Knowledge Formalization Patterns

Frank Puppe
Universität Würzburg, Lehrstuhl für Informatik VI, Am Hubland, D-97074 Würzburg, Germany, Fon: 0049-931 888 6730, Fax: 6732, Email: puppe@informatik.uni-wuerzburg.de

Abstract: Knowledge formalization patterns use the idea of object oriented design patterns and the UML notation for expressing well-tried approaches to build not to complicated knowledge systems. This is illustrated with the pattern "heuristic decision trees", which extends ordinary decision trees to cope with typical problems of heuristic classification. First experience showed that the standardization saves 1/3 of the effort for building knowledge bases by domain experts. A library of such patterns might help people from outside the knowledge based community to build knowledge systems much more easily. An open question is, to what complexity of problem solving methods the description formalism scales up.

1. Introduction

The ongoing exponential growth in all kinds of internet businesses causes a high demand for consultation. The easier it becomes to buy books, computers, software, travels, stocks etc. via the net, the more increases the demand for advise how the customers needs are matched by products as well as how to solve problems when using the product. In addition, pure consultation services like e.g. legal, wellness or medical advise independent of concrete products are in high demand. Due to the high competition between offers of products and services caused by the internet, the technology of knowledge based systems with the capability to offer individualized consultation gets increased attention. It has to compete with hypermedia based information systems and sophisticated search engines. They are easy to build, allow therefore mass fabrication by domain experts and work well in a distributed environment. The latter - in the context of knowledge systems - is dealt with multiagent systems [Wooldridge & Jennings 95] and in particular cooperating diagnostic agents [Bamberger & Puppe 99]. An equal challenge is how to organize large amounts of knowledge, both broad and deep, static and dynamic in a cost-effective manner. Traditional knowledge engineering techniques favor high quality approaches over standardized acquisition techniques. However, in the context of mass formalization of knowledge, we need rather simple templates, which have proven to be useful in practice.

For general software development, design patterns [Gamma et al. 95] fulfil similar needs and have become quite popular. Therefore it is promising to transfer the main ideas to knowledge based system development. In the next chapter, we give a short summary on the core ideas of design patterns and problem solving meth-
ods. Chapter 3 presents two examples for knowledge formalization patterns for heuristic classification and reports on first experience with its use. Chapter 4 concludes with a summary and an outlook.

2. Design patterns and problem solving methods

Design patterns are "practically useful solution structures for frequent design problems". Instead of solving each problem from scratch, experts reuse approved solutions. Equally important, design patterns improve the communication between developers, because it is much easier to describe a solution as using e.g. the "publisher-subscriber" pattern than to explain its object structure. Opposite to software libraries and frameworks, design pattern do not reuse code, but concepts. In general a design pattern is described by its name, a problem description, the solution and its consequences. The solution consists typically of several object classes, often called participants, and their interactions. Object oriented design patterns are often described in UML, a high level graphical modeling language.

In the field of knowledge based systems generalization and reuse have a long history: from the first shell, EMYCIN [Van Melle 81] which generalized the successful expert system MYCIN, over rule-based, frame-based or constraint-based tools and hybrids to the identification of general or problem-specific problem solving methods [Clancey 85, Marcus 88, Puppe 93]. The latter lead to elaborate description formalisms for problem solving methods (e.g. in COMMONKADS [Breuker 94, Schreiber et al. 00]) as well as easy-to-use tools kits for problem classes (e.g. D3 [Puppe 98] for diagnosis). Knowledge formalization patterns rest upon this work. Another pathway were psychologically motivated knowledge acquisition techniques like construct theory [Boose 86], but they usually try to uncover implicit knowledge from experts rather than to enable experts to formalize their explicit knowledge in an efficient way. Declarative visual programming techniques or graphical knowledge acquisition [Bamberger et al. 97] enable experts to enter their knowledge by themselves, but do not guide the knowledge acquisition process. Our experience with D3 and its graphical knowledge acquisition component showed that the experts were rapidly capable of using the editors, but they were often quite uncertain about the best way to structure their knowledge, which often made a reengineering effort necessary after some time. The problem solving methods offered by D3 as well as those found in the literature like heuristic, set-covering or case-based classification are quite coarse and leave plenty of room for different conceptualizations, or they are too simplistic like decision trees, decision tables or candidate-pruning. When experts work a high percentage of their time building knowledge based systems, the time for finding a mental model how to structure their knowledge is not critical. However, if experts develop knowledge based systems part-time, e.g. one day per week in average (even with longer breaks due to demands of their primary job), they would spend a high percentage of their time just to get into the stuff each session anew.
3. Requirements for knowledge formalization patterns

