Master Thesis
Preparation Report

# Towards The Reactive Web

Dominic Bosch
Departement Mathematics and Computer Science
University of Basel

July 30, 2013

**Abstract.** While identifying related work, it became clear that huge efforts in the research fields of event-driven architectures and user-driven mashup development has been done. On the other hand cloud application utilization is a most recent research field and continues to be a big challenge as they evolve. Moreover the use of cloud applications seems to be a nice feature, rather than founding the basis of research. To the best of our knowledge, no related work was found which funnels all these research fields into one, providing novel powerful ways to govern the web.

## 1 Introduction

In today's large, powerful and dynamic web, the big challenge is to retain manageable interactions between cloud applications, while adopting reactivity to them. To get a hold on this, we are tempting the next change in the evolution of the web: the live web, or reactive web. By considering cloud applications as event producers and action consumers we are able to apply a new level of abstraction to the web, which allows new perspectives and approaches to manifest the reactive web. Such architecture has the potential to overcome limitations that have been encountered during previous research.

# 2 Related Work

## 2.1 Reactivity in the Web

Research approaches to apply reactivity to the web has been done by several researchers in many fields. In [19] the authors provide an overview of general descriptions and classifications from different research efforts in terms of events, rules and reactiveness. The different research domains they point out are:

- Event/Action Logics, Transition Logics and Process Calculi. Used in [2] to specify complex actions
- Dynamic/Update/Transition Logics
- Production Rule Systems (if-do)
- Active Databases and ECA Rule Systems (on-if-do)
- Rule-Based Complex Event Processing (CEP) and Event Notification Systems

In contrast to standard ECA rules, which typically only have one global state, messaging reaction rules maintain a local conversation state that reflects the process execution state. This supports the performing of different activities within process instances managed in simultaneous conversation branches. Today, powerful event languages (such as *RuleML*) exist, which allow to express each of the afore mentioned categories.

In [9] semantic application vocabulary and business rules are examined, which together allow smart suggestions to the user in a rule-based system. Their approach bases on Distributed Object Model (DOM) interactions and mashups would have to be constructed indirectly via the DOM. Their work is interesting when allowing the users to customize their online context but it doesn't allow for cloud applications to be first-class citizens as part of on- and offline mashups.

From [31] follows, that enhancing a Service oriented Architecture (SOA) with an event-driven SOA (EDSOA), leads to more flexible and adaptive SOA applications that can be informed about states of neighbouring components. In their approach, events are only an aid to react on unexpected behaviour. But our research path leads us towards a system which builds entirely on events, using their expressive power as a basis.

## 2.2 Rules Languages

En excerpt of rule engines and languages is shortly introduced here. The recent trend seems to go towards funneling the different event-based approaches into CEP as highly expressive paradigm to incorporate all others. Of course more rule engines or languages exist, but they either go into similar categories as the presented ones, aren't developed anymore, or haven't been identified in related work as recent or active research.

### 2.2.1 XChange/Xcerpt

One notable outcome of research in the last decade is the language *XChange* [5, 22], which introduces reactivity to the web. *XChange* uses *Xcerpt* [26], a rule-based query and transformation Language for the web, to express web queries. *XChange* was part of the REWERSE research project, funded by the EU and Switzerland and is discontinued since 2008. But the paradigms provided by both XChange and Xcerpt found their way into virtually every further research that was later done in the field of reactivity in the web. It mainly influenced both *RuleML* and *JSON Rules* and through this the whole field.

### 2.2.2 JSON Rules

*JSON Rules* [10] has been invented due to an increasing need for rules in terms of semantic web applications and the emerging Rich Internet Applications (RIAs). *JSON Rules* is capable of expressing production rules (if-do) as well as ECA rules (on-if-do). The rules engine is a forward chaining rule engine using a modified RETE [8] algorithm. The RETE working memory in this rule engine is the (event-based) Document Object Model (DOM) itself. An example [17] of (DOM-)rule-based creation and execution of mashups illustrates the powerful aspects of *JSON Rules*. The current limitation to the DOM tree would require a generalization to adopt it also to other memory layouts. The development of the *JSON Rules* engine is discontinued.

