

## 5. UPML: The Language and Tool Support for Making the Semantic Web Alive

B. Omelayenko<sup>1</sup>, M. Crubézy<sup>2</sup>, D. Fensel<sup>1</sup>, R. Benjamins<sup>3</sup>, B. Wielinga<sup>4</sup>, E. Motta<sup>5</sup>, M. Musen<sup>2</sup>, Y. Ding<sup>1</sup>

### 1. Introduction

Originally, the Web grew mainly around representing static information using the HTML language, which provided a standard for document layout and was interpreted by browsers in a canonical way to render documents. On the one hand, it was the simplicity of HTML that enabled the fast growth of the WWW. On the other hand, its simplicity seriously hampered more advanced Web application in many domains and for many tasks. *The Semantic Web* [Berners-Lee & Fischetti, 1999] will transform the current World-Wide Web into a network of resources structured with the annotations defining their meaning and relationships. In this context, computers not only provide more efficient access to Web resources, but are also able to perform intelligent tasks with those resources. The explicit representation of the semantics of data, accompanied with domain theories (i.e. *ontologies*), will enable a Web that provides a qualitatively new level of service. It will weave an incredibly large network of human knowledge and will complement it with machine processability. Various automated services will help users to achieve their goals by accessing

and providing necessary information in a machine-understandable form. This process might ultimately create an extremely knowledgeable system with various specialized reasoning services – the systems which can support us in many aspects of life.

Many steps need to be taken to make this vision become true. Languages and tools for enriching information sources with machine-processable semantics must be developed (cf. [Fensel et al., 2001], [Fensel et al., 2000(b)]). Web-based reasoning services need to be developed, services which employ these semantic-enriched information sources to provide intelligent support to human users in task achievement. Specifically, such services will need to support users not only in finding and accessing information but also in achieving their goals based on this information.

The objective of the IBROW<sup>6</sup> project is to develop an intelligent broker able to configure knowledge systems from reusable components on the World Wide Web ([Fensel & Benjamins, 1998], [Benjamins et al., 1999]). IBROW brokers will handle Web requests for some classes of knowledge systems (e.g., diagnostic systems) by accessing libraries of reusable reasoning components on the Web, and selecting, adapting, and configuring them in accordance with the domain in question. This project integrates research on heterogeneous databases, interoperability, and Web technology with knowledge-system technologies, namely ontologies and problem-solving methods.

Nowadays, ontologies [Gruber, 1993] represent mainly *static* and declarative knowledge about a domain of expertise. The way to apply domain knowledge to achieve user tasks, or *dynamic* knowledge, is usually encoded in inference algorithms, which reason on the contents of the domain ontologies. Making this dynamic knowledge explicit and generic, and regarding it as an important part of the entire knowledge contained in a knowledge-based system (KBS) is the rationale which underlies *problem-solving methods* (cf. [Stefik, 1995], [Benjamins & Fensel, 1998], [Benjamins & Shadbolt, 1998], [Motta, 1999], [Fensel, 2000]).

Problem-solving methods (PSMs) provide reusable components for implementing the reasoning part of the KBSs.<sup>7</sup> PSMs refine inference engines to allow a more direct control of the reasoning process of a system to achieve a task. PSMs encode control knowledge independently of an application domain: PSMs can therefore be reused for different domains and applications. PSMs decompose the reasoning task of a knowledge-based system in a number of subtasks and inference actions, which are connected by knowledge roles representing the 'role' that knowledge plays in the reasoning process. Several libraries of problem-solving methods have been developed (cf. [Marcus, 1988], [Chandrasekaran et al., 1992], [Puppe, 1993], [Breuker & van de Velde, 1994], [Benjamins, 1995], [Musen 1998], and [Motta, 1999]) and a number of problem-solving method specification languages have been

proposed, ranging from informal notations (e.g. CML [Schreiber et al., 1994]) to formal modeling languages (see [Fensel & van Harmelen, 1994], [Fensel, 1995], [Gomez Perez & Benjamins, 1999] for summaries).

Ontologies and PSMs provide the components which need to be combined by the IBROW broker to configure a particular knowledge system on the Web. As a central requirement, both types of components, ontologies and PSMs, need to be properly marked-up to be localized on the Web and assembled into a working system by the broker. In the IBROW project, we have developed *the Unified Problem-solving Method Development Language* UPML<sup>8</sup> to specify problem-solving components and their software architectures to facilitate their semi-automatic reuse and adaptation (see [Fensel & Benjamins, 1998], [Fensel et al., 1999(a)], [Fensel et al., 1999(b)], [Fensel et al., to appear]). Concisely, UPML is a framework for specifying knowledge-intensive reasoning systems based on libraries of generic problem-solving components. Since its first definition, UPML has been adopted by the members of the IBROW consortium and has been used to specify a design library<sup>9</sup> [Motta et al., 1998] at the Open University, a library for classification problem-solving [Motta & Lu, 2000], and parts of the PSM library at Stanford University (cf. [Musen 1998]).

The goal of this chapter is to provide an overview of the UPML framework and tools and to demonstrate how this language provides the enabling technology for the next level of service

of the World-Wide Web: to provide users of the Web with online, ready-to-use problem-solving resources. In Section 2, we discuss the mission of IBROW in the context of the future Web and the central role of a specification language such as UPML. In Section 3, we introduce the UPML language for component markup. Tool support for UPML is provided by the Protégé environment, as presented in Section 4. Concluding remarks are presented in Section Conclusions.

## **2. Brokering Reasoning Components on the Web**

The mission of the IBROW project is to develop an intelligent brokering service capable of retrieving a set of knowledge components from the Web which, combined together, can solve the users' problem, according to stated requirements [Benjamins et al., 1999]. As illustrated in Figure 5.1, the main goal of the IBROW broker is to identify the components needed to solve the problem, and to adapt and configure them into a running reasoning service. In the context of the World-Wide Web, the task of the broker includes reusing third-party components available on the Web, and operating problem solvers and knowledge bases together in a distributed, 'plug-and-play' fashion.

