BUILDING NON-BRITTLE KNOWLEDGE-ACQUISITION TOOLS

Jay T. Runkel
William P. Birmingham

Electrical Engineering and Computer Science Department
University of Michigan
Ann Arbor, MI
jayr@eecs.umich.edu

Abstract

Existing model-based knowledge-acquisition tools can acquire large knowledge bases and update these knowledge bases as knowledge changes. These tools, however, are brittle. They can only be used to acquire knowledge for a particular problem solver performing a specific task and they are not easily adapted to new problem solvers. Brittleness limits the effectiveness of these tools because the dynamic nature of knowledge systems make modifications both necessary and frequent. This paper presents a model of knowledge systems that reduces brittleness by separating acquisition techniques for search-control knowledge from other types of knowledge, by driving knowledge acquisition from properties of a knowledge-level description of the task instead of the problem solver, and by using ontologies to reuse knowledge bases.

Introduction

Many knowledge systems contain knowledge-acquisition (KA) tools that help knowledge engineers and domain experts build and maintain the system's knowledge base. These tools manage the vast amount of complex and interrelated knowledge necessary to build a knowledge system and automate the process of operationalizing (representing the knowledge in a form that can be used by a problem solver) this knowledge. They exploit a collection of acquisition techniques, which consist of a user interface, a prescribed procedure for using this interface, and a method for operationalizing the acquired knowledge.

One of the most powerful classes of KA tools, called *model-based* KA tools (Birmingham & Klinker 1994), guide acquisition using the problem solver's problem-solving method (McDermott 1988) (PSM), which defines the sequence of problem-solving steps used to find a solution. Since the PSM is an executable procedure, it assumes that the knowledge it uses has certain properties. These properties are the types of knowledge necessary to perform problem solving, called *knowledge structures*, the role of each knowledge structure (*i.e.*, how knowledge will be used during problem solving), and the operationalized form of knowledge structures (Birmingham & Klinker 1994; Marcus 1988; McDermott 1988).

By exploiting these properties, model-based tools can *direct* the KA process. This direction is achieved by using interfaces customized to acquire knowledge in the form required by the PSM. The customized interfaces combined with the use of domain-specific terminology results in an efficient knowledge-transfer process. Directed dialog helps the tool's user understand exactly what knowledge the tool is asking for, how it relates to previously acquired knowledge, and how the knowledge will be used during problem solving (Marcus 1988; McDermott 1988). This ability to convey the purpose and context of the desired knowledge dramatically reduces the effort required to build a knowledge base (Birmingham & Klinker 1994; Marcus 1988; Musen 1989).

Model-based tools, though powerful, are brittle. Brittleness manifests itself in two ways:

1. Model-based tools cannot be easily adapted to acquire knowledge for new PSMs or tasks.
2. The knowledge bases typically created by these tools cannot be reused by PSMs performing similar tasks.

Brittleness significantly limits a tool's usefulness.

There are two causes of brittleness in model-based tools:

1. *Acquisition techniques are tailored to a particular PSM.* Model-based KA tools gain their acquisition power by tailoring their acquisition techniques to a particular PSM. Such close coupling, while lending power, limits the flexibility of the tools
2. *Failure to separate acquisition techniques for search-control knowledge.* Search-control knowledge, which heuristics makes problem-solving more efficient, is mostly idiosyncratic to a PSM, particularly to the way in which it is encoded. Non-search-control knowledge is mostly constant in a domain across similar types of tasks. Most KA tools fail to separate the acquisition techniques for search-control knowledge from those for other knowledge types making it difficult to update KA tools as PSMs change (with changing tasks). KA tools that acquire search-control knowledge are specific to a PSM, and therefore are brittle.

The key to eliminating the brittleness of model-based tools without sacrificing any acquisition power is to model knowledge systems at the knowledge level (Newell 1981) (KL). The KL describes the task performed by the knowledge system independently of the PSM used to solve it, and can therefore be made free of control knowledge. In addition, the KL defines many of the properties needed for model-based tools: the knowledge structures to be acquired, the relationships among these structures, and the roles the structures play during problem solving. This facilitates the development of model-based KA tools that are not PSM specific. In addition, the knowledge structures identified at the KL form an ontology of the knowledge used to perform a task. Recent experiments with the language Ontolingua (Gruber 1993) have shown that ontologies can be used to facilitate the sharing of knowledge among knowledge systems by providing an intermediate language that can be used by all of them.

