Problem-Solving Methods and Cooperative Information Agents

Mario Gomez, Enric Plaza, Chema Abasolo

IIIA - Artificial Intelligence Research Institute
CSIC - Spanish Council for Scientific Research
Campus UAB, 08193 Bellaterra, Catalonia, Spain.
{enric,mario,abasolo}@iiia.csic.es
http://www.iiia.csic.es

Abstract. Cooperative Information Agents and modern information systems in general have to access large amounts of information distributed across multiple heterogeneous sources. A great challenge of such systems is to evolve by adding new information sources or adapting the existing components for different domain knowledge. We propose the UPML framework as a methodology to build Information Agents by reusing a library of problem-solving components that are defined in a domain-independent manner. Moreover, the UPML language is used as an Agent Capability Description Language (ACDL) suitable to configure an agent-based application.
From this approach, a new application is built by linking the components of the library with a particular domain and a collection of heterogeneous information sources. Adaptability and dynamic configuration of such a system is achieved by reasoning about the UPML specification of the agent capabilities. Independence of the domain and semantic interoperability are achieved by using ontologies and bridges (mappings between ontologies), while independence from the information sources is based on the use of ontologies to achieve semantic heterogeneity, and wrappers to achieve syntactic interoperability.

1 Introduction

Modern information systems shall manage or have access to large amounts of information and computing services. The different system components can conduct computation concurrently, communicating and cooperating to achieve a common goal. These systems have been called *Cooperative Information Systems*[18]. One major goal of this field is to develop and build information systems from reusable software components. This goal can be achieved by assembling information services on demand from a montage of networked legacy applications and information sources [23]. A promising approach to this problem is provided by Cooperative Information Agents, computational software entities that accesses one or multiple, heterogeneous and distributed information sources [14]. A critical requirement for these systems is the independence of the reasoning

components from the domain knowledge. We propose UPML as a framework for developing such domain-independent information components in the context of multi-agent systems, using the notion of agent-mediated institutions. The key ideas driving our work are the use of the UPML meta-ontology as an Agent Capability Description Language —a language to describe the functionality offered by agents, plus the configuration of an application at two levels: configuration at the knowledge modelling level, that is called *brokering*, and the operationalization of the configuration, that is achieved by forming a team of agents suitable for that configuration.

From this approach, agent capabilities described in UPML becomes a *library* of problem-solving methods that is matched against a task specification describing the requirements of the problem to be solved, therefore a new configuration of the application is built for each request to solve a problem. The entire application is built by linking the components of the library with a particular domain and a collection of heterogeneous information sources. Independence of the domain and semantic interoperability are achieved by using ontologies and bridges (mappings between ontologies), while independence from the information sources is based again in the use of ontologies to overcome semantic heterogeneity and wrappers to achieve syntactic interoperability.

We have built an application that shows how to build Cooperative Information Agents by using the UPML framework: the Web Information Mediator. The overall goal of WIM is to provide a mediation service for information tasks of a professional user looking for bibliographic references. A mediator is an agent that offers an added value to the information sources it accesses[21]. Typical services offered by a mediator include selection of information sources, information retrieval, and fusion of information from different sources. We have built a library of components to solve this kind of tasks belonging to the field of Intelligent Information Integration (I3) [22]. WIM is a multi-agent information system dealing with the problem of looking for medical literature, thus it has been built connecting the components in the I3 Library with a medical domain and some web-based information sources that serve bibliographic references in medicine. We want to emphasize the independence between the library (I3), the domain knowledge (medicine) and the external information sources (Pubmed, Healthstar, iSOCO).

The WIM application is configured on-the-fly for each problem request, and the configuration is made according to the kind of problem to be solved and the preferences of the user requesting it. We view the overall process of solving a problem decomposed in three different activities:

- *Problem specification* and *Brokering*: the problem is specified by the user and transformed into a UPML problem specification. Brokering is the process of obtaining a new configuration of the application for that problem specification.
- *Team formation*: the configuration obtained during the brokering process is operationalized by forming a team of problem solving agents able to solve the problem according with that configuration.

– *Problem solving*: the activity of solving the problem for which the application is configured, this activity is carried on by a team of problem-solving agents formed during the team formation activity.

The overall description of UPML is presented in §2, the I3 Library is briefly described in §3. The WIM application is described at the conceptual level in §4. The WIM multi-agent architecture is described in §5, while the process of configuring WIM is divided in two sections:

§6 addresses the configuration at the UPML level, and §7 describes the operationalization of a configuration. Finally, some conclusions are summarized in §8.

2 An overview of UPML

Problem-solving methods provide reusable architectures and components for implementing the reasoning part of knowledge-based systems. The Unified Problem-solving Method description Language (UPML) has been developed to describe and implement such architectures, and components to facilitate their semiautomatic reuse and adaptation. Spoken in a nutshell, UPML is a framework for developing knowledge intensive reasoning systems based on libraries of generic (domain-independent) problem-solving components.

