

The Internet Reasoning Service: Delivering Configurable Problem-Solving Components to Web Users

Monica Crubézy¹, Wenjin Lu², Enrico Motta², and Mark A. Musen¹

Stanford Medical Informatics
MSOB X-215, 251 Campus Drive
Stanford, CA 94305-5479 USA
+1 650 723-6979
{crubezy, musen}@smi.stanford.edu

Knowledge Media Institute
The Open University
Milton Keynes, MK7 6AA UK
+44 1908 653800
{e.motta, w.lu}@open.ac.uk

ABSTRACT

Existing services on the Web tend to be ‘holistic’. For instance, online services for data analysis are available, but usually it is neither possible to modify the underlying reasoning system, nor to configure it for a different domain, nor to integrate different services to produce new functionalities. The goal of our research is to move beyond the current level of ‘holistic service provision’ on the Web and to develop the technologies and frameworks needed to provide sophisticated online reasoning facilities, configurable for different domains and applications. In this paper we describe the Internet Reasoning Service, a Web-based front-end enabling application developers to prototype knowledge-based applications quickly out of reusable components from distributed libraries. The ultimate aim of our work is to make sophisticated problem solving technology available to a wider audience and provide the level of intelligent support needed to allow rapid generation of web-based reasoning services.

Keywords

Tool support, generic tasks, problem-solving methods, ontologies, reusable components, knowledge systems, knowledge acquisition, knowledge reuse, mapping, adapters, software reuse, Semantic Web, Web services.

INTRODUCTION

The World-Wide Web consists of a very large repository of hypertextual resources. While most resources are relatively static, a significant number of them are dynamic: these are services that provide some functionality for end users, such as booking a plane, converting foreign currencies, or organizing a meeting. Existing services, however, tend to be ‘holistic’. For instance, online services for data analysis are available, but usually it is neither possible to modify the underlying reasoning system, nor to configure it for a different domain, nor to integrate different services to produce new functionalities. From a general point of view the issue is one of developing the necessary technologies and frameworks which can enable service identification and interoperability on the Web [3, 13]. In order to achieve this objective, the competence of each Web service needs to be specified so that brokering agents can then reason about these competence descriptions to match services to user needs and to combine services together. Ongoing work on

the IBROW project¹ is addressing these issues and aims to produce a brokering agent that can locate and configure services on the Web to solve the specific tasks requested by users [2]. The research described in this paper is situated in the context of the IBROW project and is focusing primarily on configurable reasoning services. Our goal is to move beyond the current level of ‘holistic service provision’ on the Web and to develop the technologies and frameworks needed to provide sophisticated online reasoning facilities, configurable for different domains and applications. For instance, we can imagine making available online a generic classification technology, which different communities (e.g., paleontologists, geologists, biologists) or individual users can adapt for specific domains or applications.

Our work is situated in the context of research on knowledge-based system (KBS) development by reuse [1, 4, 5, 7, 9, 14]. Research in this area has identified recurring patterns of reasoning that occur in solving knowledge-intensive problems. These domain-independent problem-solving strategies, or *problem-solving methods (PSMs)* [1, 7, 9, 14, 16] provide standard ways of addressing stereotypical problems, or *generic tasks* [5], such as diagnosis, design, and classification. To foster the reuse of PSMs, structured *libraries* have been developed, in which the methods are indexed and retrieved for different domains and purposes [1, 4, 7, 14, 16, 17]. Comprehensive frameworks and methodologies for developing KBSs from reusable components have also been developed [9, 14, 18, 22]. These approaches take a PSM-centered view of KBS development and provide methods and languages to support development by reusable components. This process involves modelling a user problem as a generic task, configuring an appropriate PSM, and integrating the generic components with domain-specific information.

This reuse-centered approach is now the most established and robust paradigm in knowledge engineering. However, the level of actual component reuse is still quite low, due to the pragmatic barriers affecting component reuse: libraries are few and far between, often they are not available online, and different libraries are not interoperable. The aim of the *Internet Reasoning Service (IRS)* presented in this paper is

¹ <http://www.swi.psy.uva.nl/projects/ibrow/home.html>

to provide a Web-based front-end enabling application developers to prototype knowledge-based applications quickly out of reusable components from distributed libraries. A side-effect of achieving this goal will be simply to make much artificial intelligence (AI) technology available online, thus making its use more widespread. In the first instance, users of the IRS will be primarily reasonably skilled developers, who wish to prototype a KBS application, or simply explore KBS technology. Over time, however, we aim to broaden the target audience of the IRS, by making it possible for less-experienced developers to create applications with the IRS. This goal can be achieved in two ways: i) by providing intelligent assistance for the various phases of the development process, such as acquiring a problem specification, and selecting and configuring an appropriate PSM; and ii) by developing customized versions of the IRS for target communities, such as customizing generic classification technology to support artifact interpretation by archeologists.