The model-based acquisition techniques and ontologies identified at the knowledge level are highly flexible and reusable because they do not contain any search-control knowledge. Since knowledge structures are identified at the KL, they are highly reusable because all problem solvers performing the task use this knowledge. Search-control knowledge, on the other hand, represents the heuristics used by a particular PSM, and is therefore less reusable. Brittleness is avoided by keeping the acquisition techniques and ontology elements for search-control knowledge separate from the techniques and elements identified at the KL. This allows KA tools and knowledge to be easily updated and reused as PSMs and tasks are changed.

In addition, brittleness can be reduced by developing KA tools with a modular architecture, where the acquisition techniques used to acquire each knowledge structure are represented in separate modules. This approach, which is similar to the one take by Protege II (Puerta et al. 1992) and Krest (Steels 1992), simplifies the process of modifying the KA tool whenever changes to the PSM or task are made. An acquisition procedure can be easily replaced with one that is more appropriate.

We have developed a system, called the Domain-Independent Design System (DIDS) (Balkany, Birmingham, & Runkel 1993) that uses these brittleness reduction ideas to generate non-brittle knowledge systems. This paper describes the model of knowledge systems used by DIDS, called the DIDS model. This model is distinguished from other similar models of knowledge systems, such as KADS (Wielinga, Schreiber, & Breuker 1992), by its focus on eliminating knowledge system brittleness. The remainder of this paper describes the elements of the DIDS model enabling non-brittle model-based tools to be developed. Results of experiments with the model are also presented.

DIDS Model

The DIDS model was designed to support the development of a new class of flexible model-based KA tools that can be easily adapted to changing tasks. The premise is that flexibility is achieved, without loss of acquisition power, by separating acquisition techniques for search-control knowledge from those techniques used to acquire other types of knowledge, thereby developing "PSM neutral" model-based acquisition tools. The model also minimizes the effort required to build and maintain these tools by allowing them to be composed from reusable KA procedures.

In the DIDS model, a knowledge system is viewed as performing problem-space search, just as in the Problem-Space Computational Model (Yost 1993). The search space is defined at three levels – KL, problem-space level (PSL), and symbol level (SL) – where each successive level is less abstract and provides more implementation details. The KL bounds the problem space that must be searched without describing its structure. It identifies the knowledge structures contained in and used to search the problem space as well as the criteria that must be satisfied by a solution. Conceptually, all PSMs performing this task must search this problem space so the knowledge-structure ontology identified at this level is highly reusable.

The PSL describes the way the task is formulated so that it can be performed efficiently. It defines the states in problem space, as well as the PSM and search-control knowledge used to search this space efficiently. The states define the set of possible solutions - both feasible and infeasible - that could be considered during problem solving. (Note, the states are characterized, but not explicitly enumerated.) The PSM encapsulates a search strategy defining how task-specific knowledge can be used to prune portions of the problem space. Pruning is necessary since problem spaces are typically too large to search exhaustively. *Search-control knowledge is the task-specific pruning knowledge used by these strategies.*

The SL consists of the programming languages statements used to implement the PSM and to represent the knowledge structures, states, and search-control knowledge.

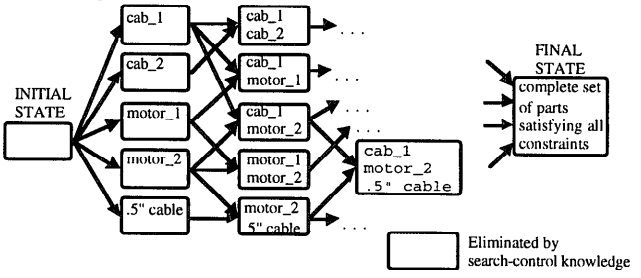


Figure 1 - The problem space for the VT task.

For example, consider the elevator-design task performed by the VT knowledge system (Marcus, Stout, & McDermott 1987). The VT task, when viewed at the KL, involves designing an elevator by selecting parts from a catalog that satisfy a user-given set of functional requirements and constraints. The knowledge structures – parts, constraints, and functional requirements – bound the space of elevator designs that may be considered.

The PSL view of the VT task is shown in Figure 1. Each state in the problem space consists of a set of elevator parts. The initial state is empty (does not contain any parts), and a final state contains a set of components that realizes the functionality of an elevator and satisfies all constraints. The types of search-control knowledge used to perform this task, however, depend on the problem-solving technique used to solve it. A generate-and-test problem solver does not use any search-control knowledge: it randomly picks a state in the problem space and checks to see if it is a final state. The VT system uses fixes, search-control knowledge describing how to repair constraint violations, to make backtracking through this problem space more efficient (Runkel & Birmingham 1994).

