Defining and Monitoring Knowledge Integrity

Fatma Mili, Krish Narayanan, Vamsi Atluri
School of Engineering and Computer Science
Oakland University
mili@oakland.edu

Abstract

Knowledge management has been claimed as the correct response to rapid change. Decisions need to be made in light of up to date knowledge. The changing nature of the knowledge raises issues of quality assessment and monitoring. In this paper, we motivate the need for system monitoring of knowledge quality and present a set of structural properties that promote knowledge consistency. We briefly discuss system monitoring of these properties.

Introduction

Knowledge Management has emerged in the past few years as a reflection of the wide recognition of knowledge as a critical resource and on the subsequent desire to capture, grow, share, and use the organizational know-how. Because knowledge management is still in its youth, almost all of its aspects are still the subject of extensive research and wide experimentation. These aspects range from the economic considerations of valuation and return on investment, to the managerial aspects of resource allocation and management, to the technical aspects of identifying, encoding, and using the knowledge (see e.g. (3; 15; 8; 22; 26; 27; 31; 35).) The knowledge targeted by these efforts ranges from the informal multimedia information found on the web (e.g. see (2; 29)), to factual time-varying data, to more formal standards, best practices, and regulations (e.g. see (17; 24; 28).)

One of the major and often cited motivators behind the knowledge management movement is the need for organizations to rapidly recognize changes and capitalize on them. It is argued that while the information technology age had focussed on offering pre-canned efficient solutions to pre-defined problems, what organizations now need are means for "adaptation and survival in face of increasingly discontinuous change" (18). Knowledge management is presented as the answer to this rapid change and is defined as the organizational process of collecting, managing, and integrating up-to-date knowledge in its decision-making processes. By adopting a knowledge management approach, organizational decisions become less dependant on the specific tools used and more dependant on the quality of the knowledge used. Because the knowledge is evolving, its quality is not necessarily invariant. It does not suffice to validate the knowledge once; the quality needs, instead, to be continuously monitored.

In this paper, we discuss the issue of knowledge quality in the context of engineering design support systems. Engineering design is a knowledge intensive activity that is bound by a large number of requirements, standards, and regulations. The different standards and regulations are set and updated by different bodies and they are all integrated in the process of supporting design engineers' decision making. Because the pace of technology evolution is rapid, the sources of knowledge are many, and employees' turnover is high, updates to the knowledge are frequent and are done by different people. The continuity and consistency of the knowledge cannot lie on anyone's shoulders. Some system support is needed.

We start this paper by motivating the need for quality support and justifying its feasibility in the context on interest. We then, present an object model used to represent the knowledge of interest. We define integrity qualities and discuss mechanisms for monitoring them.

Quality assurance in light of evolution

Quality assurance in light of changes is not unique to knowledge management. To motivate the need for and justify the feasibility of system support of quality assurance, we examine how this issue has been handled in more traditional domains. In particular, we consider the cases of software, databases, and knowledge bases. We use insight from these areas to assess the need for it and the extent to which it is feasible.

Quality assurance in computer programs

One of the prominent distinguishing features of software systems is their evolutivity. They are never set in stone and can be changed at any time in their lifecycle. Hardware systems, in contrast, wear out with time; as a result, their designers need only be concerned with the product requirements in effect at design time. When and if requirements change, only newly designed products are bound by the updated requirements. Software, by contrast, generally outlives the requirements for which it is initially developed. The highly publicized Y2K problem is only one illustration of this over-longevity and a highlight of the significance of such properties as maintainability and evolutivity in software. The ultimate concern in programming is how to keep programs correct with their requirements as requirements change. Extensive research efforts in software engineering have focussed on identifying structural characteristics that make software systems easy to modify, and formal processes that trace changes in requirements, identify components affected by the change, and support in modifying them to adapt them to the changed requirement. Programming paradigms such as structured programming, data encapsulation, and object-oriented programming have all been motivated by a desire for increased modularity in programs, thus localizing effects of changes. Software lifecycles and software methodologies have also been driven to a large extent by the need to adapt software to requirements as these requirements get refined (rapid prototyping), or simply evolve over time.

Overall, the major quality of concern for programs is their correctness with respect to their requirements. When these requirements are formalized, correctness can be formally proven and the proof can be system-supported. Changes in requirements are discrete, and clearly identified in time. Professional programmers are responsible for the subsequent adaptation of the software to the changed requirements.

