Towards a Knowledge-based Framework for Agents Interacting 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 making Web content understandable both for people and machines. Although intelligent agents can assist towards this vision, they do not have to conform to a common rule or logic paradigm. This paper reports on the first steps towards a framework for interoperating knowledge-based intelligent agents. A multi-agent system was extended with defeasible reasoning and a reusable agent model is proposed for customizable agents, equipped with a knowledge base and a Jess rule engine. Two use case scenarios display the integration of these technologies.**

Keywords: intelligent agents, multi-agent system, defeasible reasoning, brokering agents, agent negotiation

I. INTRODUCTION

The *Semantic Web* (*SW*) is an evolving extension of WWW, where the semantics of information and services is well-defined, making it possible for people and machines to understand Web content and satisfy their requests [1]. SW research is currently focusing on logic and reasoning. *Intelligent agents* (*IAs*) can be favored by SW technologies [2], because of the interoperability SW offers. The integration of *multi-agent systems* (*MAS*) and SW technology will notably affect the use of the Web – its next generation will feature groups of intercommunicating agents traversing it and performing complex actions on behalf of their users.

A core setback in agent interoperation is the variety in representation and reasoning, as there is still no globally agreed knowledge representation and reasoning formalism for agents. The diversity in representation and reasoning could be confronted through "burdening" agents with their own reasoning engines. However, since each rule engine uses a specific formalism, this would pose the need for interchange languages and/or semantic translation schemes.

This paper reports on the initial steps towards a framework for interoperating, knowledge-based IAs in the SW that avoids the drawbacks outlined above and proposes a simpler approach that does not rely on semantic interoperability, but on utilizing third-party reasoning services instead. In our approach, reasoning services are wrapped by an agent interface (the *Reasoner*), allowing communication via ACL messages, whereas SW standards (*RuleML*, *RDF/S*, *OWL*) will serve as the framework infrastructure. A JADE MAS was extended with *defeasible reasoning* [3], which offers the ability to reason with incomplete and inconsistent information.

We also propose a generic, reusable agent model for *knowledge-customizable agents* (*KC-Agents*), i.e. agents, equipped with a Jess rule engine and a knowledge base (KB) that contains environment knowledge, behaviour patterns and strategies/policies. By altering the KB, the agent's knowledge and/or behaviour is modified accordingly. This proposal is orthogonal to the reasoning service and offers declarativeness, modularity, reusability, maintainability, and eventually, interoperability of behaviour between agents. Finally, the paper presents a brokering and a negotiation scenario that illustrate the integration of the above technologies.

In the rest of the paper, section II briefly presents the deployed reasoning engine, while section III features the implemented MAS and KC-Agents. The use case scenarios are described next, followed by a description of related work and final remarks as well as directions for future work.

II. DEFEASIBLE REASONING AND DR-DEVICE

Defeasible reasoning [3] 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.

DR-DEVICE [4] is a defeasible logic reasoner that employs an OO RDF data model, treating properties as normal encapsulated attributes of resource objects. This way, properties are not scattered across several triples, resulting in increased query performance. DR-DEVICE supports a RuleML-compatible syntax for defeasible logic rules that deals with extensions regarding rule types, superiority relations among rules and conflicting literals, as well as constraints on predicate arguments and functions.

DR-DEVICE accepts as input the address of a defeasible logic rule base that contains only rules. The facts for the rule program are contained in RDF documents, whose addresses are declared in the rule base. The rule base is submitted and the designated facts are downloaded. During inference, rule conclusions are materialized inside DR-DEVICE as objects and the instances of derived classes are exported as an RDF document, which includes RDF/S definitions for the exported derived classes and those instances of the exported derived classes that have been proved. Finally, results can be accessed through a web browser or via specialized visualization software. More details regarding the system architecture and functionality can be found in [4].

III. KC-Agents

This section focuses on *KC-Agents* that are based on our customizable, knowledge-based agent model. The other contribution of the framework, the *Reasoner* agent that provides the defeasible reasoning service, is presented thoroughly in [5] and will be omitted here. *KC-Agents* are customizable agents equipped with a KB and a Jess rule engine. The KB contains the agent's knowledge (facts) and behavior pattern (production rules). KC-Agents are described by an abstract specification portrayed below that contains facts and rules; the generic rule format is represented by: *result* ← *rule* (*preconditions*).