This three-level view of a knowledge system decomposes knowledge systems into four components:

1. A *KL task description.*
2. A *process model* describing the PSL view of the task.
3. A *KA model* describing the techniques used to acquire the knowledge structures and search-control knowledge.
4. Programming language statements used to implement the process model and KA model.

Since the focus of this paper is KA, the remainder of this section describes how the search-control knowledge distinction and other properties of the KL task description enable non-brittle model-based KA tools to be constructed from reusable KA procedures; the process model and its implementations are described in other papers (Runkel, Birmingham, & Balkany 1994).

Knowledge-Level Task Description

A KL task description consists of:

1. The solution criteria identifying the properties of a desirable solution.
2. A set of knowledge structures forming an ontology of the knowledge used to perform the task.
3. Organizational requirements describing the relationships among the knowledge structures
4. A knowledge base consisting of knowledge-structure instances defining all the task-specific knowledge, excluding search-control knowledge, necessary to perform the task.

Describing a task using a problem space focuses the acquisition process on the relevant knowledge structures. In addition, the problem space also highlights relationships among these knowledge structures, called organizational requirements, that enable the development of model-based KA tools from the KL task description. For most tasks, there are repeating patterns of knowledge structures that describe each domain concept. Model-based acquisition power can be achieved by using acquisition techniques designed to acquire these patterns. For example, each part of an elevator is represented by a *part* knowledge structure, a *function* knowledge structure describing the functionality of the part, and zero or more *constraint* knowledge structures describing situations when the part cannot be used (Figure 2). In addition, for this task a part can perform exactly one function, and each function may be performed by one or more parts. These organizational requirements are the basis for model-based acquisition techniques. A model-based tool, after creating a part knowledge structure, would ask the domain expert to select the function performed by this part, and to supply any constraints specifying when the part cannot be used. In addition, the tool ensures that each part performs exactly one function and every function is performed by at least one part.

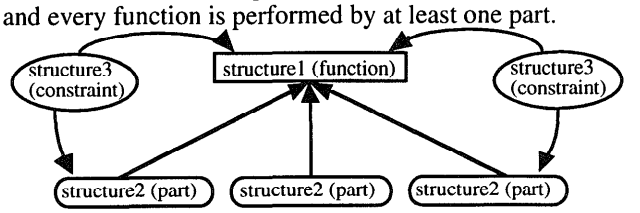


Figure 2 - The organizational requirements assumed by the constraint MeKA.

The organizational requirements enable tools to both assist with the acquisition of knowledge and to support the process of updating the knowledge base when knowledge changes. Since these requirements encapsulate the relationships among structures, they enable the knowledge engineer to see the ramifications of knowledge-base modifications. For example, it is a simple process to find all the functions and constraints affected by the removal of a part knowledge structure from the knowledge base.

Note, the knowledge structures and organizational requirements are defined independently of the PSM used to solve the task ensuring that the KA tool will work with any PSM. (To use a KA tool, it is necessary to write translator from the generic-knowledge-structures representation to a PSM-specific representation.) The organizational requirements enable these tools to be both flexible as well as powerful.

KA Model

The KA model describes the model-based techniques used to acquire and maintain knowledge. It reduces brittleness by viewing KA tools as being composed of a collection of modular, reusable acquisition procedures, called *mechanisms for knowledge acquisition* (MeKA). A KA tool, in our model, is a set of MeKAs (one MeKA for each knowledge structure and search-control knowledge type) organized to produce dialog in a meaningful way. This modular structure insures that no one MeKA acquires both a knowledge structure and search-control knowledge, thus making it simple to adapt a KA tool to acquire knowledge for a different PSM. The MeKAs for acquiring the search-control knowledge used by one PSM are replaced by MeKAs for acquiring the search-control used by another PSM. The MeKAs for acquiring knowledge structures are unaffected by a PSM change.

