Key Issues for Automated Problem-Solving Methods Reuse

Dieter Fensel[1] and **V. Richard Benjamins**[2]

Abstract. Developing software by selecting, adapting, combining, and integrating existing components instead of starting the system development process from scratch has become a key factor in economic software development. However, such a development process has to deal with four problems: First, components must be selected. Second, components must be adapted because they neither fit precisely the task that should be performed nor do they necessarily fit well to other selected components. Third, components must be combined and their interaction must be established. Fourth, it may be necessary to decompose complex problems into smaller subtasks for which components can be found. In this case, a general system frame must be established that enables to form an integrated system out of separate components. In this paper, we present our means to deal with these problems: brokers, adapters, connectors, and task structures. Although we discuss our approach in the context of problem-solving methods, some concepts are also applicable to software components in general.

1 INTRODUCTION

Knowledge-based systems are computer systems that deal with complex problems by making use of knowledge. This knowledge is mainly represented declaratively rather than encoded using complex algorithms. However, human experts can exploit knowledge about the dynamics of the problem-solving *process* and such knowledge is required to enable problem-solving in practice and not only in principle. Making this knowledge explicit and regarding it as an important part of the knowledge contained by a knowledge-based system is the rationale that underlies *problem-solving methods*. Problem-solving methods describe this control knowledge independently from the application domain thus enabling reuse of this knowledge for different domains and applications. It describes which reasoning steps and which types of knowledge are needed to perform a task. Problem-solving methods are used in a number of ways (see e.g. [9]): as a guideline for acquiring problem-solving knowledge from an expert, as a guideline for decomposing complex tasks into subtasks, as a description of the essence of the reasoning process of the expert and knowledge-based system, as a skeletal description of the design model of the knowledge-based system, and as a means to enable flexible reasoning by selecting methods during problem solving. Currently, problem-solving methods are used by nearly all frameworks for knowledge engineering. Libraries of problem-solving methods are described in [2], [8], [9], [25], and [26]. Some of these libraries provide textbook-style descriptions of reasoning strategies and others provide implemented software components. The former support the system development process during conceptual modelling and specification while the latter also provide support during implementation.

[1] University of Karlsruhe, Institute AIFB, D-76128 Karlsruhe. E-mail: fensel@aifb.uni-karlsruhe.de.
[2] University of Amsterdam, Department SWI, 1018 WB Amsterdam, the Netherlands. E-mail: richard@swi.psy.uva.nl

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

However, serious bottlenecks exist that hamper the usefulness of most of these libraries because they provide very limited support in *selecting*, *adapting*, *combining* and *integrating* problem-solving methods. That is, they are collections of methods but not sophisticated libraries. Therefore, someone trying to use these libraries encounters a number of problems: the selection of a method from a library is only weakly supported and remains a kind of “magic trick“. Often, problem-solving methods fit only approximately to particular domain- and task-specific circumstances. They fit in principle, however, significant adaptation would be necessary to actually use them as an implemented component. Problem-solving methods hardwire assumptions concerning the way in which they interact with their environment (their communication style) and this way may not match the software or task environments they are applied in (an interoperability problem). Complex problems may require the combination of different methods, possibly from different libraries, e.g., if libraries are incomplete.

Our contribution deals with these problems. We provide solutions for four key issues in reusing problem-solving methods. A *broker* is described that enables the selection of methods from different libraries assuming that (1) these libraries are made available via wide-area networks like the WorldWideWeb and (2) are annotated with ontological information that enables their informed retrieval. *Adapter*s are presented as means to enable the adaptation process of methods to tasks and domains. They are the key concept for keeping libraries manageable (i.e., limiting the number of elements) and useful (i.e., providing elements that either fit perfectly to given domain- and task-specific circumstances or provide significant support in their generalization or specialization). *Connectors* are —in software architectures— a means to implement the interaction of components (i.e. their interoperability) and we propose their use for problem-solving methods. They provide an abstraction mechanism in cases where assumptions on the communication style may conflict. Finally, we show how *task structures* serve a twofold purpose: they decompose complex functionalities into functionalities that can be handled by components, and they provide the integrated system frames for loosely coupled components.