The agent's internal knowledge is essentially a set F of facts that consists of subset F_u of user-defined facts and subset F_e of environment-asserted facts:

$$F_u \equiv \{f_{u1}, f_{u2}, \dots, f_{uk}\},\ F_e \equiv \{f_{e1}, f_{e2}, \dots, f_{em}\},\ F \equiv F_u \cup F_e$$

Agent behavior is a set P of potential actions, expressed as Jess production rules. P consists of rules that derive new facts by inserting them into the KB (subset A) and rules that lead to the execution of a special action (subset S). Special actions can either refer to agent communication (subset C) or Java calls (subset J):

$$P \equiv A \cup S,\ S \equiv C \cup J$$

$$A \equiv \{a \mid f_e \leftarrow a(f_{u1}, f_{u2}, \dots, f_{un}) \land \{f_{u1}, f_{u2}, \dots, f_{un}\} \subseteq F_u \land f_e \in F_e\}$$

$$C \equiv \{c \mid ACLMessage \leftarrow c(f_1, f_2, \dots, f_p) \land \{f_1, f_2, \dots, f_p\} \subseteq F\}$$

$$J \equiv \{j \mid JavaMethod \leftarrow j(f_1, f_2, \dots, f_q) \land \{f_1, f_2, \dots, f_q\} \subseteq F\}$$

ACLMessage is a Jess template for defining ACL messages and *JavaMethod* is a user-defined Java method. The communication rule syntax specification is:

```
(defrule Communication_Rule
    ;;; rule preconditions
    =>
    (ACLMessage (communicative-act ?c)
        (sender ?s) (receiver ?r) (content ?n)))
```

where *communicative-act*, *sender*, *receiver* and *content* are four template parameters of *ACLMessage*, according to Fipa2000 description. On the other hand, user-defined Java methods can be called inside Jess rules to perform a specialized action, like processing specialized file content. A generic syntax specification is:

```
(defrule JavaMethod_Rule
    ;;; rule preconditions
    =>
    (bind ?x (new java_class_name))
    (bind ?y (?x java_method_name $?a)))
```

where *?x* is bound to a new instance of a specific Java class, *$?a* is the list of arguments required by the specific class method and *?y* is the returned result.

IV. Cooperating Agents – Use Cases

Defeasible reasoning is applied in various applications, like brokering [6], bargaining and agent negotiations [7], which are also influenced by agent-based technology (e.g. [8]). Thus, two use cases are described (a brokering and a negotiation scenario), which are deliberately diverse, in order to display the ability of KC-Agents to easily adapt to various applications.

A. Use Case: A Brokering Scenario

The brokering scenario was adopted from [9] and presented in [5]. It is extended here, in order to introduce the newly-developed KC-Agents. The scenario involves three parties: (a) the *customer*, called Carlo, is a potential renter that wishes to rent an apartment based on his requirements (e.g. size, location, floor) and personal preferences, (b) the *broker* possesses an RDF database of available apartments; his role is to match Carlo's requirements with the apartment specifications and propose suitable flats to the potential renter, and, (c) the *reasoner* is an independent third-party service that can conduct inference on defeasible logic rule bases and produce the results as an RDF file.

As described in [5], the scenario is carried out in six distinct steps and ends up with Carlo finally deciding the most suitable apartment, based on his requirements and personal preferences.

1) Agent Specifications: Following the generic specification for agents (section III), 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, a new fact with the agent name (*agent_name*) is added to the working memory. Agent behavior is represented by rules; two of these are "*request*" and "*read*"; the former is used for communication and the latter for Java method calls. Both rules require a single precondition each: *agent_name* and *ruleml_path*, respectively.

