Visualization Techniques for the Evaluation of Knowledge Systems

Joachim Baumeister, Martina Menge and Frank Puppe
Institute of Computer Science
University of Würzburg, Germany
Contact: joba@uni-wuerzburg.de

Abstract

Although various methods for the evaluation of intelligent systems have been proposed in the past, almost no techniques are present that support the manual inspection of knowledge bases by the domain specialist. In this paper, we discuss a collection of appropriate visualization techniques that help developers of a knowledge base to interactively browse and analyze the knowledge base in order to find deficiencies and semantic errors in the implementation. We describe standard visualization methods adapted for knowledge base analysis, and we propose a novel visualization technique to support the manual inspection task. The application of the shown methods was motivated by daily practice of knowledge base development.

Introduction

Whereas the evaluation of intelligent systems has been investigated thoroughly in the past, only little attention was paid to its possible support by visualization techniques. In this paper we describe and characterize already known visualization methods that can be used in the context of the evaluation of knowledge bases, and we will introduce a novel visualization technique that supports the validation of the derivation and interview behavior of an intelligent system in a semi-automatic manner.

In the context of validation tasks many automated methods have been proposed, e.g., (Vermesan & Coenen 1999). Most of these methods consider the evaluation of the final result of the intelligent system, i.e., the derivation quality at the end of a problem-solving session. However, for real-world systems the interactivity between the user and the system is also an important issue. Here, the system implements an interactive interview with the user in order to collect only the relevant findings required for deriving the appropriate solution. The order and number of questions asked before deriving a final solution is often a critical feature of these interactive systems, because a lengthy dialog or an unintuitive order of the questions may irritate the user of the system, then canceling the interview in the worst case. Another important issue is the actual usage of the system when deployed into a real-world application. The direction of refinement of such a system can depend on the usage statistics of the system, i.e., frequently used parts of the knowledge base may be considered to be improved with respect to the interview efficiency, very rarely used parts may be considered for structural rework or deletion.

In this paper, we motivate that visualization methods can support validation tasks efficiently, especially when automated methods cannot be used. However, it is not intended to give an exhaustive discussion of visualization methods for validation tasks but rather to present a selection of useful methods for which their application was motivated by the experiences we have made during the development of various knowledge bases. In the next section we give a brief categorization of visualization techniques to be used in different evaluation tasks. Thereafter, we introduce the novel visualization technique *d/d-nets* to semi-automatically validate interactive knowledge systems in more detail. This method is applicable to validate the correct sequence and derivation progress of interview-based systems. We briefly describe some case studies in the context of medical knowledge systems, and we conclude the paper with a summary and an outlook to future work.

Visualization for Evaluation

According to (Fluit, Sabou, & van Harmelen 2006) the following general use cases of visualization techniques can be identified: *data analysis*, *querying*, and *exploration*. In the context of evaluation the use case *data analysis* can be refined to the term *knowledge base analysis*, that have been often also named *manual inspection* of the knowledge base. Here we identify the following important sub-tasks:

- The analysis of the **usage of the system** can be inspected by using visualization techniques like (pie or bar) charts and treemaps.
- The **structure of the knowledge base**, i.e., the actual implementation details, can be inspected when visualizing the knowledge base by graph-based techniques, for example derivation graphs (Seipel, Hopfner, & Baumeister 2005) and clustermaps (Fluit, Sabou, & van Harmelen 2006).
- The **workflow of the system behavior**, i.e., the actual sequence of typical interactive problem-solving sessions and the solution output of the system. To the best knowledge of the authors no current visualization technique can

be directly applied. We therefore introduce the concept of *d/d-nets* that is capable to visualize interactive problem-solving sessions.

For all presented visualization techniques the *intuitive* presentation is the most important feature, since we want to have these techniques to be used and understood by domain specialists that are often not trained knowledge engineers.

