Declaratively Querying and Visualizing Knowledge Bases in XML

Dietmar Seipel[1], Joachim Baumeister[1], and Marbod Hopfner[2]

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

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

Abstract. The maintenance of large knowledge systems usually is a rather complex task. In this paper we will show that extensions or modifications of a knowledge base can be supported by appropriate visualizations techniques, e.g. by illustrating dependencies of the considered knowledge.
In particular, we introduce a declarative approach for querying and visualizing rule-based knowledge represented as XML documents; a knowledge engineer can extract and visually inspect parts of the knowledge base by ad-hoc declarations in a flexible manner.

Keywords. knowledge systems, rule bases, PROLOG, XML query / transformations, visualization

1 Introduction

The extension and maintenance of large rule-based systems is a complex task. For instance, the deletion of a redundant or an incorrect rule is often very difficult to perform. Thus, transitive dependencies in which the rule is involved are not obvious at first sight, but they can effect the behaviour of the entire knowledge system.

In this paper we introduce a declarative approach for flexibly generating a suitable visualization of rule-based knowledge of various types, such as diagnosis rules, abstraction rules, and indication rules. The visualization of knowledge can be used, e.g., in the following scenarios:

- *Restructuring knowledge:* If the expert wants to remove or modify an existing rule, then it is helpful to inspect all dependend knowledge objects (e.g., constrained findings and inferred solution objects) and rules, respectively. A visual view of the dependencies can simplify the inspection of the knowledge base.
- *Validating knowledge:* The visual inspection of knowledge can be also helpful during the validation of the knowledge systems' reasoning behavior. Then, the visualization of the derivation graph of a solution object defined by its deriving rules can assist the expert during a debugging session.

– *Examination of the knowledge design:* We visualize the rule base as a graph with knowledge objects (i.e., findings, solutions) represented as nodes and rules depicted by corresponding edges. The design of the knowledge base can be simply analyzed by viewing the graph structure. Results of this analysis are domain dependent. E.g., a sub-graph connected to the remaining graph structure only by one node is an indicator for a vulnerable knowledge design, since a part of the implemented knowledge depends on a single object.

Besides the examples given above there exist many other applications for the visualization of rule bases. However, for the implementation of a visualization tool we face the problem that we cannot specify a predefined set of meaningful visualizations for rule bases, since the visualization depends on the requirements of the current task. Therefore, in a real-world environment a flexible and adaptive visualization tool is required for a reasonable application of visualization techniques.

In this paper we introduce a declarative approach for flexibly defining visualizations of rule bases. This declarative approach allows for a fast ad-hoc definition of visualizations based on given requirements of the expert. We assume the knowledge base to be available in an XML format, and we provide a compact set of queries and transformations for generating reports about the knowledge base in the form of HTML and graphs in GXL format, a standard format for graph-based structures [8]. In principle, our approach is not restricted to the visualization of rule bases; it can be generalized to arbitrary XML documents. For example, in [7] we have described a tool VISUR/RAR for the visualization of procedural or declarative programs represented in XML structures, e.g., JAVA or PROLOG programs, and for reasoning about the structures of these programs.

The rest of the paper is organized as follows: In Section 2 we sketch a possible XML representation of knowledge bases and motivate the processing of such XML structures using RAR. The visualization system VISUR is briefly introduced in Section 3. In Section 4 the declarative specification of visualizations is described by motivating examples. In the conclusions we summarize the presented work and we outline some promising directions for future work.

2 Management of Knowledge Bases in XML

XML has been established as a frequently used standard for representing and interchanging data and knowledge. Consequently, various different markups have been proposed for formalizing explicit knowledge; OWL is a prominent example [14].

In [6] we have presented a system VISUR/RAR. It is part of the DISLOG developers toolkit DDK [12], which contains the library FNQUERY for handling these XML data. VISUR/RAR is a system for querying, transforming, and visualizing rule-based knowledge in XML notation. It is currently generalized to work on arbitrary XML structures. The component RAR (*reasoning about rules*) is applied for query and modification tasks. Results of this task are handled by VISUR, which can *visualize* the results given in the *Graph eXchange Language* GXL [8]; we added some additional attributes to the GXL notation for configuring the graph display.

In this section we introduce the XML schema that we are using for representing knowledge bases, and then we sketch the processing of knowledge bases using RAR.

2.1 Knowledge Bases in XML

In our running example we work with a particular XML schema for knowledge bases, which is used by the D3 system [10]. The shell-kit D3 has been applied successfully for building many diagnostic knowledge systems in the medical, the technical, and the biological domain.

