# Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web

Adrian Paschke
RuleML Inc., Canada
adrian.paschke AT gmx.de

Harold Boley
National Research Council, Canada
Harold.Boley AT nrc.gc.ca

Alexander Kozlenkov
Betfair Ltd., London
alex.kozlenkov AT betfair.com

Benjamin Craig
Univ. of New Brunswick, Canada
Ben.Craig AT unb.ca

## ABSTRACT

The Rule Responder project (responder.ruleml.org) extends the Semantic Web towards a Pragmatic Web infrastructure for collaborative human-computer networks. These allow semi-automated agents – with their individual (semantic and pragmatic) contexts, decisions and actions – to form corporate, not-for-profit, educational, or other virtual teams or virtual organizations. The project develops an effective methodology and an efficient infrastructure to interchange and reuse knowledge (ontologies and rules). Such knowledge plays an important role for (semi-automatically and contextually) transforming data, deriving new conclusions and decisions from existing knowledge, and acting according to changed situations or occurred (complex) events. Ultimately, this might put AI theories on distributed multi-agent systems into larger-scale practice and might form the basis for highly flexible and adaptive Web-based service-oriented/service-component architectures (SOAs/SCAs).

## Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

## General Terms

multi agent systems, representation languages, coordination

## Keywords

Pragmatic Agent Web, Rule Interchange Format, Reaction RuleML, Complex Event Processing, Prova

## 1. INTRODUCTION


*2nd International Conference on the Pragmatic Web* Oct 22-23, 2007, Tilburg, The Netherlands.


The Semantic Web and Web Services have the potential to profoundly change the way people collaborate. The Semantic Web builds upon XML as the common machine-readable syntax to structure content and data, upon RDF [11, 14] as a simple language to express property relationships between arbitrary resources (e.g., objects or topics) identified by URIs, and ontology languages such as RDFS [4] or OWL [16] as a means to define rich vocabularies (ontologies) which are then used to precisely describe resources and their semantics. The adoption of de facto standards such as Dublin Core [7], vCard [5], Bibtex [24] and iCal [6] for metadata descriptions of Web content and the emerging organization/person-centric vocabularies such as FOAF [9] and SIOC [27] and micro-formats such as GRDDL [10] are enabling a more machine-processable and relevant Web. This also prepares an infrastructure to share the relevant knowledge and its meaning between between distributed self-autonomous agents and loosely coupled Web-based services and tools

On top of the syntactic (XML) and semantic (RDF/RDFS, OWL) layer, rules play an important role to automatically and contextually transform data, derive new conclusions and decisions from existing knowledge and behaviorally act according to changed conditions or occurred events. Rules provide a powerful and declarative way to control and reuse the manifold meaning representations published on the Semantic Web. Services and intelligent agents can exploit rules to represent their decisions on how to use knowledge for a particular purpose or goal, including active selection and negotiation about relevant meanings, achievement of tasks, and internal and external reactions on occurred events, changing conditions or new contexts. This extends the Semantic Web to a rule-based *Semantic-Pragmatic Web*[1] which puts the independent micro-ontologies and domain-specific data into a pragmatic context such as communicative situations, organizational norms, purposes or individual goals and values.

In linguistics and semiotics, pragmatics is concerned with the study of how context influences the meaning interpretation of sentences usually in the context of conversations. A distinction is made in pragmatics between sentence mean-

[1]Following [26], we will briefly call this the *Pragmatic Web*, since each of the syntactic-semantic-pragmatic layers is understood to include all the lower layers.

ing and speaker meaning, where the former is the literal meaning of the sentence, while the latter is the piece of information (or proposition) that the speaker is trying to convey. In other words, the Pragmatic Web does not intend to subsume the Semantic Web, but it intends to utilize the Semantic Web with intelligent agents and services that access data and ontologies and make rule-based inferences and autonomous decisions and reaction based on these representations. The focus is on the adequate modelling, negotiation and controlling of the use of the myriad (meta)data and meaning representations of the Semantic Web in a collaborating community of users where the individual meanings as elements of the internal cognitive structures of the members become attuned to each others' view in a communicative process. This allows dealing with issues like ambiguity of information and semantic choices, relevance of information, information overload, information hiding and strategic information selection, as well as positive and negative consequences of actions.

As a result, this Pragmatic Web becomes more usable in, e.g., decision support systems (DSS), heterogenous information systems (HIS) and enterprise application systems (EAS) for distributed human teams and semi-autonomous, agents and IT (web) services: (1) It meaningfully annotates, links, and shares distributed knowledge sources according to common ontologies. (2) It employs rule-based logic for reasoning about source content and metadata. (3) It adds rule-based delegation and integration flow logic to distribute incoming requests towards appropriate virtual (team or organization) members and to collect their responses. By using the Semantic Web as an infrastructure for collaborative networks and by extending it with a rule-based pragmatic and behavioral layer, individuals agents and (Web) services – with their individual contexts, decisions and efforts – can form corporate, not-for-profit, educational, or otherwise productive virtual teams or virtual organizations that have, beside their individual context, a shared context consisting of shared concepts, joint goals and common negotiation and coordination (communication) patterns. Ultimately, this might put the ideas of the AI community on distributed self-autonomous multi agent systems (MAS) into large scale practice and might form the basis for highly flexible and adaptive Web-based service-oriented/service component architectures (SOA/SCA) and event-driven architectures (EDA).

In this paper we contribute with a declarative rule-based service-oriented methodology and a scalable architecture to operationalize such a distributed rule-based approach where event-based communication and rule-based use of meaning plays a central role in connecting the various resources and Web-based services/agents in virtual organizations and teams [20]. The addressed application domain of virtual organizations and rule-based services is of high industrial relevance. We follow a constructivistic design science research methodology [12] and implement an improved rule-based agent technology based on a distributed rule management service and a modern enterprise service middleware providing enhanced usability, scalability and performance, as well as less costly maintenance in engineering and deploying agent/service-oriented architectures. Our Rule Responder system [20] allows to externalize and publish rules on the Web, and to manage them in various modules deployed as online services/agents which are then weaved into the main applications at runtime. In particular, the contributions are as follows:

- Extends the Semantic Web with a pragmatic rule-based layer (Pragmatic Web), which defines the rules for using information resources and ontologies to support human agents in their decisions and react partially self-autonomously by means of automated agents or services
- Blends and tightly combines the ideas of multi-agent systems, distributed rule management systems, and service oriented and event driven architectures
- Addresses real-world software engineering needs for a highly distributed, open, interoperable, efficient and scalable Semantic Web service and agent infrastructure
- Demonstrates the interoperation of various distributed platform-specific rule execution environments based on Reaction RuleML as a platform-independent rule interchange format interchanged over an enterprise service bus as transport middleware
- Applies rule-based technologies to the management of virtual organizations and collaborative teams
- Applies negotiation and distributed coordination mechanisms of rule-based complex event processing and rule-based workflow like reaction rule patterns
- Demonstrates the integration and interoperation of rule standards (RuleML), Object-Oriented programming (Java) and Semantic Web (RDF, RDFS, OWL) and metadata standards (e.g. iCal, vCard, FOAF)

The rest of the paper is organized as follows: In section 2 we propose an extension of the current Semantic Web towards a Pragmatic Agent Web where agents and services practically make use of the data, vocabularies and resources of the Syntactic and Semantic Web. In section 3 we evolve and implement the core concepts and technologies used to make this design artifact of rule-based autonomous agents and rule inference services a reality in industrial real-world settings. In section 4 we demonstrate the applicability of the proposed approach by means of a real-world use case, namely the RuleML-200x symposium organization as a virtual organization. Finally, in section 5 we conclude this paper with a summary of the approach towards a pragmatic agent web and a discussion of the applied research methodology.