Quality assurance in databases

Databases are designed to be snapshots of some world of interest. As such, the ultimate quality of a database is the extent to which it accurately represents the world it represents. Because the world of interest is outside the scope of any system, database quality is generally approximated using intrinsic qualities of consistency and integrity. Early research in database focussed on identifying structural database properties that capture the absence of redundancy and reduce the chances of internal inconsistencies. Relational normal forms are an example of such structural properties. In addition to these structural properties, domain-specific "integrity" constraints are expressed as part of the data model. These constraints formulate state and transition conditions whose violation would be indicative of problems. The monitoring of integrity in light of data changes is the responsibility of the database management system. Less frequently, changes warrant the evolution of the database model itself. These changes are discrete, infrequent, and are handled completely manually.

Overall, the quality of a database can be decomposed into the quality of its model and that of its data. The quality of both is a relative property that cannot be proven. Weaker structural and intrinsic criteria of integrity are used. Changes to data are frequent and performed by end users; quality assurance during these changes is monitored by the system. Changes to the data model are less frequent and performed by database specialists.

Quality assurance in knowledge bases

Early knowledge based systems such as MYCIN (6), PROSPECTOR (12), and XCON (1) have been successful, thanks to the painstaking efforts of teams of pioneer researchers who single handedly elicited, encoded, tested, and maintained large collections of domain knowledge. These research and experimental efforts have set the foundations for knowledge based systems and established their feasibility. Yet, their success could not be easily reproduced in the less uniform and less controlled environment of most organizations. For the promise of knowledge based systems to become a reality, general concepts, tools, and methodologies were needed to support the tasks of knowledge elicitation and management, and to promote the reusability, manageability, and maintenance of the resultant knowledge bases.

Research targeted towards facilitating the maintenance and reuse of knowledge bases has led to promoting modularity and abstraction in the representation of knowledge. Newell (25) has advocated raising the level of abstraction of these systems from their symbol level to a more abstract knowledge level. The need for such abstraction has been substantiated by Clancey's findings when analyzing MYCIN's knowledge base (9). Clancey's work (10), in turn has motivated further work developing methodologies such as KADS (34), identifying and formalizing problem solving strategies (e.g., (7; 5; 20; 21; 19)), and building tools for knowledge acquisition that can be customized to specific methods and strategies (11; 14; 13; 23; 32).

Knowledge in knowledge bases is collected for the specific purpose of solving a problem. In that respect, the major quality of interest is the extent to which, collectively, this knowledge solves the problems of interest efficiently independently of the quality of composing knowledge units. Changes in knowledge bases are generally discrete and performed by specialists. The quality of the updated knowledge base can be systematically tested for collective validity on a set of pre-defined test cases.

Quality assurance in knowledge management

Before discussing quality assurance in knowledge management, we summarize the discussion above by identifying two underlying threads: 1. A common concern for structure as preventive measure and 2. A set of dimensions that are relevant to the need for and the feasibility of an automatic quality assurance process.

A concern for structure: Modularity and abstraction In all domains, there has been a focus on the level of abstraction of the information represented and on the importance of the modularity of its representation. These qualities are also important in the context of Knowledge Management. The object model presented in section 3 is in part motivated by these criteria.

Dimensions of quality control Three dimensions have emerged in the characterization of quality control. They are:

1. The frequency of change and whether change is a continuous evolutive process or whether it is confined to discrete instances. When change is discrete and infrequent, validation can be performed along with the change. When change is frequent and continuous, quality assurance is best automated.
2. The qualifications of the person performing the change, i.e., whether they are specialists, knowledgeable about the technical issues involved, or whether they are end users with different training. Specialists can be relied on to perform the necessary validation. For end users, system support is needed.
3. The nature of the quality being assured, whether it is an intrinsic property that can be formalized and validated within the realm of the environment of interest, or whether it is relative to some outside entity. Intrinsic properties can be formalized and verified internally. For qualities relative to external entities, only weak flags can be set to detect problems.

The first two dimensions impact the need for systematic quality monitoring, whereas the third dimension impacts the feasibility of such monitoring.

We now consider the knowledge management context with respect to the three dimensions identified.

Frequency of change: Typically, the very reason why a knowledge management approach is adopted is that the knowledge of interest is changing rapidly. In the specific case of engineering design, the knowledge targeted is the set of constraints, standards, and regulations related to design. Technologies, standards, and regulations all do change with variable frequencies. The rate of change is such that some system support for validation would be invaluable.

