THE COMPONENT MODEL OF UPML IN A NUTSHELL

Dieter Fensel[1], V. Richard Benjamins[2], Stefan Decker[1], Mauro Gaspari[7], Rix Groenboom[3], William Grosso[6], Mark Musen[6], Enrico Motta[4], Enric Plaza[5], Guus Schreiber[2], Rudi Studer[1], and Bob Wielinga[2]

[1] Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany
{dfe, sde, rst}@aifb.uni-karlsruhe.de, http://www.aifb.uni-karlsruhe.de/~dfe

[2] University of Amsterdam, Department SWI, NL-1018 WB Amsterdam, NL
{richard, schreiber, wielinga}@swi.psy.uva.nl

[3] University of Groningen, Department of Computer Science, P.O. Box 800,
NL-9700 AV Groningen, NL, rix@cs.rug.nl

[4] Knowledge Media Institute, The Open University, Walton Hall, Milton Keynes, UK
E.Motta@open.ac.uk

[5] IIIA - Institut d'Investigació en Intel·ligència Artificial,
CSIC - Spanish Scientific Research Council, Campus UAB, 08193 Bellaterra, Catalonia
enric@iiia.csic.es

[6] Knowledge Modeling Group at Stanford Medical Informatics,
Stanford University, 251 Campus Drive, MSOB X-215, Stanford, California, USA
{musen, grosso}@SMI.Stanford.Edu

[7] Department of Computer Science, University of Bologna, Italy
gaspari@cs.unibo.it

Abstract. Problem-solving methods provide reusable architectures and components for implementing the reasoning part of knowledge-based systems. The Unified Problem-solving Method description Language UPML has been developed to describe such architectures and components to facilitate their semiautomatic reuse and adaptation. This paper sketches the components and connectors provided by 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. Therefore PSMs are a special type of software architectures: software architectures for describing the *reasoning* part of KBSs. Several problem solving method libraries are now available. The IBROW[3] project (Benjamins et al., 1998) has been set up with the aim of enabling

semiautomatic reuse of PSMs. This reuse is provided by integrating libraries in a internet-based environment. A broker is provided that selects and combines PSMs of different libraries. A software engineer interacts with a broker that supports him in this configuration process. As a consequence, a description language for these reasoning components (i.e., PSMs) must provide human-understandable high-level descriptions with underpinned formal means to allow automated support by the broker. Therefore, we developed the *Unified Problem-Solving Method description Language UPML* (Fensel et al., 1998b). UPML is architectural description language specialized for a specific type of systems providing *components*, *connectors* and a configuration of how the components should be connected using the connectors (called *architectural constraints*). Finally *design guidelines* provide ways to develop a system constructed from the components and connectors that satisfies the constraints.

The content of the paper is organized as follows. In section 2, we will sketch the architectural framework that is provided by UPML: ontologies, tasks, domain models, PSMs, bridges, and refiners. Section 3 briefly mentions the architectural constraints, design guidelines and tool support. A case study where UPML is applied to a library of PSMs for parametric design is described in (Motta et al., 1998). Aspects of the underlying logic and its semantics as well as a more detailed discussion of UPML can be found in (Fensel et al., 1998b).

2. Components and Connectors

The UPML architecture for describing a KBS consists of six different elements (see Figure 1): tasks, domain models, PSMs, ontologies, bridges, and refiners. A *task* that defines the problem that should be solved by the KBS, a *PSM* that defines the reasoning process of a KBS, and a *domain model* that describes the domain knowledge of the KBS. Each of these elements is described independently to enable the reuse of task descriptions in different domains, the reuse of PSMs for different tasks and domain, and the reuse of domain knowledge for different tasks and PSMs. *Ontologies* provide the terminology used in tasks, PSMs and domain definitions. Again this separation enables knowledge sharing and reuse. For example, different tasks or PSMs can share parts of the same vocabulary and definitions. A fifth element of a specification of a KBS are *adapters* which are necessary to adjust the other (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 distinguished parts of an architecture, e.g. between domain and task or task and PSM. Refiners can be used to express the stepwise adaptation of elements of a specification, e.g. a task is refined or a PSM is refined (Fensel, 1997). Very generic PSMs and tasks can be refined to more specific ones by applying a sequence of refiners to them. Again, separating generic and specific parts of a reasoning process maximizes reusability. In the following we discuss some of the elements of an UPML architecture and how they are connected.

