Reasoning about Source Code in XML–Representation

Marbod Hopfner[1], Dietmar Seipel[2],
Jürgen Wolff von Gudenberg[2], Gregor Fischer[2]

[1] University of Tübingen, Wilhelm–Schickard Institute for Computer Science
Sand 13, D – 72076 Tübingen, Germany
hopfner@informatik.uni-tuebingen.de

[2] University of Würzburg, Institute for Computer Science
Am Hubland, D – 97074 Würzburg, Germany
{seipel,wolff,fischer}@informatik.uni-wuerzburg.de

Overview of Visur/Rar

We have implemented a Prolog–tool Visur/Rar for reasoning about various types of source code, such as Prolog–rules or Java–programs. Rar provides *retrieval*, *update* and *inference* operations for a deductive database storing XML–representations of the investigated code, rules for analysing Prolog–code based on suitable *dependency graphs* and rules for recovering the design of Java–software; it uses a query language FNQuery that is based on path expressions. Visur/Rar can be applied for improving the design of rule–based systems, for computing certain *software metrics*, and for supporting *refactoring techniques*. The obtained results are visualised using graphs or tables in Visur.

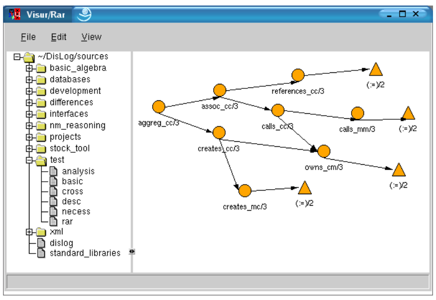

Figure 1. Graphical User Interface of Visur

Visur/Rar is part of the toolbox DisLog [7], which is also developed under XPCE/Swi–Prolog; the functionality of DisLog ranges from reasoning in disjunctive deductive databases to applications such as the management and visualisation of stock information.

Representing Source Code in XML

The following Java–method implements the well–known sorting algorithm *merge sort* on a "globally defined" array. The method for merging two sorted sub–arrays is not shown here.

```
public void sort (int l, int r) {
   if (l<r) {
      int m = (l+r)/2;
      sort(l,m);
      sort(m+1,r);
      merge(l,m,r); } }
```

In [3] an XML–language JAML is introduced for representing the parse trees of Java–code – we could also use other XML–representations such as JavaML and srcML. In the following we show an abbreviated JAML–representation for a small portion of the Java–code, namely the method invocation `sort(l,m)`:

```
<method-invocation
   name="sort" qualifier=""
   argument-0="l" argument-1="m"
   signature= "sort(int, int)"
   return-type-ref="void">
   <le literal="sort" type-ref=""/>
   <symbol kind="left-parenthesis"/>
   <le literal="l" type-ref="int"/>
   <symbol kind="comma"/>
   <le literal="m" type-ref="int"/>
   <symbol kind="right-parenthesis"/>
</method-invocation>
```

In Visur/Rar complex structured objects – i.e., Java–programs and Prolog–programs – are conceptionally represented in XML–notation, which itself is physically stored in a PROLOG–term representation called *field notation*. We *query* and *manipulate* this field notation using a library FNQUERY developed in [7]. FNQUERY also contains additional, more *advanced methods*, such as the selection/deletion of all elements/attributes of a certain pattern, the transformation of sub–components according to substitution rules in the style of XSLT, and the manipulation of path or tree expressions.

Reasoning about Source Code

The following rules define some basic relations between methods and classes using path expressions in FNQUERY. For example, `references_cc/3` describes that the class `C1` has an attribute of the type `C2`. This holds, if at arbitrary depth in the JAML–representation `Code` of the source code there exists a `class-definition`–element `U` which itself contains a `field-declaration` `V` (at arbitrary depth) with an `identifier`–subelement. In that case `C1` is taken to be the `name`–attribute of `U` and `C2` is the `type`–attribute of `V`.

```
references_cc(Code,C1,C2) :-
   U := Code^_^class-definition,
   V := U^_^field-declaration,
   C1 := U@name,
   C2 := V@type,
   _ := V^identifier.

calls_mm(Code,M1,M2) :-
   U := Code^_^method-declaration,
   V := U^_^method-invocation,
   M1 := U@signature,
   M2 := V@signature.

owns_cm(Code,C,M) :-
   U := Code^_^class-definition,
   V := U^_^method-declaration,
   C := U@name,
   M := V@signature.
```

For visualising the call structure of rule–based systems we use the concept of *dependency graphs*, which is well–known from deductive databases [2]; the *call graph* of our example is shown in Figure 1.

The second layer of rules is based on the basic predicates. These rules allow for a more complex analysis of the JAVA–source code. They have been proposed in [6] for the pattern–based design recovery of JAVA–software. For example, *aggregations* and *associations* provide a higher–level view of the original *design* in contrast to the implementational view of the source code:

```
calls_cc(Code,C1,C2) :-
   calls_mm(Code,M1,M2),
   owns_cm(Code,C1,M1),
   owns_cm(Code,C2,M2).

assoc_cc(Code,C1,C2) :-
   calls_cc(Code,C1,C2),
   references_cc(Code,C1,C2).
```

Conclusions

The integration of XML–processing, visualisation, and reasoning in the logic programming environment XPCE–PROLOG has created a powerful and flexible tool. VISUR/RAR allows for analysing various types of complex structures including PROLOG–rules, JAVA–statements, and even ER–diagrams.

In the future, we will gradually extend VISUR/RAR with additional features. E.g., it is conceivable to integrate a possibility to move predicates from one location in a file to another location in another file (or the same file) using drag–and–drop operations; heuristic rules in RAR might be used for ensuring that the moves do not change the meaning of the program.

References

[1] *S. Abiteboul, P. Bunemann, D. Suciu:* Data on the Web – From Relations to Semi–Structured Data and XML, Morgan Kaufmann, 2000.

[2] *S. Ceri, G. Gottlob, L. Tanca:* Logic Programming and Databases, Springer, 1990.

[3] *G. Fischer, J. Wolff von Gudenberg:* JAML – An XML–Representation of JAVA–Source Code, Technical Report, University of Würzburg, 2003.

[4] *M. Fowler:* Refactoring – Improving the Design of Existing Code, Addison–Wesley, 1999.

[5] *M. Hopfner, D. Seipel:* Comprehending and Visualising Software based on XML–Representations and Call Graphs. Proc. 11th Intl. Workshop on Program Comprehension, IEEE Press, 2003.

[6] *J. Seemann, J. Wolff von Gudenberg:* Pattern–Based Design Recovery of JAVA Software, Proc. Intl. Symposium on the Foundations of Software Engineering 1998.

[7] *D. Seipel:* Processing XML–Documents in PROLOG, Proc. 17th Workshop on Logic Programming WLP 2002.