Dimensions to Analyze Applications[1]

Yolanda Gil
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
gil@isi.edu

Marc Linster
Workplace Integration Technologies
Digital Equipment Corporation
200 Forest Road
Marlboro, MA 01752
linster@guess.enet.dec.com

Abstract

Applications should be an invaluable experimental source of information and challenges for AI research. The real world always stretches the limitations of our AI systems, pointing toward new research themes in many areas. In the process of implementing an application, the designer continually makes choices based on (1) the baseline architecture used to implement the application, (2) the characteristics of the problem itself, or (3) arbitrary decisions and assumptions. All these decisions are intertwined in the resulting application, and as a result, it is not easy to abstract a description of the functionality provided by the architecture itself. At the same time, we would like to base our science on real-world applications that are subject to controllable experiments whose parameters can be modified to obtain experimental results of our programs' behavior. However, real applications rarely facilitate this task. Although the AI community has developed formalisms to describe AI architectures, we are still lacking a formal language to describe tasks and problems that provides a good qualitative understanding of AI applications. We argue that this is a major deficiency that stops feedback from applications to research. This work is an effort towards descriptions of applications in terms that are useful 1) to extract conclusions from particular implementations, 2) to facilitate cross-comparisons among different architectures applied to the same problem, and 3) to facilitate comparisons among different tasks. We analyze the Sisyphus experience, and we propose a set of dimensions to describe applications that distinghish between descriptions of the properties of the architecture, the type of problem, and the data sets. We show how these dimensions can be used to produce useful distinctions in the context of the first Sisyphus task, an office assignment problem. Our hope is that the same dimensions will be useful to other researchers in describing, characterizing, and producing qualitative evaluations of their applications, as well as a useful point of comparison for future Sisyphus efforts.

[1]In Proceedings of the Ninth Knowledge Acquisition for Knowledge-Based Systems Workshop, February 26-March 3, 1995. Banff, Alberta, Canada.

1 Introduction

Knowledge-based systems research uses real-world problems to illustrate its advances and accomplishments. Real-world problems stretch the limits of architectures and provide new challenges. However, when solving real-world problems the interactions between the problem statement, the problem solution and the architectural principles employed to obtain a solution become blurred. As Figure 1 illustrates, the world is a complex place and in the process of implementing an application, the designer continually makes choices based on (1) the baseline architecture used to implement the application, (2) the characteristics of the problem itself, or (3) arbitrary decisions to simplify the problem. This means that it is difficult to assess the potential and the apropriateness of a problem-solving architecture in itself when it is illustrated with a real-world example. Moreover, it becomes difficult to assess the research contribution of a proposed architecture, that is, one cannot assess whether it was the architecture that allowed for this solution or if the solution became possible due to the designer's ability to rephrase the problem into the architecture's terminology and its system of concepts. (Most problem solvers being Turing-Machine equivalent an ingenious designer can solve any (solvable) problem with any architecture.) Such difficulties do not exist when a method is illustrated with toy problems, as their simple description makes their interaction with the architecture obvious. Frequently, one well-defined feature of the toy problem is all that is being addressed. However, in real-world problems, it is hard to define the features of a problem before starting to solve it.

The Sisyphus effort — initiated at EKAW'90 in Amsterdam [Wielinga *et al.*, 1990] — invited research groups to provide their solutions to reasonably complex problems that bridge the gap between real-world tasks and toy problems. The Sisyphus problems (e.g., office assignment, sailboat rigging, elevator configuration) are real enough so that the mapping of their features into the concepts of a solution architecture is ambiguous and allows the individual designer to conceptualize the problem in an architecture-specific manner. For example, the office-assignment problem states that smokers and non-smokers should not share offices. This statement can be abstracted into rules, constraints, action classes, or schemas [Linster, 1994]. At the same time, the Sisyphus problems are small enough for easy documentation, thus allowing many research groups to participate in the experiments.

