Personal Agents in the Rule Responder Architecture

Benjamin Larry Craig (ben.craig AT unb DOT ca)
Harold Boley (harold.boley AT nrc-cnrc DOT gc DOT ca)

Faculty of Computer Science, University of New Brunswick
Institute for Information Technology, National Research Council of Canada
Frederiction New Brunswick, Canada

Abstract. Rule Responder is an intelligent rule-based system for collaborative teams and virtual communities that uses RuleML as its knowledge interchange format. This multi-agent infrastructure allows these virtual organizations to collaborate in an automated manner. It is implemented as a Web Service application on top of Mule, an Enterprise Service Bus. It supports rule execution environments (rule/inference engines) such as Prova and OO jDREW. Rule Responder implements an effective methodology and an efficient infrastructure to interchange and reuse knowledge bases (ontologies and rules). The paper describes the design decisions for the personal agent architecture of Rule Responder. A comparison between our distributed rule bases and a centralized rule base is given. An online use case for Rule Responder, applied to the organization of a symposium, is demonstrated.

1 Introduction

Person-centered and organization-centered profile descriptions using Web 3.0 (Semantic Web plus Web 2.0) techniques are becoming increasingly popular. They are often based on the Resource Description Framework (RDF) and include Friend of a Friend (FOAF)[1], Semantically-Interlinked Online Communities (SIOC)[2], and the ExpertFinder Initiative[3]. Recent work on FindXpRT [BLB+] generalized fact-based to rule-based profiles to capture conditional person-centered metadata such as the right phone number to call a person depending on the time, the topic, the caller, or the urgency.

Rule Responder [PBKC07,BP07,Cra07] is a service-oriented middleware tool that can be used by virtual organizations for automated rule-based collaboration [PKB07]. Distributed users (humans or agents) can interact with Rule Responder by query-answer conversations or negotiation and coordination protocols. Rule Responder agents will process events, queries, and requests according to their rule-based decision and behavioral logic. It can also delegate subtasks to other agents, collect partial answers, and send the completed answer(s) back to

[1] http://www.foaf-project.org/
[2] http://www.w3.org/Submission/sioc-spec/
[3] http://wiki.foaf-project.org/ExpertFinder

the requester. The communication protocol used between the architectural components of Rule Responder (e.g., external, personal, and organizational agents) is Reaction RuleML [PKB07]. The Rule Responder Technical Group[PBKC] of RuleML is focused on implementing use cases that require the interchange of rule sets and a querying service. The use cases demonstrate rule-based collaboration in a virtual organization. A virtual organization consists of a community of independent and often distributed (sub)organizations, teams or individual agents that are members of the virtual organization. Typical examples are virtual enterprises, virtual (business) taskforces, working groups, project teams, or resource-sharing collaborations as in, e.g., grid computing or service-oriented computing (SOC).

One specific use case that will be explained in detail throughout the paper is the organization of the RuleML-2008 Symposium. Besides the contribution of an intelligent autonomous agent layer on top of the current Semantic Web, the use case demonstrates rule interchange between rule inference services using a common rule exchange format (RuleML/Reaction RuleML). It also implements a scalable and flexible communication protocol, a service-oriented architecture, and an object broker middleware solution based on enterprise service technologies.

The rest of the paper is structured as follows. Section 2 explains how Rule Responder is implemented as a rule-based multi-agent infrastructure. Section 3 details the real world use case of Rule Responder for organizing a symposium. Section 4 discusses the organizational agent architecture of Rule Responder. Section 5 describes the personal agent architecture. Section 6 compares the differences between distributed rule systems and a centralized rule system. Section 7 concludes the paper. Appendix A contains rule sets for two of our implemented personal agents described in the use case.

2 Rule Responder as a Rule-Based Multi-Agent Infrastructure

Rule Responder's architecture realizes a system of personal agents (PAs) and organizational agents (OAs), accessed by external agents (EAs), on top of an Enterprise Service Bus (ESB) communication middleware. The semi-autonomous PAs and OAs are implemented by (an instance of) a rule engine each, which acts as the inference and execution environment for the rule-based decision and behavioral logic of that semi-autonomous agent. The rule-based PAs represent, as their 'dynamic profiles', all of the participating human members of the virtual organization modeled by Rule Responder. An OA constitutes an intelligent filtering and dispatching system, using a rule engine execution environment for either blocking incoming queries or selectively delegating them to other agents. The communication middleware implements an Enterprise Service Bus (ESB) supporting various transmission protocols (e.g., JMS, HTTP, SOAP). The EAs can interact with the Rule Responder-enabled virtual organization via its public communication interface (e.g., an HTTP endpoint interface to an OA as the "single point of entry"). The current development API of Rule Responder uses a Web browser (Web form) for human-machine communication. A new produc-

tion interface has been created by the Rule Responder Technical Group [PBKC], which will be integrated into the RuleML-2008 Rule Responder webpage. It allows the translation of EA queries from Attempto Controlled English to Reaction RuleML. The ESB implementation for Rule Responder is Mule [BCC+], an efficient open source communication middleware.

