Expert Systems with Applications 39 (2012) 9571–9587

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

Cross-community interoperation between knowledge-based multi-agent systems: A study on EMERALD and Rule Responder

Kalliopi Kravari [a,*], Nick Bassiliades [a], Harold Boley [b]

[a] Department of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece
[b] Institute for Information Technology, NRC Canada, Fredericton, NB, Canada E3B 9W4

ARTICLE INFO

Keywords:
Semantic Web
Intelligent multi-agent systems
EMERALD
Rule Responder
Cross-Community Collaboration

ABSTRACT

The ultimate vision of the Semantic Web (SW) is to provide users with the capability of delegating complex tasks to intelligent agents. The latter, acting in an interoperable and information-rich Web environment, will efficiently satisfy their users' requests in a variety of real-life applications. Much work has been done on SW information agents for Web-based query answering; a variety of multi-agent platforms and Web language standards has been proposed. However, the platform- and language-bridging interoperation across multi-agent systems has been neglected so far, although it will be vital for large-scale agent deployment and wide-spread adoption of agent technology by human users. This article defines the space of possible interoperation methods for heterogeneous multi-agent systems based on the communication type, namely symmetric or asymmetric, and the MASs status, namely open or closed systems. It presents how heterogeneous multi-agent systems can use one of these methods to interoperate and, eventually, automate collaboration across communities. The method is exemplified with two SW-enabled multi-agent systems, EMERALD and Rule Responder, which assist communities of users based on declarative SW and multi-agent standards such as RDF, OWL, RuleML, and FIPA. This interoperation employs a declarative, knowledge-based approach, which enables information agents to make smart and consistent decisions, relying on high-quality facts and rules. Multi-step interaction use cases between agents from both communities are presented, demonstrating the added value of interoperation.

1. Introduction

Although humans are capable of using the Web to carry out almost any task, machines cannot accomplish similar tasks fully automatically, without human intervention. Thus, the need for a universal medium of data, information and knowledge exchange led to the foundation of the Semantic Web. Broadly, the *Semantic Web* (*SW*) (Berners-Lee, Hendler, & Lassila, 2001) is an evolving extension of the current Web that lets both people and machines fully comprehend relevant information and better satisfy task requests. The ultimate vision of the SW is to provide users with the capability of delegating complex tasks to intelligent agents (Antoniou & van Harmelen, 2004; Hendler, 2001; Shadbolt, Hall, & Berners-Lee, 2006). The latter, acting in the interoperable and information-rich Web environment, will efficiently satisfy their users' requests in a variety of real-life applications. Thus, via the use of intelligent agents, programs become organized to perform tasks more efficiently and with less human intervention. The gradual integration of multi-agent systems (MASs) with SW technology will affect the use of the Web in the future. Going beyond Web crawlers of search engines and other stand-alone information agents, the next generation of Web tools will comprise groups of intercommunicating SW agents traversing the Web.

Various MASs are already available (Park & Sugumaran, 2005; Wang, Wong & Wang, 2012); however, these systems are usually isolated from one another, as their organizational principles and architectures are typically different, and their agents usually do not share the same logic or rule representation formalism. This heterogeneity in representation and reasoning technologies and ensuing MAS disconnectedness comprises a critical drawback in agent interoperation, as agents should share a certain cross-community understanding of each others' perspectives in a dialogue or an argumentation framework. Hence, this work (performed as part of a collaboration between RuleML Inc. and Aristotle University[1]) defines the space of possible interoperation methods based on the communication type and the MASs status, each with their preconditions for applicability. It then chooses and develops one of these methods for cross-community interoperation that enables communication between closed MASs. Specifically, two SW-enabled

* Corresponding author. Tel.: +30 2310998231; fax: +30 2310998433.
E-mail addresses: kkravari@csd.auth.gr (K. Kravari), nbassili@csd.auth.gr (N. Bassiliades), harold.boley@nrc.gc.ca (H. Boley).

[1] Press release: http://ruleml.org/press/EmeraldRuleMLResponderPressRelease-2010-04-23.pdf.

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.02.160

MASs, EMERALD (Kravari, Kontopoulos, & Bassiliades, 2010a) and Rule Responder (Paschke & Boley, 2011), are analyzed in order to discover similarities and differences, and are extended with appropriate bidirectional bridges to make interoperation possible. As a result, a variety of interoperation use cases, such as for bargaining, negotiation and auction, can be implemented. Note that each agent, whether it belongs to EMERALD or Rule Responder, has its own policy, a set of private rules representing its requirements, obligations and constraints, as well as its idiosyncratic knowledge about the world, which makes interoperation a difficult task. Furthermore, since the two MASs have been developed separately, each one uses its own agent/service platforms, communication protocols and languages, as well as knowledge representation formalisms, making interoperation even more difficult.

The aim of this article is to demonstrate how heterogeneous MASs can interoperate in order to automate collaboration across communities using a declarative, knowledge-based method and exemplifying the usefulness of the general method to Cross-Community Collaboration by implementing it for two concrete systems. Such an approach was chosen since knowledge engineering is central to artificial intelligence (AI), and many of the problems that intelligent information agents are expected to solve will require extensive knowledge (Russell & Norvig, 2009). The knowledge-based approach is actually a way of managing and automating knowledge, which enables agents to make smart and consistent decisions, relying on high-quality facts and rules. The focus is on what an agent needs to know in order to behave intelligently, how this knowledge can be represented, and how automated reasoning procedures can make this knowledge available as needed. In this framework, two multi-step interaction use cases among agents are presented, demonstrating the usefulness of interoperating between the above systems.

The rest of this article consists of the following: Section 2 discusses methods for cross-community interoperation. Section 3 presents EMERALD, a multi-agent knowledge-based framework. Section 4 presents Rule Responder, an open source framework for creating virtual organizations as multi-agent systems. Section 5 presents the bidirectional (EMERALD–Rule Responder) bridges (interoperation gateways). Section 6 presents the multi-step interaction scenarios that illustrate the usefulness of the approach. Section 7 discusses related work, and Section 8 concludes with final remarks and directions for future work.

2. Cross-community interoperation methods

Cross-community interoperation between multi-agent systems is vital for agents and human users. However, heterogeneity in representation and reasoning technologies across MASs has not been much researched. Here we present four potential interoperation methods, namely how heterogeneous multi-agent systems can interface each others' agents to automate collaboration across communities. We assume that there are two MASs, with a number of agents acting in them, where the interoperation of three or more MASs could be achieved by incrementally adding a MAS to an already interoperated family of MASs. We then identify four cases defining the space of interoperation methods based on the communication type and the MASs status, as presented in Table 1.

Researchers distinguish two types of two-way communication; symmetric and asymmetric. The term symmetric refers to any MASs in which the communication quantity is the same in both directions, averaged over time. Whereas, the term asymmetric refers to any MASs in which the communication quantity differs in one direction as compared with the other direction, averaged over time. Regarding MASs status, researchers also distinguish two types; open and closed. A MAS is called open when its agents can communicate directly with any agent in any other MAS, whereas it is called closed when its agents are isolated from other systems, communicating with them only through the MAS's proxy or delegate agent.

Table 1
Cross-community interoperation methods.

	(Both) Open	(Both) Closed	Open (k) – Closed (n)
Symmetric	(A) any – any	(B) proxy – proxy	X
Asymmetric	X	(C) delegate – proxy	(D) any – delegate

Both proxy and delegate agents can be found on closed systems. They act as gateways, receiving requests from the rest of the (intra-community) agents and forward them to the other (cross-community) system's appropriate agent. However, they have an important difference; a proxy agent is able to communicate only with a specific agent, whether it is another proxy agent or a delegate agent, whereas a delegate agent is able to communicate directly with any agent in the other system.

Fig. 1 (in conjunction with Table 1) presents, in detail, the four interoperation methods (four cases), that were identified above. In the first case (A), we suppose that both systems are open and, thus, each system's agents understand each other's perspectives, performing a symmetric communication. Thus, each agent is able to directly communicate with any (cross-community) agent in the other MAS without need for a dedicated agent. If this is the case, then either the systems are based on the same technological principles (languages, platforms, etc.) or each intra-community agent has hardwired in its code a variety of representation and reasoning technologies, and thus it is able to understand the cross-community agents' perspectives.

However, usually MASs are based on different technological principles and agents do not have a plethora of representation and reasoning technologies hardwired in their code, because it is impractical. Thus, case (B) presents how two heterogeneous closed MASs can interoperate by interfacing through proxies. Each system provides a proxy agent that is acting as a gateway and is responsible to receive requests from the rest of the intra-community agents and forward them to the cross-community system's corresponding proxy agent. The latter is responsible for forwarding them to its own system's agents and vice versa. This approach does not burden agents with extra representation and reasoning technologies, as proxies are the only ones responsible for the communication between the systems, performing a symmetric communication.

Yet, communication between two closed MASs is not always symmetric. Systems usually differ in terms of representation and reasoning technologies. It would be possible for a closed system to provide a delegate agent; a more flexible and enriched with representation and reasoning technologies agent. Thus, case (C) presents two heterogeneous closed MASs, where the first one provides a delegate agent and the other provides a proxy agent. The delegate agent is able to communicate directly with any cross-community agent, whereas the proxy agent communicates only with the cross-community delegate agent, performing, thus, an asymmetric communication.

Systems are not always based on the same organizational principles. Thus, in case (D), we suppose that one of the systems is open and the other closed. If this is the case, the closed MAS has to provide a delegate agent, as proxy agent is not an option. Proxy agents are not suitable in this case, as they are able to communicate only with other gateways, namely delegate or proxy agents, which actually do not exist in an open MAS. Hence, an open and a closed system are able to communicate only in an asymmetric way.