First analyses of Sisyphus contributions [Linster, 1993b] have shown that it is possible and fruitful to compare the use of different approaches and architectures to solving the same problem. Karbach, Linster, and Voß [Karbach et al, 1990] analyzed operational knowledge-based problem solvers. They focussed on how different approaches model and represent problem-solving capabilities and the knowledge required to implement those capabilities. Besides looking at the use of terms like *task* and *modeling level*, they focus on a unifying description of the problem-solving method. The definition by McDermott [McDermott, 1988] (i.e., that a *a problem solving method provides a means of identifying, at each step, candidate actions. It provides one or more mechanisms for selecting among candidate actions and ensures that the selected action is implemented*) is extended to account for more complex

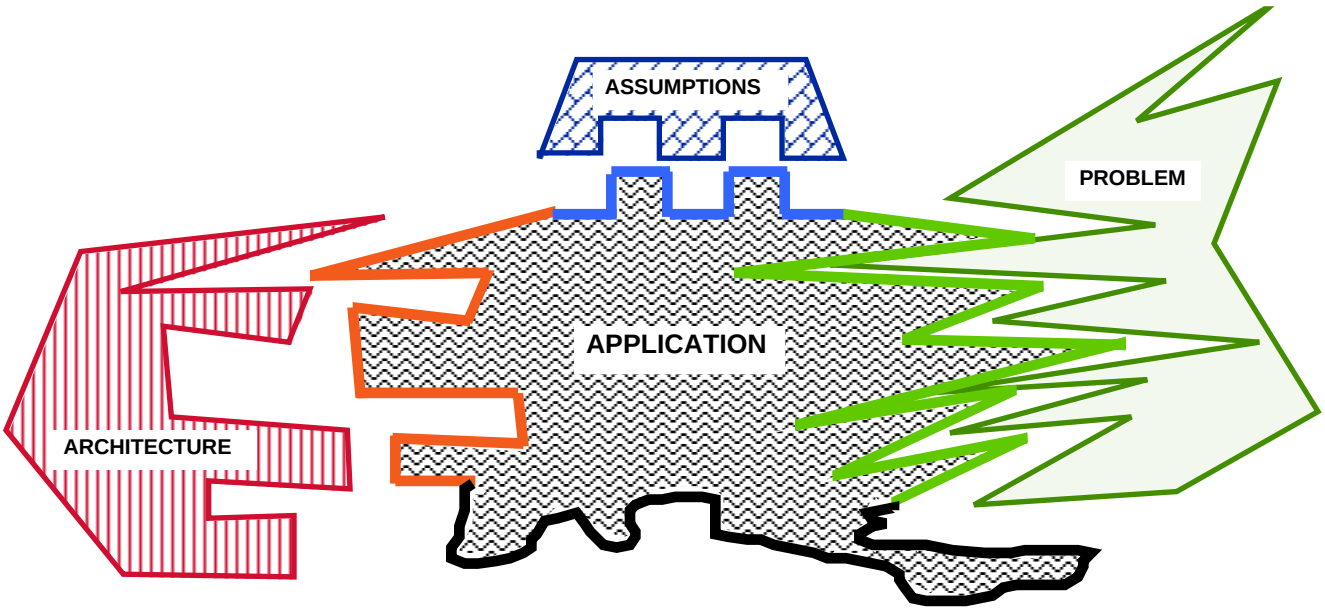

Figure 1: Applications are shaped by the architecture used, the characteristics of problems they automate, and assumptions about the problem made by the designers.

models of methods, such as they are found in KADS or Generic Tasks. Discussions at EKAW'94 showed that these analyses helped gain insight. However, we believe that the analyses to date are very focused on knowledge engineering aspects and neglect important characteristics of AI architectures, such as *search space*. This motivated us to work towards a more comprehensive and systematic framework for comparison.

The paper runs as follows. First, we summarize the dimensions for comparing systems in previous work. Then we propose a set of features to characterize problem-solving artifacts, to describe the interactions between design decisions and solution characteristics, and to compare different solutions to the same problem. We focus our discussion on the Sisyphus office assignment problem, because 1) it is the most manageable in size, 2) it is the one that has been implemented by most groups to date, and 3) it has already produced some comparative studies (see [Linster, 1994] for a description of the problem and various alternative solutions). Finally, we propose some questions for discussion at the workshop that could provide further insight in solutions to Sisyphus problems.