## 2. A RULE-BASED PRAGMATIC AGENT WEB MODEL FOR VIRTUAL ORGANIZATIONS

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) cooperations, working groups, project teams or resource sharing collaborations as in e.g. grid computing or service-oriented computing (SOC) where the vision is to build large scale resource / service supply chains (a.k.a. business services networks) which enable enterprises to define and execute Web services based transactions and business processes across multiple business entities and domain boundaries using standardized (Web) protocols.

A virtual organization is typically represented by an organizational agent and a set of associated individual or more specific organizational member agents. The organizational agent might act as a single agent towards other internal and external individual or organizational agents. In other words, a virtual organization's agent can be the single (or main) point of entry for communication with the "outer" world (external agents). Typically, the organizational agent consists of the following:

1. Common syntactic information resources about the virtual organization such as public Web pages showing general contact information, goals and service offerings, but also internal resources such databases or OO representations (e.g. EJBs) to manage customer data, shared project and task data (e.g. calendars) and data about the community members.

2. A semantic layer which describes the common context of the virtual organization such as the shared common concepts and ontologies that evolved during the interaction with the community members and other external agents.

3. A pragmatic and behavioural/decision layer which consists of the organizational norms and values (e.g. deontic norms, needs, avoidance), the joint goals/ interests/ purposes (beliefs/ wants/ desires), the strategies and decision logic (deductive logic and abductive plans), the behavioural reaction logic, and the used negotiation and coordination interchange patterns with the community members but also with external agents.

Similar to an organizational agent, each individual agent is described by its syntactic resources of personal information about the agent, the semantic descriptions that annotate the information resources with metadata and describe the meaning with precise ontologies and a pragmatic behavioural decision layer which defines the rules for using the information resources and ontologies to support human agents in their decisions or react autonomously as automated agents/services. In fact, since each individual agent might be a member of various virtual organizations in which it plays a different role, an individual agent itself might be seen as a "small virtual organization" with shared goals but also with possibly contradicting goals in each of its roles. For instance, a person might be a member of a commercial enterprise and of a research working group with different possibly orthogonal or contradicting goals and norms such as social welfare vs. individual ambitions. If the level of autonomy of decisions is low an agent reduces to a Web service and the virtual organization is implemented by a flexible composition of several services to so called service component architecture (SCAs) which enable distributed application development and integration over the Web. Figure 1 illustrates this general picture.

In this architecture of a *Pragmatic Agent Web* (PAW) model the *syntactic level* controls the appearance and access of syntactic information resources such as HTML Web pages. The formal nature of representation languages such as XML, RDF and OWL on the *semantic level* make these Web-based information more readable and processable not only to humans, but also to computers, e.g., to collect machine-readable data from diverse sources, process it and infer new knowledge. Finally, the *pragmatic level* defines the rules how information is used and describes the actions in terms of its pragmatic aspects, i.e. why, when and for what purpose or with what goals they are done. These rules e.g. transform existing information into relevant information of practical consequences, trigger automated reactions according to occurred complex events/situations, and derive answers to queries from the existing syntactic and semantic information resources.

In this paper we focus on the pragmatic and behavioural layer which makes use of the meaningful domain data and metadata knowledge from the syntactic and semantic layer and transforms the existing information into relevant information which is accessible by Web-based service interfaces. Declarative rules play an important role to represent the conditional decision and behavioural logic of the agents as well as the strategic and pragmatic contexts in which collaboration takes place such as communicative and coordination situations, beliefs, wants, needs and avoidances, individual values, organizational norms etc. This also includes (semi-)automated negotiation and discussion about the meaning of ontological concepts, since agents might use their own micro-ontologies and must agree on relevant shared concepts to enable an efficient communication and knowledge interchange between the nodes. Modularization and information hiding is another important concept for a virtual collaboration of independent agents, since each agents might have its own goals, strategies and rich tacit meaning of ontological concepts that should not or cannot be made explicit. That is, a certain level of ambiguity and hidden information should be allowed, as long as they do not endanger the higher goals and the communication of the virtual organization. Communication within the collaborative community and with external agents based on an adequate "webized" interchange format for rule sets, queries and derived answers but also for communicative, pragmatic and ontological semantic contexts is needed.

Our agent and service-oriented approach which evolves from former multi agent technologies and novel enterprise service architectures enables to naturally capture more complex constraints on what Web-based services are willing to offer and how they can be combined and collaborate in virtual organizations respectively enterprise business networks. Agents are self-autonomous, distributed, loosely-coupled, long-lived, persistent computational entities that can perceive, reason, act and communicate [13]. Depending on their behavioural and decision constraints/logic which is typically rule-based and their ongoing interactions they act with varying levels of autonomy. Because of their autonomy and heterogeneity agents are not specific to a particular underlying knowledge representation or programming paradigm and there are various possibilities to implement the rule-based logic, e.g., if-then constructs in procedural programming languages such as Java or C/C++ (with control flow), decision tables/trees, truth-functional constructs based on material implication, implications with constraints (e.g., OCL), triggers and effectors (e.g., SQL trigger), non-logical approaches such as semantic networks, frames or logical knowledge representation (KR) approaches based on subsets of first order predicate logic such as logic programming (LP) techniques. In this paper we employ a declarative logic-based approach which has several advantages: reasoning with rules is based on a semantics of formal logic, usually

