Declaratively Querying and Visualizing Knowledge Bases

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. Visualization techniques are a promising approach for simplifying the maintenance of large knowledge systems. Thus, extensions or modifications of a knowledge base can be supported by appropriate visualizations, e.g., illustrating dependencies of the considered knowledge.
In this paper, we introduce a declarative approach for querying and visualizing rule based knowledge represented as XML documents. Then, the currently required part of the knowledge base can be inspected visually by ad-hoc declarations.

Keywords. knowledge system, rule base, PROLOG, visualization, XML query/transformation

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, since (transitive) dependencies of this rule are not obvious at first sight.
In this paper we introduce a declarative approach for generating a flexible graph-based visualization of rule-based knowledge. 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 depended 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 (transitive) derivation graph of a solution object defined by its deriving rules can assist the expert during a debugging session.
- *Examination of 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. Then, 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 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. Thus, for a reasonable application of visualization techniques in a real-world environment a flexible and adaptive visualization tool is required.

In this paper we introduce a declarative approach for flexibly defining graph-based visualizations of rule bases. This declarative approach enables for a fast ad-hoc definition of visualizations based on given requirements of the expert. Then, we assume a knowledge base to be available in XML format and provide a compact set of query and transformation syntax for generating reports about the knowledge base in the form of graphs in GXL format. GXL is a standard format for graph-based structures, which can be visualized by VISUR. In principle, our approach is not restricted to the visualization of rule bases but can be generalized to arbitrary XML documents. For example, in [8] the visualization of software programs using VISUR/RAR has been demonstrated, i.e., procedural or declarative programs such as JAVA or PROLOG programs. For the scope of this paper we will focus on the application of the visualization of rule bases containing different types of explicit rule knowledge.

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. The paper is concluded in Section 5 by summarizing the presented work and giving promising directions for future work.

2 Representation of Knowledge Bases

Complex structured objects such as, e.g., knowledge bases need to be represented in XML. The presented system VISUR/RAR is part of the DISLOG toolkit [14], which provides the library FNQUERY for handling these XML data.

In general, 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. Then, RAR (*reasoning about rules*) is applied for query and modification tasks, and VISUR is applied for visualizing the results obtained by RAR. For the visualization VISUR handles results given in the *Graph eXchange Language* GXL [9]; 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 will sketch the processing of knowledge bases using RAR.

2.1 Knowledge Bases in XML

For our examples we focus on a particular XML schema for knowledge bases, which is used by the D3 system [12]. The shell-kit D3 has been applied in many medical, technical, and biological domains for building diagnostic knowledge systems.

