

# Towards The Reactive Web

Master Thesis Report

Dominic Bosch  
Departement Mathematics and Computer Science  
University of Basel

October 21, 2013

**Abstract.** t.b.d.

## 1 Introduction

t.b.d.

## 2 Related Work

### 2.1 Rules Languages

[5] gave a good overview over existing approaches in 2009. In this section we examine different existing rule languages with respect to a simple

use case. We want the rule language react on the receipt of an email (event), check for a distinct email address (condition) and store it in a remote location, via a Web API (action). The email only contains the parts we require for this use case (the sender and a subject). A JSON representation of the email would be:

```
{
  "event": "email",
  "sender": "sender@mail.com",
  "subject": "An important message!"
}
```

### 2.1.1 RDF & XML

An early ECA Rule Language for XML repositories [4] was postulated in 2003 and was picked up by many researches afterwards. It was designed to react on insert and delete events within XML repositories and as an action change XML documents.

```
ON INSERT document('inbound_queue.xml')/mails/mail
IF $delta/sender[.='sender@mail.com']
DO DELETE document('inbound_queue.xml')/mails/mail;
LET $api = resource("www.webapi.com") IN
  INSERT ($api, newcontent,
    <content>New mail: {$delta/subject}</content>)
```

Now apart from implementing a rules engine, we would also need to add an XML document event manager which interpretes and pushes events into the XML file *inbound\_queue.xml*. Then again this instance would interpret the outputs of the ECA engine, which would theoretically manifest in other XML documents, and produce meaningful actions on remote hosts. This wouldn't be an architecture which has its focus on the solution of our use case and, as a result, add complexity and create an unnecessary overhead.

### 2.1.2 Notation 3

To make the lengthy RDF definitions smaller and more readable, Notation 3 [1] was designed and announced in 2005. Through the implies

operator( $\Rightarrow$ ) an "event" can be connected to an "action", both expressed in RDF's subject, predicate, object notation, which makes the expression of ECA rules a complicated and not very intuitive task. A solution to our use case would look as follows:

```
{ ?x :event "email". ?x :sender "sender@mail.com" }
=> { :webapi :newcontent ?x }
```

It's obvious that this language is used to express relations between entities and thus not really suitable for our use case, since we would require another interpreter to infer the actions. But concepts and ideas of the work that was done in these consortias could eventually still find influence into our solution.

### 2.1.3 XChange/Xcerpt

The rule language XChange [7] was the outcome of the REVERSE project and acted as an influence in many further researches. The language was designed to add reactive behaviour to a "static" web which is represented through XML resources. Thus we have action logics to alter such resources through insertions and deletions. Since we aim to utilize web API's for our rule language we need a more generic approach which adds flexibility in term of the API provided. But the thorough research done with the language XChange holds valuable concepts, especially in terms of temporal event composition. This could be a rule according to our use case:

```
TRANSACTION
in {
  resource { "http://www.webapi.com"},
  newcontents {{
    insert newcontent { var Mail }
  }}
}
ON
xchange:event {{
  xchange:sender { "http://mailserver.com" },
  var Mail -> email {{
    sender { "sender@mail.com" }
  }}
}}
```

```

    }}
END

```

But XChange is designed to access other resources in an action and thus provides powerful tools:

```

TRANSACTION
[... ]
ON
[... ]
FROM
  in {
    resource { "http://www.weather.com"},
    temperatures {{
      var T -> temperature {{
        datetime { "2013-10-20-08:00:00AM" }
      }}
    }}
  }
END

```

#### 2.1.4 JSON Rules

In 2008 *JSON Rules* [3] was introduced as a language to easily react on specific DOM tree compositions. The usage of JavaScript allowed them to provide simple functions which could be called directly by the actions, thus abstracting functionality from the language. This key concept found influence into our language as it allows different layers of abstractions. Through this it is possible to provide generic functions for expert user as well as very limited functions with only few possibilities for parameterization to be used by unexperienced persons. A drawback of this language is its binding to DOM tree events, where we would want to react on any events happening in the world. Also the temporal composition to complex events is not a subject of their work and needs further attention.

```

{
  "id": 0,
  "conditions": [
    {

```

```

    "type": "email",
    "constraints": [
      {
        "propertyName": "sender",
        "operator": "EQ",
        "restriction": {
          "type": "String",
          "value": "sender@mail.com"
        }
      },
      {
        "bind": "$S",
        "propertyName": "subject"
      }
    ]
  },
  "actions": [
    {
      "webapi('addcontent', $S)"
    }
  ]
}

```

#### **2.1.5 R2ML**

t.b.d.