The goal of *software architectures* is learning from system developing experience in order to provide the abstract recurring patterns for improving further system development. As such, software architectures contribution is mainly methodological in providing a way to specify systems. A software architecture has the following elements: (i) components, (ii) connectors, and (iii) a configuration of how the components should be connected [11]. UPML is a modern software architecture for knowledge systems[8]. The connectors in UPML are called *bridges*, the architecture is shown in Fig.1. Furthermore, UPML offers a meta-ontology to describe the components in the architecture. This ontology defines de concepts needed to describe each component, but the object language inside UPML concepts is not defined, we are free to choose the language that better fits our requirements or preferences, like for example a First-Order Logic or Feature Terms.

First we will briefly explain the basic elements of the UPML architecture, together with the main UPML concepts used to describe each component, and second, how this architecture addresses the problem of the independence of the reasoning components from the domain.

2.1 UPML components

The UPML software architecture consists of six different elements (see Figure 1: *tasks* define the problems that should be solved (the "what") by a system,*problem-solving methods* define the reasoning components of the system (the "how"), and a *domain model* describes the domain knowledge used by tasks and problem-solving methods (the "with").

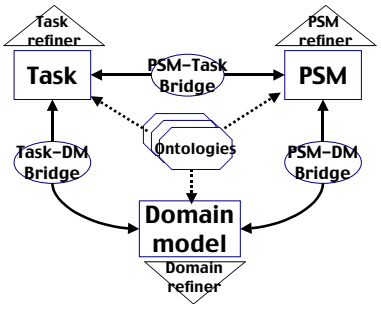


Fig. 1. The UPML software architecture.

Tasks define the problem that should be solved by the system. A description of a task specifies goals that should be achieved in order to solve a given problem, assumptions about domain knowledge and preconditions on the input. Preconditions are conditions on dynamic inputs, while assumptions are conditions on knowledge consulted by the reasoner, but not manipulated.

Task

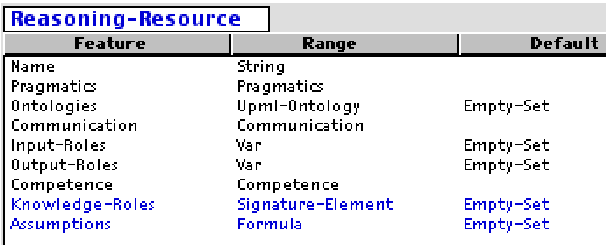
Feature	Range	Default
Name	String	
Pragmatics	Pragmatics	
Ontologies	Upml-Ontology	Empty-Set
Uses	Task	Empty-Set
Input-Roles	Var	Empty-Set
Output-Roles	Var	Empty-Set
Competence	Competence	
Assumptions	Formula	

Fig. 2. UPML description of a Task

Problem-Solving Methods (PSM) describe which reasoning steps and which types of knowledge are needed to perform a task. A PSM specifies a particular way to solve a task. The main attributes of a PSM are the input/output roles, plus the preconditions and postconditions to be fulfilled by the input and output roles. There are two subclasses of PSM: *Problem Decomposers* and *Reasoning Resources*. Problem Decomposers specify decomposition of tasks into subtasks. Reasoning Resources specify how to solve a task, it does not describe its internal structure, which is regarded as an implementation aspect.

Domain Models describe the domain knowledge that is used by tasks and PSMs. A domain model is characterized by a domain knowledge and its properties.

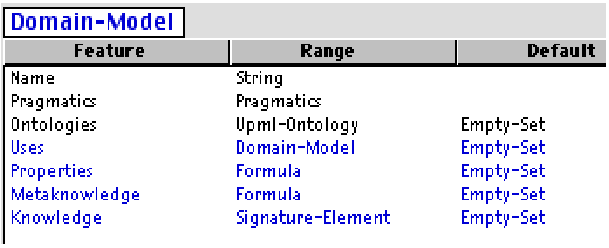
The UPML framework has the notion of *Ontology*. An ontology defines a terminology and its properties. UPML has different ontologies, one for task's description, one for PSM's description and the domain ontology for domain models.



Reasoning-Resource

Feature	Range	Default
Name	String	
Pragmatics	Pragmatics	
Ontologies	Upml-Ontology	Empty-Set
Communication	Communication	
Input-Roles	Var	Empty-Set
Output-Roles	Var	Empty-Set
Competence	Competence	
Knowledge-Roles	Signature-Element	Empty-Set
Assumptions	Formula	Empty-Set

Fig. 3. UPML description of a Reasoning Resource(PSM)



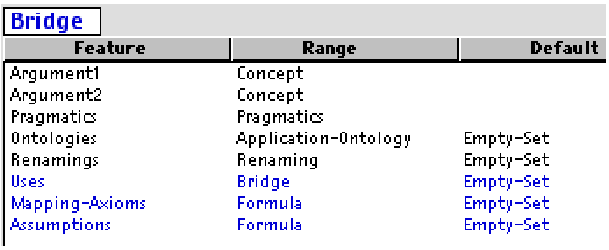
Domain-Model

Feature	Range	Default
Name	String	
Pragmatics	Pragmatics	
Ontologies	Upml-Ontology	Empty-Set
Uses	Domain-Model	Empty-Set
Properties	Formula	Empty-Set
Metaknowledge	Formula	Empty-Set
Knowledge	Signature-Element	Empty-Set

Fig. 4. UPML description of a Domain Model(PSM)

The fact of describing tasks, PSMs and domain models with different ontologies, makes task and PSM independent of the domain. This independence enables the reuse of task descriptions in different domains, the reuse of PSMs across different tasks and domain, and the reuse of domain knowledge for different tasks or PSMs.

