Loosely-Coupled and Event-Messaged Interactions with Reaction RuleML 1.0 in Rule Responder

Zhili Zhao[1], Kia Teymourian[1], Adrian Paschke[1], Harold Boley[2], Tara Athan[3]

[1] Freie Universität Berlin, Germany
{paschke, zhili, teymourian} AT inf.fu-berlin.de
[2] Information and Communications Technologies, National Research Council Canada
Fredericton, NB, Canada
harold.boley AT nrc.gc.ca
[3] Athan Services, W Lafayette, IN, USA
taraathan AT gmail.com

Abstract. Reaction RuleML is one of the two major subfamilies of RuleML and acts as an interchange format for reactive rules and rule-based event-processing languages. Exemplified with a recent instantiation of Rule Responder, a rule-based inference agent middleware, we demonstrate the event messaging features of Reaction RuleML, which supports loosely-coupled interface-based interaction using rule signatures and decoupled communication via event messages.

1 Introduction

As one of the two major subfamilies of RuleML[1], Reaction RuleML[2] presents a general compact rule interchange format for reaction rules, which are used to declaratively specify the reactive and behavioral logic of distributed systems and dynamic (Web-based) environments [17]. RuleML has broad coverage and is designed as an interchange language for the major kinds of (Web) rules. The RuleML family's top-level distinction is *Deliberation rules* vs. *Reaction rules* [3]. *Deliberation rules* permit knowledge derivation and subsume further languages such as Hornlog (hence Datalog), which (syntactically) specialize to condition-less *Fact* and conclusion-less *Query* languages (the latter subsuming *Integrity Constraint (IC)* languages). On the other hand, *Reaction rules* focus on event-driven (re)actions in distributed and dynamic environments.

Reaction RuleML is intended as a common standard for representing reactive rules and rule-based complex event processing (CEP) in a platform independent XML markup language. It provides several layers of expressiveness for adequately representing reactive logic and for interchanging events (queries, actions, event data) and rules. As a whole, Reaction RuleML is characterized by the following features:

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

1. Reaction RuleML Metamodel, Semantic Types and Data Queries. Reaction RuleML is based on a metamodel and 'pluggable' ontologies and defines general concepts such as space, time, event, action situation, process, and agent in a modularized ontological top-level structure, with a left to right vertical order in the top-level ontologies. Therefore, it is possible for Reaction RuleML to support distributed and modularized knowledge bases through direct coupling via key references within a KB, `iri` pointers, and support for query languages.
2. Rule Interface Descriptions with Semantic Profiles and Signatures. Reaction RuleML separates the interface of a rule from its implementation. The interface describes the functional and non-functional (semantic) properties of a rule. The implementation, on the other hand, requires more flexibility and can be modified without any change of its interface.
3. Reaction RuleML Messaging. The interface description language of Reaction RuleML allows for loosely-coupled interaction with distributed inference services and agent KBs. Based on event messaging, Reaction RuleML also supports decoupled communication via event messages that are produced and published as Reaction RuleML serializations, e.g. on event streams or event clouds.

In this paper, exemplified with a recent instantiation of Rule Responder[3] [16, 15, 2], we demonstrate the distributed event-messaging interactions of Reaction RuleML 1.0 in loosely-coupled and de-coupled distributed rule-based agents. Reaction RuleML acts as a standardized interface description language and interchange format between these semantic agents which run their own platform specific rule engines and rule-based knowledge base (KB) at their core. The rest of the paper is organized as follows: Section 2 introduces Reaction RuleML and its reference application Rule Responder. In Section 3 we present the semantic interpretation and translation between Reaction RuleML as a standardized rule interchange language and several platform specific rule languages as well as the platform independent controlled English ACE. Section 4 presents how distributed event messaging supports loosely-coupled interaction with inference services/agents. Section 5 deals with decoupled communication via event messages. Finally, we conclude the paper with a summary in Section 6.