This paper is organized as follows. In Section 2 we describe a *broker* that handles the selection process of components. *Adapters* are discussed in Section 3 as means to enable simple and reusable adaptations of components. *Connectors* are briefly introduced in Section 4 to provide the connections between components. Section 5 shows *task structures* as means to decompose problems and to provide integrated system frames. Finally, conclusions are provided in Section 6.

2 BROKER

Software and knowledge reuse via networks is becoming an increasingly popular topic. Developing intelligent agents that provide reasoning services based on problem-solving methods (cf. [5]) introduces an interesting application framework and raises interesting research issues. In the following, we will sketch two key issues in designing such reasoning services. First, we discuss the role of *ontologies* and second we describe the core of a *brokering* service.

Ontologies have been proposed to enhance knowledge sharing and reuse (cf. [19]). An ontology provides a conceptualization of some subject matter, which can be shared by multiple reasoning components communicating during a problem solving process. Using ontological engineering for describing problem-solving methods provides two important benefits for reuse. The resulting method specification is grounded on a common, shared terminology and its knowledge requirements are conceptualized as ontological commitments [23]. Currently, ontologies are mainly used to formalize domain or common sense knowledge. However, there are recent proposals and initiatives ([24], [5], [6], [10]) that employ ontologies to formalize problem types and problem-solving knowledge (i.e., methods). For example, the *Knowledge Annotation Initiative of the Knowledge Acquisition Community* $(KA)^2$ [6] aims at building a consensual ontology which can be used to describe the various research groups and their results in knowledge acquisition. A subpart of this ontology is concerned with tasks and problem-solving methods.

Another example is to provide axiomatic descriptions of problem-solving methods to support (semi-) automatic classification and selection of methods. Examples of such efforts include [15] where ontologies are used to describe the task parametric design, the problem-solving method propose & revise, and their mapping resulting in a task-specific ontology for the method; and [31] who defines a task ontology for diagnostic problems.

Providing different libraries of problem-solving methods via the Web requires a *broker* that mediates between customers and providers of problem-solving methods (cf. [3], [12]). The overall layered architecture of such a broker is depicted in Figure 1. The broker has to communicate with clients via *clients systems*. A client is someone that has a complex problem but can provide domain knowledge that describes it and that supports problem-solving. The providers are developer teams of problem-solving methods. Their *provider systems* are annotated libraries of problem-solving methods. The problem-solving methods are implementations that solve complex tasks by using domain knowledge for defining and solving the problem. The annotations are necessary to support their selection process and the communication process with the methods. The core of an intelligent broker for problem-solving methods consists of an *ontologist* that supports the selection process of problem-solving methods for a given application. Basically it has to provide support in building or reusing a domain ontology and in relating this ontology to an ontology that describes generic classes of application problems. This problem-type ontology has to be linked with problem-solving method-specific ontologies that allow the selection of a method.

The broker uses three different ontologies: domain ontologies (D_O), problem type ontologies (PT_O), and problem-solving method ontologies (PSM_O). These ontologies provide the contents of the communication processes between the different elements of the broker (cf. Figure 1). The different information flows are concerned with the process of finding appropriate problem-solving methods, and can be distinguished by the contributing agents and the direction of the communication. In total, four different communication flows can be identified (see Figure 1):