3. EMERALD: a multi-agent knowledge-based framework

EMERALD (Kravari et al., 2010a) is a multi-agent knowledge-based framework (Fig. 2), built on JADE (Bellifemine, Caire, Poggi,

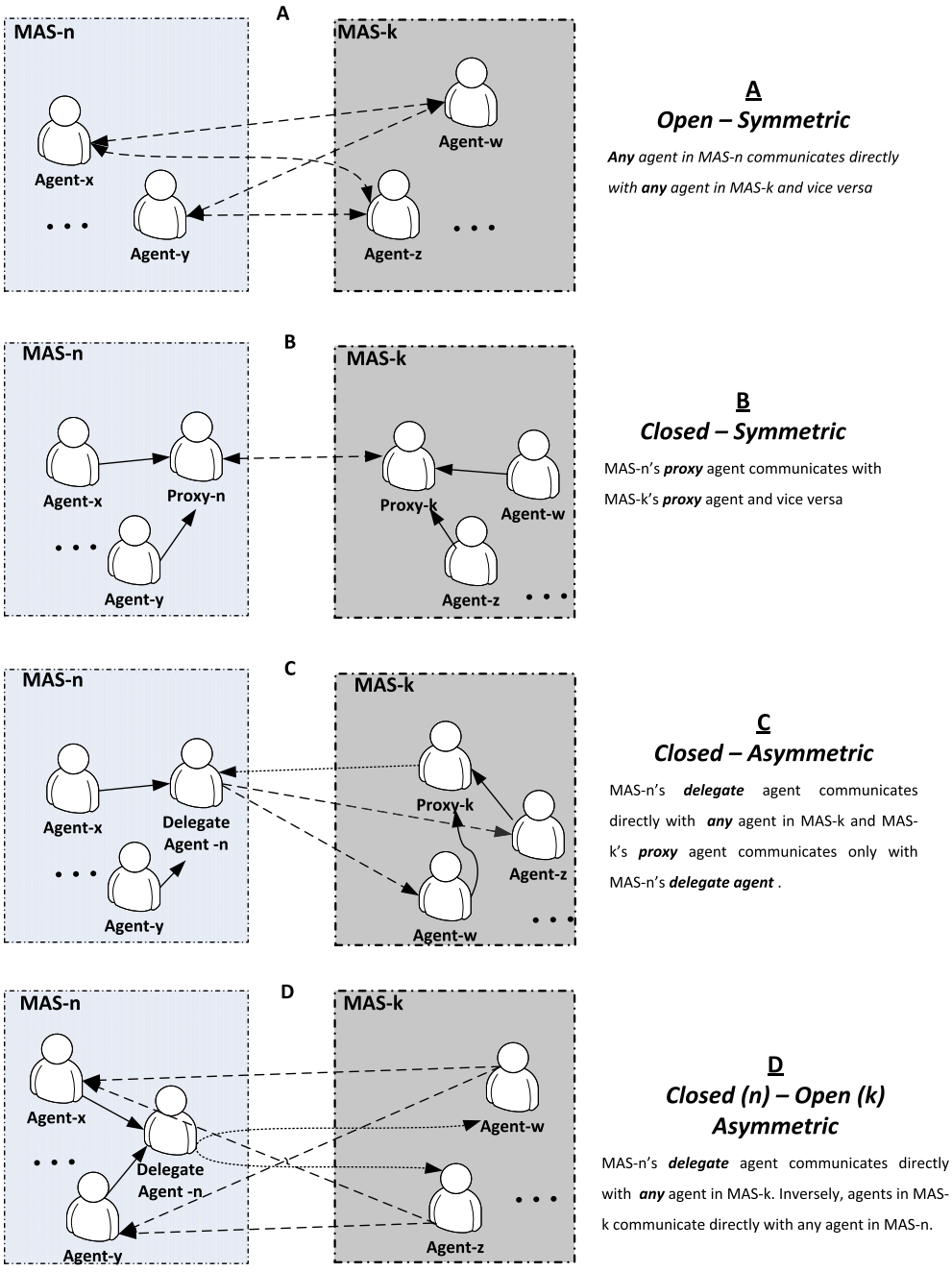

Fig. 1. Methods for cross-community interoperation.

& Rimassa, 2003), which offers flexibility, reusability and interoperability of behavior between agents, based on Semantic Web and FIPA language standards (The Foundation for Intelligent Physical Agents (FIPA): Specifications, 2002). The main advantage of this approach is that it provides a safe, generic, and reusable framework for modeling and monitoring agent communication and agreements. EMERALD supported, so far, the implementation of various applications, like brokering (Antoniou, Skylogiannis, Bikakis, Doerr, & Bassiliades, 2007; Benjamins, Wielinga, Wielemaker, & Fensel, 1999), agent negotiations (Fang & Wong, 2010; Governatori, Dumas, Hofstedeter, & Oaks, 2001; Lin, Chen, & Chu, 2011) and bargaining (Kebriaei & Majd, 2009; Muthoo, 1999; Petit & Magaud, 2006), a single issue negotiation between two parties.

3.1. KC-Agents prototype

In order to model and monitor the parties involved in an agent dialogue, 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*), is deployed (Fig. 3). Agents that comply with this prototype are equipped with a Jess rule engine (JESS, 2008) and a knowledge base (KB) that contains environment knowledge (in the form of facts), behavior patterns and strategies (in the form of Jess production rules). A short description is presented below for better comprehension.

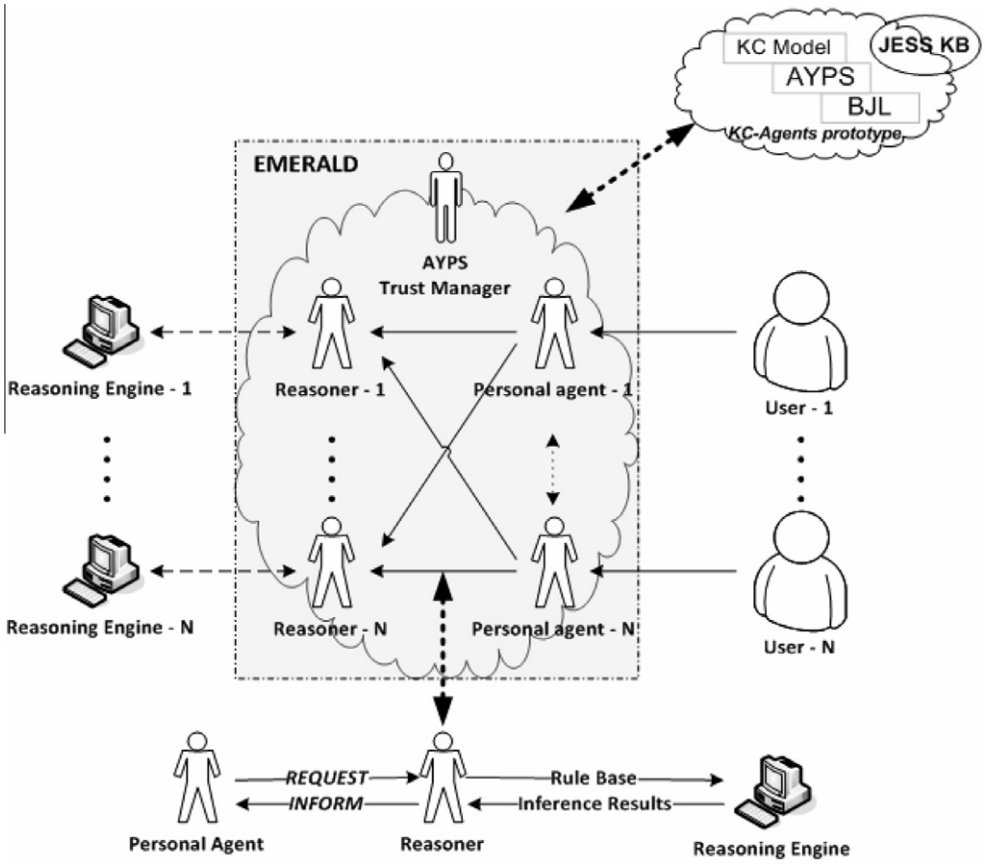

Fig. 2. EMERALD generic overview.

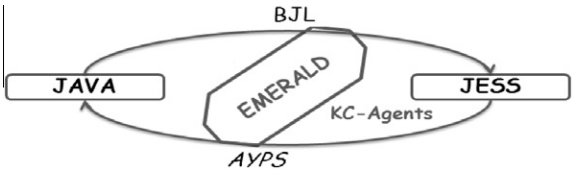

Fig. 3. The KC-Agents prototype.

The generic rule format for describing the agent's behavior is: *result* ← *rule* (*preconditions*). The agent's internal knowledge is a set of facts $F \equiv F^u \cup F^e$, where $F^u \equiv \{fu^1, fu^2, \ldots, fu^k\}$ are user-defined facts and $Fe \equiv \{fe^1, fe, \ldots, fe^m\}$ are environment-asserted facts. The agent's behavior is represented as a set of potential actions rules $P \equiv A \cup S$, where $A \equiv \{a|fe \leftarrow a(fu^1, fu, \ldots, fu^n) \wedge \{fu^1, fu^2, ..., fu^n\} \subseteq F^u \wedge f^e \in F^e\}$ 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, such as agent communication $C \equiv \{c|ACLMessage \leftarrow c(f^1, f^2, \ldots, f^p) \wedge \{f^1, f^2, \ldots, f^p\} \subseteq F\}$ or Java calls $J \equiv \{j|Java \leftarrow j(f\{j|JavaMethod \leftarrow j(f^1, f^2, \ldots, f^q) \wedge \{f^1, f^2, \ldots, f^q\} \subseteq F\}$.

The use of the KC-Agents prototype offers certain advantages, such as modularity, reusability, maintainability and interoperability of behavior between agents, as opposed to having behavior hard-wired into the agent's code.

The advanced yellow pages service (*AYPS*) provided by the *KC-Agents prototype* is a fully automated service for both registered services (services provided and advertised by an agent) and required services (specific services required by an agent). *AYPS* uses a repository to store both services, providing data privacy. Furthermore, AYPS provides a service recovery ability, which groups and sorts the registered (advertised) services according among others their domain and their synonyms, allowing (requesting) agents to make complex queries and receive the best available service. Additionally, it hosts a centralized reputation mechanism which keeps and provides the reputation value of any provider. Concerning the *KC Model*, *AYPS* returns the providers as Jess facts with a designated format: (*service_type* (*provider provider_name*)).

