

Built-ins for JSON Rules

Emilian Pascalau



JSON Rules - Goals

- Initially introduced in 2008. Adrian Giurca and Emilian Pascalau. [JSON Rules](#). In G. J. Nalepa and J. Baumeister (Eds.) Proceedings of *4th Knowledge Engineering and Software Engineering*, [KESE 2008](#), collocated with [KI 2008](#), September 23, 2008, Kaiserlautern, Germany, [CEUR vol 425](#)
- Move the reasoning process to the client-side
- Offer support for intelligent user interfaces
- Handle business workflows
- allow rule-based reasoning with semantic data inside HTML pages (reasoning with RDFa)
- create intelligent mashups – rule-based mashups
- enable users to collaborate and share information on the WWW (rule sharing and interchange)

JSON Rules - Requirements

- Rules run in the browser
- Event-Condition-Action rules
- Rules defined on top of DOM structure
- DOM Events + user defined events
- Conditions address the content of the pages
- Actions defined by users (any JavaScript function calls)

JSON Rules – properties of a rule

- id
- priority
- appliesTo
- eventExpression
- conditions:
 - Description
 - JavaScriptBooleanCondition
 - XPathExpression
- actions

Rule Example

Validate form using JSON Rules

```
{ "id": "rule101",  
  "appliesTo": [ 'http://www.example.com' ],  
  "eventExpression": { "type": "click",  
                      "target": "$X" },  
  "conditions": [  
    "$X:button(textContent==`Submit`)",  
    "$form:form($children:childNodes)"  
    "$Y:input(type==`text`, $Z:textContent)"  
    "org.w3c.rif.pred.number-greater-than($Z, 1000)",  
    "org.w3c.rif.pred.list-contains($children, $X)",  
    "org.w3c.rif.pred.list-contains($children, $Y)"  
  ],  
  "actions": [ "submitForm()" ]  
}
```

Built-ins

- Means constructed as a non-detachable part of a larger structure
- In case of rules systems encapsulate commonly used functionality, provided as predicates or functions
- Maintain the clean declarative design of rules

RIF-DTB (Rule Interchange Format DataTypes and Built-ins)

- RIF DataTypes are imported either from XML Schema Datatypes or from W3C RDF Plain Literals recommendation
- Predicates and Functions have been ported from W3C XQuery and XPath functions or from W3C RDF Plain Literal recommendation.

Artifacts used to define RIF built-in predicates and functions

- name of the built-in
- external schema of the built-in (signature of the built-in)
- for a RIF built-in function - how to map arguments into a result (in RIF terms the mapping of $I_{external}(\sigma)$ in the formal semantics of RIF-FLD and RIF-BLD)
- For a RIF built-in predicate – how it gives the truth value when arguments are substituted with values from the domain – this corresponds to the mapping of $I_{truth} \circ I_{external}(\sigma)$ in the formal semantics of RIF-FLD and RIF-BLD
- The domains of the built-ins arguments

Example – pred:numeric-greater-than

- **name** pred:numeric-greater-than
- **schema** $(?arg_1?arg_2; pred : numeric - greater - than(?arg_1?arg_2))$
- **domains** xs:integer, xs:double, xs:float, or xs:decimal for both arguments
- **mapping** When both a_1 and a_2 belong to their domains

$$I_{truth} \circ I_{external} (?arg_1?arg_2; pred : numeric - greater - than(?arg_1?arg_2))(a_1a_2) = t$$

if and only if $op : numeric - greater - than(a_1a_2)$ returns true as defined in W3C XQuery and XPath recommendation, false otherwise.

RIF-DTB states that “if an argument value is outside of its domain, the truth value of function/predicate is left unspecified”.

JavaScript context

- Closure and context based
- Closure: *“an expression (typically a function) that can have free variables together with an environment that binds those variables (that closes the expression)”*
- On top of the concepts of closure and context the namespace concept is defined.
- Weakly typed: Object, Boolean, Date, Number, String, Array

JavaScript schema convention

- **DataTypes:**
 - xs:duration(<http://www.w3.org/2001/XMLSchema#duration>)
 - org.w3c.xs.Duration
- **Built-ins:**
 - pred:numeric-greater-than
 - org.w3c.rif.pred.numeric-greater-than
 - func:count
 - org.w3c.rif.func.count

Mapping of RIF built-in Datatypes to JavaScript Datatypes

RIF datatype	JavaScript datatype	RIF datatype	JavaScript datatype	RIF datatype	JavaScript datatype
xs:string	String	xs:integer	Number, cust. type	xs:gYearMonth	Date, cust. type
xs:normalizedString	String, cust. type	xs:nonPositiveInteger	Number, cust. type	xs:gMonthDay	Date, cust. type
xs:token	String, cust. type	xs:negativeInteger	Number, cust. type	xs:gYear	Date, cust. type
xs:language	String, cust. type	xs:long	Number, cust. type	xs:gDay	Date, cust. type
xs:Name	String, cust. type	xs:int	Number, cust. type	xs:gMonth	Date, cust. type
Xs:NCName	String, cust. type	xs:short	Number, cust. type	xs:NMTOKENS	String+Array, cust. type
xs:ID	String, cust. type	xs:byte	Number, cust. type	xs:IDREFS	String+Array, cust. type
xs:IDREF	String, cust. type	xs:nonNegativeInteger	Number, cust. type	xs:ENTITIES	String+Array, cust. type
xs:NMTOKEN	String, cust. type	xs:unsignedLong	Number, cust. type	xs:QNAME	String, cust. type
xs:ENTITY	String, cust. type	xs:unsignedInt	Number, cust. type	xs:anyType	E4X XML object
xs:NOTATION	String, cust. type	xs:unsignedShort	Number, cust. type	rdf:PlainLiteral	String
xs:anyURI	String, cust. type	xs:unsignedByte	Number, cust. type	rdf:XMLLiteral	E4X XML Object
xs:hexBinary	Array, cust. type	xs:decimal	Number, cust. type		
xs:base64Binary	Array, cust. type	xs:boolean	Boolean		
xs:float	Number, cust. type	xs:dateTime	Date		
xs:double	Number, cust. type	xs:date	Date, cust. type		
xs:duration	String+Date, cust. type	xs:time	Date, cust. Type		

Duration type

- `org.w3c.xml.Duration`
- In general follows the `javax.xml.datatype.Duration`
- List of methods:
 - `add(org.w3c.xml.Duration rhs); addTo(Date date);`
 - `compare(org.w3c.xml.Duration duration);`
 - `getDays(); getHours(); getMinutes();`
 - `getMonths(); getSeconds(); getSign();`
 - `getYears();`
 - `isLongerThan(org.w3c.xml.Duration duration);`
 - `isShorterThan(org.w3c.xml.Duration duration);`
 - `negate();`

Example – pred:numeric-greater-than

```
org=new function(){};
org.w3c=new function(){};
org.w3c.rif=new function(){};
org.w3c.rif.pred=new function(){};
org.w3c.rif.pred.numericGreaterThan=function(arg1,arg2)
{
  try{
    if (isNaN(arg1) || isNaN(arg2))
      return false;
    if (arg1>arg2)
      return true;
    else
      return false;
  }catch(e){
    //log error
    //console.log(e); i.e. if Firebug is used
  }
  return false;
}
```

JSON Rules built-ins for mashups

- `org.jsonrules.builtins.mashups.load($uri,$where)`
- `org.jsonrules.builtins.mashups.retrieveAJAXResponse($from)`