Chapter 1
Rule Responder Agents: Framework and Instantiations

Harold Boley[1], Adrian Paschke[2]

Abstract R
ule Responder is a framework for specifying virtual organizations as semantic multi-agent systems that support collaborative teams. It provides the infrastructure for rule-based collaboration between the distributed members of such a virtual organization. Human members of an organization are assisted by (semi-)autonomous rule-based agents, which use Semantic Web rules to describe aspects of their owners' derivation and reaction logic.

To implement different distributed system/agent toplogies with their negotiation/coordination mechanisms Rule Responder instantiations employ three core classes of agents - Organizational Agents (OA), Personal Agents (PAs), and External Agents (EAs). The OA represents goals and strategies shared by its virtual organization as a whole, using a rulebase that describes its policies, regulations, opportunities, etc. Each PA assists a group or person of the organization, semi-autonomously acting on their behalf by using a local knowledge base of rules defined by the entity. EAs can communicate with the virtual organization by sending messages to the public interfaces of the OA. EAs can be human users using, e.g., Web forms or can be automated services/tools sending messages via the multitude of transport protocols of the underlying enterprise service bus (ESB) middleware.

The agents employ ontologies in their knowledge bases to represent semantic domain vocabularies, normative pragmatics and pragmatic context of conversations and actions, as well as the organizational semiotics. For instance, an OWL ontology is used by the OA to represent a Responsibility Assignment Matrix (RAM) to find an appropriate PA that can handle an incoming query. The OA uses reaction rules to send the query to this PA,

[1] Institute for Information Technology, National Research Council Canada
Fredericton, NB, Canada
e-mail: `harold.boleyATnrc.gc.ca`
[2] Freie Universitaet Berlin, Germany
e-mail: `paschkeATmi.fu-berlin.de`

receive its answer(s), do validation(s), and send the answer(s) back to the EA. To derive the answer(s), PAs use deliberation rules whose expressiveness can range from Datalog to Higher-Order and Modal Logics.

Several instantiations of Rule Responder have been developed, including the Health Care and Life Sciences eScience infrastructure, the Rule-based IT Service Level Managment, and Semantic BPM system, multiple versions of the deployed SymposiumPlanner system, two versions of the WellnessRules prototype, and a PatientSupporter prototype. For example, the WellnessRules instantiations answer queries about possible activities of participants in a wellness organization, where PAs on the level of coaches support individual participants of wellness groups (e.g., for running or swimming) based on participant profiles. This permits other participants to discover an existing wellness group matching their interests or create a new one.

1.1 Introduction

Rule Responder[1] extends the Semantic Web towards a Pragmatic Web infrastructure for collaborative rule-based agent networks realizing distributed inference services, where independent agents engage in conversations by exchanging messages and cooperate to achieve (collaborative) goals. Rule Responder can be characterized on three levels, from general to specific.

- It models a virtual organization of agents recursively as again being a single agent, forming what has been called [Koe67] a hierarchy of *holons* (or, a *holarchy*).
- It supports different *interaction/coordination models*, where information is interchanged within a *pragmatic context* (e.g. language action speech acts, deontic norms, etc.).
- It provides a technical Web-based *multi-agent architecture* which supports different distribution models (distributed agent system topologies).

A virtual organization as a whole is represented by an Organizational Agent (OA), which uses ontologies and rules to assign and delegate incoming tasks (e.g., queries) to responsible Personal Agents (PAs). Rule Responder agents communicate in conversations that allow implementing different agent coordination and negotiation protocols. The interaction and interpretation is driven by the organizational semiotics which details how the information flow works within and between organizations. For instance, an OA can use a responsibility assignment matrix, often represented as an ontology, to find an appropriate PA in its organization. The OA can then send a message (e.g., a query) to that PA and receive results (e.g., answers), typically using reaction rules. By means of pragmatic primitives, such as speech acts, deontic

[1] `http://responder.ruleml.org`

norms, etc., which are represented as ontologies, Rule Responder attaches the semantic and pragmatic context, e.g. organizational norms, purposes or goals and values, to the interchanged messages.

In its multi-agent architecture Rule Responder utilizes messaging reaction rules from Reaction RuleML[2] for communication between the distributed agent inference services. The Rule Responder middleware is based on modern enterprise service technologies and Semantic Web technologies for implementing intelligent agent services that access data and ontologies, receive and detect events (e.g., for complex event processing in event processing agent networks), and make rule-based inferences and (semi-)autonomous pro-active decisions for reactions based on these representations.

The core of a Rule Responder agent is a rule engine, such as Prova[3], OO jDREW, DR-Device (initially in Emerald), Euler, or Drools, which implements the decision and behavioral reaction logic of the agents' roles. An agent can employ vocabularies defined as Semantic Web ontologies (e.g., based on RDFS or OWL) to give its rules a domain-specific meaning. The vocabularies can be used within the conversation with other agents to enable a semantic and pragmatic interpretation of the messages. For the deployment of agents on the Web and for the communication in agent networks, Rule Responder uses the Mule-based enterprise service bus middleware, which supports a multitude of synchronous and asynchronous transport protocols (>40) – such as MS, SMTP, JDBC, TCP, HTTP, XMPP, Jade – to transport rulebases, queries and answers between the agents. The de facto standard Reaction RuleML is used as a platform-independent rule interchange format for agent conversation.

In summary, Rule Responder can be seen to support a *digital ecosystem*, evolving from the Semantic Web [BC07] to the Pragmatic Web, which consists of all the semantic agents in one or more virtual organizations, as well as all the other components of this environment with which the agents interact, such as other services, tools, the ESB middleware, etc.

Several instantiations of Rule Responder have been developed, including the eScience infrastructure for Health Care and Life Sciences [Pas08], the Rule-based IT Service Level Managment and Semantic BPM system [PB08, PK08], multiple versions of the deployed SymposiumPlanner system [CB08], two versions of the WellnessRules prototype [BOC09], and the PatientSupporter prototype.[4]

The rest of the chapter is organized as follows. Section 1.2 discusses the agent architecture and used technologies of the Rule Responder framework. Section 1.3 explains a typical distributed agent topology for virtual organizations and the types agents used to implement it. Section 1.4 focuses on interchange between the semantic agents which communicate by using (Reac-

[2] `http://reaction.ruleml.org`

[3] `http://prova.ws`

[4] `http://ruleml.org/PatientSupporter`

tion) RuleML as common rule interchange format. Section 1.5 demonstrates some application use cases of Rule Responder by means of selected Rule Responder instantiations. Section 1.6 concludes the paper.

1.2 The Rule Responder Framework

Three interconnected architectural layers consitute the Rule Responder framework, listed here from top to bottom:

- Computationally independent user interfaces such as template-based Web forms or controlled English rule interfaces.
- Reaction RuleML as the common platform-independent rule interchange format to interchange rules, events, queries, and data between Rule Responder agents and other agents (e.g., Semantic Web services or humans via Web forms).
- A highly scalable and efficient enterprise service bus (ESB) as agent/service-broker and communication middleware on which platform-specific rule engines are deployed as distributed agent nodes (resp. semantic inference services). These engines manage and execute the logic of Rule Responder's semantic agents in terms of declarative rules which have access to semantic ontologies.

