NRC Publications Archive Archives des publications du CNRC

Reducing the overconfidence of base classifiers when combining their decisions

Raudys, Šarunas; Somorjai, Ray; Baumgartner, Richard

For the publisher's version, please access the DOI link below. / Pour consulter la version de l'éditeur, utilisez le lien DOI ci-dessous.

Publisher's version / Version de l'éditeur:

https://doi.org/10.1007/3-540-44938-8_7

Multiple Classifier Systems: 4th International Workshop, MCS 2003 Guildford, UK, June 11–13, 2003 Proceedings, Lecture Notes in Computer Science; no. 2709, pp. 65-73, 2003

NRC Publications Archive Record / Notice des Archives des publications du CNRC :

<https://nrc-publications.canada.ca/eng/view/object/?id=8a1144bd-69c8-49fe-9ae2-8ef86f388615>

<https://publications-cnrc.canada.ca/fra/voir/objet/?id=8a1144bd-69c8-49fe-9ae2-8ef86f388615>

Access and use of this website and the material on it are subject to the Terms and Conditions set forth at

<https://nrc-publications.canada.ca/eng/copyright>

READ THESE TERMS AND CONDITIONS CAREFULLY BEFORE USING THIS WEBSITE.

L'accès à ce site Web et l'utilisation de son contenu sont assujettis aux conditions présentées dans le site

<https://publications-cnrc.canada.ca/fra/droits>

LISEZ CES CONDITIONS ATTENTIVEMENT AVANT D'UTILISER CE SITE WEB.

Questions? Contact the NRC Publications Archive team at

PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca. If you wish to email the authors directly, please see the first page of the publication for their contact information.

Vous avez des questions? Nous pouvons vous aider. Pour communiquer directement avec un auteur, consultez la première page de la revue dans laquelle son article a été publié afin de trouver ses coordonnées. Si vous n'arrivez pas à les repérer, communiquez avec nous à PublicationsArchive-ArchivesPublications@nrc-cnrc.gc.ca.

Reaction RuleML 1.0: Standardized Semantic Reaction Rules

Adrian Paschke¹, Harold Boley², Zhili Zhao¹, Kia Teymourian¹, Tara Athan³

¹ Freie Universitaet Berlin, Germany

{paschke, zhili, teymourian} AT inf.fu-berlin.de

² Information and Communications Technologies, National Research Council Canada
Fredericton, NB, Canada

harold.bole AT nrc.gc.ca

³ Athan Services, W Lafayette, IN, USA

taraathan AT gmail.com

Abstract. RuleML is a family of XML languages whose modular system of schemas permits high-precision (Web) rule interchange. The family's top-level distinction is deliberation rules vs. reaction rules. In this paper we address the Reaction RuleML subfamily of RuleML and survey related work. Reaction RuleML is a standardized rule markup/serialization language and semantic interchange format for reaction rules and rule-based event processing. Reaction rules include distributed Complex Event Processing (CEP), Knowledge Representation (KR) calculi, as well as Event-Condition-Action (ECA) rules, Production (CA) rules, and Trigger (EA) rules. Reaction RuleML 1.0 incorporates this reactive spectrum of rules into RuleML employing a system of step-wise extensions of the Deliberation RuleML 1.0 foundation.

1 Introduction

Event-driven reactive functionalities are urgently needed in present-day distributed systems and dynamic Web-based environments. Reaction rules constitute a promising approach to specify and program such reactive systems in a declarative manner. In particular, they provide the ability to reason over events, actions and their effects, and allow detecting events and responding to them automatically. A great variety of approaches has been developed for reaction rules, which have for the most part evolved separately and have defined their own domain and platform specific languages [14, 12, 16]. Novel semantics are being devised, including for the Logic-based agent and Production System language (LPS) and KELPS [6].

Reaction RuleML¹ is intended as a common standard for representing reaction rules and rule-based complex event processing (CEP) in a platform independent XML markup expression language. Reaction RuleML allows for standardized rule interchange, semantic interpretation and translation, and distributed event-messaging interactions in loosely-coupled and de-coupled distributed rule-based systems such as Web inference services and semantic agents.

¹ <http://reaction.ruleml.org/>

RuleML² has been designed for the standardized interchange of the major kinds of rules in a XML format that is uniform across rule languages and platforms. It has broad coverage and is defined as an extensible family of languages. In this paper, we introduce Reaction RuleML 1.0³, which directly builds on RuleML 1.0 [3].⁴ By describing the language features of Reaction RuleML this paper also surveys the major lines of reaction rule types. We assume that readers are already familiar with Web rule technologies and their underlying semantics.⁵

The rest of the paper is organized as follows. Section 2 introduces the main rule syntax of Reaction RuleML 1.0 and specializes it to the four subbranches addressing the major reaction rule types. Section 3 describes selected expressive features of Reaction RuleML. In section 4 we compare Reaction RuleML with other platform-independent rule standards, analyzing its representational completeness with respect to an ontological reference metamodel. Section 5 summarizes the approach of Reaction RuleML 1.0, discusses its applicability with respect to recent works, and gives an outlook on future work in Reaction RuleML 1.1.

2 Reaction RuleML for Representing Reaction Rules

Reaction rules are concerned with the invocation of actions in response to events and actionable situations [12]. 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 [14, 16].