We first present our approach to KBS development by component reuse. We then describe the design of the IRS and we report on our ongoing implementation effort. Finally, we relate our research to existing work in the area of Web and knowledge technologies and highlight the main issues still to be addressed.

MODELLING PROBLEM-SOLVING COMPONENTS

Problem-solving libraries contain reusable, operational components, which can be configured into a running application to solve users' problems. This configuration process may involve several activities: mapping generic tasks and PSMs to a domain model (e.g., mapping a generic classification framework to a database of archeological artifacts to produce an artifact-classification application), mapping PSMs to tasks (e.g., selecting a particular abstraction method for performing a data-abstraction task), or, in general, refining existing components (e.g., specializing a data-to-solutions matching component by introducing fuzziness in the matching process). To provide semi-automated support to users in this configuration activity, the IRS relies on the knowledge-level descriptions of the problem-solving components in the library.

The *Unified Problem-solving Method-development Language* (UPML)² is the common framework that we have designed in IBROW for modeling the reusable components of a knowledge system [8, 20]. UPML distinguishes the three kinds of components involved in the construction of a knowledge system (see Figure 1): *tasks*, *PSMs* and *domain models*. Each of these components is described according to a particular underlying *ontology*—a specification of the concepts important to describe the component. Each knowledge component also has a *pragmatics* definition,³

which allows developers to specify informal properties, such as authors, format and location.

PSM and task components define a set of *input* and *output roles*, as well as set of logical formulas, which provide a precise functional specification of their competence. In the case of a task component, such specification includes a description of the associated *goal*, the relevant *preconditions* on the task inputs and the *assumptions* introduced by the component on the available domain knowledge. For example, a classification task specification may introduce the assumption that only one solution to the task exists in the domain [16]. The specification of a PSM component follows a similar modelling framework, with the exception that it includes a statement on the *postconditions* associated with the PSM, rather than a goal statement. Postconditions normally express a functional relation between the outputs and inputs of the PSM. For instance the specification of an optimal heuristic classifier states that the output solutions are both admissible and optimal with respect to the input solution space. Domain components model the domain knowledge that is used by task and PSM components in the reasoning process. Figure 2 is an excerpt of the UPML specification of a PSM. For the sake of readability and for reasons of space limitation, the figure only shows informal descriptions of the components and omits the formalization aspects.

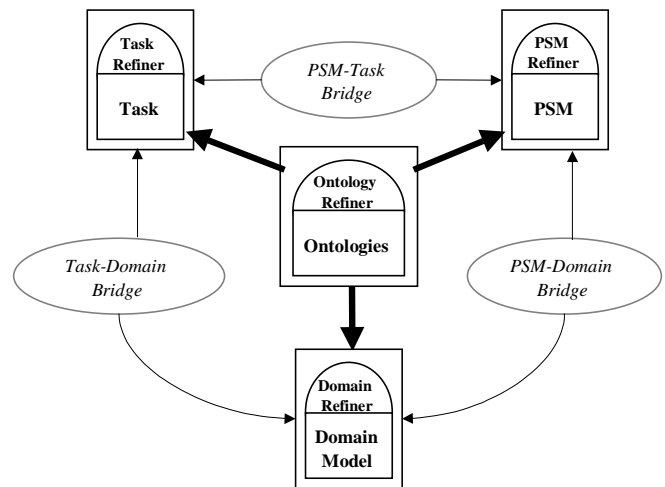


Figure 1. The UPML framework for modeling properties and interfaces of reusable knowledge components. Each PSM, Task and Domain Model component relies on an Ontology component.

Knowledge components model the generic, reusable building blocks of a KBS. These building blocks can be combined together through the use of explicit architectural elements—*adapters*. *Bridges* are a first kind of adapter, which connect two different kinds of components, such as a PSM and a domain model, by defining *mapping relations* between the underlying ontologies of both components. *Refiners* are a second kind of adapter, which express the step-wise adjustment of a knowledge component into more specialized ones. Adapters are central elements in our approach: they provide explicit means to model the reuse of

² <http://www.cs.vu.nl/~upml/>

³ <http://www.dublincore.org/>

knowledge components in different systems - see [11, 23] for more details on adaptation in UPML.

UPML only provides the framework needed to specify the (meta-)properties of problem solving components and it is open with respect to the concrete formalism used to represent the contents of the component themselves. Hence, different library providers are free to specify their components in the language they deem appropriate, as long as they subscribe to the UPML (meta-)ontology. Yet, interoperability among components expressed in different languages demands attention. In particular, knowledge-level interoperability requires shared ontologies and either common modelling languages or the use of standard knowledge-level APIs, such as OKBC [6]. Symbol-level interoperability requires the use of standard integration frameworks, such as CORBA⁴ (see ‘Discussion’ section).