In the following, the Rule Responder framework will be refined, and explained from bottom to top.

1.2.1 Mule Enterprise Service Bus

To seamlessly handle message-based interactions between the Rule Responder agents/services and other agents/services using disparate complex event processing (CEP) technologies, transports, and protocols, an enterprise service bus (ESB) – the Mule open-source ESB [5] – is used in Rule Responder as the communication middleware. This ESB allows deploying the rule-based agents as highly distributed rule inference services installed as Web-based endpoints on the Mule object broker and supports the communication in this rule-based agent processing network via a multitude of transport protocols (see Figure 1.1). That is, the ESB provides a highly scalable and flexible application messaging framework to communicate synchronously or asynchronously amongst the ESB-local agents and with agents/services on the Web.

Mule is a messaging platform-based on principles of ESB architectures, but goes beyond the typical definition of an ESB as a transit system for

[5] `www.mulesoft.org`

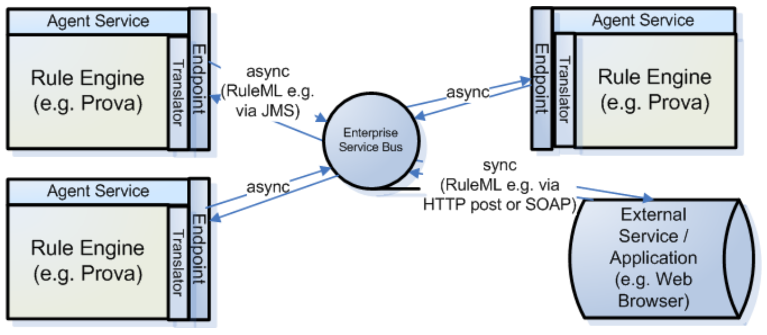

Fig. 1.1 Distributed Rule Responder Agent Services

carrying data between applications by providing a distributable object broker to manage all sorts of service components such as the Rule Responder agent services. The three processing modes of Mule are:

- Asynchronous: many events (messages) 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 an event will only be processed by one instance of the component.
- Synchronous: when a component receives an event message, in this mode the whole request is executed in a single thread.
- Request-Response: this allows for a 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 [WCB01]. 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 1.2 shows a simplified breakdown of the integration of Mule into the Rule Responders framework.

Distributed agent services (see Figure 1.1), which at their core run a rule engine, are deployed as Mule components which listen at configured endpoints, e.g., JMS message endpoints, HTTP ports, SOAP server/client addresses or JDBC database interfaces, etc. 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 execution syntaxes of rule engines, and vice versa. XSLT and ANTLR based translator services are provided as

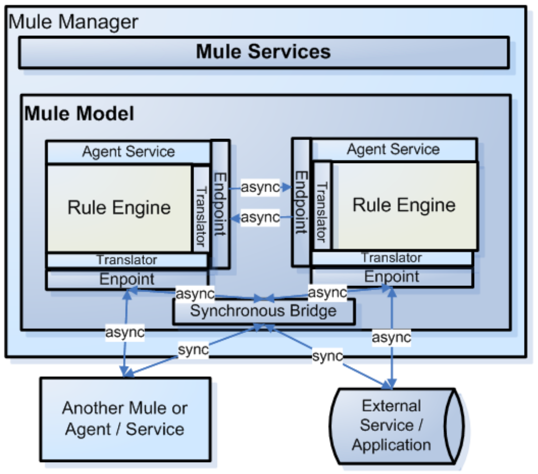

Fig. 1.2 Layering of Rule Responder on Mule ESB

Web forms, HTTP services and SOAP Web services on the Reaction RuleML Web page.

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 sent 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 sent back to the still connected external service via the HTTP-synchronous channel.

1.2.2 Selected Platform-Specific Rule Engines for Rule Responder Agents

The core of a Rule Responder agent, which is deployed as a service component on the Rule Responder ESB, is a platform-specific rule engine. These engines might differ, e.g., in their supported rule types, state representation, rule evaluation mechanism, conflict resolution and truth maintenance. Hence, depending on their expressiveness and functionalities, these rule engines might be capable of implementing agents in the strong sense of cognitive architectures for intelligent agents with goal/task-based, utility-based and learning-based functionalities, or in the weak sense of inference agent services with simple reflexive functionalities for, e.g., deductive query-answering capabilities. Following the general consensus defined by the strong notion of agency in [Woo01], a Rule Responder agent, in addition to being (semi-)autonomous, should be capable of reactive, proactive, and communicative behavior. Additionally, it is often important that certain mentalistic notions[6] can be used in the rule language for describing the agent behavior in an abstract and intuitive way, e.g. in the interactions between agents to communicate the pragmatics of the interchanged information.

In the following, the interplay between our most often used rule engines Prova, OO jDREW, Euler will be discussed, although there are other engines such as DR-Device and Drools supported by Rule Responder.

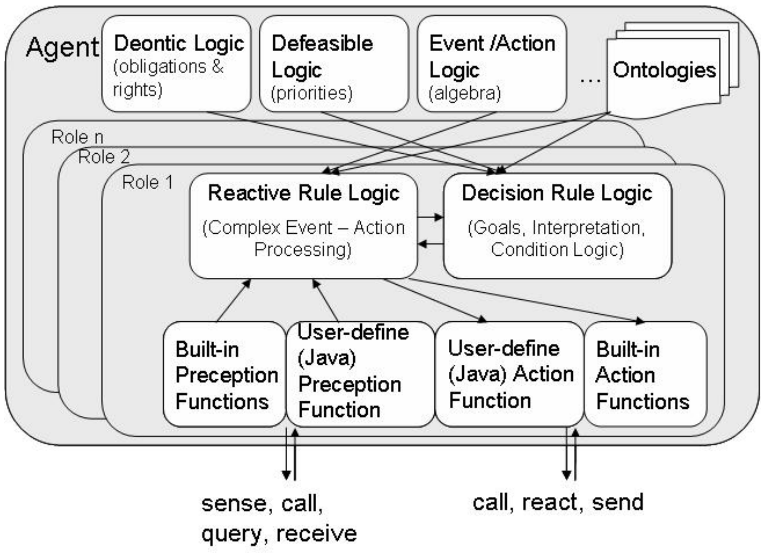
Agent
Deontic Logic (obligations & rights)
Defeasible Logic (priorities)
Event /Action Logic (algebra)
...
Ontologies
Role n
Role 2
Role 1
Reactive Rule Logic (Complex Event – Action Processing)
Decision Rule Logic (Goals, Interpretation, Condition Logic)
Built-in Preception Functions
User-define (Java) Preception Function
User-define (Java) Action Function
Built-in Action Functions
sense, call, query, receive
call, react, send

Fig. 1.3 Rule Responder Agent

Figure 1.3 shows the architecture of an intelligent cognitive Rule Responder agent which is implemented in Prova. Prova[7] is an enterprise-strength, highly expressive distributed Semantic Web logic programming (LP) rule engine. The Prova rule engine supports different rule types:

[6] The term *mentalistic notions* aka *mental attitudes* refers to human properties such as beliefs, goals, etc. when transferred to describing machine agents.