The main purpose of knowledge formalization patterns (KFPs) is to give experts a strong guideline for knowledge acquisition for non-trivial problem classes, which has proven useful in practice. In contrast to e.g. construct theory, we think, that expert need to have a mental model of the underlying problem solving method in order to formalize their knowledge adequately. Because knowledge is deeply intertwined with the inference process, KFPs should express the underlying problem solving method. The problem solving method should be not too complicated or computationally demanding, because otherwise experts might have difficulties to turn it into a mental model. In contrast to design patterns, where patterns have only one kind of addressee, namely software engineers resp. programmers, the description of algorithms is not only useful for their implementation, but primarily for guiding knowledge acquisition. Since good graphical representations help constructing mental models and also speed up the knowledge acquisition process for domain experts, KFPs should lead to them. Further on, they should scale up well for formalization of large amounts of knowledge.

The textbooks about problem solving methods/models or template knowledge models (e.g. [Stefik 95, Schreiber et al. 00, Puppe 93]) have in principal similar goals. The main difference is the granularity level. While textbooks are rather high level and do not focus on quite concrete practically proven approaches, they leave open a lot of decisions about details to the knowledge engineer. KFPs fill in this gap by adding strong guidelines, how to formalize knowledge. For example in a problem solving method, it might be sufficient to state, that rules of a certain type are needed. In KFPs the structure of the rule precondition might also be of interest, whether only one precondition, several preconditions connected by “and” or a more complicated structure is adequate, because such a recommendation might be quite valuable for an expert inexperienced in building knowledge bases. Typically, there will be one or more KFPs for one problem solving method. A major goal of KFPs is a strong guideline at the expense of generality, which is covered much better in the textbooks.

A central problem is the language in which KFPs should be expressed. We have chosen UML (in particular class and sequence diagrams), because it is widely accepted in the software and increasingly in knowledge engineering community, although it might be strange at first sight for domain experts. UML allows to represent the static knowledge structure in class diagrams (classes, relations and attributes, whose instantiations the expert has to enter in order to build a knowledge base) as well as the dynamic structure in sequence (and other dynamic) diagrams, which closely relate to class diagrams. In contrast to the CommonKADS knowledge model [Schreiber et al. 00, Chap. 5], we do without knowledge roles and inferences as independent concepts, because they add complexity and are motivated by reuse of inference knowledge, not for giving guidelines how to formalize domain knowledge. In addition, we think it to be essential to provide an example and to discuss the key elements of the pattern in plain text. In the next
chapter, we give examples for two knowledge formalization patterns for the general problem class heuristic classification.

3. Examples for knowledge formalization patterns

We describe a new pattern, heuristic decision trees, in some detail extending ordinary decision trees to deal with typical problems of heuristic knowledge. A second pattern, diagnostic scores, which is well known, is only sketched. More patterns and more details can be found in [Puppe & Martin 2000, Puppe 2000a, b].

3.1 Heuristic Decision trees

Extend and modularize decision trees for troubleshooting, consultation or similar diagnostic problem types, in order to be able to propose solutions even with partially incomplete data or uncertain knowledge.

Motivation

Decision trees are widespread for solving e.g. troubleshooting or consultation problems, because they offer a high benefit with relatively low formalization effort. However, they need categorical knowledge and assume, that the user can answer all questions. If these assumptions hold in general, but exceptions cannot be neglected, some slight extensions to heuristic decision trees are quite effective. Instead of one large decision tree, many small ones are used. They are structured according to two levels: The domain is divided in several problem areas, each of which has several decision trees (investigations). Solutions can not only be confirmed, but also suspected or ruled out. Thus, a high modularization is achieved, uncertainties are allowed and the explanation capability is enhanced by offering intermediate results.