UPML, as a software architecture, has a specific kind of connectors called *bridges*. A bridge models the relationship between two different components. The function of the bridges is to connect components with different ontologies, translating concepts among them. A bridge provides mapping axioms and assumptions about the components that it relates. There are three kinds of bridges: Task-PSM, Task-Domain Model and PSM-Domain Model (see Fig.1).



Bridge

Feature	Range	Default
Argument1	Concept	
Argument2	Concept	
Pragmatics	Pragmatics	
Ontologies	Application-Ontology	Empty-Set
Renamings	Renaming	Empty-Set
Uses	Bridge	Empty-Set
Mapping-Axioms	Formula	Empty-Set
Assumptions	Formula	Empty-Set

Fig. 5. UPML description of a Bridge

2.2 UPML and domain-independence

Once we have described the UPML components, let's see how to achieve domain-independence using UPML[10]. First of all we have to differentiate two concepts: *Library* and *Application*. A *Library* is a collection of UPML descriptions of tasks and PSMs. A Library is totally independent of the domain because tasks and PSMs are described in terms of their own ontologies, and not in terms of the domain ontology.

An *Application* is made of one or more libraries, a set of domain models, and the bridges linking the components in the library with some domain models. The mapping axioms of the bridges allow to translate the concepts of the domain model's ontology into concepts of the PSM and Task ontologies. This translation enables the PSM to work with the domain knowledge of any domain model whenever a bridge is defined.

This approach makes the library independent of the domain. This independence allows the library to be reusable, in the sense that the same library can be used to build different applications.

3 Overview of the I3 library

The *Intelligent Information Integration* (I3)[22]) library offers a collection of problem-solving methods to solve some of the usual tasks carried on by Information Agents (see for instance [4] [2] [12] [15] [17]. Information integration is a concept that originally means the integration of multiple databases and nowadays is being focused to the integration of multiple web sources with the use of ontologies [9].

Instead of adopting the information retrieval approach (IR), we adopt a vision more close to meta-search. IR focuses on improving retrieval methods, while meta-search focus on exploiting existing "information sources", where each resource posses a specific content accessible by a "local" retrieval engine. For this reason, the library and the WIM application focus on this process -and do not include components that can be found inside retrieval engines.

A second consideration is the paradigmatic distinction between the concept of "relevance" in classical IR and the more rich conceptualizations currently in use for intelligent information agents [3]. The canonical concept of relevance in IR is a test where the results for of a query by a retrieval engine are compared to a gold standard provided by a human expert that assesses false positives and false negatives. The problem of that approach is that "relevance" is independent of the purpose of the specific user in posing a query. The I3 Library includes some methods to elaborate and rank information according to an utility measure. This measures are given by external domain knowledge, therefore new utility measures can be included adding the appropriate domain knowledge. The way to do that is explained in §4.1.

I3 can be seen as an adaptation process with four subtasks: transformation of the user consultation , selection of information sources, information retrieval and integration of information from multiple sources.

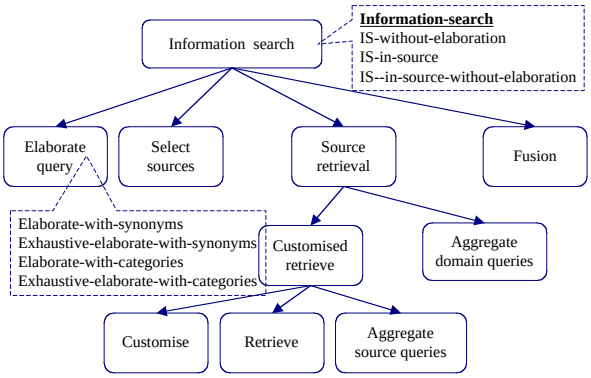


Fig. 6. Task decomposition of the IS-general-PSM

Adaption refers to the process of elaborating the user consultation to better fulfill his interest, as well as adapting the queries for the idiosyncratic syntactics of each information source. Once the retrieval is performed, the results from different queries are aggregated for each source, and finally the results for each source are aggregated again to obtain a unique result for the user consultation.

We have adopted a general approach for the overall process of information integration that is based on using *query weighting* and numerical *aggregation operators* [13]. Query weighting refers to the process of assigning weights to queries generated according to some domain knowledge, while numerical aggregation operators are the mechanism used to merge items coming from different queries and sources, and combining the different scores to obtain an unique score for each item retrieved. This mechanism allows to score documents retrieved from engines that originally do not give any score , and defining user-oriented utility measures simply by defining the appropriate knowledge categories (see §4.1).