2 Reaction RuleML 1.0

Reaction rules are concerned with the invocation of actions in response to events and actionable situations [14]. They state the conditions under which actions must be taken and describe the effects of action executions. In the last decades various reaction rule languages and rule-based event processing approaches have been developed, which for the most part have been advanced separately. The Reaction RuleML standard[4] addresses four major reaction rule types: *Produc-*

[3] http://responder.ruleml.org
[4] http://reaction.ruleml.org/

tion Rules (Condition-Action rules), *Event-Condition-Action (ECA) rules*, *Rule-based Complex Event Processing (CEP)* (CEP reaction rules, (distributed) event messaging reaction rules, query reaction rules etc.), *Knowledge Representation (KR)* Event/Action/Situation Transition/Process Logics and Calculi

Reaction rules are defined by a general `Rule` element which can be specialized in the different Reaction RuleML branches to the four major types of reaction rules (and variants of these types). The following example shows the most general rule syntax of RuleML with of focus on Reaction RuleML. We use 1- or 2-letter indicators for syntax from Deliberation (D), Reaction (R), or Deliberation+Reaction (DR) RuleML.

```
<Rule @key @keyref @style>

 <!-- rule info and life cycle management, modularization -->

     <meta>   <!-- DR: (semantic) metadata of the rule -->                </meta>
     <scope>  <!-- R: scope of the rule e.g. a rule module -->             </scope>

 <!-- rule interface description -->

     <evaluation> <!-- R: intended  semantic profiles -->                  </evaluation>
     <signature>  <!-- R: rule interface signature and modes -->          </signature>

 <!-- rule implementation -->

     <qualification>  <!-- R: e.g. qualifying rule declarations, e.g.
                          priorities, validity, strategy -->             </qualification>
     <quantification> <!-- DR: quantifying rule declarations,
                          e.g. variable bindings -->                     </quantification>

     <on>      <!-- R: event part -->                                     </on>
     <if>      <!-- DR: condition part -->                               </if>
     <then>    <!-- D: (logical) conclusion part -->                     </then>
     <do>      <!-- R: action part -->                                   </do>
     <after>   <!-- R: postcondition part after action,
                    e.g. to check effects of execution -->               </after>
     <else>    <!-- DR: (logical) else conclusion -->                    </else>
     <elsedo>  <!-- R: alternative/else action,
                    e.g. for default, exception handling -->             </elsedo>
</Rule>
```

Rule Responder[5] [16, 15, 2] is a reference application of Reaction RuleML. It is supporting distributed semantic multi-agent systems and rule-based inference services that run rule engines at their core and communicate using (Reaction) RuleML as a standardized rule interchange format. The Rule Responder Technical Group of RuleML is focused on implementing use cases that require the interchange of rule sets and support querying the distributed rule inference services. To implement different distributed system/agent topologies and semiotic structures with their negotiation/coordination mechanisms, Rule Responder instantiations employ three core classes of agents - Organizational Agents (OA), Personal Agents (PAs), and External Agents (EAs). An OA represents goals and strategies shared by its virtual organization (of agents) as a whole, using a rule base that describes its policies, regulations, opportunities, etc. OAs hence might act as centralized nodes in star-like distributed coordination networks.

[5] http://ruleml.org/RuleResponder/

They often follow an orchestration style execution logic where the OA is a centralized authority which orchestrates the other PAs. A PA assists a group or person/agent of the organization, semi-autonomously acting on their behalf by using a local knowledge base of rules defined by the entity. In decentralized distributed networks the PAs itself might communicate with each other following e.g. a choreography style coordination, e.g. for distributed problem solving. 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 of Rule Responder. The agents employ ontologies in their rule-based knowledge bases to represent semantic domain vocabularies, normative pragmatics and pragmatic context of conversations and actions, as well as the organizational semiotics.

Since the Rule Responder framework has been conceived [16], many instantiations have been developed such as the Health Care and Life Sciences eScience infrastructure [11], Rule-based IT Service Level Management and the Rule Based Service Level Agreement (RBSLA) language [13], Semantic Business Process Management (BPM) [18, 12], WellnessRules(2) [1], PatientSupporter, and SymposiumPlanner systems[6].

In this paper, we will employ the SymposiumPlanner 2011 to demonstrate the distributed event-messaging interactions in Rule Responder. SymposiumPlanner is a series of Rule Responder instantiations for the Questions&Answers (Q&A) sections of the websites of the RuleML Symposia since 2007.

3 Translator Service Framework

The design of Rule Responder follows the spirit of the OMG's Model Driven Architecture (MDA) approach [11, 15]:

1. On the computational independent level rules are engineered in a Rule Manager user interface in a natural controlled English language using blueprint templates and user-defined vocabularies and domain-specific translation rules.
2. The rules are mapped and serialized in Reaction RuleML which is used as platform independent rule interchange format to interchange rules between Rule Responder inference services (agents) and arbitrary other rule execution environments.
3. The Reaction RuleML rules are translated into the platform specific rule language for execution.