<p>PSM Pragmatics: Title: heuristic-optimal-solution-classifier Ontologies: heuristic-classification-method-ontology Input roles: ocml-element: name: has-observables; type: observables ocml-element: name: has-candidate-solutions; type: solution-space ocml-element: name: has-match-criterion; type: match-criterion ocml-element: name: has-solution-exclusion-criterion; type: candidate-exclusion-criterion ocml-element: name: has-abstractors; type: abstractors ocml-element: name: has-refiners; type: refiners Output roles: ocml-element: name: has-solutions; type: solution-space Competence: Preconditions: ocml-formula: documentation: the abstraction hierarchy has to be free of cycles ocml-formula: documentation: the exclusion criterion is correct ocml-formula: documentation: existence of a solution in the virtual solution space ocml-formula: documentation: the refinement hierarchy has to be free of cycles Postconditions: ocml-formula: documentation: the output solutions are both admissible and optimal Subtasks: abstraction rank-solutions refinement Operational Description: Intermediate roles: ocml-element: name: has-abstract-observables; type: observables ocml-element: name: has-ranked-candidates; type: solution-space ocml-element: name: has-current-solutions; type: solution-space Programs: ocml-program: documentation: heuristic-optimal-solution-classifier-body ocml-program: documentation: complete-optimal-search-procedure</p>

Figure 2. The heuristic optimal-solution classifier PSM.

Tool support for specifying knowledge components in UPML is provided by a special-purpose editor [20]. This editor is based on Protégé-2000⁵ [18], a tool that supports ontology development and ontology-driven knowledge-base construction. The Protégé-UPML editor can be customized easily for a particular contents-specification language, using the suite of reusable navigation and display facilities included in Protégé-2000. In particular, we produced a customization of the Protégé-UPML editor, which allows content specification in the OCML modelling language [14]. UPML also has a Web-compatible syntax, in the form of an RDF Schema⁶, automatically generated by Protégé-2000 [19]. Thus, library providers can publish knowledge components as resources on the Web.

⁴ <http://www.corba.org/>

⁵ <http://protege.stanford.edu/>

⁶ <http://www.w3.org/TR/rdf-schema/>

THE INTERNET REASONING SERVICE

The goal of the IRS is to provide Web-based support to users who want to make use of existing reasoning resources to achieve tasks in their application domain. The IRS provides different levels of user support, from interactive browsing and navigation facilities, which enable manual selection and configuration of reasoning components, to intelligent, semi-automated assistance in building an executable application. In the following, we detail the issues involved in each step of the process model associated with the IRS (see Figure 3), by illustrating a sample use scenario. For the sake of explanation we assume that the user is an archeologist who wishes to configure an available library of components for classification problem solving [16], to develop automated support for interpreting and classifying artifacts dug from Roman sites in Britain.

Task Selection

A IRS user initially needs to characterize the specific goal of her application in terms of one of the task models known to the IRS, e.g., *classification*. The IRS supports this activity by allowing the user to browse the library of task models, by providing user-friendly descriptions of the various task models and by providing examples of the ways task models have been used in previous applications. Currently, we do not envision providing support for this phase beyond good browsing and navigation facilities. Task selection means adopting a particular—not necessarily unique—viewpoint to impose over an application. For instance, the solutions to the benchmark Sisyphus-I office-allocation problem described in [12] included both i) approaches that modeled the office allocation problem as a design problem and ii) approaches that modeled it as a classification problem. In other words, a specific application does not necessarily fall into a particular task model. Rather, it is a task model that provides a conceptual viewpoint for characterizing an application and for focusing the knowledge-acquisition process.

Task Configuration

Having chosen a task model, say *single-solution-classification*, the user has to perform the task configuration activity. This activity consists of instantiating some or all of the task inputs for a particular domain. Roles fall into two categories, *case inputs* and *application inputs* (aka ‘static knowledge roles’). For example, the *observables* input roles of *classification* are typically a case input: each time the application is run, the input is a different set of observables to classify (e.g., a different archeological data). Application inputs instead tend to be fixed for each application. For instance, in most cases the *space of possible solutions* that is an input role to *classification* will not change for each run of a classifier. However, even from this simple analysis it is clear that case and application inputs are just pragmatic labels. In particular, i) adopting a modular approach to system development by reuse and ii) providing support tools such as the IRS, allows to experiment with different

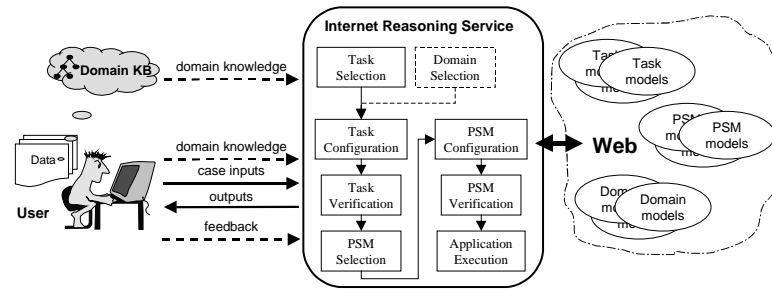


Figure 3. The IRS: use scenario and process model.

configurations of an application (e.g. different solution spaces) and therefore the distinction between case and application inputs loses its importance.