Rule Responder blends and tightly combines the ideas of multi-agent systems, distributed rule management systems, as well as service-oriented and event-driven architectures. Rule Responder has the above-discussed three types of agents (PAs, OAs, EAs) which all need to communicate with each other through the Mule ESB. In our current hierarchical use cases, agent-to-agent communication must go through the organizational agent. In particular, when an external agent asks a question to an organization, the external agent does not (need to) know any personal agent such as the one that might ultimately answer the query. Instead, the query must be sent to the organization's OA, which will then delegate it to an appropriate PA. The current Rule Responder use cases do not require direct PA-to-PA communication, but the Rule Responder architecture as described in this paper allows the evolution of our system towards such 'horizontal' (peer-to-peer) communication. The query delegation process and PA-to-PA communication will be described in detail in section 4.2.

While no other rule-based multi-agent ESB system seems to be deployed, Rule Responder could be compared to the Java Agent Development Framework (JADE) [BCR+]. JADE implements an agent-to-agent communication framework, where JADE agents can be distributed over different computers similar to Rule Responder agents. JADE agents can also be made into rule-based agents with the use of the Java-based JESS [FH]. Rule Responder agents can use any rule engine; currently OO jDREW [BC] (Object Oriented java Deductive Reasoning Engine for the Web) and the PROVA [KPS] distributed Semantic Web rule engines are used. Extensions to Rule Responder could allow communication with JADE agents, realized with an XSLT translator from Reaction RuleML to FIFA-ACL. A project similar to JADE is JASON [HBb] which provides a Java-based platform for the development of multi-agent systems. JASON has already created an environment where JASON agents can communicate with JADE agents. [HBa]. We envision that Rule Responder will follow JASON here and interoperate with JADE. This would create interesting communication chains and synergies between several multi-agent infrastructures.

3 RuleML-2008 Use Case Description

One group of use cases created to demonstrate Rule Responder is the organization of meetings such as the RuleML Symposium series, which is an example of a virtual organization that requires online collaboration within a team. Rule Responder started to support the organizing committee of the RuleML-2007 Symposium [Cra07] and was further developed to assist the RuleML-2008 Symposium. The RuleML-2008 use case consists of fully functional knowledge bases for personal agents[4] two of which are partially listed in appendix A. These use

[4] http://www.ruleml.org/RuleML-2008/RuleResponder/index.html

cases strive for embodying responsibility assignment, automating first-level contacts for information regarding the symposium, helping the publicity chair (see appendix A.2 for implementation) with sponsoring correspondence, helping the panel chair with, managing panel participants, and the liaison chair (see appendix A.1 for implementation) with coordinating organization partners. They could also aid with other issues associated with the organization of a meeting, including presentation scheduling, room allocation, and special event planning.

The RuleML-2008 use case utilizes a single organizational agent to handle the filtering and delegation of incoming queries. Each committee chair has a personal agent that acts in a rule-governed manner on behalf of the committee member. Each agent manages personal information, such as a FOAF-like profile containing a layer of facts about the committee member as well as FOAF-extending rules. These rules allow the PA to automatically respond to requests concerning the RuleML-2008 Symposium. Task responsibility for the organization is currently managed through a responsibility matrix, which defines the tasks committee members are responsible for. The matrix and the roles assigned within the virtual organization are defined by an OWL (Ontology Web Language) Lite Ontology. The Pellet [HPSM] reasoner is used to infer subclasses and properties from the ontology.

External agents and the RuleML-2008 agents can communicate by sending messages that transport queries, answers, or complete rule sets through the public interface of the OA (e.g., an EA can use an HTTP port to which `post` and `get` requests can be sent from a Web form). The standard protocol for intra-transport of Reaction RuleML messages between Rule Responder agents is JMS. HTTP SOAP is used for communication with external agents, such as Web services or HTTP clients.