**Figure 5.1 The IBROW approach. Based on user requirements about a task to be solved, IBROW broker needs to find, integrate and adapt a set of suitable knowledge components (i.e., knowledge bases and problem-solving methods) on the Web to solve a user-specified task. UPML is a central interoperability element of the IBROW approach.**

The IBROW approach offers a number of innovative aspects. Most of today's Web brokers (e.g. Metacrawler<sup>10</sup>, Ariadne<sup>11</sup>, Ontobroker<sup>12</sup>) handle only static information, whereas the IBROW broker is capable of managing dynamic information<sup>13</sup>. As a result, our approach offers an opportunity for a new type of electronic marketplaces, where reasoning services can be configured dynamically out of independent components to solve a specific task. In particular, IBROW will enable the construction of 'throw-away' applications for Web users.

A key issue in the IBROW approach is that reasoning components themselves need to be available on the Web. Moreover, components must be marked-up with machine-processable data, so that an IBROW broker can identify their capabilities and reason about them. This requirement calls for the development of a language for component markup, which must support the specification of the capabilities and assumptions of available problem solvers, the goal and assumptions of tasks, and the properties of domains. In addition, this markup language must hold non-functional, pragmatics descriptions of

available reasoning components. The language must be formal enough to express characteristics of reasoning components at the knowledge level, and it should also have a Web-compatible syntax, to allow the components to be distributed as Web resources. As explained in Section 3, the first achievement of the IBROW project was to develop the UPML language, which matches the requirements for brokering reasoning components on the Web.

**Figure 5.2 The brokering process in IBROW, in which the UPML markup language plays a central role.**

Based on UPML, the IBROW broker can reason about component annotations to determine the localization of the components and select appropriate components by matching them to the users' requirements. Reasoning capabilities are required from the broker to recognize and analyze the users' problem, to find relevant problem solvers for each (sub)task of the problem and to check the applicability of problems solvers according to the knowledge bases available. Once the broker has selected a suitable set of knowledge components, it needs to adapt them and enable their integration into a single service. Finally, the broker needs to execute the configured running system to solve the users' task. Figure 5.2 shows a view of the brokering process: libraries of problem-solving methods marked-up with UPML and correspond to UPML instances,

which are selected by the broker on the basis of their competence and the users' task specification. Given the users' domain model (shown as a 'Customer KB' in the figure), the broker selects the PSMs, which make proper assumptions about the domain model via special adaptation elements in UPML - PSM-Domain bridges. Finally, the broker passes the PSMs on a PSM server able to execute them.

Again, the UPML language for component markup, discussed in the next section, plays a kernel role at each stage of the brokering process.

### **3. UPML: The Language for Knowledge Component MarkUp**

UPML is a software architecture specially designed to describe knowledge systems. The UPML architecture presented in Figure 5.3 consists of six different kinds of elements. A task defines the problem to be solved by the knowledge system. A problem-solving method defines the reasoning process used to solve the problem. A domain model defines the domain knowledge available to solve the problem. Each of these elements is described independently to enable reuse of: task descriptions in different domains, problem-solving methods for different tasks and domains, and domain knowledge for different tasks and problem-solving methods. Ontologies provide the terminology used in the tasks, problem-solving methods, and domain definitions. Again this separation enables knowledge



sharing and reuse. For example, different tasks or problem-solving methods can share parts of the same vocabulary and definitions. Further elements of the specification are adapters, which are necessary to adjust other (reusable) elements 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 specific parts of the architecture, e.g. between a domain and a task or a task and a problem-solving method. Refiners can be used to express the stepwise adaptation of other elements of the specification, e.g. generic problem-solving methods and tasks can be refined to more specific ones by applying to them a sequence of refiners ([Fensel, 1997], [Fensel & Motta, to appear]). Again, separating the generic and specific parts of a reasoning process enhances reusability. The main distinction between bridges and refiners is that bridges change the input and output of the components to make them fit together. Refiners, by contrast, may change only the internal details, e.g. the subtasks of a problem-solving method. This provides UPML with a structured and principled approach for developing and refining heuristic reasoning components. This approach provides a three-dimensional design space where different types of adapters correspond to different types of moves in that space [Fensel, 2000] and [Fensel & Motta, to appear].

### 3.1. Overview of the UPML Framework

UPML relies on a meta-ontology that defines the concepts and relations which are then instantiated for specific knowledge components. As shown in Figure 5.4, this ontology is based on a root class *Entity*, which defines that each UPML *concept* and *relation* is represented with a list of attributes of a certain type. The root ontology of UPML defines two basic classes: *concept* and *binary relation*, both of which are subclasses of the *Entity class*. All UPML concepts and relations are subclasses of these two root classes. The hierarchy of classes used to define UPML is also presented in Figure 5.4. The main classes of this hierarchy are discussed in the remainder of this section. The concepts of UPML define parts of a problem-solving system, as described in Section 3.2. *Binary Relations* specify the interactions between the concepts as described in Section 3.3. *Architectural constraints and design guidelines* for UPML, which make it a full-fledged software architecture, are described in [Fensel & Groenboom, 1999] and [Fensel et al., to appear]. The Web syntaxes for UPML is discussed in Section 3.4.

**Figure 5.3 The UPML architecture.**

**Figure 5.4 Class hierarchy of UPML. Each concept or relation in the UPML ontology is derived from the root ontology entities represented by the boxes.**

### **3.2. Concepts**

The *Library* concept is the overarching concept of UPML architecture: it contains a pointer to each component of a UPML specification (Figure 5.5). The subclass-of relationship between two entities is denoted by the symbol <. Each concept or relation is represented as a list of attribute-type pairs (attribute : type), where the type is either a primitive type (STRING), or a class of the hierarchy. The brackets around the types denote that the corresponding attribute can have multiple values.