### 2.2.3 (Reaction) RuleML

*RuleML* [4] is a XML-based rule specification standard to express both forward and backward rules for derivation, reaction, rewriting, messaging, verification and transformation. The building blocks of *RuleML* are predicates, derivation rules, facts, queries, integrity constraints and transformation rules. Its development is driven by the Rule Markup Initiative [3].

With *RuleML* being already a useful specification, *Reaction RuleML* [21] extends *RuleML* towards reaction rules and complex event/action messages, e.g. for CEP. It adds various kinds of production, action, reaction and knowledge representation (KR) temporal/event/action logic rules, as well as (complex) event/action messages. It consists of one general reaction rule form that can be specialized, e.g. into production rules, trigger rules, ECA rules or messaging rules. Three different execution styles (active, messaging and reasoning) of rules are incorporated. Definition of inbound or outbound event messages are used to interchange events and rule bases. A reaction rule can be globally or locally nested within other reaction or derivation rules. Additionally the *RuleML* Interface Description Language (*RuleML IDL*) was provided in the same paper, a sub-language of *Reaction RuleML* and allows the description of public rule functions as interfaces to hide program logic.

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

Figure 1: Examined Rules Languages

## 2.3 Rules Engines

### 2.3.1 (OO) jDrew

Java Deductive Reasoning Engine for the Web (*jDrew* [27]) is a reasoning engine written in Java for definite clause reasoning. jDrew can be embedded into larger systems through its APIs. Object-Oriented jDrew (*OO jDrew* [1, 28]) is a Java based rule engine, it serves as a reference implementation of *RuleML*. This project seems not to be very actively developed.

### 2.3.2 Prova

*Prova* [12] is an expressive rule language and engine, both written in Java, with a main orientation to ECA rules. It uses backward-reasoning logic to formalize decisions in terms of derivation. Forward-directed messaging of reaction rules supports distributed event and action processing. It allows dynamic access to external data sources and is used by the authors of [20, 32] for the *RuleResponder's* proof of concept for transformations between different rule languages over *RuleML*. *Prova* seems to be discontinued since early 2013.

### 2.3.3 Kinetic

The Kinetic Rules Engine (KRE) is a platform presented in [29]. It is realized in Perl and uses its very own rule language, the Kinetic Rules Language (KRL). It is laid out to support CEP as well as a tight coupling with the user's browser through plugins or libraries loaded via the webpage. It allows the access to remote resources and the processing of such data before passing it along to internal storage or again external resources, such as cloud applications. A live system [30] is available for testing and if desired also for productive use. Creating an own instance is quite a challenge due to it's numerous libraries. KRL needs quite some efforts to get used to and can't be entrusted to inexperienced users, thus a layer on top of this system would have to be implemented for our purposes.

### 2.3.4 Drools Fusion

*Drools Fusion* [23] is part of the jBoss open source community and allows the application of CEP and development in an eclipse-based IDE. Recently *Drools 5* introduced the Business Logic integration Platform which provides a platform for Rules, Workflow and Event Processing. *Drools Fusion* has its own rule language, *Drools Rule Language (DRL)* This system has quite a heavy foot print, but active development is promising for a certain future stability.

### 2.3.5 Rule Responder

*Rule Responder* [20] is a project to extend the Semantic Web towards a Pragmatic Web infrastructure for collaborative human-computer networks, which they call an architecture of a Pragmatic Agent Web (PAW). It supports the formation of virtual groupings and allows semi-automated agents with their individual contexts, decisions and actions. The authors postulate agents empowered with automatic rule-driven data transformation, decision derivation from existing knowledge and reaction according to changed situations or occurred events. The work done in this project concentrates on a layer on top of a rule engine and language, and thus allows for a combination of arbitrary rule-based systems via their framework. This is achieved through the usage of general message oriented communication interfaces and a platform-independent rule interchange format (*RuleML*).