[7] `http://prova.ws`

- Derivation rules to describe the agent's decision logic
- Integrity rules to describe constraints and potential conflicts
- Normative rules to represent the agent's permissions, prohibitions and obligation policies
- Global ECA-style reaction rules to define global reaction logic which are triggered on the basis of detected (complex) events
- Messaging reaction rules to define conversation-based workflow reaction and behavioral logic based on complex event processing

Prova follows the spirit and design of the W3C Semantic Web initiative and combines declarative rules, ontologies and inference with dynamic object-oriented programming and access to external data sources via built-in query languages such as SQL, SPARQL, and XQuery.

```
File Input / Output
    ..., fopen(File,Reader), ...
XML (DOM)
    document(DomTree,DocumentReader) :-   XML(DocumenReader),...
SQL
    ... ,sql_select(DB,cla,[pdb_id,"1alx"],[px,Domain]).
RDF
    ...,rdf(http://...,"rdfs",Subject,"rdf_type","gene1_Gene"),...
XQuery
 ..., XQuery = 'for $name in StatisticsURL//Author[0]/@name/text()
    return $name', xquery_select(XQuery,name(ExpertName)),...
SPARQL
    ...,sparql_select(SparqlQuery,...
```

For instance, the following rule uses a SPARQL query to access an RDF Friend-of-a-Friend (FOAF) profile published on the Web. The selected data is assigned to variables which can be used within an agent's rule logic, e.g. to expose the agent's contact data.

```
exampleSPARQLQuery(URL,Type) :-
 QueryString = ' PREFIX foaf: <http://xmlns.com/foaf/0.1/>
                PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
                SELECT ?contributor ?url ?type
                FROM <http://planetrdf.com/bloggers.rdf>
                WHERE {
                        ?contributor foaf:name "Bob DuCharme" .
                        ?contributor foaf:weblog ?url .
                        ?contributor rdf:type ?type . } ',
 sparql_select(QueryString,url(URL),type(Type)).
```

One of the key advantages of Prova is its elegant separation of logic, data access, and computation as well as its tight integration of Java, Semantic Web technologies, with service-oriented computing and complex event processing. In particular, Prova supports external type systems such as, e.g., Java class hierarchies or Semantic Web ontologies (RDFS, OWL) via its typed order-sorted logic [Pas06]. For instance, in the following example all agents from an external OWL ontology responsibility assignment matrix (RAM) are assigned to the typed variable *Agent* of type *Organizing_Committee* (with the namespace *ruleml*2010), where *Organizing_Committee* is a concept defined in the RAM ontology. The query then selects all agent individuals of type

ProgramChair which is a subtype of *Organizing_Committee*, i.e. the query selects a subset with appropriate subtype from the bound variable.

```
% import external ontology representing responsibility assignment matrix (RAM)
import("http://2010.ruleml.org/RuleML-2010.owl").
% bind all agent instances of type "Organizing_Committee" from the RAM to the variable Agent
agent(Agent:ruleml2010_Organizing_Committee).
% query for all agents of type "ProgramChair"
:-solve(agent(Agent:ruleml2010_ProgramChair)
```

Prova can be run in a plain Java environment as stand alone application or rule inference service on the Rule Responder ESB, or as an OSGI component enabling massive parallelization of Prova agent nodes in grid/cloud environments. Prova has a modular knowledge base to implement several different roles an agent might play in the same agent instance. Each role has its own set of reaction rules to autonomously react (potentially proactive) on detected situations (complex events) and its own set of decision rules to interpret goals and derive decisions according to conditional proofs. For instance, it is possible to consult (load) distributed rulebases from local files, a Web address, or from incoming messages transporting a rulebase.

```
%load from a local file
:- eval(consult("organization2009.prova")).
% import from a Web address
:- eval(consult("http://ruleml.org/organization2010.prova")).
```

The rulebases are managed as modules in the knowledge base. Their module label can be used for asserting or retracting complete modules from the knowledge base and for scoping queries / goals to a particular module, i.e. the query only applies to the particular scoped module. In the following example the subgoal $agent(Agent)$ applies on the modules $organization2009.prova$ and not on the module $http://ruleml.org/organization2010.prova$.

```
responsible(Agent, Task):-
  @src("organization2009.prova") agent(Agent),
  ...
```

To sense the environment and trigger actions, query data from external sources such as databases, call external procedural code such as Enterprise Java Beans, and receive / send messages from / to other agents or external services, Prova provides a set of built-in functions and additionally can dynamically instantiate any Java object and call its API methods at runtime. For instance, the following simple rule creates a response sentence with the name using Java string computations and displays it via to the Java system out console.

```
hello(Name):-
  S = java.lang.String("Hello, your name is "),
  S.append (Name),
  java.lang.System.out.println (S).
```

Additional libraries can be imported, e.g. to represent rights and obligations of agents, implement conflict handling rules, or describe complex events and actions.

In its cognitive cycle a Prova agent follows the sense-reason-act pattern. However, Prova does not define one particular cognitive cycle, but allows configuring an agent with user-defined conversation-based negotiation and coordination protocols or workflow patterns. Via constructs for asynchronously sending and receiving event messages within rules, an agent interacts with the environment. The main language constructs of messaging reaction rules are: *sendMsg* predicates to send messages, reaction *rcvMsg* rules which react to inbound messages, and *rcvMsg* or *rcvMult* inline reactions in the body of messaging reaction rules to receive one or more context-dependent multiple inbound event messages:

```
sendMsg(XID,Protocol,Agent,Performative,Payload |Context)
rcvMsg(XID,Protocol,From,Performative,Paylod|Context)
rcvMult(XID,Protocol,From,Performative,Paylod|Context)
```

Here, *XID* is the conversation identifier (conversation-id) of the conversation to which the message will belong. *Protocol* defines the communication protocol. *Agent* denotes the target party of the message. *Performative* describes the pragmatic envelope for the message content. A standard nomenclature of performatives is, e.g., the FIPA Agents Communication Language (ACL). *Payload* represents the message content sent in the message envelope. It can be a specific query or answer or a complex interchanged rule base (set of rules and facts). For instance, the following rule snippet shows how a query is sent to an agent via the ESB and then an answer is received from this agent.

```
...
sendMsg(Sub_CID,esb,Agent,acl_query-ref, Query),
rcvMsg(Sub_CID,esb,Agent,acl_inform-ref, Answer),
...
```

Prova does not define a specific set of mentalistic notions as first-class programming constructs. Instead, interchanged messages besides the conversation's metadata and payload also carry the pragmatic context of the conversation such as communicative situations / acts, mentalistic notions, organizational and individual norms, purposes or individual goals and values. The payload of incoming event messages is interpreted with respect to the local conversation state, which is denoted by the conversation id, and the pragmatic context, which is given by a pragmatic performative. For instance, a standard nomenclature of pragmatic performatives, which can be integrated as external (semantic) vocabulary/ontology, is e.g., defined by the Knowledge Query Manipulation Language (KQML) (Finin et al. 1993), by the FIPA Agent Communication Language (ACL), which gives several speech act theory-based communicative acts, or by the Standard Deontic Logic (SDL) with its normative concepts for obligations, permissions, and prohibitions. Depending on the pragmatic context, the message payload is used, e.g. to update the internal knowledge of the agent (e.g., add new facts or rulebases), add new tasks (goals), or detect a complex event pattern (from the internal event instance sequence). For instance, the following example shows a reaction rule that sends a complete rule base, which is loaded from a local *File* to an agent service *Remote* using JMS as transport protocol.