Additionally, as trust has been recognized as a key issue in SW MAS, EMERALD adopts a variety of reputation mechanisms, both decentralized and centralized. Among others, two centralized approaches are provided; a reference system provided by AYPS that keeps the references given from agents interacting with Reasoners or other agents in EMERALD (Kravari et al., 2010a) and a hybrid model based on witness reputation and personal experience (Kravari et al., 2010c). Additionally, a decentralized mechanism, a combination of SPORAS (Zacharia & Maes, 2000) and Certified Reputation (CR) (Huynh, Jennings, & Shadbolt, 2006), is also provided.

3.2. Reasoners

Finally, as agents do not necessarily share a common rule or logic formalism, it is vital for them to find a way to exchange their position arguments seamlessly. To this end, EMERALD proposes the use of Reasoners (Kravari, Kontopoulos, & Bassiliades, 2010b), which are actually agents that offer reasoning services to the rest of the agent community. This approach does not rely on translation between rule formalisms, but on exchanging the results of the reasoning process of the rule base over the input data. The receiving agent uses an external reasoning service to grasp the semantics of the rule base, namely the set of entailments of the knowledge base (Fig. 4). The procedure is straightforward: each Reasoner

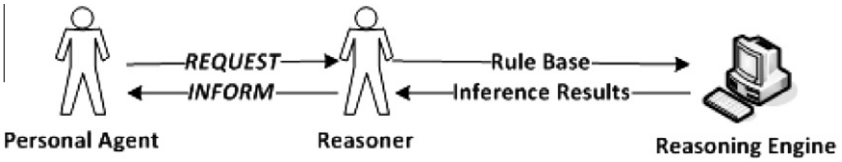

Fig. 4. Input–Output of a Reasoner Agent.

stands by for new requests and as soon as it receives a valid request, it launches the associated reasoning engine and returns the results. Thus, although Reasoners are built as agents, actually they act more like Web services.

EMERALD currently implements a number of Reasoners that offer reasoning services in two major reasoning formalisms: deductive rules and defeasible logic. Deductive reasoning is based on classical logic arguments, where conclusions are proved to be valid, when the premises of the argument (i.e. rule conditions) are true. Defeasible reasoning (Nute, 1987), on the other hand, constitutes a non-monotonic rule-based approach for efficient reasoning with incomplete and inconsistent information. When compared to more mainstream non-monotonic reasoning approaches, the main advantages of defeasible reasoning are enhanced representational capabilities and low computational complexity (Antoniou, Dimaresis, & Governatori, 2009; Maher, 2001). Table 2 displays the main features of the reasoning engines described below.

The two deductive rule reasoners that EMERALD provides are the R-Reasoner and the Prova-Reasoner. More specifically, the R-Reasoner is based on R-DEVICE (Bassiliades & Vlahavas, 2006), a deductive object-oriented knowledge base system for querying and reasoning (using entailments) about RDF metadata (in a Datalog-like fashion). On the other hand, the Prova-Reasoner is based on Prova (Kozlenkov et al., 2006), a Prolog-like rule engine for rule-based Java scripting, integrating Java with derivation rules (for reasoning over ontologies) and reaction rules (for specifying reactive behaviors of distributed agents).

Furthermore, the two defeasible reasoners are the DR-Reasoner and the SPINdle-Reasoner. The SPINdle-Reasoner is based on SPINdle (Lam & Governatori, 2009), an open-source, Java-based defeasible (propositional) logic Reasoner that supports reasoning on both standard and modal defeasible logic. The DR-Reasoner, used in this project, is based on DR-DEVICE (Bassiliades, Antoniou, & Vlahavas, 2006) a forward-chaining, first-order logic defeasible rule engine. 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, entailments generated as results of the forward chaining inference process are exported as an RDF document.

Following the above specifications EMERALD commits to SW and FIPA standards, namely, it uses among others the RuleML language (Boley, Paschke, & Shafiq, 2010) since it has become a de facto standard. In addition, it also uses the RDF model (Resource Description Framework (RDF) Model, 2004) for data representation both for the private data included in agents' internal knowledge and the reasoning results generated during the process, as used in contract agreement interactions presented in Kravari et al. (2010d).

Table 2
Reasoning engine features.

	Type of logic	Implementation
R-DEVICE	Deductive	RDF/CLIPS/RuleML
Prova	Deductive	Prolog/Java
DR-DEVICE	Defeasible	RDF/CLIPS/RuleML
SPINdle	Defeasible	XML/Java
DR-Prolog	Defeasible	RDF/RuleML/DR-Prolog
Proofing Validator	Defeasible	XML/ DR-Prolog
	Order of logic	Reasoning
R-DEVICE	2nd order	Forward chaining
Prova	1st order	Backward chaining
DR-DEVICE	2nd order	Forward chaining
SPINdle	1st order	Forward chaining

4. Rule Responder

Rule Responder (Osmun, Smith, Boley, Paschke, & Zhao, 2011; Paschke & Boley, 2011) is an open source framework for creating virtual organizations as multi-agent systems that support collaborative teams on the Semantic Web. It comes with a number of official instantiations implementing virtual organizations such as SymposiumPlanner for supporting the chairs of the RuleML Symposia[2], presented below.

4.1. Rule Responder MAS

Rule Responder provides the infrastructure for rule-based collaboration between the distributed members of such a virtual organization. Human members are assisted by semi-autonomous rule-based agents, which use SW rules that describe aspects of their owners' derivation and reaction logic. Fig. 5, adapted from Osmun et al. (2011), indicates the general architecture of a typical Rule Responder MAS.

Each Rule Responder instantiation employs four classes of agents, an Organizational Agent (OA), Personal Agents (PAs), External Agents (EAs) and Computing Agents (CAs). The OA represents goals and strategies shared by its virtual organization as a whole, using a global rule base that describes its policies, regulations, opportunities, etc. Each PA assists a person of the organization, (semi-autonomously) acting on his/her behalf by using a local knowledge base of derivation rules defined by the person. Each EA uses a Web (HTTP) interface, accepting queries from users and passing them to the OA. Each CA can be seen as an (often low level) agent that performs an automated (computational) task.

CAs are comparable to PAs, except that they are not paired with a person. Their output is meant to assist the OA in answering the query from the EA. They are designed to perform specific tasks that may involve invoking services independently from the rest of the virtual organization (see Fig. 5).

The OA employs an OWL ontology as a "responsibility assignment matrix" to find the PA that can best handle an incoming query. The OA uses reaction rules to: send the query to this PA, receive its answer(s), do validation(s), and send answer(s) back to the EA.

Rule Responder uses the Enterprise Service Bus (ESB) Mule to transfer data (Fig. 6). Mule performs this data transfer via "data endpoints". In the case of Rule Responder, each agent (EAs/OA/PAs) has their own endpoint through which data will travel. For

[2] SymposiumPlanner: http://ruleml.org/SymposiumPlanner/.

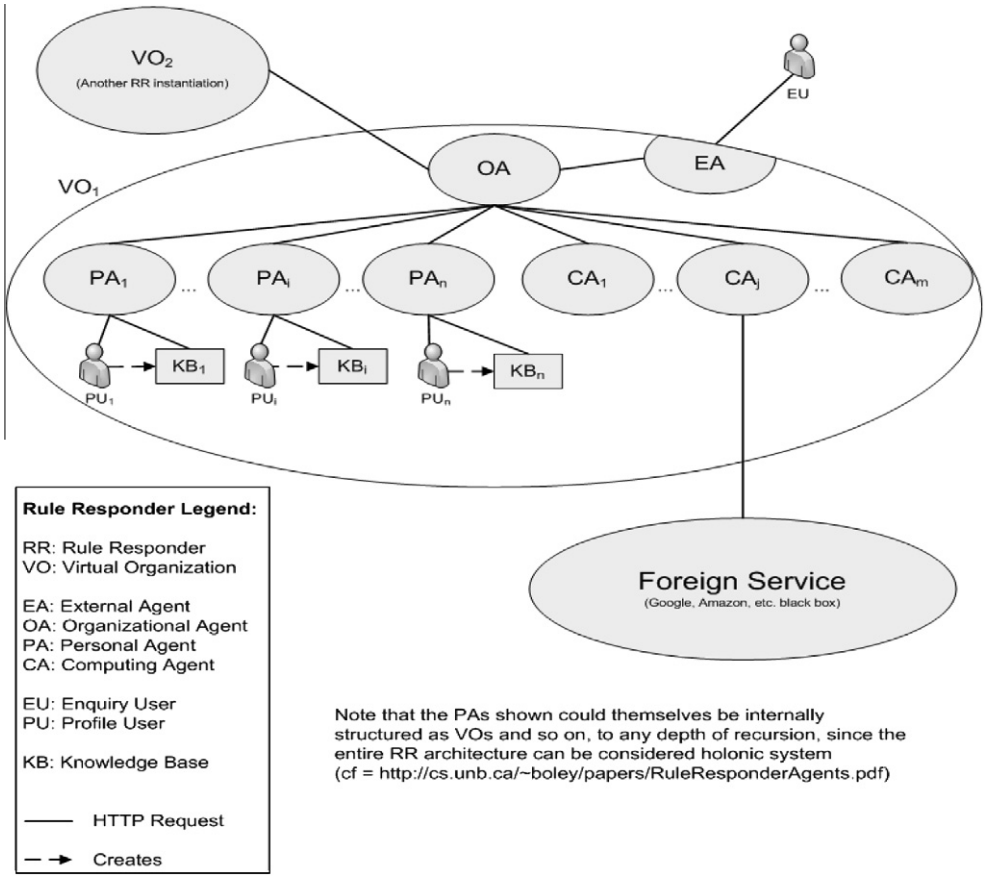

Fig. 5. Rule Responder system architecture considered as holonic system.

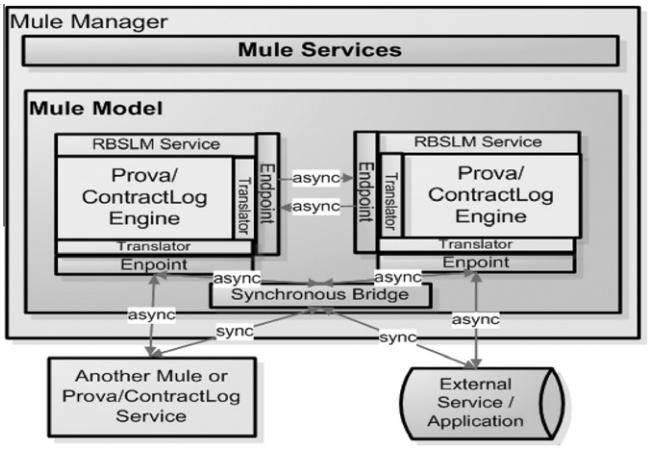

Fig. 6. Mule enterprise service bus.

our cross-community interoperation, Rule Responder only uses HTTP as the transfer protocol, although Mule allows to easily switch over to other ones (e.g, SOAP).

