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...
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 formalizing 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].

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.

Table 1. Main tasks in parametric design model.

<table>
<thead>
<tr>
<th>Parametric Design</th>
<th>Extend design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generic Design Control</td>
<td>Collect design foci</td>
</tr>
<tr>
<td>Design from State</td>
<td>Design from context</td>
</tr>
<tr>
<td>Initialize Design Space</td>
<td>Select design focus</td>
</tr>
<tr>
<td>Select Design State</td>
<td>Design from focus</td>
</tr>
<tr>
<td>Evaluate Design State</td>
<td>Collect focus operators</td>
</tr>
<tr>
<td>Evaluate feasibility</td>
<td>Order focus operators</td>
</tr>
<tr>
<td>Evaluate cost</td>
<td>Select design operator</td>
</tr>
<tr>
<td>Evaluate consistency</td>
<td>Try design operator</td>
</tr>
<tr>
<td>Evaluate completeness</td>
<td>Apply design operator</td>
</tr>
</tbody>
</table>

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).

4. Developing reusable components in UPML

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

```plaintext
ontologies 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 requiremets. */
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)[16] and MLP [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.

```plaintext
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
preconditions
/* The parameter set should not be empty */
∃ ?p (?p ∈ params);
/* Each parameter is associated to a value range */
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. 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.

The shadows indicate the ontologies already present in figure 3.

---

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

**Figure 7.** Specification of a search PSM.

---

The shadows indicate the ontologies already present in figure 3.

---

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