Pie Charts: The Usage of the System

Often, the usage of the system is related to the actual design of the knowledge base. A prominent design flaw, that corresponds to the usage of the system, is the *lazy object*, i.e., an input or output object that is never/rarely used (Baumeister & Seipel 2005). Thus, very frequently used as well as very rarely used inputs and solutions are the typical target objects. For example, very frequently derived solutions should be considered to be refined to a more diverse set of sub-solutions in order to improve the capabilities of the system. In contrast, solutions that are never derived by the system *can* point to an incorrect implementation of the system, e.g., the presence of semantically inconsistent rules that avoid the derivation of these solutions. Also, rarely or never derived solutions *can* point to an existing redundancy in the knowledge base, when they have been superseded by specialized or generalized solutions in the past, but were accidently not removed by the developer. Similar considerations can be made for questions that have been almost never answered by the user or were answered with a default answer like, e.g., *answer unknown*. These objects may either point to an unsuitable question in the given context, or may point to a question that is actually difficult to answer for standard users. Like rarely used solutions, the observation of infrequently used abstractions may also uncover semantically incorrect rules for deriving them.

Although, the analysis issues described above have to be assessed by a domain specialists in a manual manner, visualization techniques can give an efficient support. Then imbalanced usage of objects can be traced intuitively. For the usage analysis of the systems we propose different implementations of pie charts, since their semantics are clear and commonly understood.

The basic structure of a pie chart for the usage of questions is as follows: we render one pie for each question showing the distribution of its particular answers. For numerical questions we use standard binning methods, if no partitioning of the value range was already defined in the knowledge base. Within this pie two important additional aspects of the usage are tackled:

1. How often is the question answered by the users in general?
2. If the question was answered in the corresponding cases, then how often it was answered with the default answer "unknown"?

The first aspect ("how often answered in general") is rendered by an additional circle around the pie depicting all cases that are inspected: the green colored part of this circle corresponds to the percentage of given answers, whereas the rest of the circle is colored red showing the percentage of no answers given for this question. The second aspect ("how many 'unknown' answers") is tackled by an additional piece of the pie colored in red, i.e., the red colored piece corresponds to the percentage of the given answers "unknown" in all answered cases.

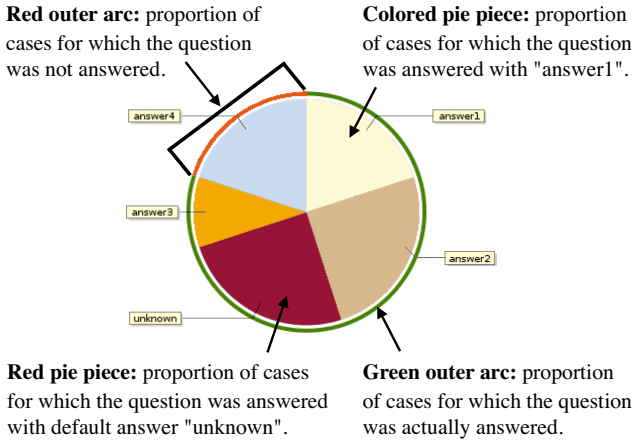

Figure 1: An example pie chart showing the usage of a question with respect to the stored problem-solving sessions.

Figure 1 shows an example pie chart for one question. Usually, the visualization method renders a separate pie chart for every question in a table like manner. While browsing the charts we can easily identify the set of questions, that were for example answered with the default answer "unknown". Furthermore, browsing the charts can give an initial impression, whether the distributions of the answers given by the users are balanced.

Derivation Graphs: The Knowledge Base Structure

The analysis of the structure of a knowledge base using a graph-based representation has been discussed extensively, e.g., (Seipel, Hopfner, & Baumeister 2005). Here, rules are represented as a subgraph where the findings of a rule condition are connected by appropriate edges. The graph-based depiction of a derivation path can help the developer during the manual inspection of the knowledge base, e.g., when investigating the surrounding area of a found anomaly. Additionally, a graph visualization can help to intuitively identify objects that are frequently used for the derivation of many solutions, e.g., *derivation hubs*. In general, derivation hubs can worsen the robustness of a system with respect to noisy or missing input data.

Figure 2 shows a part of the derivation paths for the example output "solution 1". Structural deficiencies can be identified when following the paths of the derivation graph. In a practical application users would typically ask for rendering all rules for a specific solution. For mid-size and larger knowledge systems such an approach is only reasonable when the graph can be browsed interactively, then sub-conditions are only expanded by clicking the corresponding nodes.

