Specifications of Knowledge Components for Reuse

Enrico Motta[1], Dieter Fensel[2], Mauro Gaspari[1] and Richard Benjamins[3]

[1]Knowledge Media Institute,
The Open University,
Walton Hall,
Milton Keynes, UK.
e.motta@open.ac.uk,
gaspari@cs.unibo.it[#]

[2]AIFB Institute,
University of Karlsruhe,
D-76128 Karlsruhe,
Germany.
dfe@aifb.uni-karlsruhe.de

[3]Social Science Informatics,
University of Amsterdam,
Roetersstraat 15, 1018 WB,
Amsterdam,
The Netherlands.
richard@swi.psy.uva.nl

Abstract

The IBROW project aims to support semi-automatic configuration of intelligent problem solvers out of reusable components. The project is developing solutions for the various types of technologies required to make reuse both technically and economically feasible. These technologies include innovative software architectures, modelling languages, software libraries and brokering agents. In this paper we focus on one particular aspect of the IBROW project: the specification of reusable library components. In particular, we illustrate a test case in which a pre-existing library of reusable components for parametric design is reformulated in terms of the framework and constructs provided by the IBROW modelling language. The exercise shows the advantages in terms of reusability and usability afforded by the IBROW approach. The proposed framework and language provide an effective organization for constructing libraries with large *horizontal cover*, thus maximizing reusability and avoiding the brittleness of traditional, monolithic libraries.

1. Introduction

Effective approaches to software reuse need to tackle a whole range of technical problems, which span software architectures, specification languages, software libraries and supporting tools. The IBROW project [3] aims to support the semi-automatic configuration of intelligent problem solvers out of reusable components. The project takes a holistic approach to reuse and is developing solutions for the various types of technologies required to make reuse both technically and economically feasible. These technologies include an innovative software architecture [10], the UPML modelling language [10], software libraries [21], brokering agents [2] and methodologies [10]. In this paper we focus on one particular aspect of the IBROW project: the specification of reusable library components. In particular, we will illustrate in some depth a test case in which a pre-existing library of reusable components for *parametric design* tasks [19, 20] was reformulated in terms of the architecture and constructs provided by the UPML modelling language. The purpose of this exercise is to show the advantages, in terms of reusability and usability, afforded by our approach. Specifically, we will show that the proposed architecture and language provide an effective organization for constructing libraries with large *horizontal cover*, thus maximizing reusability and avoiding the brittleness of traditional, monolithic libraries. Here we use the term "horizontal cover" to refer to the range of problem solving behaviors supported by a library.

We begin by providing an overview of the project. We will then present the library which provided the input to our test case. In section 4 we illustrate the rational reconstruction of the input library according to the UPML framework. Finally, in sections 5 and 6 we discuss related work and highlight the main conclusions from this study.

2. Overview of the IBROW project

2.1 Reuse-centered system development

The scenario envisaged by the IBROW project is one in which knowledge engineers construct intelligent systems

[#] Mauro Gaspari's contribution was carried out when visiting the Knowledge Media Institute at the Open University. His regular affiliation is with the University of Bologna, Dipartimento di Scienze dell' Informazione, Via Mura Anteo Zamboni 7, 40127, Bologna, Italy.

through a *plug&play* development process. This involves locating the appropriate components in libraries available over the World-Wide-Web and then configuring them into a complete problem solver. This process is to be supported by a specialised *software broker*[2], whose job is to acquire a *detailed task specification* from the user and then to identify an appropriate *problem solving method* (PSM), which is applicable to the given task. Given the complexity of a typical top-level task in a knowledge-based system (KBS) - consider for instance a configuration design or planning problem - this method-to-task selection and configuration process is not just a one-shot activity, but it involves decomposing a task into a number of subtasks and then recursively locating and configuring the appropriate sub-methods applicable to these more specific subtasks. Hence, the overall problem solving structure of the target system can be characterized as a *task-method structure*, where problem solving methods either solve a task directly, or decompose a task into a number of subtasks.