Mule ESB Enterprise (Mule Enterprise Service Bus (ESB), 2007) is the enterprise-class version of the Mule ESB, the most popular open source enterprise service bus. The Mule ESB is a lightweight Java-based enterprise service bus that simplifies the integration of applications and technologies, both on-site and in the cloud. It is lightweight and flexible, adapting to existing infrastructure, yet powerful enough to underpin even large and demanding enterprise SOA implementations.

The supported reasoning engines, with their languages, in Rule Responder are Prova with the Prova language, OO jDREW with POSL and Euler with N3. The present work uses OO jDREW (Ball, Boley, Hirtle, Mei, & Spencer, 2005), a deductive reasoning engine for the RuleML and POSL Web rule languages, written in Java. It is an Object Oriented extension to jDREW that implements Object Oriented features of RuleML, including Order-Sorted Types, Slots and Object Identifiers.

4.2. SymposiumPlanner

SymposiumPlanner (Fig. 7) is a series of multi-agent applications supporting the RuleML Symposium series (e.g. http://2010.ruleml.org) developed with Rule Responder. Based on profiles augmenting facts similar to those in Friend of a Friend[3] with rules, each human (co-)chair position of the RuleML Symposium (general chair, panel chair, etc.) has a Personal Agent acting as a proxy in this virtual organization. Each PA has its own private knowledge base formalizing responsibilities of the position in order to answer queries and solve problems relevant to the chair's role. Queries can range from asking simple chair contact details (e.g. name, email, phone, etc.) to complex problem solving tasks that require deep knowledge acquired by the SymposiumPlanner's knowledge engineers from the human Symposium Chairs.

5. EMERALD–Rule Responder interoperation

In order to develop an interface between EMERALD and Rule Responder (RR) so that they can interoperate, the two systems were

[3] The Friend of a Friend (FOAF) project, http://www.foaf-project.org/original-intro.

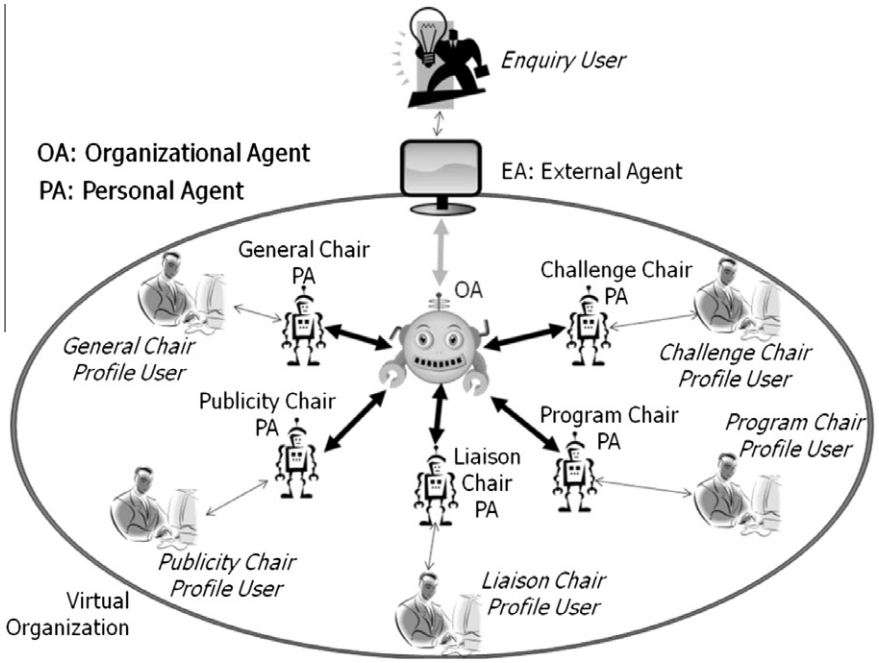

Fig. 7. Rule Responder architecture for the SymposiumPlanner application.

Table 3
Conceptual comparison between EMERALD and Rule Responder.

	Rule Responder	EMERALD
Agent technology	Java servlets/Mule	Java (JADE) agents
Interchange principles	Mule middleware	JADE (ACL)
RuleML	Reaction RuleML	(D)R-DEVICE RuleML
Agent knowledge	Internal rule base Internal & External data-knowledge base	External rule base External data-knowledge base
Reasoning	Multiple reasoning engines and instances of reasoning engines	Multiple reasoning engines (independent external services)
Directory service	NO (instead employs "responsibility assignment matrix")	AYPS
Role of Prova	OAs always written in Prova, PAs and CAs optionally	A Prova Reasoner has been developed (one of the reasoning agents); Prova 3 not yet supported, since it does not support JADE yet
Use	Use cases can be obtained as instantiations of the Rule Responder framework	Use cases can be obtained by using different reasoners and different agent behavior KBs

compared to each other according various issues. Firstly in this subsection, we will present this comparison and its results. Next we will present the two (RuleML) gateways between EMERALD and Rule Responder that were designed and implemented based on this analysis and the previously presented (Section 2) discussion on cross-community interoperation methods.

5.1. EMERALD–Rule Responder comparison

EMERALD and Rule Responder were compared to each other, according to issues regarding their agent-connection topologies, their interchange principles, their used subsets of the RuleML language, the role of the Prova language, and their use. This comparison (Table 3) resulted in several important differences. First, the systems use different technologies; Rule Responder uses the Mule ESB for data transfer and Java servlets for its agents. On the other hand, EMERALD uses JADE and Java agents. Second, they use different RuleML sublanguages; Rule Responder uses Reaction RuleML (Paschke, Kozlenkov, & Boley, 2007) whereas EMERALD uses another RuleML subset, the DR-RuleML sublanguage (Bassiliades et al., 2006).

A third very important difference is that Rule Responder has centralized management through an Organizational Agent (OA) written in Prova (Kozlenkov et al., 2006), while in the de-centralized EMERALD architecture, a Prova reasoner is just one of the supported reasoning services. These reasoning services are, actually, reasoning engines wrapped as independent external services opposed to Rule Responder where reasoning services are instances of reasoning engines.

Another difference is that agents in Rule Responder have internal rule bases but internal and external knowledge bases, whereas agents in EMERALD have both external rule bases and external data-knowledge bases. Furthermore, Rule Responder uses an assignment matrix (a OWL-lite ontology with role assignments), managed by its OA, in order to delegate tasks to agents, opposed to EMERALD where each agent searches via the centralized repository of AYPS to locate appropriate agents to delegate its own tasks.

Finally, Rule Responder's domain specific applications can be obtained as instantiations of the Rule Responder framework; on

the other hand, EMERALD's applications can be obtained by using different reasoners, different agent behavior KBs and different exchanged rule bases.

5.2. EMERALD–Rule Responder gateway architecture

Based both on the above analysis and the previously presented (Section 2) discussion on cross-community interoperation methods, two (RuleML) gateways between EMERALD and Rule Responder were designed and implemented. These two systems are closed and differ in terms of organizational principles and architecture. Thus, we adopted the third method (case C) that enables asymmetric communication between closed systems. EMERALD was extended with a Rule Responder bridge (namely a proxy agent) and, conversely, Rule Responder was extended with an EMERALD bridge (namely a delegate agent) to additional/external agents. The interoperation gateways' architecture is displayed in Fig. 8.

The EMERALD Rule Responder (EMERALD → RR) Gateway was implemented as a new proxy agent in EMERALD, communicating directly with Rule Responder's Organizational Agent (RR OA). The RR Proxy Agent (RRP) is an EMERALD agent, acting as the Rule Responder gateway. This RR Proxy agent is flexible and reusable, thus not hardwired, meaning that it can receive any (RuleML) query, connect to Rule Responder by invoking the OA, which will forward the query to the proper Rule Responder agent, and finally receive the result (through the OA). RR Proxy was developed as a Java (EMERALD) agent class that integrates API methods for interacting both with EMERALD agents and with Rule Responder's OA.

On the other hand, the Rule Responder EMERALD (RR → EMERALD) Gateway was implemented as a new Computing Agent (CA), namely a delegate agent, which handles an appropriate communication channel. In Rule Responder, CAs are implemented as Java servlets, which, in essence, act as wrappers for the corresponding reasoning engines. This CA (the gateway) is called EMERALD Chair and has been developed as a Java servlet class that integrates a) API methods for interacting with EMERALD as well as b) core RR methods for exchanging messages with the Organizational Agent (OA).

The above implementation has enabled collaboration between EMERALD and Rule Responder across their agent communities. The key feature of this approach is the interchange of information between the two systems based on RuleML. RuleML was selected as the data interchange language standard since, as a de facto standard, it is supported by both systems. Additionally, as both systems support knowledge-based technologies it is more straightforward to exchange and transform messages expressed in a rule-based language.

6. Cross-community interoperation use cases

In order to verify and demonstrate the interoperation gateways of Section 4, two use cases involving multi-step dialogues between agents of the two systems were implemented; two Symposium-Planner-2010 queries were selected for illustrating both EMERALD → RR and RR → EMERALD, as presented below.

6.1. EMERALD → RR interoperation use case

An interoperation scenario was selected to demonstrate the EMERALD → Rule Responder Gateway. It concerns an external-to-SymposiumPlanner partner (represented by an EMERALD agent) who would like to sponsor the RuleML-20XY Symposium. Fig. 9 shows all the message exchanges occurring between all the agents involved in this scenario.

In this scenario, the EMERALD agent has to decide whether and to what extent to sponsor the RuleML-20XY Symposium. The decision on the sponsoring level is based on the represented partner's preferences regarding the benefits that each sponsoring level provides. More specifically, the EMERALD agent has a maximum amount of partner money to spend but it does not want just to get whatever benefit is available for this amount ("What do I get for this amount of money?"). Rather, the partner represented by the agent wants to get specific benefits ("If I want these benefits, what amount of money do I have to spend?"), thus the agent must have access to information regarding all the sponsoring levels and their benefits. Such information can be obtained from the corresponding Rule Responder agent, namely the SymposiumPlanner's Publicity Chair PA.