Example 1.

```
% Upload a rule base read from File to the host
% at address Remote via JMS
upload_mobile_code(Remote,File) :-
    % Opening a file returns an instance
    % of java.io.BufferedReader in Reader
    fopen(File,Reader),
    Writer = java.io.StringWriter(),
    copy(Reader,Writer),
    Text = Writer.toString(),
    % variable SB will encapsulate the whole content of File
    SB = StringBuffer(Text),
    % send the rule ser to the receiver agent "Remote"
    sendMsg(XID,jms,Remote,acl_inform,consult(SB)).
```

The corresponding receiving reaction rule of the remote agent is:

```
% wait for incoming messages with pragmatic context $acl_inform$
rcvMsg(XID,jms,Sender,acl_inform,[Predicate|Args]):-
    % derive the message payload, i.e. consult the received rule set to the internal KB
    derive([Predicate|Args]).
```

This rule receives incoming JMS-based messages with the pragmatic context acl_inform and derives the message content, i.e. consults the received rule base to the local knowledge base of the remote agent. It is important to note that via the conversation id several reaction rule reasoning processes might run in parallel, local to their conversation flows. Inactive reactions (conversation partitions) are removed from the system, e.g. by timeouts. Self-activations by sending a message to the receiver "self" are possible. With the pragmatic performatives it is possible to implement different coordination and negotiation protocols. For instance, if an agent does not understand the semantics of the interchanged message payload, it can inform the sender about this, using, e.g., the $acl\_not - understood$ performative, so that the sender can additionally send the semantic information, e.g. a pointer to the ontology that defines the concepts of the payload, and the receiving agent can import this ontology to its internal knowledge base.

Several expressive logic formalisms are supported by Prova [Pas07], e.g., for updating the knowledge base (transactional update logic), defining and detecting complex events (complex event algebra), handling situations/states (event calculus), as well as for reasoning (e.g., deontic logic for normative reasoning on permissions, prohibitions, obligations) and planning (abductive reasoning on plans and goals).
In summary, Prova agents can interchange event information, rules (tasks), and queries/answers in agent conversations, including information about the semantics and pragmatics of the interchanged information.

Besides Prova, Rule Responder supports rule engines such as OO jDREW, Euler, DR-Device, and Drools for implementing such query answering agents as inference services in Rule Responder.

1.3 Rule Responder Agents

With the support of Prova's agent conversations, various distributed coordination topologies can be implemented, from centralized orchestration, executed in star-like agent nodes, to decentralized ad-hoc choreography within the Rule Responder agent network. In the following, we describe a common hierarchical agent topology which represents a centralized star-like structure for virtual organizations (and many orchestrated distributed systems). Organizational Agents (OAs) act as central orchestration nodes which control and disseminate the information flow from and to their internal Personal Agents (PAs) and the External Agents/Services (EAs).

1.3.1 Organizational Agent

An Organizational Agent (OA) represents its virtual organization as a whole. An OA manages its local Personal Agents (PAs), providing control of their life cycle and ensuring overall goals and policies of the organization and its semiotic structures. OAs can act as a single point of entry to the managed sets of local PAs to which requests by EAs are disseminated. This allows for efficient implementation of various mechanisms of making sure the PAs functionalities are not abused (security mechanisms) and making sure privacy of entities, personal data, and computation resources is respected (privacy & information hiding mechanisms). For instance, an OA can disclose information about the organization to authorized external parties without revealing private information and local data of the PAs, although this data might have been used in the PAs to compute the resulting answers to the external requester.

OAs, which require high levels of expressiveness to represent the logic of cognitive agents, are implemented using the Prova Semantic Web rule engine. In the following we will discuss some of the expressive language constructs of Prova that are required to implement the Rule Responder framework.

For implementing the Rule Responder communication flows in the OAs, Prova messaging reaction rules are used. A typical coordination pattern implemented in a Rule Responder OA is the following messaging reaction rule (Prova variables start with an upper-case letter), which waits for an incoming query from an EA and delegates this query to an internal responsible PA.

```
% receive query and delegate it to another party
rcvMsg(CID,esb, Requester, acl_query-ref, Query) :-
  responsibleRole(Agent, Query),
  sendMsg(Sub-CID,esb,Agent,acl_query-ref, Query),
  rcvMsg(Sub-CID,esb,Agent,acl_inform-ref, Answer),
  ... (other goals)...
  sendMsg(CID,esb,Requester,acl_inform-ref,Answer).
```

When activated by an incoming request from an EA, e.g. an HTTP request coming from a Web form, this messaging reaction rule first selects the responsible role for the query. Then the rule sends the query in a new sub-conversation to the selected party and waits for the answer to the query. That is, the rule execution waits until an answer event message is received in the inlined sub-conversation, which activates the process flow again, e.g. to prove further 'standard' goals, e.g. with information from the received answer, which is assigned to variables in the normal logic programming way, including also backtracking to other variable assignments. Finally, in this example, the rule sends back the answer to the original requesting EA.

The selection logic for the dissemination of queries to PAs is, e.g., implemented by a standard derivation rule which, e.g., accesses, via a Prova SPARQL query built-in, an external responsibility assignment matrix (RAM) (see section 1.3.4). The following rule selects responsible agents with a SPARQL query on a triple store Web interface, where the responsibility assignment matrix is stored.

```
responsibleRole(Agent,Responsibility) :-
  SparqlServiceURL = 'http://www.csw.inf.fu-berlin.de:4039/repositories/ram?query=SPARQLQUERY',
  QueryString = '
    select ?agent ?role ?responsibility
    from <http://www.csw.inf.fu-berlin.de/agents>
    where {?agent ?role ?responsibility }
  ',
  sparql_select(SparqlServiceURL,
              QueryString,agent(Agent),role(Role),responsibility(Responsibility)).
```

While RAMs (RACI matrix, Linear Responsibility Charts, etc.) are often used to represent stable semiotic structures of virtual organization, where responsibilities are clearly defined for each role, it should be noted that Prova OAs can also implement other well-known agent coordination and negotiation mechanisms. Prova's messaging reaction rules do not require separate threads for handling multiple conversation situations simultaneously. Hence, new subconversations can be started with other PAs in parallel. This can be used for implementing, e.g., a Contract Net coordination protocol, where PAs bid for the task offered by the OA and the OA selects the best PA according to the received bids, or a publish-subscribe protocol, where PAs are selected according to their subscriptions with the OA.

