JSON Rules

Adrian Giurca[1] and Emilian Pascalau[1]

Brandenburg University of Technology, Germany
{giurca,pascalau}@tu-cottbus.de

Abstract. This paper presents a JSON based rule language and its JavaScript-based rule engine towards providing Web 2.0 applications with rule-based inference capabilities. For interoperability purposes the Rule Interchange Format is used. While the rule engine is enough general, its main purpose is to execute production rules and Event-Condition-Action rules related to the web page DOM processing. This way the user's browsing experience will be enriched with the ability to modify on the fly the DOM of the current document as well as the browser user interface (Firefox).

1 Introduction

In the last 10 years business rules were employed to declaratively describe policies, business processes and practices of an enterprise. Applications in domains such as insurance, financial services, government, telecom, and e-commerce benefit greatly from using rule engines. Moreover, rules are becoming increasingly important in business modeling and requirements engineering, as well as in Semantic Web applications. In each of these fields different rule languages and tools are being used. At the same time the amount of Web 2.0 applications increases heavily. Actual technologies such as Asynchronous JavaScript and XML (AJAX) [8] allows the development of Rich Internet Applications (RIAs). This concept was introduced in [2] to denote a web application that typically runs in a web browser, and do not require software installation. Several Web 2.0 applications use heavily AJAX in order to provide desktop-like behavior to the user. The number of RIAs is increasing because of the broad bandwidth of today's Internet connections, as well as the availability of powerful and cheap personal computers. However, traditional ways of programming Internet applications no longer meet the demands of modern rule-enabled rich Internet applications. For example a highly responsive Web 2.0 application such as Gmail[1], might be much easily customized by using a declarative description such as rules.

The goal of this paper is to describe a rule language and a client-side rule engine. The rule language uses JavaScript Object notation(JSON) notation [5] as its main format. However, for interoperability purposes the Rule Interchange Format (RIF) [4] is used. The choice of using JSON is due to it's widely usage by JavaScript developers. JSON is used for rule descriptions as well as serialization

[1] http://mail.google.com

for data that is going to be transmitted over network. While the rule engine is enough general, its main purpose is to execute production rules and Event-Condition-Action rules related to the web page Document Object Model (DOM) processing. This way the user's browsing experience will be enriched with the ability to modify on the fly the DOM of the current document as well as the browser user interface (Firefox).

2 Related Work

While the ideas of RIAs are not new (see [2]) the rule-based RIAs proposals are quite recent. A project was started in May 2008 by Project 6 Research [2]. However, the goals of this project are limited to XPath processing i.e. rules conditions are similar with test from XSLT while the actions are not clearly specified. In overall, the concepts are far away to be clear and we did not see to much advance. Also this product is commercial and no demos are available. There are also concerns to emulate a rule parser in Adobe Flex framework[3] but the goal seems to be a client side Drools[10] parser.

The most advanced work seems to be in [12] (May 2008) where two-layer architecture for rule-enabled RIAs is described. This paper is a good starting point but as a general architecture document, it does not formally provide a Model-Driven Architecture like, platform independent model. In addition the paper is not focused on the rule language description neither to the client-side rule execution. This work was also related in [11].

3 The Rule Language

JSON notation combined with JavaScript function calls offers large capabilities to express various kinds of rules. Recall that we deal both with production rules and with Event-Condition-Action (ECA) rules i.e. rules of the form

```
Rule ID: ON EventExpression IF C1 && ... && Cn DO [A1, ..., Am]
```

where the event part is optional and denotes an event expression matching the triggering events of the rule; `C1`, ... `Cn` are boolean conditions using a Drools like syntax and [`A1`, ... `Am`] is a sequence of actions.

The metamodel of a JSON Rule is depicted in Figure 1.

Example 1 (Production Rule).