- **Sending a *request* from the client to the ontologist**. Terms of the domain and problem ontologies are the content of the message. The client interface uses a *domain ontology* to guide the interaction process with the client and it sends selected expressions to the ontologist.
- **Sending a *negotiation* from the ontologist to the client interface**. The ontologist might need further clarification from the client before it can finish the selection process of problem-solving methods. This clarification may ask for more precise definitions of terms and their relationships necessary to derive an element of the problem ontology from an element of the domain ontology.
- **Sending a *query* from the ontologist to a provider interface**. After having translated the domain-specific terminology into a problem-specific terminology,, the ontologist has to derive an expression in a problem-solving method-specific ontology. Then, this expression is passed as a query to the provider interface.
- **Sending a *response* from a provider interface to the ontologist**. The response of a provider interface may have two forms: providing a simple yes that the required service can be provided or the wish to introduce further *assumptions* on the problem that make it tractable and/or the introduction of *requirements* on domain knowledge that has to be fulfilled by the client.

Most of these information flows can be automated. However, the translation of a domain ontology in an ontology describing problem types has to be done semiautomatically for each new domain. The introduction of assumptions that introduce requirements on domain knowledge or that weaken the task (cf. [4], [35]) cannot be done without user interaction. The mapping between problem type and problem-solving methods has to be done by the providers of these method libraries. Therefore it is already established before the broker uses these libraries. The mappings needed during runtime of the method can be derived from the links between the different ontologies. These mappings are implemented by adapters that will be discussed next.

3 ADAPTER

Adapters are used to mediate between problem definitions, domain knowledge, and problem-solving methods. In the following, we will explain with an example why adapters are required for this purpose and what their implications are.

3.1. Adaptation for Reuse

Chronological backtracking defines a very generic problem-solving method (i.e., a search strategy) that does not make any assumptions

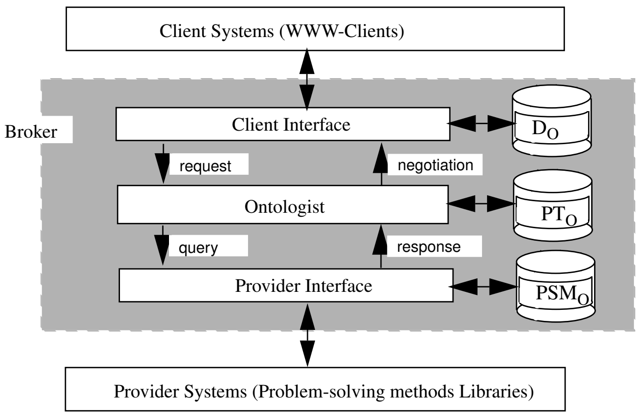

Figure 1. The layered architecture of the broker.

on the structure of nodes in the search space nor on the elementary transitions between states. [11] describe several adaptations of this generic search procedure for specific tasks. Chronological backtracking is adapted to a method for one-player board games by adding code that refines state transitions and data structures of the method. The resulting *boardgame method* adds two refinements which introduce commitments on the task and the required domain knowledge:

- the initial, intermediate and final states describe board positions, and
- the knowledge that is required to define state transitions is defined in terms of moves in games.

However, this adaptation of chronological backtracking is not kept separate in [11]. Therefore, the implementation of chronological backtracking is no longer available for defining a problem solver for, e.g., planning problems. Moreover, the adaptation to the boardgame problem cannot be used to adapt, e.g., a local search procedure to board games. Separating the descriptions of search procedures and their refinements provides reusability in two dimensions:

- search procedures remain available for different tasks and domains, and
- task- and domain-adaptations remain applicable for different search strategies.

In general, different search methods can be applied to the same task, and [7] conclude that problem-solving methods should be represented in a task-independent way. However, their task-specific (i.e. adapted) versions (for domains and tasks) provide much more support in developing systems and in acquiring the domain knowledge than the task-neutral versions. An easy way to overcome this dilemma is to externalize their adaptations through (external) *adapters*. Adapters were originally proposed in [16] to link problem-solving methods and tasks. However, they can also be used to externalize the specialization of problem-solving methods. The use of adapters in this way provides three variants of reuse that overcome the shortcomings mentioned above:

- The search procedure can be reused for different problems because the adaptation is kept separate.[1]
- The adaptation can be reused for different search methods because it is kept separate.
- The combination of search procedure and adapter can be reused as a strong problem-solving methods (i.e., as a method with many task- and domain-specific commitments) in cases where its assumptions fit.