When performing task configuration, the user still needs to assign ‘operational labels’ to the input roles, given that case and application inputs can be configured at different stages of the process. Specifically, although application inputs need to be configured during the task-configuration phase, case inputs only need to be acquired at application-execution time. However, if case inputs are to be provided in a form different from the one specified by the task ontology, the relevant mapping relations have to be defined. In our example, it is necessary to map the notion of *observable* in the task ontology to the attributes of archeological artifacts, such as *provenance*, *color*, *material*, *decorations*, *weight* and *preservation-state*. In the application-execution step, these attributes can then be instantiated by case-data values.

Contrary to case inputs, application inputs always have to be configured during the task-configuration activity. In the easiest case, the value of the application role is provided directly by the user, consistent with the task ontology. For example, we may want to define a new *match criterion* (a mechanism to assess the degree-of-fit between data and solutions) and the obvious way to do it is to express the new metric in terms of the concepts and relations provided by the task ontology. An even simpler case is to select an existing criterion from the library. The other remaining case is the one in which role-configuration requires the developer to construct a mapping to the domain knowledge. For instance, the role *solution-space* could be mapped to a hierarchy of artifacts, such as different classes of Roman pottery. An important issue for the IRS is therefore to reduce the complexity of the configuration process by providing high-level mapping clichés (see ‘Discussion’ section). Outputs behave like case-dependent inputs: clearly, they cannot be instantiated with actual knowledge structures at this stage. However, mappings may be required so that the output of the task conforms to the domain ontology.

The IRS stores the result of the task-configuration step as a set of *task-domain bridges* created for the current domain. These bridges are stored in an ontology created automatically from both the relevant task ontology and the current domain ontology—see section below. Users are able to backtrack to the task-selection step if they discover that

their domain does not match the expectations of the task model, typically by selecting another refinement of the high-level task model. In this case, the IRS keeps the mappings that still hold for the newly selected task.

Domain Selection

The IRS provides a range of pre-existing domain models stored in available UPML-compliant libraries, from which users simply can make a selection. The advantage of selecting an available domain model is that, in many cases, no configuration effort may be needed, given that mappings themselves are often reusable. In our archeological example we can see that all the mappings discussed earlier tend to be generic (observables \Rightarrow artifact features; solution space \Rightarrow hierarchy of artifact types). Alternatively, the IRS enables users to provide domain knowledge directly by filling-in knowledge-acquisition forms. Another possibility is that users provide their own knowledge base or database as input to the application development process. In this case, the user has to take care of solving the related interoperability problems, both at the knowledge and at the symbol level (see ‘Discussion’ section).

Task Verification

This step consists in checking the assumptions relevant to the chosen task model. The IRS performs this step automatically by running an assumption-checking engine on the mapped domain inputs and the assumptions of the task. The IRS notifies users about the domain inputs that do not satisfy the assumptions of the task. In this case, the IRS guides users back to the task-configuration step, so that they can re-specify the erroneous inputs. However, it is important to note that not all assumptions are necessarily verifiable. For instance, our task model for *single-solution-classification* includes the assumption that only one solution exists in the target domain. Clearly this assumption cannot be verified in the general case.

PSM Selection

This step focuses on selecting a PSM that can realize the configured task. In a way similar to that in the task selection step, the IRS provides users with the list of PSMs that match the configured task specification and supports users in browsing the description of each PSM.

The IRS uses different means to compute the list of candidate PSMs for solving the task. A simple case is the one in which a library developer simply states, as part of a

PSM specification, that the PSM in question can be used to solve a particular class of tasks. More generally, the library may include *PSM-task bridges*, which encode the connection between the task and a PSM that can realize it, together with a set of mapping relations between the ontologies of the task and the PSM. For example, a bridge connects the *single-solution-classification* task and the *heuristic-admissible-solution-classifier* PSM. Another possibility is that the IRS can try to carry out a competence-matching process, by reasoning about the competence specification of a PSM and the goal specification of the current task. In general, such reasoning requires full first-order logic theorem-proving support, which is not part of the current implementation of the IRS. An important issue to address in the future is therefore to identify or design contents specification formalisms, which strike the right balance between expressive power and efficiency in support of the competence matching process (see ‘Discussion’ section). Finally, users themselves can choose among all available PSMs. In this case, the IRS supports the creation of a *PSM-task bridge* that maps the inputs and outputs of the configured task and of the selected PSM.