> For all elements of class `'note'` having as first child a `'ul'` change the first child background color to blue. Expressed in a logical form the above example looks like: $\forall x \exists y(Element(x) \wedge x.class =' note' \wedge y = x.firstChild \wedge y.nodeName =' ul') \rightarrow changeBackground(y,' blue')$

[2] `http://www.p6r.com/articles/2008/05/22/an-xpath-enabled-rule-engine/`

[3] `http://archives.devshed.com/forums/compilers-129/writing-a-rules-parser-in-actionscript-javascript-2370024.html`

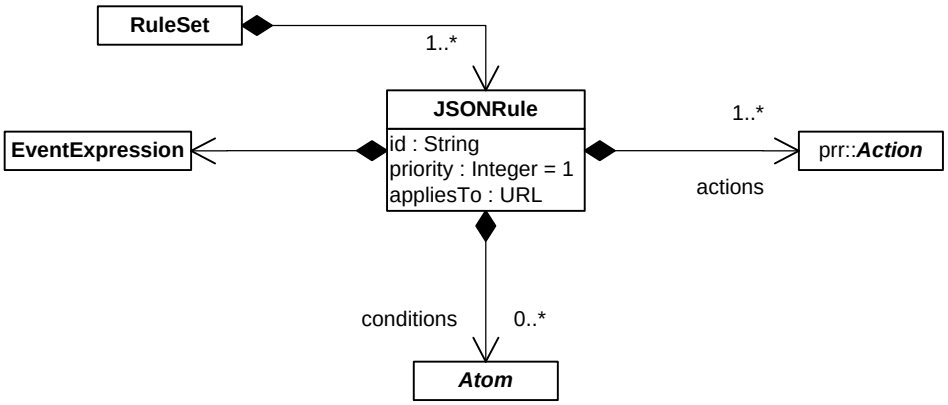

Fig. 1. Rules and Rulesets

```
{"id":"rule101",
  "appliesTo": ["http://www.example.org/JRules",
                "http://www.google.com/"],
  "condition": "$X:Element( class=='note',
                           $Y:firstChild)
                &&
               ($Y.nodeName == 'ul')",
  "actions":["changeBackground($Y, 'blue')"]
}
```

The above example shows that a JSON rule is first of all, a JSON object. The `appliesTo` property states that the rule will apply only on the specific indicated pages (URLs).

The `condition` uses a Drools like syntax and state that all elements with the class attribute equals with note (i.e. `$X:Element(class=='note')`) and with the first child an unsorted list (i.e. `$Y.nodeName == 'ul'`) must participate in the action.

The `action changeBackground($X, 'blue')` should be an available user-defined function call. If no such function is available then no action is performed.

JSON Rules are also ECA Rules i.e. they are triggered by events (see Example 4). Below we provide descriptions of the rule constituents.

3.1 Condition

A rule condition is always a conjunction of atoms. Empty conditions are interpreted as true conditions. As can be seen in the Figure 2, the language supports three kinds of atoms: *JavaScriptBooleanCondition*, *Description* and *XPathCondition*. The reader should notice that future extensions may involve other kinds of atoms.

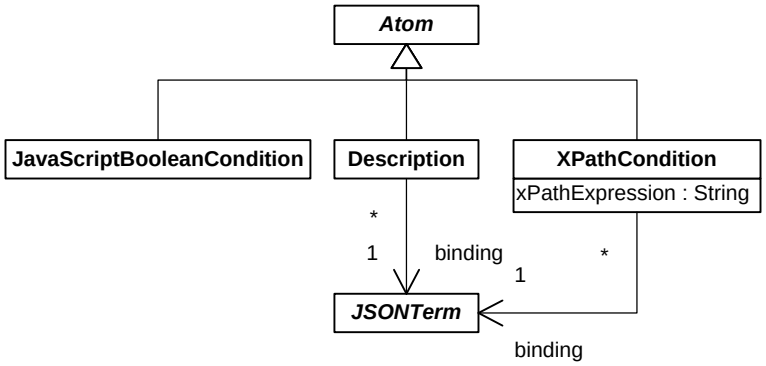

Fig. 2. The language atoms

JavaScriptBooleanCondition This is the simplest conditional atom. Any JavaScript boolean expression is allowed. For example `window.find('rule')` or `document.getElementById('id').value==10` are allowed.

Description This atom is related to the syntax of Drools pattern conditional. The metamodel of language descriptions is depicted in Figure 3.

A *Description* is bound to a *JSONTerm*, has a *type* and has a list of *constraints*. The *type* is one of the values described by *DescriptionType* enumeration. These values correspond to the node types defined in DOM Level 2 Core specification [4].

The *Description* offers two types of constraints PropertyRestriction and PropertyBinding.

