Model-Based Knowledge Acquisition

Angi Voß
Expert Systems Research Group
GMD
Postfach 1240
D-5205 Sankt Augustin

e-mail: avoss@gmdzi.uucp

Abstract

There are two knowledge engineering traditions, rapid prototyping and conceptual modeling which can further be distinguished into universal and shell-based approaches. Underlying the two traditions are two different hypotheses explaining the knowledge acquisition bottleneck. Model-based approaches try to tackle it by searching for a set of basic, generic problem solving methods that can be combined and instantiated for various applications. KADS is introduced as a universal conceptual modelling approach and it is compared with three prominent shell-based ones. Summarizing, knowledge engineering will be compared to software engineering which will suggest to view knowledge acquisition as a workbench of methods and techniques that are special to the former.

The paper will close with the insights I gained on the workshop about the relevance of knowledge engineering for data base engineering, and vice versa.

1 Two traditions in knowledge engineering

Knowledge Engineering denotes the discipline and the process of operationalising expertise on the computer. Having emerged from Artificial Intelligence, rapid prototyping or evolutionary programming approaches were initially prevailing. As soon as possible one started to build a small system which was successively refined until a satisfactory version had been obtained. If the prototype turned out to be inadequate it was discarded and a completely new system was built. At first, knowledge-based systems were directly programmed in Lisp, Prolog, in a rule-based formalism or an object-oriented programming language [Hayes-Roth, Waterman, Lenat 83]. Since these languages are universal, I will call such approaches universal prototyping.

With increasing experience, one began to extract the problem solving procedures from specific systems, generalized them and wrapped them into shells [Puppe 86], [Cunis, Guenter, Syska 87]. This was a significant improvement. If you were lucky enough, there was a suitable shell for your problem, so that you just had to fill in the application-specific knowledge. But time and again, the luck is being forced and shells are misused. I will use the term shell-based prototyping for approaches using such special purpose shells.

Applications growing more complex, the level of universal languages was felt to be inadequate. The same was true for shells, as they could be understood only in terms of their implementation languages. Higher level, problem specific, conceptual descriptions were wanted. Naturally, these were first obtained by abstracting the procedures employed in the shells to problem solving methods or generic conceptual models [Clancey 85], [Chandrasekaran 86], [McDermott 88], [Musen 89].
However, due to their different provenience, they differed in their vocabulary and were not adaptable or combinable.

The KADS group then proposed a scheme and a vocabulary for describing arbitrary models, i.e. problem solving methods and their applications to a particular domain, on the conceptual level [Wielinga, Breuker 86], [Breuker, Wielinga 89], [Schreiber et al. 87]. The vocabulary being semi-formal with an intuitively given semantics, these models were not directly executable. They were intended as specifications which are to be followed by a so-called design-model and an implementation. Due to their semi-formal nature, KADS models could be used to describe problem solving behaviour as encountered in the world, which even might not be operationalizable entirely.

By now, model-based knowledge engineering involves developing or finding a conceptual model. If the model is the abstraction of a shell, one has to fill in the application-specific knowledge as in shell-based prototyping, otherwise, the model can only be used as a specification. In analogy to the rapid prototyping approaches, I will use the terms shell-based and universal modelling approaches depending on whether the models are shell abstractions or can be freely defined. Figure 1 illustrates the descriptions obtained by the different approaches and figure 2 summarizes their properties.

![Diagram](image)

**Figure 1: Descriptions produced by rapid prototyping and modelling approaches**

<table>
<thead>
<tr>
<th>approach</th>
<th>operational</th>
<th>universal</th>
<th>conceptual level</th>
</tr>
</thead>
<tbody>
<tr>
<td>universal rapid prototyping</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>shell-based rapid prototyping</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>shell-based modelling</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>universal modelling</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>model-based prototyping</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

**Figure 2: Advantages and disadvantages of different types of approaches**

As its major advantage, the KADS approach allows to invent new models, or to adapt and combine existing ones. This is necessary since we are far away from a complete library of problem solving methods, and since every real-life application will usually combine several of them. However, KADS models cannot be tested, which seems to be a sine qua non for complex applications. You just...