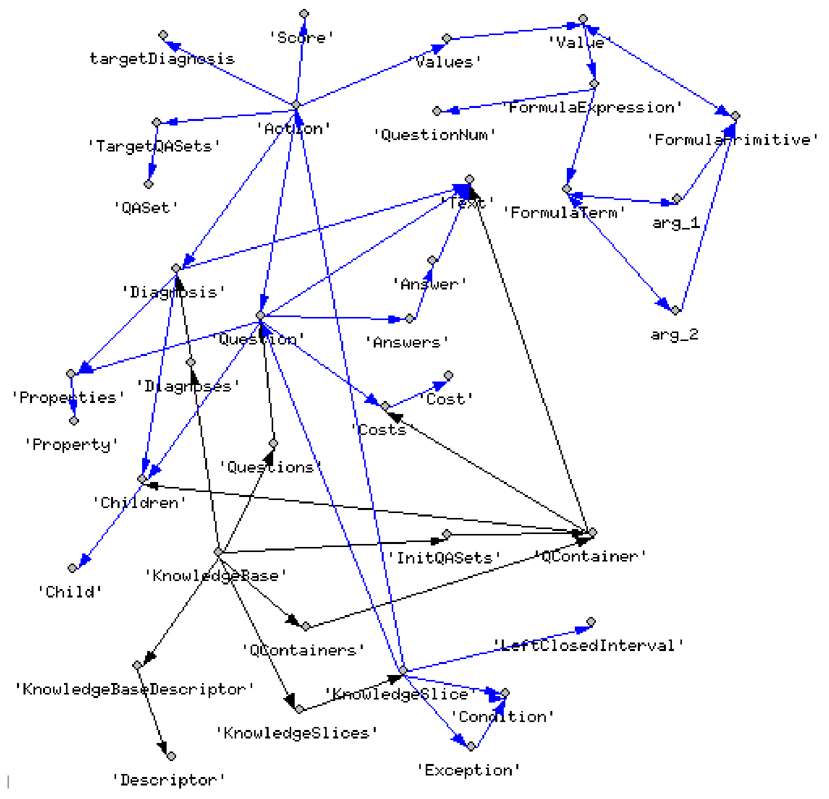

Fig. 1. Rule/Goal Graph in VISUR

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` (available questions to be asked to the user), `Diagnoses` (possible solutions inferred by the system), and `KnowledgeSlices` (available inferential knowledge connecting questions and diagnoses). All further objects, e.g., diagnoses, questions, and rules, are identified by unique IDs represented as a corresponding attribute in the XML file.

The most interesting part of the knowledge base is the collection of knowledge slices, i.e., the available rules. An exemplary rule with ID *Rfb3955* is given in the following. We can see, that a rule generally consists of an action part and a condition part.

```
<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=""/>
  </Condition>
</KnowledgeSlice>
```

In our example, the condition of the rule evaluates to true if the question with ID *Msi250* has either the value *Msi250a2*, *Msi250a3*, or *Msi250a4*. If the condition is true, then a score *P6* is given to diagnosis with ID *P165*, i.e., the diagnosis is assumed to be a possible solution. In general, the value range of scores given to a diagnoses is defined as $SC = \{N7, \ldots, N1, P1, \ldots, P7\}$ with scores $N1, \ldots, N7$ for disconfirming a diagnosis in ascending order, and with scores $P1, \ldots, P7$ for confirming a diagnosis in ascending order. In turn, the value question *Msi250* need not to be entered by a user, but can be derived by another rule also contained in the knowledge base. Thus, a knowledge base can describe transitive paths for inferring a diagnosis. For a more detailed discussion of the applied knowledge representation we refer to [12].

2.2 Processing Knowledge Bases with FnQuery

The library FnQuery can be interpreted as an extension of Prolog containing the library FnQuery for processing XML documents. 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. E.g., for the XML representation of the knowledge slice from above we get:

```
'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']:[] ] ]
```

There exist several possibilities to *access* and *update* an object `O` in field notation using 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, for a given query RAR can transform 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, which generates a GXL document. 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). 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 [7, 8] we have shown how also JAVA source code can be analysed using VISUR/RAR. For gaining sufficient performance on large programs such as DISLOG we use techniques from the field of deductive databases. 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 developed in [15] 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 illustration 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 queries and visualizations in a declarative way. Thus, a visualization required for a specific task can be easily defined by a compact FNQUERY statement. In the following we will illustrate this by examples using the knowledge base of the SONOCONSULT system [10], which is a fielded knowledge-based documentation and consultation system for sonography.

4.1 Neighbours of a Question

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

```
d3_knowledge_base_to_neighbour_of_question(KB, Edge) :-
   Rule := KB^’KnowledgeSlices’^’KnowledgeSlice’,
   R := Rule@’ID’,
   ’RuleComplex’ := Rule@type,
   V := Rule^_^’Condition’@’ID’,
   W := Rule^’Action’^_@’ID’,
   ( V = ’Msi250’
   ; W = ’Msi250’ ),
   ( Edge = V-R
   ; Edge = R-W ).
```

In Figure 2 the result of the declaration given above is visualized using VISUR.

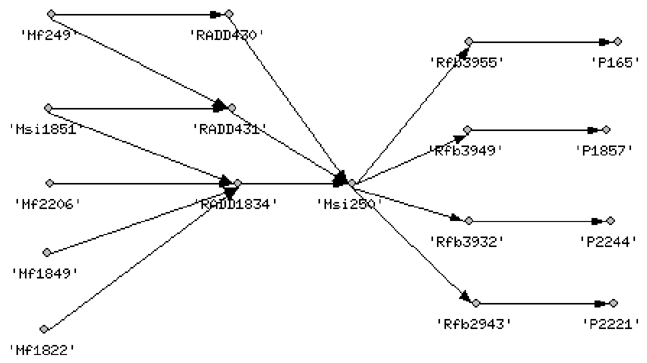

Fig. 2. Neighbours of specified question.