Applicability

Use heuristic decision trees in following situations:

- General qualification of decision trees: The most important feature of problem solving is to ask the right questions in the right (economic) sequence, from whose answers the solution can be inferred directly. This is typical e.g. for many troubleshooting problems, where many diagnoses can be recognized or ruled out from characteristic singular observations.
- Extended qualification of heuristic decision trees: Some questions may be answered with “unknown” and part of the knowledge may be uncertain.
- The domain may be rather large and can be modularized.
- Single fault assumption: It is sufficient to find one solution to the problem.
**Example** (explanation see next sections): Tree classification (strongly simplified)

**Entry investigation:**
1. Native tree  ⇒ No → Other tree
   ⇒ Yes → 1.1 Has the tree needles?  ⇒ Yes → Problem area: conifer
   ⇒ No → Problem area: deciduous tree

**Problem area: conifer**
Solutions: Spruce, fir, pine, other tree.
Investigations: Seed, needles, trunk.

**Investigation seed:**
1. Cone form: ⇒ roundish cones → pine
t   ⇒ longish cones → 1.1 cone direction ⇒ hanging → spruce
t   ⇒ standing → fir

**Investigation needles:**
1. Needles length ⇒ long → pine
t   ⇒ short → 1.1 needles arrangement ⇒ more parting → fir
t   ⇒ all around → spruce

**Problem area: deciduous tree:**
Solutions: Oak, beech, maple, lime-tree, chest nut, ash, birch, other tree.
Investigations: Fruits and flowers, leaves, trunk (s. above).

---

**Structure**

Since the pattern consists of two phases, we show the structure for ease of understanding in two figures. The first figure shows the selection of a problem area and the second figure the work out of a problem area.

Fig. 1: Selection of a problem area based on the entry investigation.

---

1 A double arrow represents two rules: One as written and the other as double negation, e.g. if not roundish cones, then exclude pines. If the answer is unknown, neither rule fires and the solution remains suspected.
Fig. 2: Work out of a problem area for finding a solution².

**Participants**

**Problem area:**
- represents a delimited problem area consisting of solutions and investigations (decision trees) necessary for finding the solution.
- When a problem area is activated, it suspects all its solutions immediately and works sequentially through its investigations till a solution is confirmed.

**Solution:**
- represents a solution (e.g. a fault) for a problem area.
- contains recommendations for repairing the fault, if available.
- is dynamically rated in the categories suspected, confirmed or excluded.

**Investigation**
- represents a decision tree and is used for systematical data gathering.
- consists of a list of entry questions.

**Question**
- consists of prompt, answer alternatives resp. a range for numerical questions and annotations if necessary.

---

² Notation in UML (Unified Modeling Language): A class is represented by name, attributes and methods. Attributes referencing objects of other classes are marked as associations (the arrow indicates the main direction) or as aggregations by an arrow starting with a rhombus. Additional parameters are noted in parentheses. The # (e.g. at the attribute valuation) indicate, that the attribute is not static as most of the attributes, but will be computed at run-time.
contains references for follow-up questions, which are activated if the respective condition over the answer alternative or the range is fulfilled by its value.

contains references for problem areas resp. solutions, which are rated by the category if the respective condition over the answer alternative or the range is fulfilled by its value.

**Interactions**

![Sequence Diagram](image)

**Fig. 3: Sequence diagram**

1. On system start the entry investigation is activated.
2. The investigation activates its entry questions.
3. The user has to answer the entry questions. Depending on the answers the systems asks follow-up questions, which are treated like the entry questions and finally selects a problem area.
4. The problem area suspects all its candidate solutions (and may present them to the user as intermediate results). In addition it activates its investigations.
5. The investigations are worked out sequentially until a solution is confirmed. In case of confirmation the current investigation is finished and the remaining investigations are ignored. The current investigation activates its entry questions.
6. The user has to answer the entry questions. Depending on his answers the system asks follow-up questions, which are treated like entry questions, may change problem areas and finally assesses solutions. Solutions can become excluded, i.e. they are removed from the list of the suspected solutions (presented to the user, s. above), the suspect can be strengthened or they can become con-
firmed. When a problem area is changed, the current problem area is cancelled (with all its suspects) and another problem area is activated.

