERALD provides a generic, reusable agent prototype for knowledge-customizable agents (*KC-Agents*) that offers certain advantages, concerning, among others, modularity, reusability and interoperability of behavior between agents. The current prototype consists of an agent model (*KC Model*), a yellow pages service (*Advanced Yellow Pages Service - AYPS*) and some external Java methods (*Basic Java Library - BJL*). Fig. 3 displays the above prototype.

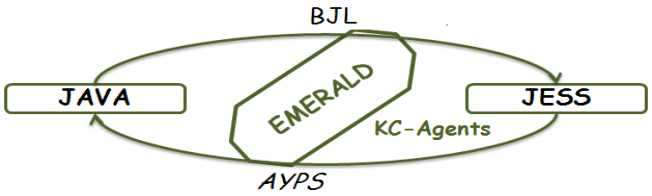

Fig. 3. The KC-Agents Prototype

4.1 The KC Model

The *KC Model* is a model for customizable agents equipped with a rule engine and a knowledge base (KB) that contains environment knowledge (in the form of facts), behaviour patterns (in the form of rules) and strategies. By altering the KB, both the agent's knowledge and behaviour are modified accordingly. KC model's abstract specification, presented in [9], contains facts and rules. A short description is presented below for better comprehension.

The generic rule format is: *result* ← *rule* (*preconditions*). The agent's internal knowledge is a set of facts $F \equiv F_u \cup F_e$, where $F_u \equiv \{f_{u1}, f_{u2}, \ldots, f_{uk}\}$ are user-defined facts and $F_e \equiv \{f_{e1}, f_{e2}, \ldots, f_{em}\}$ are environment-asserted facts. The agent's behaviour is represented as a set of potential actions–rules $P \equiv A \cup S$, where $A \equiv \{a \mid f_e \leftarrow a(f_{u1}, f_{u2}, \ldots, f_{un}) \wedge \{f_{u1}, f_{u2}, \ldots, f_{un}\} \subseteq F_u \wedge f_e \in F_e\}$ are the rules that derive new facts by inserting them into the KB and $S \equiv C \cup J$ are the rules that lead to the execution of a special action. Note that special actions can either refer to agent communication $C \equiv \{c \mid ACLMessage \leftarrow c(f_1, f_2, \ldots, f_p) \wedge \{f_1, f_2, \ldots, f_p\} \subseteq F\}$ or Java calls $J \equiv \{j \mid JavaMethod \leftarrow j(f_1, f_2, \ldots, f_q) \wedge \{f_1, f_2, \ldots, f_q\} \subseteq F\}$.

The overall goal for EMERALD is to provide a variety of *KC modules* (KC Model's implementations) based on several rule engines and languages. Currently, EMERALD provides a *KC_j module*, based on Jess [10]. *KC_j module* is equipped with a Jess rule engine and its knowledge base contains knowledge, in the form of Jess facts and behaviour patterns, in the form of Jess production rules.

4.2 Advanced Yellow Pages Service (AYPS)

Moreover, an advanced customized procedure for the yellow pages service is provided, both for registered and required services. In JADE, the traditional yellow pages service is provided by the *Directory Facilitator* (*DF*) agent. This procedure allows an agent to get the proper results from the *DF* agent, but, since these results have the form of ACL Message content, this means that the developer has to manually convert them, in order to use them as facts.

With our *advanced yellow pages service* (*AYPS*) model, the service is fully automated: the services that each agent provides and/or requests are declared in a separate file, accessible only by the specific agent. As soon as the agent wishes to advertise one or more services, AYPS is activated and places them into a repository. On the other hand, if the agent requires some specific service, AYPS traverses the repository and returns the proper providers. Concerning the KC_j module, AYPS returns the providers as Jess facts with a designated format: (*service_type* (*provider provider_name*)).

AYPS also implements a reputation mechanism (section 5) available for each EMERALD agent at any time. Thus, AYPS is able to provide not only the name but also the reputation value of a provider.

4.3 Basic Java Library (BJL)

In order to provide a standard communication interface between the KB and the JADE agent, we have developed a library of Java methods (*Basic Java Library - BJL*) that can be evoked from KC-agent's actions (in the current implementation Jess production rules). A generic syntax specification is:

```
(defrule call_method
    ;;; preconditions
  =>
    (bind ?t (new Basic))
    (bind ?str (?t method's_name argument(s))))
```

where *?t* is bound to a new instance of BJL and *?str* is the returned value.