Reaction RuleML follows the general principles of markup language design as defined in [11]. Its subbranches span across the four major reaction rule types:

- Production Rules (Condition-Action rules) in the Production RuleML branch
- Event-Condition-Action (ECA) rules in the ECA RuleML branch
- Rule-based Complex Event Processing (CEP) (complex event processing reaction rules, (distributed) event messaging reaction rules, query reaction rules etc.) in the CEP RuleML branch
- Knowledge Representation (KR) Event/Action/Situation Transition/Process Logics and Calculi in the KR Reaction RuleML branch

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 thereof). The following template shows the most general rule syntax of RuleML with 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>
```

² <http://ruleml.org/>

³ <http://reaction.ruleml.org/1.0/>

⁴ <http://ruleml.org/1.0/>

⁵ For a deeper study of Web rule / event processing technologies we refer to [10–12, 7]

```

<!-- 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>

```

These role tag elements below the general **<Rule>** element are used for representing the following information blocks in a rule.

- The attribute **@keyref** is used for creating distributed and modularized accessibility within a (distributed) knowledge base, where **@key** is the identifier key and **@keyref** is a key reference.
- The general style of a reaction rule is defined by the optional attribute **@style**, which has the following values in Reaction RuleML
 - **active**: actively polls and detects occurred events in ECA and CEP rules or changed conditions in production rules.
 - **messaging**: waits for incoming complex event message (inbound) and sends messages (outbound) as actions.
 - **reasoning**: logical reasoning as e.g. in formalisms such as event / action / transition logics (as e.g. in Event Calculus, Situation Calculus, temporal action languages formalizations) etc.
- The metadata **<meta>** is used to annotate the rule with optional metadata.
- The scope **<scope>** defines a (constructive) view on the rulebase, e.g. the rule only applies to a particular module in the rulebase.
- The evaluation semantics (interpretation semantics and/or execution semantics) of reaction rules is defined in the optional role subchild **evaluation**. This can be used to define rule evaluation semantics such as weak or strong evaluation which defines the “execution lifecycle” of the rule execution or other semantic policies, e.g. event consumption policies, transaction semantics etc.
- The **<signature>** defines the rule signature with optional input / output mode declarations. The rule signature declaration can act as public rule interface and can be published together with the intended evaluation semantics in distributed Reaction RuleML bases.
- The qualification **<qualification>** defines an optional set of rule qualifications such as a validity value, fuzzy value or a priority value.
- The quantification **<quantification>** is used to define quantifiers such as the typical existential and universal quantification; it can also be used for extensions such as variable binding patterns to restrict pattern matching in production rules or define other operator definitions.

- The **<on>** part of a rule defines the triggering events, which can be atomic or complex event definitions.
- The **<if>** part defines one or more conditions. In case of a reaction rule with actions this would be the pre-conditions.
- The **<then>** part defines the (logical) conclusions of the rule. Hybrid rules are possible which define a logical conclusion in the **<then>** part as well as actions in the **<do>** part.
- The **<do>** part of a rule defines the actions, which can be an atomic as well as a complex action definition.
- The **<after>** part of a rule defines the post-conditions which hold after the actions. They can be used, e.g., as post-conditional integrity constraints on the effects of the action execution on the knowledge state, which holds after the action has been performed. Depending on the defined **<evaluation>** semantics this might lead to roll-backs in the case of transactional semantics.
- The **<else>** part leads to if-then-else rules. Special semantics for **<else>** might be defined in the **<evaluation>** semantics of the rule.
- The **<elsedo>** part is executed instead of the **<do>** part. This allows to define e.g. compensating actions in transactional logics or default and exception handling actions if the normal execution of the **<do>** fails. Special semantics for **<elsedo>** might be defined in the **<evaluation>** semantics of the 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, as used e.g. in KR RuleML for logical event/action calculi), if-do (production rules), on-do (trigger rules), on-if-do (ECA rules).

Derivation Rule:	Production Rule:	ECA Rule:	CEP Rule:
<code><Rule style="reasoning"> <if>...</if> <then>---</then> </Rule></code>	<code><Rule style="active"> <if>...</if> <do>---</do> </Rule></code>	<code><Rule style="active"> <on> ---- </on> <if> ... </if> <do> ---- </do> </Rule></code>	<code><Rule style="messaging"> <on> event 1 </on> <do> action 1 </do> <on> event 2 </on> <if> condition</if> <do> action 2 </do> ... </Rule></code>

Before we further describe the expressive features in Reaction RuleML 1.0 in section 3, we first specialize this general reaction rules approach to the four different Reaction RuleML branches in the following subsections.

2.1 Production RuleML

A production rule is a statement of rule programming logic, which specifies the execution of one or more actions in case its conditions are satisfied, i.e. production rules react to states changes (not to explicit events as e.g. in ECA rules). The essential syntax is *if Condition do Action* as shown in the following example (this is a ‘pure’ production rule since the **<do><Assert><Atom>** action in Reaction RuleML maps to a **<then><Atom>** conclusion in Deliberation RuleML):

```
<!-- If premium customer and regular product do assert discount of 5 percent
      for customer on product -->
<Rule style="active">
  <if>
    <And>
      <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
```