- A *PropertyRestriction* (see Figure 3) describes a set of value restrictions to properties of the *JSONTerm* that are bound to it.
  - The string *property* encodes a property name of a property belonging to the corresponding bounded *JSONTerm*.
  - *operator* is relational one.
  - The *value* is either a *JSONTerm* (*Variable* or *DOM:Node*), or a *RegularExpression*, or a *String* or a *Number*.
- A *PropertyBinding* performs a variable binding of a property belonging to the related *JSONTerm*. After that this variable becomes available at the rule level (See example 1).

The condition below stands for all DOM entities of type *Element* that are text input elements with the value date of the form *yyyy-mm-dd*. Notice that for the value we used a regular expression that checks its format. Month can be only between 01-12. The regular value for day can not be greater than 31, and for month 02 the value for day can not be greater than 29.

[4] `http://www.w3.org/TR/DOM-Level-2-Core/idl-definitions.html`

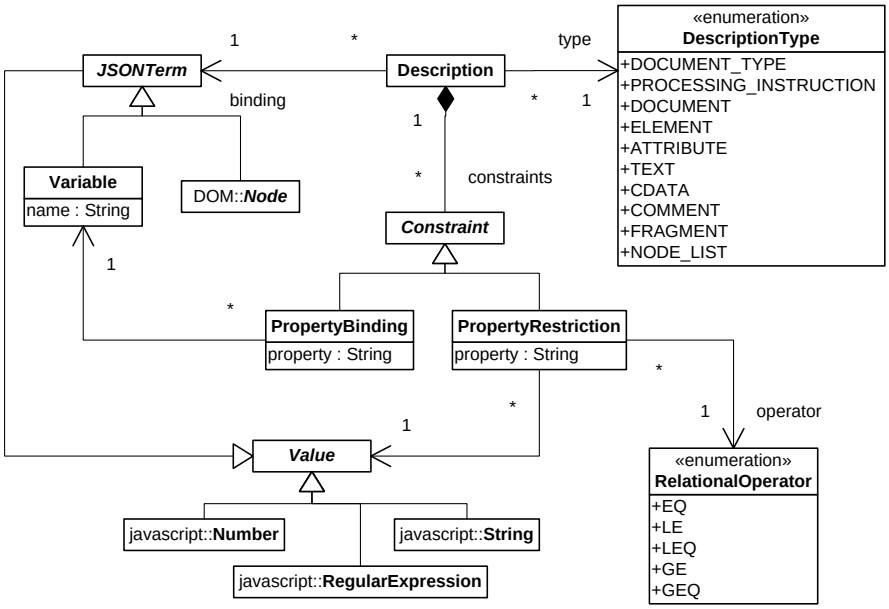

Fig. 3. Descriptions

Example 2 (Condition).

```
$X : Element(nodeName=="input", type=="text",
value=="match(^[0-9]{4}-(((0[13578]|(10|12))-(0[1-9]|[1-2][0-9]
        |3[0-1]))|(02-(0[1-9]|[1-2][0-9]))|((0[469]|11)-(0[1-9]
        |[1-2][0-9]|30)))$)")
```

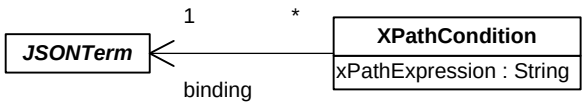

Fig. 4. The XPath Condition

XPathCondition The *XPathCondition* (see the metamodel in Figure 4) is the third type of conditional atom. This one compared with the other might be a little peculiar. For a usage example of this type of conditional we are going to use the example page below.

Example 3 (An XPath Condition). Consider the following page with the view depicted in Figure 5:

T1:row 1, cell 1	T1:row 1, cell 2
T1:row 2, cell 1	T1:row 2, cell 2
T2:row 1, cell 1	T2:row 1, cell 2
T2:row 2, cell 1	T2:row 2, cell 2

Fig. 5. Page view for XPathCondition example

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0Strict//EN"
          "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
 <head>
  <title>XPathCondition</title>
 </head>
 <body>
  <div id="_div1" >
   <table border="1">
    <tr> <td>T1:row 1, cell 1</td> <td>T1:row 1, cell 2</td> </tr>
    <tr> <td>T1:row 2, cell 1</td> <td>T1:row 2, cell 2</td> </tr>
   </table>
  </div>
  <div id="_div2">
   <table border="1">
    <tr> <td>T2:row 1, cell 1</td> <td>T2:row 1, cell2 </td> </tr>
    <tr> <td>T2:row 2, cell 1</td> <td>T2:row 2, cell 2</td> </tr>
   </table>
  </div>
 </body>
</html>
```

Such a conditional can be bound to a *JSONTerm* (Figure 3. Recall that a *JSONTerm* is either a *Variable* or a *DOM:Node*.

Assuming that we want to change the background for all rows in all tables in the DOM of the current page, using the *XPathCondition*, the condition is:

```
"$X in "html//table//tr"
```

The variable `$X` has been used along with the reserved word `in`. The meaning is *"forall $X in the collection..."* The evaluation of the *xPathExpression* returns a list of nodes (In the previous example, the evaluation of the *xPathExpression* returns 4 nodes).

On the other hand, if we want to change the background only for a specific node by using an *XPathCondition* the condition is:

```
{"nodeName":"tr",
 "firstChild":{"nodeName":"td",
               "textContent":"T2:row1, cell 1"
               }
} in "html//table//tr"
```

After evaluating the condition the background should be changed only for the first row of the second table (see Figure 5).

3.2 Actions

Our approach deals with standard actions of OMG Production Rule Representation (PRR), [9]. The reader should notice that any user-defined JavaScript functions can be called in the rule actions part. Below is the mapping of the PRR standard actions to our language:

PRR Standard Actions	**JSON Rules**
`AssignExp`	change properties of an element
`InvokeExp`	any JavaScript function call
`AssertExp`	insert a DOM node
`RetractExp`	remove a DOM node
`UpdateExp`	update a DOM node

Table 1. PRR Standard Actions and their representation

An *invoke action* is already provided in Example 1. It corresponds to a JavaScript function-call. The function must be available otherwise the action is ignored (not executed).

An *assign action* is usually intended to change the properties of an element. For example

```
$X.setAttribute("value", "25")
```

is an assign action changing the `value` attribute of an `input` element bounded to the variable `$X`. If `$X` is not bounded to an element allowing the attribute `value` then the engine will ignore such action.

An *assert action* is related to the creation of new nodes in the DOM e.g.

```
$X.appendChild(document.createElement("input"))
```

is an assert action.

A *retract action* is the inverse of the assert action i.e. deletes a node from the DOM.

An *update action* is usually related to the content update of a DOM node. For example

```
$X.replaceChild($Y,$Z)
```

is an assert action.

3.3 Event Expressions

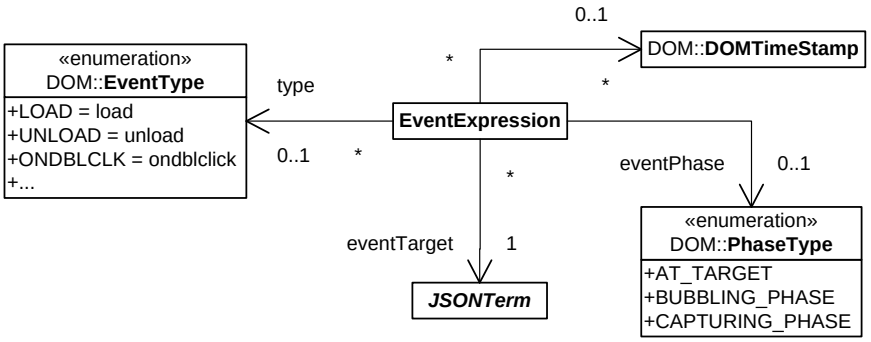

Fig. 6. Event Expressions

The JSON event expression is related to the *Event* interface specification in DOM Level 2 Events[5], therefore the properties of this expression have the same meaning as in the *Event* specification. At the runtime these properties of this expression are matched against the incoming DOM events and their values can be processed in the rule conditions and actions.

Example 4 (ECA Rule).