A detailed task specification is produced by instantiating a *generic task model* in a particular application domain. A generic task model provides a formal description of a class of problems - e.g., configuration design, classification, or fault diagnosis, which is independent from a specific application or domain. For example, modelling an office allocation application might involve selecting a generic task model for *parametric design* [19, 20] and then instantiating it in terms of the concepts in the office allocation domain - i.e. employees, offices, etc. This process can be characterised as imposing a particular *task viewpoint* over a domain. In general, several task viewpoints can be imposed on the same application domain.

Tasks and domains provide two of the basic epistemological building blocks of the IBROW approach to reuse. The third important class of components is given by *problem solving methods*. These are domain-independent specifications of generic problem solving behaviours which are reusable across domains and (possibly) generic tasks. Problem solving methods are characterised in terms of the *epistemological requirements* they impose on an application domain. Some problem solving methods, for instance *Generate&Test* or *Local Search*, are generally applicable methods which introduce relatively *weak* requirements on the underlying domain knowledge. In other words, while they are generically reusable, they provide little task-specific leverage. Other problem solving methods - e.g., model-based diagnostic methods [6] - are called *strong methods*, as they introduce significant assumptions on the available domain knowledge - e.g., the availability of a device model.

Generic tasks, domain models and PSMs normally subscribe to different terminologies, which have to be integrated when building an application by reuse. For instance, solving an office allocation problem might involve integrating a generic task model for parametric design with a task-independent domain model of an organisation and then applying a task-independent characterization of a *Propose&Revise* problem solver [11] to the resulting task specification. Two important challenges naturally arise in this scenario: i) how to formally specify the terminology used by a particular class of components - e.g. a task model for parametric design - and ii) how to model the connections between components subscribing to different terminologies. In IBROW we use *ontologies* [14] to model the former and *adapters* [9] to model the latter. Ontologies and adapters are described in the next two sections.

2.2 Ontologies in the IBROW Architecture

An ontology provides *a partial specification of a shared conceptualisation, to be used for formulating knowledge-level theories about a universe of discourse* [14, 15, 19]. Thus, the role of an ontology is to provide a common vocabulary which can be shared by different agents. Because an ontology is concerned with terminology, then it does not model implementation aspects; in other words, it is a *knowledge-level* formalisation [22]. Typically, ontologies found in the literature are *domain ontologies*: they capture (domain) knowledge about the world, characterised in a use-independent style. In IBROW we use ontologies also to characterize the terminology associated with tasks and PSMs. For instance, an ontology specifying the terminology of parametric design tasks will include concepts such as parameter, parameter value, constraint and requirement [19, 11]. Such an ontology is called a *task ontology*. Similarly we can define *method ontologies* to define the terminology associated with a problem solving method. For example, Propose&Revise can be characterised in task and domain-independent terms in terms of *states*, *state transitions*, *procedures* and *fixes* [11].

The conceptual separation between task, method and domain ontologies maximizes reusability: a problem solving method can be applied to different task models in different domains. At the same time, it introduces the need for special-purpose modelling constructs for integrating heterogeneous components. These are discussed next.

2.3 Adaptation for Reuse

As discussed in the previous section, we maintain a strong epistemological distinction between domains, generic tasks and problem solving methods, which is reflected in

the use of different ontologies for the different types of components. Hence, building an application requires configuring and integrating heterogeneous components by means of *adapters*. Integrating explicitly defined adapters in a library and/or in the application development process affords a number of advantages:

- PSMs can be reused for different tasks and domains.
- Reusable adapters can be defined - e.g. to provide 'standard' adaptation of search methods for design tasks.
- The combination of generic methods and domain-oriented or task-oriented adapters can be integrated in a library to define highly usable (i.e. specific) PSMs.

In general, two types of adapters are required to configure an application out of reusable components: *bridges*, to connect different parts of a model - e.g., mapping a task viewpoint onto a domain - and *refiners*, to model the process by which an element is defined as a specialization of an existing one. Examples of refiners will be given in section 4.3; bridges are discussed in detail in [10].

