UPML: A framework for knowledge system reuse

Dieter Fensel
Institute AIFB
Univ. of Karlsruhe
76128 Karlsruhe
Germany

V. Richard Benjamins
Dep. SWI
Univ. of Amsterdam
1018 WB Amsterdam
The Netherlands

Enrico Motta
Knowledge Media Institute
The Open University
MK7 6AA, Milton Keynes
United Kingdom

Bob Wielinga
Dep. SWI
Univ. of Amsterdam
1018 WB Amsterdam
The Netherlands

Abstract

Problem-solving methods provide reusable architectures and components for implementing the reasoning part of knowledge-based systems. The *Unified Problem-solving Method Development Language, UPML*, has been developed to describe and implement such architectures and components and to facilitate their semiautomatic reuse and adaptation. In a nutshell, UPML is a framework for developing knowledge-intensive reasoning systems based on libraries of generic problem-solving components. The paper describes the components, architectural constraints, development guidelines, and tools provided by UPML. Our focus is hereby on the meta ontology that has been developed to formalize the architectural structure and elements of UPML.

1 Introduction

Problem-solving methods (*PSMs*) for knowledge-based systems (KBSs) (cf. [Schreiber et al., 1994]; [Benjamins & Fensel, 1998]) decompose the reasoning task of a KBS in a number of *subtasks* and *inference actions* that are connected by knowledge roles. Several problem solving method libraries are now available [Breuker & van de Velde, 1994], [Motta & Zdrahal, 1998]. The IBROW project [Benjamins et al., 1998] has been set up with the aim of enabling semiautomatic reuse of PSMs. This reuse is provided by integrating PSM libraries in an internet-based environment. A *software broker* selects and combines PSMs from different libraries and provides a knowledge engineer with semi-automated support for configuring a reasoning system. Hence, a description language for these reasoning components (i.e., PSMs) must provide human-understandable, high-level descriptions, which should also be grounded on a formal representation, to allow automated support by the broker. To this purpose we have developed the *Unified Problem-Solving Method Development Language, UPML* [Fensel et al., 1999b]. UPML is a software architecture for knowledge-based systems providing *components*, *adapters* and a configuration (called *architectural constraints*) of how the components should be connected using the adapters. Finally *design guidelines* specify how to develop a system constructed from the components and connectors that satisfies the architectural constraints.

In knowledge engineering terms UPML provides a *meta-ontology* for describing knowledge-based systems. The different elements of a specification correspond to concepts of this ontology and the architectural constraints are axioms in this ontology.

In this paper we outline the main features of the approach we have taken to define a framework for knowledge sharing and reuse. In particular we illustrate the basic meta-ontology of UPML, its underlying architecture, support tools and development guidelines. Because of space constraints we can only provide a limited number of technical details. Hence, the paper is better seen as an overview report on the main issues we are facing and the solutions we are developing.

The paper is organized as follows. In Section 2, we will briefly sketch the overall structure of UPML. Then we will discuss the (meta-)ontology that can be used to formalize UPML. Section 4 introduces the architectural constraints of UPML and Section 5 shows various ways in which tools for developing, selecting, and combining PSMs can make use of the (meta-)ontology. Section 6 briefly mentions the development guidelines of UPML. Conclusions, related work and outlook are discussed in Section 7.

2 The Overall Structure of UPML

[Fensel et al., 1999a] introduce the four components types of a UPML specification:

- *Tasks* define the problems that should be solved by the KBS.
- *PSMs* define the reasoning process of a KBS in domain-independent terms.
- *Domain models* describe the domain knowledge of the KBS.
- *Ontologies* provide the terminology used in tasks, PSMs and domain definitions.