```

    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </And>
</if>
<do>
  <Assert>
    <Atom><Rel>discount</Rel><Data>5.0</Data><Var>cust</Var><Var>prod</Var></Atom>
  </Assert>
</do>
</Rule>

```

The central predefined actions in Production RuleML are: **<Assert>** (add knowledge); **<Retract>** (retract knowledge); **<Update>** (update/modify knowledge); **<Set>** and **<Get>** (assignment/dereferencing of (global) variables); **<Execute>** (execution of (external) functions). Furthermore, a generic **<Action>** is defined which allows model references to externally defined action models, e.g., **<Assert>** is a shortcut for **<Action type="ruleml:Assert">**, where the action such as for assertions is defined in the metamodel of Reaction RuleML (i.e. in the Reaction RuleML meta-ontology).

Production RuleML supports “Negation-as-failure” (**<Naf>**), which by default is interpreted with an inflationary semantics.⁶ Again, **<Naf>** is a shortcut for the more generic negation definition **<Negation type="ruleml:InflationaryNegation">** with reference to the Production RuleML metamodel. Other types of negations can be specified using the **@type** attribute.

In the **evaluation** section the semantics for the interpretation and operational execution can be specified. The Reaction RuleML metamodel predefines typical semantic profiles (see section 3.2) for different classes of production rule systems. This furthermore includes semantics for conflict resolution strategies such as **ruleml:Refraction**, **ruleml:Priority**, and **ruleml:Recency**.

The rule quantification can be used to quantify variables and define further binding patterns. For instance, the following example defines a rule with a **<Forall>** quantifier for one variable with a variable binding pattern. In contrast to Deliberation RuleML 1.0 the **<quantification>** is defined under the **<Rule>** and not outside of Deliberation RuleML’s rule element **<Implies>**. The reason for this is, because Reaction RuleML defines both the interface and the implementation under the **<Rule>**, but possibly distributed within the KB with a reference from the **<Rule>**’s interface description to the **<Rule>**’s implementation in which the quantifier are defined.

```

<Rule style="active">
  <!-- for all ?x such that ?x is John -->
  <quantification> <!-- explicit quantification -->
    <Forall>
      <declare><Var>x</Var></declare><!-- for all ?x -->
      <guard><Equal><Var>x</Var><Ind>John</Ind></Equal></guard> <!-- such that ?x is John -->
    </Forall>
  </quantification>
  <if>...</if>
  <do>...</do>
</Rule>

```

Note that not only Rules but also facts, conjunctions, disjunctions, and many other constructs can be quantified in Reaction RuleML using the quantification element. In particular, quantified variable declarations with explicit variable bindings are also possible in the **<do>** part of a production rule in order to initialize local action variables for the actions.

⁶ For a discussion in RIF-PRD see <http://lists.w3.org/Archives/Public/public-rif-wg/2008Dec/0053.html> and <http://lists.w3.org/Archives/Public/public-rif-wg/2008Dec/0055.html>

2.2 ECA Reaction RuleML

In contrast to production rules, Event-Condition-Action (ECA) rules define an explicit event part which is separated from the conditions and actions of the rule. Their essential syntax is *on Event if Condition do Action*. ECA RuleML syntactically extends Production RuleML with an explicit `<on>` event part and rich (complex) event and action constructs and semantics defined in event/action libraries. Variants of this standard ECA rule are, e.g., Event-Action triggers (EA rules) and ECAP rules (ECA rules with postconditions after the action part).

<code><!-- ECA rule --></code>	<code><!-- EA trigger rule --></code>	<code><!-- ECAP rule with postcondition --></code>
<code><Rule style="active"></code>	<code><Rule style="active"></code>	<code><Rule style="active"></code>
<code><on>***</on></code>	<code><on>***</on></code>	<code><on>***</on></code>
<code><if>...</if></code>	<code><on>***</on></code>	<code><if>...</if></code>
<code><do>---</do></code>	<code><do>---</do></code>	<code><do>---</do></code>
<code></Rule></code>	<code></Rule></code>	<code><after>---</after></code>
		<code></Rule></code>

We modify our discount example as follows:

```
<!-- On the order event of a product from a customer, if the customer is premium
do offer a discount of 5 percent to the customer -->
<Rule style="active">
  <on>
    <Event>
      <signature>
        <Atom><Rel per="value">order</Rel><Var>cust</Var><Var>prod</Var></Atom>
      </signature>
    </Event>
  </on>
  <if>
    <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
  </if>
  <do>
    <Action>
      <signature>
        <Atom><Rel per="effect">offer</Rel><Var>cust</Var><Var>prod</Var><Data>5.0</Data></Atom>
      </signature>
    </Action>
  </do>
</Rule>
```

With the `@per` attribute it can be indicated whether the atomic `<Rel>`ations will be uninterpreted ("copy"), interpreted ("value"), effectful ("effect") or modal ("modal").

`<Event>` and `<Action>` (already introduced in Production RuleML) are used to represent events and actions. We distinguish between the `<signature>` of the event pattern definition, the concrete event instance `<content>` and other properties of the event (similar to the `<Rule>` properties).

```
<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>

```

The following standard library defines a set of typical event, action, counting, temporal, and interval algebra operators for defining, e.g., complex events, actions, and intervals:

```

Action Algebra
  Succession (Ordered), Choice (Non-Deterministic Choice),
  Flow (Parallel Flow), Loop (Loops), Operator (generic Operator)

Event Algebra
  Sequence (Ordered), Disjunction (Or), Xor (Mutually Exclusive), Conjunction (And),
  Concurrent (Parallel), Any, Aperiodic, Periodic, Operator (generic Operator)

Counting Algebra
  Counter, AtLeast, AtMost, Nth, Operator (generic Operator)

Negation / Absence Algebra
  Not, Unless, Operator (generic Operator)

Temporal operators
  Timer, Every, After, Any, Operator (generic Operator)

Interval Algebra (Time/Spatial/Event/Action Intervals)
  During, Overlaps, Starts, Precedes, Succeeds, Meets,
  Equals, Finishes, Operator (generic Operator)

```

With its typed logic, RuleML provides support for (re)using external temporal, spatial, situation, event, and action ontologies and metamodels which can be applied in the definition of semantic event/action types and temporal and spatial relations (see section 3.1). Reaction RuleML defines generic elements such as **Event**, **Action**, **Situation**, **Time**, **Location**, **Interval**, **Operator**. The type of this generic elements can be defined by an **@type** reference to such external ontologies, e.g. to the Reaction RuleML metamodel. For instance, **<Operator type="ruleml:Sequence">** instead of **<Sequence>**. The following examples shows a complex event pattern definition.