Rule Responder provides a translator service framework with Web form interfaces accepting controlled natural language inputs or predefined selection-based rule templates for the communication with external (human) agents on the computational independent level, as well as HTTP Rest and Web service interfaces,

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

which can be used for translation into and from Reaction RuleML. In Rule Responder SymposiumPlanner 2011[7], we also implemented a user client supporting queries in Attempto Controlled English (ACE) [5], which is a rich subset of controlled English designed to serve as a knowledge representation language. The demonstration of the SymposiumPlanner 2011 user client can be found at[8]. Before sending them to Rule Responder, the queries are translated into a discourse representation structure (DRS) by the Attempto Parsing Engine (APE)[9]. It is then fed into an XML parser which translates it into Reaction RuleML by an ACE2RML translator, which makes use of domain specific semantic vocabularies and domain rules [21].

On the platform-independent and platform specific level, Reaction RuleML can be translated or mapped into several domain specific reaction rule languages, which are run by platform specific rule engines, such as: Prova[10], OO jDREW[11], Emerald[12], Euler, etc. The translator services are using different translation technologies such as XSLT stylesheet, JAXB, etc. to translate from and to Reaction RuleML and are configured in the transport channels of the inbound and outbound links of the deployed rule engines on the ESB. That is, incoming Reaction RuleML messages (receive) are translated into platform-specific rule bases which can be executed by different platform specific rule engines, e.g. Prova, and outgoing rule bases (send) are translated into Reaction RuleML in the outbound channels before they are transferred via a selected transport protocol such as HTTP or JMS, etc.

For example, a user query in ACE format: "Which papers are full and accepted?", which is used to get all full papers accepted by RuleML2011@IJCAI[13] is firstly translated into Reaction RuleML:

```
<?xml version="1.0" encoding="GBK"?>
<RuleML xmlns="http://www.ruleml.org/1.0/xsd"
        xsi:schemaLocation="http://www.ruleml.org/reaction/1.0/xsd
                            http://ibis.in.tum.de/research/ReactionRuleML/1.0/rr.xsd"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <oid>
    <Ind>Generated message from ACE text "Which papers are full and accepted?".</Ind>
  </oid>
  <Message directive="query-sync">
    <oid>
      <Ind>RuleML-2011-IJCAI</Ind>
    </oid>
    <protocol>
      <Ind>esb</Ind>
    </protocol>
    <sender>
      <Ind>User</Ind>
    </sender>
    <receiver>
```

[7] http://ruleml.org/SymposiumPlanner/documentation.html
[8] http://de.dbpedia.org/redirects/ruleml/ACE2ReactionRuleML/index.jsp
[9] http://attempto.ifi.uzh.ch/site/
[10] http://www.prova.ws/
[11] http://www.jdrew.org/oojdrew/
[12] http://lpis.csd.auth.gr/systems/emerald/
[13] http://www.defeasible.org/ruleml2011/

```
      <Ind>RuleML-2011-IJCAI</Ind>
    </receiver>
    <content>
      <Atom>
        <Rel>getPapers</Rel>
        <Ind>full</Ind>
        <Ind>accepted</Ind>
        <Var>B</Var>
      </Atom>
    </content>
  </Message>
</RuleML>
```

This example above also indicates the general message syntax of a Reaction Message [17]. In Reaction RuleML 1.0, each event message (the `Message` element) consists of a conversation identifier (the `oid` element), a pragmatic context description (the `directive` attribute), a transport protocol (the `protocol` element), such as HTTP, JMS, SOAP, etc., a sender (the `sender` element)/receiver (the `receiver` element) agent of the message and a message payload (the `content` element). When a message is sent from an External Agent, Rule Responder picks up the message, translates into a domain specific rule language and then sends it to a target agent. For example, the message of Reaction RuleML mentioned above is translated into a Prova message via XSLT sheet in SymposiumPlanner 2011, shown as follows:

```
[httpEndpoint:3,esb,httpEndpoint,query,[getPapers,full,accepted,<2901>]].
```

Each Prova message describes the messages which are received and sent by Prova agents and consists of constants, variables, or lists. For more information, see the Prova 3.0 Users Guide[14]. After the above Prova message is processed in the Prova rule engine, the resulting answer will be translated to Reaction RuleML before sending it to other agents.