#### **Figure 5.5 The *Library* concept.**

There are four main types of knowledge components in UPML: *Ontology*, *Domain Model*, *Task*, and *PSM* (see Figure 5.6). All types of components are defined as subclasses of the root concept *Knowledge Component*, also presented in Figure 5.6. Each component has a pragmatics description and relies on one or more ontologies, which define its universe of discourse. All attributes represented in the figure model part-of relationship besides uses.<sup>14</sup>

An *Ontology* (cf. [Fensel, 2001]) is used in the definition of *tasks*, *PSMs*, and *domain models*. An ontology provides an explicit specification of a conceptualization, which can be reused and shared by multiple reasoning components. It enables the definition of reusable terminology used by all other components through a signature, theorems and axioms. Ontologies are the key instrument of interchange among knowledge components. The core of an ontology specification in UPML is its *signature* definition, which captures the ontological elements used by a component, or *signature elements*. *Signature elements* are expressed in a certain modeling language. An *ontology* also provides axioms which characterize logical properties of the *signature elements*. Additional *theorems* may list useful statements, which are implied by the axioms.

The *Task* concept specifies the task to be achieved by the PSMs of the library. The *input roles* and *output roles* together with the *competence* property define the input/output specification of the task. The *input roles* specify the input of case data and the *output roles* specify the output of case data. The *Competence* concept represents the functional input/output specification of the *Task* component. *Competence* includes *preconditions* restricting valid inputs and *postconditions* which describe the output of a method or task. The *assumptions* property of *Task* contains requirements

regarding the knowledge used to define the goal. A *Task* can import and refine other tasks via its *uses* attribute.

The *Domain Model* concept introduces domain knowledge, merely the formulas used by the problem-solving methods and tasks. The *Domain Model* consists of three elements: *properties*, *meta-knowledge*, and the domain *knowledge* itself. *Meta-knowledge* captures the implicit and explicit assumptions made in the domain model of the real world. *Meta-knowledge* is assumed to be true. In other words, it has not been proven or cannot be proven, and corresponds to our assumptions about the domain. *Domain knowledge* is the knowledge base of the domain necessary to define the task in a given application domain and to carry out the inference steps of the selected problem-solving method. *Knowledge* is specified under the assumption that the *meta-knowledge* is true. *Properties* (a synonym for theorems) can be derived from *domain knowledge*

**Figure 5.6 Knowledge Components: Ontology, Domain Model, Task, and PSM.**

The *PSM* component represents a problem-solving method, defined by its *competence* and *communication* properties. The *input roles* and *output roles* of the PSM specify its inputs and outputs, similarly to their function in the *Task* component. The PSM's *communication* property describes its interaction protocol with its environment in particular with other (PSM) components.

The *PSM* concept has the following two subclasses: *Problem Decomposer* and *Reasoning Resource* presented in Figure 5.7. A *Problem Decomposer* decomposes a task to be solved into a set of *subtasks*. Its *Operational Description* specifies the control structure over the subtasks and internal data flow among the subtasks. A *Reasoning Resource* solves a primitive step (or subtask) of a problem provided by the *problem decomposer*. It specifies *assumptions* regarding the domain knowledge, which must be fulfilled in order to perform a primitive reasoning step. Its internal structure is usually not specified, as it is considered as an implementational aspect of no interest to the architectural specification of a problem-solving system. The *knowledge roles* attribute specifies the input of the (domain) knowledge to the reasoning resource.

**Figure 5.7 Problem Decomposer and Reasoning Resource.**

Considering knowledge components as Web resources themselves, UPML defines the concept of *Pragmatics*, which holds attributes which describe practical and reference information about a component. *Pragmatics* attributes are derived mainly from the Dublin Core<sup>15</sup> metadata recommendation for annotating Web resources.

UPML does not commit to any logical language to express formulas about the knowledge components, or to any procedural language to describe the operational control of a problem decomposer. In several places, developers can extend UPML with

additional concepts to hold the primitives of their languages of choice.

### 3.3. Binary Relations

*Binary Relations* specify the interactions between the UPML knowledge components. The root binary relation of UPML is *Adapter* (Figure 5.8). The *Adapter* connects two components, called *arguments*, through a set of *renaming* correspondences between the terms of both arguments. Similar to knowledge components, an *Adapter* holds *pragmatics* information and refers to specific *ontologies*. UPML introduces two subclasses of *Adapter*: a *Bridge* and a *Refiner* shown in Figure 5.9.

#### Figure 5.8 Adapter.

The *Bridge* relation connects two *Knowledge Components* of different kinds. It defines *mapping axioms* and additional *assumptions* about the components it relates. The *Bridge* relation has three subrelations: a *PSM-Domain Bridge* connects a *PSM* with a *Domain Model*; a *PSM-Task Bridge* connects a *PSM* and a *Task*; and a *Task-Domain Bridge* connects a *Task* and a *Domain Model*.

#### Figure 5.9 Bridge and Refiner

The *Refiner* relation connects two knowledge components of the same type and so expresses the stepwise adaptation of one

component into the other. Very generic problem-solving methods and tasks can be refined to more specific ones by applying a sequence of refiners to them. A *Refiner* assumes that its two attributes *in* and *out* have the same type. This serves to ensure that the refiner modifies a given component, as opposed to mapping it to a different kind of component via the *Bridge* relation.

Each main UPML component has its own associated type of refiner. Consequently, the *Refiner* relation has four component-specific subrelations: a *Domain Refiner*, an *Ontology Refiner*, a *Task Refiner*, and a *PSM Refiner*. The definition of a refiner includes the attributes specific to each kind of component. Each refiner has its own restrictions on *in*-put and *out*-put: the *Domain Refiner* contains redefined *properties*, *metaknowledge* and *knowledge* in the refined component. Similarly, the *Ontology Refiner* contains refined *signature*, *theorems*, and *axioms*. The *Task Refiner* refines *competence* and *assumptions*, and the *PSM Refiner* refines *communication* and *competence*.