```

<Event key="ce2" type="ruleml:ComplexEvent">
  <signature>
    <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>
    <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>

```


2.3 CEP RuleML

Complex Event Processing (CEP) is about the detection of complex events and reaction to complex events in near realtime [7]. CEP rules might adopt the style of ECA rules in CEP RuleML, where the `<on>` event part is a complex event type definition; or, they might adopt the style of CA production rules where the (complex) event types are defined as restrictions on the variable binding definitions in the rule quantifications while event detection is then done in the condition part with the conditions representing the event pattern definitions. However, it is also possible to represent serial messaging CEP reaction rules which `<Receive>` and `<Send>` events in arbitrary combinations, leading to an event processing workflow-style logic following a forward-directed serial control flow for the literals in the rule. A serial (messaging) reaction rule starts either with a receiving event `on` – the trigger of the *global* reaction rule – or with a rule conclusion `then` – the queryable head literal of another messaging reaction rule or derivation rule (without further events or actions) – followed by an arbitrary combination of conditions `if`, events (`Receive`), and actions (`Send`) in the body of the rule. This approach allows to combine forward-directed serial workflow-style execution of reaction rules with backward-reasoning derivation rules.⁷ This flexibility in terms of representing rule-based workflow style branching logics with a serial execution control flow for the rule literals and with support for modularization and aspect-oriented weaving of reactive rule code in combination with derivation 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="messaging">
  <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
    using backward-reasoning derivation rules and/or </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
```

For better modularization the sub-conversation can be also written with a second reaction rule as follows:

```
<Rule>
  <on><Receive> receive event from agent 1 </Receive></on>
  <if> <!-- this goal activates the reaction rule via backward reasoning -- see below -->
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>
<!-- the "if" goal from the first rule applies to the "then" conclusion of the second rule
and activates it via backward reasoning (rule chaining) -->
<Rule>
  <then>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </then>
  <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>
</Rule>
```

In the example the first reaction rule is triggered by a received event (`on`) which starts the forward-directed serial execution flow of the rule. It then deductively proves

⁷ see Prova's serial messaging rules; <http://www.prova.ws/>

the condition (**if**) by applying it as a backward-reasoning subgoal on the knowledge base. This subgoal unifies with the head (**then**) of the second rule (which can be a standard derivation rule or a serial messaging reaction rule with further send and receive subgoal literals in the body). The second rule starts a subconversation sending a query to another agent 2 (**do**) and on receiving an answer from this agent (**on**) the execution of the second rule terminates successfully, which means the backward-reasoning subgoal of the first rule evaluates to true with the variable bindings for the variable **prod**. The first reaction rule then proceeds with its serial execution flow using the variable bindings for sending the results back to agent 1 (**do**) – either one by one in multiple messages for each resulting variable binding (default semantics according to backward-reasoning resolution semantics) or as a collection of all resulting answers (like in the `findall` built-in of Prolog or in join semantics of workflow branching logics).

That is, the semantics of this serial reaction rules combines standard logic programming (with resolution, backtracking, and variable binding) on the literals in combination with a serial forward directed execution of the rule literals which includes receiving and sending event queries to other distributed inference services and event processing agents (see section 3.3). This expressiveness allows us to represent workflow logics with parallel execution branches. The messaging reaction rules can be translated, e.g., into serial messaging Horn rules and executed in the Prova⁸ rule engine.

2.4 KR Reaction RuleML

Event/action logics, which have their origins in the area of knowledge representation (KR), focus on inferences that can be made from happened or planned events/actions, i.e. they describe inferences about the effects of events/actions on changeable properties of the world (situations, states). That is, the focus of this calculi typically is on the logical inferences and effects of events/actions without actually effecting any events/actions as in active reaction rules (although there can be combinations of the logical effects and the actual execution of actions and events). Reaction RuleML also defines syntax and semantics for knowledge representation event/action calculi such as Situation Calculus, Event Calculus, and Temporal Action Languages, etc. (for an overview see [14]) Specifically the notion of an explicit **<Situation>** (a.k.a. as state or fluent in Event Calculus) is introduced in KR Reaction RuleML. Situations are changeable fluents which are initiated or terminated as the effect of events. An event, which **<Happens>**, can **<Initiate>** or **<Terminate>** a situation. That is, a situation explicitly represents the abstract effect of occurred events and executed actions. Such states can e.g. be used for situation reasoning, e.g. in the condition part of reaction rules.

<code><!-- Initiates a situation --></code>	<code><!-- Terminates a situation --></code>	<code><!-- event happens --></code>
<code><Rule></code>	<code><Rule></code>	<code><Happens>*</Happens></code>
<code><on>*</on></code>	<code><on>*</on></code>	
<code><if>.</if></code>	<code><if>.</if></code>	<code><!-- situation holds? --></code>
<code><do><Initiate>-</Initiate></do></code>	<code><do><Terminate>-</Terminate></do></code>	<code><if><Holds>-</Holds></if></code>
<code></Rule></code>	<code></Rule></code>	

3 Reaction RuleML Features

Reaction RuleML provides several layers of expressiveness for adequately representing reactive logic and for interchanging events (queries, actions, event data) and rules. In the following, some of the expressive constructs of Reaction RuleML 1.0 are described.