The adapter concept is therefore essential in preventing combinatorial explosion. Already a textbook style description of problem-solving methods in [8] provides hundreds of methods. Implementing these methods adds additional level of details and therefore distinctions. *Externalizing* adaptation is the key factor in component-based development of problem-solving methods. A simple illustration should clarify the point. Assuming n search strategies and m problem types. Without adapters $n*m$ components would be necessary to cover all these combinations. By using adapters only $n+m$ components are necessary. Without adapters one ends up either with

- too many different components ($n*m$) leading to too much implementation and retrieval effort, or
- a small set of too generic components (n generic search procedures), or
- a small set of useful components which, however, are often too specific for given problems (x problem specific refinements with $m<x<n*m$).

The effect of adapters becomes obvious when taking a look at some examples of search methods on the one hand, and of task- and domain-refinements on the other hand.

- A local search algorithm has four main parameters that determine its search character [22]: the selection of start nodes, the generation of successors nodes, the selection of better nodes, and the definition of the preference relation. Different values for these parameters distinguish between, e.g., best-first search, hill-climbing and beam search. Keeping the more precise definitions of these parameters externally of the core definition of the method, enables to provide a large variety of search methods with a small number of components.
- The definition of a task may be less or more specific. A parametric design task (cf. [27]) can be defined by a design task that is refined to configurational design (where the elements of the artifact are given) and further to parametric design (where in addition the structure of the artifact, i.e. the selection of the components is given). [13], [17] discuss the idea of adapter piling to express this refinement process. A core definition of design problems (which is already an adapter for global-optimum problems) is enriched with an adapter for configurational design and another adapter adds commitments for parametric design.

Implementing a component for each variation of a search method or each task- and domain-specific refinement is intractable. A tractable and structured approach for reusing (usable) components can only be achieved by performing refinements via adapters and implementing different aspects or degrees of refinement by different adapters. Thus, a refined method is achieved by connecting to it a *pile* of adapters.

3.2. Related Approaches in Software Engineering

Adapters are of general importance for component-based software development. [20] introduces an adapter pattern in a textbook on design patterns for object-oriented system development. An adapter enables reusable descriptions of objects and allows to combine objects that differ in their syntactical input and output descriptions. Adapters for problem-solving methods extend this approach in several aspects:

- Adapters introduce *assumptions* necessary to close the gap between a problem definition (task) and the competence of a problem-solving methods (cf. [14], [18]).
- Adapters express the *task-specific refinements* of a problem-solving method (cf. [17]). Therefore, an adapter also specifies reusable knowledge. It does not only provide some application-specific glue, but specifies refinements relying on a problem type.
- Adapters can be piled on top of each other to express *stepwise refinement* of problem-solving methods (cf. [13]). This refinement may reflect stepwise adaptation to a problem or stepwise refinement of a generic search strategy according to refined requirements on domain knowledge.

In software engineering, each adaptation is viewed as a (nonreusable) application-specific one. *Reusable* adapters in knowledge engineering are possible because of the amount of work on *reusable task* and *problem descriptions* (cf. [9], [8]). The refinement of a problem-solving method for a specific type of

1. This implies more than just keeping a copy of the old implementation of the search procedure because during maintenance only one search procedure need to modified and not each of its copies.

problems is reusable for applications that are instances of the same problem or task type.