PSM Configuration

This step is similar to the task-configuration step, at the level of the selected PSM: the IRS guides users in specifying the domain entities that fill-in the input-output roles of the PSM. Some of the input-output roles for the PSM are “inherited” from the task configuration, through their corresponding *PSM-task bridge*. In addition, the selected PSM may define supplemental input roles, especially case-independent ones. For example, the *heuristic-admissible-solution-classifier* PSM defines the notion of an *abstractor*. In our archeology domain, this means that we may want to abstract from the specific geographical site in which the artifact was found to a more coarse-grained area, which may be more useful to classify the artifact. The IRS supports the acquisition of domain-method mapping knowledge in a way similar to the task-domain mapping for task configuration. The IRS also stores the result of the PSM configuration as a *PSM-domain bridge* created for the user’s domain.

PSM Verification

This step focuses on verifying that the selected PSM can be applied to the task and the domain in accordance with the associated assumptions and the results of the configuration process. This step is essentially the same as in the task-verification step. For instance, in the case of our chosen heuristic classifier, the assumption-checking engine will verify that the solution hierarchy (i.e., the hierarchy of artifact types) contains no cycles.

As a result, the IRS notifies users about the domain inputs that do not satisfy the assumptions or preconditions of the PSM. In this case, the IRS guides users back either to the task-configuration step, to re-specify the task-level inputs that are not satisfied or to the PSM-configuration step, to re-specify the method-level inputs that are not satisfied.

Application Execution

This final step consists in running the configured PSM to realize the specified task, with domain case data entered by user. The IRS first acquires case data from the user and instantiates the case inputs of the PSM with those case data by interpreting the domain-PSM mapping relations. The IRS also checks the preconditions of the PSM and task on the mapped case data. The IRS then invokes the PSM code with the mapped inputs, by running a code interpreter either locally or remotely. Finally, the IRS fills in the domain outputs with the results of PSM execution, interpreted with domain-PSM mapping relations.

CURRENT IMPLEMENTATIONS OF THE IRS

We are currently realizing the design described above as two prototype tools, which are being developed at the two partners’ sites and exploit knowledge, web and HCI technologies already available to the two groups.

The IRS-KMi tool

The *IRS-KMi* implementation of the IRS capitalizes on existing knowledge and Web technologies available at the Knowledge Media Institute (KMi). These include the WebOnto ontology editor⁷, the OCML modelling language [14] and a large library of ontologies and problem solving components available online through the WebOnto editor and accessible through a knowledge-level API. In addition, the infrastructure also includes support for delivering knowledge-based services on the web, support for ontology-driven knowledge acquisition (KA) and support for formalism-independent query answering facilities [15]. The underlying reasoning mechanism of IRS-KMi is based on the OCML modelling and execution environment. In particular, the set of base ontologies supporting OCML modelling also includes an ontology which characterizes task and PSM components, in accordance with the UPML framework. As a result, it is possible to use OCML both to describe the UPML-level properties of an application and also to describe the sub-UPML level (i.e., the actual contents of a UPML specification). IRS-KMi is directly connected to the OCML-LispWeb server, an online interpreter that can process OCML requests.

An initial complete implementation of the IRS-KMi has been developed, which supports all activities in the process model described in the previous section. Browsing support and ontology-driven KA facilities make it possible for a user to instantiate role values directly, or to select pre-existing ones from a library. Mapping support is also provided, both through OCML and through high-level clichés. The latter includes clichés such as ‘Map to Hierarchy’, which allows the user to simply state that a particular domain hierarchy instantiates a particular role. IRS-KMi automatically generates the right OCML definitions to operationalize the mapping. OCML also provides mapping constructs, for class and relation mappings, as well as ‘free-form’ constructs for expressing arbitrarily complex mappings [14]. However,

⁷ <http://webonto.open.ac.uk>

task and PSM configuration is still reasonably hard for non-expert users: we plan to extend the library of mapping clichés currently available to facilitate this activity.

IRS-KMi also supports assumption-checking, thanks to the powerful constraint checking mechanism included in OCML. A limited form of competence matching is also supported, which relies on explicitly defined relations between tasks and PSMs. Finally, IRS-KMi supports the application-execution step by invoking the OCML environment to interpret the PSM code on the case data inputs that the user enters in special-purpose forms.

The IRS-Protégé tool

The *IRS-Protégé* tool is implemented as an extension to Protégé-2000 [18]. IRS-Protégé interfaces a domain knowledge base to libraries of PSMs, specified in UPML, with the Protégé-based UPML editor. IRS-Protégé assumes that users provide domain knowledge in a Protégé-compatible form. IRS-Protégé benefits from the full-fledged knowledge-acquisition support of Protégé to enable users to create or select a domain knowledge base, and to enter additional knowledge as needed in the process.