7. An investigation may occur in several problem areas. Of course, the questions are asked only once to the user.

8. As final result, the system presents the current problem area and the confirmed solution and its recommendation or the suspected solutions, which may be ordered by the number of suspects they got.

Knowledge Acquisition

We show, what kind of knowledge the expert has to enter and what kind of knowledge editors are needed.

a) Entry of objects representing problem areas and solutions in a diagnosis hierarchy, where the first level of diagnoses beneath the root object might be (for convenience) interpreted as problem areas and the second level below the problem areas as solutions with automatic suggestion, if a problem area is confirmed. Exceptions from these defaults should be allowed and represented by additional attributes of the diagnoses.

b) Entry of objects representing investigations in an investigation hierarchy. Their relations to problem areas and solutions might be entered in simple lists for each diagnosis. This is completed by a list with (usually one) entry investigation(s).

c) Entry of decision trees for all investigations with entry questions and follow-up-questions in a special decision tree editor, which allows not only confirmation of diagnoses, but also their exclusion.

d) Entry of additional non-formal multimedial information for all objects as needed.

Consequences:

• Advantages: Heuristic decision trees bank on the success of decision trees and extend their capabilities by a standardized way of modularization and the capability to deal with partially incomplete data and uncertain knowledge.

• Recommendations:
  − Large decision trees are difficult to maintain. Therefore the domain should be modularized in as much as possible specific problem areas, each of which in turn in independent investigations, so that the single decision trees for each investigation are small.
  − Investigations can reused in different problem areas. If a domain has highly variable and more stable subdomains, they should get separated in different investigations, so that the stable parts can be reused.
  − A lot of compiled knowledge is represented within the sequence of investigations and follow-up questions. Usually, the sequence should be determined by a trade-off between the costs of an investigation (or a question) and its probability to confirm a solution.
- 9 -

• Problems:
  – If solutions can be inferred only by combinations of observations instead of single observations other knowledge formalization patterns (e.g., diagnostic scores, s. below) might be better suited.
  – Faulty inputs usually result in faulty solutions. If faulty inputs must be expected, diagnostic scores with some redundancy in data gathering are the better choice.

• Extensions: The applicability of heuristic decision trees in border situations can be extended further: For example, if the single fault assumption does not hold, the termination criterion may be changed, i.e. several problem areas may be activated and all its investigations are followed, even if a confirmed solution is already found. This assumes that the multiple faults don’t mask each other. Another borderline situation is the accumulation of uncertain evidence for a solution from different observations, which can be integrated in the basic structure of heuristic decision trees to some degree, but in general can be treated better with other patterns. However, such extensions complicate the simple basic structure and therefore usually cannot compete with other better suited patterns.

Implementation

An implementation is possible from scratch, but might be more easy with using a diagnostic shell and restricting its use accordingly. An step-by-step tutorial for the shell kit D3 can be found in [Martin 2000].

Related Patterns and known applications:

The pattern is closely related to simple decision trees. It is used in particular for various kinds of troubleshooting and consultation.

Experiences

The pattern summarizes the experiences in several large service support projects (see [Puppe et al. 2000]) and in particular with a service support project with a large engineering company for building knowledge bases diagnosing letter sorting machines [Daniel et al. 2000]. The general approach in these projects was, that experts from the respective domains formalized their knowledge themselves with D3 part-time, while continuing their normal jobs most of the time. The overall task was modularized in different subdomains done by different experts according to their specializations. However, even the subdomains were large enough, that an ad-hoc approach (usually based on ordinary decision trees) resulted often in complications and restructuring of the knowledge bases. The pattern is the result of such restructuring discussions and reflects the approach of the domain experts when solving problems without computer-support. It became an accepted guide-
line for experts building new knowledge bases. Although no formal assessment was done, the project coordinator guessed, that the pattern accelerated the knowledge acquisition process significantly (about 1/3 of the time). We illustrate the use of the pattern by an example from diagnosing laser printers being connected by ethernet:

In this module (one subdomain of letter sorting machines) several aggregates exist like ethernet board, toner, paper collecting device etc. However, they are not good candidates for problem areas (PA), since it is difficult to select one of them from the primary fault description asked in the entry investigation. Therefore PA’s like “bad printing picture” or “printer doesn’t print” are better. If the printer has a self diagnostic unit, its fault messages can also be used as problem areas. Each PA has investigations (1) to select among its candidate solutions, e.g. the PA “printer doesn’t print” has investigations, that the printer is online, that the paper box isn’t empty and also investigations concerning the above mentioned aggregates. For a fault message “error 25: toner empty”, an investigation about the toner device is needed. The same investigation may also be used for other PA’s like “printer doesn’t print”.