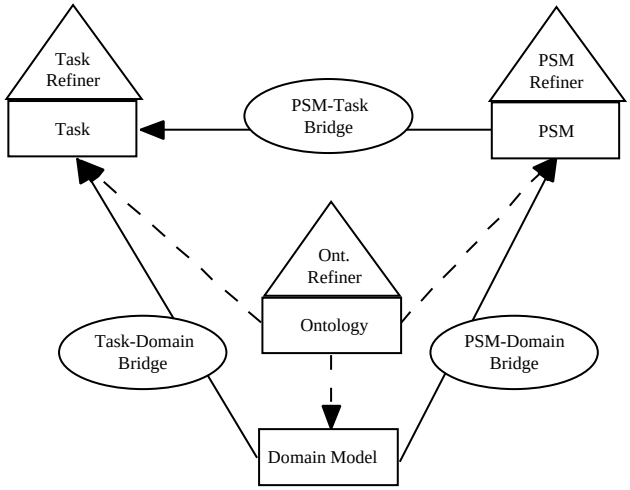

Figure 1. The IBROW architecture for knowledge-based systems.

To summarize, the IBROW framework comprises six classes of components: ontologies, tasks, methods, domain models, bridges and refiners - see figure 1. In section 4 we will show how this framework can be applied to drive the development of a library of problem solving components and we will highlight the resulting benefits. First, we introduce the library of parametric design components - henceforth, "the MZ library" - which provided the input to our test case.

3. A task-specific library of knowledge components

The MZ library, which was developed by Enrico Motta and Zdenek Zdrahal [19, 20], aims to reconcile the advantages in terms of knowledge acquisition and reuse afforded by task-specific formulations with the clear theoretical foundations and problem generality provided by task-independent problem solving paradigms, such as search. Specifically, the architecture of the MZ library relies on three key ideas:

- Like in IBROW, different kinds of formal ontologies are used to specify the generic structure of a class of problems (task ontologies) and the knowledge requirements of problem solving methods (method ontologies).
- All PSMs applicable to a class of problems, say P, are characterized as refinements of a common, task-specific, but method-generic *problem solving model*. This model comprises a set of generic problem solving components, (sub-)tasks and (sub-)methods, which provide the high-level building blocks necessary to construct PSMs applicable to P. The model is associated with a *generic method ontology*, which expresses the minimal knowledge requirements which have to be satisfied by a PSM applicable to P.
- In order to bridge the gap between the method and task 'dimensions' and to provide a task-independent foundation to a task-specific library of problem solving methods, the construction of the generic problem solving model discussed in the previous bullet, say M, is driven by a task ontology and by the selection of a *generic problem solving paradigm.* In particular the MZ library makes use of the search paradigm. The advantage of this choice is that it does not constrain the range of problem solving methods which can be modelled as specializations of the model M - i.e. all problem solving methods can be seen as performing search.

Figure 2 provides a diagrammatic representation of the architecture and process model underlying the MZ library. First, a task ontology for parametric design was developed. Then, a task-specific problem solving model and a method ontology were produced by integrating a model of problem solving as search with the parametric design task ontology. Finally, several problem solving methods were defined as specializations of the generic problem solving model. The advantage of this approach is

that the strong organizational structure provided by the task-specific problem solving model makes it possible to define new PSMs as relatively simple refinements/configurations of the generic problem solving model: typically only few components are needed to define new PSMs - on average six or seven. Moreover, the resulting PSMs are characterized in a homogeneous style - i.e. they share the same high-level components and inherit a common control structure from the generic problem solving model. This approach makes it easier both to compare and contrast PSMs and to define 'hybrid' PSMs by integrating components from pre-existing PSMs - see [19] for more details.