Besides these simple derivation graphs we propose a *clustered derivation graph* for rule bases that was inspired by

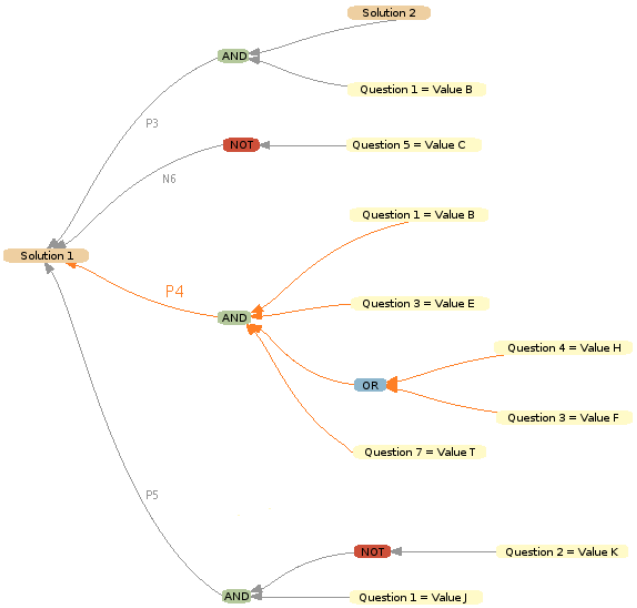

Figure 2: An example derivation graph for a solution with four rules.

the ClusterMaps technology for the visualization of lightweight ontologies (Fluit, Sabou, & van Harmelen 2006). It is useful for depicting the joint use of findings by different solutions. Findings are rendered together in a joint node of the graph, if they are used for deriving more than one solution. Figure 3 shows a plant classification system taken from the biological domain; here, the clustered derivation graph of four solutions (depicted by bubbles with their names) is presented. Each ball in the outer clusters (only connected to one solution) represents one finding that is only used to derive the specific solution. The inner clusters are connected to more than one solution, and findings contained in those clusters are used to derive the connected solutions, i.e., clusters labeled with (a) - (d). For example, the three findings contained in cluster c are used for the derivation of "Busch-Wind…" as well as for the derivation of "Echter Steinklee…". The balls are colored to indicate the derivation weight of the finding: balls having a red or orange color are mainly used for excluding the connected solutions, and balls in light and dark green color represent findings for enforcing the derivation of the connected solutions. In a dynamic variant of the clustered derivation graph the usage of the findings is additionally depicted by the size of the particular balls, i.e., frequently used findings are represented by larger balls than less frequently used findings. It is worth noting that every finding only occurs once in a clustered derivation graph. With such a clustered variant the developer can easily identify groups of findings that are commonly used by multiple solutions, i.e., derivation hubs. In an interactive application a click on a specific cluster opens a detailed derivation graph for a further inspection of the corresponding rules.

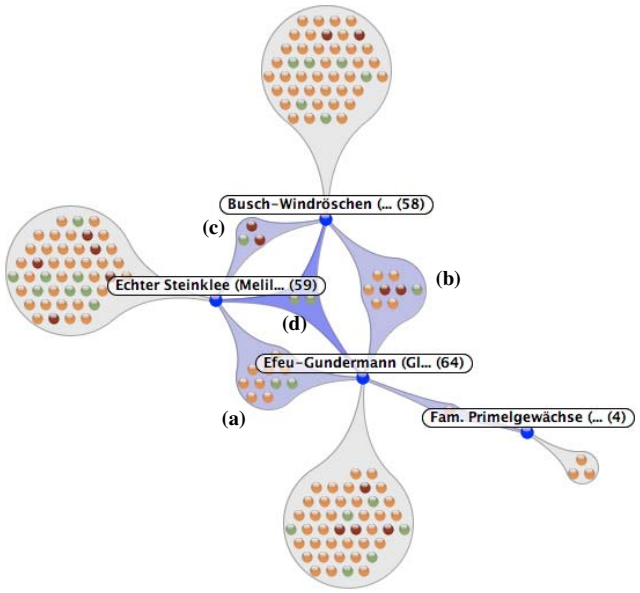

Figure 3: The clustered derivation graph of a plant classification system.