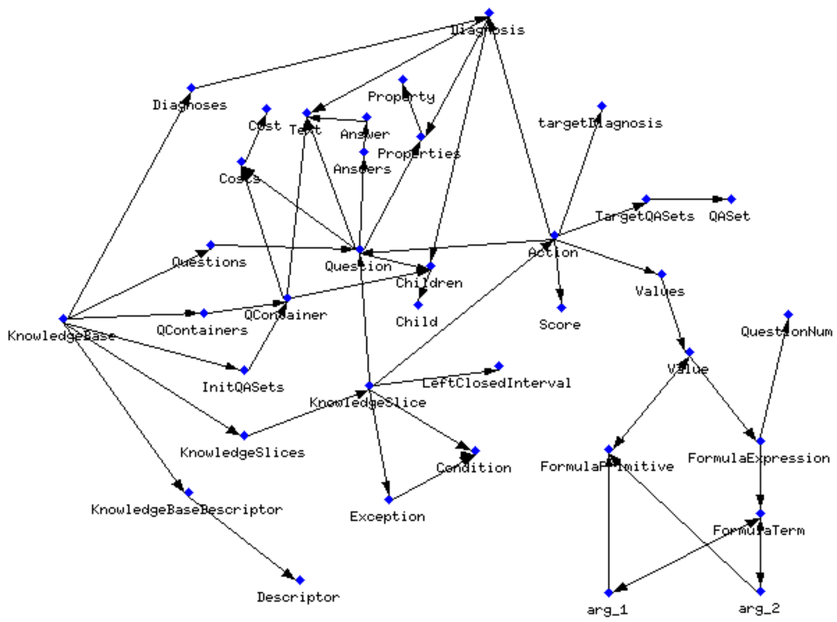

Fig. 1. Knowledge Schema Graph of D3

The schema graph of D3 knowledge bases is depicted in Figure 1. Given the root tag `KnowledgeBase`, the successor tags are as follows: `KnowledgeBase-Descriptor` (meta information about the given knowledge base), `InitQASets` (initial questions asked to the user of the system), `Questions` (questions to be asked to the user), `Diagnoses` (possible solutions derived by the system), and `Knowledge-Slices` (inferential knowledge connecting questions and diagnoses). All objects, e.g., diagnoses, questions, and rules, are identified by unique IDs in the XML representation of the knowledge base.

In Figure 2 a small excerpt of the SonoConsult knowledge base [9] is shown. SonoConsult is a fielded knowledge-based documentation and consultation system for medical sonography. Diagnoses and questions are described by a *text* containing the name of the knowledge object; for each question a list of the possible answers is given. The rules are *production rules*: if the given condition evaluates to true, then the production rule assigns a score category to the specified diagnosis.

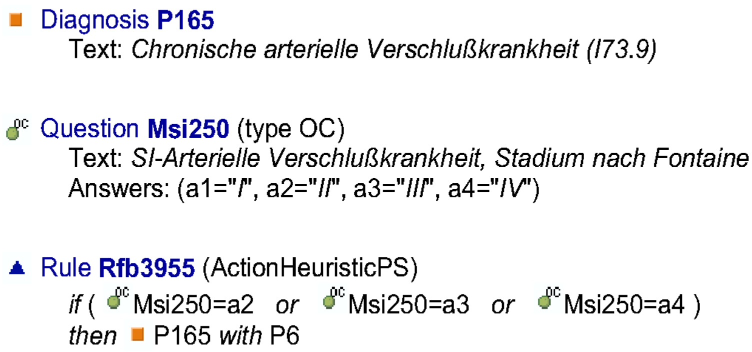

Fig. 2. Excerpt of the Knowledge Base SonoConsult

The knowledge base SonoConsult contains about 8.000 rules. The rule with the ID *Rfb3955* from above is represented as the following knowledge slice:

```
<KnowledgeSlice ID="Rfb3955" type="RuleComplex">
  <Action type="ActionHeuristicPS">
    <Score value="P6"/>
    <Diagnosis ID="P165"/>
  </Action>
  <Condition type="or">
    <Condition type="equal" ID="Msi250" value="Msi250a2"/>
    <Condition type="equal" ID="Msi250" value="Msi250a3"/>
    <Condition type="equal" ID="Msi250" value="Msi250a4"/>
  </Condition>
</KnowledgeSlice>
```

The condition of the rule *Rfb3955* evaluates to true if the question *Msi250* has one of the values *Msi250a2*, *Msi250a3*, or *Msi250a4*. In that case, the score *P6* is given to the diagnosis *P165*; due to the high score of *P6* the diagnosis will be considered as a possible solution. In general, the value range of the scores given to diagnoses is $SC = \{N7, \ldots, N1, P1, \ldots, P7\}$ with ascending categories $P1, \ldots, P7$ for confirming a diagnosis and categories $N1, \ldots, N7$ for disconfirming a diagnosis.