3.3. Related Approaches in Intelligent Information Integration

Adapters manipulate the syntactical structure of the input and output of components. In that way, they enable the application of components to different tasks and the combination of components that otherwise could not interact properly. Note, that input of a component may be case data or more complex domain knowledge. In the latter case, adapters fulfil a similar purpose as *mediators* for heterogeneous information and knowledge systems. Instead of assuming a global data schema, heterogeneous information and knowledge systems have a mediator [33] that translates user queries into sub-queries on the different information sources and integrates the sub-answers. Therefore, adapters that modify components may be combined with mediators to enable different components to access the same knowledge source in different ways or to provide homogeneous access to distributed and heterogeneous knowledge sources for a component. The former minimize the adaptation effort because the same knowledge is presented in different styles. The latter minimize the adaptation effort because heterogeneous knowledge sources are presented homogenous and uniform.

4 CONNECTOR

Adapters are mechanisms for coupling components that help to overcome syntactic and semantic differences. However, further problems may arise from different communication styles of the component. For example, a component C_1 may continuously send data to another component C_2 and wrongly assume that C_2 stores the received data during problem solving. Therefore, data might get lost. Similarly, C_1 may assume that C_2 processes its input with the LIFO principle (last in first out, i.e., its input store is a stack), however, it may apply LILO (last in last out, i.e., its input store is a queue). Such communication mismatches require additional means that properly implement the communication between components. In software architectures, *connectors* are introduced to fulfil this purpose (cf. [1], [34]). Connectors provide intermediate stores for data and organize these data in a way that fulfils the assumptions of the sending and receiving components. A variety of connectors and their according architectural styles is given in [30]. Because we have never encountered any difference in the communication styles of “ordinary” software components and knowledge-based components—actually most times the latter assume simple batch mode processing—these connectors of software architectures can immediately be applied to connect knowledge-based components.

5 TASK STRUCTURE

Task structures [9], Generalized Directive Models (GDM) [32] and CommonKADS inference structures [28] are different means to decompose a complex task into more tractable subtasks. Such an approach is necessary when the problem cannot be solved directly by available components. In this case the task of the broker becomes more complex. It has to select an appropriate task structure that decomposes the entire problem into smaller pieces for which either directly a component can be selected, or for which recursively a new refinement must be selected.

Besides being a means for decomposing a problem, task structures also provide the definition of an integrated system. They can be considered as a *system frame* for components by associating to each of its subtasks either a component or, recursively, a more refined task structure. A task structure defines an integrated system by defining the control[2] and dataflows between its subtasks. Therefore, such system frames are not only means to decompose problems, but express vice versa how a complex system can be configured out of its components. Such task structures or system frames corresponds to software architectures mentioned above.

In a network of several providers of problem-solving components, such an integrated system frame may refer to components of different providers. The integrated system only exists virtually in that case. Executing one of its subtasks may imply a call for reasoning service via the network. [21] describe the use of the CORBA protocol for this purpose, which allows distributed execution of reasoning tasks where the substeps may be performed at different servers.

6 CONCLUSIONS

In this paper we presented a principled approach to reusing problem-solving components. *Components* provide reasoning services for problems or fragments of problems. *Adapters* glue components to other components and to domain- and problem-specific circumstances to which they are otherwise not applicable. *Connectors* manage communication aspects. *Task structures* decompose problems and integrate loosely coupled components into a system frame. In order to enable problem-solving method reuse through the Web we sketched a *broker* architecture that integrates and exploits these four components. We are currently working on such a broker in the IBROW3 project [5]. We will briefly summarize these different aspects of component reuse.

At a technical or architectural level, implementing integrated reasoning systems —either directly or virtually in distributed networks— does not differ from implementing arbitrary software systems with *connectors* and *software architectures* suitable for such environments. The main extension to existing work in software engineering stems from the fact that research in knowledge engineering has accumulated a significant number of problem-specific *task structures* that can be used as problem-specific architectures. These architectures are not specific for a domain —like domain architectures in software engineering— but are specific for a *class* of problems and a specific decomposition paradigm that assumes specific types of knowledge for efficiently executing its substeps.