2 Previous Work

Linster [Linster, 1993b] compares several Sisyphus solutions along different dimensions to address the following questions:

1. What are the building blocks of the model?
2. Which components are generic, which are non-reusable?
3. What is the purpose of the tool/methodology (e.g., code generation, visualization, conceptualization, elicitation, knowledge management)?
4. Where in the development cycle is the tool or methodology most useful?
5. Who is the user of the tool or methodology?

These dimensions reflect an engineering point of view on knowledge modeling, instead of a cognition-oriented one. For example the analysis does not consider the adequacy of the models nor the efficiency of the approaches in capturing the *right kind* of knowledge (see [Burton *et al.*, 1990] for such a study). The analysis does not look at the methodological aspects of the approaches either, that is, it does not study how much guidance the approaches provide for the practitioner developing a model (see [Allemang & Rothenfluh, 1992] for such a study).

What are the building blocks used to model knowledge?

Building blocks were defined as discrete and identifiable constructs that the framework provides to describe the knowledge that goes into an application. Note that according to this definition interpreters or other predefined and invariable elements of a system are not considered building blocks. Rules, objects, knowledge sources, agents, classes, are examples of building blocks. The focus was on declarative elements of the knowledge representation. This appears to be the only clear boundary between general-purpose programming languages, such as Lisp or HyperTalk and other means of representation that are more commonly referred to as knowledge representation. We will distinguish three kinds of representational primitives: (1) domain-knowledge representation facilities; (2) method elements, that is the elementary generic problem-solving building blocks, their aggregation principles and the control description that combines building blocks into problem-solving methods; and (3) primitives to connect method definitions with domain knowledge.

What is the purpose of the tool/methodology?

Four categories of usage were employed to discuss the different approaches: *knowledge conceptualization*, *visualization*, *implementation*, and *knowledge management*. A tool helps in the conceptualization phase if it allows to represent observations and interpretations of observations previous to their formalisation, and if it supports the user in the transition from informal to formal knowledge. We refer to a tool as a knowledge visualisation tool if its interface emphasizes graphical communication of knowledge. For a tool to be categorized as

an implementation tool it must provide directly operational formalisms or strong methodological support to transform pre-operational knowledge into an operational representation. Simply attaching an editor for Lisp code doesn't do it. Moreover, delivering running systems must be the intention of the tool developers, as opposed to tools built to deliver executable specifications or feasability studies. Knowledge management refers to repository, dictionary, and browsing capabilities for the tool's knowledge representation primitives. An environment provides active knowledge acquisition support if it derives guidance for the ongoing acquisition from the current contents of its knowledge base.

Where in the development cycle is the tool most useful?

The study used a description of system development as a cyclic process consisting of a set of distinct activities. These activities are organized in a logical sequence, not to be confused with a waterfall-oriented series of phases; we see them as being the central activities of a spiral development cycle, so that all phases can be repeated and previous results can be refined and even undone if indicated by the ongoing elaboration process. A more detailed description of this view on the development cycle is given in [Linster, 1993a].

1. **Initial knowledge acquisition.** The knowledge engineer goes through initial interviews with the domain expert, records first protocols, and if possible she uses techniques such as the knowledge acquisition grid [LaFrance, 1987] to obtain initial structures and to get a first overview of the application task.

2. **Data interpretation and knowledge structuring.** The knowledge engineer identifies recurring and potentially more abstract structures in the domain. These structures can be of different kinds.

   (a) **Identification of domain structures.** The knowledge engineer develops a structured terminology for the domain, for example she can define a T-box in KL-ONE–like approaches [Brachman & Schmolze, 1985] or a set of classes in an object-oriented approach.

   (b) **Identification of inferences and roles that knowledge elements play in the problem-solving process.** The knowledge engineer defines the actions, goals, and decision criteria to give an abstract (possibly knowledge level) description of the problem-solver.

   (c) **Identification of other structures,** such as, task sharing, task decomposition, data flow, or modality.

   (d) **Integration and mapping of the different structures into a coherent model of the task.** The different knowledge structures, identified in the previous phases, must be merged into a coherent model. This phase is most important, as it represents the creative interaction between the different points of view represented by the different knowledge structures.

3. **Acquisition of the detailed knowledge in the framework defined by the task model.** The model of the task provides structures that are now stuffed with detailed knowledge about the application.