The acquisition techniques embodied in MeKAs are model based. These techniques are customized to the organizational requirements identified in the KL task

description, not the PSM. A MeKA contains three types of model-based KA techniques – acquire, verify, and generalize – corresponding to the three types of techniques found in model-based tools (Birmingham & Klinker 1994). The acquire component describes the type of knowledge structure or search-control knowledge acquired by the MeKA and specifies the organizational requirements to which the MeKA's acquisition techniques are customized. The verify component describes the types of error and consistency checks performed by a MeKA. These techniques ensure that the acquired knowledge is organized as required by the organizational requirements, and that the user has provided all the knowledge specified in the task description. The generalize component describes the techniques the MeKA uses to generalize acquired knowledge. These techniques reduce the number of questions that the domain expert must answer.

For example, the constraint MeKA (Figure 3) is designed to acquire the constraint-knowledge structure when the task description is organized as shown in Figure 2. The MeKA acquires an algebraic relationship (constraint) between two classes of knowledge structures (parts and functions) when they are related by a one-to-many relationship. If the knowledge base was organized differently, then a different MeKA would be used. For example, if constraints were a binary relationship that organized the parts into a hierarchy, then a MeKA customized to acquiring hierarchies would be used.

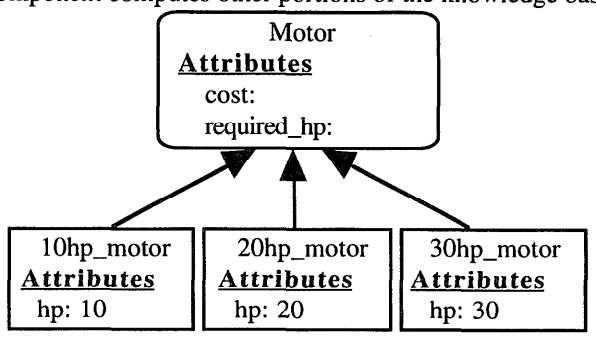
The three components of the constraint MeKA are customized to the organizational requirements on constraints in the VT task. The verify component of the constraint MeKA ensures that every function is associated with a constraint, and that this constraint defines a relationship between the function and its parts. The generalize component of the constraint MeKA asks the domain expert to broaden the scope of the acquired constraint. If the domain expert had entered a constraint specifying that the least costly part should be selected, then he may want the scope of this constraint to cover the entire knowledge base.

Constraint MeKA

Acquire:	Knowledge Structure Type: relation among structures (constraint) Organization: knowledge base organized as shown in Figure 2
Verify:	Warn if constraint does not relate functions to parts
Generalize:	Ask if constraint should be generalized to other portions of the knowledge base

Figure 3 - The components of the constraint MeKA.

The constraint MeKA is operationalized by an interface that acquires constraints by showing a hierarchy containing a function, all its parts, and the attributes associated with the parts and the function (Figure 4). The hierarchy displayed by the MeKA is likely to be all the knowledge the domain expert will need to review before providing a constraint to select among the function's parts. The question presented by the MeKA is derived from the role it expects the constraint to play during problem solving. Therefore, the MeKA asks the user for a constraint that selects among each function's parts. Finally, after the user has entered a constraint, the verification component is used to check the acquired knowledge, and the generalize component computes other portions of the knowledge base.




```
Enter the constraint used to select from the
possible designs of Motor (10hp_motor,
20hp_motor, 30hp_motor):
```

Figure 4 - Interface generated by the constraint MeKA.

Results

The effectiveness of KA tools based upon the DIDS model has been tested by running a number of experiments using DIDS. DIDS contains libraries of MeKAs and tools for creating KL task descriptions and building KA tools by combining MeKAs. These tools and libraries were used to demonstrate the flexibility of DIDS-generated KA tools, and to verify the brittleness-reduction properties of the model.

The tasks used for the experiments were all configuration-design tasks (Mittal & Frayman 1989), because we are most familiar with this class of tasks, making it easier to build the appropriate MeKA library. Configuration was also chosen because our working relationship with an industrial affiliate gives us access to a number of real-world configuration tasks and the knowledge necessary to perform them. The tasks automated during these experiments are summarized below:

1. Room Assignment (RA). Assigning workers to offices so that no constraints are violated (Linster 1992).
2. Elevator-design (VT0). Designing elevators that satisfy a given set of functional requirements and constraints by selecting parts from a catalog (Yost 1992). This task was solved without search-control knowledge.
3. Elevator design (VT1). Same task as VT0 except that search-control knowledge is used to make problem solving more efficient.