Interview Trees: Tracking the User Answers

Larger real-world systems often embody the definition of various solutions and consequently many questions are included that are used to derive the particular solutions. Then, for a specific solution to be derived not the entire set of questions is required to be answered by the user. For this reason, it is reasonable to implement an adaptive dialog strategy that focusses on refined questions that are suitable for the problem currently stated by the user. An example of such a technique is the dialog strategy of *decision trees*; here, users are guided through a dialog path by follow-up questions that depend on the previously given answers.

For larger systems, questions are commonly grouped in questionnaires, i.e., lists of questions, that are jointly answered before follow-up questionnaires are presented to the user. The proposed graph-based visualization technique presents the structure of the follow-up questionnaires and questions of the knowledge base. Such an overview of the dialog tree can help the developer to find redundant calls of questionnaires and inefficiencies of the dialog sequence. Furthermore, unreachable questionnaires can easily be identified since such elements are displayed as isolated nodes in the graph.

Figure 4 depicts the interview structure of a larger medical consultation system. Questionnaires are represented by boxes with rectangles and single (follow-up) questions by boxes containing green balls.

The shown graph is extended by displaying the usage of the particular paths corresponding to a given set of cases that were stored in previous problem-solving sessions. Here, the usage of the dialog sequences is shown by the thickness of the edges, i.e., frequently used paths are thicker than less used paths. Additionally, the number of uses is attached to

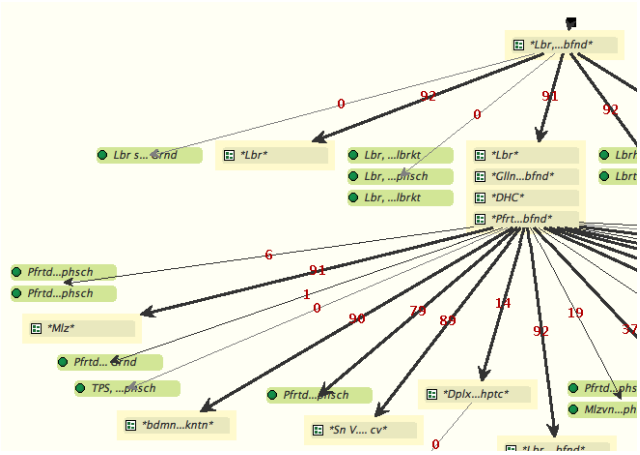

Figure 4: An excerpt of the interview structure of a medical consultation system with actual uses of the particular paths.

the edges, i.e., the number of cases that used this sequence. Besides the static variant that is displaying the complete interview sequences of the knowledge base, we also provide an interactive version where the developer starts from an initial questionnaire. By clicking on that questionnaire all possible follow-up questions and questionnaires are unwrapped and the graph is extended subsequently by further clicks of the user.

d/d-nets: A Black-Box Inspection Technique

An important aspect of the evaluation is the analysis of the derivation results together with the sequence of the interview during the problem-solving process. To the best knowledge of the authors there is no method available that solves this task sufficiently. For example, typical empirical testing methods only consider the derivation quality of the knowledge system, without taking care of how the required findings were acquired from the user. For the automatic testing of the interview structure *partially ordered question sets* and *diagnosis related question sets* were introduced by (Baumeister 2004, p. 140ff), but these techniques only evaluate the interview at the coarse level of questionnaires and not the order of the actually presented questions.

In this section, we introduce the novel black-box testing method *d/d-nets* (for *dialog/derivation nets*) to analyze the derivation and interview behavior of an intelligent system in a semi-automated manner. The benefit of this method is the independence from the underlying representation of the interview and derivation knowledge. The only precondition of the method with respect to the inference engine is that – at any time – we can query the current states of the possible solutions and that we can retrieve the next question to be presented to the user. However, in its current state d/d-nets are limited to smaller knowledge bases due to the exponential growth of the net during its construction process.

In the following we give an overview of the method, where we describe the process of the net construction and the subsequent testing phase in more detail.

Introduction to d/d-nets In general, a d/d-net is a tree-like graph, in which every path from the root to a leaf of the tree represents a possible dialog of the user. Inner nodes of the trees describe (follow-up) questions that are asked by the system, whereas outgoing edges of a node are annotated by the possible answers of the particular question. A question with more than two possible answers will generate the corresponding number of edges. Thus, d/d-nets are not limited to binary branching. Edges of the d/d-net are also attached by a box listing all solutions that are currently derived as possible solutions at this stage of the problem-solving process. Leaves of a d/d-net are the final solutions of the system that are derived in this particular case, e.g., see Figure 5a. In order to retain the tree structure of the graph, one question can appear multiple times within the tree.

