The Economics of Prolog Declarative Programming in Java

Felician ALECU, PhD, Lecturer
Department of Economic Informatics and Cybernetics
Bucharest University of Economic Studies, Romania
E-mail: alecu[at]ase[dot]ro; Web Page: http://alecu.ase.ro

Abstract: This paper explores the possibilities of combining the PROLOG simplicity with the power of JAVA in order to create better user friendly software applications for the market.

Keywords: Prolog, Java, JPL, declarative programming, economics.

JEL: O33, O38.

Introduction

As exemplified in a previous article hosted right inside this prestigious journal, the PROLOG programming language can be successfully used to solve complex computational problems by specifying the goal, not the algorithmic steps to be followed.

This is why the PROLOG version of a classical program is much simple, readable and reliable.
Unfortunately, we can foresee some problems in using the PROLOG for real life software implementations, like the following:

- java is the most popular programming language of the moment, so most of the programmers will choose java, not prolog, to solve an issue even if the declarative programming approach would be better;
- for the existing applications (already written in java, for example), it is almost impossible to be rewritten just because a small part of the code is more suitable to be expressed in PROLOG;
- applications of today have complex user interfaces that just cannot be realized by simply using the PROLOG.

Mixing Prolog with Java

So, is there any way to combine the java capabilities and maturity with the PROLOG features in order to obtain better software products for the mass market?

Fortunately, the answer is YES, in both ways:

- java code could be included in PROLOG programs

and

- prolog structures can be used directly in java

Let’s imagine we are still charmed by the simplicity of the PROLOG isArrayElement program (exemplified in a previous article) having the following source code:
isArrayElement.pl file

isArrayElement(Element,[Element|_]).

isArrayElement(Element,[_|Array])

:- isArrayElement(Element,Array).

By using this two lines piece of code, it is quite simple to check if an element is part of a list, by issuing statements like:

?- isArrayElement(abc,[aba,abc,abd,abe])
true.

?- isArrayElement(xyz,[aba,abc,abd,abe])
false.

PROLOG is also able to answer to some complex questions, like displaying all the elements (or just the first one) of a given array:

?- isArrayElement(X,[aba,abc,abd,abe])
X = aba ;
X = abc ;
X = abd ;
X = abe ;
true.

So, a good idea will be integrate the PROLOG code (very simple but with no graphical user interaction) into a complex java application exposing a well-designed graphical user experience. Since I personally
use SWI PROLOG (http://www.swi-prolog.org/), for the java integration the natural choice would be the JPL that is a bidirectional Prolog/Java interface (more details under http://www.swi-prolog.org/FAQ/Java.html).

For this reason, I created a new class file, called isArrayElement, presented below:

```java
import java.util.Hashtable;
import jpl.*;
import jpl.Query;

public class isArrayElement
{
    public static void main( String argv[] )
    {
        Query query =
            new Query("consult('isArrayElement.pl')");

        if (!query.hasSolution()){
            System.out.
                println("consult('isArrayElement.pl') failed");
            return;
        }
    }
```
String t2 = "isArrayElement(abc,[aba,abc,abd,abe])";
Query q2 = new Query(t2);
System.out.println(t2 + " is " +
    (q2.hasSolution() ? "TRUE" : "FALSE"));

String t3 = "isArrayElement(xyz,[aba,abc,abd,abe])";
Query q3 = new Query(t3);
System.out.println(t3 + " is " +
    (q3.hasSolution() ? "TRUE" : "FALSE"));

String t4 = "isArrayElement(X,[aba,abc,abd,abe])";
Query q4 = new Query(t4);
System.out.println("one solution of " + t4 +
    ": X = " + q4.oneSolution().get("X"));

java.util.Hashtable[] ss4 = q4.allSolutions();
System.out.println("all solutions of " + t4);
for (int i=0 ; i<ss4.length ; i++) {
    System.out.println("X = " + ss4[i].get("X"));
}

System.out.println("more solution of " + t4);
while (q4.hasMoreSolutions()) {
    java.util.Hashtable s4 = q4.nextSolution();
    System.out.println("X = " + s4.get("X"));
}

To compile the source code it is enough to launch
javac isArrayElement.java

while to run the code the java should be used, like
java isArrayElement
The results seem to be quite impressive, since the PROLOG code is successfully executed inside the JAVA VM, as presented below.

\[
isArrayElement(abc, [aba, abc, abd, abe]) \text{ is TRUE}
\]
\[
isArrayElement(xyz, [aba, abc, abd, abe]) \text{ is FALSE}
\]

one solution of
\[
isArrayElement(X, [aba, abc, abd, abe]): \text{ } X = \text{aba}
\]

all solutions of isArrayElement(X, [aba, abc, abd, abe])
\[
X = \text{aba}
\]
\[
X = \text{abc}
\]
\[
X = \text{abd}
\]
\[
X = \text{abe}
\]

more solution of isArrayElement(X, [aba, abc, abd, abe])
\[
X = \text{aba}
\]
\[
X = \text{abc}
\]
\[
X = \text{abd}
\]
\[
X = \text{abe}
\]
Conclusions

As validated by this article, the PROLOG capabilities can be successfully combined with the JAVA complexity for higher source code quality that may lead to better software applications for the market.

References