Thus, the EMERALD agent (on behalf of the partner) has to communicate with the PublicityChair in the SymposiumPlanner application. First of all, it sends its query to the Rule Responder Proxy agent which is an EMERALD agent too, acting as a gateway between the two multi-agent systems. This query, presented in Fig. 10, requests the sponsoring levels and their benefits. It is written in Reaction RuleML and corresponds to the "`?getBenefits(Level, Amount, Benefits)`" prolog query.

The RR Proxy agent, on his behalf, forwards this query to the PA of the Publicity Chair (through the RR OA) and waits for the reply. The PA processes the query and returns the available sponsoring levels and their benefits; the corresponding part of the PublicityChair PA's knowledge base is presented in compact d-POSL syntax in Fig. 11. As soon as, it receives the response, the RR Proxy agent returns it back to the partner.

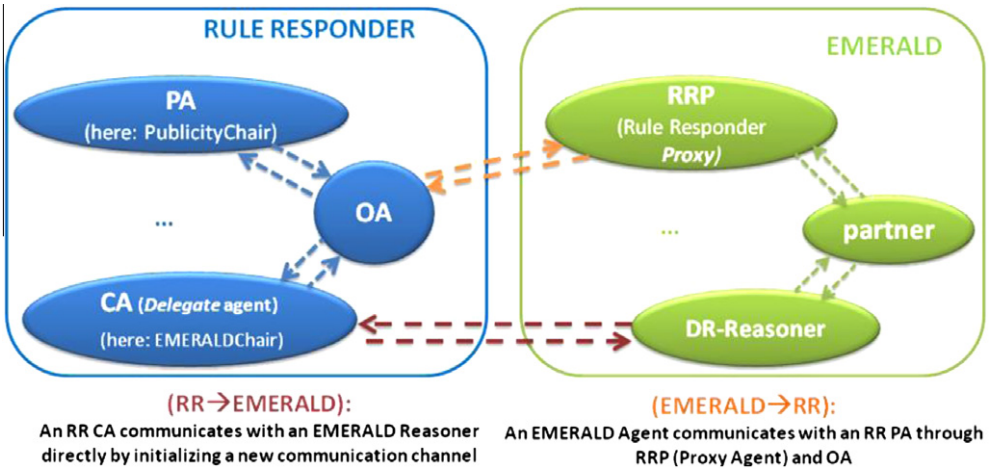

Fig. 8. The interoperation gateways' architecture.

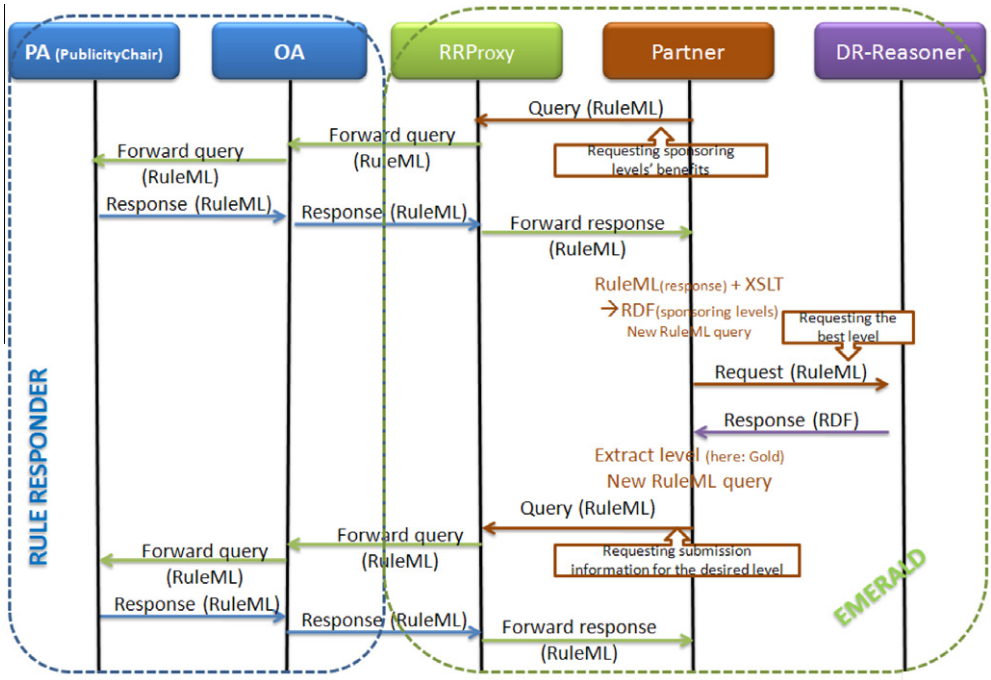

Fig. 9. The EMERALD → RR scenario overview.

```
<RuleML
    xmlns="http://www.ruleml.org/0.91/xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
    http://ibis.in.tum.de/research/
    ReactionRuleML/0.2/rr.xsd"
    xmlns:ruleml2007="http://ibis.in.tum.de/projects/paw#">

    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>RuleML-2010</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind>User</Ind>
        </sender>
        <content>
            <Atom>
                <Rel>getBenefits</Rel>
                <Var>Level</Var>
                <Var>Amount</Var>
                <Var>Benefits</Var>
            </Atom>
        </content>
    </Message>

</RuleML>
```

Fig. 10. The query (in Reaction RuleML) for requesting the sponsoring levels and their benefits.

The decision making of the EMERALD agent is based on rules, and more specifically on defeasible logic rules. This decision making rule base, containing the partner's personal preferences, is presented in compact d-POSL syntax in Fig. 12. These rules indicate that the partner is looking for a sponsoring level that provides at least a free registration and a demo opportunity for the company's products during the Symposium program, without spending more than $ 5000. If there are more than one appropriate sponsoring levels, the cheapest one will be preferred. Part of this rule base is, also, presented in Fig. 13, where the rule about gold sponsoring level is presented in DR-RuleML syntax.

Hence, the EMERALD agent in order to find out which is the cheapest sponsoring level has to apply its rules (the above rule base) to the received message that contains the answers to the query; namely the sponsoring levels and their benefits. The message, however, is formed in a RuleML dialect but according to the functionality of the DR-Device rule engine, which has been presented in Section 2, the (input) data have to be in an RDF format. Hence, the received RuleML message must be transformed to RDF (by the EMERALD agent), in order to be used as facts for the rule base. For this purpose, we developed some domain-dependent XSL transformation rules (Fig. 14) which were used by the EMERALD agent.

At this point, the EMERALD agent possesses both its rule base and the necessary facts; namely the transformed data. Next, it sends its rule base and a link to the data that will be used (the transformed data in RDF format, Fig. 15) to the defeasible logic reasoner (DR-Reasoner), hosted by EMERALD, in order to find the most appropriate sponsoring level.

Then, DR-Reasoner calls the associated reasoning engine (DR-DEVICE) in order to perform inference and provide results. As soon as the inference results are available, DR-Reasoner forwards them to the EMERALD agent. In this case the choice was between the gold and platinum sponsoring level, among the five available levels presented in Table 4, as the other levels were rejected. Both bronze and silver levels were rejected since they did not meet the requirements and the emerald sponsoring level was also rejected since it was too expensive. Thus, the decision was the gold sponsoring level since it was the cheapest available choice.

Afterwards, the EMERALD agent receives back DR-Reasoner's response and sends a new query (Fig. 16) to the PublicityChair (through the RR Proxy agent) requesting the appropriate submission information for that level; e.g. to contact the appropriate chair by e-mail/phone or to wait for his/her call. This query is also written in Reaction RuleML and corresponds to the "`?ask-Info(Level, Action, Info)`" prolog query.

In this case, the Publicity Chair is responsible for the sponsorship (Fig. 17). Its action list determines what action it should do depending on the level of the donation. The decision making part

```
requestSponsoringLevel(?Level, ?Amount, ?Benefits):-
      benefits(?Level, ?Benefits),
      sponsoringLevel(?Rank,?Level, us$[?Amount:integer]).

sponsor_action(?Level, ?Action, ?Info) :-
      actionPerformed(?Action,?Level,?),
      get_info(?Action, ?Info).
```

Fig. 11. Part of PublicityChair PA's knowledge base on sponsoring levels (d-POSL syntax).

```
r1: possibleOffer(level->?x) :=
       sponsorLevel(level->?x).
r2: possibleOffer(level->?x) :=
       sponsorLevel(level->?x, demo->false).
r3: possibleOffer(level->?x) :=
       sponsorLevel(level->?x, amount->?y), ?y>5000.
r4: possibleOffer(level->?x) :=
       sponsorLevel(level->?x, free-registration->?y), ?y<1.
r5: makeOffer(level->?x) :=
       possibleOffer(level->?x),
       sponsorLevel(level->?x, amount->?z),
       \+ (possibleOffer(level->?y), ?y \=?x,
              sponsorLevel(level->?y, amount->?w), ?w<?z).

r2>r1.
r3>r1.
r4>r1.
```

Fig. 12. The decision making rule base in defeasible logic (d-POSL syntax).

```
<Implies ruletype="defeasiblerule">
     <oid><Ind uri="r4">r4</Ind></oid>
     <head>
       <Neg><Atom>
         <op><Rel>possibleOffer</Rel></op>
         <slot><Ind>level</Ind><Var>x</Var></slot>
       </Atom></Neg>
     </head>
     <body>
       <Atom>
         <op><Rel uri="sp:SponsorLevel"/></op>
         <slot><Ind uri="sp:level"/><Var>x</Var></slot>
         <slot><Ind uri="sp:free-registration"/>
          <ComplexArg> <and_ComplexArg>
           <Var>y</Var>
           <Expr><Funin="yes"></Fun>
           <Var>y</Var>
           <Ind>1</Ind></Expr>
          </and_ComplexArg></ComplexArg>
         </slot>
       </Atom>
     </body>
  <superior><Ind uri="r1"/></superior>
</Implies>
```

Fig. 13. Rule about gold sponsoring level in DR-RuleML.

of its knowledge base is presented in compact d-POSL syntax in Fig. 18. According to this, the Publicity Chair should phone the potential partner.