⁸ <http://prova.ws>

3.1 Reaction RuleML Metamodel, Semantic Types and Data Queries

Reaction RuleML is based on metamodels and ‘pluggable’ ontologies. Figure 1 shows the top level structure of the Reaction RuleML metamodel. The Reaction RuleML

Fig. 1. Reaction RuleML Semantic CEP Metamodel

metamodel 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. For instance, the concepts and relations for time and space are used in the event and action ontology, which is employed in the situation ontology etc. These general concepts defined in the top-level ontologies can be further specialized with existing domain ontologies and ontologies for generic tasks and activities (e.g. situation processing, processes/workflows, agents including their pragmatic protocols etc.). The applications ontologies for specialize these domain and task concepts with respect to a specific application, often on a more technical platform specific level. For instance, figure 2 shows the top-level ontology for situations. A

Fig. 2. Reaction RuleML Situation Metamodel

situation description **hasProperties** defined by other ontologies such as time, event, etc., and **hasContent** with the main content of a situation description. A situation can be distinguished into two heterogeneous situation types – homogenous situation and

heterogeneous situation. These two types can be further specialized in domain-specific ontologies with more specific situation types as illustrated in the figure.

There are many existing ontologies which can be used in this generic modularized Reaction RuleML metamodel. For instance, the ontological constructs from the metamodel for the Bunge-Wand-Web representation model from [17] such as **Conceivable State Space**, **Lawful State Space**, **State Law**, **Stable State**, **Unstable State**, **History**, and further specializations of the top-level concepts such as event, thing, system, etc.

The Reaction RuleML metamodel defines the event, action, and interval algebra operators for complex events, actions, and intervals as well as other semantic concepts in the respective ontologies. These semantic definitions allow distinguishing, e.g., event/action algebra operators from interval operators and from logical operators (logical connectives). As described in section 2.2 Reaction RuleML comes with a library of predefined syntactic constructs which are syntactic short cut notations in the Reaction RuleML syntax for a more generic syntax element which points to its (semantic) type definition in the Reaction RuleML metamodel (or uses other external ontological definitions). To allow these external type definitions Reaction RuleML uses the **@type** attribute in the generic Reaction RuleML elements such as **Event**, **Action**, **Situation**, **Time**, **Location**, **Interval**, **Operator**, **Neg**, **Quantifier** etc., as illustrated by the following examples:

```
<Operator type="ruleml:Conjunction"> == <Conjunction>
<Negation type="ruleml:InflationaryNegation"> == <Naf> (in production rules)
<Action type="ruleml:Assert"> == <Assert>
<Event type="ruleml:SimpleEvent"> == <Atom> ... </Atom>
<Event type="ibm:CommonBaseEvent"> == IBM CBE
<Operator type="snoop:Sequence"> == Snoop == <Operator type="ruleml:Sequence"> == <Sequence>
```

While **@type** is used to refer to external type definitions (defined e.g. in external ontologies), the **@iri** attribute can be used as a general pointer to a (Web) resource having the semantics of containing either a standard IRI with a possible query specification (after the "?") or an XPointer and XPath expression as manipulation and query language to point into and select data from external XML data sources. Note that XPointers and normal IRI references are syntactically distinguishable so that there is no problem in processing them differently.

```
<Atom>
  <Rel>name</Rel>
  <Ind iri="person.xml#xpointer(//Person/LastName[1]/text())"/>
</Atom>
```

It is possible to define XPointer-based query and manipulation expressions that operate on (large) resource sets instead on singleton resources, e.g. to specify a constructive view over a set of external nodes specified by a XPath query expression. The following example selects four **CommonBaseEvent** from an XML document.

```
<Event iri="cbe.xml#xpointer(//CommonBaseEvent[1]/range-to(//CommonBaseEvent[4]))"/>
<Action iri="BPEL.xml#xpointer(//invoke[@name=checkHotel])"/>
```

That is the IRIs are query expressions that return constructive views on resource sets that are treated as singletons, i.e. a constructive view over a set of resources is a singleton with an unique IRI denoted by the query expression. It is possible to assign these views to variables and reuse the variables in the local reasoning/execution scope.

This external data and semantic knowledge can be used for Semantic Complex Event Processing (SCEP). In [19] we showed how the usage of semantic background knowledge about events and other related concepts can improve the quality of event processing. We described how to formalize semantic complex event patterns based on a logical interval-based event algebra, namely the interval-based Event Calculus [8, 9].

3.2 Rule Interface Descriptions with Semantic Profiles and Signatures

Reaction RuleML distinguishes between the interface of a rule and its implementation. The interface describes the functional and non-functional (semantic) properties of the rule. Reaction RuleML provides an interface definition language for the **<signatures>**s of publicly accessible rule functions together with their mode and type declarations. Modes are states of instantiation of the predicate/function 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(Result, Arg1, Arg2)* is *add(-, +, +)*, i.e. the function predicate *add* returns one output argument followed by two input arguments. Also, all arguments must be integer values. Serialized in Reaction RuleML this would be:

```
<signature>
  <Atom>
    <Rel>add</Rel>
    <Var mode="-" type="xs:integer"/>
    <Var mode="+" type="xs:integer"/>
    <Var mode="+" type="xs:integer"/>
  </Atom>
</signature>
```

The following examples illustrates the use of such signatures declarations in the interface descriptions of rules and distinguishes the interface from the implementation.

```
<!-- 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>discount</Rel><Var mode="-"/><Var mode="?"><Var mode="+"></Atom>
  </signature>
</Rule>

<!-- implementation of rule 1 which is interpreted either by WFS or by ASS semantics
  according to it's interface definition. -->
<Rule key="r1" style="reasoning">
```

```

<if>
  <And>
    <Atom><Rel>premium</Rel><Var>cust</Var></Atom>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </And>
</if>
<then>
  <Atom><Rel>discount</Rel><Data>5.0</Data><Var>cust</Var><Var>prod</Var></Atom>
</then>
</Rule>