The authors of Rule Responder built their reference system [18] on top of the Mule [13] open-source Enterprise Service Bus (ESB) which acts as a communication middleware. The decision to use Mule was made because it goes beyond the typical definition of an ESB by providing a distributable object broker to manage all sorts of service components. Each agent runs its own arbitrary rule engine. For demonstration purposes *Prova* and *OO jDrew* were used to demonstrate the rule interchange between different rule engines.

As research continued in terms of reaction rules and *Rule Responder*, the authors of [32] showed the adoption of event paradigms to support scientific workflow execution. In their work they point out the limitations of ECA frameworks when adapted to their use case. For highly distributed and loosely coupled scientific workflows, complicated conditional procedures and rules, which can also have local scopes, are required. This shows us their work is going towards large distributed systems with a highly developed rule language that subsumes research from several fields.

## 2.4 Mashups

In [15], one of the founders of *JSON Rules* proposes to look at mashups as user-behaviour in a certain context. A mashup, to achieve a user-defined goal, is modelled via Unified Modeling Language (UML) as a map containing contexts and behaviour descriptions. Concepts are defined as unit of knowl-

| Rules Engines | Classification |
|---|---|
| **OO jDrew**<br>Marcel Ball, Harold Boley, David Hirtle, Jing Mei, Bruce Spencer | • Java-based reference implementation of RuleML<br>• Event-driven reactivity<br>• Very slow development |
| **Prova**<br>Kozlenkov, Paschke | • Java-based rules engine<br>• Rules Language represented through POJO's<br>• Event-driven reactivity<br>• Cloud Application Access through java<br>• Maybe discontinued since early 2013 |
| **Kinetic**<br>Phil J.Windley | • Perl based rules engine<br>• Kinetic Rules Language (KRL)<br>• Event-driven reactivity<br>• Cloud Application Access built into KRL |
| **Esper** | • Java based rules engine<br>• Event Processing Language (EPL)<br>• Event-driven reactivity<br>• Cloud Application Access through Java<br>• CEP and event streams |
| **Drools Fusion**<br>jBoss | • JBoss platform<br>• Drools Rule Language (DRL)<br>• Event-driven reactivity<br>• Cloud Application Access through Java |
| **Rule Responder**<br>Adrian Paschke, Harold Boley | • Java-based distributed rule engine, on top of ESB<br>• Event-driven reactivity<br>• Cloud Application Access through Java<br>• “Mashups” |

Figure 2: Examined Rules Engines

edge created by unique combination of characteristics. Contexts are defined as a set of concepts. A mashup is then defined as set of contexts behaviour, where behaviour consists of rules and processes. The conceptual work done in this paper is interesting but the promoted example isn't accessible.

It is also important to understand what users expect from service mashups, in order to provide a useful platform to them. In [14] research is done in identifying user perceptions of services, their composition, user working ways and expectations towards a composition tool. They come up with a set of recommendations to aid the development of mashup environments. A survey [7] that went into a similar direction categorizes the different frameworks for user driven mashup development as based on:

- Programming Paradigm

- Scripting Languages
- Spreadsheets
- Wiring Paradigm
- Programming by Demonstration
- Automatic Creation of Mashups

But even though large efforts are made in all these research fields, inexperienced users are still not able to build mashups without knowledge about numerous aspects of the framework or programming.

### 2.4.1 useKit

The idea of *useKit* [25] missions shows us the potential of user-manageable cloud application mashups. While their approach is not event-based, it can be regarded as a base for the web's evolution towards user-programmable reactive cloud application mashups.