IRS-Protégé supports the selection of a task and a PSM from connected libraries in a highly graphical way. Driven by browsable diagrams that picture the input–output and decomposition structure of PSM and task components, all elements of a specification, such as the underlying ontology, are accessible from the IRS-Protégé.

IRS-Protégé supports the task and PSM-configuration steps by providing a clear methodology for mapping inputs and outputs of the component to corresponding domain entities. Users instantiate *declarative mapping relations* between entities of the task or PSM ontology and of the domain knowledge base. Our methodology provides a typology of mapping-relation templates—a *mapping ontology* [21], which covers simple renaming mappings, as well as lexical expressions or complex functions of entities and their attributes. IRS-Protégé guides users in this activity based on the underlying mapping ontology and the input–output definition of the task or PSM. For instance, a mapping relation can be created to translate a domain hierarchy of *object types* (such as Roman pottery classes) into the *solution space* input of *classification*. IRS-Protégé also enables users to acquire additional domain knowledge on the fly, to fill-in roles defined by the task or PSM that do not have direct equivalents in the domain. In addition, IRS-Protégé incorporates a *mapping interpreter* [21], which applies the mapping relations to the domain entities to create entities understandable by the task or PSM. These ‘mapped domain entities’ can then be checked against task and PSM assumptions and used for application execution.

DISCUSSION

Our approach is closely related to the emerging area of ‘configurable Web services’, although we adopt a perspective very different from existing work.

Configurable Web Services

A number of approaches are emerging to provide online, configurable services on the Web. One kind of approach concerns the syntactic-level interoperability of Web components, based on standardized protocols for component-functionality description and communication (e.g., UDDI⁸, WSDL⁹, Microsoft’s .Net¹⁰, Sun’s Jini¹¹). Advertizing the services that they can provide, software components can be discovered by other components, usually through the use of a centralized look-up service. Although these distributed Web services can cooperate together to achieve a certain goal, they typically provide only predefined, limited functionalities.

Another, more semantic approach, stems from earlier research in agent technology such as KQML [10]. Web services usually declare their capabilities (e.g., finding information about computers) and requirements (e.g., input format, communication mode) in a more or less expressive description language. Special-purpose agents can then mediate services to users, by locating, matching and connecting services that are available online [23]. The use of shared ontologies of typical services and plan-like procedures further enhances the capacity of agents to perform tasks for users, by dynamically chaining primitive steps realized by corresponding Web sites [13]. Besides the fact that the services in question are relatively fine-grained, ad-hoc procedures that solve circumscribed problems, such approaches make the assumption that users and services are in the same ‘world,’ ignoring the issue of mapping concepts between domain, task and method ontologies.

Our goals in this research ultimately also fall into providing configurable information-processing services on the Web. However, the services that we propose to provide are complete, robust, and reusable methods for solving common knowledge-intensive tasks, such as design, planning and classification, in various domains. In that sense, our approach is closer to supporting users in operating online scientific-data analysis packages. Common available packages, such as for image processing, often lack flexibility in the tasks and procedures that can be performed, or require too deep an understanding from their users. Our ontology-level approach to component adaptation provides a full framework to model problem solvers that the IRS can then reuse for constructing knowledge-based applications for different domains and contexts. Also, the user-centered support that the IRS provides can assist users efficiently.

Component Configuration

A central part of our approach concerns the configuration of task and PSM components for the user’s domain. As explained before, creating explicit architectural elements that isolate the specific configuration knowledge for a set of

⁸ <http://www.uddi.org/>

⁹ <http://msdn.microsoft.com/xml/general/wsdsl.asp>

¹⁰ <http://www.microsoft.com/net/>

¹¹ <http://www.sun.com/jini/>

components maximizes the reusability of these components. However, this activity is complex: it requires that users understand the connections and adaptations that need to be specified about knowledge components so that they can be assembled into a working system. Yet this activity is complicated to automate: ontology-merging techniques cannot be applied easily to this problem, because they usually assume that source ontologies cover similar domains. Typically, domain knowledge and problem-solving knowledge are expressed with very different vocabularies and structures, which are not easy to map. Domain knowledge is sometimes even insufficient, and needs to be augmented with additional concepts to match problem-solving requirements [7, 14].