1.3.2 Personal Agents

Personal Agents (PAs) assist the local entities of a virtual organization. Often these are human roles in the orgnization. But, it might be also services or applications in, e.g. a service oriented architecture. A PA runs a rule engine which accesses different sources of local data and computes answers according to the local rule-based decision logic of the PA. Depending on the required expressivness to represent the PAs rule logic arbitrary rule engines can be used as long as they provide an interface to ask queries and receive answers which are translated into the common Reaction RuleML interchange format in order to communicate with other agents.

Importantly, the PAs might have local autonomy and might support privacy and security implementations. In particular, local information used in the PA rules becomes only accessible by authorized access of the OA via the public interfaces of the PA which act as an abstraction layer supporting security and information hiding. A typical coordination protocol is that all communication to EAs is via the OA, but the OA might also reveal the direct contact address of a PA to authorized external agents which can then start an ad-hoc conversation directly with the PA [BP07]. A PA itself might act as a nested suborganization, i.e. containing itself an OA providing access to a suborganization within the main virtual organization. This can be usefull to represent nested organizational structures such as departments, project teams, service networks, where the department chair is a personal agent within the organization and at the same time an organizational chair for the department, managing the personal agents of the department.

1.3.3 External Agents

External Agents (EAs) constitute the points-of-contact that allow an external user or service to query the Organizational Agent (OA) of a virtual organization. An EA is based, e.g., on a Web (HTTP) interface that allows such an enquiry user to pose queries, employing a menu-based Web form, which gets translated to equivalent RuleML/XML message that is sent to the OA via HTTP Post or Get. An external agent – from the point of view of a Rule Responder agent organization – can be an external human agent, a service/tool, or another external Rule Responder organization, i.e. leading to cross-organizational Rule Responder communications.

1.3.4 Responsibility Assignment Matrix

As one possible way for coordination in a virtual organization the Rule Responder framework uses a 'pluggable' Responsibility Assignment Matrix (RAM) to support the OA in its selection of a PA and its optional participating profiles underneath. A RAM describes the responsibility of agent roles in completing certain tasks or deliverables in a virtual organization. A standard RAM is a RAI matrix, with

- *R*esponsible – agents who do the work to achieve the task. There is typically one role with a participation type of Responsible, although others

can be delegated to assist in the work required. Typically, the PAs are the responsible roles.
- *A*ccountable (also Approver or final Approving authority) – agent who is ultimately accountable for the correct and thorough completion of the deliverable or task, and the one to whom Responsible is accountable. Typically, this is the OA which receives the answer from the PA and further processes it before forwarding it to the EA.
- *I*nformed – the agent who is kept up-to-date on progress, often only on completion of the task or deliverable; and with whom there is just one-way communication. Typically, this is the EA who is informed about the result by the OA.

In a simple star-like Rule Responder agent topology, a single RAI matrix can be used in the OA to map an incoming query to the PA whose local knowledge base is deemed to be best suited for answering it. The RAI matrix is represented as an OWL ontology (OWL Lite) and can be used by a Rule Responder agent via querying it with the Semantic Web built-ins of Prova, binding the respective roles and their responsibilities to typed variables in the agent's rule logic. For instance, the following returns an agent's responsibility and role in a virtual organization by querying the concrete values from the imported RAM matrix (an OWL document) using the special Prova RDF query built-in (a simple RDF triple query instead of the more complex Prova SPARQL query built-in).

```
importRAM("http://www.csw.inf.fu-berlin.de/agents/ram.owl");
assigned(Agent,Responsibility,Role) :-
  importRAM(URL),
  rdf(URL,Reasoner,Agent,Role,Responsibility).
```

Many variants of the RAM with different role distinctions are possible such as RACI (with *C*onsulted agents), RASCI (with *S*upporting agents) etc. - see, e.g., table 1.1.

Table 1.1 Responsibility Assignment Matrix

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

For instance, the RAM has been split so that role responsibility assignment is done on the 'higher' level of a Group Responsibility Matrix (GRM) in the OA and on the 'lower' level of a Profile Responsibility Matrix (PRM) in the PAs. Figure 1.4 shows these two central matrices in the larger context of the Rule Responder architecture used in the WellnessRules instantiations (cf. Section 1.5.2), which has been further evolved for the PatientSupporter instantiation (cf. Section 1.5.3).

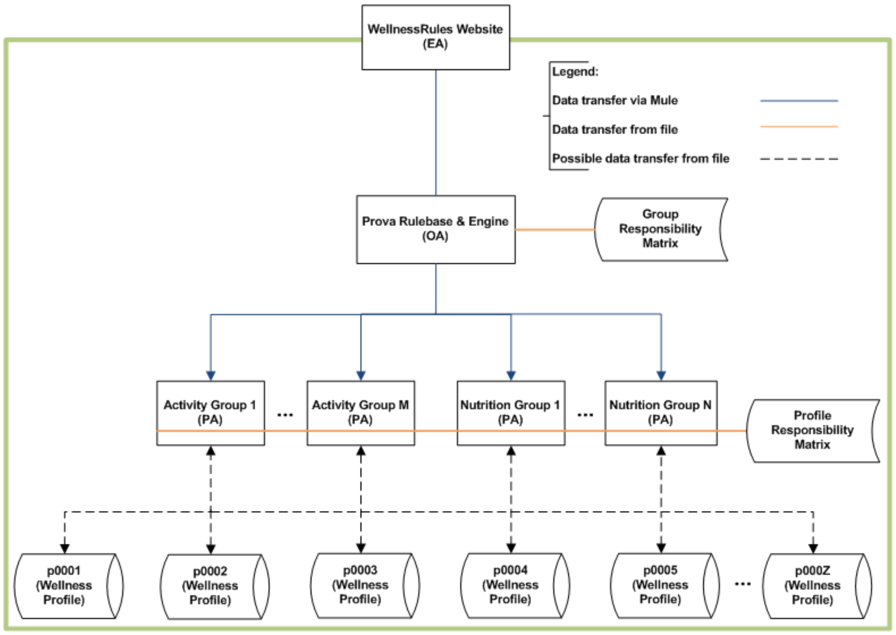

Fig. 1.4 Rule Responder architecture instantiated to WellnessRules

The GRM maps, many-to-one, relevant kinds of queries to a PA, who may represent a group. The GRM is usually specified as an OWL light ontology. The PRM lists a PA's profiles, participating in its group, along with the format (Prova, POSL, N3, etc.) each profile knowledge base is written in. The PRM is specified as an XML document.

1.4 Translation Between Rule Responder Agents

Rule Responder permits agents to use local languages and engines, only requiring that all rulebases, queries, and answers will be translated to RuleML for transmitting them to other agents over the Mule ESB.

Reaction RuleML provides a translator service framework with Web form interfaces accepting controlled natural language input or predefined selection-based rule templates for the communication with external (human) agents on the computational independent level, as well as Servlet HTTP interfaces, and Web service SOAP interfaces, wich can be used for translation into and from platform-specific rule languages such as Prova.

On the computation-independent level, online user interfaces allow external human agents issuing queries to Rule Responder agents (typically the OA) in a controlled natural language or with template-driven Web forms and receive answers. The translation between the used controlled English rule language (Attempto Controlled English [SG06]) and Reaction RuleML is based on domain-specific language translation rules in combination with a controlled English translator service.