The separation of generic and specific parts of a reasoning process maximizes reusability. UPML offers two ways of combining components of the same type. Both serve a similar purpose, however they provide complementary means. First, a component can import another component via the *uses* attribute. Hence, the component can make use of definitions imported from the other component, monotonically refine them, and extend



them. In this case, the *uses* relationship is not modeled by an explicit entity in the UPML specification but rather via an attribute of an existing component (the one that imports another component). This first approach corresponds to a monotonic extension of a component. Second, a component can be defined as a refinement of another component via the *Refiner* relation. In this case, the former component can rewrite the aspects of the latter component via the *renamings* attribute. Also, in this case we model the *uses*-relationship by an explicit entity of the UPML specification (i.e., a *Refiner*). This second approach enables non-monotonic modification of a component via an explicit element of the architecture. As mentioned earlier, this provides UPML *with a structured and principled approach for developing and configuring heuristic reasoning components* by adapting and refining generic components (cf. [Fensel, 2000], [Fensel & Motta, to appear]).

### **3.4. Web Syntaxes**

In this section we describe the Web syntaxes for UPML meta-ontology based on XML and RDF. A Web syntax is crucially important for UPML because UPML is posed as a standard for knowledge component markup on the Web. The syntax consists of three documents: an XML DTD definition for UPML, an XML Schema definition for UPML, and an RDF Schema definition for UPML as described in the next section.

### 3.4.1. XML Syntax

*XML*<sup>16</sup> is one of the Web standards which can be used to describe UPML. XML is a widely supported, domain-independent language for representing, storing, sharing, and exchanging data. It provides the means to mark-up the semantics of data, as well as to validate, and exchange data structures. XML documents consist of XML tags, which have their names and associated values. The tags can be nested into one another to represent the hierarchical structure of a document. This structure provides a mechanism to impose constraints on the storage layout and logical structure.

*XML Schema*<sup>17</sup> is a W3C standard aimed to specify the structure of XML documents. Parts of a document are specified with a set of data types, either primitive or complex, which can be inherited from one another. This inheritance of data structures allows explicit encoding of UPML structures as XML Schema structures.

Figure 5.10 shows a fragment of XML Schema syntax for the *Knowledge Component* concept.

**Figure 5.10 Part of the XML Schema defining the XML representation of the *Knowledge Component* concept.**

A relatively new standard, XML Schema is still not widely supported by industry. Hence, we also provide a *DTD* specification for UPML, part of which is presented in Figure

5.11. However, DTDs are only capable of representing the structure of document instances; they cannot capture the hierarchy of the UPML structures.

**Figure 5.11 Part of the DTD for UPML defining the *Library* concept.**

Both the full XML Schema and the full DTD for UPML are available from the UPML website<sup>18</sup>.

### 3.4.2. RDF Syntax

RDF<sup>19</sup> is an upcoming standard for representing machine-processable semantics of on-line information resources. Unlike XML, which enables serialization of trees, RDF provides extensive representation of options to perform knowledge-level markup. The foundation of RDF is a model for representing named properties and property values. RDF properties may be thought of as attributes of resources. In this respect, they correspond to traditional attribute-value pairs used in UPML. RDF properties also represent relationships between the resources. The RDF model can therefore resemble an entity-relation diagram.

The structure of RDF documents is specified with *RDF Schema* (RDFS) [Brickley & Guha, 2000]. RDF resources represent some components and correspond to the *Concept* class in the UPML hierarchy. Hence, in the RDF Schema of UPML we define *Entity* as a subclass of *rdfs:resource*, and *Concept* as a

subclass of *Entity*. We define all other concepts of UPML as direct or indirect subclasses of the RDF class *Concept*.

RDF properties are conceptually equivalent to the UPML *Binary Relations*. Hence, the latter can be represented as properties. However, RDF Schema does not allow defining properties of properties. Accordingly, if we were to define *Binary Relation* as a subclass of *rdfs:property*, then we would have no way of describing the attributes of binary relations. Consequently, in the RDF Schema of UPML we defined the *Binary Relation* as a subclass of *Entity*.

As a result, we used RDF classes to define both the *Concepts* and the *Binary Relations*. Each attribute of *Concepts* and *Binary Relations* is defined as a property of the corresponding class in RDFS. Each property has the corresponding class as its domain, and the type of this attribute, as defined in the UPML specification, as its range.

The RDF syntax of UPML is generated from the Protégé-2000<sup>20</sup> knowledge-acquisition tool, which supports import and export of ontologies from and to RDF. For example, a sample of the RDFS specification for the *Knowledge Component* concept is shown in Figure 5.12, while the whole specification is available from the UPML website.

**Figure 5.12 A fragment of the RDF Schema for UPML.**

The tool for specifying UPML components is discussed in the next section.

#### 4. An Editor for UPML Specifications based on Protégé-2000

As described in the previous section, the UPML framework can be seen as an ontology composed of classes and relations describing reusable knowledge components. Instances of these classes and relations are particular *knowledge components* and *adapters* (e.g., a classification task and a heuristic classifier problem decomposer). To enable developers to specify and annotate libraries of knowledge components, we created an editor for UPML using Protégé-2000<sup>21</sup>. Protégé is an extensible ontology-editing and knowledge-acquisition environment assisting users in the construction of large electronic knowledge bases [Grosso et al., 1999]. Protégé-2000 allows users to create, browse, and edit domain ontologies using a frame-based representation, compliant with the OKBC knowledge model [Chaudhri et al., 1998].

In Protégé-2000, an ontology is represented with a multiple-inheritance hierarchy of the classes of concepts which are important in a domain. Slots are attached to these classes and define their attributes. Facets restrict the type of value that a slot can take. Protégé automatically generates a graphical knowledge-acquisition tool from the ontology, which enables application specialists to enter the detailed content knowledge required to define specific applications [Puerta et al., 1992]. Protégé allows developers to custom-

tailor this knowledge-acquisition tool directly by configuring graphical entities on forms, which are attached to each class in the ontology for the acquisition of instances (particular exemplars of the classes). Consequently, the application specialists can enter domain information by filling in the blanks of intuitive forms and by drawing diagrams composed of selectable icons and connectors. Protégé-2000 is able to store the knowledge bases in several formats, including RDF [Noy et al., to appear].