3.1 Task decomposition

Tasks are decomposed into subtasks following a hierarchical task/subtask decomposition schema. The top-level task of our library is called *Information-Search*. There are four PSMs of the problem-decomposer class for solving the Information-search (IS) task: IS-general-PSM, IS-without-elaboration, IS-in-source and IS-in-source-without-elaboration. The more general one is IS-general-PSM that decomposes Information-Search in four tasks: elaborate-query, select-sources,source-retrieval and fusion.

3.2 Adaptation of queries

We adopt a very well known approach to queries as vectors of keywords instead of complex database query languages. This decision is justified because nowadays professional databases could often be accessed through the use of a web-based

search-engine, where queries are made of *keywords* belonging to a particular domain. We also include search *filters* as optional constrains allowing to restrict the search. A bibliographic ontology have been used to model the kind of filters allowed by professional bibliography search-engines, like *publication date*, *language* and so on. Let's see the both types of query adaptation: adaptation of queries with respect to the domain, and customization of queries for particular information sources.

Query elaboration: refers to the adaptation of queries for a particular user interest, within a particular domain. This task can be achieved using "semantic" domain knowledge, like synonyms, hyponyms or predefined knowledge categories.

Query customization: a query is customized for a particular information source by translating keywords and filters from the common ontology into the *search modes* and filters of a particular search engine. This task is different from the one performed by wrappers, where keyword-based queries are transformed in the particular syntax of the source, following the rules and particular features of each source at the syntactic level.

Selection of sources: it isn't a query elaboration method, but is needed when more than one source is available, so it is very related with the query adaptation process, and in particular, with the query customization task. The selection of sources could be done by asking the user or by using an automatic method, like Case-Based Reasoning (CBR). In the current version of WIM there are four information sources (search engines) available.

3.3 Aggregation of results

Aggregation is a kind of merging where the rankings assigned to the repeated apparitions of an item are combined using an aggregation operator to obtain a unique ranking that summarized the utility or relevance of each item for the user.

A numerical aggregation operator is necessary because of the nature of the query adaptation procedure, where queries are weighted with numerical values. Hence, the results for a query inherit the weight of the query. It means that even results non ranked by the retrieval engine can be ranked, by using the weight associated to the query for which they are a result. If the queries are weighted according to "utility" rather than relevance, then the results will also be ranked taking into account these utility criteria (See example 2 in §4.1).

Four numerical aggregation operators have been implemented as problem-solving methods in the the library: the *arithmetic-mean*, the *weighted-mean*, the *Ordered Weighting Average (OWA)* and the *Weighted OWA*[20].

4 The WIM application

Before to describe the WIM application we want to clarify the distinction between a library of reusable components and a configurable application. In IBROW,

a library is a collection of tasks and PSMs described in terms of its own ontologies. A library is reusable because it can be employed to build different applications. An application is a system composed of one or more libraries, a set of domain models, and the bridges linking components in the library with domain models and external resources. A configurable application refers to an application where there are different components to solve problems of the same class. In particular, a UPML based application is configurable when it has different PSMs to solve the same task, or there are different domain models available. A configurable application is suited to solve one (or more) class(es) of problems (Tasks) with some domain models, but this problems could present some differences, and the same problem could be solved with a different configuration, thus the application configuration is driven by the problem requirements, where the user preferences may be embodied.

The core of WIM is the I3 Library, but there is other components needed to build an application. In our approach, an application is built linking the reasoning components the library some domain knowledge and a collection of information sources.

4.1 Linking the library with the domain knowledge

Domain knowledge does not belong to the library; this is one of the most important features of our UPML, because the independence from the domain is considered a basic requirement to achieve reuse[16].

The domain chosen to build the WIM application is medicine, and in particular *Evidence-Based Medicine* (EBM). EBM proposes a medicine practice which calls for careful clinical judgment in evaluating the "best available evidence"[7]. The main task for the WIM application is looking for medical literature, and the utility criteria used to rank documents are those given by the EBM community to asses the quality of medical bibliography. Hence, we need also some bibliographic knowledge to describe queries and results for the queries, and some knowledge about the information sources accessed by the application. Let's see the different domain models and how are they used by problem-solving methods:

- A general *medical thesaurus* is used to select the keywords to pose a query and during the elaboration of the queries. We have chosen *MeSH*, a thesaurus that can be accessed through a web-based retrieval engine called the *MeSH Browser*. This domain model is used by the PSM query-elaboration-with-synonyms.
- An ontology about *bibliographic data*is used to describe the filters typically allowed by bibliographic search engines. This domain model is used to pose the queries and by the PSM query-customization.
- A collection of *source descriptions*, where the search modes and allowed filters of each source are described, including the translation sentences between the common bibliographic ontology and the different search-engines. This domain knowledge is used by the PSM query-customization.

– A collection of predefined categories describing the keywords and filters that are useful to rank documents according to the EBM criteria. This knowledge is used by the PSM *query-elaboration-with-categories*.