4 Organizational Agent Architecture

Organizational agents are used to describe the organization as a whole; for example, an OA contains a knowledge base that describes the organization's policies, regulations, and opportunities. This knowledge base contains condition/action/event rules as well as derivation rules. An example query that the OA can answer for the RuleML-2008 Symposium is: "Who is the contact responsible for the symposium's panel discussion?" When a RuleML-formalized version of this query is received by the OA, this agent must first determine who the correct contact person is for the panel discussion. When the correct contact person for the panel discussion has been selected, the OA delegates the query to that committee member's personal agent. The PA will then respond with the member's name and contact method (e.g., email or telephone number, depending on contact preferences in their FOAF-like profile). Alternatively, if that contact person was on vacation or currently busy, then the PA would respond back to the OA that the contact person is unavailable. If the first-line contact person cannot be reached, then the OA will use the responsibility matrix (i.e., which committee members are responsible for certain tasks, and what members can fill their role if they are unavailable) to try to contact the next PA. This is one way that Rule Responder can act in an automatic process by chaining subqueries

that find the best contact person at the time the original query is posed. The responsibility matrix is a method that the OA utilizes for query delegation; an in-depth look at query delegation will be presented in section 5.1. This paper does not focus on the OAs of Rule Responder [PBKC07] but rather on its PAs, which will be discussed next.

5 Personal Agent Architecture

The personal agents used by Rule Responder contain FOAF-extending profiles for each person of the organizational team. Beyond FOAF-like facts, person-centric rules are used. All clauses (facts and rules) are serialized in Naf Hornlog RuleML [HBG+], the RuleML sublanguage for Horn logic (allowing complex terms) enriched by Naf (Negation as failure). These FOAF-extending profiles have access to RDF (BibTeX, vCard, iCard, Dublin Core) and RDFS/OWL (role and responsibility models). The RuleML-2008 Symposium use case [PBC] assists each organization committee member by an implemented personal agent. So the panel chair, general chair, publicity chair, etc. each have their own PA. Each PA contains a knowledge base that represents its chair's responsibilities to answer corresponding queries. For example, the query "What benefits would I receive for sponsoring the symposium with 500 dollars as opposed to 1000 dollars" will be delegated to the publicity chair's agent (see appendix A.2) because it deals with sponsoring for the symposium.

5.1 Query Delegation to Personal Agents

Query delegation is done by the organizational agent, but the personal agents can help the OA in this responsibility. Currently, in the RuleML-2008 use case, task responsibility in the symposium organization is managed through a responsibility matrix, which defines the tasks that committee members are responsible for. The matrix, defined by an OWL Lite Ontology, assigns roles to topics within the virtual organization. As an example, query delegation for sponsoring topics is determined by assigning the publicity chair role to the sponsoring topic in the responsiblity matrix. The ontology also defines which committee chairs can fill in for which other ones. For example, if the publicity chair cannot be reached for queries regarding sponsoring, the general chair can step in and answer such queries. Should there be still no unique PA to delegate a query to, the OA needs to make a heuristic delegation decision and send the query to the PA that most likely would be able to answer the query. For example, if a query about media partners was sent to the OA, it could decide to delegate the query to the publicity chair's PA rather than the liaison chair's PA. Only if the publicity chair's PA was unable to answer the query, would the OA then delegate the query to the liaison chair's PA.

The PAs can help the OAs in query delegation by advertising what kind of queries they can answer via FOAF-like metadata in their knowledge bases. This FOAF-like data could tell the OA what kind of queries the agent is able to solve. This would allow the OA to not have to rely on an assignment matrix defined by an ontology. Such an OA would implement an expert finder approach[BP07] where autonomous agents would search for an expert chair that can answer their queries. A single PA might not be able to answer a query as a whole. Another

extension to query delegation would thus be query decomposition, followed by delegation of its decomposed parts to multiple PAs, and finally re-integration of the PAs' answers.

For example, the following rule is currently used to respond to symposium sponsoring queries.

```
sponsor(contact[?Name,?Organization],
        ?Amount:integer,
        results[?Level,?Benefits,?DeadlineResults],
        performative[?Action]) :-
  requestSponsoringLevel(?Amount:integer,?Level),
  requestBenefits(?Level,?Benefits),
  checkDeadline(?DeadlineResults),
  checkAction(?Action,?Level,?Amount:integer).
```

The contact term has two variables, one being the person's name and the other their organization that may want to sponsor the symposium. The amount variable captures how much the organization considers to sponsor. The results term has three variables: for the level of sponsoring (e.g., gold or platinum), the benefits for sponsoring that amount of money, and to find out if the deadline for sponsoring has passed or not. The last term is the performative (action) the publicity chair's agent should take because of the sponsoring query. For this rule to be solved its four premises must be fulfilled.