Qualifications of the person performing the changes: Knowledge in knowledge management typically comes from various sources with no centralized control or coordination. In engineering design, the constraints of interest come from different sources of expertise and sources of regulation. For example, the design of an engine is bound by functionality constraints, defined by mechanical engineers; casting constraints, defined by die casting experts; assemblability constraints, defined by assembly plant managers; environmental constraints, defined by federal and professional agencies; and so forth. The knowledge is owned and set by different bodies, at different times; there is no uniformity in training. More importantly, there is no centralized oversight over the consistency across sources and across time.

Vehicle Interior	
Attributes	
driverSeat:	Seat
driverSeatPosition:	Position
population:	PopulationRange
Methods	
hipPoint(Seat,Position):	Position
Constraints	
Sits 95th percentile	
Adequate reach 95th percentile Male	
Adequate reach 5th percentile Female	
Adequate rear view 95th percentile Male	
Adequate rear view 5th percentile Female	

Figure 1: Class Vehicle interior extended with constraints

Nature of the qualities of interest: Ruggles (30) identifies four dimensions of knowledge quality: timeliness, comprehensiveness, accuracy, and cognitive authority. These qualities are at least in part non-intrinsic; they cannot be proven internally. Weaker intrinsic properties similar to database integrity constraints can be defined. Because knowledge is more expressive than data (richer semantics), and because knowledge from different sources is overlapping, the integrity constraints that can be formulated are potentially more expressive and more effective than in other settings.

Model: Classes with Constraints

We briefly introduce the object model used to capture engineering design constraints. The representation is devised for individual knowledge bases developed by specific regulation bodies. All knowledge bases of interest come together in the context of an intelligent CAD/CAM system. An object-oriented model is used to represent each of these knowledge bases. For the sake of modularization and encapsulation, the constraints are grouped together and associated with the class of objects that they constrain. For the sake of abstraction, the constraints are not represented in an operational fashion within the class methods or as active rules. They are, instead represented declaratively as part of the class. To minimize redundancy, we have opted to represent each constraint exactly once, in the most general context in which it applies. We illustrate the representation in Figure 1 where we show the example of a Vehicle Interior class defined using the UML (4) modeling language. We added category constraints to the class diagram.

Because the classes are mostly an encapsulation tool for the constraints, no attempt is made to fully define them. We only include those attributes and methods used in the expression of the constraints. Constraints are instances of the class Constraint. For example, the first constraint in the Vehicle Interior is illustrated in Figure 2.

Sits 95th percentile:Constraint	
name:	sits 95th percentile
in class:	Vehicle interior
text:	The distance between driver's seat's hip point and the foot pedal must be longer than the length of the leg of the 95th percentile male by at least d centimeters.
illustration:	CAD drawings.
rationale:	Drivers must be comfortable enough to sustain the sitting position for extended period of time.
cost:	If violated, vehicle will not be usable by the tallest segment of the driver population.
source:	SAE standard.
formula:	distance(hipPoint, footPedal) > population.legLength[95] + d
tolerance:	10 % of d
parameters:	driverSeat.position, population hipPoint.position
efficiency:	Always true for buses and vans (implied by buses and vans standard measurements).

Figure 2: Example of a constraint instance.

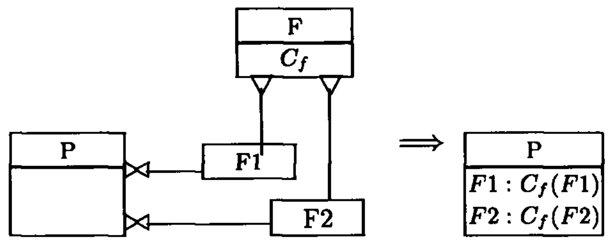

Figure 3: Constraint Propagation via Part-Feature association

Scope of the constraints

The requirement that constraints be associated with a unique class is a measure of redundancy reduction. A constraint may apply to any number of classes. The actual scope (set of classes where it applies) is derived using the following set of inference rules.