Adapters are also present in component-based software engineering (cf. [20]). However, the notion of reusable adapters and the use of several adapters to achieve stepwise refinement of components is rather non-standard. In knowledge engineering, problem-solving methods and problem types (i.e., tasks) are well-studied. This enables the reuse of adaptations of reasoning components and of problem-specific refinements. In this paper, we argued that this use of adapters is the key issue in *tractable* libraries (i.e., the effort in implementing the required components and in searching for them is low) and in *usable* libraries (i.e., by combining components and adapters one finds combined elements that fit to domain- and task-specific circumstances).

The key-effort required for providing advanced *brokering* services for reusing problem-solving methods is ontological engineering. Ontologies of problem types and tasks could support the selection process of components and help to establish a link from the generic vocabulary of problem-solving methods to the domain-specific terminologies of case data and domain knowledge.

2. Not all approaches to task structures provide control. Therefore, such approaches miss an important aspect of our needs.

The method specifications are grounded on a common, shared terminology and their knowledge requirements are conceptualized as ontological commitments. Defining such ontologies proper and consensual is essential for the success of component reuse because component retrieval and adaptation cannot not be done fully automatically (cf. [29] for the merits and limitations of theorem proving for component retrieval). Therefore, the human has to be kept in the loop which introduces strong requirements on understandability and user guidance through these ontologies.

Acknowledgment

Richard Benjamins was partially supported by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organisation for Scientific Research (NWO).

7 References

[1] R. Allen and D. Garlan: A Formal Basis for Architectural Connection, *ACM Transactions on Software Engineering and Methodology*, 6(3):213—249, July 1997.

[2] V. R. Benjamins: Problem Solving Methods for Diagnosis And Their Role in Knowledge Acquisition, *International Journal of Expert Systems: Research and Application*, 8(2):93—120, 1995.

[3] R. Benjamins: Problem-Solving Methods in Cyberspace. In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems (W26)* of the *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* Nagoya, Japan, August 23-29, 1997.

[4] R. Benjamins, D. Fensel and R. Straatman: Assumptions of Problem-Solving Methods and Their Role in Knowledge Engineering. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, August 1996.

[5] V. R. Benjamins, Enric Plaza, E. Motta, D. Fensel, R. Studer, B. Wielinga, G. Schreiber, Z. Zdrahal, and S. Decker: An Intelligent Brokering Service for Knowledge-Component Reuse on the WorldWideWeb. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98)*, Banff, Canada, April 18-23, 1998.

[6] R. Benjamins and D. Fensel: The Ontological Engineering Initiative $(KA)^2$. In *Proceedings of the International Conference on Formal Ontologies in Information Systems (FOIS-98)*, Trento, Italy, N. Guarino (ed.), Frontiers in Artificial Intelligence and Applications, IOS-Press, June 1998.

[7] P. Beys, R. Benjamins, and G. van Heijst: Remedying the Reusability-Usability Trade-off for Problem-solving Methods. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[8] J. Breuker and W. Van de Velde (eds.): *The CommonKADS Library for Expertise Modeling*, IOS Press, Amsterdam, The Netherlands, 1994.

[9] B. Chandrasekaran, T.R. Johnson, and J. W. Smith: Task Structure Analysis for Knowledge Modeling, *Communications of the ACM*, 35(9): 124—137, 1992.

[10] B. Chandrasekaran, J. R. Josephson, and R. Benjamins: The Ontology of Tasks and Methods, In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98)*, Banff, Canada, April 18-23, 1998.

[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] D. Fensel: An Ontology-based Broker: Making Problem-Solving Method Reuse Work. In *Proceedings of the Workshop on Problem-Solving Methods for Knowledge-based Systems (W26)* of the *Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97),* Nagoya, Japan, August 23-29, 1997.

[13] D. Fensel: The Tower-of-Adapter Method for Developing and Reusing Problem-Solving Methods. In E. Plaza et al. (eds.), *Knowledge Acquisition, Modeling and Management*, Lecture Notes in Artificial Intelligence (LNAI), 1319, Springer-Verlag, 1997.