The first premise will calculate what level the sponsor will receive for donating the amount to the symposium. The second premise will check the benefits that the sponsor will receive. The third premise will determine if the deadline for sponsoring has passed or not. The last premise will conclude what the publicity chair's PA will do in response to the sponsoring query. All of these premises could be solved by the publicity chair's PA. Otherwise, the query would have to be decomposed; for example, in this query the publicity chair's PA may not have the most current date for the deadline for sponsoring or it may not have the newest sponsoring levels or benefits. In order for query decomposition to occur, either the OA has to decompose the query or the PA has to decompose it.

If the OA decomposes the query, it must delegate each premise to a PA. Then the PAs will each solve their premises and send the partial answers back to the OA. The OA will then combine the answers into a single answer. If the PA decomposes the query then the PA would be responsible for finding and delegating the premises to other PAs. The PA would finally collect all of the partial answers and send the complete answer back to the OA. Rule Responder could be extended to return conditional facts (rules) as results of queries. For example, a rule in response to a sponsoring query could be "If sponsoring takes place before the August 31st deadline, then benefits will be obtained".

5.2 Performatives

Reaction RuleML is a general, practical, compact and user-friendly XML-serialized sublanguage of RuleML for the family of reaction rules. It incorporates various kinds of reaction rules as well as (complex) event/action messages into the native

RuleML syntax using a system of step-wise extensions. For Rule Responder we use Reaction RuleML as our interchange language between agents. The following is an example of a communication message that would be delegated to the publicity chair because the content contains a query for sponsoring the symposium.

```
<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-2008</Ind></oid>
    <protocol><Ind>esb</Ind></protocol>
    <sender><Ind>User</Ind></sender>
    <content>
      <Atom>
        <Rel>sponsor</Rel>
        <Expr>
            <Fun>contact</Fun>
            <Ind>Mark</Ind>
            <Ind>JBoss</Ind>
         </Expr>
         <Ind type="integer">500</Ind>
         <Expr>
            <Fun>results</Fun>
            <Var>Level</Var>
            <Var>Benefits</Var>
            <Var>DeadlineResults</Var>
         </Expr>
         <Expr>
            <Fun>performative</Fun>
            <Var>Action</Var>
        </Expr>
      </Atom>
    </content>
  </Message>
</RuleML>
```

The document is encapsulated by `<RuleML>` tags, and the message is contained within `<Message>` tags. The attributes for a message are mode and directive. The mode is the type of a message, either an inbound or an outbound message. The directive is an attribute defining the pragmatic wrapper of the message for example it could be a FIPA-ACL performative. The `<oid>` is the conversation id used to distinguish multiple conversations and conversation states. The `<protocol>` is the transport protocol such as HTTP, JMS, SOAP, Jade, and Enterprise Service Bus (ESB). The `<sender>`/`<receiver>` tags are the sender/receiver of the message. The `<content>` is the message payload transporting a (Reaction) RuleML query, an answer, or a rule base.

The directive attribute corresponds to the pragmatic performative, i.e. the characterization of the message's communication wrapper. External vocabularies defining these performatives could be used by pointing to their conceptual descriptions. The typed logic approach of RuleML enables the integration of external type systems such as Semantic Web ontologies or XML vocabularies. A standard nomenclature of pragmatic performatives is defined by the Knowledge Query Manipulation Language (KQML) and the FIPA Agent Communication Language (ACL). They specify several message exchange/interaction protocols, speech act theory-based performatives, plus content language representations. Other vocabularies such as OWL-QL or the normative concepts of Standard Deontic Logic (SDL) to define, e.g., action obligations or permissions and prohibitions, might be used as well.

