

“With these abundant beacons, the banishment of snags, plenty of daylight in a box and ready to be turned on whenever needed, and a chart and compass to fight the fog, piloting, at a good stage of water, is now nearly as safe and simple as driving stage, and it is hardly more than three times as romantic.”

Mark Twain, *Life on the Mississippi*

Preface

Researchers in Artificial Intelligence have traditionally been classified into two categories: the “neaties” and the “scruffies”. According to the scruffies, the neaties concentrate on building elegant formal frameworks, whose properties are beautifully expressed by means of definitions, lemmas, and theorems, but which are of little or no use when tackling real-world problems. The scruffies are described (by the neaties) as those researchers who build superficially impressive systems that may perform extremely well on one particular case study, but whose properties and underlying theories are hidden in their implementation, if they exist at all.

As a life-long, non-card-carrying scruffy, I was naturally a bit suspicious when I first started collaborating with Dieter Fensel, whose work bears all the formal hallmarks of a true neaty. Even more alarming, his primary research goal was to provide sound, formal foundations to the area of knowledge-based systems, a traditional stronghold of the scruffies - one of whom had famously declared it “an art”, thus attempting to place it outside the range of the neaties (and to a large extent succeeding in doing so). However, even an unreconstructed scruffy such as myself can recognize a good neaty when he comes across one. What Dieter has managed to produce with his research on problem solving methods is what all neaties hope to do, but few achieve: a rigorous and useful theory, which can be used analytically, to explain a range of phenomena in the (real) world and synthetically, to support the development of robust and well defined artifacts.

Specifically, this book provides a theory, a formal language and a practical methodology to support the specification, use, and reuse of problem solving methods. Thus, knowledge engineering is not characterized as an art any longer, but as an engineering discipline, where artifacts are constructed out of reusable components, according to well-understood, robust development methods. The value of the framework proposed by Dieter is illustrated extensively, by showing its application to complex knowledge engineering tasks - e.g., diagnosis and design - and by applying it to the specification of libraries with both scope and depth (i.e., both usable and reusable). Another important contribution of this book is that it clarifies the similarities and the differences between knowledge-based and 'conventional' systems. The framework proposed by Dieter characterizes knowledge-based systems as a particular type of software architecture, where applications are developed by integrating generic task specifications, problem solving methods, and domain models by means of formally defined *adapters*. The latter can be used to map the terminologies used by the different system components, and also to formally introduce the assumptions on the domain knowledge required by an intelligent problem solver. This notion of assumption is central to Dieter's characterization of knowledge-based systems: these are defined as systems that make assumptions for the sake of efficiency. Thus, Dieter is able to build a continuum of assumption-making systems, ranging from “weak” search methods to “strong”, task-specific methods. As a result we can now see clearly the relationship between all these various classes of algorithms, which have traditionally been treated as distinct.

In conclusion, I believe this is the first 'real' theory of knowledge engineering to come out of several decades of research in this area. It describes the class of systems we are talking about, how to model them and how to develop them. I also believe that it is very important that this theory has come out at a time when the explosion of internet-based services is going to provide unprecedented opportunities for deploying and sharing knowledge-based services. I advise anybody who plans to be a player in this area to read this book and learn what robust knowledge engineering is about.

January 2000

Enrico Motta

Acknowledgments

First of all, I would like to thank Rudi Studer for providing me with the possibilities to continue my research after my Ph.D. in 1993. He always stimulated my research and encouraged me to relate my work to the international state of the art. He showed great patience and provided the degree of freedom necessary for a creative atmosphere.