[14] D. Fensel and R. Benjamins: The Role of Assumptions in Knowledge Engineering, *International Journal of Intelligent Systems (IJIS)*, to appear.

[15] D. Fensel, E. Motta, S. Decker, and Z. Zdrahal: Using Ontologies For Defining Tasks, Problem-Solving Methods and Their Mappings. In E. Plaza et al. (eds.), *Knowledge Acquisition, Modeling and Management*, Lecture Notes in Artificial Intelligence (LNAI), 1319, Springer-Verlag, 1997.

[16] D. Fensel and R. Groenboom: Specifying Knowledge-Based Systems with Reusable Components. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.

[17] D. Fensel and E. Motta: Dimensions for Method Refinement. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´98),* Banff, Canada, April 1998.

[18] D. Fensel and R. Straatman: The Essence of Problem-Solving Methods: Making Assumptions to Gain Efficiency, *International Journal of Human-Computer Studies (IJHCS)*, 48():181—215, 1998.

[19] N. Fridman Noy and C.D. Hafner: The State of the Art in Ontology Design, *AI Magazine*, 18(3):53—74, 1997.

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides: *Design Patterns*, Addison-Wesley Pub., 1995.

[21] J. H. Gennari, A. R. Stein, and M. A. Musen: Reuse for Knowledge-Based Systems and CORBA Components. In *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, November 9-14, 1996.

[22] R. P. Graham, Jr. and P. D. Bailor: Synthesis of Local Search Algorithms by Algebraic Means. In *Proceedings of the 11th Knowledge-Based Software Engineering Conference (KBSE-96)*, 1996.

[23] T. R. Gruber: Toward Principles for the Design of Ontologies Used for Knowledge Sharing, *International Journal of Human-Computer Studies (IJHCS)*, 43(5/6):907—928, 1995.

[24] R. Mizoguchi, J. Vanwelkenhuysen, and M. Ikeda: Task Ontologies for reuse of Problem Solving Knowledge. In N. J. I. Mars (ed.), *Towards Very Large Knowledge Bases,* IOS Press, 1995.

[25] E. Motta: *Reusable Components for Knowledge Modeling*, Ph.D. Thesis, Knowledge Media Institute, The Open University, UK, 1997.

[26] F. Puppe: *Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods*, Springer-Verlag, Berlin, 1993.

[27] A. Th. Schreiber and B. Birmingham (eds.): *Special Issue on Sisyphus, The International Journal of Human-Computer Studies (IJHCS)*, 44(3-4), 1996.

[28] A. TH. Schreiber, B. Wielinga, J. M. Akkermans, W. Van de Velde, and R. de Hoog: CommonKADS. A Comprehensive Methodology for KBS Development, *IEEE Expert*, 9(6):28—37, 1994.

[29] J. Schumann and B. Fischer: NORA/HAMMER: Making Deduction-Based Software Component Retrieval Practical. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 1997.

[30] M. Shaw and D. Garlan: *Software Architectures. Perspectives on an Emerging Discipline*, Prentice-Hall, 1996.

[31] A. ten Teije: *Automated Configuration of Problem Solving Methods in Diagnosis*, Ph.D. thesis, University of Amsterdam, the Netherlands, 1997.

[32] P. Terpstra, G. van Heijst, B. Wielinga, and N. Shadbolt: Knowledge Acquisition Support Through Generalized Directive Models. In M. David et al. (eds.): *Second Generation Expert Systems*, Springer-Verlag, 1993.

[33] G. Wiederhold and M. Genesereth: The Conceptual Basis for Mediation Services, *IEEE Expert*, September/October, pp. 38—47,1997.

[34] D. M. Yellin and R. E. Strom: Protocol Specifications and Component Adapters, ACM Transactions on Programming Languages and Systems, 19(2):292—333, 1997.

[35] A. M. Zaremski and J. M. Wing: Specification Matching of Software Components, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 6(4):333—369, 1997.