Deduction (Inheritance) Class specialization is one of the foundational features of object orientation. It allows the definition of classes in a modular way. If a constraint C constrains a class G and if class S is a subclass of G, then C constrains also S. This is captured by the following rule:

Inference Rule 1 Constraint propagation by deduction.

$$\frac{\begin{array}{c}\text{C constrains G}\\ \text{S subclass of G}\end{array}}{\text{C constrains S}}$$

Contextual Qualification In any domain, there are features that can occur in any number of classes, yet maintain the same set of properties across classes. For example, in the building domain, a rectangular opening, such as a window opening, carries the same requirements, whether it occurs on a kitchen wall, a bathroom wall, or a garage wall. For abstraction purposes, instead of repeating feature constraints in all classes where the feature occurs, we define a separate class for the feature and relate it to the containing classes using a part-feature relationship. Graphically, we represent this relationship using a bowtie on the part side. When a constraint Cf constrains a class F, and if class P has a feature F1 of type F, then constraint Cf constrains P at F1, or P.F1. This is captured by the rule below and illustrated in Figure 3.

Inference Rule 2 Contextual qualification

$$\frac{\begin{array}{l}\text{C constrains F}\\ \text{P has feature F1}\\ \text{F1 instance of F}\end{array}}{\text{C(F1) constrains P}}$$

Assembly binding Assembly is another common association between classes. It is usually put under the umbrella of aggregation (16). A travel coffee mug for example, is an assembly of a cup, a cup handle, and a lid. The lid fits tightly with the cup opening; the cup handle has an attachment mechanism that mates with an attachment mechanism on the cup. Generally, parts of the same assembly have interfaces to other parts of the same assembly. The corresponding interfaces have to be compatible, resulting in a binding (relationship such as equality) between attributes of the different parts and bindings between attributes of the whole and attributes of the parts. We represent assembly associations by creating an assembly class template that connects the whole and the different parts. Constraints on the common (bound) attributes are placed in the assembly and shaped by the different participants (whole and parts). If a constraint C constrains attribute A on an assembly, then any of the participants of the assembly that has an attribute A or that has an attribute bound to A is also constrained by C. This is captured by the following two inference rules covering the two cases:

Inference Rule 3 Assembly propagation

$$\frac{\begin{array}{l} C \text{ constrains Assbly} \\ P \text{ part in Assbly} \\ A \text{ attributes of } P \\ A \in C.\text{parameters} \end{array}}{C \text{ constrains } P} \qquad \frac{\begin{array}{l} C \text{ constrains Assbly} \\ P, P' \text{ parts in assbly} \\ A, A' \text{ attributes of } P, P' \\ A' \in C.\text{parameters} \\ \text{bound}(A, A') \end{array}}{C \text{ constrains } P}$$

Knowledge Quality

As we have discussed in section 2, the quality of the knowledge cannot be fully captured as an intrinsic property. Instead, weaker measurable criteria of integrity are defined. These criteria are generally intrinsic criteria of consistency rather than relative criteria of correctness. Criteria of consistency characterize the state of being free of certain inconsistencies. In an object model, two types of inconsistencies can occur: inconsistency within a single class or inconsistency across classes. Inconsistencies can be structural or logical. We focus here on logical inconsistencies and show a sample set.

Defining Inconsistency

Inconsistency within a class A class is inconsistent if it has no instances because it is over constrained. This is captured by the following definition.

Definition 1 *A class A is said to be logically* **inconsistent** *if the set $C = \{c|c \text{ constrains } A\}$ is a contradiction, i.e. $C \Rightarrow false$.*

Inconsistency across classes Inconsistency across classes generally is a result of some sort of redundancy. In the traditional relational database model, a set of normal forms has been defined to indicate the absence of specific types of redundancies, and thus the absence of resulting inconsistencies. We use the same model to define four normal forms. Whereas with the relational model, higher normal forms represent stronger conditions, it is not the case here. Therefore, except for the first normal form, we will not number the normal forms, we name them instead. The first normal form, as is the case in the relational model, is in fact a pre-requisite, not directly related to redundancy.

Definition 2 *A knowledge base is in* **first normal form** *iff all properties of interest are individually identified.*

This applies in particular to constraints. For maintenance, evolution, and validation purposes, it is important to be able to group and classify constraints other than by the object to which they apply. Attributes such as rationale, cost, and source, need to be clearly identified (rather than embedded in the text of the constraint). For example, the standard placement of controls on the control panel is devised with the assumption that drivers are right-handed. In an agile design situation where the driver is left-handed, it is important to be able to identify those constraints and revise them.

Definition 3 *A knowledge base is in* **generalization normal form** *iff all constraints are associated with the most general class to which they apply.*

A constraint applicable to all openings for example, should not be associated with specific types of openings. Instead, it should be associated with the general class opening. This eliminates redundancy across siblings and redundancy between parents and children.