### 2.4.2 DashMash

The DashMash [6] platform is an approach to give end-users the graphical tools in a browser to mash up web applications in a dashboard. A resource of (for stability reasons) trimmed services (such as GoogleMaps or TripAdvisor), filters, viewers and generic components is accessible to the users. DashMash uses an event-driven model of the presentation level, similar to a JSON Rules approach in [17]. There are events sent by the client to the server, but they are only used to update all viewers with the actual data the user is looking at.

| **Mashups** | **Classification** |
|---|---|
| **useKit**<br>Sven Rizzotti, Helmar Burkhart | • Mashups<br>• Cloud Applications Access<br>• User-friendly |
| **DashMash**<br>Cappiello. Matera, Picozzi, Sprega, Barbagallo, Francalanci | • Web based platform<br>• Mashups<br>• User-friendly<br>• Cloud Application Access only over DOM |

Figure 3: Examined Mashup Approaches

# 3 Use Case Study

In order to verify some of the identified related work, use cases around the successor of useKit [25] (ProBinder [24]) have been derived and investigated. Three use cases have been identified to verify the feasibility of certain existing infrastructures.

## 3.1 Binder Watcher

Binder Watcher is about binders being watched and actions that are taken after certain changes to a binder. Users of ProBinder, which are involved in many different companies and project binders, tend to be confronted with a large amount of information. It is a tedious task to get the user's context back into a clean state, where the ProBinder system is ready to reflect new recent changes in an optimal way to the user. By allowing the users to identify resources (binder tabs in this case, but it could also be complete binders, persons, companies, ... ) of interest, the user task can be automated to a certain extent. As soon as changes are made to the resources of interest, they are marked as read and summarized. These summaries are then provided to the user, which allows him to identify the most important changes.

The Binder Watcher use case was implemented in KRL (see Appendix A) and provided the important insight that the realization of such a use case in an ECA is a time-consuming challenge.

## 3.2 Activating Legacy Resources

To bring reactivity to the static web, we have to enhance legacy resources, which are not event-ready. Changes in such resources have to be detected via e.g. web spiders or crawlers and transformed into events that can be processed by the rules engine. Such an auxiliary cloud application would turn the static web into a reactive system and free a future architecture from incorporating the pulling of events, thus allowing it to concentrate on its main business, i.e. event handling.

### 3.2.1 From the Web to the Cloud

To make the static web active, an *activator* monitors a predefined remote resource, e.g. a public ProBinder tab, accessed via the normal webpage, not the API. Changes to the tab are detected and pushed into the event-based system as events. In our case we built a reference system on top of *node.js*. These events then have to be processed according to rules and finally actions are invoked, which could also be events again. For our study the action taken was again *ProBinder* which was used to create an entry in another binder, this time via the API. This use case was mainly used to proof the accessibility of ProBinder via a remote system and the activation of the static web.

### 3.2.2 From the Cloud to the Web

Since cloud applications already provide quite a lot of notifications, they offer themselves as event producers. But most of todays notifications are received by pulling. Only few offer the possibility to register webhooks where they would push events to. The number of cloud applications that offer webhooks will most likely increase in the future and continue to make the web more reactive. But for now after dealing with the static web in the use case above, we need to deal with a semi-reactive web as well. These pull requests to such resources could again be taken care of by introducing an *activator*, which polls periodically for updates and pushes them into the even-based system. Now the system is able to check these events for conditions and invokes actions according to existing rules.

If we have the static web as an action destination, the question is how a static resource is going to react to such an action invocation. There are basically three possibilities to that, first we could implement an interface to the provider of the static resource in order to really change the resource, visible for every body in the web. Secondly, we simulate reactivity through our system. Actions taken on behalf of such a resource are stored in an internal state representation of the resource and if certain rules are met, the simulator could produce again events. Each user that is connected to our system, would then be able to experience this reactivity of static resources, and eventually also if other users interacted with the system. The third possibility is, rules could also run in a lightweighted architecture on the client browser, which influences the DOM tree to existing rules. Like this it would be easier to maintain a private view for users, but state changes could only be maintained during the browser sessions. Storing these state changes on the server side of the event-based system could overcome that shortfall and a combined solution seems to be a good proof of concept strategy.