5.3 Shared Knowledge Between Personal Agents

Shared knowledge is an important aspect of Rule Responder because sharing rules improves flexibility and reduces redundancy. Rules and facts may be relevant to several PAs. The obvious central location for such shared clauses is the OA, who knows where each PA is located. In the RuleML-2008 use case, personal agents can share such OA-centralized knowledge. For example, the following rule can be shared via the OA to obtain contact information for each chair in the symposium's committee.

```
contactChair(?Meeting, ?Chair, ?FirstName, ?LastName,
             ?Title, ?Email, ?Telephone) :-
  person(
  symposiumChair[?Meeting, ?Chair],
  foafname[firstName[?FirstName],lastName[?LastName]],
  foaftitle[title[?Title]],
  foafmbox[email[?Email]],
  exphones[telephoneNumbers[office[?Telephone],cellPhone[?]]]
```

5.4 Query Answering for Personal Agents

In some cases, the OA can try to solve a query from an external agent by itself, but in the following we consider only cases where it delegates queries to PAs. When a PA receives a query, it is responsible for its answering. If there are multiple solutions to a query, the PA attempts to send an enumeration of as many of the solutions to the OA as possible (it is of course impossible when there are infinitely many solutions). There are different methods for processing multiple solutions to a query. A naive method of the PAs would be to first compute all of the solutions and then send all of the answers back to the OA, one at a time. After the last answer message is sent, an `end-of-transmission` message is sent to let the OA know that there will be no more messages. The main problem with computing all of the answers before sending any of them is obvious: in case of an infinite enumeration of solutions the OA will not receive any answer. The way our implementation addresses the infinite solutions problem is to interleave backtracking with transmission. When a solution is found, the PA immediately sends the answer, and then begins to compute the next solution while the earlier

answer is being transferred. When the OA has received enough answers from such a (possibly infinite) enumeration, it can send a `no-more` message to the PA, stopping its computation of further solutions (this is inverse to the `more/ ;` command in sequential Prologs). Once all solutions have been found in a *finite* interleaved enumeration, the PA can send an `end-of-transmission` message.

If a PA is delegated a query and the agent does not have any solutions for it, a `failure` message is sent right away back to the OA. If this situation or a timeout occurs (i.e., the PA is offline and did not respond back to the OA within the preset time period), then the OA can try to delegate the query to another PA to see if *it* is able to solve the query. If no solution can be found in any of these ways, a `failure` message is sent back to the external agent that states that the OA (representing the entire organization) cannot solve the query.

5.5 Communication Between Personal Agent and Expert Owner

One problem that can arise when a personal agent works on a query is that the PA may require help or confirmation from its human 'owner'. The PA may not be able to (fully) answer the query until it has communicated with its human owner. The way we approach this problem is to allow PAs to send messages to their owners and vice versa, e.g. in the form of emails. When the owner receives a PA email, he or she can respond to help finding the answer to the external agent. The message format will be in a language the PAs can understand; we propose to again use Reaction RuleML, which can also be generated from Attempto Controlled English [Hir06]. When the personal agent has received the answer from its owner, the PA can use it to complete the answer to the original query.

5.6 Agent Communication Protocols

Rule Responder implements different communication protocols which our agents can utilize. The protocols vary by the number of steps involved in the communication. We try to follow message patterns similar to the Web Service Description Language (WSDL) [GLS]. For example, there can be **in-only**, **request-response**, and **request-response-acknowledge** protocols, as well as entire **workflow** protocols. The current Rule Responder use cases primarily focuses on the request-response protocol. The different protocols are explained below with examples.

In-Only: In the **in-only** communication protocol agent_1 sends a message to agent_2, which then executes the performative. For example, the OA sends a performative to a PA to either assert or retract a clause.

Request-Response: The **request-response** communication protocol starts like **in-only**, but agent_2 also sends a response back to agent_1. For example, the OA sends a query to a PA, and the PA solves the query and sends the solution back to the OA.

Request-Response-Acknowledge: The **request-response-acknowledge** communication protocol starts like **request-response**, but agent_1 then sends an acknowledgment message back to agent_2. For example, the OA sends a query to a PA, the PA solves the query and sends the answer back to the OA, and the OA sends an acknowledgement about having received the query's answer to the PA.

Workflows: A **workflow** communication protocol generalizes the above sequential protocols by allowing an arbitrary composition of agent messages from sequential, parallel, split, merge, conditional elements, and loops. For example, the EA sends a query to the OA, the OA decomposes and delegates the subqueries to two PAs, the PAs solve their subqueries sending back the answers to the OA, and the OA integrates the answers and sends the final answer back to the EA.