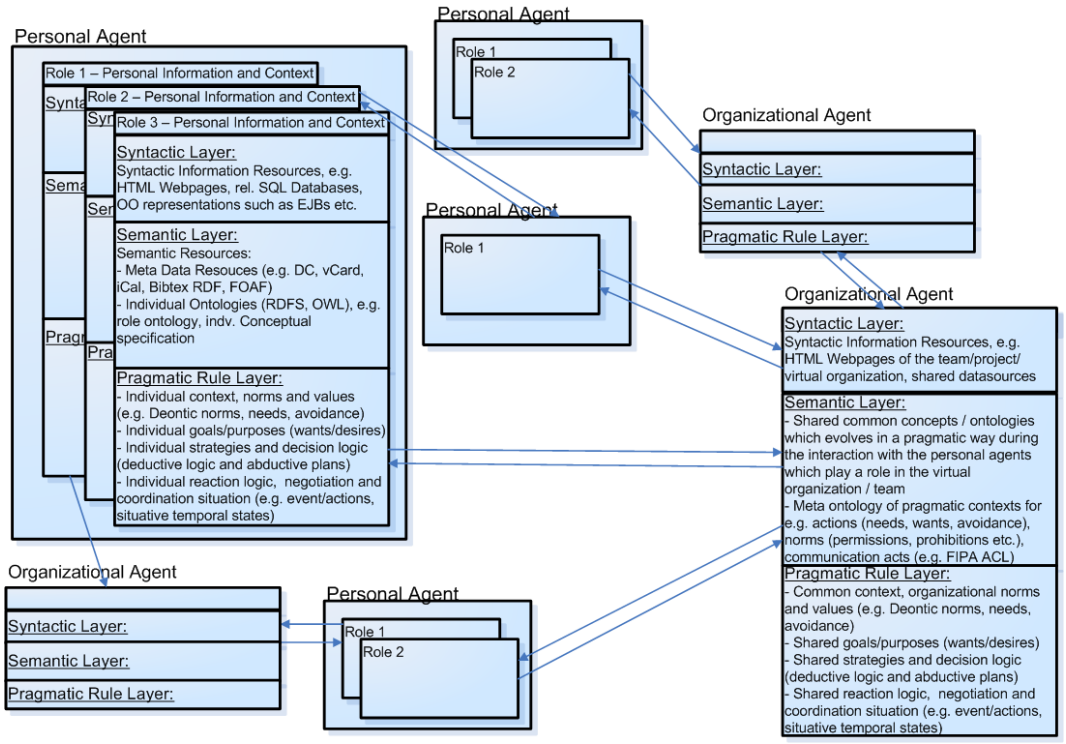


**Figure 1: A Pragmatic Agent Web for Virtual Organizations**

a variation of first order predicate logic which also underpins Semantic Web ontology languages, and it is relatively easy for the end user to write rules. The basic idea is that users/agents employ rules to express *what* they want, the responsibility to interpret this and to decide on *how* to do it is delegated to an interpreter (e.g., an inference/rule engine or a just in time rule compiler). Traditionally, rule-based systems have been supported by two types of inferencing algorithms: forward-chaining as e.g. in production rule systems and backward-chaining as in logic programming systems such as Prolog derivatives. We are not specific which particular rule language and rule engine (execution environment) is used on the platform dependent layer since this technical layer is wrapped by our rule management middleware which provides general message-oriented communication interfaces using arbitrary transport protocols such as HTTP, JMS, SOAP, ... and translation services into a standardized platform-independent rule interchange format in order to enable interaction with other agents and services which implement different rule execution environments.

We build the rule-based pragmatic agent layer upon existing technologies and common language formats of the Semantic Web such as HTML/XML Web pages, RDF/RDFS and OWL variants of de facto standards such as Dublin Core, vCard, iCal or BibTeX/BibTeXML and other emerging vocabularies such as FOAF or SIOC, which are used to describe personal and institutional metadata and information, project and event data as well as ontological conceptualizations of the individual and common domains/vocabularies. We assume that there is already a critical mass of such data sources on the semantic and syntactic layer, e.g. RDF Bibtext libraries of publications, RDF vCard or FOAF profiles for each member and role, online event calendars using vCal or gData feeds. Furthermore, we integrate data and functionality from legacy applications such as rel. databases, enterprise applications or Web services into the rule-based decision and execution logic. Depending on the particular rule execution environment the integration can happen dynamically at runtime or by pre-transformation and replication of the external data into an internal executable format (e.g. a set of logical facts replicated in the internal knowledge base).

This general approach towards a rule-based PAW model includes a great variety of technical design science and Software Engineering decisions, such as how to access the various external data sources and ontologies (e.g. homogenous translation and integration vs. heterogeneous integration), how to manage and maintain the rule modules on the various levels (e.g. distributed scoped knowledge based vs. centralized knowledge base in central organizational agent node), how to integrate and interoperate with various execution environments (e.g. various rule engines with various logical expressiveness classes), how to communicate and negotiate semantics and pragmatic meaning, how to deal with complex events and situations, what is a scalable approach to operationalize and communicate between the agent nodes (e.g. enterprise service bus vs. ad-hoc communication e.g. via SOAP or JMS messages). Figure 2 exemplifies these technical design and implementation questions of a PAW model. In the next section we will detail the main technical components of this architecture.

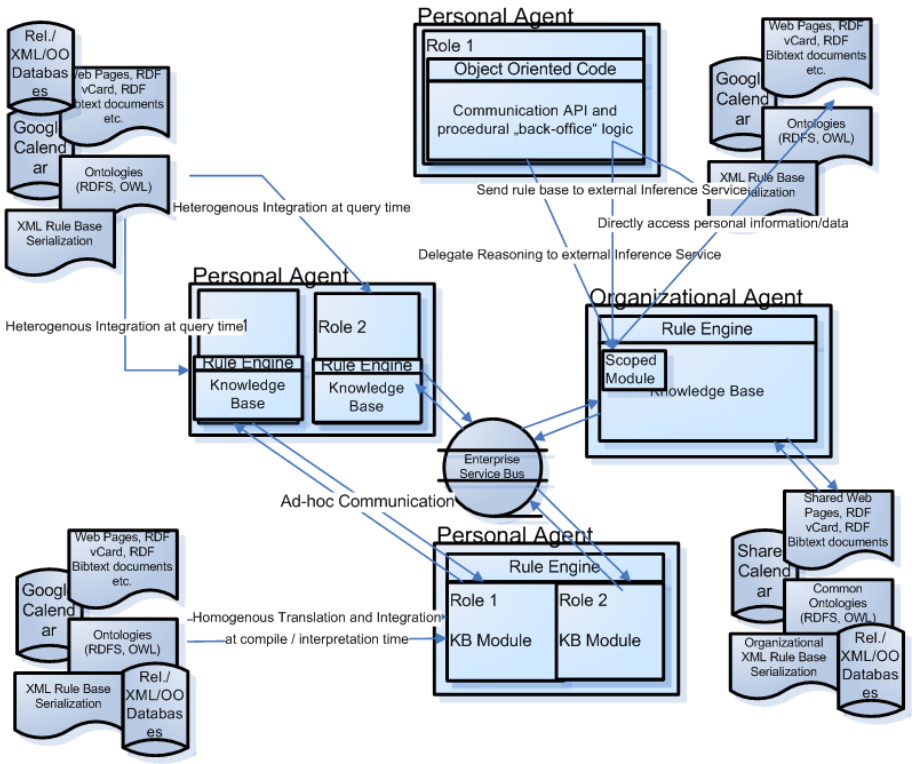


**Figure 2: Rule-based Pragmatic Agent Web Architecture**

## 3. DISTRIBUTED RULE RESPONDER AGENT SERVICES

In this section we will introduce the main components of the distributed Rule Responder architecture for a Pragmatic Agent Web [20]. The three core parts are (1) a common platform-independent rule interchange format to interchange rules and events between arbitrary agent services, (2) a highly scalable and efficient agent/service-broker and communication middleware, and (3) platform-specific rule engines and execution environments.