Rule Responder's translation framework also supports the elementary translation between Drools[15] and Reaction RuleML. Drools is a business rule management system (BRMS) with a forward chaining production rule engine [20]. The production rule pattern of "when-then" in Drools can be represented by the pattern of "if-do" in Reaction RuleML, as shown in Figure 1. For more implementation details of the translation see [6].

4 Loosely-Coupled Interaction

Reaction RuleML allows distributed event messaging interactions in loosely-coupled and decoupled distributed rule-based systems such as Web inference services and semantic agents. In this Section we will demonstrate how event messaging interaction plays an important role in Rule Responder.

The loosely-coupled interaction leads to a resilient relationship between distributed agents with some kind of exchange relationship. Each agent makes its

[14] http://www.prova.ws/index.html?page=documentation.php
[15] http://www.jboss.org/drools

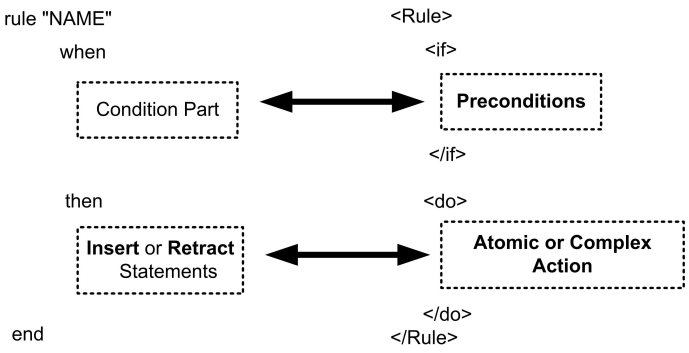

Fig. 1. The Mappings between Drools and Reaction RuleML

requirements explicit and makes use of the public interface definitions of other agents for communicating with them, i.e., an agent publishes an interface definition (containing the public rule signatures), which can be accessed in one or many concrete ways by other agents - typically by a query to the agent using one of its public rule interface signatures. Instead of queries and answers, also an interchange of complete rules and rule bases as mobile rule code to an agent is possible. Their loosely-coupled dependency and their intended interpretation and execution semantics is specified by the interface and brings flexibility that a change in the underlying rule implementation does not necessarily require a change in the rule signature, except if the rule signature itself changes. Moreover, while the interfaces might be published publicly and can be queried by requesting agents, the concrete implementation of the rule base might be hidden and privately encapsulated in the knowledge base of the agent. Figure 2 demonstrates the loosely-coupled interaction in Rule Responder.

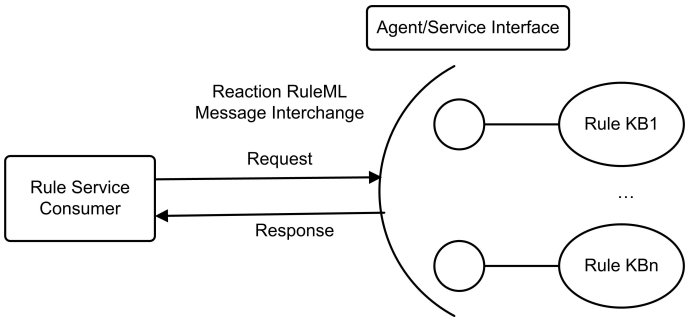

Fig. 2. Loosely-Coupled Communication via Messages to Agent Interface

Reaction RuleML 1.0 employs the Reaction RuleML Interface Description Language (RuleML IDL) [16] for describing functional and non-functional prop-

erties of a rule inference service and its rule-based KB. The functional description among others contains the signatures of public rule functions together with their term modes (input, output or arbitrary terms) and type declarations. For example, the signature of the aforementioned query of "getPapers" of SymposiumPlanner 2011 can be described as follows:

```
<signature>
  <Atom>
    <Rel>getPapers</Rel>
    <Var type="java://java.lang.String" mode="+"/>
    <Var type="java://java.lang.String" mode="+"/>
    <Var type="java://java.lang.String" mode="-"/>
  </Atom>
</signature>
```