Each of these elements is described independently to enable the reuse of task descriptions in different domains, the reuse of PSMs across different tasks and domain, and the reuse of domain knowledge for different tasks and PSMs. Therefore,

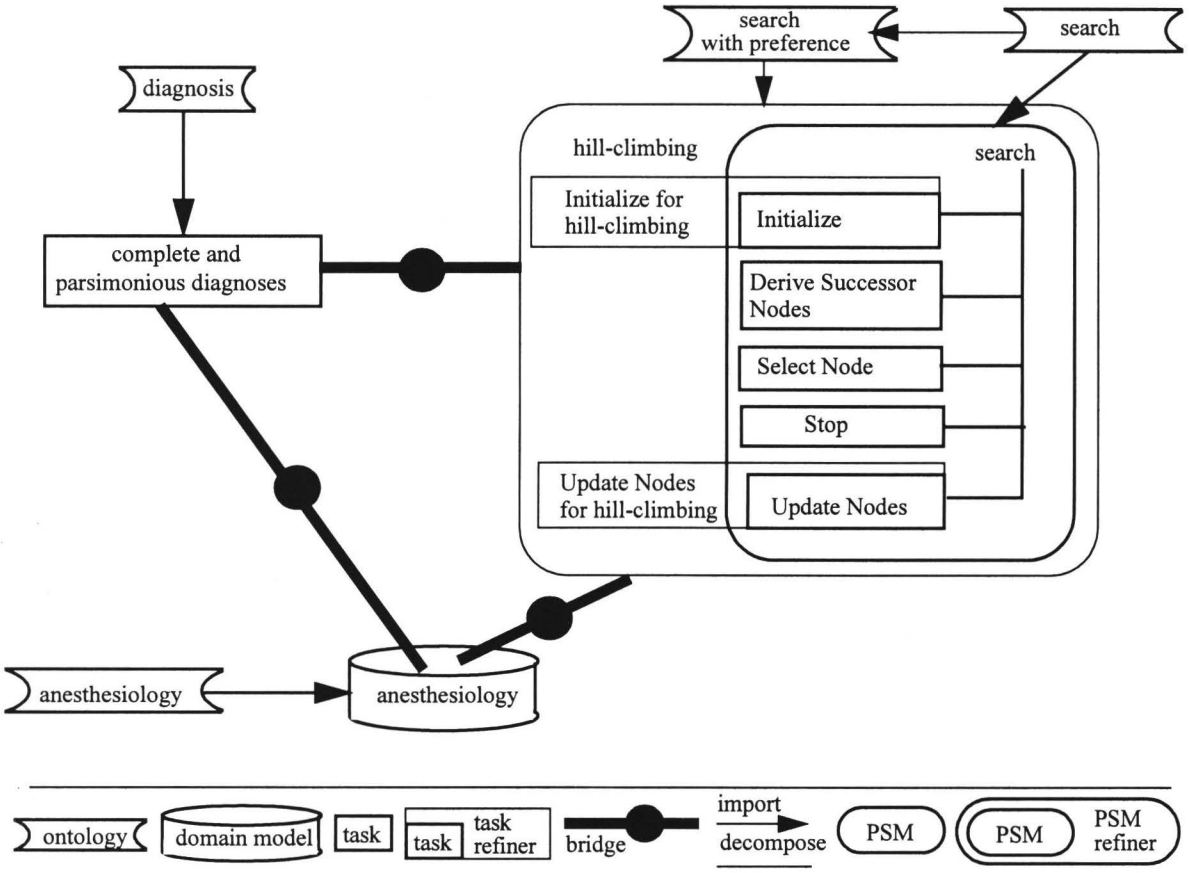

Fig 1. The overall structure of a UPML specification.

adapters are required to adjust the (reusable) parts to each other and to the specific application problem. UPML provides two types of adapters: *bridges* and *refiners*.

- *Bridges* explicitly model the relationships between two distinct parts of an architecture, e.g. between domain and task or task and PSM.
- *Refiners* can be used to express the stepwise specialization of a class of elements of a specification, e.g. a task is refined or a PSM is refined.

Very generic PSMs and tasks can be refined to more specific ones by applying a sequence of refiners (cf. [Fensel, 1997]). Again, separating generic and specific parts of a reasoning process maximizes reusability.