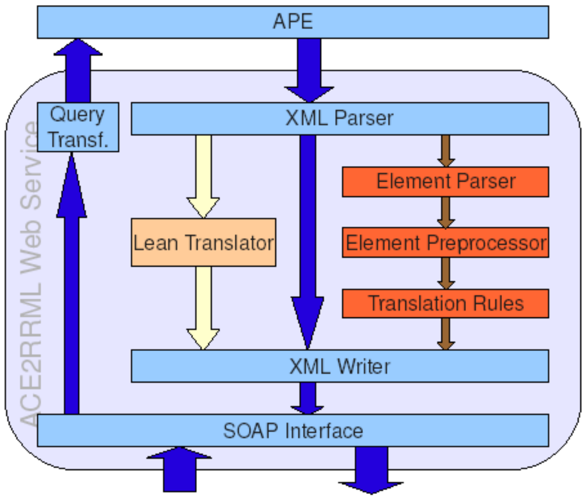

Fig. 1.5 ACE2RuleML Translator Web Service

Figure 1.5 shows the architecture of the ACE-to-Reaction RuleML translator Web service of the Rule Responder infrastructure. Queries to Rule Responder are formulated in Attempto Controlled English. The ACE2RML translator forwards the text to the Attempto Parsing Engine (APE), which translates the text into a discourse representation structure (DRS) and/or advices to correct malformed input. The DRS gives a logical/structural representation of the text. It is fed into an XML parser which translates it into a domain-specific Reaction RuleML representation of the query. Besides parsing and processing the elements of the DRS, the parser additionally employs domain-specific transformation rules to correctly translate the query into a public interface call of a Rule Responder OA.

On the platform-independent and platform-specific level, the translator services are using different translation technologies such as XSLT stylesheet, JAXB, etc. to translate from and to Reaction RuleML as a general rule interchange format. Reaction RuleML 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. The general syntax of reaction rules is as follows:

```
<Rule style="active|messaging|reasoning" eval="strong|weak|defeasible|fuzzy">
```

```
      <oid>     <! object id -->                          </oid>
      <label>   <! meta data of the rule -->              </label>
      <scope><! scope of the rule e.g. a rule module --> </scope>
      <qualification> <! e.g. priorities, validity, fuzzy levels -->
                                                          </qualification>
      <quantification> <!- e.g. variable bindings-->     </quantification>

      <on>      <! event part -->                         </on>

      <if>      <! condition part -->                     </if>
      <then>    <! (logical) conclusion part -->          </then>

      <do>      <!--  action part -->                     </do>

      <after>   <! postcondition part after action, e.g.
                    to check effects -->                  </after>
</Rule>
```

Depending on which parts of this general rule syntax are used different types of reaction rules can be expressed, e.g. if-then (derivation rules), if-do (production rules), on-do (trigger rules), on-if-do (ECA rules). For communication between distributed rule-based (agent) systems Reaction RuleML provides a general message syntax:

```
<Message>
  <oid>       <!-- conversation ID-->                     </oid>
  <protocol>  <!-- used protocol -->                      </protocol>
  <agent>     <!-- sender/receiver agent/service -->      </agent>
  <directive> <!-- pragmatic primitive, i.e. context --> </directive>
  <content>   <!-- message payload -->                    </content>
</Message>
```

Using these messages agents can interchange events (e.g., queries and answers) as well as complete rule bases (rule set modules), e.g. for remote parallel task processing. Agents can be engaged in long running possibly asynchronous conversations and nested sub-conversations using the conversation id to manage the conversation state. The protocol is used to defines the message passing and coordination protocol. The directive attribute corresponds to the pragmatic instruction, i.e. the pragmatic characterization of the message context broadly characterizing the meaning of the message.

The Reaction RuleML translator services are configured in the transport channels of the inbound and outbound links of the deployed rule engines on the ESB. Incoming Reaction RuleML messages (receive) are translated into platform-specific rulebases which can be executed by the rule engine, e.g. Prova, and outgoing rulebases (send) are translated into Reaction RuleML in the outbound channels before they are transferred via a selected transport protocol such as HTTP or JMS.

The semantic agent architecture in Rule Responder supports privacy and security implementations. In particular, local information used in the PAs becomes only accessible by authorized access via the public interfaces of the OAs which act as an abstraction layer supporting security and information hiding. To achieve this Prova supports a interface definition language which allows descriptions of the signatures of publicly accessibly rule functions together with their mode and type declarations. *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)

For instance, the interface definition for the function $add(Arg1, Arg2, Result)$ is $interface(add(-, +, +))$, i.e. the function is a public interface which expect two input arguments and returns one output argument. $add(X, 1, 1)$ would be a valid query to this public function.

External agents can access the virtual organization only via these public interfaces, which often only reveal abstracted information to authorized users and hence hide local information of the organization and its PAs.

1.5 Recent Rule Responder Instantiations

Early instantiations of Rule Responder include the Health Care and Life Sciences eScience infrastructure [Pas08], the Rule-based IT Service Level Managment, and Semantic BPM system [PB08, PK08]. Recent instantiations include multiple versions of the deployed SymposiumPlanner system [CB08], two versions of the WellnessRules prototype [BOC09], and PatientSupporter. We will here highlight the principles of Rule Responder instantiations with an emphasis on the recent ones.

1.5.1 SymposiumPlanner

SymposiumPlanner is a series of deployed applications created with Rule Responder for the Q&A parts of the official websites of the RuleML Symposia.

Rule Responder started to support the organizing committee of the RuleML-2007 Symposium [Cra07] and was further developed to assist the yearly RuleML Symposia since. These applications embody responsibility assignment, automated first-level contacts for information regarding the symposium, helping the publicity chair with sponsoring correspondence, helping the panel chair with managing panel participants, and the liason chair with coordinating organization partners.

SymposiumPlanner 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 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.

External agents and the RuleML-2008 agents can communicate by sending messages that transport queries, answers, or complete rulebases 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 Rule Responder instantation to SymposiumPlanner is further described and demoed online.[8]

1.5.2 WellnessRules

This is a Web 3.0 case study, where ontology-structured rules (including facts) about wellness opportunities are created by participants in rule languages such as Prolog and N3, and translated for interchange within a wellness community using RuleML/XML. The wellness rules are centered around participants, as profiles, encoding knowledge about their activities, nutrition, etc. conditional on the season, the time-of-day, the weather, etc. This distributed knowledge base extends fact-only FOAF profiles with a vocabulary and rules about wellness group networking.

The communication between participants is organized through Rule Responder, permitting translator-based reuse of wellness profiles and their distributed querying across engines. WellnessRules interoperates between rules and queries in the relational (Datalog) paradigm of the pure-Prolog subset of POSL and in the frame (F-logic) paradigm of N3. These derivation rule languages are implemented in the engines OO jDREW and Euler, and connected via Rule Responder to support wellness communities.