One of the goals of the IRS is to alleviate the complexity involved in the configuration activity. The IRS provides support adapted to the level of understanding that users may have in the configuration process, allowing for either a direct reuse of pre-existing configurations or a more advanced experimentation. The IRS guides users in the configuration process by providing a range of mechanisms for mapping domain entities to inputs and outputs of tasks and PSMs, from selecting default values to instantiating structured mapping-relation templates. In particular, the mapping ontology and mapping interpreter provided by IRS-Protégé and the ‘mapping clichés’ and general-purpose mapping constructs provided by IRS-KMi go some way towards facilitating the configuration process. Nevertheless, as already mentioned, more research is needed to facilitate the configuration process further.

To support further the process of mapping domains to tasks and PSMs, we are also investigating the use of pre-existing cases of task or PSM application as templates for configuring the task or PSM for another use. Currently, users can browse such example cases from the IRS, which compiles them from the definition of the component and its existing bridge connections. The IRS also enables users to review the summary of their settings for a particular task-PSM configuration. The IRS could store the result of configuring a task and a PSM for a domain as an ‘application case,’ compiled out of the set of mapping bridges created for the application, plus a set of example test cases. Building such a history of applications associated to a component would enable users to improve their understanding of the component gradually. Furthermore, these cases could be used as reusable templates that can be adapted for slightly different situations.

Competence Matching

An important issue that the IRS currently under-addresses concerns the retrieval of PSMs that can realize the user-specified task. The list of possible candidate PSMs can be computed by a competence-matching process. This process involves reasoning over the assumptions and competence description of both the configured task and the available PSMs. Candidate PSMs are selected if their postcondition statement fulfill the goal of the task to be solved, and if their

preconditions do not contradict the assumptions of the task. Agent-based approaches such as the RETSINA project include an efficient competence-matching process, which retrieves agents on the Web, based on the services that they advertize in the LARKS language [23]. However, there is a trade-off here, as LARKS provides relatively weak representational machinery to express the formal competence of agents.

Furthermore, the IRS must address the possible mismatch between the ontologies of the task and those of the available PSMs. Although usually not as different as the ontologies of domains and tasks can be, the ontologies of tasks and PSMs can express their competence roles with slightly different concepts and terms. These concepts need to be mapped through explicit relations, which in turn need to be interpreted as part of a competence-matching process. Finally, when configuring a PSM out of several sub-PSM components, the IRS must provide support in matching the input-output competence of available PSMs to each other and to the specified subtasks. In this case too, ontological mismatches must be resolved by mapping relations and interpreted in the matching process.

Interoperability

Component-interoperability issues are at the center of the IRS—and, more generally, the IBROW—approach. Typically, each problem-solving library uses a specific knowledge-modeling language to express properties of its components, and includes pieces of code in a particular programming language. The federated UPML framework alleviates this difficulty only at the modeling level. At the execution level, configured components—including possible domain knowledge provided by users in a proprietary format—need to be assembled in an interoperable system. The use of distributed-computing standards, such as CORBA, has proven to be a successful means for encapsulating problem-solving components and combining them into platform-independent, operational systems [11] and indeed our initial IBROW prototype also used a CORBA-based architecture to operate heterogeneous components [2]. However, CORBA does not provide ways to describe semantics in the interface specification and therefore more work needs to be carried out to integrate CORBA technology into our ontology-based approach.

CONCLUSION

The IRS approach strongly builds on proven knowledge-engineering techniques. The key asset that the IRS adds to these techniques is a principled methodology and user-centered tool support for prototyping knowledge systems by configuring reusable components from structured libraries. Although some symbol-level and knowledge-level interoperability issues remain to be addressed, we believe that the IRS approach will foster the wide diffusion of knowledge-engineering paradigms, by lowering the current cognitive barriers to applying them.

The Web is a vehicle that will allow us to perform a large evaluation experiment of the IRS approach. At the same

time, the heterogeneous, distributed and versatile nature of the Internet will raise new issues that we will need to address in our approach. The Web will challenge us to improve the IRS towards incorporating more automated support, with simplified procedures and customized interactions, so that less-experienced users also can benefit from advanced distributed problem-solving services.

Finally, although the current focus of our research is on facilitating user access to sophisticated reasoning services, which subscribe to heterogeneous ontologies, we anticipate that it will also be possible to harness the technologies produced in this research in support of more 'mundane' levels of service provision on the web. In these cases we can take advantage of scenarios in which ontology mapping and service interoperability issues are greatly simplified.

ACKNOWLEDGEMENTS

This work is part of the IBROW Project, funded by the IST program of the European Community. We would like to thank all our collaborators in the project for the many stimulating discussions on the topics discussed in this paper.