For instance, method *createRulebase* creates a backup of a rule base, determining the RDF data input, and method *extractTriples* extracts triples from an RDF file and stores them as facts. In order to extract 'knowledge' from these facts, suitable rules can be executed. A sample rule finding out the attribute A of B is the following, in the context of KC_j module:

```
(defrule find_A
    (triple (subject ?x) (predicate rdf:type) (object dr-device:xx))
    (triple (subject ?x) (predicate dr-device:B) (object ?A))
  =>
   (assert(A ?A)))
```

5 Trust

Trust has been recognized as a key issue in SW MAS, where agents have to interact under uncertain and risky situations. Thus, a number of researchers have proposed, in different perspectives, models and metrics of trust, some involving past experience or using only a single agent's previous experience. Five such metrics are described in [11], among them *Sporas* seems to be the most widely used metric, although *CR* (*Certified Reputation*) is one of the most recently proposed methodologies. The overall goal for EMERALD is to adopt a variety of trust models, both proposed in the literature and original.

Currently, EMERALD adopts two reputation mechanisms, a decentralized and a centralized one. The decentralized mechanism is a combination of Sporas and CR, and was presented in [8]. In the centralized approach, presented here, AYPS keeps the references given from agents interacting with Reasoners or other agents in EMERALD. Each reference is in the form of $Ref_i=(a, b, cr, cm, flx, rs)$, where *a* is the *trustee*, *b* is the *trustor* and *cr* (*Correctness*), *cm* (*Completeness*), *flx* (*Flexibility*) and *rs* (*Response time*) are the evaluation criteria. Ratings vary from -1 (*terrible*) to 1 (*perfect*), $r \in [-1,1]$, while newcomers start with reputation equal to 0 (*neutral*). The final reputation value (R_b) is based on the weighted sum of the relevant references stored in AYPS and is calculated according to the formula:

$\sum R_b = w_1 * cr + w_2 * cm + w_3 * flx + w_4 * rs$, where $w_1 + w_2 + w_3 + w_4 = 1$. AYPS supports two options for R_b, a default where the weights are equivalent, namely $w_{k \in [1,4]} = 0.25$ each and a user-defined, where the weights vary from 0 to 1 depending on user priorities.

The simple evaluation formula of the above approach, compared to the decentralized one, leads to time profit as it needs less evaluation and calculation time. Moreover, it provides more guaranteed and reliable results (R_b) as it is centralized (AYPS), overcoming the difficulty to locate references in a distributed mechanism. Agents can use either one of the above mechanisms or even both, complementarily, namely they can use the centralized mechanism provided by AYPS in order to find the most trusted service provider and/or they can use the decentralized approach for the rest EMERALD agents.

6 Use case

Reasoning is widely used in various applications. This section presents an apartment renting use case paradigm that applies both deductive and defeasible logic. The scenario aims at demonstrating the overall functionality of the framework and, more specifically, the usability of the Reasoners and the modularity of the KC-Agents prototype and its ability to easily adapt to various applications.

The scenario, adopted from [12], involves two independent parties, represented by IAs and one of the four Reasoners provided in EMERALD. The first party is the *customer*, a potential renter that uses defeasible logic and wishes to rent an apartment based on his requirements (e.g. size, location) and personal preferences. The other party is the *broker*, who uses deductive logic and possesses a database of available apartments. His role is to match customer's requirements with the apartment specifications and eventually propose suitable apartments to the potential renter. The *R-Reasoner* and the *DR-Reasoner* are the two reasoners involved in the paradigm. The scenario is carried out in ten distinct steps (shown in Fig 4). A similar but more simplistic brokering scenario was presented in [8], where only one type of logic (defeasible) was applied and the broker did not possess any private interaction strategy, expressed in any kind of logic, but it was just mediating between the customer and the Reasoner.

Initially, the customer finds a broker, by asking the AYPS (step 1). The AYPS returns a number of potential brokers accompanied with their reputation ratings (step 2). Customer selects the more trusted broker and sends his requirements to him, in order to get back all the available apartments with the proper specifications (step 3). The broker has a list of all available apartments which cannot be communicated to the customer, because they belong to the broker's private assets. However, since the broker cannot process customer's requirements using defeasible logic, he finds a defeasible logic reasoner (DR-Reasoner) (step 4), by using the AYPS (this step is not shown). DR-Reasoner returns the apartments that fulfill all requirements (step 5); however, the broker checks the results in order to exclude the unavailable apartments or apartments reserved for a special private-to-the-broker reason. Thus, the broker agent requests from the R-Reasoner to process the results with his own private

interaction strategy expressed in a deductive logic rulebase (step 6). When he receives the remaining apartments (step 7), he sends them to customer's agent (step 8).