3.2 Diagnostic Scores

Problem description: Diagnostic scores are applicable, if the goal is to find the best solution from a list of candidates, where the choice does typically not depend on one key feature like in decision trees, but on the overall picture. Further, it is necessary, that each feature is meaningful by itself and must not to be logically combined for rating solutions (this would form another pattern, e.g. heuristic decision tables).

Solution description: The basic idea of diagnostic scores is to rate all possible individual features systematically both in favor and against all solution objects. The rating should be done by a standardized scheme, e.g. with different predefined categories or with numbers or pseudo-probabilistic percentages. Since they are usually quite robust against small variations, a rather coarse scheme with few categories is often sufficient. Each solution object has an account, where it sums up the positive or negative ratings from the observed features in the case. The solution object with the highest account (and a sufficient difference to the second best) is chosen. Knowledge acquisition is done mainly with table, where the features (questions with their answer alternatives) resp. the solutions are the rows resp. the columns.

Consequences: Diagnostic scores tolerate incomplete and even some mistaken observations. In its pure form, they cannot deal with multiple diagnoses, with elaborate data gathering strategies and with low-level features. However, this can

- 10 -
be compensated with slight extensions: For diagnosing multiple solutions, one can divide the list of candidates in different groups (similar to the problem areas of the heuristic decision trees). The system first establishes groups and then refines for each group the best solution among the competitors within the group (establish-refine-strategy in an and/or-tree). Data gathering can be done with the hypothesize-and-test-strategy, where the currently best rated unconfirmed solution activates its most useful investigations in turn, until a solution is confirmed. Low-level features can be aggregated to more expressive features by means of data abstraction. The latter two mechanisms are general solutions useful for many diagnostic knowledge formalization patterns, which should be combined as necessary. The key issue is, whether an expert can easily assess the many individual ratings. Competing patterns are the Theorem of Bayes, set covering diagnosis, case-based diagnosis and heuristic decision tables.

Examples: A famous example is INTERNIST [Miller et al 82] (later renamed and commercialized as QMR), for whole internal medicine, which extends the pattern to deal with multiple diseases and with data gathering (but without data abstraction). We found the pure pattern very successful for object identification like e.g. plant classification [Ernst 96].

4. Summary and Outlook

Knowledge formalization patterns emphasize an object-oriented view on problem solving methods. By describing them as object-oriented patterns in a standard notation (UML), they strongly resemble design patterns. However, they should not only support their implementation, but also give the domain experts a mental model of the problem solving method and the knowledge representation. Thus it is intended to give the experts a guideline, what knowledge is required and how it is formalized. Therefore, in comparison to other models of problem solving methods and knowledge representations, a more detailed description is necessary, even at the risk, that slight modifications or extensions might be necessary for different domains. If KFPs are supported by a shell kit like D3 allowing graphical knowledge acquisition, they are an additional building block for domain experts to implement their knowledge bases by themselves, even if they work only part-time on this task.

The clear distinction between problem description (motivation, applicability) and solution (structure, participants, interactions, knowledge acquisition) is also consistent with the requirement to describe the assumptions [Fensel & Benjamins 98] first and the consequences section leaves plenty room for discussion. A book similar to [Gamma et al. 95] with a set of proven, easy to use knowledge patterns would probably have a wide-spread effect on their dissemination in various communities.
However, since many problem solving methods are rather complex, it might be difficult to describe them as easy-to-understand object-oriented patterns. In addition, if there are many variants of a problem solving method or if it is composed of several independent components it might be also difficult to express them in a compact manner. To answer these questions requires more experience, but even if there are complexity barriers, knowledge formalization patterns are a good way to formalize less complicated problem solving methods in a popular form.

Literature

Puppe, F.: Datenabstraktion und Testindikation [Data Abstraction and Test Indication], in [Puppe et al. 2000], Chap. 4, 2000 (b).