As soon as, the partner (EMERALD agent) receives the response, it is informed that the proper procedure is to wait for a call. Thus, the partner is able to submit a sponsoring request and wait for the

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xmlns:sp="file:///c:/WP3/sponsor-levels.rdfs#" xmlns: . . .
	<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
	<xsl:template match="/">
		<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:sp="file:///c:/WP3/sponsor-levels.rdfs#" xmlns: . . .
			<xsl:apply-templates select="//n:Atom"/>
		</rdf:RDF>
	</xsl:template>
	<xsl:template match="n:Atom">
		<sp:SponsorLevel rdf:about="{concat('file:///c:/WP3/sponsor-levels.rdfs#sp_',n:Ind[1])}">
			<sp:level>
				<xsl:value-of select="n:Ind[1]"/>
			</sp:level>
			<sp:amount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
				<xsl:value-of select="n:Ind[2]"/>
			</sp:amount>
			<xsl:choose>
				<xsl:when test=".//n:Expr[n:Fun='logo' and n:Expr/n:Fun='on' and n:Expr/n:Ind='site']">
					<sp:logo-on-site rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">true</sp:logo-on-site>
				</xsl:when>
				<xsl:otherwise>
					<sp:logo-on-site rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</sp:logo-on-site>
				</xsl:otherwise>
			</xsl:choose>

			 . . .

				<xsl:otherwise>
					<sp:free-registration rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</sp:free-registration>
				</xsl:otherwise>
			</xsl:choose>
		</sp:SponsorLevel>
	</xsl:template>
</xsl:stylesheet>
```

Fig. 14. Part of the domain-dependent XSL transformation rules.

```
<sp:SponsorLevel rdf:about="#sp_gold">
  <sp:level>gold</sp:level>
  <sp:amount rdf:datatype="&xsd;integer">3000</sp:amount>
  <sp:logo-on-site rdf:datatype="&xsd;boolean">true</sp:logo-on-site>
  <sp:ackn-in-proc rdf:datatype="&xsd;boolean">true</sp:ackn-in-proc>
  <sp:sponsor-student rdf:datatype="&xsd;boolean">true</sp:sponsor-student>
  <sp:logo-in-proc rdf:datatype="&xsd;boolean">true</sp:logo-in-proc>
  <sp:demo rdf:datatype="&xsd;boolean">true</sp:demo>
  <sp:advance-publ rdf:datatype="&xsd;boolean">false</sp:advance-publ>
  <sp:brochures-dist rdf:datatype="&xsd;boolean">false</sp:brochures-dist>
  <sp:free-registration rdf:datatype="&xsd;integer">1</sp:free-registration>
</sp:SponsorLevel>
```

Fig. 15. Information about the gold sponsoring level in RDF.

Table 4
The sponsoring levels.

Sponsoring level	Amount	Benefits
Bronze	$500	Logo on website Acknowledgement in proceedings
Silver	$1000	*Bronze level benefits* + Sponsor student participants
Gold	$3000	*Silver level benefits* + Logo in proceedings Show demo 1 free registration
Platinum	$5000	*Gold level benefits* + Name included in all advance publicity Distribution of material to all participants 1 additional free registration
Emerald	$7,500	*Platinum level benefits* + 1 additional free registration

Publicity Chair's phone call or to continue this conversation in order to get any additional information.

6.2. RR → EMERALD interoperation use case

In order to demonstrate the RR → EMERALD interoperation gateway (Fig. 19), we have developed the RuleML-2010 Symposium Planner query called *Suggest Sponsoring Level* [*EMERALD/DR-DEVICE/Publicity* Chair Agent], which is available at the RuleML-2010 Conference website.[4]

In this scenario, a potential partner would like to sponsor the RuleML-2010 Conference, thus he/she has to communicate with the appropriate Personal Agent in the SymposiumPlanner applica-

[4] RuleML 2010: Rule Responder, http://ruleml.org/RuleML-2010/RuleResponder/RuleResponder.htm.

```
<RuleML
    xmlns="http://www.ruleml.org/0.91/xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
    http://ibis.in.tum.de/research/
    ReactionRuleML/0.2/rr.xsd"
    xmlns:ruleml2007="http://ibis.in.tum.de/projects/paw#">

    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>RuleML-2010</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind>User</Ind>
        </sender>
        <content>
            <Atom>
                <Rel>askInfo</Rel>
                <Ind>Level</Ind>
                <Var>Action</Var>
                <Var>Info</Var>
            </Atom>
        </content>
    </Message>

</RuleML>
```

Fig. 16. The query for requesting the appropriate sponsor offer submission procedure.

```
get_info(email, person[?Name,?Email]) :-
       person(symposiumChair[ruleML_2010,publicity],
                     ?Name,?Title,?Email,?Phones).

get_info(phone, person[?Name,?Phones]) :-
       person(symposiumChair[ruleML_2010,publicity],
                     ?Name,?Title,?Email,?Phones).
```

Fig. 17. Part of PublicityChair PA's knowledge base on submission information (d-POSL syntax).

tion. This is actually the PA of the Publicity Chair. The PA is implemented "inside" the Rule Responder multi-agent system, but its personal knowledge is evaluated remotely in the EMERALD system by using the reasoning services of a Reasoner. Thus, in this case, the Publicity Chair is rather a CA (Computing Agent), called EMERALD Chair, because it is independent from the rest of the virtual organization. The potential partner, first of all, has to issue its query, providing the *Sponsor* name and the *Amount* of money ($1400 in this case).This query is written in Reaction RuleML (Fig. 20) and corresponds to the "`?suggestSponsoringLevel(Sponsor, Amount, Level)`" prolog query. Then, he/she has to wait for the suggested sponsoring level, namely the largest level that fits into the offered amount (Fig. 21). According to the sponsoring list presented in Table 4 (Section 5.1), this level is the silver sponsoring level.

The EMERALD Chair CA has no reasoning abilities, as explained, so it has to communicate with an external reasoning engine, such as the DR-DEVICE system (Bassiliades et al., 2006). This reasoning engine resides in the EMERALD system as a Reasoner Agent (DR-Reasoner), so the EMERALD Chair CA needs to communicate with DR-Reasoner. The EMERALD Chair CA, firstly, initializes a new agent called MyGateWayAgent (in the EMERALD environment) that is used as a dispatcher. Then, the EMERALD Chair receives the partner's query (provided via the browser) and transforms it (through an XSLT template – Fig. 22) into an appropriate new RDF document which is required as input data for the DR-Reasoner in EMERALD. Notice that this file is domain-dependent, so in a different use case it must be changed to reflect the classes and properties of the input RDF facts for the defeasible logic rule base.

Then, the EMERALD Chair CA has to initialize the Blackboard object, which is actually a communication channel, in order to initialize the bidirectional communication between Rule Responder and EMERALD. Consequently, the EMERALD Chair sends a new RuleML query and the RDF document to the DR-Reasoner, through the MyGateWayAgent agent. The DR-Reasoner, on the other side, receives the query and conducts the appropriate inference. The conclusions, which are contained actually in an RDF document, consist of the highest level that fits into the offered amount. However, before the results can be returned to the EMERALD Chair, they must be transformed to RuleML, which is the format understood by the EMERALD Chair. The DR-Reasoner transforms the output RDF document via another domain-dependent XSLT document (Fig. 23) and sends the RuleML document with the results (see Fig. 21) back to the EMERALD Chair (again through MyGateWayAgent). The latter displays it to the browser, as an answer to the original query, helping the partner to make the best decision.

7. Related work

In this section we review selected publications related to three main topics; namely related MAS, MAS interoperation architectures and interchange standards.

Regarding the first topic, an architecture for intelligent agents similar to EMERALD is presented in Wang, Purvis, and Nowostawski (2005), where various reasoning engines are employed as plug-in components, while agents intercommunicate via FIPA-based communication protocols. The framework is built on top of the OPAL agent platform (Purvis, Cranefield, Nowostawski, & Carter, 2002) and, similarly to EMERALD, features distinct types of reasoning services that are implemented as reasoner agents. The featured reasoning engines are 3APL (Dastani, van Riemsdijk, & Meyer, 2005), JPRS (Java Procedural Reasoning System) and ROK (Rule-driven Object-oriented Knowledge-based System) (Nowostawski, 2001). 3APL agents incorporate BDI logic elements and first-order logic features, providing constructs for implementing agent beliefs, declarative goals, basic capabilities and reasoning rules, through which an agent's goals can be updated or revised. JPRS agents perform goal-driven procedural reasoning and each JPRS agent is composed of a world model (agent beliefs), a plan library (plans that the agent can use to achieve its goals), a plan executor (reasoning module) and a set of goals. Finally, ROC agents are composed of a working memory, a rule-base (consisting of first-order, forward-chaining production rules) and a conflict set.

Thus, following an approach similar to EMERALD, the framework integrates the three reasoning engines into OPAL in the form of OPAL micro-agents. The primary difference between the two frameworks lies in the variety of reasoning services offered by EMERALD. While the three reasoners featured in Wang et al. (2005) are all based on declarative rule languages, EMERALD proposes a variety of reasoning services, including deductive, defeasible and modal defeasible reasoning, thus, comprising a less integrated, and more open and flexible solution. Furthermore, and most importantly, the approach of Wang et al. (2005) is not based on Semantic Web standards, like EMERALD and Rule Responder, for rule and data interchange.

Regarding MAS interoperation there were several efforts, as the architecture proposed in Georgousopoulos, Rana, and Karageorgos (2003) is similar to our philosophy in terms of system interoperability. However, the authors investigate only FIPA-compliant systems. Their approach proposes FIPA-compliant gateways that could be connected to a legacy MAS to provide automated FIPA interoperability. In their approach a special Java agent (GatewayAgent) was implemented to facilitate the realization of the generic of the FIPA-compliant gateways. However, the proposed

```
%Action list determine what action the Publicity Chair should do depending on the
level of the donation.

checkAction(?Action,?Level,?Amount:integer) :-
   actionPerformed(?Action,?Level,?Amount:integer).