WellnessRules is a system supporting the management of wellness practices within a community based on rules plus ontologies. The idea is the following. As in Friend of a Friend (FOAF)[9], people can choose a (community-unique) nickname and create semantic profiles about themselves, here about their wellness practices, for their own planning and to network with other people supported by a system that 'understands' those profiles. As in FindXpRT [LBBM06], such FOAF-like fact-only profiles are extended with rules to capture conditional person-centered knowledge such as each person's wellness activity depending on the season, the time-of-day, the weather, etc. People

[8] `http://ruleml.org/SymposiumPlanner`

[9] `http://www.foaf-project.org/`

can use rules of various refinement levels and rule languages ranging from pure Prolog to N3, which will be interoperated through RuleML/XML [Bol07].

Interoperating with translators, WellnessRules thus frees participants from using any single rule language. In particular, it bridges between Prolog as the main Logic Programming rule paradigm and N3 as the main Semantic Web rule paradigm. The distributed nature of Rule Responder profiles, each queried by its own (copy of an) engine, permits scalable knowledge representation and processing.

WellnessRules has recently been developed to WellnessRules2, using a new kind of agents, a Service Agent (SA), for accessing Google weather data. From the point of the OA, an SA can be queried similarly to a PA. However, while a PA is a personal assistant to a human owner, an SA is just a machine agent, in WellnessRules2 acting as a wrapper for a Google service.

The Rule Responder instantations to WellnessRules are further described and demoed online.[10]

1.5.3 PatientSupporter

Patients are increasingly seeking interaction in support groups, which provide shared information and experience about diagnoses, treatment, etc. We present a Social Semantic Web prototype, PatientSupporter, that will enable such networking between patients within a virtual organization. PatientSupporter is an instantiation of Rule Responder that permits each patient to query other patients' profiles for finding or initiating a matching group.

Rule Responder's External Agent (EA) is a Web-based patient-organization interface that passes queries to the Organizational Agent (OA). The OA represents the common knowledge of the virtual patient organization, delegates queries to relevant Personal Agents (PAs), and hands validated PA answers back to the EA. Each PA represents the medical subarea of primary interest to a corresponding patient group. The PA assists its patients by advertising their interest profiles employing rules about diagnoses and treatments as well as interaction constraints such as time, location, age range, gender, and number of participants.

PAs can be distributed across different rule engines using different rule languages (e.g., Prolog and N3), where rules, queries, and answers are interchanged via translation to and from RuleML/XML. We discuss the implementation of PatientSupporter in a use case where the PA's medical subareas are defined through sports injuries structured by a partonomy of affected body parts.

PatientSupporter uses ontologies and rules for organizing geographically distributed patients – here, suffering from sports injuries – into virtual sup-

[10] `http://ruleml.org/WellnessRulesandhttp://ruleml.org/WellnessRules2`

port groups around classes of an ontology of injuries – here, a sports-injury partonomy. The prototype is designed to help patients with a similar sports injury to interact with a virtual support group having that common interest. Patients in an online PatientSupporter virtual organization create their semantic profile referring to classes in a disease ontology – here a partonomy of body parts affected by sports injuries. Profiles contain rules about diagnoses and treatments as well as interaction constraints such as time, location, age range, gender, and number of participants. A patient can pose queries against the semantic profiles of other patients in his or her virtual organization to find or initiate a matching group.

PatientSupporter allows patients to have their profiles expressed in either Pure Prolog (Logic Programming rules) or N3 [BLCK$^+$08] (Semantic Web rules). Providing these quite different rule language paradigms permit patients to choose the language that best suits them. Rule Responder handles the interoperation between the rule languages of different patients using translators to and from RuleML/XML as the interchange format [BTW01, Bol07].

Patients using the PatientSupporter Social Semantic Web portal are able to initiate the virtual support group about their sports injury on a global scale. They also benefit from PatientSupporter's interoperation facility in the background – to transform patient profiles between Pure Prolog and N3 through RuleML/XML. The system employs a partonomy of sports-injury-affected body parts (a 'body partonomy'), which makes it easy for patients to navigate hierarchically up or down to increase recall or precision, respectively. A patient's queries invoke other patients' interaction rules, allowing him or her to narrow down the search in a step-wise fashion. All of this saves a patient from browsing through a large set of irrelevant patient profiles and permits him or her to efficiently converge on a first Skype call.

The Rule Responder instantation to PatientSupporter is further described and demoed online.[11]

1.5.4 Reputation Management System

The Rule Responder reputation management system [APM10] is based on distributed Rule Responder rule agents, which use rules for implementing the reputation management functionalities as rule agents, and which use Semantic Web ontologies for representing simple or complex multi-dimensional reputation objects. This Semantic Web reputation ontology model enables reputation portability, eases the management of reputation data, mitigates risks in open environments, and enhances the decision making process in the reputation processing agents. The reputation management system computes,

[11] `http://ruleml.org/PatientSupporter`

manages, and provides reputation about entities which act on the Web. It is implemented as a *Reputation Processing Network (RPN)* consisting of *Reputation Processing Agents (RPAs)* that have two different roles:

1. *Reputation Authority Agents (RAAs)*: Act as reputation scoring services for the reputee entities whose Reputation Objects (ROs) are being considered or calculated in the agents' rule-based Reputation Computation Services (RCSs). An RCS runs a rule engine which accesses different sources of reputation (input) data from the reputors about an entity and evaluates an RO based on its declarative rule-based computational algorithms and contextual information available at the time of computation.
2. *Reputation Management Agents (RMAs)*: Act as a reputation trust center offering reputation management functionalities. An RMA manages the local RAAs providing control of their life-cycle in particular, and also ensuring goals such as fairness. It might act as a Reputation Service Provider (RSP) which aggregates reputations from the reputation scores of local RAAs. Based on the final calculated reputation, it might also perform actions, e.g. compute trust-worthiness, make automated decisions, or trigger reactions. It also manages the communication with the reputors, collecting data about entities from them, generates reputation data inputs for the reputation scoring, and distributes the data to the RAAs. It might also act as central point of communication for the real reputee entities (e.g., persons) giving them legitimate control over their reputation and allowing entities the governance of their reputations.

For instance, the following rule makes decisions on the basis of rules which haven been authored by different persons and only applies those rules from trusted authors.

```
%simplified decision rules of an agent
@author(dev22) r2(X):-q(X).
@author(dev32) r2(X):-s(X).
q(2).
s(-2).

% for simplicity this is a fact, but could be also a complex rule
% which computes the trust value from the reputation value of dev22
trusted(dev22).

% Author dev22 is trusted but dev32 is not, so one solution is found: X=2
p1(X):-
 @author(A)
 r2(X) [trusted(A)].

% for all query
:-solve(p1(X1)).
```

This example uses metadata annotations on rules for the head literals $r2/1$ and on the literal $r2(X)$ in the body of the rule for $p1(X)$. Since variable A in $@author(A)$ is initially free, it gets instantiated from the matching target rule(s). Once A is instantiated to the target rule's $@author$ annotation's value ($dev22$, for the first $r2$ rule), the body of the target rule is dynamically non-destructively modified to include all the literals in the guard

$trusted(A)$ before the body start, after which the processing continues. Since $trusted(dev22)$ is true but $trusted(dev32)$ is not, only the first rule for predicate r2 is used and so one solution $X1 = 2$ is returned by $solve(p1(X1))$.