REFERENCES

1. Benjamins, V.R. Problem Solving Methods for Diagnosis, University of Amsterdam, 1993.
2. Benjamins, V.R., Plaza, E., Motta, E., Fensel, D., Studer, R., Wielinga, R., Schreiber, G., Zdrahal, Z. and Decker, S., An intelligent brokering service for knowledge-component reuse on the World-Wide Web. in *Eleventh Banff Workshop on Knowledge Acquisition, Modeling, and Management*, (1998).
3. Berners-Lee, T., Fischetti, M. and Dertouzos, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Harper, San Francisco, 1999.
4. Breuker, J.A., and van de Velde, W. (ed.), *The CommonKADS Library for Expertise Modeling*. IOS Press, Amsterdam, 1994.
5. Chandrasekaran, B. Generic tasks for knowledge-based reasoning: High-level building blocks for expert system design. *IEEE Expert*, 1 (3). 23-30.
6. Chaudhri, V.K., Farquhar, A., Fikes, R., Karp, P.D. and Rice, J.P., OKBC: A programmatic foundation for knowledge base interoperability. in *Fifteenth National Conference on Artificial Intelligence*, (Madison, Wisconsin, 1998), AAAI Press/The MIT Press, 600-607.
7. Eriksson, H., Shahar, Y., Tu, S.W., Puerta, A.R. and Musen, M.A. Task modeling with reusable problem-solving methods. *Artificial Intelligence*, 79. 293-326.
8. Fensel, D., Benjamins, V.R., Motta, E. and Wielinga, R., UPML: A Framework for knowledge system reuse. in *International Joint Conference on Artificial Intelligence (IJCAI-99)*, (Stockholm, Sweden., 1999).
9. Fensel, D. and Motta, E. Structured Development of Problem Solving Methods. *To appear in IEEE Transactions on Knowledge and Data Engineering*.
10. Finin, T., McKay, D., Fritzson, R. and McEntire, R. KQML: An Information and Knowledge Exchange Protocol. in Fuchi, K. and Yokoi, T. eds. *Knowledge Building and Knowledge Sharing*, Ohmsha and IOS Press, 1994.
11. Gennari, J.H., Cheng, H., Altman, R. B., Musen, M.A. Reuse, CORBA, and Knowledge-Based Systems. *International Journal of Human-Computer Studies*, 49 (4). 523-546.
12. Linster, M. Problem Statement for Sisyphus: Models of Problem Solving. *International Journal of Human-Computer Studies*, 40 (2). 187-192.
13. McIlraith, S.A., Son, T.C. and Zeng, H. Semantic Web Services. *IEEE Intelligent Systems*, 16 (2). 46-53.
14. Motta, E. *Reusable Components for Knowledge Modelling: Principles and Case Studies in Parametric Design*. IOS Press, Amsterdam, 1999.
15. Motta, E., Buckingham-Shum, S. and Domingue, J. Ontology-Driven Document Enrichment: Principles, Tools and Applications. *International Journal of Human-Computer Studies*, 52 (6). 1071-1109.
16. Motta, E. and Lu, W., A Library of Components for Classification Problem Solving. in *PKAW 2000: The 2000 Pacific Rim Knowledge Acquisition Workshop.*, (Sydney, Australia, 2000).
17. Musen, M.A., Modern Architectures for Intelligent Systems: Reusable Ontologies and Problem-Solving Methods. in *AMIA Annual Symposium*, (Orlando, FL, 1998), 46-52.
18. Musen, M.A., Ferguson, R.W., Grosso, W.E., Noy, N.F., Crubézy, M. and Gennari, J.H., Component-Based Support for Building Knowledge-Acquisition Systems. in *Conference on Intelligent Information Processing (IIP 2000) of the International Federation for Information Processing World Computer Congress (WCC 2000)*, (Beijing, China, 2000).
19. Noy, N.F., Sintek, M., Decker, S., Crubézy, M., Ferguson, R.W. and Musen, M.A. Creating and Acquiring Semantic Web Contents with Protégé-2000. *IEEE Intelligent Systems*, 16 (2).
20. Omelayenko, B., Crubézy, M., Fensel, D., Benjamins, V.R., Wielinga, B.J., Motta, E., Musen, M.A. and Ding, Y. UPML: The Language and Tool Support for Making the Semantic Web Alive. in D. Fensel, J.H., H. Liebermann, and W. Wahlster (eds.) ed. *To appear in: Creating the Semantic Web*, MIT Press, 2001.
21. Park, J.Y., Gennari, J.H. and Musen, M.A., Mappings for Reuse in Knowledge-Based Systems. in *Eleventh Banff Knowledge Acquisition for Knowledge-Based Systems Workshop*, (Banff, Alberta., 1998).
22. Schreiber, A.T., Akkermans, J.M., Anjewierden, A.A., de Hoog, R., Shadbolt, N.R., van de Velde, W., and Wielinga, B.J. *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press, Cambridge, 2000.
23. Sycara, K., Lu, J., Klusch, M. and Widoff, S., Matchmaking among Heterogeneous Agents on the Internet. in *AAAI Spring Symposium on Intelligent Agents in Cyberspace*, (Stanford, CA, 1999).