### 3.2.3 Node.js

In order to implement some proof of concept examples, *node.js* was used as a basis to implement a custom module. *node.js* is a powerful tool to build cloud applications. Because of its modularity and the package manager, slim instances can be run. Its main purpose is to be used as a webserver. By incorporating JavaScript's asynchronous design of callback functions into their architecture, locking situations are basically omitted. Thus it is a highly-scalable, asynchronous, event-driven architecture which found broad acceptance in the open-source comunity as well as for enterprises.

The package manager *npm* can be used to maintain dependencies of a custom *node.js* module to other modules. This helpful feature ensures that everybody working with the user-defined module uses the same external modules and the respective versions. Also project repositories do not need to store the dependent libraries through this mechanism, thus saving space and traffic. A name and a version are mandatory for a *node.js* module. Knowing the name, e.g. diff, one can look up the latest version in the *npm* repository by issuing following command:

```
$ npm show diff version
npm http GET https://registry.npmjs.org/diff
npm http 200 https://registry.npmjs.org/diff
1.0.5
```

Now this dependency, together with the version one would like to use, can be inserted into the own project description file as shown below. For our small use case proof of concept project we only had to define the project description file called package.json in the root folder of the project:

```
{
        "name": "msc-apps",
        "description": "Proof of concept apps",
        "version": "0.0.1",
        "private": true,
        "dependencies": {
        "request": "2.22.0",
        "express": "3.3.4",
        "diff": "1.0.5"
        }
}
```

Everybody who downloads this project needs to download the dependencies before using the module. This is done by running following command in the root directory of the project:

```
$ npm install
```

If any dependencies change in the future the same command can be run again to update the local project. A simple Hello World example of *node.js* looks like this:

```
var http = require('http');
var server = http.createServer(function (request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.end("Hello World\n");
});
server.listen(8000);
```

After executing this by typing (if the above code was stored in a file called index.js)

```
$ node index.js
```

the webserver is accessible on port 8000 (by typing localhost:8000 in the browser bar), and responds with "Hello World".

# 4 Conclusion

In [16], the founders of *JSON Rules* [10] describe a lightweight architecture that allows to react and proact on behalf of events in the ontology of web browsers. *JSON Rules* is promising for our work because of its lightweight architecture and specialization on production and ECA rules. But the existing working memory architecture needs to be generalized to allow a different environment, other than just the DOM tree. The existing architecture could be used to allow the user to create local rules that do not access remote systems and thus runs into authentication issues.

Other approaches are server side rule engines either written in Perl or Java, where powerful tools are provided to process events and invoke manifold actions. But these approaches all lack an abstraction layer to introduce the programmability of the reactive web to a large audience of inexperienced users. Also, current approaches concentrate on data flows instead of event flows, thus not incorporating reactivity to the web.

With *RuleML* a powerful, interchangeable expression language for event-based systems is present. It paves the way for distributed rule engines, such as one running in the browser and another on a server for cloud application access. But a rule engine that incorporates the requirements of a lightweighted architecture and user-readiness is missing. Under related work, [14] pointed out the difficulties of inexperienced users to tackle the execution flow. This issue arose from their approach of displaying all services as very similar UI's. By introducing an event-based system, event producers would be clearly different from action consumers. This would address this issue in an user-intuitive way.

Large systems such as *RuleResponder* weave stubs or proxies of existing service into a message oriented middleware (MoM). We envision the web itself is used as the middleware. Through this a lightweighted and performant event-based architecture can be realized, which allows the orchestration of existing web and cloud applications.

# 5 Future Work

Developing an event-driven architecture which regards cloud applications as event-based first-class citizens and allows for an intuitive user-driven mashup development is a research field that has, to the best of our knowledge, not been addressed yet. Looking at cloud-based applications as event producers and action consumers gives new ways to bring reactivity into the existing web. Such representations require a rule-based system that allows their interweaving. Not solely the interaction between cloud applications should be addressed, but also with the browser itself, since it is a tool which is predominantly used to access the web. This would empower the user to predefine influences and interactions on existing cloud applications before they are accessed, providing novel powerful ways to govern the web.