2.1. Ontology

An ontology provides an explicit specification of a conceptualization, which can be shared by multiple reasoning components communicating during a problem solving process. In our framework ontologies are used to define the terminology and its properties used to define tasks, PSMs, and domain models. UPML does not commit to a specific language style for defining a signature and its corresponding axioms. However, we provide two styles as possible ways for specifying signature and axioms. First we provide logic with sorts. Second, we provide a frame-based representation using concepts and attributes.

2.2. Task

The description of a *task* specifies goals that should be achieved in order to solve a given problem. A second part of a task specification is the definition of assumption about domain knowledge and preconditions on the input. These parts establish the definition of a problem that should be solved by the KBS. Contrary to most approaches in software engineering this problem definition is kept domain independent, which enables the reuse of generic problem definitions for different applications. A second particular feature is the distinction between preconditions on input and assumptions about knowledge. In an abstract sense, both can be viewed as input, however, distinguishing case data that are processes (i.e., input) from knowledge that is used to define the goal reflect a particular feature of *KBS*s. Preconditions are conditions on dynamic inputs. Assumptions are conditions on knowledge consulted by the reasoner but not manipulated. Often, assumptions can be checked in advance during the system building process, preconditions cannot. They rather restrict the valid inputs.

2.3. Domain Model

The description of the *domain model* introduces domain knowledge as it is required by the PSM and the task
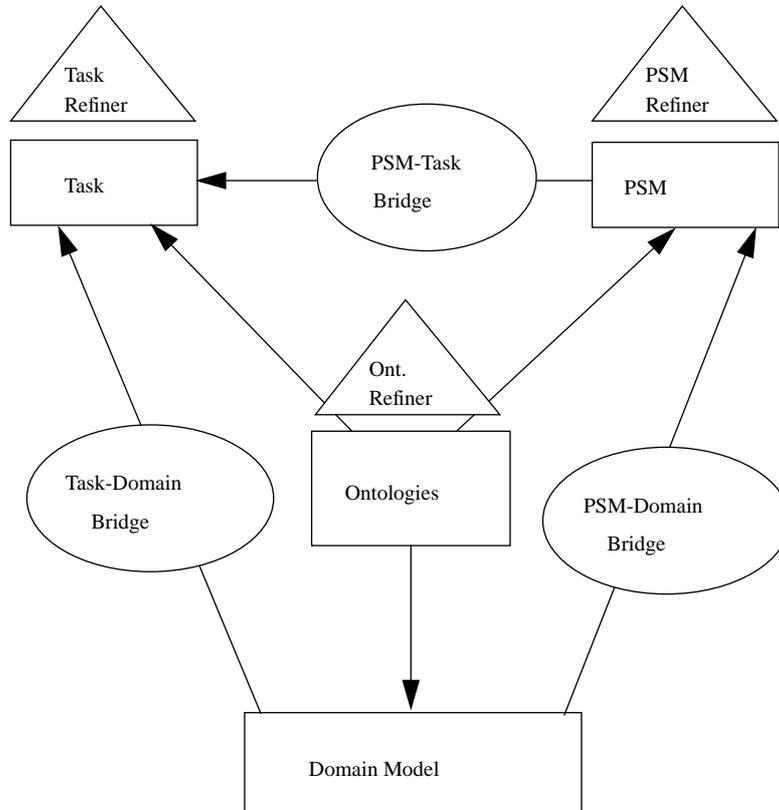

Figure 1.   The UPML architecture for knowledge-based system.

definition. Our framework for defining a *domain model* provides three elements: a characterization of properties of the domain knowledge, the domain knowledge, and (external) assumptions of the domain model. Again, ontologies are an externalized means for defining the terminology.

The *properties* of the domain knowledge are the counterpart of the requirements on domain knowledge introduced by the other parts of a specification. The *domain knowledge* is necessary to define the task in the given application domain and necessary to carry out the inference steps of the chosen PSM. *Assumptions* relate the domain knowledge to the actual domain. These (external) assumptions capture the implicit and explicit assumptions made while building a domain model of the real world. Technically they can be viewed as the missing pieces in the proof that the domain knowledge fulfills its assumed properties. Some of these properties may be directly inferred from the domain knowledge whereas others can only be derived by introducing assumptions about the environment of the system and the actually provided input. For example, typical external assumptions in model-based diagnosis are: the fault model is complete (no fault appears that is not captured by the model), the behavioral description of faults is complete (all fault behaviors of the components are modeled), the behavioral discrepancy that is provided as input is not the result of a measurement fault, etc.