5.7 Translation Between the Interchange Language and Proprietary Languages

Having an interchange language is a key aspect in a distributed rule system. Each agent must be able to understand one common language that every other agent can interpret. The interchange language carries performatives that each agent is able to understand and react to. In Rule Responder we use Reaction RuleML as our interchange language, which carries RuleML queries, answers, and rule bases in the content part of messages. Agents can understand the content of a Reaction RuleML message by interpreting its performative. Since most of our performatives involve rule engines, we need to translate RuleML to proprietary rule engine languages. Each rule engine can have its own syntax and, in order to adopt a rule engine as a Rule Responder agent, there must exist a translator between RuleML and the execution syntax of that rule engine. For example, there exists an XSLT translator from RuleML to Prova, so that Prova can execute RuleML facts and rules. In the case of OO jDREW, there exists a bi-directional translator between RuleML and POSL (RuleML human-readable, Prolog-like syntax) [Bol04].

6 Comparison of a Distributed Rule System vs. a Centralized Rule System

Rule Responder is implemented as a distributed rule system. It connects OAs and PAs so that they can share knowledge and external agents can query this knowledge. Each PA and OA has its own set of rules and facts. The PA's rules (a FOAF-extending profile) correspond to their expert owner while the OA's knowledge describes the virtual organization as a whole. All of the PAs with their rule bases are stored at distributed locations.

In contrast, a centralized rule system would contain all of the facts and rules in one knowledge base or in one centralized location. So, all of the knowledge would be contained in a single file or database. The advantages of a distributed rule system over a centralized system include the ease of distributed maintenance, achieving a fault-tolerant system by using distribution for redundancy, and improved efficiency through distributed processing.

Distributed maintenance allows agents to update their rules and facts without affecting the rule bases of other agents (their consistency, completeness, etc.). If all of the knowledge was stored in one central rule base, problems introduced by one agent would affect the entire system. Distributed maintenance has proved useful when updating the RuleML-2007 use case to the RuleML-2008 one. The PAs are divided into groups for cross-Atlantic maintenance.

Also, if an agent is not currently running (i.e. the server shut down that the agent is running on), the system does not enter a faulty state because not all known agents are running. If a PA is not responding when an OA delegates a query to it, a timeout would be received and either the OA would try another PA that may be able to answer the query or would respond back that the PA is currently offline. When any part of a centralized rule system causes it to go offline, the entire system will be offline until the centralized system starts back online.

In a distributed rule system the knowledge is spread over many different physical locations and communication overhead can become a problem, but the overhead may not be noticeable to external agents. Rules execute faster when there are less clauses for the engine to process, so our distributed approach improves efficiency because we have multiple rule engines working on smaller knowledge modules instead of one rule engine working on a large knowledge base as in a centralized approach.

7 Conclusion

Rule Responder has been used to implement a number of use cases, including the RuleML-2007/2008 symposium organization and the W3C Health and Life Science (HCLS). The middleware used by Rule Responder allows the simultaneous deployment of these use cases. The ESB provides the communication backbone to synchronously or asynchronously interchange messages between multiple agents. RuleML is a descriptive rule interchange language that so far was able to implement all logical structures that were necessary for Rule Responder. For more information about Rule Responder and use case demos, see [PBKC] (section "Use Cases"). Rule Responder is an open source project and is currently using the open source rule engines Prova [KPS] and OO jDREW [BC]. The Rule Responder Technical Group is currently planning to extend the number of rule engines used by the system. Appendix A contains a condensed version of the personal agent's knowledge bases. For a full listing of the PA implementions readers are referred to the use case home page [PBC].

References

[BC] Marcel Ball and Benjamin Craig. Object Oriented java Deductive Reasoning Engine for the Web. http://www.jdrew.org/oojdrew/.

[BCC+] Antoine Borg, Travis Carlson, Alan Cassar, Andrew Cookeand, Stephen Fenech, and More. Mule. http://mule.codehaus.org/display/MULE/Home.

[BCR+] Fabio Bellifemine, Giovanni Caire, Giovanni Rimassa, Agostino Poggi, Tiziana Trucco, Elisabetta Cortese, Filippo Quarta, Giosu Vitaglione, Nicolas Lhuillier, and Jrme Picault. Java Agent DEvelopment Framework. http://jade.tilab.com/.

[BLB+] Harold Boley, Jie Li, Virendrakumar C. Bhavsar, David Hirtle, and Jing Mei. FindXpRT: Find an eXpert via Rules and Taxonomies. http://www.ruleml.org/usecases/foaf/findxprt.