Together, the six UPML building blocks define a *software architecture*. The overall structure of a UPML specification is presented in Figure 1 (a more detailed discussion of the example can be found in [Fensel et al., 1999b]). A task called "complete and parsimonious diagnoses" is defined by importing an ontology called "diagnosis". The PSM applied to solve the task is "hill climbing". A bridge is required to connect the generic terminology of hill climbing with the diagnostic task: states and states transitions of the method have to be rephrased in terms of the task ontology.

Hill climbing is only one possible refinement of a generic search method that decomposes an entire search task into five more elementary subtasks: Initialize, Derive Successor Nodes, Select Node, Stop and Update Nodes. Hill climbing can be derived from this generic search method by (i) refining one of its subtasks (i.e., update node forgets all earlier nodes and only processes the currently derived successors further) and (ii) introducing a preference ordering.

PSM-mediated task decomposition and PSM specialization through a refiner are analogous to the ***part-of*** *and* ***is-a*** *constructs of knowledge representation formalisms.* Subtasking corresponds to the part-of construct because it decomposes a task into subtasks. The refinement of problem-solving methods, as introduced in [Fensel, 1997], corresponds to the is-a relationship of knowledge representation formalisms - e.g., Hill-climbing is a specialization of a general search method by refining some of its attributes (i.e., subtasks).

3 The Meta Ontology of UPML

We used PROTÉGÉ-II [Puerta et al., 1992] to develop a *meta* ontology of UPML. PROTÉGÉ-II is a knowledge acquisition tool-generator. After defining an ontology it semiautomatically generates a graphical interface for collecting the knowledge that is described by the ontology. The ontology can be described in terms of classes and

- Root
  - Competence
    - Competence CPSM
    - Competence Task
      - Competence PPSM
  - Cost expression
  - Formula
  - MCL-Program
  - Operational Description
  - Pragmatics
  - Renaming
  - Role
    - Input/Output Role
      - Input Role
      - Output Role
    - Intermediate Role
    - Knowledge Role
  - Signature
  - Specification Units
    - Bridge
      - PSM-Domain Bridge
      - PSM-Task Bridge
      - Task-Domain Bridge
    - Domain Model
      - Domain Refiner
    - Ontology
      - Ontology Refiner
    - PSM
      - Complex PSM
      - Primitive PSM
      - PSM Refiner
        - Complex PSM Ref.
        - Primitive PSM Ref.
    - Task
      - Task Refiner
  - UPML Specification

Fig 2. The class hierarchy of the UPML meta ontology.

attributes and organized with an is-a hierarchy and attribute inheritance. Viewing UPML as an ontology and structuring this ontology with the help of PROTÉGÉ-II, helped us to realize some missing elements in sub-specifications and we obtained a much clearer view of the overall structure of UPML. It turned out that the two adapter types, refiners and bridges are rather different entities in the ontology. Refiners are a subconcept of the specification element they refine, while bridges are a class by themselves. Figure 2 provides the class hierarchy of the UPML meta ontology. The organizational principle of the class hierarchy was to minimize the definitions of attributes, i.e. to maximize attribute inheritance. The definitions of the attributes of the class *task* and *task competence* are provided in Figure 3.

This ontology has been used to formulate architectural constraints, and to develop tools like editors, browsing, querying and reasoning services for UPML.

4. Architectural Constraints

Architectural constraints ensure well-defined components and composed systems. The conceptual model of UPML decomposes the overall specification and verification tasks into subtasks of smaller grainsize and clearer focus. The architectural constraints of UPML consist of requirements that are imposed on the intra- and interrelationships of the different parts of the architecture. They either ensure a valid part (for example, a task or a problem-solving method) by restricting possible relationships between its subspecifications or they ensure a valid composition of different elements of the architecture (for example, constraints on connecting a problem-solving method with a task). These architectural constraints can be formulated as axioms of the UPML meta ontology.

For example, we require the consistency of a task specification, i.e.

$\mathbf{T_1}$ *ontology axioms* $\cup$ *preconditions* $\cup$ *assumptions* must have a model.