Definition 4 *A knowledge base is in* **smallest context normal form** *iff all constraints are associated with the smallest context to which they apply.*

For example, all constraints about window openings should be placed with the class window opening rather than repeated with every occurrence of this feature and associated with the wall in which they appear. This eliminates repetition within a same class with multiple occurrences of a feature, and repetitions across classes sharing a feature.

Definition 5 *A knowledge base is in* **encompassing context Normal form** *iff all constraints are associated with classes that include all of the attributes being constrained.*

In other words, a class cannot "enforce" a constraint if the constraint refers to attributes of other classes. This normal form justifies our definition of the template assembly for example. Constraints related to attributes that are shared between the cup, the lid, and the cup holder cannot be left as the responsibility of any of the individual classes. They belong to the encompassing assembly class. More generally, this normal form states that every constraint involving more than one class implies the existence of an encompassing class that contains (or involves) both. A constraint binding the heat generated by an engine and the cooling power of a radiator for example, should not be placed within the radiator nor within the engine. It belongs in the encompassing unit (engine block).

Monitoring and preserving Integrity

The normal forms defined here can be used as general guidelines for modeling the knowledge base. The initial design may identify additional domain specific knowledge integrity constraints. The preservation of the normal forms and other knowledge integrity constraints can be done as classes and constraints are added, retracted, and modified. A set of triggers (event-condition-action rules (33)) can be placed to guide the user updating the knowledge base. Note that because the knowledge integrity normal forms are based on intentional criteria rather than extensional properties, the system can only detect potential problems, suggest courses of actions, and support the user in selecting and executing a course of action. For example, given a class G with two subclasses A and B, if class A houses a constraint C and the user adds the constraint C to class B, the system will raise a possibility of violation of the generalization normal form, and suggest that the constraint C be moved up to class G.

Similar triggers can be set up to detect the violation of other normal forms as users modify constraints and classes. We have found that users need help primarily in deciding where to place new classes, i.e. how to relate new classes to existing classes. A user defining a sunroof opening for example, may start by defining a new class from scratch. As the system detects intersections and similarities, it may advise the user and suggest creating relationships to other classes. The system may find for example, that the new class sunroof opening has much in common with the class window opening. The system suggests creating a common parent to both classes.

Overall, the normal forms can be used as a basis for building a friendly interface for knowledge acquisition. Such interface would guide the user in deciding where to place new classes, how to connect them, and where to place the constraints.

Summary, Conclusion

In this paper, we have discussed the issue of assessing and preserving the quality of organizational knowledge repositories. Because such repositories are contributed to by a variety of people, and because their value is in the capture of know-how of workers who may no longer be part of the organization, monitoring of quality and consistency is essential. We have defined an object model in which domain constraints are stored in their declarative form. We have listed some of the class relationships that we have found to be key in design environments, and defined a set of normal forms around these relationships.

Because the normal forms are intentional, their enforcement by the system must be semi-automatic. The system supports the user by detecting problems and suggesting fixes. The user has the final say.

Acknowledgements

This research has been supported by the Daimler Chrysler Corporation and the Michigan Research Excellence Fund.

References

[1] V. E. Barker and D. E. O'Connor. Expert systems for configuration at digital: Xcon and beyond. *Communications of the ACM*, 32(3), March 1989.

[2] Irma Becerra-Fernandez. Searchable answer generating environment (sage): A knowledge management system for searching for experts in florida. In *Proceedings, 12th Annual FLAIRS Conference*, 1999.

[3] Frank Blackler, Michael Reed, and Alan Whitaker. Editorial introduction: Knowledge workers and contemporary organizations. *Journal of Management Studies*, 30(6):852–862, November 1993.

[4] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison Wesley, 1994.

[5] D. C. Brown and B. Chandrasekaran. Design problem solving: Knowledge structures and control strategies. In *Research Notes in Artificial Intelligence*. Pitman, 1989.

[6] Bruce G. Buchanan and Edward H. Shortliffe. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison Wesley Publiching Company, 1985.

[7] B. Chandrasekaran. Design problem solving: A task analysis. *AI Magazine*, 40(11):59–71, 1990.

[8] Chun Wei Choo. *The Knowing Organization*. Oxford University Press, 1998.

[9] W. J. Clancey. The epistemology of a rule-based expert system: a framework for explanation. *Artificial Intelligence*, 20:215–251, 1983.

[10] W.J. Clancey. Heuristic classification. *Artificial Intelligence*, 27(3):289–350, 1985.