Reaction RuleML distinguishes between the interface of a rule base or rule and its implementation. The signatures are defined in the interface either directly together with the implementation in one `<Rule>` or for better modularization and information hiding separated from the implementation of the rule on the level of a RuleML rule base `<Rulebase>` and asserted rule module `<Assert>`. The following example illustrates the use of such signature declarations in the interface descriptions of rules and distinguishes the interface from the implementation referring from the interface to the implementation via an XML key-keyref connection.

```
<-- rule interface with two alternative interpretation semantics and a signature.
    The interface references the implementation identified by the corresponding key -->
<Rule keyref="r1">
 <evaluation index="1">
   <!-- WFS semantic profile define in the metamodel -->
   <Profile type="ruleml:Well-Founded-Semantics" direction="backward"/>
 </evaluation>
 <evaluation index="2">
  <!-- alternative ASS semantic profile define in the metamodel -->
  <Profile type="ruleml:Answer-Set-Semantics" direction="backward"/>
 </evaluation>
 <!-- the signature  defines the queryable head of the backward-reasoning rule -->
 <signature>
   <Atom><Rel>getPapers</Rel><Var mode="+"/><Var mode="+"/><Var mode="-"/></Atom>
 </signature>
</Rule>

<!-- implementation of rule 1 which is interpreted either by WFS or by ASS semantics
     and onyl allows queries according to it's signature definition. -->
<Rule key="r1" style="reasoning">
  <if>...  </if>
  <then>
    <Atom><Rel>getPapers</Rel><Var>Type</Var><Var>Status</Var><Var>Papers</Var></Atom>
  </then>
</Rule>
```

The signatures can be also defined or just referred to via key-keyref in the `<signature>` of a `<Rulebase>`.

This enables a loosely-coupled interaction with the inference service / agent, where queries can be posed against the public interface `signature` and interpreted with the intended semantics `evaluation`. Therefore, the interface also defines the applicable evaluation semantics, which in the example uses predefined semantic `Profile`s from the RuleML metamodel. This is in particular

useful for mobile code, i.e. rule bases which are uploaded to an inference service, since the underlying rule engine needs to support the intended semantics. It is also useful for verification and validation [8, 10, 7, 4], explanations, and proofs of answers to queries which are dependent on the applied semantics.

During the communication, Rule Responder represents the interactions between distributed agents via constructs for asynchronously sending and receiving event messages. Therefore it uses Reaction RuleML's support for messaging in the CEP Reaction RuleML branch. For sending and receiving (event) messages, Reaction RuleML 1.0 supports serial messaging CEP reaction rules that `<Receive>` and `<Send>` events in arbitrary combinations. A serial (messaging) reaction rule starts with a receiving event (`<on>`) followed by any combination of conditions (`<if>`), events (`<Receive>`), and actions (`<Send>`) in the body of the rule for expressing complex event processing logic. This flexibility with support for modularization and aspect-oriented weaving of reactive rule code is in particular useful in distributed systems where event processing agents communicate and form a distributed event processing network, as e.g. in the following example:

```
<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <do><Send> query agent 2 for regular products in a new sub-conversation </Send></do>
  <on><Receive> receive results from sub conversation with agent 2   </Receive></on>
  <if> prove some conditions, e.g. make decisions on the received data </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
```

These Reaction RuleML messaging constructs can directly map to the messaging reaction rules in Prova with: *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, shown as follows:

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

where *XID* is the conversation identifier. *Protocol* defines the communication protocol. *From* denotes the source of the message. *Performative* describes the pragmatic context in which the message is sent. And *Payload—Context* denotes the actual content of the event message.

The event messages between distributed agents conversation invoke the rule functions of the receiving agents if there exists a matching rule interface. For instance, the example given in Section 3 indicates that the receiver agent "RuleML-2011-IJCAI" needs to specify an appropriate signature for "getPapers" queries. In SymposiumPlanner 2011, the receiver "RuleML-2011-IJCAI" agent is a Prova engine, which implements the interface definition via its platform specific rule syntax: interface(getPapers(Type, Status, Papers),getPapers("+", "+", "-"), "return related papers of RuleML-2011@IJCAI."). This public interface can be queried in backward-reasoning style in a Prova engine and a "no_further_answers" message will be sent to the sender if there is no suitable public interface is found:

```
% look-up interface
processMessage(XID,From,Primitive,[X|Args]):-
  not(interface([X|Args],ModeDeclarations,Description)),
  sendMsg(XID,esb,From,"answer", noPublicInterface(interface([X|Args]))),
  sendMsg(XID,esb,From,"no_further_answers", [X|Args]),
  fail().
```