### 3.1 RuleML as Platform-Independent Rule Interchange Format

*The Rule Markup Language (RuleML).*

[2] is a modular, interchangeable rule specification standard to express both forward (bottom-up) and backward (top-down) rules for deduction, reaction, rewriting, and further inferential-transformational tasks. It is defined by the Rule Markup Initiative [3], an open network of individuals and groups from both industry and academia that was formed to develop a canonical Web language for rules using XML markup and transformations from and to other rule standards/systems. The language family of RuleML covers the entire rule spectrum, from derivation rules to reaction rules including rule-based complex event processing (CEP) and messaging (Reaction RuleML [23]), as well as verification and transformation rules. In the following we will briefly summarize the key components of RuleML language (Horn logic layer of RuleML) and then introduce the Reaction RuleML language [23, 22] which extends RuleML with additional language constructs for representing reaction rules and complex event / action messages, e.g. for complex event processing. The building blocks of RuleML are: [2]

- Predicates (atoms) are n-ary relations defined as an $< Atom >$ element in RuleML. The main terms within an atom are variables $< Var >$ to be instantiated by ground values when the rules are applied, individual constants $< Ind >$, data values $< Data >$ and complex expressions $< Expr >$.
- Derivation Rules ($< Implies >$) consist of a body part ($< body >$) with one or more conditions (atoms) connected via $< And >$ or $< Or >$ and possibly negated by $< Neg >$ which represents classical negation or $< Naf >$ which represents negation as failure and a conclusion ($< head >$) which is derived from existing other rules or facts applied in a forward or backward manner.
- Facts are deemed to be always true and are stated as atoms: $< Atom >$
- Queries $< Queries >$ can either be proven backward as top-down goals or forward via bottom-up processing. Several goals might be connected within a query and negated.

Besides facts, derivation rules and queries, RuleML defines further rule types such as integrity constraints and transformation rules [2].

### *Reaction RuleML.*

[23, 22] is a general, practical, compact and user-friendly XML-serialized sublanguage of RuleML for the family of reaction rules. It incorporates various kinds of production, action, reaction, and KR temporal/event/action logic rules as well as (complex) event/action messages into the native RuleML syntax using a system of step-wise extensions. The building blocks of Reaction RuleML (version 0.2) are: [23]

- One general (reaction) rule form ($< Rule >$) that can be specialized to e.g. production rules, trigger rules, ECA rules, messaging rules ...
- Three execution styles defined by the attribute @*style*
  - *Active*: 'actively' polls/detects occurred events in global ECA style, e.g. by a ping on a service/system or a query on an internal or external event database
  - *Messaging*: waits for incoming complex event message and sends outbound messages as actions
  - *Reasoning*: Knowledge representation derivation and event/action logic reasoning and transitions (as e.g. in Event Calculus, Situation Calculus, TAL formalizations)
- Messages $< Message >$ define inbound or outbound event message

A reaction rule might apply globally as, e.g. global ECA rules or locally nested within other reaction or derivation rules as e.g. in the case of messaging reaction rules (e.g. complex event processing rules). The general syntax of a reaction rules consists of six partially optional parts:

```
<Rule style="active" evaluation="strong">
   <label> <!-- metadata  --> </label>
   <scope> <!-- scope --> </scope>
   <qualification> <!-- qualifications --> </qualification>
   <oid> <!-- object identifier --> </oid>
   <on>  <!-- event -->  </on>
   <if>  <!-- condition --> </if>
   <then> <!-- conclusion --> </then>
   <do>  <!--  action -->  </do>
   <after> <!-- postcondition --> </after>
   <else> <!-- else conclusion --> </else>
   <elseDo> <!-- else/alternative action --> </elseDo>
   <elseAfter> <!-- else postcondition --> </elseAfter>
</Rule>
```

Inbound and outbound messages $< Message >$ are used to interchange events (e.g. queries and answers) and rule bases (modules) between the agent nodes:

```
<Message mode="outbound" directive="pragmatic performative">
  <oid> <!-- conversation ID--> </oid>
  <protocol> <!-- transport protocol --> </protocol>
  <sender> <!-- sender agent/service --> </sender>
  <content> <!-- message payload -->  </content>
</Message>
```

- @$mode = inbound|outbound$ - attribute defining the type of a message
- @*directive* - attribute defining the pragmatic context of the message, e.g. a FIPA ACL performative
- $< oid >$ - the conversation id used to distinguish multiple conversations and conversation states
- $< protocol >$ - a transport protocol such as HTTP, JMS, SOAP, Jade, Enterprise Service Bus (ESB) ...
- $< sender >< receiver >$ - the sender/receiver agent/service of the message
- $< content >$ - message payload transporting a RuleML / Reaction RuleML query, answer or rule base

The directive attribute corresponds to the pragmatic instruction, i.e. the pragmatic characterization of the message context. External vocabularies defining pragmatic performatives might 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. [2, 18] A standard nomenclature of pragmatic performatives is defined by the Knowledge Query Manipulation Language (KQML) and the FIPA Agent Communication Language (ACL) which defines several speech act theory-based communicative acts. [8] Other vocabularies such as OWL-QL or the normative concepts of Standard Deontic Logic (SDL), e.g., to define action obligations or permissions and prohibitions, might be used as well.

The conversation identifier is used to distinguish multiple conversations and conversation states. This allows to associate messages as follow-up to previously existing conversations, e.g. to implement complex coordination and negotiation protocols, message-oriented workflows and complex event processing situations. For an overview and description of several negotiation and coordination protocols see [21]. Via sub-conversations it is possible to start e.g. meaning negotiations about the common shared pragmatic context and the shared ontologies which are necessary to understand the rule and event-based content of the interchanged messages.

The protocol might define lower-level ad-hoc or enterprise service bus transport protocols such as HTTP, JMS, and higher-level agent-oriented communication protocols such as Jade or Web Service protocols such as SOAP. More than 30 different transport protocols are supported by the enterprise service bus which is the main communication backbone in our implementation.

The content of a message might be a query or answer following a simple request-response communication pattern or it might follow a complex negotiation or coordination protocols where complete rule sets, complex events or fact bases serialized in RuleML / Reaction RuleML are interchanged.

### *The RuleML Interface Description Language.*

(RuleML IDL) as sublanguage of Reaction RuleML adopts the ideas of interface definition languages such as Corbas' IDL or Web Service WSDL. It describes the signatures of public rule functions together with their mode and type declarations and narrative human-oriented meta descriptions.