We modeled the set of concepts and relationships of UPML as a hierarchy of classes in Protégé-2000, with slots and facets attached to them. Both concepts and binary relations in UPML are reified as classes in Protégé, so they can have attributes, and be subclassed.

Figure 5.13 shows most of the hierarchy of the classes we used to model UPML and the definition of the class Library. The UPML ontology in Protégé reflects the fact that UPML does not commit to any logical or procedural language to express the formulas and programs that define knowledge components and adapters. By means of the ontology-extension and ontology inclusion mechanisms of Protégé, developers can extend the UPML ontology with the primitives necessary to write expressions in their object language of choice. For example, we recently used the UPML editor to specify a library of classification problem solving components [Motta & Lu, 2000]. The components of the library are coded using the logical and

operational OCML language [Motta, 1999]. First, we modeled the OCML set of basic primitives (such as classes, relations, axioms and functions) themselves as a meta-ontology in Protégé-2000. We then included this ontology in the UPML editor (as partially shown in the lower left part of Figure 5.13). From there, we were able to extend (subclass) the UPML concepts *Signature*, *Signature Element*, *Formula* and *Program* with the definitions which refer to the OCML primitives, as presented in Figure 5.13.

**Figure 5.13 A snapshot of the Protégé-based UPML editor: The ontology of UPML modelled in Protégé as a hierarchy of classes, extended with OCML-specific classes (left panel) and the definition of the Library concept, which holds a pointer to every component in the architecture (right panel).**

Given this model of UPML, Protégé-2000 automatically generated an RDF Schema representation holding the UPML classes and properties for annotating reasoning resources. Based on the UPML ontology, Protégé-2000 also generated a graphical editor for instantiating specific UPML specifications, e.g., the components of the library for classification problem-solving. We then custom-tailored this editor to center the knowledge-acquisition process on the use of diagrammatic metaphors. In particular, we defined specific kinds of diagrams to enter the task decomposition (*inference*

*structure*) of a problem decomposer and the control regime of its corresponding operational description (*control structure*).

As shown in Figure 5.14, the UPML editor makes it possible to browse the list of instances of a selected class and to view and edit the knowledge-acquisition form associated to the selected instance. In this figure, the editor displays the 'heuristic optimal solution classifier' instance of the *Problem Decomposer* class. The knowledge-acquisition form for a *problem decomposer* PSM contains a number of user interface components to enter the values of the slots defined for the *Problem Decomposer* class. For example, this form contains a sub-form specifying the *operational description* slot of the *PSM* (on the right). This sub-form includes an inference structure diagram, which helps users to specify the competence slots of the *PSM* (*input roles*, *output roles* and *subtasks*) by directly drawing nodes and links in the diagram, which, in turn, automatically creates and fills in the corresponding instances of the *Signature Element* and the *Task* classes.

Once created, the specification of a set of UPML knowledge components (instances) can be exported as a corresponding set of RDF statements, which refer to the RDF Schema of UPML. When Protégé-2000 generates an RDF specification from an ontology, it resolves the differences between the knowledge models of Protégé and RDF Schema by adding specific RDF statements to express facets such as cardinality constraints on slots, or multiple domains or ranges for a given slot. Consequently, the



resulting RDF code for UPML contains the complete translation of the UPML ontology; however, some parts of the ontology are only understandable by Protégé-2000 (see [Noy et al., to appear] for a discussion).

As a result, the UPML editor in Protégé-2000 provides a guided framework, which helps developers to define the UPML specifications and to annotate the knowledge components, which they want to publish on the Web. Protégé-2000 is extensible through its API (application programming interface) [Musen et al., 2000]. We envisage enhancing the UPML editor with services which will help the users to configure the reasoning resources and problem decomposers for different domains and tasks in connection to the IBROW broker. The resulting configurations of problem solvers will help augment the UPML specification of the libraries available with the appropriate bridges and refiner components.

**Figure 5.14 A snapshot of the Protégé-based UPML editor: the 'heuristic optimal solution classifier' instance of the Problem-Decomposer class, with its inference structure diagram on the right.**

## **Conclusions**

UPML defines an *architecture* for describing heuristic reasoning components (cf. [Berners-Lee & Fischetti, 1999] for

the vision of the latter) for the Semantic Web. UPML is language-neutral in the sense that different formal languages can be plugged in to describe the elementary slots defined by UPML. We already used order-sorted logic [Fensel et al., 1998], frame logic [Kifer et al., 1995], and OCML [Motta, 1999] successfully for adding formal semantics to UPML specifications (cf. [Fensel et al., to appear]). We also tried to use the OIL language [Fensel et al., 2001] which has been developed within the Ontoknowledge<sup>22</sup> project (cf. [Fensel et al., 2000(b)]) as a Web-based ontology language. OIL was a significant source of inspiration for the ontology language called DAML+OIL<sup>23</sup> developed in a joined EU/US working group on language standardization, which is currently present as a W3C working group on the Semantic Web. However in the case of OIL, the results achieved were somewhat disappointing (cf. [Fensel et al., 2000(a)]). OIL does not provide an adequate expressive power for many of the axiomatic parts of UPML specifications. Extending the expressive power of OIL seems to be absolutely necessary for making it usable in this context.

In general, UPML is concerned with describing the dynamic reasoning aspect of the Semantic Web. Therefore, it defines a new layer on top of the currently developed language standards for the Semantic Web. This layer provides machine-processable semantics for the dynamic information sources of the Semantic Web (see Chapter 1 for relevant discussion).

UPML is a description language and does not require operational semantics. However, the IBROW broker must be able to reason about the expressions in the description language. In that sense, one can view the broker as a special-purpose 'interpreter' of the language. The current priority in the IBROW project is to use UPML for annotating large libraries of problems-solving methods and implementing UPML-based reasoning services to enable their intelligent brokering on the Web. Here we can employ many concepts of the component retrieval area developed for software engineering. The use of formal techniques for software component retrieval is discussed in [Jeng & Cheng, 1992], [Jeng & Cheng, 1995], [Penix & Alexander, 1995], [Chen & Cheng, 1997], [Jiliani et al., 1997], [Mili, 1997], [Mili et al., 1997], [Schuman & Fischer, 1997], [Penix et al., 1997], and [Zaremski & Wing, 1997].