%When a sponsor makes a donation under 500 the Publicity Chair encourage them to
donate more

actionPerformed(?Action:string,preSponsor,?Amount:integer):-
   subtract(?Result:integer,500:integer,?Amount:integer),
     stringConcat(?Action,?Result:integer).
   presponsor(encourage[donate[300]]).

%When a sponsor makes a bronze donation the Publicity Chair should email the
organization

actionPerformed(email,bronze,?Amount:integer).

%When a sponsor makes a silver donation the Publicity Chair should email the
organization

actionPerformed(email,silver,?Amount:integer).

%When a sponsor makes a gold donation the Publicity Chair should phone the
organization

actionPerformed(phone,gold,?Amount:integer).

%When a sponsor makes a platinum donation the Publicity Chair should phone the
organization

actionPerformed(phone,platinum,?Amount:integer).

%When a sponsor makes a emerald donation the Publicity Chair should phone the
organization

actionPerformed(phone,emerald,?Amount:integer).
```

Fig. 18. Part of PublicityChair PA's knowledge base on submission actions (d-POSL syntax).

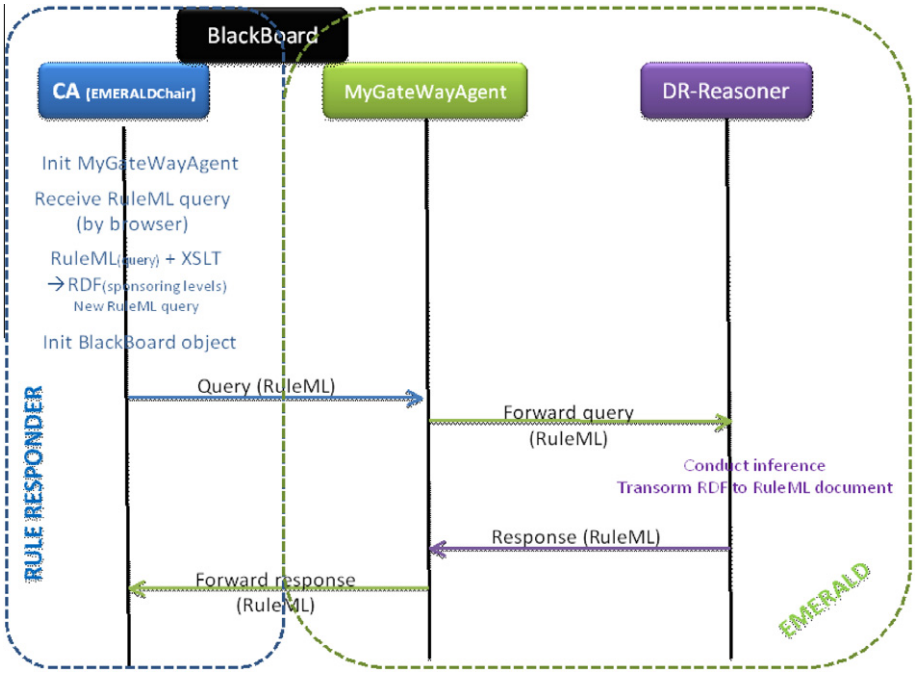

Fig. 19. The RR → EMERALD interoperation scenario overview.

architecture supports just a limited number of performatives (7 out of 22). Due to this limitation this approach is not applicable to systems that require complex interoperable communication i.e. e-commerce or e-market which involve negotiation, co-operation or co-ordination of heterogeneous.

In Armor, Fuentes, and Troya (2003) is presented a component-based approach for interoperability across FIPA-compliant platforms. The authors propose, similar to our philosophy to an extent, the use of proxy agents that work as gateways to communicate agents inside a running agent-based application. Contrary to our

```
<RuleML
    xmlns="http://www.ruleml.org/0.91/xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ruleml.org/0.91/xsd
    http://ibis.in.tum.de/research/
    ReactionRuleML/0.2/rr.xsd"
    xmlns:ruleml2007="http://ibis.in.tum.de/projects/paw#">

    <Message mode="outbound" directive="query-sync">
        <oid>
            <Ind>RuleML-2010</Ind>
        </oid>
        <protocol>
            <Ind>esb</Ind>
        </protocol>
        <sender>
            <Ind>User</Ind>
        </sender>
        <content>
            <Atom>
                <Rel>suggest_sponsoring_level</Rel>
                <Ind>sponsor_1</Ind>
                <Ind>1400</Ind>
                <Var>Level</Var>
            </Atom>
        </content>
    </Message>

</RuleML>
```

Fig. 20. The partner's query (in Reaction RuleML).

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML xmlns="http://www.ruleml.org/0.91/xsd" xmlns:xsi="http://www.w>
http://ibis.in.tum.de/research/ReactionRuleML/0.2/rr.xsd">
                                                              . . .

<Message mode="outbound" directive="answer">
  <oid>
     <Ind>RuleResponder@iitfrdextdev02.iit-iti.priv27</Ind>
  </oid>
  <protocol>
     <Ind>esb</Ind>
  </protocol>
  <sender>
     <Ind>RuleResponder</Ind>
  </sender>
  <content>
     <Atom>
        <Rel>noPublicInterface</Rel>
        <Expr>
           <Fun>interface</Fun>
           <Expr>
              <Fun>suggest_sponsoring_level</Fun>
              <Ind>sponsor_1</Ind>
              <Ind>1400</Ind>
           </Expr>
        </Expr>
     </Atom>
  </content>
</Message>



</RuleML>
```

Fig. 21. The answer to the query in Fig. 20.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
       xmlns:sp="http://155.207.113.24:8080/EMERALDfiles/rrdemo/sponsors-schema.rdf#"
       xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:n="http://www.ruleml.org/0.91/xsd">
       <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
       <xsl:template match="/">
              <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                     xmlns:sp="http://155.207.113.24:8080/EMERALDfiles/rrdemo/sponsors-schema.rdf#"
                     xmlns:sp1="http://155.207.113.24:8080/EMERALDfiles/rrdemo/sponsors_input.rdf#"
                     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
                     <sp:offered_sponsorship rdf:about="http://155.207.113.24:8080/EMERALDfiles/rrdemo/sponsors_input.rdf#sp_1">
                            <sp:sponsor>
                                   <xsl:value-of select="//n:content//n:Ind[1]"/>
                            </sp:sponsor>
                            <sp:amount rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
                                   <xsl:value-of select="//n:content//n:Ind[2]"/>
                            </sp:amount>
                     </sp:offered_sponsorship>
              </rdf:RDF>
       </xsl:template>
</xsl:stylesheet>
```

Fig. 22. The XSLT document that transforms the initial RuleML query to an RDF document.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>
        <xsl:template match="/">
   <RuleML xmlns="http://www.ruleml.org/0.91/xsd"
     <Message mode="outbound" directive="answer">
             <oid>
                     <Ind>RuleResponder@kalliopi-6b34b73</Ind>
             </oid>
             <protocol>
                     <Ind>esb</Ind>
             </protocol>
             <sender>
                     <Ind>ruleml2010_EMERALDChair</Ind>
             </sender>
             <content>
                     <Atom>
                             <Rel>suggested_sponsoring_level</Rel>
                             <Ind>
                                     <xsl:value-of select="//export:sponsor"/>
                             </Ind>
                             <Ind>
                                     <xsl:value-of select="//export:amount"/>
                             </Ind>
                             <Ind>
                                     <xsl:value-of select="//export:level"/>
                             </Ind>
                     </Atom>
             </content>
     </Message>
   </RuleML>
 </xsl:template>
</xsl:stylesheet>
```

Fig. 23. The XSLT document that transforms the output RDF document to a RuleML document.

approach, the basis of their approach is the use of component technology for the development of adaptive software agents. The compositional software agent separates the distribution of messages according to different transport services, which reduces platform dependency on the agent coordinated behavior. However, this approach is limited to FIPA-compliant platforms.

Another work dealing with agent interoperability was presented in Giampapa, Paolucci, and Sycara. (2000)). This work presents an implemented agent, called RETSINA-OAA InterOperator, that allows interoperability across two MASs; RETSINA (Sycara, Decker, Pannu, Williamson, & Zeng, 1996) and OAA (SRI's Open Agent Architecture) (Martin, Cheyer, & Moran, 1999). The authors

describe the issues and challenges regarding the design and implementation of this interoperator. They define the multi-agent system interoperator as an entity that provides agents of one MAS architecture access to the desired capabilities and services offered by another MAS architecture. Our proposal on the other hand deals with heterogeneous systems, FIPA and non-FIPA-compliant, providing guidelines for a general interoperability architecture by using a variety of SW standards based on system gateways.

Regarding interchange, there are several efforts aiming at rule interchange and building a general rule markup and modeling standard for the (Semantic) Web. General standardization approaches, including RuleML (Boley et al., 2010), W3C RIF (Boley & Kifer, 2010), and Common Logic (Common Logic. Common Logic. ISO/IEC 24707:, 2007), have been proposed. However, before the MAS interoperation introduced in this work, no methodological and architectural design and comprehensive implementation has existed that makes the idea of a practical distributed rule layer on the Semantic Web a reality.

8. Conclusions and future work

This article argues that the gradual integration of multi-agent systems (MASs) with SW technology will affect the use of the Web in the future. It presents methods for cross-community interoperation; namely how heterogeneous multi-agent systems can interface each others' agents to automate collaboration across communities. Additionally, this article presents EMERALD and Rule Responder, two semantic multi-agent systems. EMERALD is a fully FIPA-compliant MAS, developed on top of JADE, which uses trusted, independently-developed reasoning services. It constitutes a generic, reusable MAS prototype for knowledge-customizable agents, comprising an agent model, a yellow pages service and several external Java methods. Rule Responder is an open source framework for creating virtual organizations as multi-agent systems that support collaborative teams on the Semantic Web. It has pioneered the use of RuleML as a common MAS interoperation language.

In this article, these two systems were compared and analyzed. Based on the comparison and the potential cross-community interoperation methods, bidirectional gateways between Rule Responder and EMERALD were implemented. This approach provided automated interoperation and collaboration across their agent communities using a declarative, knowledge-based approach, which enables agents to augment their intelligence by making consistent and smart choices. Finally, the article presents multi-step interaction use cases between agents from both communities that illustrate the usability of the framework and demonstrating the added value of interoperation.