In the last few years I have cooperated intensively with a number of colleagues who I would like to thank at this point. During my guest stay at the University of Amsterdam I profited much from cooperating with Richard Benjamins, Remco Straatman, Frank van Harmelen, Annette ten Teije, and Bob Wielinga in getting a better understanding of what problem-solving methods for knowledge-based systems are about. Putting this understanding into a software engineering flavoured framework and formulas of modal logic would not have been possible without the cooperation with Rix Groenboom and Gerard Renardel de Lavalette¹⁾ from the University of Groningen, a small town in the North of The Netherlands with highly skilled researchers. Back in Karlsruhe I worked on realizing a verification framework for knowledge-based systems that took into account the ideas I collected and developed abroad. I found an excellent tool environment KIV and the helping hand of Arno Schönege whenever the task exceeded my limited skills in mathematical proof techniques. Discussions with him and Wolfgang Reif (University of Ulm) provided many fruitful ideas on how to provide a development framework for problem-solving methods. Finally it was the cooperation with Enrico Motta from the Open University in England which allowed me to ground my ideas with an existing library of problem-solving methods dealing with real-world problems. In addition to these key players there were also a lot of colleagues from different places who provided me with helpful hints and new insights. So I would like to thank Joost Breuker (University of Amsterdam), Stefan Decker (University of Karlsruhe), Joeri Engelfriet (Vrije Universiteit Amsterdam), Pascal van Eck (Vrije Universiteit Amsterdam), Gertjan van Heijst (CIBIT, The Netherlands), Yde Venema (Vrije Universiteit Amsterdam), Marc Willems (Vrije Universiteit Amsterdam), and Zdenek Zrdahal (Open University in Milton Keynes, England) for helpful discussions and contributions. Shame on me for all the further names I do not mention. However, I should not forget Jeffrey Butler who carried out the Sisyphus-VI project of improving mein English.

Clearly, my thanks are also devoted to Estela who kept life going on watching my back in front of a computer.

¹⁾ The mathematical theory and knowledge underlying MCL (see Chapter 4) is contributed by him.

Introduction

Knowledge-based systems are computer systems that deal with complex problems by making use of knowledge.¹⁾ This knowledge may be acquired from humans or automatically derived with abductive, deductive, and inductive techniques. This knowledge is mainly represented declaratively rather than encoded using complex algorithms. This declarative representation of knowledge economizes the development and maintenance process of these systems and improves their understandability. Therefore, knowledge-based systems originally used simple and generic inference mechanisms to infer outputs for provided cases. Inference engines, like unification, forward or backward resolution, and inheritance, covered the dynamic part of deriving new information. However, human experts can exploit knowledge about the dynamics of the problem-solving *process* and such knowledge is required to enable problem-solving in practice and not only in principle. [Clancey, 1983] provided several examples where knowledge engineers implicitly encoded control knowledge by ordering production rules and premises of these rules, which together with the generic inference engine, delivered the desired dynamic behavior. Making this knowledge explicit and regarding it as an important part of the entire knowledge contained by a knowledge-based system is the rationale that underlies *problem-solving methods*. Problem-solving methods refine the generic inference engines mentioned above to allow a more direct control of the reasoning process. Problem-solving methods describe this control knowledge independent from the application domain thus enabling reuse of this strategical knowledge for different domains and applications. Finally, problem-solving methods abstract from a specific representation formalism in contrast to the general inference engines that rely on a specific representation of the knowledge.

Problem-solving methods enable the reuse of reasoning knowledge. [Clancey, 1985] reported on the analysis of a set of first generation expert systems developed to solve different tasks. Though they were realized using different representation formalisms (e.g. production rules, frames, LISP) and applied in different domains, he discovered a common problem solving behavior. Clancey was able to abstract this common behavior to a generic inference pattern called *heuristic classification*, which describes the problem-solving behavior of these systems on an abstract level (cf. [Newell, 1982]). When considering the problem-solving method *heuristic classification* in some more detail (see Fig.) we can identify the three basic inference actions *abstract*, *heuristic match*, and *refine*. Furthermore, four knowledge roles are defined: *observables*, *abstract observables*, *solution abstractions*, and *solutions*. It is important to see that such a description of a problem-solving method is given in a generic way. Thus the reuse of such a problem-solving method in different domains is made possible.

¹⁾ A good introduction to the field is provided by [Stefik, 1995]. However, the horizon of this book ends at the American borders and we would also very much like to recommend the reader the textbook on CommonKADS [Schreiber, 1999]. A survey on the state of the art can be found in [Studer et al., 1998].