```
{"id":"rule102",
  "appliesTo":["http://mail.yahoo.com/"],
  "eventExpression": { "eventType": "click",
                       "eventTarget": "$X"
                     },
  "condition":" ($X.nodeName == 'a',
                $X.href=='match(showMessage\?fid=Inbox)')",
  "actions":["append($X.textContent)"]
}
```

The rule from example 4 concerns the Yahoo mail and states that when `click` event is raised, if the event came from an `a` element, and if the `href` property matches the regular expression (rudimentary check that the link is an inbox Yahoo message link) then call `append` function with the message subject as parameter.

3.4 Additional parameters

In addition to its main constituents a rule provides some other parameters:

[5] `http://www.w3.org/TR/DOM-Level-2-Events/`

– The *id* is required and denotes the unique identifier of a rule.
– The *appliesTo* property is required and holds a list of URLs on which the rule must apply e.g. the rule from Example 1 can be applied to the pages `http://www.example.org/JRules` and to `http://www.google.com/`.
– *Priority* expresses the order of a rule in a ruleset. If no value is provided for it, default value is "1". Based on priorities the rule engine must execute the rules in a down counting order (from greater values to lower values). The execution order is not relevant for rules having the same priority.

4 The Rule Engine

The main characteristics of the rule engine are:

– Is a forward chaining rule engine using a modified RETE algorithm (see [6] for the standard version) for production rules ;
– Uses the above rule language as well as RIF XML.
– Deals with two different types of rules: production rules and ECA Rules
– DOM events are processed as atomic events (i.e. no duration).
– Rules are stored locally or remote or both.
– The engine execute rulesets (a ruleset is the set of all rules referring to a specific URL).
– The RETE working memory is the document DOM itself. Rule property restrictions are matched against DOM entities (such as elements processing instructions, attributes an so on).

The component view of the engine is depicted in the Figure 7.

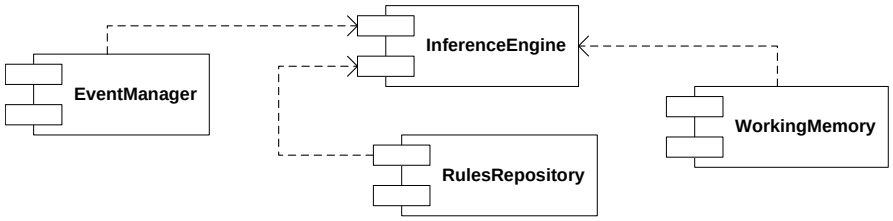

Fig. 7. Rule Engine

A UML state diagram describing the functionality of the engine is depicted in the Figure 8.

When the *InferenceEngine* is started it loads the corresponding rules from the *RuleRepository*. After the rules are loaded, the *EventManager* gets active and it listens for events from the *WorkingMemory*. When it receives events from the *WorkingMemory*, the *EventManager* informs the *InferenceEngine* about it. The InferenceEngine computes rules to fire. If rules are found then it fires them, and the WorkingMemory is changed. When no rules are computed then it stops.

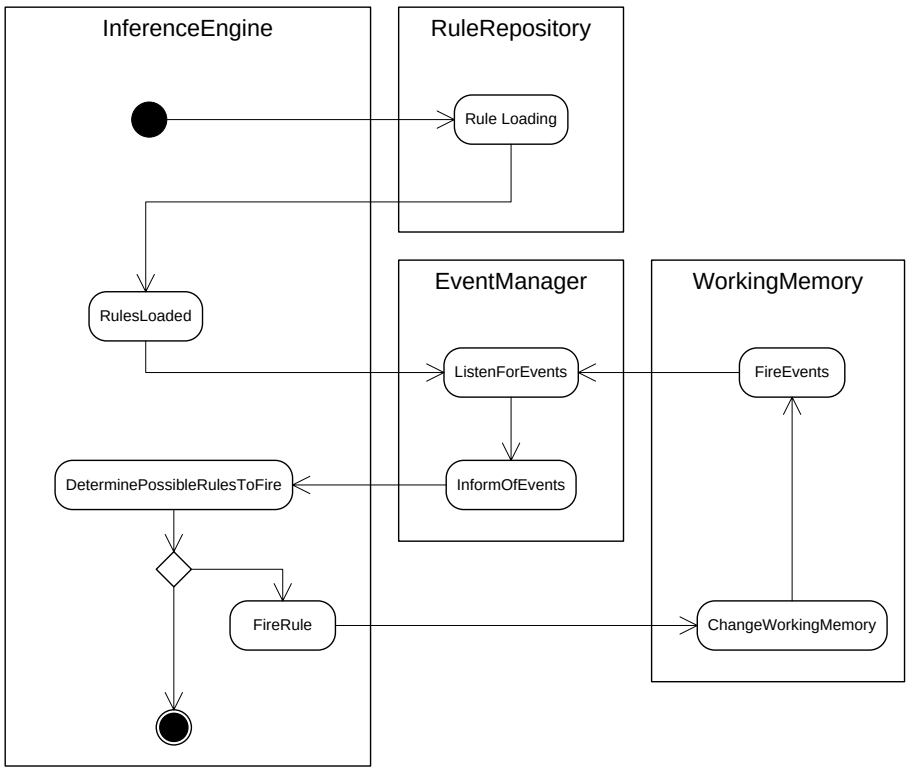

Fig. 8. The Rule Engine State Diagram

Opposed to usual rule engines, that stop their activity when no rules can be fired, this stops only when the WorkingMemory exists no more. When rules can not be fired, but the WorkingMemory still exists, the engine gets into an idle state, and waits to be informed about new events from the EventManager.

4.1 The Working Memory

Is represented by the DOM of the current page. The WorkingMemory is special because it is event based, DOM is changed through events.

4.2 The Inference Engine

Is the "brain" of the system. Based on the facts (DOM) of the Working Memory and based on the current page corresponding ruleset, it performs the matching operation and executes rules.

4.3 The Event Manager