Task	M1	M2	M3	M4	M5	M6	M7	Problem-Solving Method
RA	x	x	x	x				Constraint satisfaction with distance heuristic
VT0	x	x	x	x	x	x		Constraint satisfaction
VT1	x	x	x	x	x	x	x	Constraint satisfaction with fixes
VT Val.	x	x	x	x	x	x		OPS5
P.C.	x	x	x	x				Constraint satisfaction
Truck	x	x	x	x	x			Optimal part selection

Figure 5 - The reuse of MeKAs across tasks and PSMs.

4. Elevator-design validation (VT Val.). Given an elevator design, verify that it satisfies all functional requirements and constraints (Runkel, Birmingham, & Balkany 1994).
5. Personal-computer validation (PC). Given a list of personal computer components (disk drives, monitors, memory chips, etc.) verify that these components can be combined into a working personal computer.
6. Truck design (Truck). Complete a truck design started by a human designer by selecting parts from a catalog such that all functional requirements are met, no constraints are violated, and some design parameter - such as cost, payload, or fuel economy - is maximized or minimized.

For each of these tasks, DIDS was used to capture a task description at the KL, to create a KA tool, and to build a problem solver capable of performing the task. The KA tool was then used to build a knowledge base, and the problem solver used this knowledge base to run test cases.

A KA tool was created for each task by starting with the same baseline tool and modifying it by swapping MeKAs (Figure 5). The baseline tool is a model-based tool for acquiring parts, the functions they perform, and the constraints among them. It contains four MeKAs (M1-M4), which consist of a graphical node-link-diagram editor designed to guide the domain expert in the process of organizing parts into a hierarchy based upon the functions they perform, a spread-sheet-like interface designed to reduce the amount of effort required to specify the attributes of parts and functions, a text editor facilitating the specification of constraints among parts and functions, and a set of knowledge-base browsers.

These results demonstrated the power of our approach: the same baseline KA tool was reused, even though the six knowledge systems built with it performed different tasks, used a variety of problem-solving techniques, and were implemented in a variety of languages. The RA task used a constraint network combined with a chronological-backtracking PSM that used the distances among the rooms to prune the search space. In contrast, the VT validation system was built from OPS5 rules, and the truck-design system used an optimal part-selection algorithm (Haworth, Birmingham, & Haworth 1992) written in C.

By reusing MeKAs, KA tools are easy to build. This reuse resulted in significant savings in KA tool development time. The RA task was the first to be automated. To build a KA tool for this task, the baseline MeKAs were handcoded requiring about 6 person months. (Figure 6). The other KA tools were built by reusing these MeKAs, and therefore required significantly less time to develop. Only VT0 KA tool, in addition to the RA tool, required more than a day to develop. This was the time necessary to code MeKAs M5 and M6.

The KA tools created from MeKAs are as powerful as existing model-based tools. This acquisition power was demonstrated by the three VT experiments. The automation of the VT tasks required the creation of a knowledge base with many complex parts and constraints. The DIDS-generated KA tool was not only capable of acquiring this knowledge base, but also saved acquisition time by detecting errors as knowledge was acquired (Runkel & Birmingham 1994).

The reuse of knowledge bases among the VT0, VT1, and VT Val. tasks demonstrated the reduction in KA time resulting from the use of ontologies, and by separating search-control knowledge. The knowledge base for the VT0 task took almost 400-person hours to develop. This same knowledge base was reused for both the VT1 task, which used a slightly different PSM and the VT Val. task where both the task and PSM were significantly different (validation vs. design, constraint satisfaction vs. rules). Reuse of the same knowledge base among various PSMs was possible because the search-control knowledge was clearly identified and removed. In addition, the use of ontologies made it easy to translate the knowledge base into the required PSM-specific representations. The total KA time for both the VT1 and VT Val. tasks was only 10 person hours.

These experiments demonstrated that the DIDS model contains four properties reducing brittleness:

1. Modular structure of KA tools. Since KA tools are composed from MeKAs, it was a simple process to adapt the KA tool to a new task by adding or replacing MeKAs.
2. KA techniques used by MeKAs based on the KL task description's organizational requirements. This feature enabled powerful model-based KA tools to be developed that were not tailored to a particular PSM. The experiments showed that adapting the KA tool to a new task or PSM required minimal effort.
3. Use of ontologies and the identification of search-control knowledge. The combination of these two

approaches allowed the same knowledge base to be reused with a number of different PSMs.