Example 1: The query weighting approach adopted for the query adaptation task has great advantages to rank documents using different criteria, not only classical IR's relevance. To introduce new utility criteria we have built a method to elaborate queries with predefined knowledge categories.

From the point of view of the EBM, it is very important to use the bibliographic references according to the quality of the evidence they rely on; hence, we have defined some categories expressing concepts about evidence quality in terms of a medical thesaurus and the defined ontology for bibliographic data, furthermore these concepts are weighted, allowing to weight the queries according to these "evidence quality" indicators.

A category is a collection of terms and filters associated to one topic, which are weighted according to the strength of that association. For example *Good-Evidence* is a category that defines some filters to get only papers based on a good evidence quality (below is shown a partial sample of this category):

```
(define (Category :id Good-Evidence)
  (name "Good Evidence Quality")
  (terms Empty-set)
  (filters
   (define (Filter-Weighting)
     (filter (define (Filter)
               (attribute "Publication Type")
                 (value "Meta-Analysis")))
     (weight 1))
   (define (Filter-Weighting)
     (filter (define (Filter)
               (attribute "Publication Type")
                 (value "Randomized Controlled Trial")))
     (weight 0.9)))
```

Given the query Q = (Levofloxacin, Pneumonia) and applying the PSM *Query-expansion-with-categories* with this category, we get the following set of queries:

```
Q1 = (Levofloxacin, Pneumonia, Publication Type = Meta-Analysis),
     weight = 1.0
Q2 = (Levofloxacin, Pneumonia, Publication Type = Randomized
     Controlled Trial), weight = 0.9
```

4.2 Linking the library with the information sources

Information sources are not domain models, they are accessed through search-engines, components of the system modelled as external problem-solving methods suitable for the the task retrieval. There is a domain model called source-descriptions containing the knowledge used by the tasks customization and retrieval.

Example 2: The query-customization PSM expands a query expressed in a source independent way in a collection of queries in terms of a particular information source, using the search modes and filters allowed by that source. This knowledge is described in the sources domain model. For example, this is our description of the HealthStar information source -when accessed through the retrieval engine called Internet Grateful Med (only a partial description is shown):

```
(define (Source :id Healthstar-Igm)
  (name "HealthStar-IGM")
  (weight 1)
  (search-attributes
       (define (Attribute-Weighting)
           (attribute "Title+Word")
           (weight 0.5)))
  (basic-attribute
       (define (Attribute-Weighting)
           (attribute "Subject")
           (weight 1)))
  (filter-attributes
       (define (Attribute-Translation)
           (domain-attribute "Author Name")
           (source-attribute "Author+Name"))
       (define (Attribute-Translation)
           (domain-attribute "Publication Type")
           (source-attribute "publication"))
       (define (Attribute-Translation)
           (domain-attribute "Begin Year")
           (source-attribute "begyear"))))
```

Given the query Q = (AIDS, Diagnosis, Begin Year = 1980), the resultant set of queries, after applying the Query-Customization method is given below:

- Q1 = (Subject = AIDS, Subject = Diagnosis, (begyear = 1980), Weight = 1)
- Q2 = (Title+Word = AIDS, Subject= Diagnosis, begyear = 1980), Weight = 0.5
- Q3 = (Subject =AIDS, Title+Word = Diagnosis, begyear = 1980), Weight = 0.5

New sources can be added to the application by including their descriptions according to the sources domain model, and building the appropriate wrappers (task-psm bridges) between the retrieve task and the retrieval engines for that sources.

5 The WIM multi-agent architecture

This section deals with the mapping between the knowledge components of WIM —within and without the library— and the WIM agent-based application.

Different multi-agent architectures and methodologies could be used to build agent-based applications. We propose a new methodology that combines the

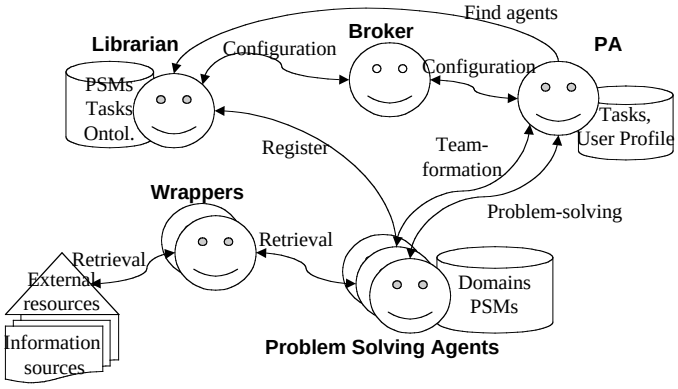


Fig. 7. WIM multi-agent architecture

UPML architecture with the approach of *Agent-Mediated Institutions* (AMIs, also called *e-institutions*[6].

This section will briefly describe the kind of agents used in the WIM application and the use of UPML as an Agent Capability Description Language (ACDL). Our goal when integrating UPML and AMIs is to develop an open agent architecture designed to build dynamically configurable applications. Furthermore, we claim our framework is well suited to design scalable applications which components are easy to reuse.