#### **2.1.6 OWL**

t.b.d.

#### **2.1.7 SWRL**

t.b.d.

#### **2.1.8 ETALIS**

t.b.d.

### 2.1.9 ESPER

t.b.d.

### 2.1.10 KRL

[8] t.b.d.

### 2.1.11 (Reaction) RuleML

*RuleML* [2] [6]

Rules Languages	Classification
<b>Language XChange</b> Francois Bry, Paula-Lavinia Patranjan	<ul style="list-style-type: none"><li>• EU &amp; Swiss project</li><li>• Paradigm</li><li>• Event-driven reactivity</li><li>• Influences into all other research in the field of web reactivity</li><li>• Discontinued since 2008</li></ul>
<b>JSON Rules</b> Adrian Giurca, Emilian Pascalau	<ul style="list-style-type: none"><li>• JSON based rule language</li><li>• (DOM-) Event-based reactivity</li><li>• Discontinued since 2009</li></ul>
<b>RuleML</b> Harold Boley, Adrian Paschke	<ul style="list-style-type: none"><li>• XML based rule language</li><li>• Event-driven reactivity</li><li>• “Cloud Application Access”</li></ul>

Figure 1: Examined Rules Languages

### 2.1.12 (OO) jDrew

t.b.d.

### 2.1.13 Prova

t.b.d.

#### 2.1.14 Drools Fusion

t.b.d.

### 3 Conclusion

Most of the examined rule languages are designed for the interchangeability of rules between different service providers. We do not attempt to jump into this domain but we rather pick up important concepts to manifest web API's as first class citizens of our rule language. This allows the ad-hoc design and implementation of reactive rules between existing web API's without the need for their cooperation in setting up their endpoint in a special way.

### 4 Future Work

t.b.d.

### References

- [1] Tim Berners-Lee. Notation 3 Logic. <http://www.w3.org/DesignIssues/Notation3.html>, 2005. Accessed: 2013-10-21.
- [2] Harold Boley. The RuleML family of web rule languages. In *Principles and Practice of Semantic Web Reasoning*, pages 1–17. Springer, 2006.
- [3] Adrian Giurca and Emilian Pascalau, Json rules. *Proceedings of the Proceedings of 4th Knowledge Engineering and Software Engineering, KESE*, 425:7–18, 2008.
- [4] George Papamarkos, Alexandra Poulovassilis, Ra Poulovassilis, and Peter T. Wood. Event-Condition-Action Rule Languages for the Semantic Web. In *In: Workshop on Semantic Web and Databases*, pages 309–327, 2003.
- [5] A. Paschke and H. Boley. Rules Capturing Events and Reactivity, 2009.

- [6] Adrian Paschke, Harold Boley, Zhili Zhao, Kia Teymourian, and Tara Athan. Reaction RuleML 1.0: Standardized Semantic Reaction Rules. In Antonis Bikakis and Adrian Giurca, editors, *Rules on the Web: Research and Applications*, volume 7438 of *Lecture Notes in Computer Science*, pages 100–119. Springer Berlin Heidelberg, 2012.
- [7] Paula-lavinia Patranjan. *The Language XChange*. PhD thesis, Ludwig-Maximilians-Universität München, 2005.
- [8] P. Windley. *The Live Web: Building Event-Based Connections in the Cloud*. Cengage Learning PTR, 2011.



## Appendix

t.b.d.

## A ECA Rules Engine Resources

### A.1 Node.js Rules Engine Code

#### A.1.1 ecaserver.js

```
'use strict';  
var express = require('express');  
var qs = require('querystring');  
var engine = require('./ecainference');
```