The agent-based approach for an online reputation management ensures efficient automation, semantic interpretability and interaction, openness in ownership, fine-grained privacy and security protection, and easy management of semantic reputation data on the Web.

1.5.5 Semantic Complex Event Processing Agent Network

The Event Processing Network (EPN) [KJP09] consists of Semantic Event Processing Agents (EPA) implemented as distributed Prova inference services which detect complex events using Prova's rule-based Semantic Complex Event Processing (SCEP) logic. [TP09]. The multi-agent approach allows for a highly-available distributed implementation with redundant Event-Calculus based state processing where events are processed concurrently in the EPN.

1.6 Conclusion

Rule Responder is a framework for specifying virtual organizations as semantic multi-agent systems. Its recent instantiations include SymposiumPlanner-2007 through SymposiumPlanner-2010, WellnessRules, WellnessRules2, and PatientSupporter. The software is available open source in Sourceforge[12].

It has used three kinds of agents, OAs, PAs, EAs, but has also accessed more conventional kinds of data and programs wrapped as services. This access includes vCard, RDF triple stores, and – in the WellnessRules2 instantiation – Google weather data. To accommodate for this on the architectural level, a fourth kind of agents, Computing Agents (CAs), has recently been added to the Rule Responder framework.

Characteristics of Rule Responder include

- the coverage of the distributed processing spectrum from Web Services to agents in one framework
- the recursive (holonic) modeling of a virtual organization of services and agents as a single agent,
- the use of ESBs, especially Mule, as a Semantic and Pragmatic Web infrastructure,

[12] http://mandarax.svn.sourceforge.net/viewvc/mandarax/PragmaticAgentWeb

- the use of Semantic-Pragmatic Web rules as the main knowledge representation, complemented by ontologies,
- the introduction of PAs as human-assisting agents within a virtual organization, complementing the usual self-contained CAs,
- the design of a 'pluggable' agent-finding mechanism from role assignment to Semantic Service discovery.

The Rule Responder framework, with its increasing number of users and engines (Prova, OO jDREW, DR-Device, Euler, and Drools), is thus being proposed as a reference architecture for distributed knowledge representation and processing.

Current work extends the static Role Assignment Matrix (and Group Responsibility Matrix) with dynamic matchmaking algorithms. In the present system, a PA is described by one concept which can be assigned to topics of incoming queries. In the new system, a PA will be described by a general combination of concepts, ranging form a conjunction to a tree or DAG, to a (weighted) Directed Labeled Graph (DLG), i.e. a kind of (weighted) RDF Graph. While role assignment consists of look-up of a unique PA in the RAM, matchmaking algorithms compute a ranked list of PAs based on similarities. A Prova-based OA can pick the PAs in the ranked order, so that in case of lacking or insufficient answers by a PA backtracking will occur and the next one tried.

1.7 Acknowledgement

The international Rule Responder initiative has greatly helped us with work leading to this chapter. In particular, we want ot thank Alexander Kozlenkov, Benjamin Craig, Taylor Osmun, Derek Smith, Omair Shafiq, Mahsa Kiani, Kia Teymourian, Rehab Alnemr, Irfan ul Haq, Nick Bassiliades, Stratos Kontopoulos, and Kalliopi Kravari.

References

[APM10] Rehab Alnemr, Adrian Paschke, and Christoph Meinel. Enabling reputation interoperability through semantic technologies. In *ACM International Conference on Semantic Systems*. ACM, 2010.

[BC07] Harold Boley and Elizabeth Chang. Digital Ecosystems: Principles and Semantics. In *Proc. IEEE Intl. Conf. Digital Ecosystems and Technologies, Cairns, Australia*, February 2007.

[BLCK$^+$08] Tim Berners-Lee, Dan Connolly, Lalana Kagal, Yosi Scharf, and Jim Hendler. N3Logic: A Logical Framework For the World Wide Web. *Theory and Practice of Logic Programming (TPLP)*, 8(3), May 2008.

[BOC09] Harold Boley, Taylor Michael Osmun, and Benjamin Larry Craig. Social Semantic Rule Sharing and Querying in Wellness Communities. In Asunción Gómez-Pérez, Yong Yu, and Ying Ding, editors, *ASWC*, volume 5926 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2009.

[Bol07] Harold Boley. Are Your Rules Online? Four Web Rule Essentials. In A. Paschke and Y. Biletskiy, editors, *Proc. Advances in Rule Interchange and Applications, International Symposium (RuleML-2007), Orlando, Florida*, volume 4824 of *LNCS*, pages 7–24. Springer, 2007.

[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.

[BTW01] Harold Boley, Said Tabet, and Gerd Wagner. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Proc. Semantic Web Working Symposium (SWWS'01)*, pages 381–401. Stanford University, July/August 2001.

[CB08] Benjamin Larry Craig and Harold Boley. Personal Agents in the Rule Responder Architecture. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, *RuleML*, volume 5321 of *Lecture Notes in Computer Science*, pages 150–165. Springer, 2008.

[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.

[KJP09] Alexander Kozlenkov, David Jeffery, and Adrian Paschke. State management and concurrency in event processing. In *DEBS*, 2009.

[Koe67] A. Koestler. *The Ghost in the Machine*. Hutchinson & Co, London, 1967.

[LBBM06] Jie Li, Harold Boley, Virendrakumar C. Bhavsar, and Jing Mei. Expert Finding for eCollaboration Using FOAF with RuleML Rules. In *Montreal Conference of eTechnologies 2006*, pages 53–65, 2006.

[Pas06] Adrian Paschke. A typed hybrid description logic programming language with polymorphic order-sorted dl-typed unification for semantic web type systems. *CoRR*, abs/cs/0610006, 2006.

[Pas07] A. Paschke. *Rule-Based Service Level Agreements - Knowledge Representation for Automated e-Contract, SLA and Policy Management*. Idea Verlag GmbH, Munich, 2007.

[Pas08] Adrian Paschke. Rule responder hcls escience infrastructure. In *ICPW '08: Proceedings of the 3rd International Conference on the Pragmatic Web*, pages 59–67, New York, NY, USA, 2008. ACM.

[PB08] Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated sla management. *Decis. Support Syst.*, 46(1):187–205, 2008.

[PK08] Adrian Paschke and Alexander Kozlenkov. A rule-based middleware for business process execution. In *Multikonferenz Wirtschaftsinformatik*, 2008.

[SG06] Geoff Sutcliffe and Randy Goebel, editors. *Proceedings of the Nineteenth International Florida Artificial Intelligence Research Society Conference, Melbourne Beach, Florida, USA, May 11-13, 2006*. AAAI Press, 2006.

[TP09] Kia Teymourian and Adrian Paschke. Towards semantic event processing. In *DEBS*, 2009.

[WCB01] M. Welsh, D. Culler, and E. Brewer. SEDA: An Architecture for Well Conditioned, Scalable Internet Services. In *Proceedings of Eighteeth Symposium on Operating Systems (SOSP-18)*, Chateau Lake Louise, Canada, 2001.

[Woo01] M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2001.