[Bol04] Harold Boley. POSL: An Integrated Positional-Slotted Language for Semantic Web Knowledge. http://www.ruleml.org/submission/ruleml-shortation.html, May 2004.

[BP07] Harold Boley and Adrian Paschke. Expert querying and redirection with rule responder. In Anna V. Zhdanova, Lyndon J. B. Nixon, Malgorzata Mochol, and John G. Breslin, editors, *FEWS*, volume 290 of *CEUR Workshop Proceedings*, pages 9–22. CEUR-WS.org, 2007.

[Cra07] Benjamin Craig. The OO jDREW Engine of Rule Responder: Naf Hornlog RuleML Query Answering. In Adrian Paschke and Yevgen Biletskiy, editors, *RuleML-2007*, volume 4824 of *Lecture Notes in Computer Science*. Springer, 2007.

[FH] Ernest Friedman-Hill. JESS - The Rule Engine for the Java Platform. http://herzberg.ca.sandia.gov/.

[GLS] Martin Gudgin, Amy Lewis, and Jeffrey Schlimmer. Web Services Description Language (WSDL) Version 1.2 Part 2: Message Patterns. http://www.w3.org/TR/2003/WD-wsdl12-patterns-20030611/.

[HBa] Jomi F. Hbner and Rafael H. Bordini. Interoperation between Jason and JADE Multi-Agent Systems. http://jason.sourceforge.net/mini-tutorial/jason-jade/.

[HBb] Jomi F. Hbner and Rafael H. Bordini. Jason - A Java-based interpreter for an extended version of AgentSpeak. jason.sourceforge.net/.

[HBG+] David Hirtle, Harold Boley, Benjamin Grosof, Michael Kifer, Michael Sintek, Said Tabet, and Gerd Wagner. Naf Hornlog XSD. http://www.ruleml.org/0.91/xsd/nafhornlog.xsd.

[Hir06] David Hirtle. TRANSLATOR: A TRANSlator from LAnguage TO Rules. In *Canadian Symposium on Text Analysis (CaSTA)*, pages 127–139, Fredericton, Canada, October 2006.

[HPSM] James Hendler, Bijan Parsia, Evren Sirin, and More. Pellet: The Open Source OWL DL Reasoner. http://pellet.owldl.com/.

[KPS] Alex Kozlenkov, Adrian Paschke, and Michael Schroeder. Prova: A Language for Rule Based Java Scripting, Information Integration, and Agent Programming. http://www.prova.ws/.

[PBC] Adrian Paschke, Harold Boley, and Ben Craig. RuleML-2008 Use Case. http://www.ruleml.org/RuleML-2008/RuleResponder/index.html.

[PBKC] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule Responder: A RuleML-Based Pragmatic Agent Web for Collaborative Teams and Virtual Organizations. http://www.responder.ruleml.org.

[PBKC07] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web. In *2nd ACM Pragmatic Web Conference 2007*. ACM, 2007.

[PKB07] Adrian Paschke, Alexander Kozlenkov, and Harold Boley. A Homogenous Reaction Rule Language for Complex Event Processing. In *Proc. 2nd International Workshop on Event Drive Architecture and Event Processing Systems (EDA-PS 2007)*. Vienna, Austria, September 2007.

A Rulebases and Queries for Personal Agents

The below POSL listings shorten the OO jDREW-implemented PAs of [PBC].

A.1 Abbreviated Version of the Liaison Chair Agent

```
%%% FOAF-like facts concerning the Liaison Chair
```

```
% One of three Liaison Chairs is shown here
person(
  symposiumChair[RuleML_2008,Liaison],
  foafname[firstName[Mark],lastName[Proctor]],
  foaftitle[title[Dr]],
  foafmbox[email[MarkATemailDOTcom]],
  exphones[TelephoneNumbers[office[4133],cellPhone[5546]]]).

% Facts about partner organizations
partnerOrganization(RuleML_2008, AAAI).
partnerOrganization(RuleML_2008, W3C).
partnerOrganization(RuleML_2008, BPM_Forum).
% ...other partner organizations omitted for space reasons...

% Rules regarding partner organizations
viewOrganizationPartners(?Partner) :-
  partnerOrganization(?Conference, ?Partner).
% ...more rules omitted...

% Facts Regarding Sponsors
sponsor(RuleML_2008, Vulcan_Inc, Gold).
sponsor(RuleML_2008, Model_Systems, Silver).
sponsor(RuleML_2008, STI_Innsbruck, Bronze).
% ...other sponsors omitted for space reasons...

% Rules Regarding Sponsors
viewSponsors(?Conference, ?Company) :-
  sponsor(?Conference, ?Company, ?SponsorLevel)
% ...more rules omitted...
```