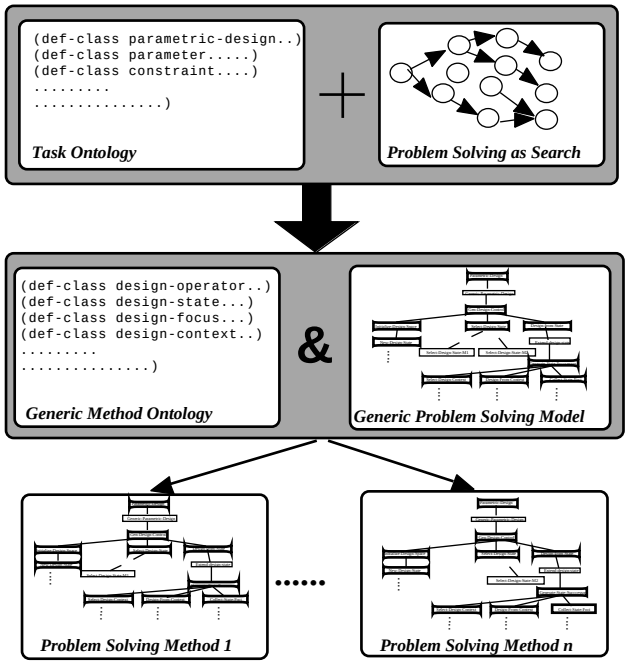

Figure 2. Problem solving foundation of the MZ library.

A listing of the main tasks included in the generic model of parametric design problem solving is given in table 1. This suite of high-level tasks allowed us to model several classes of parametric design PSMs. These include: case-based design methods; design PSMs based on search algorithms, such as hill-climbing and A*; several variations of Propose&Revise and other PSMs drawn from the design and scheduling literature, including *Propose&Exchange* [25] and *Propose&Backtrack* [26]. More details can be found in [19, 20, 30, 31].

The MZ library has been tested successfully on several application domains, including a number of real-world engineering design applications [28]. The results show that this technology leads to significant improvements in the performance of the design process (typical improvement by a factor 10).

Parametric Design	Extend design
Generic Design Control	Collect design foci
Design from State	Design from context
Initialize Design Space	Select design focus
Select Design State	Design from focus
Evaluate Design State	Collect focus operators
Evaluate feasibility	Order focus operators
Evaluate cost	Select design operator
Evaluate consistency	Try design operator
Evaluate completeness	Apply design operator

Table 1. Main tasks in parametric design model.

4. Developing reusable components in UPML

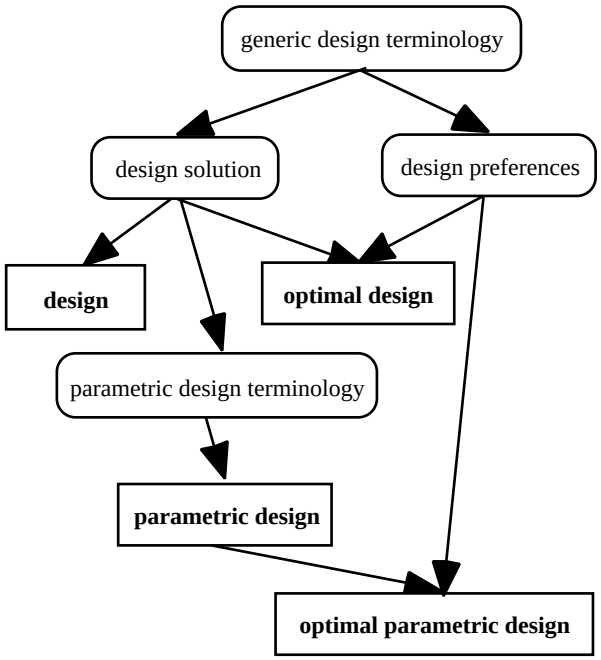

Figure 3. Hierarchy of ontologies and task specifications in the UPML library.

As discussed in the previous section, the MZ library is an extensive and established resource for building parametric design components. However, while it provides strong support for developing parametric design applications, reusing it for other problem types requires considerable re-engineering. Of course, this is hardly surprising: this library was actually designed to be task-specific. However, given that the library is conceptually based on the search paradigm, it is clearly unnecessary that the entire library is task-specific: the reusability of the library could be greatly enhanced by making explicit in the library itself the methodological process which led to its development. Specifically, what is required here is i) to separate the specification of the search component from that of the parametric design task ontology and ii) to include explicitly in the library the connection between

these two models. In what follows we describe this 'rational reconstruction' process, thus illustrating the IBROW approach to library development.