WIM is a configurable application; the components used to solve a problem are chosen for each problem-solving request. A configuration is a structure of UPML components suited to solve a problem given by the user or its personal assistant. WIM uses a configuration described in UPML to form a team of cooperative problem-solving agents suited to solve the problem that originates that configuration. The architecture presented here is based on the interaction between four different classes of agents: personal assistants, brokers, librarians and problem-solving agents. A broker is an agent able to obtain a configuration of the application for a given problem. Personal assistants mediate between users specifying problems in a non formal manner, and brokers, that work with problems described in UPML. Librarians acts like "yellow pages", providing the link between the knowledge level (UPMLconfigurations) and the operational level (agent teams). Figure 7 shows the five classes of agents in WIM and the main scenes where they interact:

1. Personal Assistant: An agent acting on behalf of a human user. This agent is responsible of mediating between the user request and the services offered by the application. The PA is able to specify problems in terms understood by the broker, that is UPML. Furthermore, the PA is able to use a configuration expressed in UPML terms to form and instruct a team of cooperating problem-solving agents able to solve the problem given by the user.

2. Broker: The role of the broker is to configure an application for a user (through its representing PA) requesting to solve a problem. Brokers in IBROW use the UPML specification of a problem to generate a *configuration:* a structure of UPML components matching the specification of the problem. A broker in our scenario do not carry on the operationalization of the application configuration, in agents terms: formation of a team of agents committed to solve a problem. The configuration obtained by the broker will be used later by the PA during the team formation.

3. Librarian: This agent holds the UPML descriptions of the reusable components: tasks, PSMs, domain models and ontologies. PSMs are the declarative representation of the agent capabilities. The library can be dynamically updated or extended with new functionalities. The key to do that is that new agents can enter WIM by registering their capabilities to the librarian in UPML. Therefore, the librarian agent is a dynamic repository of UPML described components, allowing other agents or humans to query about components in the library. Moreover, the librarian could be used as "yellow pages", just keeping and up-to-date register of the association between UPML components and the agents implementing them. Once a requester agent knows which component to call, it can query the librarian to know which agent is capable of executing that component.

4. Problem-Solving Agents (PSA): These agents are responsible of executing the components in the library. UPML is used to describe the capabilities of the agents. Problem-solving agents can enter or leave the system dynamically, just informing the librarian and registering or deregistering its capabilities. This is a simple way to make the librarian aware and up-to-date of the capabilities available in the system at any moment. There are two complementary visions of these agents. One approach is considering these agents as implementing the problem-solving methods of an existing UPML library. From this point of view, the development process starts specifying the library of UPML components, then some PSAs are built fulfilling the specification of tasks and methods in the library. Another approach is to consider existing agents, and then express its capabilities using UPML. Both approaches are compatible and complementary. We present a framework allowing both approximations to the design of agent-based (also knowledge-based) applications.

5. Wrappers: This is a very specific kind of agents responsible of solving the interoperability issues between agents in the system, and external —not agent— resources. In our architecture, wrappers implement a class of Task-PSM bridges where the tasks belong to the library, while the PSMs are external components. In WIM, these wrappers are used to connect the retrieval task to multiple retrieval-engines accessed through the *http* protocol and the *CGI* protocol.

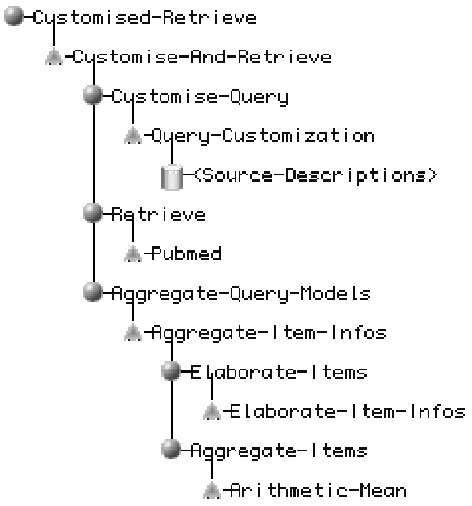


Fig. 8. WIM configuration example

6 Problem specification and brokering

The brokering process has the goal to find a configuration of UPML components (specified in a library managed by a Librarian agent) given an input specifying some problem to be solved. This input specifies not only the problem, but the kind of task it belongs to, and the properties that are to be met in the problem solving process. Thus the process starts when a user puts a new request to its Personal Assistant (or a scheduled task is automatically launched by the PA as derived from some initial user request). The user specifies the problem using its own concepts on a web interface, not UPML. In the context of the WIM medical application this means that the user use concepts of the medical domain to select the terms and criteria for the information gathering task she is interested in. Then, the PA transforms the user input into a description of *task requirements* in terms of the UPML specification and later the PA requests the broker for an adequate configuration with a message containing these requirements.

