Proposal: Implementing SWRL Built-Ins and Typed Tokenizing for OO jDREW in Java

Semantic Web Techniques 2012

Olav Hermansen, Ikhtear Md Sharif Bhuyan, Nicolas Neu
Introduction

OO jDREW (Object oriented deductive reasoning engine for the web) [1] is an open source implementation of the RuleML Web rule language, built upon jDREW. It supports a Bottom-Up as well as a Top-Down approach and is able to handle POSL and RuleML input. In order to enable community driven social coding OO jDREW is hosted by Github, where everyone has the possibility to fork their own version of the software [2].

Currently, there are two main shortcomings of the software which shall be addressed in the context of this project:

1. Complement the current Built-Ins to establish rudimentary date and time features. Depending on the complexity of the task, other Built-Ins, for example proposed by W3C RIF, shall be included as well.
2. Arguments for Built-Ins require a type definition to work properly. It should be possible for the system to automatically infer the correct type of the data.

For further details, please refer to chapter Objectives.

Objectives

Expanding the Built-Ins

To allow complex queries, OO jDREW uses so called Built-Ins, based on the SWRL Built-Ins. This way, the user can define predicates which can then be used in rules [5]. Built-Ins facilitate comparisons like equal or greater-than, mathematical operations like floor, ceil or multiply or string operations like string-length or string-uppercase. As on can learn from the listing of all Built-Ins, which can be found at [3], there are currently a lot of ideas for Built-Ins waiting for implementation. This project focuses on the implementation of the most important Built-Ins used for calculations with dates, time and duration, missing math and String Built-Ins. It was decided to implement various Built-Ins concerned with different areas of operation, namely math, boolean algebra, string and date functions. The main reason for this choice is to give us insight into more than one area. While the Built-Ins of the different areas aren’t really coherent, there was a focus on choosing related Built-Ins inside one area.

The following Built-Ins shall be implemented during this project:
8.2 Math Built-Ins

| unaryPlus | Satisfied iff the first argument is equal to the second argument with its sign unchanged. |
| unaryMinus | Satisfied iff the first argument is equal to the second argument with its sign reversed. |

**Rationale:** Extra functions for mathematical calculations. The `unaryMinus` function reverses the sign in front of a value which is needed for some mathematical operations. The `unaryPlus` operation leaves the sign unchanged which semantically results in no operation at all. The latter function is therefore just added for completeness sake.

**Priority:** low

8.3. Built-Ins for Boolean Values

| booleanNot | Satisfied iff the first argument is true and the second argument is false, or vice versa. |

**Rationale:** An important operator used in boolean algebra. Use can be avoided but implementing this makes certain Use Cases easier.

**Priority:** high

8.4 Built-Ins for Strings

| replace | Satisfied iff the first argument is equal to the value of the second argument with every substring matched by the regular expression the third argument replaced by the replacement string the fourth argument. |
| substringBefore | Satisfied iff the first argument is the characters of the second argument that precede the characters of the third argument. |
| substringAfter | Satisfied iff the first argument is the characters of the second argument that follow the characters of the third argument. |

**Rationale:** String operations are a powerful tool for character manipulation. Especially when regular expressions are used, complex issues can be described in a concise manner.

**Priority:** The implementation could prove difficult, as regular expressions are a very sophisticated matter. For the latter reason, we decided to rate the priority as low.

8.5 Built-Ins for Date, Time and Duration

| dateTime | Satisfied if the first argument is the xsd:dateTime representation consisting of the year the second argument, month the third |


<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>Satisfied iff the first argument is the xsd:time representation consisting</td>
</tr>
<tr>
<td></td>
<td>of the hours the second argument, minutes the third argument, seconds the</td>
</tr>
<tr>
<td></td>
<td>fourth argument, and timezone the fifth argument.</td>
</tr>
<tr>
<td>date</td>
<td>Satisfied iff the first argument is the xsd:date representation consisting</td>
</tr>
<tr>
<td></td>
<td>of the year the second argument, month the third argument, day the fourth</td>
</tr>
<tr>
<td></td>
<td>argument, and timezone the fifth argument.</td>
</tr>
</tbody>
</table>

**Rationale:** Currently, there are no date or time related Built-Ins present in OOjDREW. The *date*, *time* and *dateTime* Built-Ins can serve as a basis for further enhancements in this area. Also, those Built-Ins are kind of a minimum requirement when working with Strings representing a date or time. It is now possible to, for example, define a point in time when programming which is needed in many practical applications.

**Priority:** High

Time permitting, additional Built-Ins will be considered. This depends on the complexity of the aforementioned Built-Ins.
Automatic type inference

Currently, the datatypes of arguments for a Built-In have to be explicitly declared. For example, the constant 18 is written as either 18:Integer or as 18^^Integer, depending of the used language and version. This is cumbersome for the author who has to write redundant and repetitive code. It also hinders the easy readability of the code. This is why an automatic type detection system shall be implemented for basic datatypes so that the type information doesn’t have to be explicitly written. To revisit the earlier used example, the constant 18 should be treated exactly the same as 18:Integer or 18^^Integer would be treated by the parser.

This automatic type detection shall be implemented for the following datatypes:

- Integer
- Float
- String

Testing

In order to guarantee a bugfree end result the implementation shall be tested in a manual as well as automatic manner. For each new feature, several test-cases will be submitted. While this assures a high quality of the delivered code, it can also serve as documentation to ease understanding of the feature and its implementation later on.

Methodology

As Github is used as a hosting platform for the project, tools for project management provided by Github will be used. The first step will be to file a number of issues on Github which will describe the requirement for a single feature. Each team member then forks the original repository. When the implementation of a requirement is finished, a pull request referencing the issue is filed, so the code can be merged back into the original repository. Tests will be submitted the same way.

Extensive documentation on how to properly implement new Built-Ins can be found in the OO jDREW Built-In Creation Guide [4].
References