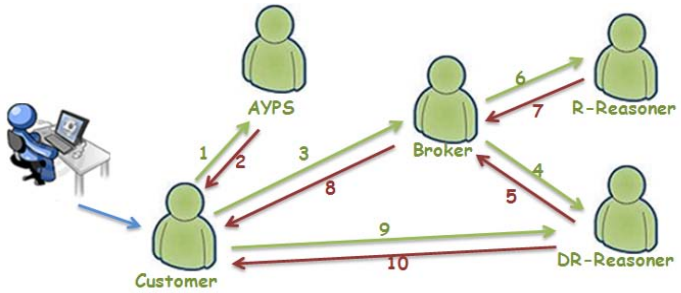

Fig. 4. Apartment renting scenario steps.

Eventually, the customer receives the appropriate list and has to decide which apartment he prefers. However, his agent does not want to send customer's preferences to the broker, in order not to be exploited; thus, customer's agent selects his most preferred apartment by sending to the DR-Reasoner his preferences, as a defeasible logic rulebase, along with the list of acceptable apartments (step 9). The Reasoner replies and proposes the best apartment to rent (step 10). The apartment selection procedure ends. Now the customer has to negotiate with the owner for the renting contract. This process is carried out in two basic steps, as shown in Fig. 5: first the customer's agent has to find out the apartment owner's name and then negotiate with him for the rent. The customer sends a *REQUEST* message to the broker containing the chosen apartment and waits for its owner's name. The broker, sends back his reply via an *INFORM* message. Afterwards, the customer starts a negotiation process with the owner, negotiating among others, the price and rental duration.

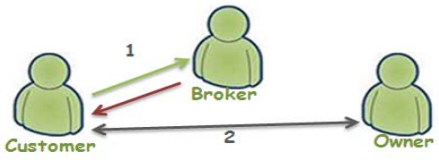

Fig. 5. Negotiation scenario steps.

Following the generic, abstract specification for agents, the customer agent's description contains a fact, *ruleml_path*, which is part of its internal knowledge and represents the rulebase URL. Moreover, due to the dynamic environment (AYPS is constantly updating the environment), a new fact with the agent name (*agent_name*) is added to the working memory. Agent behavior is represented by rules; one of these is the '*read'* rule that calls the BJL's *fileToString* method. It has only a single precondition (actually fact), the *ruleml_path*, as shown below.

$F_u^{cust} \equiv \{ruleml\_path\},\ F_e^{cust} \equiv \{agent\_name\}$

$J^{cust} \equiv \{rule\_base\_content \leftarrow (bind\ ((new\ Basic)\ fileToString\ ruleml\_path))\}$

Similarly, the broker agent's description contains facts and rules: fact *url* represents (part of) its internal knowledge and stands for the URL of the RDF document containing all available apartments, while *reasoner_name* (DR-Reasoner's name) is added by the environment due to AYPS and rules "*request*" and "*read*" (BJL's *fileToString*) comprise part of the agent's behavior.

$F_u^{brok} \equiv \{url\},\ F_e^{brok} \equiv \{reasoner\_name\}$

$C^{brok} \equiv \{(ACLMessage\ (communicative\text{-}act\ REQUEST)$
$(sender\ Broker)\ (receiver\ reasoner\_name)\ (content\ "request"))$
$\leftarrow request\ (reasoner\_name)\}$

$J^{brok} \equiv \{rule\_base\_content \leftarrow (bind\ ((new\ Basic)\ fileToString\ url))\}$

7 Related Work

DR-BROKERING, a system for brokering and matchmaking, is presented in [13]. The system applies RDF in representing offerings and a deductive logical language for expressing requirements and preferences. Three agent types are featured (Buyer, Seller and Broker) and a DF agent plays the role of the yellow pages service. Also, *DR-NEGOTIATE* [14], another system by the same authors, implements a negotiation scenario using JADE and DR-DEVICE. Similarly, our approach applies the same technologies and identifies roles such as Broker and Buyer. Conversely, we provide a number of independent reasoning services, offering both deductive and defeasible logic. Moreover, our approach takes into account trust issues, providing two reputation approaches in order to guarantee the interactions' safety.