The Broker is an agent able to reason about UPML component descriptions. In particular, it has to obtain a configuration of the application that is able solve a problem specified in UPML. The brokering algorithms are briefly addressed later in this section. The output of the brokering process is a UPML *task configuration* where a) each task has an associated PSM that can achieve it an each Reasoning Resource has associated those Domain Models it needs, and b) the whole configuration complies to the input requirements. Figure 8 shows a configuration found by the broker agent.

The brokering related interactions include two steps:

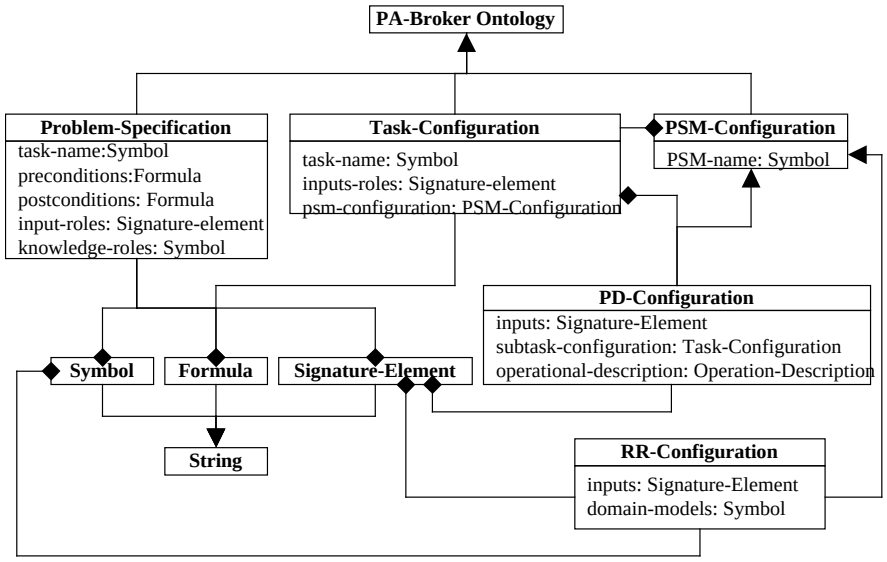


Fig. 9. PA-Broker ontology

1. The PA sends a request to the broker, asking for a configuration. The content of the message include the UPML specification of the problem.

2. If the broker accepts the request, it sends an "accept" message (if not it send a "reject" message). When agreed, the broker starts a brokering process over the UPML components described in the library. If a configuration is found, the broker sends an "inform" with a Configuration description in the content, if not, the broker must send a "failure" message to indicate that no configuration has been found.

6.1 The brokering process

The input for the brokering process is a *task requirements* specification composed of: a) the name of the task to be achieved, b) the pre-conditions that are established to hold, c) the postconditions that have to hold when the task is achieved, and d) the knowledge sources (domain models) that are available for achieving the task.

We will now summarize the broker agent strategy for constructing a configuration of UPML components such that the input task requirements is satisfied.

We view the the brokering process as a state-space search in the space of configurations. The process starts when the broker receives a *Problem specification* and generates an initial state, it continues generating new states until one of the new states is assessed as a final state: a state such that the configuration is complete (all task have an associated PSM and every Reasoning Resource has the Domain Models it needs) and valid (all pre- and post-conditions are satisfied).

State

Feature	Range	Default
Goals	Formula	Empty-Set
Assumptions	Formula	Empty-Set
Inputs	Signature-Element	Empty-Set
Met-Goals	Formula	Empty-Set
Met-Assumptions	Formula	Empty-Set
Met-Knowledge	Domain-Model	Empty-Set
Used-Knowledge	Used-Knowledge	Empty-Set
Open-Bindings	Tp-Binding	Empty-Set
Open-Knowledge	Signature-Element	Empty-Set
Top-Task	Task	
Tp-Bindings	Tp-Binding	Empty-Set
Current-Tp-Binding	Tp-Binding	

Fig. 10. Features and default values of a *State*

Later we will see how the new states are generated from a given state. The open bindings are tasks with no PSM bound to them. So, taken a task that has no PSM bound, the will retrieve form the Library those PSMs that "match" with the task, in the sense of component matching explained presently.

Component matching has two parts, signature matching and specification matching. Thus, for two components ($T \in \mathcal{T}$, $P \in \mathcal{P}$) their matching is:

$$T \preceq_M P = T \preceq_{SIG} P \wedge T \preceq_{SPEC} P$$

Signature matching requires that their input and output signatures match, and in our context this means that a problem solving method P has input and output signatures that are equal or that refine those of a task T, i.e. $T \preceq_{SIG} P = T_{in} \preceq P_{in} \wedge T_{out} \preceq P_{out}$.

Moreover, specification matching requires that a problem solving method P has weaker preconditions than a task T and stronger postconditions than T, thus: $T \preceq_{SPEC} P = P_{pre} \preceq T_{pre} \wedge T_{post} \preceq P_{post} \wedge P_{asm} \preceq T_{asm}$. When a task has assumptions ($T_{asm} \neq \emptyset$) only a reasoning resource can match since problem decomposers do not have assumptions. For all the PSMs that match with the task a new state is generated, updating the bindings and the *met-goals* and *met-assumptions*.