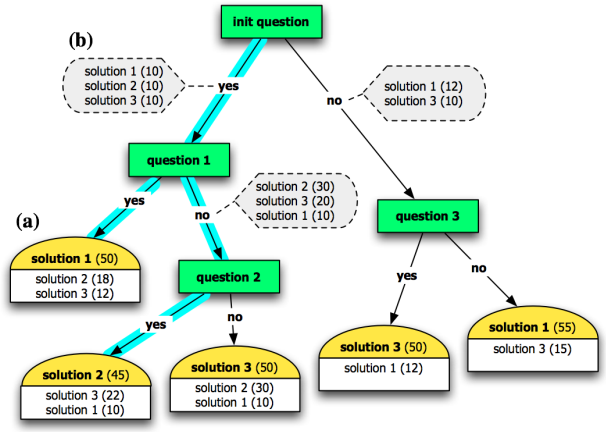

Figure 5: An example d/d-net graph.

A schematic d/d-net is depicted in Figure 5. For example, if "init question" is answered with the alternative "yes", then the follow-up question "question 1" is presented to the user and the solutions 1-3 are derived as intermediate solutions as denoted in the attached ballon, e.g., see Figure 5b. A ranking of the solutions is defined by the numbers given in parentheses behind the solution names. If "question 1" is then answered with the alternative "yes", then the final result "solution 1" is derived with the solutions 2 and 3 as alternative results (having a significant lower rating than "solution 1"). We can see that the proposed visualization can be intuitively used by a domain specialist to inspect the dialog and the derivation behavior of the developed knowledge system, i.e., the developer has to trace every path of the tree in order to validate the system behavior.

The d/d-net Process Model We describe the overall method for using d/d-nets in a semi-automatic evaluation task:

1. **Initialization:** Create an initially empty collection of previously reviewed cases $PRC = \{\}$.
2. **Construction and visualization of the d/d-net:** The graph is visualizing all possible interview traces of a developed knowledge system.

(a) All possible cases are recorded using an automated interview bot, i.e., we fill the set of recorded cases RC. The bot simulates an interactive dialog with the knowledge system by iteratively answering the possible values of the currently presented question. After providing an answer to the current question, the bot recursively retrieves the next follow-up question to be answered. A new case is stored if no follow-up question is asked by the system any more. During the simulation of an interview the bot also stores the intermediate solutions, that are derived during the problem-solving session.

(b) The recorded cases are rendered using a rooted tree graph drawing algorithm, see (Sugiyama 2002). The sequences of the recorded but previously reviewed cases $c \in PRC \cap RC$ are highlighted in the d/d-net as for example shown in the left branch in Figure 5.

3. **Manual review of the d/d-net:** Every possible case, i.e., every path from the root to a leaf, is manually inspected by a domain specialist (not necessarily the developer of the knowledge base). Here, only previously unreviewed cases $c \notin RC$ need to be reviewed.
   For this step we recommend to print out the entire graph on a poster in order to obtain a better overview of the interview workflow. Here, already traversed and reviewed paths can be easily highlighted with a text marker. For example, this was done in Figure 5 for the left branch of the tree, i.e., the cases
   {(init question = yes; question 1 = yes),
   (init question = yes; question 1 = no; question 2 = yes)}.
4. **Storing the test suite:** If all reviewed cases are inspected successfully and are marked as *correct* by the domain specialist, then these cases are also stored in the test suite of "previously reviewed cases" PRC.
5. **Knowledge modification:** After changing the knowledge base, the previous steps are iterated starting with step 2. All previously reviewed cases – that have not changed in this iteration – are highlighted in the tree. Thus, the domain specialist intuitively identifies the new or changed paths in the tree that have to be reviewed in this iteration.

Im comparison to a directed-acyclic graph (DAG) we choose the tree as the most appropriate visualization of a d/d-net because of the following two reasons: First, the intermediate results (states of solutions) in one case may not hold for all dialog paths (representing other cases) incoming to a particular node. Second, a tree figure greatly simplifies the manual highlighting technique when tagging an reviewed case; using a DAG may result in multiple markers of one edge for more than one case.