For many questions the value can be derived by another rule in the knowledge base, since the knowledge base can describe transitive paths for deriving a diagnosis. Then it is not necessary that the user enters this value. For a more detailed discussion of the applied knowledge representation we refer to [10].

2.2 Processing Knowledge Bases with FNQUERY

The library FNQUERY offers a variety of PROLOG predicates for processing XML documents and two different data structures, which we call field notation and database notation. In *field notation* a complex object can be represented as an *association list*

$$[\mathsf{a}_1 : \mathsf{v}_1, \ldots, \mathsf{a}_n : \mathsf{v}_n],$$

where a_i is an *attribute* and v_i is the associated *value*; this representation is well–known from the field of artificial intelligence. Using the field notation has got several advantages compared to ordinary PROLOG facts "$\mathsf{object}(\mathsf{v}_1, \ldots, \mathsf{v}_n)$" : The sequence of attribute/value–pairs is arbitrary. Values can be accessed by attributes rather than by argument positions. Null values can be omitted, and new values can be added at runtime. In the PROLOG library FNQUERY this formalism has been extended to the *field notation* for XML documents: an XML element

$$\langle \mathsf{Tag}\ \mathsf{a}_1 = "\mathsf{v}_1" \ldots \mathsf{a}_n = "\mathsf{v}_n" \rangle\ \mathsf{Contents}\ \langle/\mathsf{Tag}\rangle$$

with the tag "Tag" can be represented as a PROLOG term $\mathsf{Tag}:\mathsf{As}:\mathsf{C}$, where As is an association list for the attribute/value–pairs $\mathsf{a}_i = "\mathsf{v}_i"$, and C represents the contents, i.e., the subelements of the element. E.g., for the XML representation of the knowledge slice from Section 2.1 we obtain:

```
'KnowledgeSlice':['ID':'Rfb3955', type:'RuleComplex']:[
   'Action':[type:'ActionHeuristicPS']:[
      'Score':[value:'P6']:[],
      'Diagnosis':[ID:'P165']:[] ]
   'Condition':[type:or]:[
      'Condition':[
         type:equal, 'ID':'Msi250', value:'Msi250a2']:[],
      'Condition':[..., value:'Msi250a3']:[],
      'Condition':[..., value:'Msi250a4']:[] ] ]
```

Another way of representing XML in PROLOG is the *database notation*, which is based on the object-relational mapping. We are currently comparing the efficiency of the two approaches.

The operations for *accessing* and *updating* an object O are the same for both representations. They use a binary infix predicate ":=", which evaluates its right argument and tries to unify the result with its left argument. Given an element tag `E` and an attribute `A`, we use the call `X := O^E` to select the `E`–subelement `X` of `O`, and we use `Y := O@A` to select the `A`-value `Y` of `O`; the application of selectors can be iterated, cf. path expressions in XML query languages [1]. On *backtracking* all solutions can be obtained.

```
?- KS =
   'KnowledgeSlice':[
      'ID':'Rfb3955', type:'RuleComplex']:[
      'Action':[type:'ActionHeuristicPS']:[
         'Score':[value:'P6']:[],
         'Diagnosis':[ID:'P165']:[] ]
      'Condition':[type:or]:[
         'Condition':[ ..., value:'Msi250a2']:[],
         'Condition':[ ..., value:'Msi250a3']:[],
         'Condition':[ ..., value:'Msi250a4']:[] ] ],
   Type := KS@type,
   findall( Value,
      Value := KS^'Condition'^'Condition'@value,
      Values ).

Type = 'RuleComplex',
Values = ['Msi250a2', 'Msi250a3', 'Msi250a4']

Yes
```

To change the values of attributes or subelements, the call `X := O*As` is used, where `As` specifies the new elements or attribute/value–pairs in the updated object `X`. The following statements assign the value 'P3' to the score of an action:

```
?- A1 = 'Action':[type:'ActionHeuristicPS']:[
            'Score':[value:'P6']:[],
            'Diagnosis':[ID:'P165']:[] ],
   A2 := A1*[^'Score'@value:'P3'].

A2 = 'Action':[type:'ActionHeuristicPS']:[
        'Score':[value:'P3']:[],
        'Diagnosis':[ID:'P165']:[] ]

Yes
```

The library FNQUERY also contains additional, more *advanced methods*, such as the selection/deletion of all elements/attributes of a certain pattern, the transformation of subcomponents according to substitution rules in the style of XSLT, and the manipulation of path or tree expressions. Thus, FNQUERY can be used for transforming a given knowledge base into an arbitrary document, e.g., the GXL format for visualizing graphs.