Since the vision of on- and offline rules can't be covered with a single server application, the utilization of a fast, flexible and widely used technology such as JavaScript to tackle this challenge seems to be favorable future work. JavaScript was mainly invented for browsers and is spread all over the web by now. Additionally, applications such as *node.js* [11], bring JavaScript to the server-side and tear down the communication efforts between cloud applications through JSON messages which are directly understood by modern browsers and cloud applications. Preferably a lightweighted rules engine would be used to run the user-generated mashups. The KRE suits the demand for a certain coupling between the users browser and the remote rules engine to provide a powerful system. On the other hand the rules engine is not (yet) well documented, not lightweighted and forged in Perl, a programming language that wasn't encountered during the research for related work on rule based systems.

Sharing and thus exchanging of rules gives new ways for collaboration and the possibility for expert users to aid less experienced ones, giving them the chance to catch up with the future platform. In terms of usability an easy to understand way to create rules would have a large benefit. We envision a graphical toolkit that empowers users to build their own complex event-based cloud application mashups, powerful RIAs. A first approach could be to write rules in a certain language, e.g. *RuleML* or *JSON Rules*, then simplify the vocabulary until meaningful graphical representations are possible.

It is striking but not surprising, that all related work we looked at, only used public cloud applications, thus omitting the challenge of authorization and the safety of the user's private context. If users are provided the tools to access public cloud APIs and create mashups with them, they are going to benefit from this. But mostly users these days are traveling in the web within their private, secured context. Thus the access to such resources are providing even more power- and meaningful tools to the user. The future reactive web requires cloud applications that allow the registration of webhooks in order for the rules engine to receive events from resources other than the own system, thus using the internet as middleware for communication of push events.

# References


[1] Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer. The OO jDREW reference implementation of RuleML. In *Proceedings of the First international conference on Rules and Rule Markup Languages for the Semantic Web*, RuleML'05, pages 218–223, Berlin, Heidelberg, 2005. Springer-Verlag.

[2] E. Behrends, O. Fritzen, W. May, and F. Schenk, Embedding Event Algebras and Process for ECA Rules for the Semantic Web. *Fundam. Inf.*, 82(3):237–263, August 2008.

[3] H. Boley and S. Tabet. The Rule Markup Initiative. `http://ruleml.org`. Accessed: 2013-07-07.

[4] Harold Boley. The RuleML family of web rule languages. In *Principles and Practice of Semantic Web Reasoning*, pages 1–17. Springer, 2006.

[5] Francois Bry and Paula lavinia Patranjan, Reactivity on the Web: Paradigms and Applications of the Language XChange. *J. of Web Engineering*, 5:2006, 2005.

[6] Cinzia Cappiello, Maristella Matera, Matteo Picozzi, Gabriele Sprega, Donato Barbagallo, and Chiara Francalanci. DashMash: A Mashup Environment for End User Development. In Sören Auer, Oscar Díaz, and GeorgeA. Papadopoulos, editors, *Web Engineering*, volume 6757 of *Lecture Notes in Computer Science*, pages 152–166. Springer Berlin Heidelberg, 2011.

[7] Thomas Fischer, Fedor Bakalov, and Andreas Nauerz. An Overview of Current Approaches to Mashup Generation. In *Wissensmanagement*, pages 254–259, 2009.

[8] Charles Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligences*, 19(1):17–37, 1982.

[9] A. Giurca, M. Tylkowski, and M. Mueller. RuleTheWeb!: Rule-based Adaptive User Experience, 2012.

[10] Adrian Giurca and Emilian Pascalau, Json rules. *Proceedings of the Proceedings of 4th Knowledge Engineering and Software Engineering, KESE*, 425:7–18, 2008.