Otherwise we would define an inconsistent task specification which would never be solved. In addition, it must hold:

$\mathbf{T_2}$ $\forall x_1,..,x_n$ (*ontology axioms* $\cup$ *preconditions* $\cup$ *assumptions* $\rightarrow \exists y_1,..,y_m$ *goal*)

That is, if the ontology axioms, preconditions, and assumptions are fulfilled by a domain and the provided case data then the goal of a task must be achievable. This constraint ensures that the task model makes the underlying assumption of a task explicit. For example, when defining a global optimum as a goal of a task it must be ensured that a preference relation exists and that this relation has certain properties. For example that $x < y$ and $y < x$ (i.e., reflexivity) is prohibited because otherwise the existence of a global optimum cannot be guaranteed.

These are the two architectural constraints UPML imposes to guarantee well-defined task specifications. A third optional constraint ensures minimality of assumptions and preconditions (called *weakest preconditions* in software engineering) and therefore maximizes the reusability of the task specification. It prevents overspecialization of assumptions and preconditions (i.e., it ensures that a task is not applied to an unsuitable domain).

$\mathbf{T_3}$ $\forall x_1,..,x_n \exists y_1,..,y_m$ (*goal* $\rightarrow$ *ontology axioms* $\cup$ *preconditions* $\cup$ *assumptions*)

Similar constraints have been developed for the other components. Correct relationships between components are formalized as axioms concerning the relevant bridges. For

Task[
  pragmatics → Pragmatics
  ontologies → Ontology
  import → Task
  competence → Competence Task]

Competence Task[
  goal → Formula
  precondition → Formula
  roles → Input/Output Role
  assumptions → Formula]

Fig 3. The attributes of the UPML meta ontology.

example

TPB$_2$

renaming(*PSM preconditions* ∪ *PSM assumptions* ∪ *PSM postconditions*) ∪
task ontology ∪ *task preconditions*∪*task assumptions*∪
bridge ontology ∪ *mappingaxioms*∪ *bridgeassumptions*
→ *task goal*

ensures that the goal of the task is fulfilled by the postcondition of the selected method. Further axioms can be found in [Fensel et al., 1999b].

5. Tool Support

We used PROTÉGÉ-II to implement an editor for UPML specifications. First PROTÉGÉ-II helps to define an ontology (in our case the meta ontology of UPML). In a second step it automatically derives an editor from it that requires some human interaction to derive a suitable tool from it.

The output of the UPML editor delivers files of the ontology and UPML specifications in a lisp-like syntax. We implemented a tool that translates these files into *Frame Logic* [Kifer et al., 1995]. The reason for this is to be able to use *On2broker*[1] as a browsing and query interface for UPML specifications. On2broker (cf. [Fensel et al., 1998]) is an advanced tool for browsing and querying WWW information sources. It provides a hyperbolic browsing and querying interface for formulating queries, an inference engine used to derive answers, and a webcrawler used to collect the required knowledge from the web. Figure 4 provides the hyperbolic presentation of the UPML meta ontology: classes in the center are depicted with a large circle, whereas classes at the border of the surrounding circle are only marked with a small circle. The visualization technique allows a quick navigation to classes far away from the center as well as a closer examination of classes and their vicinity. The structure of the frame-based representation language is used to define a tabular querying interface that frees users from typing logical formulas (see Figure 4). When a user selects a class from the hyperbolic ontology view, the class name appears in the class field of the tabular query interface and the user can select one of the attributes from the attribute choice menu because the pre-selected class determines the possible attributes. The discussed tool set is implemented in Java and available via the WWW.

However, typical UPML queries may be more complex. For example, Figure 5 shows parts of an On2broker's answer.to a complex query which asks for all attribute values of a task specification. Such queries are closer to short logical programs than to typical database queries. To ameliorate this problem we have used the UPML meta-ontology to define generic query patterns such as the one shown in Figure 5 which are instantiated for specific queries. Moreover, because the query interface is implemented as a Java Remote Method Invocation (RMI) Server.