3 Visualization of Knowledge Bases in VISUR

Queries and transformations on XML knowledge bases are applied using RAR. The results of the transformation process are visualized by graphs and tables using the component VISUR. VISUR/RAR can significantly improve the development cycle of logic programming applications, and it facilitates the implementation of techniques for syntactically analyzing and visualizing a given knowledge base. For obtaining efficiency and for representing complex deduction tasks we have used techniques from deductive databases and non–monotonic reasoning.

The goal of the system VISUR/RAR is to support the application of *knowledge engineering* and *refactoring* techniques, and the further system development. VISUR/RAR facilitates program comprehension and review, design improvement by refactoring, the extraction of subsystems, and the computation of software metrics (such as, e.g., the degree of abstraction). When we apply it to rule-based knowledge systems, then it helps developers in becoming acquainted with the knowledge base by visualizing dependencies between different questions and diagnoses defined by the available rules. It is possible to analyse knowledge bases customized to the individual needs of a user, and to visualize the results graphically or in tables.

In previous papers [6, 7] we have shown how also JAVA source code represented in XML can be analysed using VISUR/RAR. In the future, we will gradually extend VISUR/RAR with additional features. We intend to implement sophisticated methods for *program analysis* from *software engineering* [3–5], and we want to integrate further *refactoring techniques* for XML knowledge bases, some of which have been described in [13] for PROLOG.

4 Declarative Specification of Visualizations

In this section we give motivating examples for visualizing XML knowledge bases using VISUR/RAR. As mentioned in the introduction an appropriate visualization of the available knowledge depends on the current task. For large rule bases an exhaustive listing of all dependencies defined by the rules is not helpful, and also a focussing selection of the generated graph may be not meaningful. VISUR/RAR enables the developer of a rule base to flexibly generate visualizations in a declarative way; a visualization required for a specific task can be easily defined by a compact FNQUERY statement. In the following, we illustrate this by examples using the knowledge base of the SONOCONSULT documentation and consultation system for medical sonography [9].

4.1 Neighbours of a Question

A frequent maintenance operation for knowledge systems considers the restructuring of an already available question. E.g., the value range of a choice question is extended or the question is refined by two more precise questions. Such a restructuring operation can be simplified by a preceding visual analysis of the considered question and its dependencies with other knowledge objects, respectively. The following declaration determines all neighbours of a specified question (here with ID *Msi250*) defined by incoming and outgoing derivation rules:

```
knowledge_base_to_neighbours_of_question(KB, Edges) :-
   findall( Edge,
      ( knowledge_base_to_neighbour_of_question(KB, Edge),
        edge_is_incident_with_node(Edge, 'Msi250') ),
      Edges ).

knowledge_base_to_neighbour_of_question(KB, Cid-Rid-Qid) :-
   Rule := KB^'KnowledgeSlices'^'KnowledgeSlice',
   Rid := Rule@'ID',
   'RuleComplex' := Rule@type,
   Cid := Rule^_^'Condition'@'ID',
   ( Qid := Rule^'Action'^'Question'@'ID'
   ; Qid := Rule^'Action'^_@'ID' ).
```

In Figure 3 the result of this declaration can be visualized using VISUR.

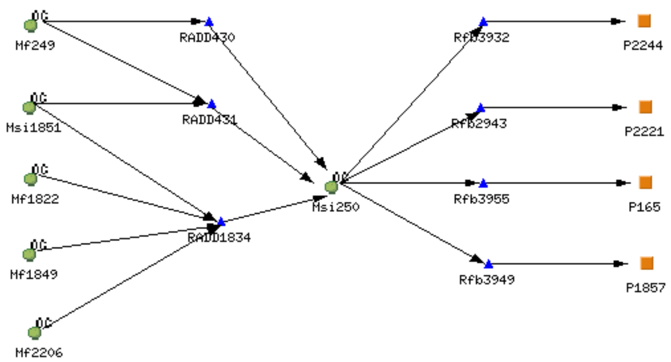

Fig. 3. Neighbours of Question Msi250.