The implementation of a rule interface can be implemented by arbitrary rule agents, which might have different levels of expressiveness. For example, the implementation of the interface "getPapers" in Prova is shown as follows:

```
getPapers(XID, Type, Status, Papers):-
  sysTime(CT),

  @paperType(Type)
  getAcceptedPapers(Papers)[validate(CT)].

validate(CT) :-
   compare(CT,'>',datetime(2011,5,31,0,0,0)).

@paperType(full)
getAcceptedPapers(Papers) :-
  QueryString = '
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX dc: <http://purl.org/dc/elements/1.1/>
    PREFIX swrc: <http://swrc.ontoware.org/ontology#>
    SELECT ?paper ?title
      FROM <http://de.dbpedia.org/redirects/ruleml/ruleml2011.rdfs>
      WHERE {
        ?paper a ?type .
        ?paper dc:title ?title .
                FILTER (?type = <http://ruleml.org/ontology#FullPaper> ) .
      }
  ',
  sparql_select(QueryString,[title(Papers)]).

@paperType(short)
getAcceptedPapers(Papers) :-
  QueryString = '
    PREFIX foaf: <http://xmlns.com/foaf/0.1/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX dc: <http://purl.org/dc/elements/1.1/>
    PREFIX swrc: <http://swrc.ontoware.org/ontology#>
    SELECT ?paper ?title
      FROM <http://de.dbpedia.org/redirects/ruleml/ruleml2011.rdfs>
      WHERE {
        ?paper a ?type .
        ?paper dc:title ?title .
                FILTER (?type = <http://ruleml.org/ontology#ShortPaper> ) .
      }
  ',
  sparql_select(QueryString,[title(Papers)]).

...
```

Prova supports modularization of its knowledge base and allows constructing metadata based views on the knowledge base, so called scopes. For example, the annotation "@paperType(Type)" on the followed goal literal "getAcceptedPapers(Papers)" is a scope constraint which applies the goal literal only on the target rule with matching metadata ("@paperType(full)", "@paperType(short)", etc.) during unification, i.e. there must be a match between the value given for the annotation @paperType and the value listed for the key in the target rule

of getAcceptedPapers. In the example, it would bind the metadata annotation values "full", "short", etc. to the variable "Type". The metadata can act as an explicit scope for constructive queries (creating a view) on the knowledge base and enables scoped (meta) reasoning with the semantic annotations. Besides, Prova supports literal guards which act as additional precondition constraints. In the above example, the goal literal is only available "after 31st, May, 2011", which is defined by the guard "[validate(CT)]" and its implementation as a rule "validate(CT):- compare(CT,'¿',datetime(2011,5,31,0,0,0))."

Reaction RuleML 1.0 provides corresponding expressiveness for metadata annotations `<meta>`, scope definitions `<scope>` and guards `<guard>`, which can be defined on the global level of a rule module and rule base as well as on the level of rules and literals. Scopes defined on the level of rule bases/modules set the context in which the knowledge of the rule base/module is applied, i.e. all queries and goal literals automatically apply within the scope. Nested scopes can be defined which override and specialize the outer (global) scopes, e.g. a scope within a rule `<Rule>` and on a particular goal literals `<Atom>` within the body of a rule. Scopes are e.g. useful to implement and distinguish different (behavioral) roles of a rule-based agent as scoped rule modules in the agent's knowledge base. Scopes are also useful to implement reactive workflow logics and (transactional) update logics [9].

5 Decoupled Interaction

The event messaging in Rule Responder also enables completely decoupled interaction via standardized Reaction RuleML event messages. Here some agents are event producers which publish events, e.g. in an event stream or in an event cloud / data source, irrespective of the event consumers. Other agents are consumers which try to detect and consume relevant events on those streams applying rule-based complex event processing techniques. That is, in difference to the loosely-coupled interaction, where the events are sent directly to other agents and the interaction with them takes place in a loosely-coupled way according to their interface definitions, the events in the decoupled scenario are just published, but there is no direct interaction with the consumers of those events.