1. http://www.aifb.uni-karlsruhe.de/www-broker.

6. Development Guidelines

Design guidelines define a process model for building complex KBSs out of elementary components. In general, the guidelines of UPML fall into three categories (cf. [Fensel et al., 1999b]):

- How to develop an application system out of reusable components. Guidelines describe the sequence and interrelationships of component selection and adaptation in developing an application system.
- How to develop a library of reusable task definitions and problem-solving methods. A three dimensional design space with predefined transition types provides structured support in developing and refining PSMs according to algorithmic paradigms, task terminologies and assumptions on domain knowledge.
- Which components of UPML correspond to an implementation and how can such components be implemented in an object-oriented framework. We developed and refined some *Design Patterns* that guide this translation process and defined certain interface guidelines.

7. Conclusions

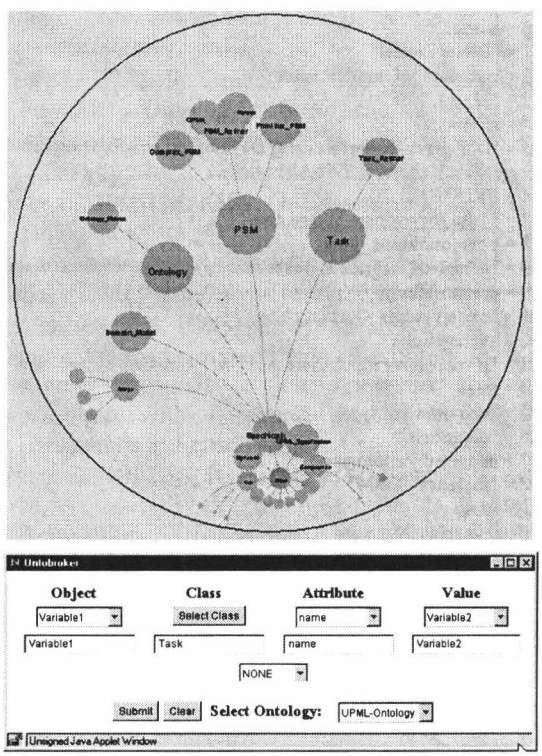

Fig 4. Hyperbolic Query Interface of On2broker.

FORALL $V_1,V_2,V_3,...,V_{39},V_{40}$ <-
V_{39}:UPML_Specification[V1->>V2] &
V_2:Task[name->>"Complete and parsimonious diagnosis"] &
V_2:Task[name->>V_4] & V_2:Task[pragmatics->>V_{32}] &
V_2:Task[V_5->>V_{32}] & V_{32}:Pragmatics[explanation->>V_9] &
V_{32}:Pragmatics[V_8->>V_9] & V_{32}:Pragmatics[V_{10}->>V_{11}] &
V_{32}:Pragmatics[author->>V_{11}] &
V_{32}:Pragmatics[last_date_of_modification->>V_{13}] &
V_{32}:Pragmatics[V_{12}->>V_{13}] & V_{32}:Pragmatics[V_{14}->>V_{15}] &
V_{32}:Pragmatics[reference->>V_{15}] &
V_{32}:Pragmatics[URL->>V_{17}] & V_{32}:Pragmatics[V_{16}->>V_{17}] &
V_2:Task[imported_ontologies->>V_{28}] &
V_2:Task[competence->>V34] &
V_{34}:Competence[postconditions_goal->>V_{35}] &
V_{34}:Competence[V_{18}->>V_{35}] & V_{35}:Formula[formula->>V_{19}] &
V_{34}:Competence[preconditions->>V_{36}] &
V_{34}:Competence[V_{20}->>V_{36}] & V_{36}:Formula[formula->>V_{21}] &
V_{34}:Competence[assumptions->>V_{37}] &
V_{34}:Competence[V22->>V_{37}] &V_{37}:Formula[formula->>V23] &
V_{34}:Competence[roles->>V_{38}]&V_{34}:Competence[V24->>V_{38}] &
V_{38}:Input_Role[name->>V_{25}] &V_{34}:Competence[roles->>V_{40}] &
V_{40}:Output_Role[name->>V_{26}] & V_2:Task[V_{27}->>V_{28}] &
V_{28}:Ontology[name->>V_{30}] & V_{28}:Ontology[V_{29}->>V_{30}].