The event manager is a combination between JavaScript and the Working Memory - DOM of the current page. The Event Manager takes advantage of the fact that the DOM itself is event based. All changes and in general all DOM events are reported to the main document object. This is based on the bubbling effect of DOM events.

4.4 The Rule Repository

Rules are stored in the repository. The repository can be local or remote. In both cases the storing language is JSON based as described in Section 3.

5 Conclusion and future work

This paper describes an approach of enriching RIAs with rule-based reasoning. JSON Rules provides the JavaScript engine with reasoning capabilities and the users can write their own rules. The rules syntax is based on JSON notation, therefore does not require high effort to accommodate it. Rules are simpler but powerful, their main goal being to change the DOM of the page they apply. The rule actions comply with the proposed OMG standard for production rules and are enough general to achieve all kind of DOM changes as well as any kind of side effects. The next immediate step will take into account the engine interaction with both user defined events and *XMLHTTPRequest* events to increase the power of the reaction rules to dynamically handle the page AJAX-based interactions. On a medium term JSON Rules should deal with metadata (with emphasis on RDF[7]) both for embedded metadata (such as RDFa [1]) and external metadata (such as RSS [3] and Atom[13]). In addition, sharing rules is a feature that can improve user experience and also, might spare him the time

of writing rules by himself, in the case he can find rules for the page being interested in. The user can share with his friend, or with everybody. Rule sharing goes hand in hand with rule publishing.

References

1. Ben Adida and Mark Birbeck. RDFa Primer: Embedding Structured Data in Web Pages. W3C Working Draft 17 March 2008. http://www.w3.org/TR/xhtml-rdfa-primer/.
2. Jeremy Allaire. Macromedia Flash MXA next-generation rich client. http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf, March 2002.
3. RSS Advisory Board. RSS 2.0 Specification. http://www.rssboard.org/rss-specification.
4. Harold Boley and Michael Kifer. RIF Basic Logic Dialect. http://www.w3.org/2005/rules/wiki/BLD, October 2007.
5. Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). http://tools.ietf.org/html/rfc4627, July 2006.
6. Charles Forgy. Rete – A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.
7. Klyne G. and Caroll J.J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-concepts/.
8. Jesse James Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php, February 2005.
9. OMG. Production Rule Representation (PRR), Beta 1. Technical report, November 2007.
10. Mark Proctor, Michael Neale, Michael Frandsen, Sam Griffith Jr., Edson Tirelli, Fernando Meyer, and Kris Verlaenen. Drools 4.0.7. http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html_single/index.html.
11. Kay-Uwe Schmidt, Jörg Dörflinger, Tirdad Rahmani, Mehdi Sahbi, Susan Thomas, and Ljiljana Stojanovic. An User Interface Adaptation Architecture for Rich Internet Applications. In *Proceedings of 5th European Semantic Web Conference 2008, ESWC 2008, Tenerife, Spain*, volume 5021 of *Lecture Notes in Computer Science*. Springer Verlag, June 2008. (accepted).
12. Kay-Uwe Schmidt and Ljiljana Stojanovic. From Business Rules to Application Rules in Rich Internet Applications. In *Proceedings of Business Information Systems 11th International Conference, BIS 2008, Innsbruck, Austria, May 5-7, 2008*, volume 7 of *Lecture Notes in Business Information Processing*, pages 447 – 458. Springer Berlin Heidelberg, 2008. http://dx.doi.org/10.1007/978-3-540-79396-0_39.
13. Atom WG. Atom Publishing Format and Protocol . http://tools.ietf.org/wg/atompub//.