4.1 Introducing modularity in the specification

```
ontology design-solution
  pragmatics
    Defines the predicates and axioms needed to describe
    solutions to design tasks
  import generic-design-terminology;
  signature
    predicates
    admissible_solution: Design_Model x Constraints x
    Requirements;
    consistent: Design_Model x Constraints;
    suitable: Design_Model x Requirements;
    variables
    ?d: Design_Model; ?r,: Requirement; ?c: Constraint;
    ?rs: Requirements; ?cs: Constraints;
  end signature
  axioms
    /* A design model is an admissible solution iff it is
    consistent wrt the problem's constraints and it is suitable
    wrt the problem's requirements. */
    admissible_solution (?d, ?cs, ?rs) ↔
    consistent (?d, ?cs)    suitable (?d, ?rs);
    /* A design model is consistent with respect to a set of
    constraints iff none of them is violated. */
    consistent (?d, ?cs) ↔ ¬    ?c (?c    ?cs
                                ?c    violated_statements (?d));
    /* A design model is suitable iff all the requirements are
    satisfied. */
    suitable (?d, ?rs) ↔    ?r (?r    ?rs →
                                ?r    satisfied_statements (?d));
end ontology
```

Figure 4. Ontology specification in UPML.

One of the main modules of the MZ library specifies a task ontology for parametric design. When reconstructing this module in UPML, we decided to decouple the various parts of this ontology, to make it more reusable and to tease out the various ontological commitments. For instance, separating optimality aspects from the rest of the ontology ensures that i) simple parametric design applications do not need to instantiate optimality requirements and ii) the same generic optimality machinery can be used in the specification of other generic tasks - e.g., planning tasks. Another way in which we generalized the parametric design task ontology given in the MZ library was by defining it as a refinement of the specification of a generic design task. The overall organization of the task-oriented part of the resulting UPML library is shown in figure 3. The figure shows that what had been defined as a single ontology/task specification in the MZ library has now been subdivided into four ontologies and four task specifications (the latter are shown as rectangles with the text in bold). This organization allows us to have multiple task specifications while at the same time minimizing the number of definitions in each particular one. For instance, task specifications which do not require optimality criteria do not inherit the optimality machinery from ontology *design-preferences*.

4.2 Modelling Ontologies and Tasks

An UPML ontology is defined by providing a signature and the axioms associated with the elements in the signature. The specification can be expressed either in a sorted logic - which is close to the specification style of $(ML)^2$ [16] and MLPM [7] - or in Frame-Logic [17]. In the examples shown in this paper we have used sorted logic. Examples of the use of frame logic in UPML can be found in [10]. To illustrate the style of the specification we show in figure 4 part of the ontology *design-solution*. This ontology formally specifies what constitutes a solution to a design task.

As shown in figure 5, a task is specified by providing its *input* and *output roles*, its *goal*, and its *preconditions* - see [10] for more details.

```
task parametric-design
  ontology
    import parametric-design-terminology;
  specification
    roles
    input
      reqs: Requirements; cs: Constraints;
      params: Parameters; vrs: Value_Ranges;
    output
      design: Parametric_Design_Model;
    end roles
    goal
    parametric_solution (design, cs, reqs, params, vrs);
    preconditions
    /* The parameter set should not be empty */
      ?p (?p    params);
    /* Each parameter is associated to a value range*/
      ?p (?p    params →    ?vr
            (?vr    vrs    ?vr = <?p, ?vs>));
end task
```

Figure 5. Definition of task *parametric design.*

4.3 Specifying problem solving behavior

In this section we illustrate how the model of parametric design problem solving included in the MZ library was reconstructed in the UPML library, to provide a more reusable resource than the original one.

As shown in figure 6, the monolithic task-specific model of parametric design problem solving defined in the MZ

library has now been factorized into a number of PSM specifications and ontologies[1]. In particular, we now provide a distinct specification of search problem solving, which is shown in figure 7. Thus, we make explicit in the UPML library the search foundation of the MZ library.