In many aspects the RETSINA<sup>24</sup> project (cf. [Sycara et al., 2001]) is similar to IBROW. Although it is focused on multi-agent systems, this project deals with aspects similar to those we encounter in IBROW. Heterogeneous agents must be able to communicate with each other by means of a common language. The language needs to be coordinated effectively across distributed networks of information. An agent capability description language called LARKS (*Language for Advertisement and Request for Knowledge Sharing*) has been developed (cf. [Sycara et al., 2001]) addressing the problem of agent interoperability. This common language is used by middle or

matchmaking agents to pair service-requesting agents with service-providing agents, which meet the requesting agents' requirements. The matching engine of the matchmaker agent contains five different filters for context matching, word frequency profile comparison, similarity matching, signature matching, and constraint matching. The user configures these filters to achieve the desired tradeoff between performance and matching quality. The main differences between UPML and LARKS are:

- UPML defines a richer architecture for describing the reasoning components compared to LARKS.
- UPML is a full-fledged methodological framework for developing Web-enabled libraries of such components.
- LARKS fixes the language used to describe the competence of these components, whereas UPML 'merely' provides an architecture in which several languages can be plugged into.

Finally, one has to admit that the actual retrieving component in RETSINA is far more developed than the current brokering support in IBROW. This also indicates the main direction for further development in IBROW. In particular, we foresee that the scope of the knowledge components to be brokered in IBROW will need to be broadened from traditional problem-solving methods to agent-based components capable of performing reasoning steps autonomously. Both UPML and the broker will need to take into account these new kinds of components [Abasolo et al., 2001].

During the early phase of the IBROW project, we demonstrated how our brokering approach based on UPML could be used for the toy problem of classifying apples for users<sup>25</sup>. The broker used an approach based on Prolog and CORBA to localize, compose, and integrate the heterogeneous components of the configured system, such as knowledge bases of the apple domain and classification problem-solving methods [Benjamins et al., 1999]. The results of this experiment appeared very promising and form the foundations for the work that we have planned for the next phases of the IBROW project. Consequently, we are expecting even more promising results from this stage, results that will help to make the vision of the Semantic Web a reality.

## References

- [Abasolo et al., 2001] C. Abasolo, J.-L. Arcos, E. Armengol, M. Gómez, J.-M. López-Cobo, M. López-Sánchez, R. López de Mántaras, E. Plaza, C. van Aart, and B. Wielinga: Libraries for Information Agents, In: IBROW Project IST-1999-19005: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable 4, 2001.
- [Benjamins & Fensel, 1998] V. R. Benjamins and D. Fensel: Special issue on problem-solving methods, *International Journal of Human-Computer Studies*, 49(4), 1998.
- [Benjamins & Shadbolt, 1998] V. R. Benjamins and N. Shadbolt: Special Issue on Knowledge Acquisition and Planning, *International Journal of Human-Computer Studies*, 48(4), 1998.
- [Benjamins et al., 1999] V. R. Benjamins, B. Wielinga, J. Wielemaker, and D. Fensel: Brokering Problem-Solving Knowledge at the Internet. In: D. Fensel et al. (eds.), *Knowledge Acquisition, Modeling, and Management, Proceedings of the European Knowledge Acquisition Workshop (EKAW-99)*, LNAI 1621, Springer-Verlag, May, 1999.

- [Benjamins, 1995] V. R. Benjamins: Problem Solving Methods for Diagnosis And Their Role in Knowledge Acquisition, *International Journal of Expert Systems: Research and Application*, 8(2):93-120, 1995.
- [Berners-Lee & Fischetti, 1999] T. Berners-Lee and M. Fischetti: *Weaving the Web*, Harper, San Francisco, 1999.
- [Breuker & van de Velde, 1994] J. Breuker and W. van de Velde (eds.): *The CommonKADS Library for Expertise Modeling*, IOS Press, Amsterdam, The Netherlands, 1994.
- [Brickley & Guha, 2000] D. Brickley and R. Guha: Resource Description Framework (RDF) Schema Specification 1.0, W3C Candidate Recommendation, March, 2000; available online at <http://www.w3.org/TR/rdf-schema>
- [Chandrasekaran et al., 1992] B. Chandrasekaran, T. Johnson, and J. Smith: Task Structure Analysis for Knowledge Modeling, *Communications of the ACM*, 35(9):124-137, 1992.
- [Chaudhri et al., 1998] V. Chaudhri, A. Farquhar, R. Fikes, P. Karp, and J. Rice: OKBC: A programmatic foundation for knowledge base interoperability. In: *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI-98) and of the 10th Conference on Innovative Applications of Artificial Intelligence (IAAI-98)*, AAAI Press, 1998, p. 600–607.
- [Chen & Cheng, 1997] Y. Chen and B. Cheng: Facilitating an Automated Approach to Architecture-based Software Reuse. In *Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97)*, Incline Village, Nevada, November 3-5, 1997.
- [Fensel, 1995] D. Fensel: Formal Specification Languages in Knowledge and Software Engineering, *The Knowledge Engineering Review*, 10(4), 1995.
- [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*, LNAI 1319, Springer-Verlag, 1997.
- [Fensel, 2000] D. Fensel: *Problem-Solving Methods: Understanding, Description, Development, and Reuse*, LNAI 1791, Springer-Verlag, 2000.
- [Fensel, 2001] D. Fensel: *Ontologies: Silver Bullet for Knowledge Management and Electronic Commerce*, Springer-Verlag, 2001.