In summary, the described method proposes a complete life-cycle methodology for the validation of knowledge systems. It suggests a mixed-initiative technique that not only considers the derivation quality of the final solutions but also the intermediate solutions together with the interview structure. When changing the knowledge base the system computes all possible interview sequences and compares these sequences with the already reviewed cases. Due to the visual labeling of the unchanged interview sequences the domain specialist only has to review new or changed cases. The visualization technique is very important in this context, since it allows for an intuitive demonstration of the context of the particular cases, which is very important for the domain specialist during a manual inspection.

Case Studies

The presented visualization techniques were developed as a plug-in of the system KnowME, a modeling environment for the agile development of knowledge systems, that already provides tools for the evaluation and refactoring of knowledge bases (http://d3web.sourceforge.net). For all the visualization methods we identified the application of abbreviation techniques as a very important issue, since typical users want to see the names of the particular objects in a graph or pie chart at any time. Having the object names in the visualization at any time improves the efficiency of the manual inspection significantly. However, printing the full name of each object will commonly result in a cluttered visualization and will hinder any positive inspection experience. In our case study we found a combined abbreviation technique to be most useful (Stum *et al.* 1991), i.e., erasing the center of the word and a conservative elimination of vocals. In the following, we demonstrate the presented visualization methods pie charts and interview trees by some real-world knowledge bases.

Interview Trees

The visualization of interview flows is helpful for the developer for almost all larger knowledge bases that embody a dialog logic. We exemplify its use describing a consultation and documentation system for dental medicine, i.e., a system that is used to record the data and patient's findings acquired during the examination by a dentist. Typically, the first examination of a patient implies the acquisition of the basic patient's data together with a complete registration of the usual findings of the patient's teeth. Follow-up consultations normally record a less detailed collection of findings, but will provide a detailed questionnaire if indicated by the physician. The knowledge base contains 455 questions grouped by different questionnaires. Figure 6a shows an expanded tree depicting the possible sequences of the interview. Rectangular boxes represent questionnaires containing the particular questions, whereas rounded boxes with green circles show single questions. The tree is expanded by these questions, if the user clicks on the corresponding questionnaire rectangle. Starting with the root, i.e., the initial questions, edges represent the possible indications of further questionnaires. In summary, the tree of dialog sequences is very helpful to track the possible indications of questionnaires for the particular use cases.

Pie Charts for Detecting Lazy/Busy Objects

Figure 6b shows the pie charts of a questionnaire taken from a medical consultation system for sonography (Hüttig *et al.* 2004). Questions that were never answered in the given cases can be easily identified since they are displayed as