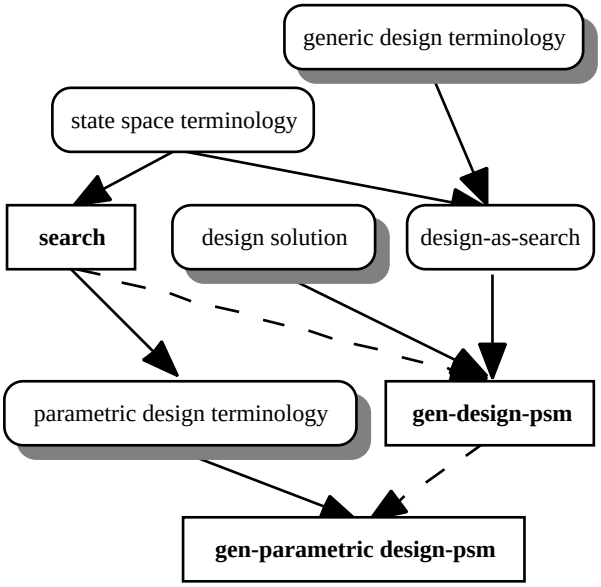

Figure 6. Ontology inclusion and PSM refinement in the UPML library.

```
problem solving method search
   ontology state-space-terminology;
   competence
     roles
     /* The input is an initial state*/
     input state;
     output state;
     preconditions
       input ≠
     subtasks
       initialize, select_state, derive_successor_states,
       update_state_space;
     postconditions
     /* The output is either a solution state or the
     empty set, if the PSM fails */
       solution_state (output);
     assumptions
     /*To guarantee that this PSM succeeds we introduce the
     assumption that a solution state exists and can be reached
     from the input state */
          ?s. solution_state (?s)    successor (input, ?s)
end problem solving method
```

Figure 7. Specification of a search PSM.

[1] The shadows indicate the ontologies already present in figure 3.

```
PSM refiner search → gen-design-psm
   pragmatics
     This refiner specialises the generic search PSM for design
     tasks. It refines the generic input to a search PSM to be a
     set of requirements and constraints; it replaces task
     initialize with the design-specific initialize_design_model
     and adds an axiom linking the  predicate solution_state to
     the design-specific predicate admissible_solution.
   ontology
     basic-design-as-search, design_solution;
   auxiliary terminology
     constants
     reqs: Requirements; cs: Constraints;
     design: Design_Model
   competence
     refined roles
       input reqs, constraints;
     refined preconditions
       reqs ≠
     refined-subtasks
     /* we specialise subtask initialize by replacing it with
     subtask initialize_design_models */
     initialize → initialize_design_model;
     axioms
       ?s solution_state (?s) ↔
     admissible_solution (?s, reqs, constraints);
end PSM refiner
```

Figure 8. Defining a generic design problem solver as a refinement of search.

A problem solving method is defined in UPML as a logical specification characterized by input and output roles, preconditions, *postconditions* and *assumptions*. The latter define the requirements that the domain knowledge has to fulfill in order to guarantee that the method succeeds. In this case the requirement is that a solution state must be reachable from the input state. In general, not all assumptions are necessarily provable for a particular domain. In some cases proving an assumption may require a reasoning effort as computationally expensive as solving the task in hand. Nevertheless, assumptions provide an important aspect of a problem solving method specification. They define the 'glue' that ensures that the complexity gap between a PSM and a task is heuristically bridged by the domain knowledge. Hence, they provide a way to formalize the informal notion of knowledge-based systems as heuristic problem solvers.

The ontology *design-as-search* fills the terminological gap between search and design problem solving. Having defined this ontology we can now specify a refiner to derive a generic design problem solver as a refinement of the search PSM - see figure 8. The important aspect of this refiner is that it links the generic notion of *solution state* in the search ontology to the design-specific notion of *admissible solution*. It also refines the input roles, by adding the design-specific notions of *requirements* and

constraints. Finally, it refines the generic subtask *initialize* with a design-specific *initialize-design-model*.

Using this style of specification we have reconstructed the generic model of parametric design included in the MZ library - see [21] for the full specification. The result is a more flexible and reusable resource, whose development process and problem solving foundations are made explicit in the library itself.