- [Fensel & Benjamins, 1998] D. Fensel and V. Benjamins: Key Issues for Problem-Solving Methods Reuse. In: Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98), Brighton, UK, August 23-28, 1998, p. 63-67.
- [Fensel & Groenboom, 1999] D. Fensel and R. Groenboom: An Architecture for Knowledge-Based Systems, The Knowledge Engineering Review, 14(3):153-173, 1999.
- [Fensel & Motta, to appear] D. Fensel and E. Motta: Structured Development of Problem Solving Methods, IEEE Transactions on Knowledge and Data Engineering, to appear; available online at <http://www.cs.vu.nl/~dieter/pub.html>
- [Fensel & van Harmelen, 1994] D. Fensel and F. van Harmelen: A Comparison of Languages which Operationalize and Formalize KADS Models of Expertise, The Knowledge Engineering Review, 9(2), 1994.
- [Fensel et al., 1998] D. Fensel, R. Groenboom, and G. Renardel de Lavalette: Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems, Data and Knowledge Engineering, 26(3):243-269, 1998.
- [Fensel et al., 1999(a)] D. Fensel, V. R. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Plaza, G. Schreiber, R. Studer, and Bob Wielinga: The Unified Problem-Solving Method Development Language UPML. In: IBROW3 ESPRIT Project 27169: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web, Deliverable 1.1, 1999.
- [Fensel et al., 1999(b)] D. Fensel, V. Benjamins, E. Motta, and B. Wielinga: UPML: A Framework for knowledge system reuse. In: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-99), Stockholm, Sweden, July 31 - August 5, 1999.
- [Fensel et al., 2000(a)] D. Fensel, M. Crubézy, F. van Harmelen, and I. Horrocks: OIL & UPML: A Unifying Framework for the Knowledge Web. In Proceedings of the Workshop on Applications of Ontologies and Problem-solving Methods, 14th European Conference on Artificial Intelligence ECAI'00, Berlin, Germany August 20-25, 2000.
- [Fensel et al., 2000(b)] D. Fensel, F. van Harmelen, H. Akkermans, M. Klein, J. Broekstra, C. Fluyt, J. van der Meer, H.-P. Schnurr, R. Studer, J. Davies, J. Hughes, U. Krohn, R. Engels, B. Bremdahl, F. Ygge, U. Reimer, and I. Horrocks: OnToKnowledge: Ontology-based Tools for Knowledge Management. In:

- Proceedings of the eBusiness and eWork 2000 Conference (EMMSEC 2000), Madrid, Spain, October 18-20, 2000.
- [Fensel et al., 2001] D. Fensel, I. Horrocks, F. van Harmelen, D. McGuinness, and P. Patel-Schneider: OIL: Ontology Infrastructure to Enable the Semantic Web, IEEE Intelligent Systems, March/April, 2001.
- [Fensel et al., to appear] D. Fensel, E. Motta, V. R. Benjamins, M. Crubézy, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, F. van Harmelen, M. Musen, E. Plaza, G. Schreiber, R. Studer, and B. Wielinga: The Unified Problem-solving Method Development Language UPML, to appear in Knowledge and Informational Systems (KAIS); available online at <http://www.cs.vu.nl/~dieter/pub.html>
- [Gomez Perez & Benjamins, 1999] A. Gomez Perez and V. R. Benjamins: Applications of ontologies and problem-solving methods. AI Magazine 20(1):119– 122, 1999.
- [Grosso et al., 1999] W. Grosso, H. Eriksson, R. Ferguson, J. Gennari, S. Tu, and M. Musen: Knowledge Modeling at the Millennium (The Design and Evolution of Protégé-2000). In: Proceedings of the Twelfth Workshop on Knowledge Acquisition, Modeling and Management (KAW99), Banff, Alberta, Canada, October 16-21, 1999.
- [Gruber, 1993] T. Gruber: Towards Principles for the Design of Ontologies Used for Knowledge Sharing, In: N. Guarino and R. Poli (eds.), Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, Deventer, The Netherlands, 1993.
- [Jeng & Cheng, 1992] J.-J. Jeng and B. H. Cheng: Using Automated Reasoning Techniques to Determine Software Reuse, International Journal of Software Engineering and Knowledge Engineering, 2(4):523-546, 1992.
- [Jeng & Cheng, 1995] J.-J. Jeng and B. H. Cheng: Specification Matching for Software Reuse: A Foundation. In: Proceedings of the ACM Symposium on Software Reuse, Seattle, Washington, April, 1995, pp. 97-105.
- [Jiliani et al., 1997] L. Jilani, J. Desharnais, M. Frappier, R. Mili, and A. Mili: Retrieving Software Components That Minimize Adaptation Effort. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.
- [Kifer et al., 1995] M. Kifer, G. Lausen, and J. Wu: Logical Foundations of Object-Oriented and Frame-Based Languages, Journal of the ACM, 42(4):741-843, 1995.
- [Marcus, 1988] S. Marcus (ed.): Automating Knowledge Acquisition for Experts Systems, Kluwer Academic Publisher, Boston, 1988.