It is easy to see that the five questions *Mf249*, *Msi1851*, *Mf2206*, *Mf1849*, and *Mf1922*, derive a value for the question *Msi250* using the three rules *RADD430*, *RADD431*, and *RADD1834*. Furthermore, there exist four outgoing rules *Rfb3955*, *Rfb3949*, *Rfb3932*, and *Rfb2943*, that derive values for the diagnoses *P165*, *P1857*, *P2244*, and *P2221*, depending on the value of *Msi250*.

4.2 Derivation Tree of a Diagnosis

The visualization of a derivation tree for a specified diagnosis can be helpful during the validation task. Then, the debugging of a false reasoning behavior (e.g.: a given diagnosis is not derived as a possible solution) is simplified by depicting its derivation graph. The following declaration generates the derivation tree for the diagnosis *P181*:

```
knowledge_base_to_rule_edges(KB, Edges) :-
    findall( Edge,
       knowledge_base_to_rule_edge(KB, Edge),
       Edges_2 ),
    reaching_edges('P181', Edges_2, Edges).

knowledge_base_to_rule_edge(KB, Cid-Rid-Qid) :-
   Rule := KB^'KnowledgeSlices'^'KnowledgeSlice',
   Rid := Rule@'ID',
   'RuleComplex' := Rule@type,
   Cid := Rule^_^'Condition'@'ID',
   Qid := Rule^'Action'^_@'ID'.
```

The resulting VISUR presentation is depicted in Figure 4.

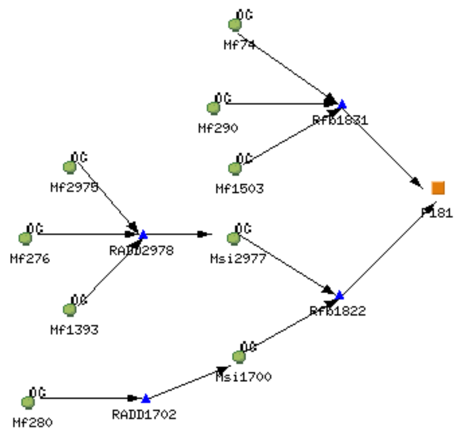

Fig. 4. Derivation Tree of Diagnosis P181.

For example, the derivation tree shows that the diagnosis *P181* is directly derived by the two rules *Rfb1831* and *Rfb1822*.

4.3 Compact Views of Knowledge Bases

Another possible application of the presented work is the transformation of an existing knowledge base into a more compact representation: existing XML structures and tags are combined or deleted in order to remove redundant or uninteresting information. The result of such a transformation can for instance be used for reporting issues.

The following declarations shrink the verbose representation of the *or* and *equal* condition by condensing disjunctions $Q = a_1 \vee \ldots \vee Q = a_n$ of equalities to more readable set conditions $Q \in \{ a_1, \ldots, a_n \}$.

```
knowledge_base_view(KB_1, KB_2) :-
   Substitution = [
      (X:As:Es)-('Condition':[type:X|As]:Es),
      Y-X-shorten_value_by_id(X, Y),
      In-Or-or_condition_to_in_condition(Or, In) ],
   fn_transform_elements_fixpoint(Substitution, KB_1, KB_2).

or_condition_to_in_condition(Or, In) :-
   fn_item_parse(Or, or:[]:Equals),
   first(Equals, equal:['ID':Id, value:_]:[]),
   maplist( d3_equal_to_in(Id),
      Equals, Vs ),
   In = in:['ID':Id]:Vs.

equal_to_in(Id, Equal, In) :-
   fn_item_parse(Equal, equal:['ID':Id, value:V]:[]),
   In = element:[value:V]:[].
```

The predicate `fn_transform_elements_fixpoint/3` of the library FnQuery offers a powerful substitution mechanism: E.g., the substitution

```
(X:As:Es)-('Condition': [type:X|As]:Es)
```

transforms a `'Condition'`-element of `type X` with the further attributes `As` and the sub-elements `Es` into a new element with the tag `X`, the attributes `As` and the sub-elements `Es`; the substitution

```
In-Or-or_condition_to_in_condition(Or, In)
```

transforms an XML-element `Or = or:[]:Equals`, such that all conditions in the list `Equals` have the same `'ID'`, into a new element `In = in:['ID':Id]:Vs`, where `Vs` contains elements `element:[value:V]:[]` with their different values.