```

The example above defines two rule interfaces which reference the implementation of rule 1 and rule 2 via **key-keyref** attributes. This approach supports modularization of the knowledge base and reusability of (XML) code by reference. By separating the interface description from the rule implementation, information hiding can be realized in distributed KBs. This is in particular useful for distributed rule inference service and rule agents which might publish some of the rule interfaces of their KBs publicly on the Web, but hide the concrete implementation of the rules. This enables a loosely-coupled interaction with the inference service / agent, where queries can be posed against the public interface signature **descriptions**. The interface defines the applicable evaluation semantics, which in the example uses predefined semantic **Profiles** from the RuleML metamodel.

Different interpretation, selection, consumption, and (transactional) execution policies for (complex) events and actions can be specified in the rule's **<evaluation>** semantics and the complex event/action descriptions.

The **<Profile>** can either point to externally defined profile type using **@type** (e.g. in the RuleML metamodel) or using **@iri** for general pointers to external definitions. Furthermore, since **<Profile>**'s content model is of type **xs:any** an arbitrary XML based definition of a profile can be specified here. This gives maximum flexibility in defining application-specific interpretation semantics and execution policies.

3.3 Reaction RuleML Messaging

As described in the previous section, the interface description language of Reaction RuleML allows for loosely-coupled interaction⁹ with distributed Reaction RuleML inference services and agent's KBs. For the communication between distributed rule-based (agent) systems, Reaction RuleML provides a general message syntax:

```

<Message directive="PRAGMATIC CONTEXT" >
  <oid>          <!-- conversation ID-->          </oid>
  <protocol>      <!-- transport protocol -->      </protocol>
  <sender>        <!-- sender agent/service -->    </sender>
  <receiver>      <!-- receiver agent/service -->  </receiver>
  <content>       <!-- message payload -->         </content>
</Message>

```

In the context of these Reaction RuleML 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 define the message passing and co-ordination protocol. The directive attribute corresponds to the pragmatic instruction,

⁹ Decoupled interaction is also possible by just publishing event (messages), e.g. in an event stream or publish-scribe middleware

e.g. a FIPA ACL primitive, i.e. the pragmatic characterization of the message context broadly characterizing the meaning of the message.

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>
```

For better modularization the sub-conversation logic can be also written with an inlined reaction rule as follows:

```
<Rule style="active">
  <on><Receive> receive event from agent 1 </Receive></on>
  <if> <!-- this goal activates the inlined reaction rule -- see below -->
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </if>
  <do><Send> reply to agent 1 by sending results received from agent 2 </Send></do>
</Rule>

