TRANSLATOR Developer Guide

Note: this guide is incomplete

Java Web Start

TRANSLATOR has been made available as a Java Web Start application in order to promote accessibility (works with practically all browsers, etc). A few things had to be done in order to accomplish this.

On the code level, Translator.java was modified to use `ClassLoader.getResource()` to look up images used for the GUI.

Since TRANSLATOR requires access to the local machine's internet connection (to access the APE webservice), the code has to be digitally signed
(see http://java.sun.com/j2se/1.5.0/docs/guide/javaws/developersguide/faq.html#201).
After creating the jar with, e.g.,

```
jar vcf translator.jar images org
```

the jar file can be signed using Java's jarsigner tool as follows:

```
jarsigner -keystore translatorKeystore translator.jar DavidHirtle
```

Note that a keystore is required; in the example above, it's named `translatorKeystore`. You can generate your own using Java's keytool, but for convenience I uploaded `translatorKeystore` onto the TRANSLATOR site:
http://www.ruleml.org/translator/translatorKeystore

Note that the alias is DavidHirtle and the password is pa55w0rd.
See http://java.sun.com/docs/books/tutorial/deployment/jar/signindex.html for more info on signing jar files.

After the jar file is signed, it can be uploaded to `http://www.ruleml.org/translator`, but be sure a backup exists before overwriting existing files. Also, it's a good idea to keep a version history, documenting each change.

Finally, a jnlp file is required. The following (also online at http://www.ruleml.org/translator/translator.jnlp) should suffice unless significant changes (e.g., adding a new library) are made:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+"
      codebase="http://www.ruleml.org/translator"
      href="translator.jnlp">
   <information>
      <title>TRANSLATOR</title>
      <vendor>David Hirtle</vendor>
      <homepage href="http://www.ruleml.org/translator"/>
      <description>TRANSLATOR: TRANSlator from LAnguage TO Rules</description>
      <description kind="short">TRANSLATOR</description>
      <offline-allowed/>
   </information>
   <security>
      <all-permissions/>
   </security>
   <resources>
      <j2se version="1.4+"/>
      <jar href="translator.jar" main="true" download="eager"/>
      <jar href="xom.jar" download="eager" />
      <jar href="BareBonesBrowserLaunch.jar" download="eager" />
   </resources>
   <application-desc main-class="org.ruleml.translator.gui.Translator"/>
</jnlp>
```

Dependencies
TRANSLATOR's chief dependency is on the APE webservice, specifically the (non-prettyprint) DRS representation. All other notations (e.g., FOL, paraphrase) are only displayed, not processed, so them being altered will not impact TRANSLATOR's overall functionality. Even subtle changes to the DRS notation, on the other hand, will likely result in a parsing error from the DRS parser used by TRANSLATOR.

The flavour of DRSs used in ACE are documented in a technical report "Extended Discourse Representation Structures in Attempto Controlled English". The Attempto team has so far kept it updated on the publications section of the website (http://www.ifi.unizh.ch/attempto/publications). As of the time of this writing, the current version is:
http://www.ifi.unizh.ch/attempto/publications/papers/drs_report_06.pdf

Note that changes to the DRS representation should only require modifying the DRS parser, found in the `org.ruleml.translator.parser` package (as opposed to `org.ruleml.translator.gui`). In particular, the JavaCC grammar file `DRSParser.jj` is the only one that needs modifying; the actual .java files are automatically generated using JavaCC (using the command `javacc DRSParser.jj`).

Keeping up-to-date
The news section of the Attempto website (http://www.ifi.unizh.ch/attempto/news) should include mention of any changes that may affect TRANSLATOR. Note that an RSS feed is available.

Working example
On April 21, 2006, the DRS representation of nouns was simplified. Previously, structure and quantity were separate predicates, e.g.,

```
structure(B, atomic)-1
quantity(B, cardinality, count_unit, C, eq, 1)-1
object(B, dog, object)-1
```

but the new representation includes everything in the object predicate:

```
object(B, atomic, dog, object, cardinality, count_unit, eq, 1)-1
```

This kind of change is quite straightforward. Doing a search for "structure" and "quantity" is a good way to see how extensive the changes will be. The strings "structure" and "quantity" were declared as tokens, so those lines could be deleted. Also, there is a grammar rule for each kind of predicate; the StructurePred() and QuantityPred() rules are also no longer necessary (should be removed from the Predicate() rule), but parts could be reused for the now extended ObjectPred() rule.

Another change was the implication and disjunction operators (`=>` and `v`) were changed from being infix to prefix, e.g.,

drs([], [**drs(...)=>drs(...)**])

changed to

drs([], [**=>(drs(...), drs(...))**])

This more structural change turns out to be less work; only a handful of lines of the DRSParser.jj file need to be udpated. Specifically, the second and third choices of the ComplexCondition grammar rule (i.e., the <Or> and <Implies> ones):

```
Element ComplexCondition() :
{ Element neg, element, exists, exists2, implies, or; }
{
       neg=NegDRS()
       { return neg; }
|
       LOOKAHEAD( DRS() <OR> ) // syntactic lookahead
       exists=DRS() <OR> exists2=DRS()
       {
              or = new Element("Or", RULEML_NAMESPACE);
              or.appendChild(exists);
              or.appendChild(exists2);
              return or;
       }
|
       element=RDRS() <IMPLIES> exists=DRS()
       {
              if (element.getLocalName().equals("Implies"))
              { element.appendChild(exists); }
              else // element is "forall"
              {
                     implies = element.getChildElements("Implies", RULEML_NAMESPACE).get(0);
                     implies.appendChild(exists);
              }
              return element;
       }
}
```

(Recall that a choice between alternatives in BNF notation uses the pipe character "`|`".)

The line

```
LOOKAHEAD( DRS() <OR> ) // syntactic lookahead
```

specifies the lookahead to be used by JavaCC.

so the end result

```
Element ComplexCondition() :
{ Element neg, element, exists, exists2, implies, or; }
{
       neg=NegDRS()
       { return neg; }
|
|
       <OR> <LPAREN> exists=DRS() <COMMA> exists2=DRS() <RPAREN>
       {
               or = new Element("Or", RULEML_NAMESPACE);
               or.appendChild(exists);
               or.appendChild(exists2);
               return or;
       }
|
       <IMPLIES> <LPAREN> element=RDRS() <COMMA> exists=DRS() <RPAREN>
       {
               if (element.getLocalName().equals("Implies"))
               { element.appendChild(exists); }
               else // element is "forall"
               {
                       implies = element.getChildElements("Implies", RULEML_NAMESPACE).get(0);
                       implies.appendChild(exists);
               }
               return element;
       }
}
```