When applied to the rule *Rfb3955* given in Section 2.1 the resulting knowledge slice is the following, which is more compact and readable than the original representation:

```
<KnowledgeSlice ID="Rfb3955" type="RuleComplex">
  <Action type="ActionHeuristicPS">
    <Score value="P6"/>
    <Diagnosis ID="P165"/>
  </Action>
```

```
    <in ID="Msi250">
      <element value="a2"/>
      <element value="a3"/>
      <element value="a4"/>
    </in>
  </KnowledgeSlice>
```

Our declarative query processing techniques can also be applied for producing an overview of the knowledge base in a reader-friendly format. Figure 5 displays an excerpt of the SonoConsult rule base in HTML format, which was also generated using the FNQUERY library (see Figure 2 for another excerpt).

Rule **Rfb3956** (ActionHeuristicPS)
if (Msi158 = a8 *and* M768 = a1 *and* Msi1510 = a1)
then P320 *with* P6

Rule **Rfb3955** (ActionHeuristicPS)
if (Msi250 ∈ { a2, a3, a4 })
then P165 *with* P6

Rule **Rfb3918** (ActionHeuristicPS)
if (Mf333 ∈ { a1, a2 })
then P529 *with* P6

Rule **Rfb3917** (ActionHeuristicPS)
if (Mf333 ∈ { a3, a4 })
then P530 *with* P6

Rule **Rfb4036** (ActionHeuristicPS)
if (Mf4027 = a2 *and* Mf1915 = a1)
then P4025 *with* P3

Fig. 5. Rules of SonoConsult in Compact Form

5 Conclusions

In this paper we have presented a declarative approach for querying and visualizing rule-based knowledge. We have introduced a possible XML representation for rule bases and we have briefly described the system VISUR/RAR: The component RAR allows for compact queries and transformations on XML documents, e.g., for transforming a specified part of an XML knowledge base into GXL or HTML format. The component VISUR in turn uses GXL documents for graph-based visualizations. The applicability of VISUR/RAR was demonstrated by motivating examples taken from a real world application.

In the future we are planning to generalize the presented work to further types of knowledge, e.g., case-based knowledge and model-based knowledge. Furthermore, the usability of the system can be increased by improving the interactivity between the user and the system. Thus, extended browsing techniques for the generated visualizations are necessary.

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. *S. Diehl (Ed.):* Software Visualization: International Seminar, Dagstuhl Castle, Germany, Springer LNCS 2269, 2002.
4. *H. Erdogmus, O. Tanir (Eds.):* Advances in Software Engineering - Comprehension, Evaluation, and Evolution, Springer, 2002.
5. *M. Fowler:* Refactoring – Improving the Design of Existing Code, Addison–Wesley, 1999.
6. *M. Hopfner, D. Seipel:* Reasoning about Rules in Deductive Databases, Proc. 17th Workshop on Logic Programming WLP 2002.
7. *M. Hopfner, D. Seipel, J. Wolff von Gudenberg:* Comprehending and Visualising Software based on XML Representations and Call Graphs, Proc. 11th IEEE International Workshop on Program Comprehension IWPC 2003.
8. *R. Holt, A. Winter, A. Schürr:* GXL: Towards a Standard Exchange Format, Proc. Working Conference on Reverse Engineering WCRE 2000, http://www.gupro.de/GXL/
9. *M. Hüttig, G. Buscher, T. Menzel, W. Scheppach, F. Puppe, H.-P. Buscher:* A Diagnostic Expert System for Structured Reports, Quality Assessment, and Training of Residents in Sonography, Medizinische Klinik, 2004, in press.
10. *F. Puppe:* Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3, International Journal of Human-Computer Studies (49), 627–649, 1998.
11. *A. Serebrenik, B. Demoen:* Refactoring Logic Programs, Proc. Intl. Conference on Logic Programming ICLP 2003 (Poster Session).
12. *D. Seipel:* Processing XML Documents in PROLOG, Proc. 17th Workshop on Logic Programming WLP 2002.
13. *R. Seyerlein:* Refactoring in deduktiven Datenbanken am Beispiel des Informationssystems Qualimed, Diploma Thesis, University of Würzburg, 2001.
14. *M. Smith, C. Welty, D. McGuinness:* OWL Web Ontology Language Guide, February 2004, `http://www.w3.org/TR/2004/REC-owl-guide-20040210/`
15. *J. Wielemaker, A. Anjewierden:* Programming in XPCE/PROLOG `http://www.swi-prolog.org/`