<Rule style="active">
  <then>
    <Atom><Rel>regular</Rel><Var>prod</Var></Atom>
  </then>
  <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>
</Rule>
```

4 Comparison with Other Reaction Rule Representations

In this section we discuss and analyze related representation languages for reaction rules. We focus on standards which are on the same modeling level as Reaction RuleML – that is, the Platform Independent Model (PIM) according to OMG’s Model Driven Architecture (MDA).

For an overview on standardizations see [16]. For a discussion of rule markup languages and reaction rule languages see [12, 11]. Current standardization efforts are also under way in the Event Processing Technical Society (EPTS) on a common event processing glossary and vocabulary as well as reference architectures and design patterns [15]. Besides many existing ontologies for events, time, space etc., and industry standards for event communication, there have been also many different approaches for rule-based event processing and reaction rule languages [14].

Based on the metamodel of the Bunge-Wand-Weber (BWW) representation model [17] we analyze the representational completeness of Reaction RuleML 1.0 with respect to an ontological reference metamodel and compare it with the following rule standards: OMG PRR, W3C SWRL, and W3C RIF. We also add Derivation RuleML 1.0 to this

comparative coverage assessment, in order to show the differences and additional expressiveness of Reaction RuleML. Before we describe the analysis methodology and the analysis results we first introduce the languages.

W3C SWRL The Semantic Web Rule Language (SWRL)¹⁰ is defined as a language combining sublanguages of the OWL Web Ontology Language (OWL DL and Lite) with those of the Rule Markup Language (Unary/Binary Datalog). Rules in SWRL are of the form of an implication between an antecedent (body) conjunction and a consequent (head) conjunction, where description logic expressions can occur on both sides. The intended interpretation is as in classical first-order logic: whenever the conditions specified in the antecedent hold, then the conditions specified in the consequent must also hold.

Relationships between RuleML and SWRL: The W3C member submission SWRL combines an earlier version of RuleML with OWL 1.0. An effort was recently started to update SWRL, including for RuleML 1.0 and OWL 2.0.

W3C RIF The W3C Rule Interchange Format (RIF) Working Group¹¹ is an effort, influenced by RuleML, to define a standard RIF for facilitating the exchange of rule sets among different systems and to facilitate the development of intelligent rule-based application for the Semantic Web. So far, the RIF Working Group has published two dialects as recommendations – the Production Rules Dialect (PRD) for non-ground production rules with inflationary negation and the Basic Logic Dialect (RIF-BLD), which semantically corresponds to a Horn rule language with equality but without negation. A common condition language (RIF-Core) is shared between these two dialects. Like RuleML, RIF-BLD has a number of syntactic extensions with respect to ‘regular’ Horn rules, including F-logic-like frames, and a standard system of built-ins drawn from Datatypes and Built-Ins (RIF-DTB). The connection to other W3C Semantic Web languages is established via RDF and OWL Compatibility (RIF-SWC). Relationships between Production RuleML and RIF-PRD: Members of the Reaction RuleML Technical Group have co-edited the W3C RIF Production Rule Dialect (RIF-PRD). RIF-PRD with inflationary negation is a less expressive subset of PR RuleML. Syntactically, production rules in RIF-PRD are written in **if-then** syntax instead of PR RuleML’s **if-do** syntax, the latter allowing a clear semantic distinction of a conclusion (**then** part) and an action (**do** part), e.g. when both are allowed for the same rule. In RIF-PRD **Do** is used as a type tag to syntactically denote a compound action which is a sequence of standard production rule actions (**Assert**, **Retract**, and **Modify**), whereas Reaction RuleML supports expressive complex action definitions using action algebra operators. Quantifying variable binding declarations are supported by RIF-PRD (**declare**) and by Production RuleML (quantification), which in addition also supports rule qualifications.

OMG PRR OMG’s Production Rule Representation (PRR)¹² is a modeling language for production rules. It uses OMG MOF to define a generic metamodel for production rules and extends UML for modeling production rules. PRR includes two types of rules: Forward chaining inference rules (e.g. Rete) and sequentially processed procedural rules. PRR is defined at two levels. The adopted PRR Core of PRR 1.0 includes the

¹⁰ <http://www.w3.org/Submission/SWRL/>

¹¹ http://www.w3.org/2005/rules/wiki/RIF_Working_Group

¹² <http://www.omg.org/spec/PRR/1.0/PDF/>

general rule and production rule model. The PRR OCL, which is currently also in the focus of the PRR 1.1 effort, includes an extended OCL expression language enabling compatibility with non UML representations.

Relationships between Production RuleML and OMG PRR: Based on [20] and Production RuleML, members of the Reaction RuleML Technical Group have co-edited the OMG Production Rule Representation (PRR). RuleML is one of the languages whose features are to be covered by PRR on an abstract level. Since PRR is a meta-language, Production RuleML's XML syntax can be used as a concrete expression language instantiating PRR models. That is, OMG PRR provides a way to include rules into the (UML) model of an application at design time and Production RuleML then provides a standard means of translating the model and feeding the executable rules into a PR application at run time.

Event Processing Standards Besides Reaction RuleML, there exist no overarching interchange and representation standard for reaction rules and rule-based event processing. Although several of the standards efforts such as PRR and RIF (a Reaction Rules Dialect has been started but not finalized in RIF 1.0) have reaction rules on their long-term agenda, no concrete standardization effort started yet [16]. There have been several proposals for ECA-style rule languages and platform-specific languages coming from research project (Rewerse R2ML, ...) or from industry (IBM SRML, CBE, ...), partially defined as XML language. Several event ontologies have been proposed. Relationships between CEP RuleML and EPTS work: RuleML is a founding member of the Event Processing Technical Society (EPTS)¹³. Members of the Reaction RuleML Technical Group are contributing to the work on an Event Processing glossary, event processing language models, use cases, reference architectures, and design patterns. [15] With its flexible and extensible approach, CEP RuleML is a highly expressive rule-based Event Processing Language (rule-based EPL) which can make use of external event and action metamodels / ontologies such as the many existing event ontologies or the planned OMG Event Metamodel and Profile (EMP). Since CEP RuleML syntactically builds on top of Production RuleML and ECA RuleML – besides flexible (messaging) reaction rules – both major rule types can be used for representing (complex) event processing rules. Moreover, CEP RuleML can adequately represent typical use cases and functionalities in Event-Driven Architectures (EDAs) and (distributed) EP architectures.