[11] Joyent Inc. node.js. `http://nodejs.org/`. Accessed: 2013-07-07.

[12] A. Kozlenkov and A. Paschke. Prova Rule Language. `https://prova.ws/`. Accessed: 2013-07-07.

[13] R. Mason. muleESB. `http://www.mulesoft.org`. Accessed: 2013-07-07.

[14] A. Namoun, T. Nestler, and A. De Angeli. End User Requirements for the Composable Web, 2010.

[15] E. Pascalau. Mashups: Behavior in Context(s), 2011.

[16] Emilian Pascalau and Adrian Giurca, A Lightweight Architecture of an ECA Rule Engine for Web Browsers. *Proceedings of 5th Knowledge Engineering and Software Engineering, KESE*, 486, 2009.

[17] Emilian Pascalau and Adrian Giurca. A Rule-Based Approach of Creating and Executing Mashups. In Claude Godart, Norbert Gronau, Sushil Sharma, and Gérôme Canals, editors, *Software Services for e-Business and e-Society*, volume 305 of *IFIP Advances in Information and Communication Technology*, pages 82–95. Springer Berlin Heidelberg, 2009.

[18] A. Paschke. Rule Responder - Rule-based Semantic Agent Architecture. `http://www.corporate-semantic-web.de/rule-responder.html`. Accessed: 2013-07-14.

[19] A. Paschke and H. Boley. Rules Capturing Events and Reactivity, 2009.

[20] A. Paschke, H. Boley, B. Craig, and A. Kozlenkov. Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web, 2007.

[21] 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.

[22] Paula-lavinia Patranjan. *The Language XChange*. PhD thesis, Ludwig-Maximilians-Universität München, 2005.

[23] RedHat. Drools - The Business Logic integration Platform. `http://www.jboss.org/drools`. Accessed: 2013-07-07.

[24] S. Rizzotti. ProBinder - Your secure online teamwork platform. `https://probinder.com/`. Accessed: 2013-07-07.

[25] Sven Rizzotti and Helmar Burkhart. useKit-lightweight mashups for the personalized web, 2010.

[26] Finn Sebastian Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. PhD thesis, Ludwig-Maximilians-Universität München, 2004.

[27] B. Spencer. jDREW. `http://sourceforge.net/projects/jdrew/`. Accessed: 2013-07-07.

[28] B Spencer, T. Athan, and B. Craig. OO jDREW. `http://extoojdrew.weebly.com/index.html`. Accessed: 2013-07-07.

[29] P. Windley. *The Live Web: Building Event-Based Connections in the Cloud*. Cengage Learning PTR, 2011.

[30] P.J. Windley. kynetx apps. `http://apps.kynetx.com/`. Accessed: 2013-07-07.

[31] Chunyang Ye and Hans-Arno Jacobsen. Event Exposure for Web Services: A Grey-Box Approach to Compose and Evolve Web Services. In Mark Chignell, James Cordy, Joanna Ng, and Yelena Yesha, editors, *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 197–215. Springer Berlin Heidelberg, 2010.

[32] Zhili Zhao and Adrian Paschke. Event-Driven Scientific Workflow Execution. In Marcello Rosa and Pnina Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 390–401. Springer Berlin Heidelberg, 2013.