Ontobroker found the following:

V1 = task
V2 = "instance_00003"
V4 = "Complete and parsimonious diagnosis"
V5 = pragmatics
V8 = explanation
V9 = "The task asks for a complete and minimal diagnoses"
V10 = author
V11 = "Dieter Fensel"
V12 = last_date_of_modification
V13 = "May 2, 1998"
V14 = reference
V15 = "D. Fensel: Understanding, Developing and Reusing Problem-Sol
Universitaet Karlsruhe, 1998."
V16 = URL
V17 = " http://www.psm-library.com "
V18 = postconditions_goal
V19 = "complete(diagnosis, observations) & parsimonious(diagnosis)"
V20 = preconditions
V21 = "observations <> empty"
V22 = assumptions
V23 = "If we receive input there must be complete hypothesis. observ
V24 = roles
V25 = "observations"
V26 = "diagnosis"
V27 = imported_ontologies
V28 = "instance_00004"
V29 = name
V30 = "diagnoses"
V32 = "instance_00046"
V34 = "instance_00150"
V35 = "instance_00153"
V36 = "instance_00154"
V37 = "instance_00155"
V38 = "instance_00151"
V39 = "instance_00001"
V40 = "instance_00152"

Fig 5. Querying the specification of a task.

Ontologies were introduced as means to support knowledge sharing and reuse (cf. [Gruber, 1993]). UPML provides an ontology for sharing and reusing knowledge-based systems. That is, it provides a (meta-)ontology for describing tasks, PSMs, domain models, ontologies, their mappings via bridges and refiners that express adaptation of the components. A number of tools have been developed and configured for supporting the definition and use of UPML components. A PROTÉGÉ-II based editor enables the development of UPML descriptions, while On2broker can be used to browse and query such descriptions. In addition, the IBROW broker matches user requirements with knowledge components specified in UPML and supports their distributed execution.

Related Work

UPML is close in spirit to CML which has been developed in the CommonKADS project (cf. [Schreiber et al., 1994]). CML provides a layered conceptual model of KBS by distinguishing between domain, inference, and task layers according to the CommonKADS model of expertise. UPML took this model as a starting point, but refined it according to the component-oriented style of software architectures. UPML decomposes a knowledge-based system - via an architecture - into a set of related elements: tasks, problem-solving methods, domain models and bridges that define their relationships. CML does not provide task-independent specification of problem-solving methods nor the encapsulation mechanism of UPML for problem-solving method. In UPML, the operational specification of a method is an internal aspect that is externally described by the competence of the method. In addition, CML does not provide means to refine tasks and problem-solving methods. In general, UPML is much more oriented to problem-solving method reuse (i.e., component reuse) than CML. Finally, CML is a semiformal language whereas UPML can be used as a semiformal language (using its structuring primitives) and as a formal language (UPML provides logical formalisms to formally define the elementary slots).

UPML has also many similarities with other standardization efforts in the area of knowledge-based systems. OKBC [Chaudhri et al., 1998] jointly developed at SRI International and Stanford University, provides a set of functions that support a generic interface to underlying frame representation systems. The *Knowledge Interchange Format* [KIF] is a computer-oriented first-order language for the interchange of knowledge among disparate programs. [KQML] or the *Knowledge Query and Manipulation Language* is a language and protocol for exchanging information and knowledge. KQML can be used as a language for an application program to interact with an intelligent system or for two or more intelligent systems to share knowledge in support of cooperative problem solving. The distinctive feature of UPML is that it is about sharing and exchange of problem-solving methods,

i.e. software components that realize complex reasoning tasks of knowledge-based systems. Moreover, UPML is less a standardization *formalism* than a standard *architecture*, which is defined by its meta-ontology. A similar approach is taken in Ontolingua [Gruber, 1993], which defines a meta-ontology for describing frame-based ontologies. Although UPML aims for a much broader scope, Ontolingua could be used at the object level of UPML for describing the elementary attribute values of specifications.