For the decoupled interaction the message content itself is an event. Like for rules, the generic syntax pattern for an `Event` again distinguishes between the general event information, the event interface with the signature defining the event pattern (event type) and the concrete implementation in terms of an event instance.

```
<Event @key @keyref @iri @type>
  <!-- event info and life cycle management, modularization -->
     <oid>    <!-- R: event instance object id  -->                          </oid>
     <meta>   <!-- R: (semantic) metadata of the event -->                   </meta>
     <scope>  <!-- R: scope of the event -->                                 </scope>
  <!-- event pattern description -->
     <evaluation>  <!-- R: semantics: selection, consumption policies -->    </evaluation>
     <signature>   <!-- R: event pattern declaration -->                     </signature>
  <!-- event instance  -->
     <qualification>  <!-- R: e.g. qualifying event declarations, e.g.
```

```
                        priorities, validity, strategy -->              </qualification>
    <quantification> <!-- R: quantifying rule declarations -->          </quantification>

    <content>        <!-- R: event instance content -->                 </content>
</Event>
```

Reaction RuleML 1.0 provides the support for rule-based event processing and semantic complex event processing. With its typed logic, RuleML provides the support for (re)using external temporal, spatial, situation, event, and action ontologies and a metamodel which can be applied in the definition of semantic event/action types and temporal and spatial relations [3, 17]. Reaction RuleML defines a library of typical event, action, interval algebra operators and generic elements such as `Event`, `Action` , `Situation`, `Time`, `Location`, `Interval`, `Operator`. The type of these generic elements can be defined by an `@type` reference to external ontologies, e.g. to the Reaction RuleML metamodel (see [17]). For instance, `<Operator type="ruleml:Sequence">` instead of `<Sequence>`. The following example shows a complex event pattern definition:

```
<Event key="ce2" type="ruleml:ComplexEvent">
  <signature> <!-- pattern signature definition -->
    <Sequence>
      <!-- atomic event -->
      <signature>
        <Event type="ruleml:SimpleEvent">
          <signature><Atom>...event_A...</Atom></signature>
        </Event>
      </signature>
      <!-- nested complex event referenced by @keyref -->
      <signature><Event type="ruleml:ComplexEvent" keyref="ce1"/></signature>
      <!-- Common Base event selected via xpointer/xpath query in iri attribute -->
      <signature>
        <Event type="cbe:CommonBaseEvent" iri="cbe.xml#xpointer(//CommonBaseEvent)"/>
      </signature>
    </Sequence>
  </signature>
</Event>

<Event key="ce1">
  <signature> <!-- event pattern signature -->
    <Concurrent>
      <Event><meta><Time>...t3</Time></meta><signature>...event_B</signature></Event>
      <Event><meta><Time>...t3</Time></meta><signature>...event_C</signature></Event>
    </Concurrent>
  </signature>
</Event>
```

Such a complex event pattern definition can be used for event detection in the `<on>` part of a reaction rule of a rule-based event consuming agent:

```
<Rule style="active">
 <on><Event keyref="ce2"/></on>
 ...
 <do> ... </do>
</Rule>
```

These Reaction RuleML rules for Complex Event Processing (CEP) can be translated and executed e.g. in Prova. For an overview on typical (complex) event pattern functions and their implementations see [19][16].

[16] slides at `http://goo.gl/E30Vu`

In our SymposiumPlanner demo scenario we consume and process the events of the symposium, such as the news from the Twitter feed, calendar events (deadlines etc.), etc. We apply a typical publish-subscribe approach where users can subscribe their information needs in terms of (complex) event pattern to the rule-based semantic event processing agents. The agents actively inform the subscribers if they detect the relevant event patterns by continuously processing the published events on the news feeds.

6 Summary

In this paper, we presented how the standardized Reaction RuleML 1.0 interchange format supports loosely-coupled and de-coupled event-messaged interactions in the rule-based semantic multi-agent system Rule Responder. We demonstrated several expressiveness features of Reaction RuleML 1.0 on the example of the Symposium Planner use case. We also showed how the computational independent (natural) language Attempto Controlled English (ACE) is used to construct user queries against rule-based KBs in distributed Rule Responder agents (inference services), which are using Reaction RuleML as an intermediary platform-independent language between the computational independent user interface language (ACE) and the platform-specific execution languages (Prova, OO jDrew, Drools, ...).