*Modes* are states of instantiation of the predicate described by mode declarations, i.e. declarations of the intended input-output constellations of the predicate terms with the following semantics:

- "+" The term is intended to be input
- "−" The term is intended to be output
- "?" The term is undefined/arbitrary (input or output)

We define modes with an optional attribute @*mode* which is added to terms in addition to the @*type* attribute, e.g.

$< Var\ mode = "-"\ type = "java://java.lang.Integer > X < /Var >$, i.e. the variable $X$ is an output variable of type $java.lang.Integer$. By default the mode is undefined "?".

For instance, the interface definition for the function $add(Arg1, Arg2, Result)$ with the modes $add(+, +, -)$ is as follows:

```
<Interface>
    <label>
       <Expr>
          <Fun uri="dc:description"/>
          <Ind>Definition of the add function which takes two Java
               integer values as input and returns the Integer result
               value
          </Ind>
       </Expr>
    </label>
    <Expr>
        <Fun>add</Fun>
        <Var type="java://java.lang.Integer" mode="-">Result</Var>
        <Var type="java://java.lang.Integer" mode="+">Arg1</Var>
        <Var type="java://java.lang.Integer" mode="+">Arg2</Var>
    </Expr>
</Interface>
```

## 3.2 Enterprise Service Bus as Communication Middleware

To seamlessly handle message-based interactions between the responder agents/services and with other applications and services using disparate complex event processing (CEP) technologies, transports and protocols an enterprise service bus (ESB), the Mule open-source ESB [17], is integrated as communication middleware. The ESB allows deploying the rule-based agents as highly distributable rule inference services installed as Web-based endpoints in the Mule object broker and supports the Reaction RuleML based communication between them. That is, the ESB provides a highly scalable and flexible application messaging framework to communicate synchronously but also asynchronously with external services and internal agents.

Mule is a messaging platform based on ideas from ESB architectures, but goes beyond the typical definition of an ESB as a transit system for carrying data between applications by providing a distributable object broker to manage all sorts of service components. The three processing modes of Mule are [17]:

- Asynchronously: many events can be processed by the same component at a time in various threads. When the Mule server is running asynchronously instances of a component run in various threads all accepting incoming events, though the event will only be processed by one instance of the component.
- Synchronously: when a UMO Component receives an event in this mode the whole request is executed in a single thread
- Request-Response: this allows for a UMO Component to make a specific request for an event and wait for a specified time to get a response back

The object broker follows the Staged Event Driven Architecture (SEDA) pattern [28]. The basic approach of SEDA is to decomposes a complex, event-driven application into a set of stages connected by queues. This design decouples event and thread scheduling from application logic and avoids the high overhead associated with thread-based concurrency models. That is, SEDA supports massive concurrency demands on Web-based services and provides a highly scalable approach for asynchronous communication.

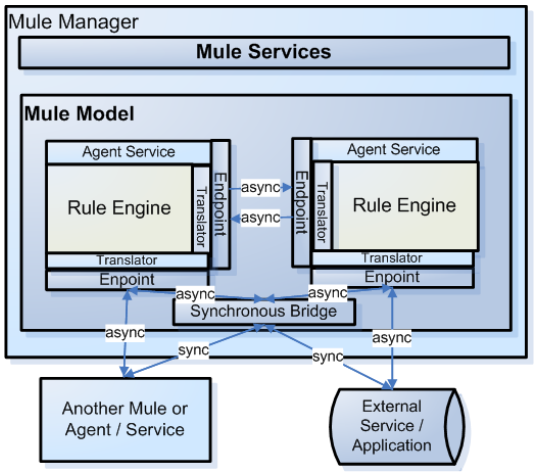


**Figure 3: Integration of Mule into RBSLM**

Figure 3 shows a simplified breakdown of the integration of Mule into Rule Responders' Pragmatic Agent Web.

Several agent services which at their core run a rule engine are installed as Mule components which listen at configured endpoints, e.g., JMS message endpoints, HTTP ports, SOAP server/client addresses or JDBC database interfaces. Reaction RuleML is used as a common platform independent rule interchange format between the agents (and possible other rule execution / inference services). Translator services are used to translate inbound and outbound messages from platform-independent Reaction RuleML into the platform-specific rule engines execution syntaxes and vice versa. XSLT and ANTLR based translator services are provided as Web forms, HTTP services and SOAP Web services on the Reaction RuleML Web page [23].

The large variety of transport protocols provided by Mule can be used to transport the messages to the registered endpoints or external applications / tools. Usually, JMS is used for the internal communication between distributed agent instances, while HTTP and SOAP is used to access external Web services. The usual processing style is asynchronous using SEDA event queues. However, sometimes synchronous communication is needed. For instance, to handle communication with external synchronous HTTP clients such as Web browsers where requests, e.g. by a Web from, are send through a synchronous channel. In this case a synchronous bridge component dispatches the requests into the asynchronous messaging framework and collects all answers from the internal service nodes, while keeping the synchronous channel with the external service open. After all asynchronous answers have been collected they are send back to the still connected external service via the synchronous channel.

## 3.3 Platform-dependent Rule Engines as Execution Environments

Each agent service might run one or more arbitrary rule engines to execute the interchanged queries, rules and events and derive answers on requests. In this subsection we will

introduce Prova [15, 19], a highly expressive Semantic Web rule engine which we used in our reference implementation for agents with complex reaction workflows, decision logic and dynamic access to external Semantic Web data sources. Another rule engine which we applied was the OO jDrew rule engine [1] in order to demonstrate rule interchange between various rule engines. Further rule engines and event correlation engines (CEP engines) are planned to join the Rule Responder project.

Prova follows the spirit and design of the recent W3C Semantic Web initiative and combines declarative rules, ontologies and inference with dynamic object-oriented Java API calls and access to external data sources such as relational databases or enterprise applications and IT services. One of the key advantages of Prova is its elegant separation of logic, data access, and computation and its tight integration of Java and Semantic Web technologies. It includes numerous expressive features and logical formalisms such as:

- Easy to use and learn ISO Prolog related scripting syntax
- Well-founded Semantics for Extended Logic Programs with defeasible conflict resolution and linear goal memoization
- Order-sorted polymorphic type systems compatible with Java and Semantic Web ontology languages RDF/RDFS and OWL
- Seamless integration of dynamic Java API invocations
- External data access by e.g., SQL, XQuery, RDF triple queries, SPARQL
- Meta-data annotated modular rule sets with expressive transactional updates, Web imports, constructive views and scoped reasoning for distributed rule bases in open environment such as the Web
- Verification, Validation and Integrity tests by integrity constraints and test cases
- Messaging reaction rules for workflow like communication patterns based on the Prova Agent Architecture
- Global reaction rules based on the ECA approach
- Rich libraries and built-ins for e.g. math, date, time, string, interval, list functions

For a detailed description of the syntax, semantics and implementation of several of these formalisms see e.g. [19].

## 4. RULE RESPONDER USE CASE

In this section we describe a real-world use case, namely the *RuleML-200x Symposium organization*, which address typical problems and tasks in a virtual organization. Further use cases can be found on the Rule Responder project site: *responder.ruleml.org*.