- [Mili et al., 1997] R. Mili, A. Mili, and R. Mittermeir: Storing and Retrieving Software Components: A Refinement Based System, *IEEE Transactions on Software Engineering*, 23(7):445-460, 1997.
- [Mili, 1997] F. Mili: Transformational Based Problem Solving Reuse. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering (SEKE-97)*, Madrid, Spain, June 18-20, 1997.
- [Motta & Lu, 2000] E. Motta and W. Lu: A Library of Components for Classification Problem Solving. In: *Proceedings of the 2000 Pacific Rim Knowledge Acquisition Workshop*, Sydney, Australia, December 11-13, 2000.
- [Motta et al., 1998] E. Motta, M. Gaspari, and D. Fensel: UPML Specification of a Parametric Design Library, In: *IBROW3 ESPRIT Project 27169: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*, Deliverable D4.1, 1998.
- [Motta, 1999] E. Motta: *Reusable Components for Knowledge Modeling*, IOS Press, Amsterdam, 1999.
- [Musen 1998] M. Musen: Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. In: C.G. Chute(ed.), *1998 AMIA Annual Symposium*, Orlando, FL, 1998, p. 46-52.
- [Musen et al., 2000] M. Musen, R. Ferguson, W. Grosso, N. Noy, M. Crubezy, and J. Gennari. Component-Based Support for Building Knowledge-Acquisition Systems. In *Proceedings of the Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000)*, Beijing, 2000.
- [Noy et al., to appear] N. Noy, M. Sintek, S. Decker, M. Crubézy, R. Ferguson, M. Musen: One Size Does Fit All: Acquiring Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, Special issue on Semantic Web Technology, to appear.
- [Omelayenko et al., 2000] B. Omelayenko, M. Crubézy, D. Fensel, Y. Ding, E. Motta, and M. Musen: Meta Data and UPML, In: *IBROW Project IST-1999-19005: An Intelligent Brokering Service for Knowledge-Component Reuse on the World-Wide Web*, Deliverable 5; available online at: <http://www.cs.vu.nl/~upml/>
- [Penix & Alexander, 1995] J. Penix and P. Alexander: Design Representation for Automating Software Component Reuse. In *Proceedings of the First International Workshop on Knowledge-Based Systems for the (Re)use of Program Libraries*, Sophia Antipolis, France, November 23-24, 1995.

- [Penix et al., 1997] J. Penix, P. Alexander, and K. Havelund: Declarative Specification of Software Architectures. In Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.
- [Puerta et al., 1992] A. Puerta, J. Egar, S. Tu, and M. Musen: A Multiple-method Knowledge-Acquisition Shell for the Automatic Generation of Knowledge-acquisition Tools, *Knowledge Acquisition*, 4(2):171-196, 1992.
- [Puppe, 1993] F. Puppe: Systematic Introduction to Expert Systems: Knowledge Representation and Problem-Solving Methods, Springer-Verlag, Berlin, 1993.
- [Schreiber et al., 1994] A. Schreiber, B. Wielinga, J. Akkermans, W. van de Velde, and R. de Hoog: CommonKADS. A Comprehensive Methodology for KBS Development, *IEEE Expert*, 9(6):28-37, 1994.
- [Schuman & Fischer, 1997] J. Schuman and B. Fischer: NORA/HAMMER: Making Deduction-Based Software Component Retrieval Practical. In: Proceedings of the 12th IEEE International Conference on Automated Software Engineering (ASEC-97), Incline Village, Nevada, November 3-5, 1997.
- [Shaw & Garlan, 1996] M. Shaw and D. Garlan: Software Architectures. Perspectives on an Emerging Discipline, Prentice-Hall, 1996.
- [Stefik, 1995] M. Stefik: Introduction to Knowledge Systems, Morgan Kaufman Publ., San Francisco, 1995.
- [Sycara et al., 2001] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa: The RETSINA MAS Infrastructure. In: Robotics Institute Technical Report #CMU-RI-TR-01-05, 2001; available online at <http://www.cs.cmu.edu/~softagents/publications.html>
- [Zaremski & Wing, 1997] A. Zaremski and J. Wing: Specification Matching of Software Components, *ACM Transactions on Software Engineering and Methodology*, 6(4):335-369, 1997.

## Footnotes

<sup>1</sup>Vrije Universiteit Amsterdam, Division of Mathematics and Computer Science, De Boelelaan 1081a, 1081 HV Amsterdam, the Netherlands, {borys, dieter, ying}@cs.vu.nl

<sup>2</sup>Stanford University, Stanford Medical Informatics, 251 Campus Dr., Suite 215, Stanford, CA 94305-5479, USA, {crubezy, musen}@smi.stanford.edu

<sup>3</sup>Intelligent Software Components, S.A., iSOCO Madrid, C. Hernandez de Tejada 7, 1st floor, 28027 Madrid, Spain, richard@isoco.com

<sup>4</sup>University of Amsterdam, Department of Social Science Informatics (SWI), Roetersstraat 15, 1018 WB Amsterdam, The Netherlands, bob@swi.psy.uva.nl

<sup>5</sup>The Open University, Knowledge Media Institute, Walton Hal, MK7 6AA, Milton Keynes, United Kingdom, e.motta@open.ac.uk

<sup>6</sup><http://www.swi.psy.uva.nl/projects/ibrow/home.html>

<sup>7</sup>As such, PSMs are a special type of software architectures ([Shaw & Garlan, 1996]).

<sup>8</sup>[www.cs.vu.nl/~upml/](http://www.cs.vu.nl/~upml/). The recent version is described in [Omelayenko et al., 2000]

<sup>9</sup><http://webonto.open.ac.uk>

<sup>10</sup>[www.metacrawler.com](http://www.metacrawler.com)

<sup>11</sup>[www.isi.edu/ariadne](http://www.isi.edu/ariadne)

<sup>12</sup>[www.aifb.uni-karlsruhe.de/Projekte/ontobroker/inhalt\\_en.html](http://www.aifb.uni-karlsruhe.de/Projekte/ontobroker/inhalt_en.html)

<sup>13</sup>Note that we distinguish this from automatically generated Web pages, which are called dynamic opposite to static HTML pages.

<sup>14</sup>Uses is a very important attribute that will be explained later on.

<sup>15</sup><http://dublincore.org/>

<sup>16</sup>[www.w3c.org/xml](http://www.w3c.org/xml)

<sup>17</sup>[www.w3.org/XML/Schema](http://www.w3.org/XML/Schema)

<sup>18</sup>[www.cs.vu.nl/~upml](http://www.cs.vu.nl/~upml)

<sup>19</sup>[www.w3c.org/rdf](http://www.w3c.org/rdf)

<sup>20</sup>See the next section for a detail description

<sup>21</sup><http://protege.stanford.edu/>

<sup>22</sup>[www.ontoknowledge.org](http://www.ontoknowledge.org)

<sup>23</sup><http://www.cs.man.ac.uk/~horrocks/DAML-OIL/>

<sup>24</sup><http://www.cs.cmu.edu/~softagents/retsina.html>

<sup>25</sup><http://www.swi.psy.uva.nl/projects/ibrow/>