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

$$C^{cust} \equiv \{(ACLMessage\ (communicative\text{-}act\ REQUEST)$$
$$(sender\ agent\_name)\ (receiver\ Broker)$$
$$(content\ \text{“}request\text{”})) \leftarrow request\ agent\_name)\}$$

$$J^{cust} \equiv \{rule\_base\_string \leftarrow$$
$$(bind\ ((new\ java\_class)\ read\ ruleml\_path))\}$$

The broker agent's description contains facts and rules: *url* represents its internal knowledge and stands for the URL of the RDF document containing all apartments, while *reasoner_name* (i.e. the Reasoner's name) is added by the environment and rules "*request*" and "*read*" comprise part of the agent's behavior.

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

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

$$J^{brok} \equiv \{rule\_base\_string \leftarrow$$

(*bind* ((*new java_class*) *read url*))}

B. *Use Case: A Negotiation Scenario*

As soon as the brokering trade ends and Carlo is able to make a choice, he delegates a new task to his IA (*Customer*), to negotiate for the rent of the chosen apartment. The scenario involves again three agents, where the Reasoner is substituted by the owner's agent (*Owner*). No reasoning service is deployed here, although the agents could indeed utilize the Reasoner for applying their negotiation strategies.

Customer's agent initially has to find out the apartment owner's name and, thus, sends a *REQUEST* message to the broker's agent (*Broker*) containing the chosen apartment and waits for the owner's name. The Broker sends back the reply via an *INFORM* message. Afterwards, the Customer starts a negotiation process with the Owner. Meanwhile, the Broker observes the negotiation and in the end asks for a fee via an *INFORM* message, depending on the agreed rent.

1) Negotiation Protocol: A suitable 1-1 negotiation protocol was implemented, based on FIPA ACL-compliant performatives. The protocol encodes the allowed sequences of actions for the automation of the negotiation process among KC-Agents and is represented as a finite state machine with discrete states and transitions, obeyed by all agents. Initially, one agent starts the negotiation by sending a call-for-proposal (CFP) message to the other agent. After several rounds, in which proposals are exchanged, the negotiation ends. This happens when one side either accepts the other side's proposal, or just terminates the negotiation process without any further explanation.

Neither agent is aware of the other's constraints. The agents make alternate bids, with the Owner bidding first; each agent bids only once per round. A bid is represented by bid_i^a, where *a* is the agent offering the bid and *i* is the bidding round. Acceptance/rejection of a bid by agent *a* during round *i* is represented by $ACCEPT_i^a$ / $REJECT_i^a$, respectively. If a rent cannot be agreed on before an agent runs out of time, the negotiation terminates.

2) Negotiation Strategies: Each agent's strategy is designed in line with a particular protocol. There is a plethora of classified strategies, according to different criteria. The two agents involved in this use case adopt strategies that are partially based on [10].

The Customer's strategy increases the bid gradually, depending on its interest for the apartment and is constrained by: (a) *ttr* (time-to-rent): the number of negotiation rounds within which the agent has to rent, (b) *min_profit*: the amount of min. rent decrement, and (c) *interest*: the degree of "how much" the agent wants the apartment (range: 1-10).

The strategy also contains criteria (strategy rules) for decision making that designate whether the agent rejects or accepts the offer. The three main rules are:

- **IF** bid_1^{own} **THEN** $REJECT_2^{cust}$ (the Owner's first offer is always equal to the max. rent and is rejected)
- **IF** $bid_{i-1}^{own} > bid_{i-1}^{cust} + 10*IN$ **THEN** $REJECT_i^{cust}$
- **IF** $bid_{i-1}^{own} < bid_{i-1}^{cust} + 10*IN$ **AND** $bid_{i-1}^{own} < MAX$ **THEN** $ACCEPT_i^{cust}$

where $IN = interest + (i-2)$ and $MAX = bid_1^{own} - min\_profit$. If an offer from the Owner is rejected during round *i*, the Customer has to counter-offer a new bid: $bid_{i+1}^{cust} = 10\% * IN * MAX$. The rules here are represented via a generic format; section 4.2.4 displays the rules, according to the specifications in section 3.1.