The RuleML-200x Responder use case implements the RuleML-200x symposium organization as a virtual organization consisting of self-autonomous rule-based agents who fulfil typical conference organization and project management tasks and who respond to incoming requests from external

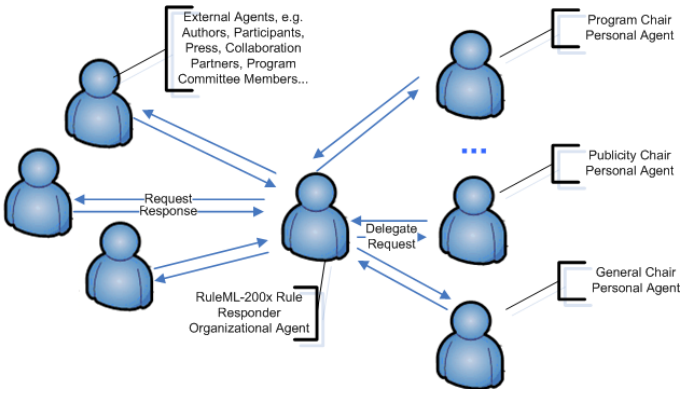


**Figure 4: RuleML-200x Use Case**

agents, e.g., from authors, participants, program committee members ... (see figure 4).

The RuleML-200x Responder agent (organizational agent) acts as a single point of entry for the RuleML-200x organization. It filters, decides and delegates incoming queries and requested tasks to the organizations' members (e.g. the organizing committee members) which are implemented as distributed rule-based personal agents. Project management techniques such as a responsibility assignment matrix (see table 1) and role models are implemented by the RuleML-2007 Responder as ontological models (in OWL) to describe the roles and responsibilities of the personal agents in the virtual organization. Negotiation and distributed coordination protocols are applied to manage and communicate with the project team and external agents.

**Table 1: Responsibility Assignment Matrix**

| | General Chair | Program Chair | Publicity Chair | ... |
|---|---|---|---|---|
| Symposium | responsible | consulted | supportive | ... |
| Website | accountable | responsible | | ... |
| Sponsoring | informed, signs | verifies | responsible | ... |
| Submission | informed | responsible | | ... |
| ... | ... | ... | ... | ... |

The personal agents act as self-autonomous agents having their own rule-based decision and behavioural logic on top of their personal information sources, Web services, vocabularies / ontologies and knowledge structures. This allows them, e.g., to selectively reveal personal information such as contact information (e.g. show only parts of FOAF profiles or vCards) or react and proactively plan according to the occured situation (e.g. schedule a meeting based on personal iCal calendar data - see figure 7).

As shown in figure 5, each agent in the RuleML-200x virtual organization is implemented as a Web-based service consisting of a set of internal or external data and knowledge sources and a rule execution environment (a rule engine). Reaction RuleML is applied as common rule interchange and event messaging format, Prova and OO jDrew are used as two exemplary rule engines in the implementation of the organizational and personal agents, and the Mule ESB is used as communication middleware between the agent endpoints. Reaction RuleML messages (event messages) are transported by the ESB to the appropriate

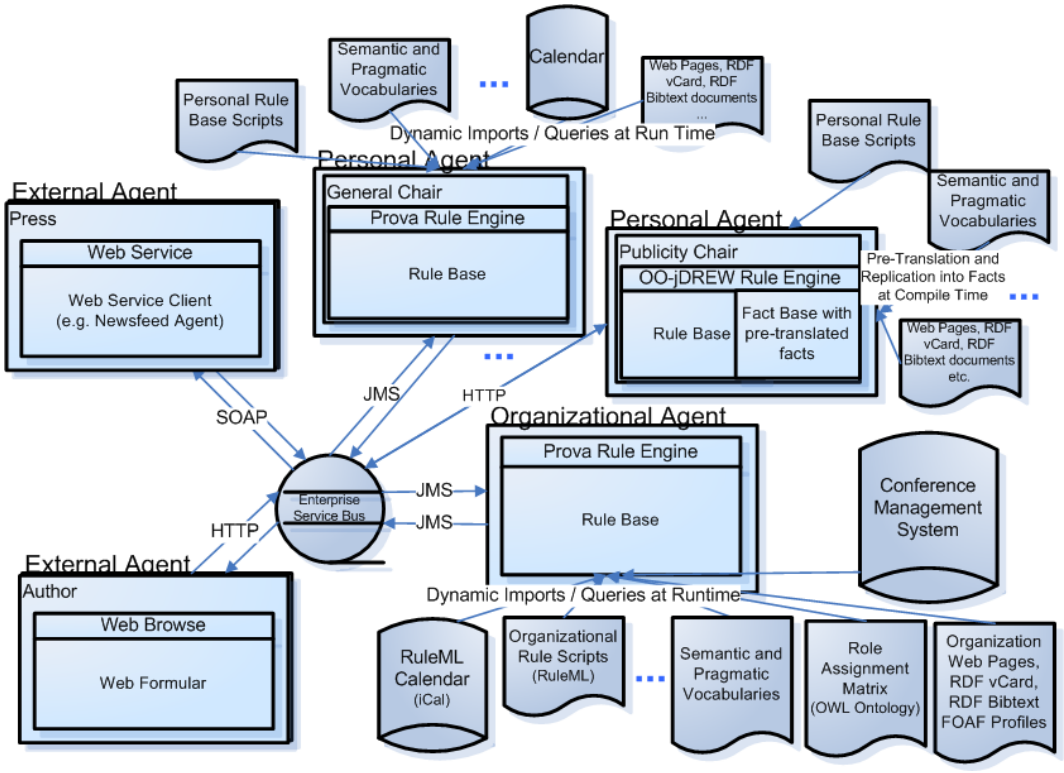


**Figure 5: RuleML-200x Use Case Implementation**

internal agent nodes or external communication interfaces based on a broad spectrum of selectable transport protocols such as HTTP, JMS, Web Service protocols (SOAP) or e.g. agent communication languages (JADE). The platform-independent interchanged RuleML messages which contain the message payload, e.g. queries or answers, as well as meta information about the conversation and the pragmatic context of the message, are translated by translator services (e.g. XSLT style sheets) into the platform-dependent, specific execution language of the rule-based execution environment at the agent endpoint(s).

The Role Activity Diagram (RAD) shown in figure 6 describes a simple query-answer (request-response) pattern. An external agent requests some information from the RuleML-2007 organization. The organizations' responder agent tries to understand the query and starts a sub-conversation informing the requester if the pragmatic context or the message content was not understood. In this case the requester agent informs the organization agent with further relevant information which is need to understand the query, e.g. references to relevant ontology parts or synonyms. If the message is understood by the organizational agent it delegates the query (possibly executing some further preprocessing) in a new sub-conversation to the responsible personal agent (according to the responsibility assignment matrix). Other roles (personal agents) might be informed in parallel (not shown here). The personal agent derives the answers and sends them back one by one to the organizational agent which might simply forward them to the original external requesting agent.