Finally, [Fensel et al., 1999b] put UPML in the general context of software architectures and also sketch how it can be translated into UML.

Outlook

An important issue concerns the integration of components subscribing to different ontologies. This integration can be specified by means of bridges. Bridges can either be defined by hand (i.e. by library providers or application developers) or can be generated automatically by an intelligent broker (cf. [Benjamins et al., 1999]). However, the automatic generation of bridges requires that the library providers do not only agree on a shared language (i.e., UPML) but also on a shared vocabulary, i.e., they do not only have to use the same UPML meta-ontology but also partially the same ontology at the object level.

Acknowledgment. We thank Igor Becker, Stefan Decker, Mauro Gaspari, John Gennari, Rix Groenboom, William Grosso, Frank van Harmelen, Mark Musen, John Park, Rainer Perkun, Enric Plaza, Guus Schreiber, Rudi Studer, Annette ten Teije, and Andre Valente for valuable comments on early drafts of the paper.

References

[Benjamins & Fensel, 1998] V. R. Benjamins and D. Fensel: Special issue on problem-solving methods of the *International Journal of Human-Computer Studies (IJHCS)*, 49(4), 1998.

[Benjamins et al., 1998] V. R. Benjamins, E. 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 World-WideWeb. In *Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW'98)*, Banff, Canada, April 18-23, 1998.

[Benjamins et al., 1999] V. R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel: Brokering Problem-Solving Knowledge at the Internet. In *Proceedings of the European Knowledge Acquisition Workshop (EKAW-99)*, D. Fensel et al. (eds.), Lecture Notes in Artificial Intelligence, Springer-Verlag, May 1999.

[Breuker & van de Velde, 1994] J. A. Breuker and W. van de Velde: *CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, The Netherlands.

[Chaudhri et al., 1998] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice: *Open Knowledge Base Connectivity 2.0*, Knowledge Systems Laboratory, KSL-98-06, January 1998. http://www-ksl-svc.stanford.edu:5915/doc/project-papers.html

[Fensel, 1997] 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.

[Fensel et al., 1998] D. Fensel, S. Decker, M. Erdmann, and R. Studer: Ontobroker: The Very High Idea. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibal Island, Florida, USA, 131-135, May 1998.

[Fensel et al., 1999a] D. Fensel, V. R. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Motta, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: The Component Model of UPML in a Nutshell. In *WWW Proceedings of the 1st Working IFIP Conference on Software Architectures (WICSA1)*, San Antonio, Texas, USA, February 1999.

[Fensel et al., 1999b] D. Fensel, E. Motta, V. R. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: *The Unified Problem-solving Method development Language UPML*. ESPRIT project number 27169, IBROW3, Deliverable 1.1, Chapter 1. http://www.aifb.uni-karlsruhe.de/WBS/dfe/publications99.html.

[Gruber, 1993] T. R. Gruber: A Translation Approach to Portable Ontology Specifications, *Knowledge Acquisition*, 5:199—220, 1993.

[KIF] KIF: http://logic.stanford.edu/kif/kif.html.

[Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, *Journal of the ACM*, vol 42, 1995.

[KQML] KQML: http://www.cs.umbc.edu/kqml/.

[Motta & Zdrahal, 1998] E. Motta and Z. Zdrahal: An approach to the organization of a library of problem solving methods which integrates the search paradigm with task and method ontologies. In [Benjamins & Fensel, 1998].

[Puerta et al., 1992] A. R. Puerta, J. W. Egar, S. W. Tu, and M.A. Musen: A Multiple-method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-acquisition Tools, *Knowledge Acquisition*, 4(2):171—196, 1992.

[Schreiber et al., 1994] 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.