4. **Knowledge implementation.** The task model is transformed into an operational system[2].

5. **Testing and debugging.**

6. **Knowledge maintenance after system delivery.** We distinguish two types of knowledge maintenance:

   (a) **Maintenance of the structures that constitute the framework of the task model.** If structures are modified, then this changes the task model. Such modifications require re-engineering of the task model, and if necessary, restructuring of the detailed knowledge.

   (b) **Maintenance of the detailed knowledge in the structures of the task model.** Within the framework of the task model, maintenance of the detailed knowledge is similar to the acquisition of the detailed knowledge.

   This distinction depends strongly on the implementation phase. It can only be drawn if the implementation maintains the distinction between structures of the task model and the detailed knowledge.

Who is the user?

The study differentiated between four classes of users: (1) domain experts, that is, users with a lot of knowledge of the area that the tool will be used in, but without systems analysis skills and little or no programming knowledge; (2) knowledge engineers, that is, people with good systems analysis and programming skills; (3) analysts, such as workplace analysts, who do not necessarily have programming knowledge; and (4) teams consisting of domain experts working under the guidance of knowledge engineers or analysts.

2.1 Some conclusions

Many different approaches were used to solve the Sisyphus tasks, including configuration design, situated classification, constraint satisfaction, and genetic algorithms [Linster, 1994]. After categorizing the Sisyphus contributions in this framework, we drew conclusions such as the following ones:

[2]Obviously this phase is obsolete if the task model is defined using operational primitives.

If one wants to support the creative aspects of model building in the early phases of knowledge acquisition, then having independent languages for method and domain modeling appears to be crucial.

If one wants support for initial knowledge acquisition and bottom-up structure development, then one is bound to get little help for the acquisition and maintenance of the detailed knowledge.

If one wants to have good support for the implementation phase, the acquisition and maintenance of the detailed knowledge, then one should use a shell.

It is important to keep in mind that these conclusions are based on a first analysis of a limited number of datapoints. Moreover, the dimensions used for analysis and the positioning of the datapoints along the dimensions are not based on a generally accepted method of evaluation. We render these conclusions to illustrate the potential that an agreed-upon system of dimensions and an agreed-upon assessment method have. Right now, the conclusions are only of interest in the framework used in [Linster, 1993b] to evaluate the Office-Allocation contributions.

3 Dimensions to Compare Knowledge-Based Approaches and Architectures

Linster [Linster, 1993b] takes a very knowledge-engineering oriented point of view, looking at systems to find out how they support the knowledge engineer in his/her tasks. This point of view does not look at characteristics of the problem to be addressed (it focusses on the Sisyphus office allocation task), nor does it consider the application system that results from the knowedge engineering process. To generalize Linster's work, it appears necessary to include other — orthogonal— points of view. In order to produce useful descriptions of applications, it is important to make distinctions between 1) the architecture, a domain-independent set of tools designed a priori by the knowledge engineer, 2) the problem that needs to be automated, and 3) the application system, resulting from applying the architecture to the problem.

In a concrete problem-solving situation, the architecture is the given and there are varying degrees of interaction between architecture, problem characterisation, and the application system as the characterisation of the problem is being influenced by a user's familiarity with the architecture and its underlying metaphors.

In the following list of characterisations we do not aim at being comprehensive, but rather at presenting what we believe are useful factors to analyze applications and to describe architectures, problems, and applications.

3.1 Describing an Architecture

1. Scope of Architecture
   - task coverage — types of tasks that the architecture can be used for
   - capabilities — types of reasoning that the system is able to do: uncertainty, default reasoning, learning, etc.
2. Representations
   - state descriptions — what formalism is used to represent static descriptions of objects? is there a representation of time?
   - procedural knowledge — can the system represent deduction rules? can action changes be represented?
   - decisions — can preferences be expressed explicitely? can constraints be represented?
3. Finding solutions
   - search strategy — how does the system find solutions?
   - search control — are there domain-independent search-control mechanisms? how is domain-dependent search-control knowledge formulated?
   - abstraction — is there an abstraction mechanism in the system?
4. Building blocks used to construe the problem and phrase the solution
   - generic building blocks
   - grain size of building blocks
   - knowledge-level vs. symbol-level building blocks
   - formal vs. operational vs. informal
   - process vs. structure oriented building blocks
5. Activities supported in the development cycle of an application
   - elicitation
   - knowledge structuring
   - acquisition of detailed knowledge
   - knowledge implementation
   - testing
   - maintenance