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

4.2 Transitive Derivation Tree of a Diagnosis

The visualization of a transitive 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 is derivation graph. The following declaration generates the transitive derivation tree for the diagnosis with ID *P181*.

```
d3_knowledge_base_to_rule_edges(KB, Edges) :-
    findall( Edge,
       d3_knowledge_base_to_rule_edge(KB, Edge),
       Edges_2 ),
    reaching_edges('P181', Edges_2, Edges).

d3_knowledge_base_to_rule_edge(KB, Edge) :-
   Rule := KB^'KnowledgeSlices'^'KnowledgeSlice',
   R := Rule@'ID',
   'RuleComplex' := Rule@type,
   V := Rule^_^'Condition'@'ID',
   W := Rule^'Action'^_@'ID',
   ( Edge = V-R
   ; Edge = R-W ).
```

The resulting VISUR presentation is depicted in Figure 3.

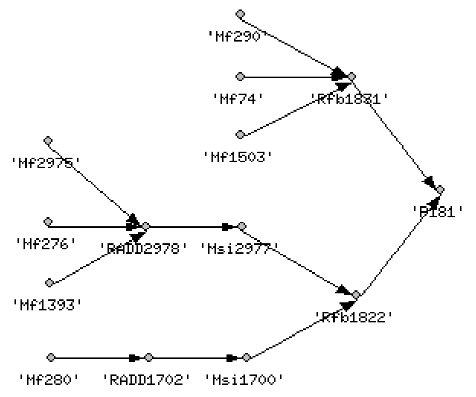

Fig. 3. Transitive derivation tree of the diagnosis *P181*.

For example, the transitive derivation tree shows that the diagnosis *P181* is directly derived by two rules (i.e., *Rfb1831*, *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. Then, existing XML structures and tags are combined or deleted in order to remove redundant or uninteresting information. The result of such a transformation can be, e.g., used for reporting issues. The following declarations shrink the verbose representation of the *or* and *equal* condition.

```
d3_knowledge_base_view(KB_1, KB_2) :-
   Sub = [
      (X:As:Es)-('Condition':[type:X|As]:Es),
      _-(equal:['ID':_,value:_]:[])-shorten_value_by_id,
      _-(_)-d3_or_condition_to_in_condition ],
   fn_transform_elements_fixpoint(Sub, KB_1, KB_2).

d3_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.

d3_equal_to_in(Id, Equal, In) :-
   fn_item_parse(Equal, equal:['ID':Id, value:V]:[]),
   In = element:[value:V]:[].
```

If applied to the rule with ID *Rfb3955* given in Section 2.1 the resulting knowledge slice is the following:

```
<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>
```

The resulting XML structure is more compact and readable than the previous one.

5 Conclusions

In this paper we have presented a declarative approach for querying and visualizing rule-based knowledge. We introduced a possible XML representation for rule bases and described the system VISUR/RAR consisting of the parts RAR and VISUR: RAR allows for compact queries and transformations on XML documents, e.g., for transforming a specified part of an XML knowledge base into GXL format. 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 other 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. Hanus, J. Koj:* An Integrated Development Environment for Declarative Multi–Paradigm Programming, Proc. Workshop on Logic Programming Environments WLPE 2001.
7. *M. Hopfner, D. Seipel:* Reasoning about Rules in Deductive Databases, Proc. 17th Workshop on Logic Programming WLP 2002.
8. *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.
9. *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/
10. *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, submitted to Medizinische Klinik, 2003.
11. *IBM:* The Integrated Development Environment ECLIPSE, http://www.eclipse.org/
12. *F. Puppe:* Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3, International Journal of Human-Computer Studies (49), 627–649, 1998.
13. *A. Serebrenik, B. Demoen:* Refactoring Logic Programs, Proc. Intl. Conference on Logic Programming ICLP 2003 (Poster Session).
14. *D. Seipel:* Processing XML Documents in PROLOG, Proc. 17th Workshop on Logic Programming WLP 2002.
15. *R. Seyerlein:* Refactoring in deduktiven Datenbanken am Beispiel des Informationssystems Qualimed, Diploma Thesis, University of Würzburg, 2001.
16. *J. Wielemaker, A. Anjewierden:* Programming in XPCE/PROLOG http://www.swi-prolog.org/