For the analysis we selected a set of relevant ontological constructs from the BWV representation model (BWV RM) and classified them according to typical top-level categories of expressiveness for reaction rules. We slightly adapt the BWV model, introducing e.g. an explicit **Action** concept instead of an **Event** which **acts on**. We also introduce the concept of an **Agent** instead of the unspecific concept **System**. Following the methodology of [18] we then used this as a reference model for our analysis by comparing the representation constructs of the analyzed rule languages to the representation constructs of the reference model. We analyze these results according to the (ontological/metamodel) completeness and clarity of the rule representation / modeling languages. Therefore, we identify the representation deficiencies using the following metrics: construct *deficit* (with respect to a construct in the BWV reference model), *redundancy* (multiple language constructs map to the same construct in the BWV reference model), *overload* (the same language construct maps to multiple (semantically different) constructs in the BWV reference mode), and *excess* (language

¹³ <http://www.ep-ts.com/>

constructs that do not have a mapping to the BWW reference model). Table 1 shows the results of the comparison with the top-level categories. At this top level, Reaction

Table 1. Comparison of Rule Markup and Modeling Standards

<i>BWWRM</i>	<i>SWRL</i>	<i>PRR</i>	<i>RIF</i>	<i>DerivationRuleML</i>	<i>ReactionRuleML</i>
Thing	+	-	+	+	+
Event	-	-	-	-	+
Action	-	+	+	-	+
State (Situation)	-	-	-	-	+
System (Agent)	-	-	-	-	+

RuleML shows high ontological completeness and clarity with respect to the adapted BWW reference model, which is due to its close semantic similarity to the top level constructs in the Reaction RuleML metamodel. The **System** perspective in Reaction RuleML relates to the **Agent** metamodel in Reaction RuleML, where Reaction RuleML serves as a standardized rule and event interchange format between the agents / inference services (including the agent’s interface descriptions). The agents internally run platform-specific rule engines such as Prova, OO jDREW, Drools, and Emerald¹⁴ – see e.g. the Rule Responder project [13]¹⁵ for a reference implementation. Besides its use for smart reactive agents on the Web, future application domains of Reaction RuleML might directly enrich Rich Internet Applications (RIA), which are then translated into a domain-specific (Web browser) reaction rule language such as JSON Rules [4].¹⁶ Translators, such as the Reaction RuleML translator service framework and the MYNG syntax configurator¹⁷ of RuleML’s Relax NG schemas [1], contribute to the interoperability by translating from the platform specific languages into (Reaction) RuleML and vice versa, as well as between RuleML/XML and other XML-based languages such as RIF/XML. RIF RuleML interoperation was started with a common subset [2]. Other systems include rule editors (e.g., Acumen Business Rule Manager and S2Red¹⁸).

5 Conclusion

The four major reactive rule types were surveyed based on Reaction RuleML. Its uniform and flexible syntax can be specialized to branches of the corresponding subfamily of RuleML. With its metamodel and ability to plug in external vocabularies and type systems, Reaction RuleML has been developed as an extensible semantic rule language that is highly configurable via external semantic definitions. Reaction RuleML supports distributed and modularized knowledge bases through direct coupling via key references within a KB, *iri* pointers, and support for query languages. With messaging, it also supports loosely-coupled interface-based interaction using rule signatures and decoupled communication via event messages. The specification of Reaction RuleML 1.0 is being developed bi-schematically, ‘Rosetta’-style, in XML Schema and Relax NG.

¹⁴ <http://www.prova.ws>; <http://www.jdrew.org/ojdrew>; <http://www.jboss.org/drools/>; <http://lpis.csd.auth.gr/systems/emerald/>

¹⁵ <http://responder.ruleml.org>

¹⁶ See e.g. RuleTheWeb Firefox extension <http://ruletheweb.org> [5]

¹⁷ <http://ruleml.org/1.0/myng/>

¹⁸ <http://www.acumenbusiness.com> and <http://sourceforge.net/projects/s2red>

References

1. T. Athan and H. Boley. Design and Implementation of Highly Modular Schemas for XML: Customization of RuleML in Relax NG. In *RuleML America*, pages 17–32, 2011.
2. H. Boley. RIF RuleML Rosetta Ring: Round-Tripping the Dlex Subset of Datalog RuleML and RIF-Core. In *RuleML*, pages 29–42, 2009.
3. H. Boley, A. Paschke, and O. Shafiq. RuleML 1.0: The Overarching Specification of Web Rules. In *RuleML*, pages 162–178, 2010.
4. A. Giurca and E. Pascalau. JSON Rules. In *CEUR Workshop Proceedings of 4th Knowledge Engineering and Software Engineering (KESE)*, volume 425, 2008.
5. A. Giurca, M. Tylkowski, and M. Müller. RuleTheWeb: Rule-Based Adaptive User Experience. In *CEUR Workshop Proceedings of the 6th International Rule Challenge at RuleML 2012*, 2012.
6. Robert Kowalski and Fariba Sadri. A Logic-Based Framework for Reactive Systems. 2012. In these proceedings.
7. D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman, Amsterdam, 2002.
8. A. Paschke. ECA-LP / ECA-RuleML: A Homogeneous Event-Condition-Action Logic Programming Language. In *RuleML-2006*, Athens, Georgia, USA, 2006.
9. A. 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. A. Paschke. Rules and Logic Programming for the Web. In *Reasoning Web*, pages 326–381, 2011.
11. A. Paschke and H. Boley. Rule Markup Languages and Semantic Web Rule Languages. In A. Giurca, D. Gasevic, and K. Taveter, editors, *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*, pages 1–24. IGI Publishing, May 2009.
12. A. Paschke and H. Boley. Rules Capturing Events and Reactivity. In A. Giurca, D. Gasevic, and K. Taveter, editors, *Handbook of Research on Emerging Rule-Based Languages and Technologies*, pages 215–252. IGI Publishing, May 2009.
13. A. Paschke and H. Boley. Rule Responder: Rule-Based Agents for the Semantic-Pragmatic Web. *Int'l Journal Artificial Intelligence Tools*, 20(6):1043–1081, 2011.
14. A. Paschke and A. Kozlenkov. Rule-Based Event Processing and Reaction Rules. In *RuleML*, pages 53–66, 2009.
15. A. Paschke, P. Vincent, A. Alves, and C. and Moxey. Tutorial on Advanced Design Patterns in Event Processing. In *DEBS*, 2012.
16. A. Paschke, P. Vincent, and F. Springer. Standards for Complex Event Processing and Reaction Rules. In *RuleML America*, pages 128–139, 2011.
17. M. Rosemann and P. Green. Developing a Meta Model for the Bunge-Wand-Weber Ontological Constructs. *Inf. Syst.*, 27(2):75–91, April 2002.
18. M. Rosemann and M. Indulska. A Reference Methodology for Conducting Ontological Analyses. In *Proceedings of ER 2004 - 23rd International Conference on Conceptual Modeling*, pages 110–121. Springer, 2004.
19. K. Teymourian and A. Paschke. Semantic Rule-Based Complex Event Processing. In *RuleML 2009: Proceedings of the International RuleML Symposium on Rule Interchange and Applications*, 2009.
20. G. Wagner, G. Antoniou, S. Tabet, and H. Boley. The Abstract Syntax of RuleML – Towards a General Web Rule Language Framework. In *Web Intelligence*, pages 628–631. IEEE Computer Society, 2004.