6. Adaptability — can the system evolve as its environment requires?
    - execution — can the system handle unexpected events during plan execution?
    - learning — does the system adapt autonomously, or does it require manual adaptation?
    - factual knowledge — how can objects and facts be added/deleted/updated? can the same factual knowledge be used for a different task?
    - task knowledge — can the definition of the task be changed? can the goal be changed and in which ways?
7. Prototyping
    - fast prototyping — how much effort is required to produce a working prototype?
    - prototype complexity — how much complexity of representation is required for simple problems?
8. User-system communication
    - User intervention — Does the system share the task with the user? This is a dimension spawned by the following extreme points: (1) the user keys in an instance of the problem statement; the system comes back with a ready-made solution; and (2) the user and the system cooperate in solving problems.
    - inspection — can the user understand how the system is solving a problem?
    - explanation — can the system justify its behavior?

3.2 Describing a Problem

1. Entities
    - types of objects
    - properties of individual objects
    - properties of classes of objects
    - properties that relate objects
2. Actions and Change
    - temporal considerations — reasoning about events and durations
    - state transformations — transitions and action models
3. Restrictions — can be preferences or hard-set restrictions
    - restrictions on objects

- restrictions on actions

4. Solutions
   - are there solutions to all the instances of the problem?
   - criteria satisfaction — does the problem require satisficing solutions or are partially satisficing solutions sufficient?
   - solution quality — is there a notion of better quality solutions? can the user specify a range for the quality of the resulting solution?

A problem does not necessarily need to be modelled in these terms. Our intention in enumerating the above set is rather to provide a list of issues that need to be addressed in modelling the problem.

Data sets instantiate the problem statement in all the respects listed above. Data sets may consist of a description of all the objects of each type and their properties.

3.3 Describing an Application System

1. Representation — the knowledge necessary for solving problems
   - object types — how many different types of objects?
   - constraints — are there restrictions in the way the objects interact?
   - relations — are there relations among objects?
   - actions — what changes and transitions are possible?
   - time — does the system do some type of temporal reasoning?
2. Search Space — is the system exploring a search space in finding a solution?
   - branching factor — what is the branching factor?
   - search depth — what is the depth of the search tree?
   - goal interactions — how can the interactions between goals be characterized? (independent, serializable vs non-serializable, positive vs negative)
   - backtracking — does the system ever need to backtrack?
   - solution density — what is the average solution density when comparing the size of the search tree to the number of solutions?
3. Efficiency — how does the system address the combinatorial complexity of the problem?
   - quantitative measures — empirical results on the system efficiency

- qualitative measures — a description of the mechanisms used to make the system efficient

4. Delivery — what happens when the system is put to use?

   - users — who are the users of the application? does the system achieve a useful task according to users?
   - environment — how does it fit in the overall work environment?

The description of the application can also include a more detailed account of the dimensions for describing an architecture.

4 Describing the Sisyphus Office Assignment Problem

The Sisyphus office assignment problem statement [Linster, 1994] asks for the design of a running system that assigns members of an imaginary research group to their offices. The problem statement consists of a description of offices, members of the research group, the roles they play in the group (such as, manager, senior researcher, etc.), an annotated transcription of a thinking-aloud protocol of a wizard who routinely solves this kind of task, and a list of constraints and preferences for the solution (such as, management and secretaries should be assigned close offices, senior researchers should be assigned single offices, smokers and non-smokers should not share offices, etc.). A second version of the problem statement is a variant of the first data set but with one more smoker. This small change produced a data set that had no satisficing solution.

In our set of dimensions we would characterize this problem as follows:

1. Objects

   - types of objects: rooms, people, projects, job types (group head, project head, manager, secretary, project member)
   - properties of classes of objects: person x is a smoker or a non-smoker, person x is a hacker or a staff member, room x is double or single size
   - properties that relate objects: person x has job of type y, person x works with person y, person x works with person y.
   - properties of individual objects: the actual people, projects, and rooms (not included here for lack of space).