The Owner's strategy is constrained by: (a) *tts* (time-to-sell): the number of negotiation rounds within which the agent has to sell, (b) *min_rent*: the minimum rent amount, (c) *start_rent*: the first offer amount, and (d) *L*: a timer parameter, indicating on which round (starting from the end) the Owner's offer starts to rapidly decrease. The Owner is keen to making deals, but, when time is available, it attempts to get a better deal by delaying commitment by one round. Additionally, each new offer has a small discount.

The Owner's decision making rules are:

- **IF** $bid_i^{cust} > min\_rent$ **AND** $bid_{i-1}^{cust} > min\_rent$ **THEN** $ACCEPT_i^{own}$
- **IF** $bid_i^{cust} < min\_rent$ **THEN** $REJECT_i^{own}$

Moreover, the Owner's bids are based on two rules:

- **IF** $(tts > L)$ **THEN** $bid_i^{own} = bid_{i-1}^{own} - i\% * bid_{i-1}^{own}$
- **IF** $(tts < L)$ **THEN** $bid_i^{own} = bid_{i-1}^{own} - 2i\% * bid_{i-1}^{own}$

Finally, the Broker, who observes the negotiation, demands a standard 20% of the final rent as his fee.

3) A Negotiation Example: For the example described here, Table I outlines the agents' constraints, while Table II represents the negotiation step by step.

TABLE I. AGENTS' CONSTRAINTS

Customer	*Owner*
ttr = 5	*tts* = 8
min_profit = 50	*min_rent* = 230
interest = 6	*start_rent* = 350
	L = 3

TABLE II. NEGOTIATION ROUNDS

Round 1 (i=1):
bid_1^{own}=350, *tts*=8, *ttr*=5
Round 2 (i=2):
IN=6+(2–2)=6, $REJECT_2^{cust}$ (reject Owner's first offer)
MAX=350–50=300, bid_2^{cust}=6*10%*300=180, *ttr*=4
bid_2^{cust}=180 < 230 (=*min_rent*), $REJECT_2^{own}$
tts=7, *L*=3, *tts* > *L*, bid_2^{own}=350–2%*350=343
Round 3 (i=3):
IN=7, 180+7*10=250<343, $REJECT_3^{cust}$
bid_3^{cust}=7*10%*300=210, *ttr*=3
bid_3^{cust}=210 < 230, $REJECT_3^{own}$
tts=6, *L*=3, *tts* > *L*, bid_3^{own}=343–3%*343=333
Round 4 (i=4):
IN=8, 210+8*10=290<333, $REJECT_4^{cust}$
bid_4^{cust}=8*10%*300=240, *ttr*=2

bid_4^{cust} =240 > 230 (!), $REJECT_4^{own}$ (delay commitment)
tts=5, *L*=3, *tts* > *L*, bid_4^{own} =333–4%*333=320
Round 5 (i=5):
IN=9, 240+9*10=330>320, 320>300 (=MAX), $REJECT_4^{cust}$
bid_5^{cust} =9*10%*300=270, *ttr*=1
bid_5^{cust} =270 > 230, $ACCEPT_5^{own}$

The negotiation ends successfully after 5 rounds; agreed rent: 270, Broker's fee: 54 (=20%*270).

4) *Agent Specifications:* The rules here refer exclusively to agent communication (subset $C \subseteq S$). The two main communicative-acts are PROPOSE and *ACCEPT-PROPOSAL*. The Customer's description is extended with a fact containing the Owner's name (*owner_name*) and the parameters in the previous section and its communication behavior now contains bid proposal and acceptance:

$F_u^{cust} \equiv \{owner\_name, ttr, min\_profit, interest\}$

$F_e^{cust} \equiv \{agent\_name\}$

$C^{cust} \equiv$ {(*ACLMessage* (*communicative-act PROPOSE*)
(*sender Customer*) (*receiver Owner*)
(*content "bid"*)) ← *propose* (*bid*),
(*ACLMessage*
(*communicative-act ACCEPT-PROPOSAL*)
(*sender Customer*) (*receiver Owner*)
(*content "price"*)) ← *accept* (*price*)}

Similarly, the owner agent's description contains the parameters in the previous section and its communication behavior also contains the proposal and acceptance of bids:

$F_u^{own} \equiv \{customer\_name, tts, min\_rent, start\_rent, L\}$

$F_e^{own} \equiv \{agent\_name\}$

$C^{own} \equiv$ {(*ACLMessage* (*communicative-act PROPOSE*)
(*sender Owner*) (*receiver Customer*)
(*content "bid"*)) ← *propose* (*bid*),
(*ACLMessage*
(*communicative-act ACCEPT-PROPOSAL*)
(*sender Owner*) (*receiver Customer*)
(*content "price"*)) ← *accept* (*price*)}

V. Related Work

DR-BROKERING [11] is a brokering and matchmaking system that represents offerings in RDF and expresses requirements and preferences in a deductive logical language. It features three agent types (Buyer, Seller and Broker). Also, *DR-NEGOTIATE* [8], by the same authors, deploys a negotiation scenario using JADE and DR-DEVICE. Similarly, our approach identifies distinct roles, but also provides a defeasible reasoning service and a reusable knowledge-based agent model that can be used in various scenarios.

Rule Responder [12] builds a service-oriented methodology and a rule-based middleware for interchanging rules in virtual organizations, as well as negotiating about their meaning. It demonstrates the interoperation of distributed platform-specific rule execution environments via Reaction RuleML. Our approach also shares a similar view of reasoning service for IAs and usage of RuleML and allows deploying a variety of rule engines. However, contrary to Rule Responder, our framework is fully FIPA-compliant.

VI. Conclusions and Future Work

This paper argues that agents are vital in realizing the Semantic Web vision and presents a JADE MAS designed for the SW. The system features a defeasible reasoning service implemented as an agent, as well as a reusable agent model that allows creating customizable, knowledge-based agents, equipped with a KB and a Jess rule engine. Via Jess, agents can insert newly derived facts into their KB, altering their behavior accordingly. The paper also presents two diverse use cases that illustrate the technologies presented.

As for future directions, it would be interesting to verify our model's capability to adapt to an even wider variety of scenarios. Another goal is to integrate more reasoning engines, thus, forming a generic environment for cooperating agents in the SW. A final direction could be towards trust, which is essential for making subjective trust judgements about the services provided by other agents [1].

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web", Scientific American, 2001, 284(5):34-43.

[2] J. Hendler, "Agents and the semantic web", IEEE Intelligent Systems, 2001, 16(2):30-37.

[3] D. Nute, "Defeasible reasoning", 20th Int. Conference on Systems Science, IEEE Press, 1987, pp. 470-477.

[4] N. Bassiliades, G. Antoniou, and I. Vlahavas, "A defeasible logic reasoner for the semantic web", IJSWIS, 2006, 2(1):1-41.

[5] K. Kravari, E. Kontopoulos, and N. Bassiliades, "A trusted defeasible reasoning service for brokering agents in the semantic web", IDC'09, unpublished.

[6] R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel, "An intelligent agent for brokering problem-solving knowledge", IWANN (2), 1999, pp. 693-705.

[7] G. Governatori, M. Dumas, A.H.M. ter Hofstede, and P. Oaks, "A formal approach to protocols and strategies for (legal) negotiation", ICAIL'01, pp. 168-177.

[8] T. Skylogiannis, G. Antoniou, N. Bassiliades, G. Governatori, and A. Bikakis, "DR-NEGOTIATE - A system for automated agent negotiation with defeasible logic-based strategies", DKE, 2007, 63(2):362-380.

[9] G. Antoniou and F van Harmelen, A semantic web primer, MIT Press, 2004.

[10] E.Tsang and T. Gosling, "Simple constrained bargaining game", AAMAS-2002, Bologna, Italy, 2002.

[11] G. Antoniou, T. Skylogiannis, A. Bikakis, and N. Bassiliades, "DR-BROKERING – A defeasible logic-based system for semantic brokering", EEE'05, IEEE, 2005, pp. 414-417.

[12] A. Paschke, H. Boley, A. Kozlenkov, and B. Craig, "Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web", 2nd Int. Conf. on Pragmatic Web, ACM, vol. 280, 2007, pp. 17-28.