# Appendix A  Binder Watcher KRL code

```
ruleset a2236x4 {
  meta {
    name "ProBinder Flag Notification Handler"
    description "This is a first example on how to react on ProBinder Events"
    author "dominic.bosch"
    //ProBinder IDs:
    // userID: 10595
    // companyID: 643
    // contextID: 16694
    // followerID: 12613

    logging on
  }

  dispatch {}

  global {}

  // Reset all entitiy variables
  rule resetAll {
    select when probinder resetall
      send_directive("Full Reset");
      fired {
        clear ent:userID;
        clear ent:companyID;
        clear ent:contextID;
        clear ent:credentials;
        clear ent:followers;
        clear ent:newContents;
        clear ent:summary;
        clear ent:temp;
      }
  }

  // reset the unread content data structures
  rule reset {
    select when probinder reset
      send_directive("Reset, user credentials and followers still kept");
      fired {
        clear ent:newContents;
        clear ent:summary;
        clear ent:temp;
      }
  }

  // The user registers himself with email and password for the ProBinder API...
  rule register_user {
    select when probinder register
      if (event:attr('userID').as("str") neq 'null'
          && event:attr('companyID').as("str") neq 'null'
          && event:attr('contextID').as("str") neq 'null'
          && event:attr('email').as("str") neq 'null'
          && event:attr('password').as("str") neq 'null') then {
        send_directive("user registered");
```

```
    }
    fired {
      set ent:userID event:attr('userID');
      set ent:companyID event:attr('companyID');
      set ent:contextID event:attr('contextID');
      set ent:credentials uri:escape(event:attr('email')) + ":" + uri:escape(event:attr('password'));
    }
}

// The user sent an event that tells us he wants to follow somebody
rule new_user_to_follow {
  select when probinder newfollower
    pre{
      listFollowers = ent:followers || {};
      newfollower = event:attr('followerID').as("str");
      listFollowers = listFollowers.put([newfollower], "true");
    }
    if (event:attr('userID') == ent:userID
        && newfollower neq "null") then {
          send_directive("New ProBinder User added to followers");
    }
    fired{
      set ent:followers listFollowers
    }
}

// Let the KRE check ProBinder for new unread content and process it immediately
rule check_for_unread_content {
  select when probinder check
    pre {
      r = http:get("https://" + ent:credentials + "@probinder.com/service/36/unreadcontent");
      arr = r{"content"}.decode();
    }
    send_directive("Checked ProBinder for unread content, found: " + arr.length());
    fired {
      set ent:newContents arr;
      raise explicit event processnewcontents;
    }

}

// Work (new unread content) from ProBinder to process
rule process_new_contents {
  select when explicit processnewcontents
  // Process only the unread contents from people we are following,
  // filter condition omits unnecessary rules invocation
  foreach ent:newContents.filter(
    function(d) {ent:followers.pick("$."+d.pick("$.userId")) != null}
  ) setting(nc)
    pre {
      s = ent:summary || {};
      cid = nc.pick("$.id");
      r = http:get("https://" + ent:credentials
        + "@probinder.com/service/2/get?id=" + cid
        + "&service=" + nc.pick("$.serviceId"));
      arr = r{"content"}.decode();
```

```
      userid = arr.pick("$.userId");
      storeKey = arr.pick("$.lastModified");
      truncStr = arr.pick("$.text");//.extract(re/^.{100}/gi); // should shorten the text...

    //TODO Process different kind of unread contents differently
      str = {"content": truncStr}; //[0]
      s = s.put([userid, storeKey], str);
    }
    http:get("https://" + ent:credentials + "@probinder.com/service/2/setread?id=" + cid);
    always {
      set ent:summary s;
    }

}

rule send_summary{
  select when probinder heartbeat
  always {
    clear ent:temp;
    raise explicit event filltemp;
  }
}

rule fill_temp{
  select when explicit filltemp
  always {
    set ent:temp ent:summary;
    raise explicit event mergecontent;
  }
}

// When somebody sends a periodic heartbeat, this summary is produced
// The periodic invocation of this rule might be possible to implement in the KRE
rule merge_content {
  select when explicit mergecontent
  foreach ent:temp setting (userID)
    pre {
      s = ent:temp;
      userBulk = s.pick("$."+userID);
      sumry = userBulk.pick("$..content").join(" ");
    }
    http:get("https://" + ent:credentials + "@probinder.com/service/27/save?companyId="
      + ent:companyID + "&context=" + ent:contextID + "&text=test");
    send_directive("Stored summary in your predefined binder:" + sumry);

}

rule print_summary {
  select when probinder printsum
    send_directive(ent:summary);
}
}
```