Preconditions and Results

Preconditions on a service are a list of SWRL atoms which must evaluate to true in order for the service to be invoked. Results make assertions that can be assumed true if the result condition evaluates to true.

Table of Contents

Annotations

There are three main annotations involved in preconditions and results. *@OwlsLocal*, *@OwlsResult* and *@OwlsPreconditions*.

@OwlsLocal is used to declare local variables. These are typed, and are scoped over the whole of the preconditions and results for a process. One can either use a single *@OwlsLocal* on a method, or wrap it in *@OwlsLocals* and having an array of *@OwlsLocal* annotations as the value. Each *@OwlsLocal* needs a unique name (it also cannot clash with the name of a ResultVar) and the URI of an OWL Class to use as the type. Example:

```
@OwlsLocals( { @OwlsLocal(name = "valX", type = "&xsd;#int"),
   @OwlsLocal(name = "valY", type = "&xsd;#int") })
```

@OwlsPrecondition is used to define the preconditions on a service. This takes a single string as its value, which contains a list of SWRL atoms. This is translated into an XML representation of SWRL in the final output which the OWL-S API can use. The syntax for this is detailed below. Example:

```
@OwlsPreconditions("swrlb:greaterThanOrEqual(#valX,
0),swrlb:greaterThanOrEqual(#valY, 0),"
   + " num:hasValue(#x,#valX), num:hasValue(#y,#valY).")
```

@OwlsResult can be used on its own on a method to define a single result, or, as with Locals, it can be wrapped in the value of *@OwlsResults* if multiple results are desired. In each *@OwlsResult* there are two SWRL strings: the effects (assertions of truth) and conditions (which determine whether the effects can be assumed true). These use the same syntax as the preconditions, the only difference being that effects do not have a full stop at the end, but conditions do. It is also possible to define synthetic outputs which do not directly relate to the output from the grounding of the service: you can treat these as normal parameters when defining effects to make assertions about them.

Additionally, *@OwlsResult* contains an array of *@OwlsResultVars*. These are essentially the same as Local variables, only they are scoped to a particular result. As with Local variables, they must have a unique name, and are typed. Example:

```
@OwlsResult(effect = "swrlb:sameIndividual(#rv,#testSynthetic1),
swrlb:isa(#rv,&num;#EvenNumber)", condition =
"swrlb:mod(0,#sum,2),swrlb:add(#sum,#valX,#valY).", vars = { @OwlsResultVar(name =
"sum", type = "&xsd;#int") }, outputs="testSynthetic1")
```

SWRL

A small logic style language is included in OWLSBuilder to allow one to define SWRL in a simple manner.

Syntax

The language for defining preconditions and results uses a Prolog-like syntax with a full stop at the end of the lists condition atoms (the full stop is not present for effects in results) e.g.

```
swrlb:add(%AnswerVariable, 0.9, 1.0), swrlb:equal(%AnswerVariable, 1.9).
```

An ANTLR Grammar for this language is included in the OWLSBuilder source code.

The body is made from a list of comma-separated predicates (atoms). Predicates can be prefixed by a namespace followed by a colon (as in XML), which can be used to allow for properties from an external ontology to be used as atoms. These replacements are defined in the import annotation, and cannot contain a full stop ('.').

Atoms

Atom names are words beginning with a letter and optionally containing letters, numbers and underscore after this initial letter. Atoms whose name starts with a capital letter must be escaped by placing a single quote in front of the name. e.g.
Valid:

```
books:'OnShelf()
onshelf()
```

Invalid:

```
:'OnShelf()
books:OnShelf()
'onshelf()
```

Atoms are always followed by an opening and closing bracket.

Parameters

Atoms can take parameters as a comma separated list between the brackets. There are three main types of parameter which can be passed to an atom: numbers, variables and individuals.

- Numbers are signed and floating point, and can be optionally written in scientific notation.
  e.g.

  ```
  swrlb:add(%NewTotal, 1, %OldTotal)
  swrlb:add(%NewTotal, 1.0, %OldTotal)
  swrlb:add(%NewTotal, -0.5e-4, 0.21e3)
  ```

- Variables are created on their first use. They begin with an upper case letter or underscore, and can contain any alphanumerical or underscore character after that. By default variables are of Individual type. To make a variable store a Data type, its name must be prefixed by a '%'.
  e.g.

  ```
  onshelf(Book) Book is an Individual variable
  onshelf(%ISBN) ISBN is a Data variable
  ```

- Individuals are either complete URIs, or an ID local to the service base prefixed by a '#'
  e.g.

  ```
  onshelf(#TheHobbit)
  ```

  Individuals may also be prefixed by an XML-style entity, as defined in the external ontology imports. This will be replaced by the URI of the ontology to which it refers.
  e.g.

  ```
  onshelf(&books;#TheHobbit)
  ```

  Both Local variables and ResultVars are treated individuals, and should be referenced as such when writing code.

SWRL Built-ins

SWRL contains a number of predefined built-in atoms. These are accessed in OWLSBuilder by using the 'swrlb' namespace. The following builtins are supported:

- add
- subtract
- multiply
- divide
- mod
- pow
- equal
- notEqual
- lessThan
- lessThanOrEqual
- greaterThan
- greaterThanOrEqual
- sameIndividual
- differentIndividuals
- isa (ClassAtom, i.e. used for asserting and determining types)
  Further information can be found in the OWL-S API Javadocs.