5. Related Work

A number of libraries of problem solving components have been proposed over the last decade or so in knowledge engineering. The earlier ones [5, 18] were simply collections of complete problem solving methods, providing strong but inflexible support. A certain degree of flexibility was introduced with second generation libraries [1, 4, 23], which were structured according to the task-method organization. As discussed in section 2.1 the basic principle of task-method structures is that, given a task, it is possible to find a number of methods which can be used to solve it. While this 'method-solves-task' association is adequate for the purpose of navigating a library and configuring a PSM, it does not, on its own, provide a strong enough organization model for developing a library. As a result it is quite difficult to get the task-method structure right [24]. In contrast with these approaches our library is based on a clear theoretical basis, given by a generic problem solving paradigm. Moreover, while adaptation is usually handled implicitly in second generation libraries, our approach considers adapters as first class library components, resulting in a more flexible, reusable and comprehensive resource [9]. The notion of adapter in our context plays a role similar to that of *mediators* in heterogeneous information systems [29], *connectors* in software architectures [27], and adapters in design patterns [13]. The idea underlying all of these approaches is essentially the same: some kind of 'external kit' is required in order to allow the interaction of reusable components and their configuration for different computational scenarios. The externalization of this adaptation process has the advantage that the original components remain unchanged, while they become usable in the new situation. Our use of adapters takes advantage of the task-specific architectures developed in knowledge engineering over the years. These architectures, e.g. the generic model of parametric design outlined in this paper, are not specific for a domain but generalize from classes of applications. This aspect is specific to knowledge engineering research and implies that our adapters tend to be rather powerful and potentially reusable. For instance, the adaptation of search for design can be used to refine different search methods for different types of design problem solving.

6. Conclusions

We have shown an approach to the development of a library of problem solving components for knowledge-based systems which arguably provides a number of advantages: it avoids the brittleness of traditional monolithic libraries; supports the development of usable, task-specific PSMs; maximizes reusability by allowing the integration in the library of general purpose, task-independent problem solving components and addresses the horizontal cover problem by making the component adaptation process explicit. In addition, our library is formally specified, thus opening the door to automatic support for verification and validation [12] and to semi-automatic configuration of PSMs [2].

Of course, enabling reuse requires more than just providing the right library. Hence, in the IBROW project we are developing a web-based software infrastructure which aims to reduce the costs associated with reuse [2]. We believe that the success of our enterprise (and in general of IBROW-like enterprises) is crucial to break the cost/skill barrier which is currently hampering the take-up of knowledge engineering technology.

References

[1] Benjamins, R. (1993). *Problem Solving Methods for Diagnosis*. PhD Thesis, Department of Social Science Informatics, University of Amsterdam.

[2] Benjamins, R., Wielinga, B. J., Wielemaker, J. and Fensel, D. (1999). Brokering Problem-Solving Knowledge on the Internet. To appear in the Proceedings of the *11th European Workshop on Knowledge Acquisition, Modeling, and Management (EKAW '99)*. Springer Verlag.

[3] The home page of the IBROW project is at URL: http://www.swi.psy.uva.nl/projects/IBROW3/home-ibrow.html.

[4] Breuker, J. A. and Van de Velde, W. (1994). *CommonKADS Library for Expertise Modelling*. IOS Press, Amsterdam, The Netherlands.

[5] Breuker J., Wielinga B.J., van Someren M., de Hoog R., Schreiber G., de Greef P., Bredeweg B., Wielemaker J., Billault J.-P., Davoodi M. and Hayward S. (1987). *Model Driven Knowledge Acquisition: Interpretation Models*. Deliverable D1, Esprit Project P1098, KADS. Un. of Amsterdam.

[6] Console, L. and Torasso, P. (1990). Hypothetical Reasoning in Causal Models. *International Journal of Intelligent Systems, 5(1)*, pp. 83-124.

[7] Fensel, D. and Groenboom, R. (1996). MLPM: Defining a Semantics and Axiomatization for Specifying the Reasoning Process of Knowledge-

based Systems. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)*, Budapest, August 1996.

[8] Fensel, D., Benjamins, V. R., Motta, E. and Wielinga, B. J. (1999). A Framework for knowledge system reuse. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99)*. Stockholm, Sweden, July 31 - August 5, 1999.

[9] Fensel, D. and Motta, E. (1998). Structured Development of Problem Solving Methods. *Submitted for publication*.