More information and related source code about the EMERALD–Rule Responder interoperation project is available at the project's site, available at: http://lpis.csd.auth.gr/systems/EMERALDRR.

In future work, we plan to develop a benchmark suite for bidirectional RuleML-based gateways such as between Rule Responder and EMERALD. We also plan to adapt the RuleML gateways to other interoperation needs, such as interchanging proofs between agents or sharing agent directories. Moreover, we envision the formalization of the interaction scenarios among EMERALD and Rule Responder agents, similarly to Kravari et al. (2010d). Finally, we would like to explore further cross-community agent interoperation needs and provide generalized gateway principles and architectures based on SW standards.

References

Antoniou, G., & van Harmelen, F. (2004). *A Semantic Web primer*. Cambridge, USA: The MIT Press. ISBN13:978-0-262-01210-2.

Antoniou, G., Skylogiannis, T., Bikakis, A., Doerr, M., & Bassiliades, N. (2007). DR-BROKERING: A semantic brokering system. *Knowledge-Based Systems, 20*(1), 61–72.

Antoniou, G., Dimaresis, N., & Governatori, G. (2009). A modal and deontic defeasible reasoning system for modelling policies and multi-agent systems. *Expert Systems with Applications, 36*(2, Part 2), 4125–4134. ISSN 0957-4174.

Armor, M., Fuentes, L., & Troya, J. (2003). A component-based approach for interoperability across FIPA-compliant platforms. In *Proceedings of CIA'2003* (pp. 266–280).

Ball, M., Boley, H., Hirtle, D., Mei, J., & Spencer, B. (2005). The OO jDREW reference implementation of ruleML. In A. Adi, S. Stoutenburg, & S. Tabet (Eds.), *RuleML 2005. LNCS* (Vol. 3791, pp. 218–223). Heidelberg: Springer.

Bassiliades, N., & Vlahavas, I. (2006). R-DEVICE: An object-oriented knowledge base system for RDF metadata. *International Journal on Semantic Web and Information Systems, 2*(2), 24–90.

Bassiliades, N., Antoniou, G., & Vlahavas, I. (2006). A defeasible logic reasoner for the Semantic Web. *IJSWIS, 2*(1), 1–41.

Bellifemine, F., Caire, G., Poggi, A., & Rimassa, G. (2003). JADE: A white paper. *EXP in Search of Innovation, 3*(3), 6–19.

Benjamins, R., Wielinga, B., Wielemaker, J., & Fensel, D. (1999). An intelligent agent for brokering problem-solving knowledge. *IWANN, 2*, 693–705.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American Magazine, 284*(5), 34–43 (Revised 2008).

Boley, H., & Kifer, M. (2010). A guide to the basic logic dialect for rule interchange on the web. *IEEE Transactions on Knowledge and Data Engineering*, 1593–1608.

Boley, H., Paschke, A., & Shafiq, O. (2010). RuleML 1.0: The overarching specification of web rules. *4th International web rule symposium: Research based and industry focused (RuleML'10)* (Vol. 6403). Springer, pp. 162–178.

Dastani, M., van Riemsdijk, M. B., & Meyer, J-J. C. (2005). *Programming multi-agent systems in 3APL. Multi-agent programming: Languages platforms and applications* (Vol. 15). Berlin: Springer, pp. 39–67.

Fang, F., & Wong, T. N. (2010). Applying hybrid case-based reasoning in agent-based negotiations for supply chain management. *Expert Systems with Applications, 37*(12), 8322–8332. ISSN 0957-4174.

The Foundation for Intelligent Physical Agents (FIPA): Specifications (2002). Available from: <http://www.fipa.org/specifications>.

Georgousopoulos, C., Rana, O., & Karageorgos, A. (2003). Supporting FIPA interoperability for legacy multi-agent systems. *LNS, 2935*, 361–379.

Giampapa, J., Paolucci, M., & Sycara K. (2000). Agent interoperation across multiagent system boundaries. In *4th International conference on autonomous agents* (pp 179–186).

Governatori, G., Dumas, M., Hofstedeter, A., & Oaks, P. (2001). A formal approach to protocols and strategies for (legal) negotiation. *ICAIL, 2001*, 168–177.

Hendler, J. (2001). Agents and the Semantic Web. *IEEE Intelligent Systems, 16*(2), 30–37.

Huynh, T. D., Jennings, N. R., & Shadbolt, N. R. (2006). Certified reputation: How an agent can trust a stranger. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS '06)*. USA: ACM, pp. 1217–1224.

Common Logic (2007). ISO/IEC 24707:2007-Information technology—Common Logic (CL): A framework for a family of logic-based languages. Available from: <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39175>.

JESS, the Rule Engine for the Java Platform (2008). Available from: <http://www.jessrules.com/>.

Kebriaei, H., & Majd, V. J. (2009). A simultaneous multi-attribute soft-bargaining design for bilateral contracts. *Expert Systems with Applications, 36*(3, Part 1), 4417–4422. ISSN 0957-4174.

Kozlenkov, A., Penaloza, R., Nigam, V., Royer, L., Dawelbait, G., & Schroeder, M. (2006). Prova: Rule-based java scripting for distributed web applications: A case study in bioinformatics. In *Workshop on reactivity on the web at the international conference on extending database technology (EDBT 2006)*. Springer, pp. 899–908.

Kravari, K., Kontopoulos, E., & Bassiliades, N. (2010a). EMERALD: A multi-agent system for knowledge-based reasoning interoperability in the Semantic Web, 6th Hellenic conference on artificial intelligence (SETN 2010). *LNCS, 6040*(2010), 173–182.

Kravari, K., Kontopoulos, E., & Bassiliades, N. (2010b). Trusted reasoning services for Semantic Web agents. *Informatica: International Journal of Computing and Informatics, 34*(4), 429–440.

Kravari, K., Malliarakis, C., & Bassiliades, N. (2010c). T-REX: A hybrid agent trust model based on witness reputation and personal experience. *Proceeding of 11th international conference on electronic commerce and web technologies (EC-Web 2010). Lecture notes in business information processing, LNBIP* (Vol. 61, Part 3, pp. 107–118). Springer.

Kravari, K., Kastori, G.-E., Bassiliades, N., & Governatori, G. (2010). Contract agreement policy-based workflow methodology for agents interacting in the Semantic Web. In *Semantic Web rules, proceedings 4th international web rule symposium (RuleML 2010), LNCS* (Vol. 6403, pp. 225–239). Springer.

Lam, H., & Governatori, G. (2009). The making of SPINdle. *RuleML-2009 international symposium on rule interchange and applications* (pp. 315–322). Springer.

Lin, C.-C., Chen, S.-C., & Chu, Y.-M. (2011). Automatic price negotiation on the web: An agent-based web application using fuzzy expert system. *Expert Systems with Applications, 38*(5), 5090–5100. ISSN 0957-4174.

Maher, M. J. (2001). Propositional defeasible logic has linear complexity. *Theory and Practice of Logic Programming, 1*(6), 691–711.

Martin, D., Cheyer, A., & Moran, D. (1999). The open agent architecture: A framework for building distributed software systems. *Applied Artificial Intelligence, 13*(1–2), 92–128.

Mule Enterprise Service Bus (ESB) (2007). Available from: <http://www.mulesoft.org> (Update 2011).

Muthoo, A. (1999). *Bargaining theory with applications*. Cambridge: Cambridge University Press.

Nowostawski, M. (2001). *Kea enterprise agents documentation*.

Nute, D., 1987. Defeasible reasoning. In *20th international conference on systems science* (pp. 470–477). IEEE Press.

Osmun, T., Smith, D., Boley, H., Paschke, A., Zhao Z. (2011). Rule Responder Guide. <http://www.ruleml.org/RuleResponder/RuleResponderGuide/>, RuleML Report.

Park, S., & Sugumaran, V. (2005). Designing multi-agent systems: A framework and application. *Expert Systems with Applications, 28*(2), 259–271. ISSN 0957-4174.

Paschke, A., & Boley, H. (in press). Rule Responder – Rule-based semantic agents for the pragmatic web. *International Journal on Artificial Intelligence Tools (IJAIT). Special Issue on Intelligent Distributed Systems*.

Paschke, A., Kozlenkov, A., & Boley, H. (2007). a homogenous reaction rule language for complex event processing. In *2nd International workshop on event drive architecture and event processing systems (EDA-PS 2007)*.

Petit, C., & Magaud, F.-X. (2006). Multiagent meta-model for strategic decision support. *Knowledge-Based Systems, 19*(3), 202–211.

Purvis, M., Cranefield, S., Nowostawski, M., & Carter, D. (2002). *Opal: A multi-level infrastructure for agent-oriented software development. Information Science Discussion Paper Series, number 2002/01*. New Zealand: University of Otago. ISSN 1172-602.

Resource Description Framework (RDF) Model and Syntax Specification (2004). Available from: <http://www.w3.org/TR/PR-rdf-syntax/>.

Russell, S. J., & Norvig, P. (2009). *Artificial intelligence: A modern approach* (3rd ed.). Upper Saddle River, New Jersey: Prentice Hall. ISBN-13: 978-0-13-604259-4.

Shadbolt, N., Hall, W., & Berners-Lee, T. (2006). The Semantic Web revisited. *Intelligent Systems. IEEE, 21*(3), 96–101.

Sycara, K., Decker, K., Pannu, A., Williamson, M., & Zeng, D. (1996). Distributed intelligent agents. *IEEE Expert Systems and their Applications, 11*(6), 36–45.

Wang, M., Purvis, M., & Nowostawski, M. (2005). An internal agent architecture incorporating standard reasoning components and standards-based agent communication. In *IEEE/WIC/ACM international conference on intelligent agent technology (IAT'05), Washington, DC* (pp. 58–64).

Wang, X., Wong, T. N., & Wang, G. (2012). An ontological intelligent agent platform to establish an ecological virtual enterprise. *Expert Systems with Applications, 39*(8), 7050–7061.

Zacharia, G., & Maes, P. (2000). Trust management through reputation mechanisms. *Applied Artificial Intelligence, 14*(9), 881–908.