References

1. Harold Boley, Taylor Osmun, and Benjamin Craig. Social Semantic Rule Sharing and Querying in Wellness Communities. In Asuncin Gmez-Prez, Yong Yu, and Ying Ding, editors, *The Semantic Web*, volume 5926 of *Lecture Notes in Computer Science*, pages 347–361. Springer Berlin / Heidelberg, 2009.
2. Harold Boley and Adrian Paschke. Rule Responder Agents Framework and Instantiations. In Atilla Eli, MamadouTadiou Kon, and MehmetA. Orgun, editors, *Semantic Agent Systems*, volume 344 of *Studies in Computational Intelligence*, pages 3–23. Springer Berlin Heidelberg, 2011.
3. Harold Boley, Adrian Paschke, and Omair Shafiq. RuleML 1.0: The Overarching Specification of Web Rules. In *RuleML*, pages 162–178, 2010.
4. Jens Dietrich and Adrian Paschke. On the Test-Driven Development and Validation of Business Rules. In *ISTA*, pages 31–48, 2005.
5. Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Attempto Controlled English Meets the Challenges of Knowledge Representation, Reasoning, Interoperability and User Interfaces. In Geoff Sutcliffe and Randy Goebel, editors, *FLAIRS Conference*, pages 664–669. AAAI Press, 2006.
6. Tichomir Jabarski. Design and Development of A Translator Framework for Rule Languages Based on RuleML, Master Thesis. Master's thesis, Free University Berlin, 2012.
7. A. Paschke, J. Dietrich, A. Giurca, G. Wagner, and S. Lukichev. On Self-Validating Rule Bases. In *Int. Semantic Web Enabled Software Engineering Workshop (SWESE'06)*, 2006.

8. Adrian Paschke. The ContractLog Approach Towards Test-driven Verification and Validation of Rule Bases - A Homogeneous Integration of Test Cases and Integrity Constraints into Dynamic Update Logic Programs and Rule Markup Languages (RuleML). In *IBIS, TUM, Technical Report 10/05*, 2005.
9. Adrian Paschke. ECA-RuleML: An Approach Combining ECA Rules with Temporal Interval-based KR Event/Action Logics and Transactional Update Logics. *CoRR*, abs/cs/0610167, 2006.
10. Adrian Paschke. Verification, Validation and Integrity of Distributed and Interchanged Rule Based Policies and Contracts in The Semantic Web. In *In Second International Semantic Web Policy Workshop (SWPW06)*, pages 2–16, 2006.
11. Adrian Paschke. Rule Responder HCLS eScience Infrastructure. In *Proceedings of the 3rd International Conference on the Pragmatic Web: Innovating the Interactive Society*, ICPW '08, pages 59–67, New York, NY, USA, 2008. ACM.
12. Adrian Paschke. A Semantic Rule and Event Driven Approach for Agile Decision-Centric Business Process Management - (Invited Paper). In *ServiceWave*, pages 254–267, 2011.
13. Adrian Paschke and Martin Bichler. Knowledge Representation Concepts for Automated SLA Management. *Decision Support Systems*, 46(1):187–205, 2008.
14. Adrian Paschke and Harold Boley. Rules Capturing Events and Reactivity. In Adrian Giurca, Dragan Gasevic, and Kuldar Taveter, editors, *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, pages 215–252. IGI Publishing, May 2009.
15. Adrian Paschke and Harold Boley. Rule Responder: Rule-Based Agents for The Semantic-Pragmatic Web. *International Journal on Artificial Intelligence Tools*, 20(6):1043–1081, 2011.
16. Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Larry Craig. Rule responder: RuleML-based Agents for Distributed Collaboration on The Pragmatic Web. In *ICPW*, pages 17–28, 2007.
17. Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian, and Tara Athan. Reaction RuleML 1.0: Standardized Semantic Reaction Rules. In *Proceddings of RuleML 2012*, 2012.
18. Adrian Paschke and Alexander Kozlenkov. A Rule-based Middleware for Business Process Execution. In *Multikonferenz Wirtschaftsinformatik*, 2008.
19. Adrian Paschke, Paul Vincent, Alexandre Alves, and Catherine Moxey. Tutorial on Advanced Design Patterns in Event Processing. In *DEBS*, pages 324–334, 2012.
20. L.M. Surhone, M.T. Tennoe, and S.F. Henssonow. *Drools*. VDM Verlag Dr. Mueller AG & Co. Kg, 2010.
21. Zhili Zhao, Adrian Paschke, Chaudhry Usman Ali, and Harold Boley. Principles of The SymposiumPlanner Instantiations of Rule Responder. In *Proceedings of The 5th International Conference on Rule-based Modeling and Computing on The Semantic Web*, RuleML'11, pages 97–111, Berlin, Heidelberg, 2011. Springer-Verlag.