# JSON Rules - The JavaScript Rule Engine

Emilian Pascalau[1] and Adrian Giurca[2]

[1]Hasso Plattner Institute, Germany,
emilian.pascalau@hpi.uni-potsdam.de
[2]Brandenburg University of Technology, Germany,
giurca@tu-cottbus.de

**Abstract.** TOOL PRESENTATION: There is a considerable browser potential in being able to easily wire together different services into new functionality. Usually, developers use JavaScript or related technologies to do browser programming. This short paper presents, JSON Rules, a JavaScript rule engine running Event-Condition-Action rules triggered by Document-Object-Model Events.

## 1 Introduction

The Rule Engine implementing the JSON Rules [1] language was designed to fulfill at least the following requirements:

- create and execute rules in a Web browser
- support for ECA and PR rules
- forward chaining rule engine, influenced by the RETE algorithm;
- process atomic event-facts;
- the Working Memory contains beside regular facts, event facts.

The main goal of the rule engine is to empower users with the client side abilities to model/execute web scenarios/applications/mashups by means of business rules (See [1] and [2]). Particularly intelligent UI scenarios are in the main stream of interest.

For a better understanding of the context we consider the following situation: *We are looking for a job using the Monster Job Search Service. Once the job is obtained the location is shown on Google Maps.*

## 2 The JSON Rules language

The language was initially introduced in [1]. 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

```
RuleID: 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.

### 2.1 Ontology of events - DOM events

The JSON event expression is related to the Event interface specification in DOM Level 3 Events[1], therefore the properties of this expression have the same meaning as in the Event specification. At 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 1 (ECA Rule).*

```
{"id": "rule101",
  "appliesTo": ["http://mail.yahoo.com/"],
  "eventExpression": {"type": "click",
                      "target": "$X"
                     },
  "condition":[
              "$X:HTMLAnchorElement($hrefVal:href)",
              "new RegExp(/showMessage\?fid=Inbox/).test($hrefVal)"
              ],
  "actions":["append($X.textContent)"]
}
```

## 3 The Engine

There is an important difference between the actual rule engines and the JavaScript Rule Engine implementing the JSON Rules language for at least two reasons: events facts are ***not static facts*** that require usual operation such as: `delete`, `update` on the Working Memory but they are ***dynamic facts***. They are dynamically consumed based on the appearance time. Second the whole engine is a **live** system: it is **reactive** because reacts based on events and it is **proactive** for by itself produces events.

The project is hosted on Google Code platform[2].

### 3.1 How you can use the engine

The engine is programmed in JavaScript and can be used as any JavaScript framework. Basically, the lifetime of the rule engine is in the scope of the lifetime of the current DOM inside the browser. Simple steps to make it run are:

1. Load the engine in your page:

```
<script type="text/javascript"
  src="http://www.domain.com/jsonRulesEngine_Version.js">
</script>
```

[1] `http://www.w3.org/TR/DOM-Level-3-Events/`
[2] `http://jsonrules.googlecode.com`

2. Create an instance of the engine:

```
var jsonRulesEngine=new org.jsonrules.JSONRulesMainSystem();
```

3. Run the engine by calling `run()` with the URI of location of the repository as input parameter:

```
jsonRulesEngine.run("http://www.domain.com/rulesRepo.txt");
```

When the engine and the rulesets are available, the main things that happen are:

- When an event is raised, the `EventManager` catches that event. Then the `EventManager` checks the `ActionProcessor` state.
- If the *ActionProcessor* is running, then the `EventManager` stores the event in the queue of events that the `InferenceEngine` must later on process.
- However if the `ActionProcessor` is idle then the `EventManager` sends a message to the `InferenceEngine` containing the queue of events that must be processed. The `InferenceEngine` responds back to the `EventManager`, and informs it that it has received/consumed the queue such that the `EventManager` can reset its own queue.
- Events are processed one by one. For each event rules triggered by that event will be matched against the `WorkingMemory`. The action of each executable rule is added to the list of executable actions (to be processed by the `ActionProcessor`) according with possible priority of rules.
- The list of executable actions it is send to the `ActionProcessor`, to execute them. Any JavaScript functions can be called in the rule actions' part.

## 4 Conclusions

This paper describes shortly the general ideas behind an ECA rule-based and forward chaining engine for browsers.

## References


1. Adrian Giurca and Emilian Pascalau. JSON Rules. In *Proceedings of the Proceedings of 4th Knowledge Engineering and Software Engineering, KESE 2008*, volume 425, pages 7–18. CEUR Workshop Proceedings, 2008.
2. Emilian Pascalau and Adrian Giurca. A Rule-Based Approach of Creating and Executing Mashups. In *Proceedings of the 9th IFIP Conference on e-Business, e-Services, and e-Society (I3E 2009)*, LNCS. Springer, 2009. forthcoming.