The *Rule Responder* [15] project builds a service-oriented methodology and a rule-based middleware for interchanging rules in virtual organizations, as well as negotiating about their meaning. Rule Responder demonstrates the interoperation of distributed platform-specific rule execution environments, with Reaction RuleML as a platform-independent rule interchange format. We have a similar view of reasoning service for agents and usage of RuleML. Also, both approaches allow utilizing a variety of rule engines. However, contrary to Rule Responder, EMERALD is based on FIPA specifications, achieving a fully FIPA-compliant model and deals with trust issues. Finally, and most importantly, our framework does not rely on a single rule interchange language, but allows each agent to follow its own rule formalism, but still be able to exchange its rule base with other agents, which will use trusted third-party reasoning services to infer knowledge based on the received ruleset.

8 Conclusions and Future Work

This paper argued that agents are vital in realizing the Semantic Web vision and presented EMERALD, a knowledge-based multi-agent framework that provides

reasoning interoperability, designed for the SW. EMERALD, developed on top of JADE, is fully FIPA-compliant and features trusted, third party reasoning services, a generic, reusable agent prototype for knowledge-customizable agents, consisted of an agent model, a yellow pages service and several external Java methods. Also, since the notion of trust is vital here, a reputation mechanism was integrated for ensuring trust in the framework. Finally, the paper presents a use case scenario that illustrates the usability of the framework and the integration of all the technologies involved.

As for future directions, it would be interesting to verify our model's capability to adapt to an even wider variety of Reasoners and KC modules, in order to form a generic environment for cooperating agents in the SW. Another direction would be towards developing and integrating more *trust* mechanisms. As pointed out, trust is essential, since each agent will have to make subjective trust judgements about the services provided by other agents. Considering the parameter of trust would certainly lead to more realistic and efficient applications. A final direction could be towards equipping the KC-Agents prototype with user-friendly GUI editors for each KC module, such as the KC_j module.

References

[1] Hendler J.: Agents and the Semantic Web. *IEEE Intelligent Systems*, 16(2):30-37 (2001)

[2] JADE, http://jade.tilab.com/

[3] Bassiliades N., Vlahavas I.: R-DEVICE: An Object-Oriented Knowledge Base System for RDF Metadata. *Int. J. on Semantic Web and Information Systems*, 2(2): 24-90 (2006)

[4] Prova, www.prova.ws

[5] Nute D.: Defeasible Reasoning, *20th Int. Conference on Systems Science*, IEEE Press, pp. 470-477 (1987)

[6] Bassiliades N., Antoniou G., Vlahavas I.: A Defeasible Logic Reasoner for the Semantic Web. *Int. J. on Semantic Web and Information Systems*, 2(1):1-41 (2006)

[7] Lam H., Governatori G.: The Making of SPINdle. *RuleML-2009 Int. Symp. on Rule Interchange and Applications*, Springer, pp. 315-322 (2009)

[8] Kravari K., Kontopoulos E., Bassiliades N.: A Trusted Defeasible Reasoning Service for Brokering Agents in the Semantic Web. *3rd Int. Symp. on Intelligent Distributed Computing (IDC'09)*, Springer Berlin/Heidelberg, Vol. 237, pp. 243-248, Cyprus (2009)

[9] Kravari K., Kontopoulos E., Bassiliades N.: Towards a Knowledge-based Framework for Agents Interacting in the Semantic Web, *IEEE/WIC/ACM Int. Conf. on Intelligent Agent Technology (IAT'09)*, Vol. 2, pp. 482-485, Italy (2009)

[10] JESS, http://www.jessrules.com/

[11] Macarthur K.: Trust and Reputation in Multi-Agent Systems. *AAMAS*, Portugal (2008)

[12] Antoniou G., Harmelen F.: *A Semantic Web Primer*. MIT Press (2004)

[13] Antoniou G., Skylogiannis T., Bikakis A., Bassiliades N., "DR-BROKERING – A Defeasible Logic-Based System for Semantic Brokering", *IEEE Int. Conf. on E-Technology, E-Commerce and E-Service*, pp. 414-417 (2005)

[14] Skylogiannis T., Antoniou G., Bassiliades N., Governatori G., Bikakis A.: DR-NEGOTIATE - A System for Automated Agent Negotiation with Defeasible Logic-based Strategies. *Data & Knowledge Engineering*, 63(2):362-380 (2007)

[15] Paschke A., Boley H., Kozlenkov A., Craig B., "Rule responder: RuleML-based Agents for Distributed Collaboration on the Pragmatic Web", *2nd Int. Conf. on Pragmatic Web*, ACM, vol. 280, pp. 17-28 (2007)