A.2 Abbreviated Version of the Publicity Chair Agent

```
%%% FOAF-like facts concerning the Publicity Chair
% One of two Publicity Chairs is shown here
person(
  symposiumChair[RuleML_2008,Publicity],
  foafname[firstName[Tracy],lastName[Bost]],
  foaftitle[title[Dr]],
  foafmbox[email[TracyATemailDOTcom]],
  exphones[TelephoneNumbers[office[0314],cellPhone[1234]]]).

% Main sponsor Rule
sponsor(contact[?Name,?Organization],?Amount:integer,
                results[?Level,?Benefits,?DeadlineResults],
                performative[?Action]) :-
  requestSponsoringLevel(?Amount:integer,?Level),
```

```
  requestBenefits(?Level,?Benefits),
  checkDeadline(?DeadlineResults),
  checkAction(?Action,?Level,?Amount:integer).

% Rule to check chair's action
checkAction(?Action,?Level,?Amount:integer) :-
  ...see online version...

% Rule to check if deadline has passed
checkDeadline(passed[deadline]):-
  date(?X:integer),
  deadline(sponsoring, ?D:integer),
  greaterThan(?X:integer,?D:integer).

% Rule to check if deadline is ongoing
checkDeadline(onGoing[deadline]):-
  date(?X:integer),
  deadline(sponsoring, ?D:integer),
  lessThan(?X:integer,?D:integer).

% Deadline date
deadline(sponsoring,20080830:integer).

% Rules to determine sponsor level
requestSponsoringLevel(?Amount:integer,?Level) :-
  sponsoringLevel(rank0,?Level,
  under[us$[?UnderBronzeAmount:integer]]),
  lessThan(?Amount:integer,?UnderBronzeAmount:integer).
requestSponsoringLevel(?Amount:integer,?Level) :-
  sponsoringLevel(rank1,?Level,us$[?BronzeAmount:integer]),
  greaterThanOrEqual(?Amount:integer,?BronzeAmount:integer),
  sponsoringLevel(rank2,silver,us$[?SilverAmount:integer]),
  lessThan(?Amount:integer,?SilverAmount:integer).

% ...rules omitted for each sponsor level...
requestSponsoringLevel(?Amount:integer,?Level) :-
  sponsoringLevel(rank5, ?Level,us$[?EmeraldAmount:integer]),
  greaterThanOrEqual(?Amount:integer,?EmeraldAmount:integer).

% Facts that determine amount of money for each level
sponsoringLevel(rank0, preSponsor,under[us$[500:integer]]).
sponsoringLevel(rank1, bronze,us$[500:integer]).
sponsoringLevel(rank2, silver, us$[1000:integer]).
sponsoringLevel(rank3, gold, us$[3000:integer]).
sponsoringLevel(rank4, platinum, us$[5000:integer]).
```

```
sponsoringLevel(rank5, emerald, us$[7500:integer]).

% Rule to request benefits
requestBenefits(?Level,?Benefits) :-
  benefits(?Level,?Benefits).
benefits(preSponsor,benefits[none]).
benefits(bronze, benefits[
                  logo[on[site]],
                  acknowledgement[in[proceedings]]]).
% ...facts omitted for each sponsor levels...
benefits(emerald, benefits[
               logo[on[site]],
               acknowledgement[in[proceedings]],
               option[sponsor[student]],
               free[registration,amount[3]],
               logo[in[proceedings]],
               option[demo],
               name[all[advance[publicity]]],
               distribution[brochures[all[participants]]]]).
```

A.3 Selection of Queries for the Personal Agents

The answers are generated online through the EA http interfaces at [PBC].

```
%%% Queries for both chairs
% Query to contact the agents
contactChair(?Conference,?Chair,?FirstName,
                   ?LastName,?Title,?Email,?Telephone)

%%% Queries for Publicity Chair
% Sponsoring query
sponsor(contact[ben,nrc],500:Integer,
  results[?Level,?Benefits,?DeadlineResults],
  performative[?Action])

%%% Queries for Liaison Chair
% Query to view organization partners regardless of event
viewOrganizationPartners(?Partner)

% Query to view sponsors regardless of sponsoring level
viewSponsors(?Conference, ?Company)
```