Towards The Reactive Web

Master Thesis

Dominic Bosch
Departement Mathematics and Computer Science
University of Basel

December 16, 2013

Abstract. t.b.d.

Contents

1 Introduction

t.b.d.

1.1 Related Work

t.b.d.

2 Conceptual Model for Reactive Web Systems

t.b.d.

3 The "XYZ" Prototype System

Examples for XYZ:

- **DECADE (Demonstrative ECA Dedicated Engine)**
- **RECAST (Reactive ECA Service Trigger)**
- **PECAN (Productive ECA eNgine)**
- **ICECAP (Inet-Service Calls through ECA Paradigm)**

R:

- Reactive
-
-
-

S:

- System
- Services
-
-

T:

- Tenant
- Tutor
- Time
- Trigger
- Tender
- Tunnel
- Tinderbox
- Toolbox
- Toolmaker
- Transporter
- Transmitter

Other names:

- pecan (Productive ECA eNgine)
- decade (Demonstrative ECA Dedicated Engine)
  dedicated, derive
- decant
- recall (Reactive ECA Leverage)

- recap (Reactive ECA Paradigm)
- recast
- icecap (Inet-Service Calls through ECA Paradigm)
- theca: outer sheath of the pupa of certain insects
- hecate: (Greek mythology) Greek goddess of fertility who later became associated with Persephone as goddess of the underworld and protector of witches

4 Discussion & Results

Use Cases

Rules Languages	Classification
Language XChange Francois Bry, Paula-Lavinia Patranjan	• EU & Swiss project • Paradigm • Event-driven reactivity • Influences into all other research in the field of web reactivity • Discontinued since 2008
JSON Rules Adrian Giurca, Emilian Pascalau	• JSON based rule language • (DOM-) Event-based reactivity • Discontinued since 2009
RuleML Harold Boley, Adrian Paschke	• XML based rule language • Event-driven reactivity • "Cloud Application Access"

Figure 1: Examined Rules Languages

[5] gave a good oveview 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!"
}
```

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 interprete the ouptuts 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.

To make the lengthy RDF definitions smaller and more readable, Notation 3 [1] was designed and announced in 2005. Through the implies operator(=>) 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 eventully still find influence into our solution.

The rule language XChange [7] was the outcome of the REWERSE 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 evet 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
```

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)"
  ]
}
```

A most recent (2011) open-source development is the Kinetic Rules Engine together with the Kinetics Rule Language [8]. It is built for the purpose of adding reactivity to the cloud. The language is based on declarative syntax, enriched with imparative elements. But it is a tedious task to get into a whole new language and their caveats. *authorization?*

```
rule store_mail {
  select when mail newmail
      sender re#sender@mail.com#
      subject re#*# setting(subj)
    http:post("http://www.webapi.com/newcontent")
      with params = {
        "text": subj
      }
}
```

```
ruleset a2236x5 {
  rule register_temperature {
    select when temperature update
      if (event:attr("temp") > 20
        && ent:old_temp <= 20) then {}
      fired {
        raise explicit event temp_over_20;
      }
      always {
        set ent:old_temp event:attr("temp");
      }
  }

  rule temp_over_threshold {
    select when explicit event temp_over_20
      http:get("https://" + ent:credentials
        + "@probinder.com/service/27/save?companyId="
        + ent:companyID + "&context=" + ent:contextID
        + "&text=temp over 20 degrees.");
  }

}
```

The basis of *RuleML* [2] is datalog, a language in the intersection of SQL and Prolog. In 2012 the *Reaction RuleML* [6] language incorporated several different types of rules into the RuleML syntax, to establish a uniform syntax and interchangability of rules. *Reaction RuleML* is a valuable resource in terms of manifold research that has been done in the domain of rule languages, but the syntax is not user-friendly.

R2ML allows usage for RuleML together with many other dialects. Really!?

```
<Rule style="active">
  <on>
    <Event>
      <Atom>
        <Rel per="value">mail</Rel>
        <Var>sender</Var>
        <Var>subject</Var>
      </Atom>
    </Event>
  </on>
  <if>
    <Atom>
      <op><Rel>equals</Rel></op>
      <Var>sender</Var>
      <Ind>sender@mail.com</Ind>
    </Atom>
  </if>
  <do>
    <Atom>
      <oid><Ind uri="http://webapi.com"/></oid>
      <Rel>newcontent</Rel>
      <Var>subject</Var>
    </Atom>
  </do>
</Rule>
```

```
on mail
if sender="sender@mail.com"
do webapi->newcontent(subject)
```

Would be translated into:

```
{
  "event": "mail",
  "conditions": [
    { "sender": "sender@mail.com" },
  ],
  "actions": [
    {
      "api": "webapi",
      "method": "newcontent",
      "arguments": {
        "text": "$X.subject"
      }
    }
  ]
}
```

```
on weather->tempRaisesAbove(20)
do probinder->addContent(temp)
```

```
on emailyak->newMail
if FromAddress="dominic.bosch.db@gmail.com"
do probinder->newContent(TextBody)
```

```
on probinder->unreadContent
if serviceId=32
do probinder->markread(id),
   probinder->createContent(id, title, tab_name)
```

```
function call(args) {
  require('needle').post(
    'https://probinder.com/service/'
      + args.service + '/' + args.method,
    args.data,
    args.credentials
  );
};
```

```
function newContent(txt){
  call({
    service: '27',
    method: 'save',
    data: {
      companyId: '961',
      context: '17930',
      text: txt
    }
  });
}
```

```
on mail
do probinder->createContent(subject)
```

```
on mail
do probinder->call("27","save",
     ["961", "17930", subject]
   )
```

```
on probinder->unread
if serviceId=32
do probinder->setRead(id),
   probinder->makeFileEntry(service, id)
```

```
  "event": "emailyak->newMail",
  "condition": { "FromAddress": "dominic.bosch.db@gmail.com"},
  "actions": [
    {
      "module": "probinder->newContent",
      "arguments": {
        "content": "Received from EmailYak: $X.TextBody"
      }
    }
  ]
```

```
function newMail(callback) {
  needle.get('https://api.emailyak.com/v1/'+key+'/json/get/new/email/',
    function (error, response, body){
        var mails = JSON.parse(body).Emails;
        for(var i = 0; i < mails.length; i++) callback(mails[i]);
    }
  );
}
```


```
{

  "event": "emailyak->newMail",
  "ToAddressList": "test@mscliveweb.simpleyak.com",
  "FromAddress": "dominic.bosch.db@gmail.com",
  "TextBody": "Lengthy body [...]",
  "Subject": "Fwd: test subject",
  [...]

}
```


Most of the examined rule languages are designed for the interchangability 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.

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] 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.

[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.

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