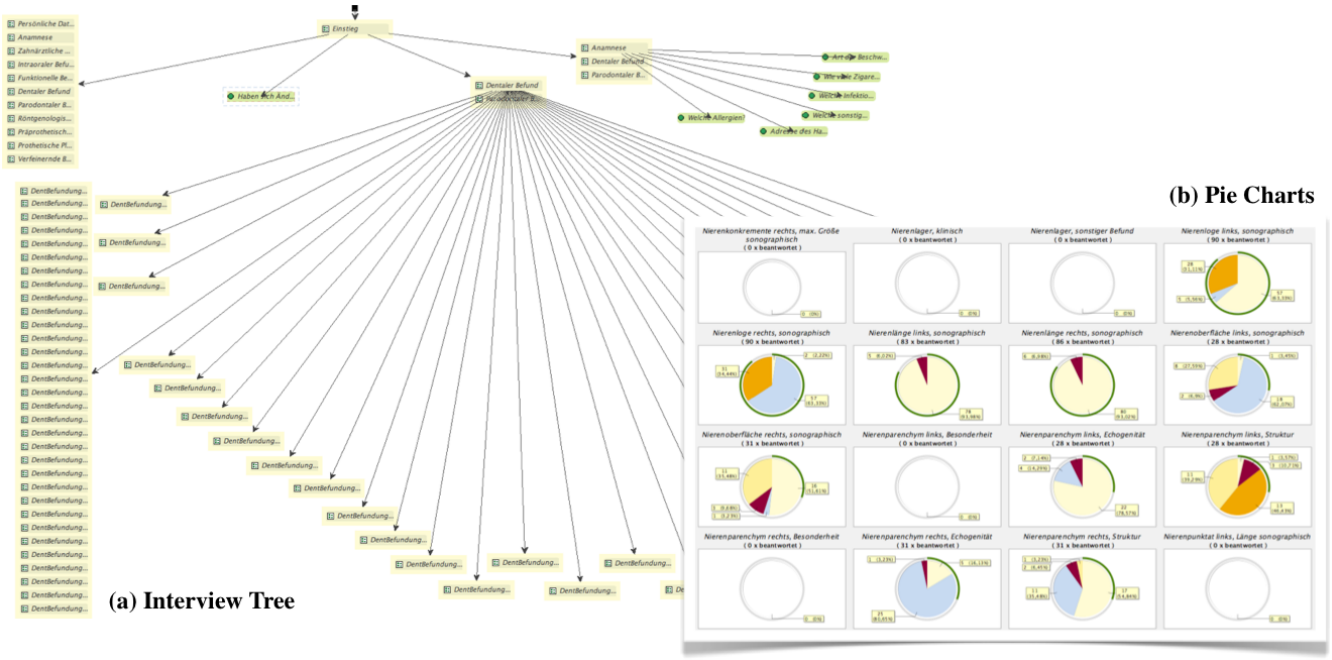

Figure 6: (a) The complete dialog logic of a knowledge system for dental consultation and documentation. (b) Pie charts depicting the usage of questions in a questionnaire of a medical documentation system for sonography.

white, empty circles. For the remaining questions the percentage of answers given can be inspected by the green outer circle. Last but not least, the dark red pie pieces represent the percentage of how much the default answer "unknown" was provided.

Conclusions

Although, many automatic methods for the evaluation of intelligent systems have been proposed in the past often the manual inspection of the implemented knowledge is necessary, e.g., to interactively discover design anomalies or deficiencies in the knowledge base. Visualization techniques can significantly support the manual inspection of a knowledge base. We presented a non-exhaustive overview of visualization techniques that can be applied for the manual inspection of a knowledge system. We focussed on a selection of methods that we found helpful to support the evaluation tasks in our daily practice for developing and maintaining knowledge bases. In general, there exists a vast collection of diverse visualization techniques, and therefore our future work will consider a normative description and discussion of the applicability of the particular classes of visualization methods with respect to evaluation tasks. We demonstrated the applicability of the visualization methods pie charts and interview trees by case studies. d/d-nets will be deployed in a real-world case study in the near future; however, the visualization method and the process model was invented together with the domain specialist that is intended to accomplish the evaluation.

References

Baumeister, J., and Seipel, D. 2005. Smelly Owls – Design Anomalies in Ontologies. In *FLAIRS'05: Proceedings of the 18th International Florida Artificial Intelligence Research Society Conference*, 215–220. AAAI Press.

Baumeister, J. 2004. *Agile Development of Diagnostic Knowledge Systems*. IOS Press, AKA, DISKI 284.

Fluit, C.; Sabou, M.; and van Harmelen, F. 2006. Ontology-based information visualization. In *Visualizing the Semantic Web*. Springer. 36–48.

Hüttig, M.; Buscher, G.; Menzel, T.; Scheppach, W.; Puppe, F.; and Buscher, H.-P. 2004. A Diagnostic Expert System for Structured Reports, Quality Assessment, and Training of Residents in Sonography. *Medizinische Klinik* 3:117–22.

Seipel, D.; Hopfner, M.; and Baumeister, J. 2005. Declarative Querying and Visualizing Knowledge Bases in XML. In *INAP/WLP'04: Applications of Declarative Programming and Knowledge Management (selected papers), LNAI 3392*, 16–31. Berlin, Germany: Springer.

Stum, G. M.; Demasco, P. W.; ; and McCoy, K. F. 1991. Automatic Abbreviation Generation. In *RESNA 14th Annual Conference*, 97–99. Washington, D.C.: RESNA Press.

Sugiyama, K. 2002. *Graph Drawing and Applications for Software and Knowledge Engineers*. World Scientific.

Vermesan, A., and Coenen, F. 1999. *Validation and Verification of Knowledge Based Systems. Theory, Tools and Practice*. Kluwer Academic Publisher.