[10] Fensel, D., Motta, E., Benjamins, V. R., Decker, S., Gaspari, M., Groenboom, R., Grosso, W., van Harmelen, F., Musen, M., Plaza, E., Schreiber, G., Studer, R., ten Teije, A. and Wielinga, B. J. (1999). The Unified Problem-Solving Method Development Language, UPML. *ESPRIT project number 27169, IBROW3, Deliverable 1.1, Chapter 1*.

[11] Fensel, D., Motta, E., Decker, S. and Zdrahal, Z. (1997). 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.

[12] Fensel, D. and Schönegge, A. (1998). Inverse Verification of Problem-Solving Methods. *International Journal of Human-Computer Studies 49(4)*, pp. 339-361.

[13] Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995). *Design Patterns.* Addison-Wesley.

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

[15] Guarino, N. and Giaretta, p. (1995). Ontologies and Knowledge Bases: Towards a Terminological Clarification. In N. Mars (Ed.), *Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing.* IOS Press, pp. 25-32.

[16] van Harmelen, F. and Balder, J. R., (1992). $(ML)^2$: A Formal Language for KADS Models of Expertise. *Knowledge Acquisition, 4(1)*, pp. 127-161.

[17] Kifer, M., Lausen, G. and Wu, J. (1995). Logical Foundations of Object-Oriented and Frame-Based Languages. *Journal of the ACM, Vol. 42.*

[18] Marcus S. (Editor) (1988). *Automatic Knowledge Acquisition for Expert Systems.* Kluwer Academic.

[19] Motta E. (1997). *Reusable Components for Knowledge Models*. PhD Thesis. Knowledge Media Institute. The Open University. UK. Available from URL http://kmi.open.ac.uk/~enrico/thesis/thesis.html.

[20] Motta, E. and Zdrahal, Z. (1998). An approach to the organization of a library of problem solving methods which integrates the search paradigm with task and method ontologies. *International Journal of Human-Computer Studies 49(4)*, pp. 437-470.

[21] Motta, E., Gaspari, M. and Fensel, D. (1998). UPML Specification of a Parametric Design Library. *ESPRIT project number 27169, IBROW3, Deliverable 4.1*.

[22] Newell A. (1982). The knowledge level. *Artificial Intelligence, 18(1)*, pp. 87-127.

[23] O'Hara, K. (1995). The GDM Grammar, v.4.6. *VITAL Project Report NOTT/T252.3.3*. Available from the author at AI Group, Department of Psychology, University of Nottingham, UK.

[24] Orsvärn, K. (1996). Principles for Libraries of Task Decomposition Methods - Conclusions from a Case-study. In N. Shadbolt, K. O'Hara, and G. Schreiber (Editors). *Advances in Knowledge Acquisition - EKAW '96*. Lecture Notes in Artificial Intelligence, 1076. Springer-Verlag, pp. 48-65.

[25] Poeck, K. and Puppe, F. (1992). COKE: Efficient Solving of Complex Assignment Problems with the Propose-and-Exchange Method. *5th International Conference on Tools with Artificial Intelligence*. Arlington, Virginia.

[26] Runkel, J. T., Birmingham, W. B. and Balkany, A. (1996). Solving VT by Reuse. *International Journal of Human-Computer Studies 44 (3/4)*, pp. 403-433.

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

[28] Valasek M. and Zdrahal Z. (1997). Experiments with Applying Knowledge Based Techniques to Parametric Design. In A. Riitahuhta (editor), *Proceedings of the International Conference on Engineering Design - ICED 97*. Volume 1, pp. 277-280, Tampere University of Technology, Finland.

[29] Wiederhold, G. and Genesereth, M. (1997). The Conceptual Basis for Mediation Services, *IEEE Expert, September/October Issue*, pp. 38-47.

[30] Zdrahal Z. and Motta E. (1995). An In-Depth Analysis of Propose & Revise Problem Solving Methods. In B. R. Gaines and M. Musen (editors), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*. Banff, Canada.

[31] Zdrahal Z. and Motta E. (1996). Improving Competence by Integrating Case-Based Reasoning and Heuristic Search. In B. R. Gaines and M. Musen (editors), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW´96)*, Banff, Canada, 9-14 November, 1996.