The implementation in Prova uses messaging reaction rules

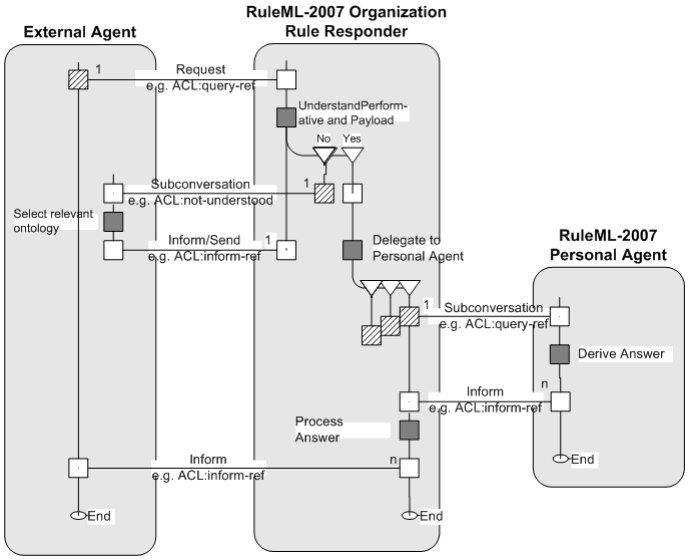


**Figure 6: Role Activity Diagram for a simple Query-Answer Conversation**

which send and receive outbound and inbound messages. For instance, the rule receives a query from the ESB, sends it to another agent in a new sub-conversation, receives the answer from the agent, and sends back the answer to the original requester: (variables start with upper case letters)

```
rcvMsg(CID,esb, Requester, acl_query-ref, Query) :-
 ...
 sendMsg(Sub-CID,esb,Agent,acl_query-ref, Query),
 rcvMsg(Sub-CID,esb,Agent,acl_inform-ref, Answer),
 ...
 sendMsg(CID,esb,Agent,acl_inform-ref,Answer).
```

On the PIM layer this Prova rule is serialized in Reaction RuleML as a reaction rule:

```
<Rule style="active">
   <event> <!-- receive inbound message -->
       <Message mode="inbound">...</Message>
   </event>
   <condition>
     <And>
        <Rule style="active"> <!-- send outbound message -->
           <action>
              <Message mode="outbound">...</Message>
           </action>
        </Rule>
        <Rule style="active"> <!-- receive inbound messages -->
           <event>
              <Message mode="inbound">...</Message>
           </event>
        </Rule>
     </And>
   </condition>
   <action> <!-- send outbound message -->
       <Message mode="outbound">...</Message>
   </action>
</Rule>
```

The corresponding reaction rule on the personal agents' side might look at follows:

```
% answers query
rcvMsg(XID, esb, From, Performative, [X|Args]):-
    derive([X|Args]),
    sendMsg(XID,esb,From, answer, [X|Args]).
```

This rule tries to derive every incoming query and sends back the answers. The list notation $[X|Args]$ will match with arbitrary n-ary predicate functions, i.e., it denotes a kind of restricted second order notation since the variable $X$ is always bound, but matches to all functions in the signature of the language with an arbitrary number of arguments $Args$. For instance, a function $p(X, Y)$ is equivalent to a list $[p, X, Y]$ where the function name being the first element in the list. Note, that the complete conversation is local to the conversation ID, the used protocol and the pragmatic context denoted by the performative(s).

With this flexible reaction rules process flows can be implemented such as complex negotiation and coordination protocols [21]. For instance, a typical process in a virtual organization such as the RuleML-200x organization is the scheduling of a meeting (e.g. a telephone conference) as described in figure 7.

The RuleML-200x organizational agent creates a possible date for the meeting from the public organizational calendar (accessed e.g. via iCAL) and proposes this date to all personal agents. The agents compare this date with their personal calendars and send counter-proposals if the deadline does not fit according to their personal decision logic. The organizational agent then creates a new proposal. This process is repeated until all agents agreed on the proposed

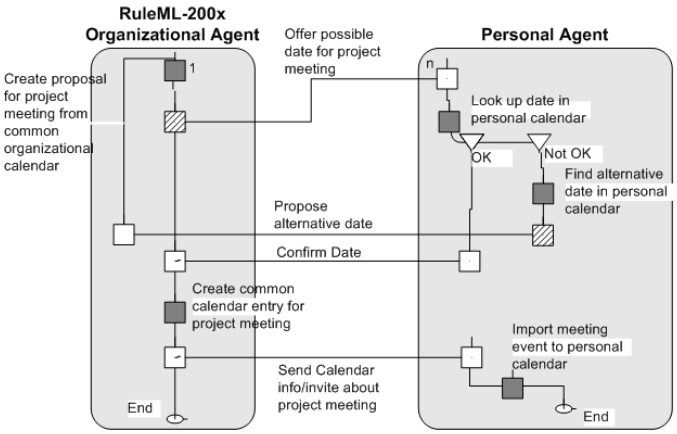


**Figure 7: Role Activity Diagram for Scheduling a Meeting**

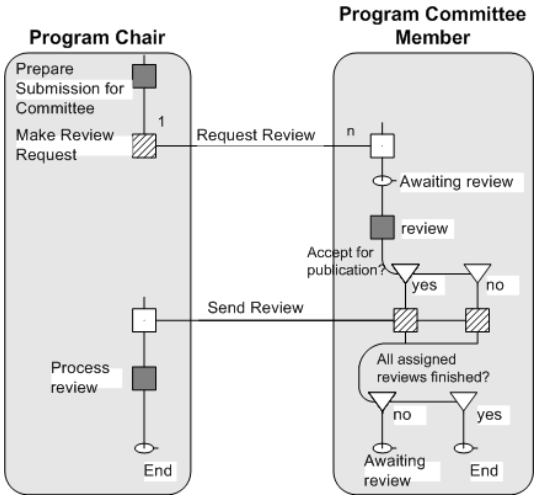


**Figure 8: Role Activity Diagram for the Reviewing Process**

meeting date; the organizational agent then creates an entry in the public calendar and informs all agents about this date. The personal agents add this date to their personal (not public) calendars. Note, that the personal agents implement their own, self-autonomous decision logic. For instance, they can lie and pretend they have no time at a certain date or propose strategic counter-proposals.

Another scenario is the reviewing process of submissions to the conference, as modelled in figure 8.

The program chair assigns papers to program committee members and sends requests to them. Each program committee members then reviews the assigned submissions and informs the program chair about the review comments and the acceptance or rejection decision. If there are further pending reviews the program committee member agent iterates the "awaiting review" state until all assigned papers have been reviewed. The program chair processes the received reviews.

Several other typical processes in conference organizations

demonstrating the interplay of the different agent roles, e.g., between the responsible, supportive, consulted, sings, informed, ... role for a particular task according to the role assignment matrix, have been implemented in this use case.