2. Actions and Change

- temporal considerations — reasoning about events and durations: temporal considerations do not apply — except for the sequence of activities when assigning people to offices. The sequence of assignements (i.e., who is assigned first) may be an important factor in the user-interactioon aspects of the system.
- state transformations — transitions and action models: At first sight, a state transition model may apply; states being partial assignments (who is in which office, and who is not assigned yet) and transitions/actions being the assignment of people to offices.

3. Restrictions and preferences

   - restrictions
     (a) single rooms accomodate one person, double rooms accomodate two.
     (b) there is only one group head
     (c) there is only one manager
     (d) there is only a project head per project
   - preferences
     (a) offices of the group head and staff should be close.
     (b) group heads should be given a double-size office.
     (c) project heads should be close to the group head.
     (d) twin offices should be shared by people in different projects.
     (e) researchers are not eligible for single offices.
     (f) a non-smoker cannot share a room with a smoker.

4. Solutions

   - are there solutions to all the instances of the problem? Not all problems can be solved — at least not without relaxing some of the criteria. For example, the second problem statement — characterized through an odd number of smoking junior researchers — can only be solved by relaxing some constraints, for example, by putting a junior researcher into a single office, or by assigning a smoking senior researcher to a twin office together with a smoking junior researcher.
   - criteria satisfaction — If finding a solution requires violating some constraints, there was no specification of how the human expert would solve the problem.
   - solution quality — Solutions that satisfied all the constraints were presumable preferred to solutions that violated some constraints. No other characterisation of the relative quality of solutions was given in the problem statement.

5 Discussion

The data sets provided often determine whether a constraint in the problem definition is making the problem hard to solve. The data set provided made the problem relatively simple. For example, there were a lot of double-size and single-size offices compared to how many people needed to be assigned. There were only 4 smokers. There were 8 projects, each of 1-3 people. These data made the constraints relatively easy to satisfy. In this case, the fact that the number of smokers is even makes this data set simpler to solve with regard to the non-smoking constraint, compared to data sets that include many smokers. And in the extreme, if there are no smokers at all then the constraint becomes nonrelevant. Similarly, if there are many more rooms than people, then this constraint does not really affect the complexity of the problem.

The second statement of the Sisyphus problem had no satisficing solution, which made some architectures fail to return any solution at all. This sort of exercise is very useful to understand the strengths and weaknesses of an architecture. But it is nonetheless a small variation along one single direction.

Despite the wide variation in the approaches taken by the different Sisyphus participants, the fact is that each one could successfully implement the problem statements published to date. The question that we should pose is not whether we can implement an application using a certain kind of approach, since this seems to have a clearly positive answer. Rather, we should ask whether our implementations can provide solutions to complex data sets, whether they can satisfy interacting restrictions, and whether they can support the additional functionalities that real applications demand. This last question is central to evaluating the knowledge acquisition tools that we build for our systems, which is the ultimate goal of the Sisyphus participants. Can our systems be corrected if the designers had any misconceptions or misunderstandings about the task they were automating? Can our systems support additional functionalities that applications may require, such as explanation, adaptation, and scaling up? How could we use our knowledge acquisition tools to modify the current implementations of the Sisyphus problem to accomodate new requirements that our human expert may get in his job in subsequent months? Could we update our systems to accomodate requirements such as:

- More rooms, more people, more projects (this would test scaling up issues)
- More (and more problematic) smokers (as in second Sisyphus task, but more than one)
- More managers
- More people in each project
- Absolutely no assignments with smoker/non-smoker
- Hackers cannot share rooms with non-hackers

- New types of workers (e.g., visiting researchers)
- Required to respect officemate assignments as they are currently stand in the building before the group moves to the castle
- The user has strong preferences on which constraints to violate

We propose as the next step to analyze the Sisyphus contributions by showing how each approach would support users in incorporating new requirements (such as the ones listed here). This would provide additional insight in comparing alternative approaches because it would demonstrate their strengths and weaknesses as knowledge acquisition tools. We believe that the set of dimensions that we propose in this paper can make useful distinctions regarding this kind of analysis. The task would be to identify what specific design decisions in terms of these dimensions facilitate knowledge acquisition.