[11] H. Eriksson, Y. Shahar, S.W. Tu, A.R. Puerta, and M.A. Musen. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79(2):293–326, 1995.

[12] J. Gaschnig. Application of the prospector system to geological exploration problems. In J. E. Hayes, D. Michie, and Y-H Pao, editors, *Machine Intelligence*. Wiley, 1982.

[13] J.H. Gennari, R.B. Altman, and M. A. Musen. Reuse with protege-ii: From elevators to ribosomes. In *ACM-SigSoft 1995 Symposium on Software Reusability*, pages 72–80, 1995.

[14] John Gennari, Samson W. Tu, Thomas E. Rothenfluh, and Mark A. Musen. Mapping domains to methods in support of reuse. In B.R. Gaines and M.A. Musen, editors, *10th Banff Knowledge Acquisition for Knowledge-based Systems Workshop*, 11-1, 11-20 1996.

[15] William E. Halal, editor. *The infinite resource: Creating and leading the knowledge entreprise.* Jossey-Bass Publishers, 1998.

[16] Haim Kilov and James Ross. *Information Modeling: An object oriented approach.* Printice Hall, 1994.

[17] Alistair Lowe, Chris McMahon, and Steve Culley. A method for the study of information user profiles for design engineers. In *ASME DETC99/DTM-99*, volume 24, 1999.

[18] Yogesh Malhotra. Knowledge management for the new world of business. *Asian Strategy Leadership Institute Review, Special Issue on Knowledge Management*, 6, 1998.

[19] S. Marcus and J. McDermott. Salt: A knowledge-acquisition laguage for propose-and-revise systems. *Artificial Intelligence*, 39(1), 1989.

[20] Sandra Marcus, editor. *Automating Knowledge Acquisition for Expert Systems.* Kluwer Academic Publishers, 1988.

[21] J. McDermott. Preliminary steps toward a taxonomy of problem solving methods. In Sandra Marcus, editor, *Automating Knowledge Acquisition for Expert Systems*, pages 225–256. Kluwer Academic Publishers, 1988.

[22] Rick Mullin. Knowledge management: A cultural evolution. *Journal of Business Strategy*, September/October 1996.

[23] M.A. Musen, J.H. Gennari, H. Eriksson, S.W. Tu, and A.R. Puerta. PROTÉGÉ-II:computer support for development of intelligent systems from libraries of components. In *Proceedings of MED-INFO'95, Eighth World Congress on Medical Informatics*, pages 766–770, 1995.

[24] Krishnakumari Narayanan. *Knowledge Modeling for Engineering Design.* PhD thesis, School of Engineering, Oakland University, 2000.

[25] A. Newell. The knowledge level. *Artificial Intelligence*, 18:87–128, 1982.

[26] Ikujiro Noraka and Hirotaka Takeuchi. *The knowledge-Creating Company.* Oxford University Press, 1995.

[27] Daniel E. O'Leary. Entreprise knowledge management. *IEEE Computer*, 31(3):54–61, March 1998.

[28] Daniel E. O'Leary. Developing a theory-based ontology for "best practices" knowledge bases. In *AAAI Workshop on Knowledge Management*, 2000.

[29] B. Roberts. Internet as knowledge manager. *Web Week*, page 30, 9 September 1996.

[30] Rudy Ruggles. Knowledge tools: Using technology to manage knowledge better. Technical report, Ernest & Young LLP Center for Business Innovations, April 1997.

[31] Gerald H. Taylor. Knowledge companies. In William E. Halal, editor, *The infinite resource*, pages 97–110. Jossey-Bass Publishers, 1998.

[32] S.W. Tu, H. Eriksson, J. Gennari, Y. Shahar, and M.A. Musen. Ontology-based configuration of problem-solving methods and generation of knowledge-acquisition tools: Application of protÉgÉ-ii to protocol-based decision support. *Artificial Intelligence in Medicine*, 7(3):257–289, 1995.

[33] Jennifer Widom and Stefano Ceri, editors. *Active dabase systems: triggers and rules for advanced database processing.* Morgan Kaufman Publishers, Inc., 1996.

[34] B.J. Wielinga, A. Th. Schreiber, and J.A. Breuker. Kads: A modeling approach to knowledge engineering. *Knowledge Acquisition*, 1(4), 1992.

[35] Sidney G. Winter. Knowledge and competence as strategic assets. In Treece David, editor, *The Competitive Challenge: Strategies for Industrial Innovation and Renewal.* Harper & Row Publishing, New York.