2.4. Task-Domain Bridge

We will now discuss a specific adapter type of UPML, the *task-domain bridge*. This connector type instantiates tasks for specific domains and enable therefore their domain-independent and reusable description. Mapping of different terminologies can either directly be achieved by renaming or indirectly by linking their properties via mapping axioms. A connector may be forced to state assumptions about domain knowledge to ensure that the mapped terminologies respect the definitions of a task specification. These assumptions are the subset of assumptions of the task specification that are not part of the meta-level description of the domain model, i.e., which are not fulfilled by the current model.

2.5. Problem-Solving Method

UPML distinguishes two different types of PSMs: *complex PSMs* that decompose a task into subtasks and *primitive PSMs* that make assumptions about domain knowledge to perform a reasoning step. They do not have an internal structure, i.e. their internal structure is regarded as an implementational aspect not of any interest for the architectural specification of the entire KBS.

UPML provides six elements for describing a *complex* PSM: the already provided notion of (1) *pragmatics*, the (2) *costs* of a PSM, the (3) *communication policy*[1] describes the communication style of the method and its components, the (4) *ontology* provides a signature used for describing the method, the (5) *competence* description provides functional specification of the PSM and the (6) *operational* specification complements this with the description of the actual reasoning process of the method. The *competence* description of a method introduces two types of requirements that may be fulfilled to guarantee that the method is able to achieve its postcondition as specified by its competence. First, preconditions restrict valid

---

1.   It defines the ports of each building block and for each port how the block reacts on incoming events and how it provides outcoming events.

inputs. Second, assumptions describe the required functionality of components that implement the subtasks of the method. A complex method decomposes a task into subtasks and therefore recursively relies on other methods that proceed its subtasks. Such a subtask may describe a complex reasoning task that may further be decomposed by another PSM or may directly be preformed by a primitive PSM. An *operational description* defines the dynamic reasoning of a PSM. Such an operational description explains how the desired competence can be achieved. It defines the data and control flow between the main reasoning steps so as to achieve the functionality of the PSM.

The specification of a *primitive* PSM closely resembles the definition of a complex one with two significant differences: A primitive PSM does not provide an operational specification (this is regarded as implementational aspect) and the definition of the competence slightly differs. The assumptions on subcomponents are replaced by assumptions on domain knowledge. A complex PSM assumes subcomponents to provide parts of its functionality and a primitive PSM directly accesses domain knowledge for this purpose.

2.6. Problem-Solving Method Refiner

(Fensel, 1997) provide a principled approach for the adaptation of PSMs that provide more refined and therefore usable methods but prevent the combinatorial explosion of requiring too many components. This is achieved by *externalizing* the adaptation of methods. The generic method can still be used for cases in which none of its refinements fit to task and domain-specific circumstances. Moreover, refinements themselves can also be specified as reusable components and used to refine different methods.

3. Constraints, Guidelines and Tools

UPML is accompanied with a number of *constraints* ensuring well-defined systems. For example, for a task specification must hold:

ontology axioms $\cup$ *preconditions* $\cup$ *assumptions* $\models$ *goal*
ontology axioms $\cup$ *preconditions* $\cup$ *assumptions* must have a model.

That is, if the ontology axioms, preconditions, and assumptions are fulfilled by a domain and the provided case data then the goal must be achievable. In addition, we require that a task specification is consistent.

Design guidelines define a process model for building complex KBSs out of elementary components. The overall process is guided by tasks that provide generic descriptions of problem classes. After selecting, combining and refining tasks they are connected with PSMs and their combination are instantiated for the given domain. A more detailed discussion on this process model can be found in (Fensel et al., 1998b).

An editor for UPML was developed using PROTÉGÉ-II (Puerta et al., 1992). The output of the editor is translated into Frame-logic (Kifer et al., 1995) allowing browsing and querying UPML specifications with Ontobroker[2]. Still missing is a full-fledged theorem prover to reason about the formal specification parts of UPML.

————————————

2.   Ontobroker provides an ontology browser and a query interface to WWW documents (Fensel et al., 1998a) and could nicely be reused for our purpose.

References

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

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

Fensel, D: (1997): 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.

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

Fensel, D., Benjamins, V. R., Decker, S., Gaspari, M., Groenboom, R., Motta, E., Plaza, E., Schreiber, G., Studer, R., and Wielinga, B. (1998): *The Unified Problem-solving Method description Language UPML*, Deliverable Esprit Project 27169, IBROW3, 1998. ftp://ftp.aifb.uni-karlsruhe.de/pub/mike/dfe/spool/upml.{pdf,ps}.

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

Motta, E., Gaspari, M., and Fensel, D: (1998): *UPML Specification of a Parametric Design Library*, Deliverable Esprit Project 27169, IBROW3.

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

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