References

[Allemang & Rothenfluh, 1992] Allemang, D., and Rothenfluh, T. 1992. Acquiring knowledge of knowledge acquisition: A self-study of Generic Tasks. In Wetter, T.; Althoff, K.-D.; Boose, J.; Gaines, B.; Linster, M.; and Schmalhofer, F., eds., *Current developments in knowledge acquisition–EKAW92*, volume 599 of *LNAI*, 353 – 372. Heidelberg: GI, ECCAI.

[Brachman & Schmolze, 1985] Brachman, R., and Schmolze, G. 1985. An overwiev of the KL-ONE knowledge representation system. *Cognitive Science* 9(11):216–260.

[Burton *et al.*, 1990] Burton, A. M.; Shadbolt, N. R.; Rugg, G.; and Hedgecock, A. 1990. The efficacy of knowledge elicitation techniques: A comparison across domains and levels of expertise. *Knowledge Acquisition* 2(2):167 – 178.

[Karbach et al, 1990] Karbach, W., Linster, M. and Voß, A. 1990. Models, methods, roles and tasks: Many labels - one idea?. *Knowledge Acquisition* 2(4):279 – 300.

[Karbach & Voß, 1991] Karbach, W., and Voß, A. 1991. Glueing together small solutions: An office planning model in MODEL-K. In Linster, M., ed., *Sisyphus'91: Models of problem solving*, volume 663 of *Technical Report*. St. Augustin: GMD.

[LaFrance, 1987] LaFrance, M. 1987. The knowledge acquisition grid: A method for training knowledge engineers. *Int. Journal of Man-Machine Studies* 26:245–255.

[Linster, 1993a] Linster, M. 1993a. Explicit and operational models as a basis for second generation knowledge-acquisition tools. In David, J.-M.; Krivine, J.-P.; and Simmons, R., eds., *Second generation expert systems*. Springer Verlag. 477 – 506.

[Linster, 1993b] Linster, M. 1993b. A review of Sisyphus 91 & 92: Models of problem-solving knowledge. In Aussenac, N.; Boy, G.; Gaines, B.; Linster, M.; Ganascia, J.-G.; and Kodratoff, Y., eds., *Knowledge Acquisition for Knowledge-Based Systems*, volume 723. Toulouse: Springer Verlag.

[Linster, 1994] Linster, M. 1994. Special Issue on Sisyphus: Models of Problem Solving. *Int. Journal of Human-Computer Studies* M. Linster (Ed.) 40 (2).

[Marcus & McDermott, 1989] Marcus, S., and McDermott, J. 1989. SALT: A knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence* 39(1):1–37.

[Sandra Marcus, 1988] Sandra Marcus, Jeffrey Stout, J. M. 1988. VT: An expert elevator designer that uses knowledge-based backtracking. *AI Magazine* 9(1).

[McDermott, 1988] McDermott, J. 1988. Preliminary Steps Towards a Taxonomy of Problem-Solving Methods. In S. Marcus ed., *Automating Knowledge Acquisition for Expert Systems*. Boston: Kluwer Academic Publishers.

[Newell, 1982] Newell, A. 1982. The knowledge level. *Artificial Intelligence* 18:87 – 127.

[Stefik *et al.*, 1982] Stefik, M.; Aikins, J.; Balzer, R.; Benoit, J.; Birnbaum, L.; Hayes-Roth, F.; and Sacerdoti, E. 1982. The organization of expert systems: A tutorial. *Artificial Intelligence* 18(2).

[Wielinga *et al.*, 1990] Wielinga, B.; Boose, J.; Gaines, B.; Schreiber, G.; and Someren, M. V., eds. 1990. *Current Trends in Knowledge Acquisition; Proceedings of EKAW90*, Amsterdam: ECCAI.

[Yost, 1993] Yost, G. R. 1993. Sisyphus 1993: Configuring elevator systems. Technical report, Workplace Integration Technologies Group, Digital Equipment Corporation, 111 Locke Drive (LM02/K11), Marlboro, MA 01752. Unpublished.