4. Separation of acquisition techniques for search-control knowledge. Adapting the KA tool from the RA task to the VT0 task and then again from the VT0 task to the VT1 task demonstrated the flexibility gained from this separation. In each case, in order to reuse the KA tool, the McKAs for acquiring search-control knowledge were removed. This removed any dependence between the KA tool and a particular PSM leaving the reusable portions of the tool intact. The tool was then adapted to the new task by adding MeKAs to acquire the search-control knowledge used by the new PSM.

	RA	VT0	VT1	VT Val.	PC	Truck
KB Size						
know. structs.	4020	424	424	424	2000	589
search control	50	0	62	0	0	0
% reuse	0	0	87	100	100	0
Develop. Time						
task description	3	3	0	1	3	3
KA tool	2080	395	5	1	1	1
KA	2	240	10	0	0	60
% MeKA reuse	0	67	86	100	100	100

Figure 6 - Development time (in person hours) reduction resulting from reuse.

Summary

The DIDS model defines a KA tool architecture that enables the development of model-based KA tools that can be easily modified and extended to acquire knowledge for new tasks and PSMs. The key to this flexibility is viewing knowledge systems at the KL, PSL, and SL. The use of these three levels enables powerful, but PSM-independent, KA tools to be developed. These capabilities are achieved by separating acquisition techniques for search-control knowledge from other types of knowledge, and by creating MeKAs that derive their model-based acquisition techniques from the organizational requirements identified in the KL task description. In addition, flexibility is gained by using the ontology of knowledge structures identified at the KL to reuse knowledge bases. Our experiments have demonstrated the flexibility gained by this approach.

Acknowledgment. This work was funded, in part, by a gift from Digital Equipment Corporation and by the National Science Foundation Grant MIPS-905781. Alan Balkany made significant contributions to this work.

References

Balkany, A.; Birmingham, W.P.; and Runkel, J.T. 1993. Solving Sisyphus by Design. *Knowledge Acquisition*. forthcoming.

Birmingham, W.P. and Klinker, G. 1994. Knowledge-Acquisition Tools with Explicit Problem-Solving Models. *The Knowledge Engineering Review* 8 (1):

Gruber, T.R. 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition* 5 (2): 199-220.

Haworth, M.S.; Birmingham, W.P.; and Haworth, D.E. 1992. Optimal Part Selection. Technical Report, CSE-TR-127-92, Department of Electrical Engineer and Computer Science, University of Michigan.

Linster, M. 1992. Document for the Participants of Sisyphus '92, Part 2: Models of Problem Solving.

Marcus, S. eds. 1988. *Automating Knowledge Acquisition for Expert Systems*. Kluwer Academic Publishers.

Marcus, S.; Stout, J.; and McDermott, J. 1987. VT: An Expert Elevator Designer that Uses Knowledge-based Backtracking. *AI Magazine*.

McDermott, J. 1988. Preliminary Steps Toward a Taxonomy of Problem-Solving Methods. In *Automating Knowledge Acquisition for Expert Systems*, eds. Sandra Marcus. Kluwer Academic Publishers.

Mittal, S. and Frayman, F. 1989. Towards a Generic Model of Configuration Tasks. In Proceedings of the 11th IJCAI, Morgan Kaufman.

Musen, M.A. 1989. *Automatic Generation of Model-Based Knowledge-Acquisition Tools.* San Mateo, California: Morgan Kaufmann Publishers, Inc.

Newell, A. 1981. The Knowledge Level. *AI Magazine*.

Puerta, A.; Egar, J.; Tu, S.; and Musen, M. 1992. A multiple-method knowledge-acquisition shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition* 4 (2).

Runkel, J.T. and Birmingham, W.P. 1994. Solving VT by Reuse. In Proceedings of 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada.

Runkel, J.T.; Birmingham, W.P.; and Balkany, A. 1994. Separation of Knowledge: A Key to Reusability. In Proceedings of 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Alberta, Canada.

Steels, L. 1992. Reusability and Configuration of Applications by Nonprogrammers. Technical Report, VUB AI Memo 92-4, VUB AI Lab Pleinlasn 2, 1050.

Wielinga, B.J.; Schreiber, A.T.; and Breuker, A.J. 1992. KADS: A Modeling approach to knowledge engineering. *Knowledge Acquisition.*

Yost, G.R. 1992. Configuring elevator systems. Technical Report, Workplace Integration Technologies Group, Digital Equipment Corporation, 111 Locke Drive (LM02/K11), Marlboro, MA 01752.

Yost, G.R. 1993. Acquiring Knowledge in Soar. *IEEE Expert* 8 (3): 26-34.