In summary, conversations via event messages are used to coordinate the agents in the RuleML-200x organization. A conversation is local to the conversation id, the pragmatic context, the used protocols and the sender,receiver agents. Each agent implements his own decision and reaction logic in terms of rules with public interfaces which can be accessed and queried and not public rules. These rules might e.g., represent personal strategies, failure handling policies or negotiation patterns, e.g. for meaning clarification. External data sources such as calendars, vocabulary definitions, databases, web pages, meta data sources, personal data (e.g. FOAF profile, vCard) are dynamically queried at runtime and used as facts in the internal knowledge base of an agents.

# 5. CONCLUSION

Recently, there have been many efforts aiming on rule interchange and building a general rule markup and modelling standard for the (Semantic) Web. This includes several important *general standardization or standards-proposing efforts* including RuleML [3], W3C RIF [25], OMG PRR and others. However, to the best of our knowledge no methodological and architectural design and comprehensive implementation exists which makes this idea of a practical distributed rule layer in the Semantic Web a reality. Moreover, in the current rule interchange formats the pragmatic aspect is missing.

In the Rule Responder project we follow a constructivists design science research methodology [12] and contribute with a rule-based middleware based on modern efficient and scalable enterprise service technologies, complex event processing techniques and standardized web rule and Semantic Web languages in combination with existing meta data vocabularies and ontologies to capture and negotiate the individual and shared semantic and pragmatic context of rule-based agents and service networks. The application in virtual organizations such as Agent communities or (business) service networks is of high practical relevance and transfers the existing work in multi-agent systems (e.g. Jade, FIPA-OS) to the Semantic-Pragmatic Web and rule-based service architecture.

Rule Responder builds upon these existing ideas and technologies in multi-agent systems and tackles the manifold challenges which are posed by the highly complex, dynamic, scalable and open distributed nature of semi-automated pragmatic agents communities or service component architectures. Our proposed design artifact exploits RuleML and Reaction RuleML for the XML-based representation of reaction rules and message based conversations at the platform-independent level as a compact, extensible and standardized rule and event interchange format. A highly scalable and efficient enterprise service bus is integrated as a communication middleware platform and web-based agent/service object broker. Via translator services the interchanged RuleML messages are translated into the platform-specific execution syntaxes of the arbitrary agents' rule execution environments such as Prova. In sum, the proposed design artifact addresses many practical factors of rule-based service technologies ranging from system engineering features like modular management and encapsulation to interchangeability and interoperability between system and domain boundaries in open environments such as the Web.

# 6. REFERENCES


[1] M. Ball, H. Boley, D. Hirtle, J. Mei, and B. Spencer. The OO jDrew Reference Implementation of RuleML. In *RuleML 2005*, Galway, 2005.

[2] H. Boley. The Rule-ML Family of Web Rule Languages. In *4th Int. Workshop on Principles and Practice of Semantic Web Reasoning*, Budva, Montenegro, 2006.

[3] H. Boley and S. Tabet. RuleML: The RuleML Standardization Initiative, http://www.ruleml.org/, 2000.

[4] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/, accessed June 2005, 2004.

[5] A. Consortium. vCard: The Electronic Business Card, http://www.imc.org/pdi/vcardwhite.html, 1997.

[6] F. Dawson and D. Stenerson. Internet Calendaring and Scheduling Core Object Specification (iCalendar), http://www.ietf.org/rfc/rfc24, 1998.

[7] DCMI. Dublin Core Metadata Initiative, http://dublincore.org/, accessed June 2004, 2001.

[8] FIPA. FIPA Agent Communication Language, http://www.fipa.org/, accessed Dec. 2001, 2000.

[9] FOAF. The Friend-Of-A-Friend project, http://www.foaf-project.org/, accessed June 2005, 2005.

[10] GRDDL. Gleaning Resource Descriptions from Dialects of Languages, www.w3.org/2004/01/rdxh/spec, accessed June 2001, 2001.

[11] P. Hayes. RDF Semantics, http://www.w3.org/TR/2004/REC-rdf-mt-20040210, accessed Dec. 2005, 2004.

[12] A. Hevner, S. March, J. Park, and S. Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–101, 2004.

[13] M. N. Huhns and M. P. Singh. *Readings in Agents*. Morgan Kaufmann, San Francisco, 1998.

[14] G. Klyne and J. Caroll. Resource Description Framework (RDF): Concepts and Abstract Syntax, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210, accessed Nov. 2005, 2004.

[15] A. Kozlenkov, A. Paschke, and M. Schroeder. Prova, http://prova.ws, accessed Jan. 2006, 2006.

[16] D. L. McGuinness and F. v. Harmelen. OWL Web Ontology Language, http://www.w3.org/TR/owl-features/, accessed June 2005, 2004.

[17] Mule. Mule Enterprise Service Bus, http://mule.codehaus.org/display/MULE/Home, 2006.

[18] A. Paschke. A Typed Hybrid Description Logic Programming Language with Polymorphic Order-Sorted DL-Typed Unification for Semantic Web Type Systems. In *OWL-2006 (OWLED'06)*, Athens, Georgia, USA, 2006.

[19] A. Paschke. *Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management*. Idea Verlag GmbH, Munich, 2007.

[20] A. Paschke, B. Harold, A. Kozlenkov, and B. Craig. Rule Responder: A RuleML-Based Pragmatic Agent Web for Collaborative Teams and Virtual Organizations, http://ibis.in.tum.de/projects/paw/, 2007.

[21] A. Paschke, C. Kiss, and S. Al-Hunaty. NPL: Negotiation Pattern Language - A Design Pattern Language for Decentralized (Agent) Coordination and Negotiation Protocols. In R. Banda, editor, *E-Negotiation - An Introduction*. ICFAI University Press, ISBN 81-314-0448-X, 2006.

[22] A. Paschke, A. Kozlenkov, and H. Boley. A Homogenous Reaction Rules Language for Complex Event Processing. In *International Workshop on Event Drive Architecture for Complex Event Process (EDA-PS 2007)*, Vienna, Austria, 2007.

[23] A. Paschke, A. Kozlenkov, H. Boley, M. Kifer, S. Tabet, M. Dean, and K. Barrett. Reaction RuleML, http://ibis.in.tum.de/research/ReactionRuleML/, 2006.

[24] O. Patashnik. *BibTeXing*. 1998.

[25] RIF. W3C RIF: Rule Interchange Formant, http://www.w3.org/2005/rules/, accessed Oct. 2005, 2005.

[26] M. Schoop, A. Moor, and J. Dietz. The Pragmatic Web: A manifesto. *Communications of the ACM*, 49(5), 2006.

[27] SIOC. Semantically-Interlinked Online Communities, http://sioc-project.org/, accessed June 2005, 2005.

[28] M. Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for WellConditioned, Scalable Internet Services. In *Proceedings of Eighteeth Symposium on Operating Systems (SOSP-18)*, Chateau Lake Louise, Canada, 2001.