In the meantime various problem-solving methods have been identified and the concept *problem-solving method* is present in a large number of current knowledge-engineering frameworks (e.g. GENERIC TASKS [Chandrasekaran, 1986]; ROLE-LIMITING METHODS [Marcus, 1988], [Puppe, 1993]; KADS [Schreiber et al., 1993] and CommonKADS [Schreiber et al., 1994]; the METHOD-TO-TASK approach [Eriksson et al., 1995]; COMPONENTS OF EXPERTISE [Steels, 1990]; GDM [Terpstra et al., 1993]; MIKE [Angele et al., 1998]). Libraries of problem-solving methods are described in [Benjamins, 1995], [Breuker & Van de Velde, 1994], [Chandrasekaran et al., 1992], [Motta & Zdrahal, 1996], and [Puppe, 1993]. In general a problem-solving method describes which reasoning steps and which types of knowledge are needed to perform a task. This description should be domain and implementation independent. Problem solving methods are used in a number of ways in these frameworks (see e.g. [Chandrasekaran et al., 1992]): as a guideline for acquiring problem-solving knowledge from an expert, as a guideline for decomposing complex tasks into subtasks, as a description of the essence of the reasoning process of the expert and knowledge-based system, as a skeletal description of the design model of the knowledge-based system, and as a means to enable flexible reasoning by selecting methods during problem solving.

Given this amount of literature the reader may ask why there should be a need for another volume on this subject. The motivation for this volume stems from an analysis of the problem-solving method *propose & revise* and its application to the configuration of a vertical transportation system (VT-domain, cf. [Marcus et al., 1988], [Schreiber & Birmingham, 1996]). This example was used by several research groups in knowledge engineering as a common case study. The solution in which we participated is reported in [Poeck et al., 1996]. In [Fensel, 1995a], we analysed our

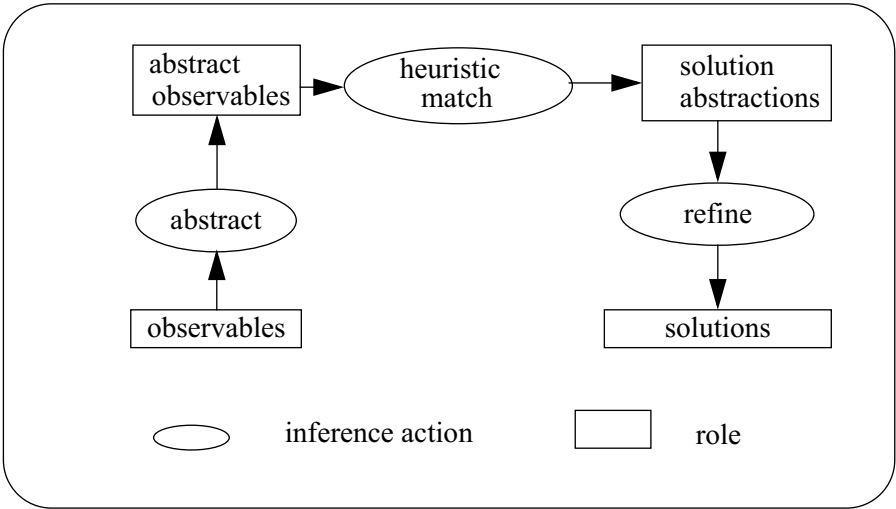


Fig. 1. The Problem-Solving Method *heuristic classification*

solution in more depth having the goal of deriving a precise and reusable specification of *propose & revise*. However, we encountered a number of difficulties. First, *propose & revise* makes a great number of very strong assumptions for solving the given configuration problem. Different variants of *propose & revise* can be identified according to the precise definition of these assumptions. These assumptions influence the definition of elementary inferences as well as the control and competence of the method. In fact, *the competence of the method can only be defined in terms of the underlying assumptions*. Some of the assumptions we encountered are rather strong and as showed by [Zdrahal & Motta, 1995], they are not always fulfilled in the VT-domain [Yost & Rothenfluh, 1996]. In consequence, we identified three aspects that require some deeper mining as a precondition to providing reusable problem-solving methods and seriously considering libraries of them.²⁾

- (i) What is the rationale and the role of the numerous assumptions we encountered when taking a closer look at *propose & revise*? Do they imply that *propose & revise* is a strange and poorly designed method or is there some more deeper rationale for introducing them?
- (ii) We felt the need to give assumptions a much more prominent role when describing a problem-solving method. They characterize the conditions under which a method can be applied to a domain and a task and are necessary for understanding most of the inferences of a method. Existing work on problem-solving methods only treated them as a side aspect. KADS-like inference and task structures [Schreiber et al., 1994] focus on an operational description of the method. However, for successful reuse of methods it seems much more important to establish a notation that describes a method in terms of assumptions and the competence as a consequence of these assumptions. The operational description is in that sense only an explanation of how the competence can be achieved by making use of the underlying assumptions.
- (iii) In [Fensel, 1995a] we identified numerous variations of *propose & revise*. None of them could be determined to be the gold standard of this method. Therefore, hardwiring all assumptions of one variant in a specific operational description with a fixed competence may lead to non-reusable problem-solving methods. Actually, this was the experiences of many shell developers who derived shells from some applications and encountered serious problems when trying to apply it to new and slightly different tasks and domains. Putting all possible variants of *propose & revise* into the library also does not look very promising given the large number of different variants. *Propose & revise* is only one method. Applying this strategy to all problem-solving methods would result in a nearly infinitely large library. In consequence, there seems to be only one reasonable strategy: (1) Identifying generic patterns from which the numerous variants can be derived by adaptation and (2) providing support for this adaptation process.

These three aspects already roughly summarize the content of this volume which—not surprisingly—consist of three sections. In the following, we will briefly sketch each section.

²⁾ Providing a library of problem-solving methods was the initial goal of the research project.

What Are Problem-Solving Methods: Reasoning Strategies that Gain Efficiency through *Assumptions*. Section I deals with the question why problem-solving methods have to introduce *assumptions* and why this is not a bug but a feature. We discuss this perspective in Chapter 1 and continue in Chapter 2 with an empirical survey of assumptions used in model-based diagnosis to ground our argument.

How Can Problem-Solving Methods Be Described: With a Software Architecture, MCL, and KIV. The structured development of problem-solving methods requires a structured framework to describe them. This is the subject of Section II. In Chapter 3, we provide a *software architecture* for describing knowledge-based systems focusing on the problem-solving method and its related parts. KADS-like operational specifications are supplemented by describing the competence and assumptions of a method and using adapters to relate them to the task and domain. “A specification can provide a way of making explicit those assumptions which are otherwise hidden consequences of an algorithm.” [Jones, 1990] Chapter 4 investigates the requirements of a logical framework for formalizing problem-solving methods and presents the two logics *MLPM* [Fensel & Groenboom, 1996] and *MCL* [Fensel et al., 1998 (c)] which both integrate the specification of dynamics in a declarative framework. Chapter 5 provides a framework for verifying such architectural specifications of knowledge-based systems. It is based on the Karlsruhe Interactive Verifier (KIV) [Reif, 1995] which shifts verification from a task that can be done in principle to a task that can be done in practice. KIV can be used to establish the competence of a problem-solving method given some assumptions or to find the assumptions that are required to achieve such a competence. The latter aspect is discussed in Chapter 6.1 as *inverse verification*.

How Can Problem-Solving Methods Be Developed and Reused: By *Stapling Adapters* and Hunting for Assumptions with *Inverse Verification*. Section III provides the means for the structured development and reuse of problem-solving methods. Chapter 6 discusses the context dependency of knowledge which arises as a problem when trying to reuse it. We provide two methods that deal with this problem. We discuss a method called *inverse verification* to support the explication of context and *adapters* to support the adaptation to a new context (i.e., domain or task). Chapter 7 uses this principle to describe a library of methods for solving design problems. This methods library was developed at the Knowledge Media Institut (cf. [Motta & Zdrahal, 1996], [Motta, 1999]) and used in several applications³⁾. We identify a small number of key patterns and derive all possible variants of problem-solving methods through a navigation process in a three dimensional space. The organizational principles for this library provide a solution for dealing with all the different variants of problem-solving methods: a small number of generic patterns and support in their adaptation.

³⁾ Office allocation problem, elevator design, sliding bearing design, problems of simple mechanics, initial vehicle (truck) design, design and selection of casting technologies, and sheet metal forming technology for manufacturing mechanical parts [Motta, 1999].



<http://www.springer.com/978-3-540-67816-8>

Problem-Solving Methods

Understanding, Description, Development, and Reuse

Fensel, D.

2000, XII, 160 p., Softcover

ISBN: 978-3-540-67816-8