The search process used by the Broker agent can use any of the usual search techniques like depth-first or best-first. Currently, WIM uses a Case-based Reasoning (CBR) approach to guide the search process using past "cases" (configurations elaborated in the past by the same Broker). Let us now consider the search process as a twofold process where a) new states are expanded from an existing state, and b) open states (those not yet expanded) are ordered by a heuristic from more to less likely to yield a final solution.

The main issue to be represented in a state is the set of task-method bindings used in a partial configuration. That is, a state represents not the whole configuration but only the subset of tasks included in a configuration and the tasks that have some binding with a specific PSM at a given point in time of the brokering process. The second important issue for a state is determining which

pre- and post-conditions of the User Consult are satisfied by the components involved in a partial configuration—and which are not yet satisfied.

The generation of *successor* states from a given state is a follows. For an open state, check whether it is complete: if it is valid a solution is found, if not the state is an invalid complete configuration and the state is discarded. Otherwise successor states are generated and become open states. For an open task in the state, the broker retrieves PSM components from the Library such that satisfy they match the task (with matching criterion explained before). For each retrieved PSM, a new state (called successor state) is generated. This new state represents a new partial configuration where all relevant information is updated.

The broker follows this process of generating new successor states and checking if they are complete and valid until if finds a solution. The way in which these states are explored is the strategy of the search process; e.g. if states are stored in a stack and the broker pops the topmost state we have a depth-first strategy. The CBR strategy is a best-first strategy where all open states are assigned a "goodness" value based on the similarity of that state to the cases (the configurations constructed in the past by the broker and stored in a case base). The configurations of the case base are used just to improve the efficiency of the search, i.e. the likelihood of expanding less states to reach a solution state.

More information about the brokering strategy and the broker case-based reasoning approach can be found in [1]. The result of finding a solution state is the specification of a configuration of UPML components capable of achieving the task specified in the input. Recall that we propose a two-level approach to configuring an agent-based application the first level is conceptual and results in an abstract description of a configuration of components, as we have explained in this section. The next section explains the second level: the formation of a team of agents capable of realizing the configuration of components.

7 Team formation

This section describes the operationalization of the configured application as the process of forming a team of problem solving agents that is able to solve a problem, according to the UPML specification of the configuration obtained during the brokering process. This section is organized in five subsections: §7.1 describes the relation between some UPML notions and concepts of agent-mediated institutions, §7.2 deals with the agent management activities allowing the operationalization of a configuration: registering and deregistering agent capabilities, §7.3 addresses the issue of finding out agents with the required capabilities, while §7.4 explains how to select the agents to form a team, and finally, the process of instructing the agents in the team is described in §7.5

7.1 From UPML to MAS: UPML as an ACDL

The PA receives a problem described by the user and transforms it into a UPML specification. This specification is sent to the broker, the agent responsible for

finding a configuration of components suited to solve that problem while fulfilling its requirements, and this configuration is also expressed in UPML terms: tasks, problem-solving methods and domains. The PA should use the configuration given by the broker to form a team of problem-solving agents that will cooperate to solve the user problem. To find and select the appropriate agents for a UPML configuration, we will now establish a mapping between UPML concepts and agent notions. Our approach is to describe agent capabilities in UPML, thus using UPML as an agent capability description language (ACDL). We propose the following association between UPML components and agents:

- *Tasks* are associated to *agent roles*. Solving a task is equivalent to play a role in a team of problem solving agents willing to solve a problem. This approach is very useful when describing the operational description of a problem decomposer, as explained later in this section.

- *Problem-solving methods* are mapped to *agent capabilities*. This relation is very intuitive: both concepts refer to the basic "building blocks" or "reasoning" components of a system. Problem-solving methods are the reasoning components of an application in the UPML architecture, whereas capabilities are the elements used to describe the services offered by agents.

Since according to UPML there are two classes of problem-solving methods we will consider two kind of capabilities:

Reasoning resources are primitive capabilities, they cannot be further decomposed. This means that tasks requiring these activities are not solved by a team, they are solved by a single agent.

Problem decomposers are specifications of a task decomposition into subtasks, including the *operational description* of the decomposition process. In our framework, the operational description is described as a *scene:* the pattern of interactions between agent roles. Tasks decomposed by a problem decomposer can be solved by a team of agents, each one playing a different role (solving a different task).

We have decided to use just one scene to describe the operational description of a problem decomposer. The operational description in UPML describes the control scheme and internal data flow over the subtasks provided by the problem-decomposer. A scene is the concept of an e-institution that better fits this notion, this is the main reason to stablish a one-to-one mapping from an *operational description* to a *scene*. Since a scene is described in terms of roles, and we map tasks to roles, a scene for an operational description describes the pattern of interactions that occur between the role corresponding to the task being decomposed, and the roles associated to the subtasks in which it is decomposed.

Figure 11 shows an example of such a operational description. The task being decomposed is the Information-search task. The problem decomposer IS-without-source-selection-without-elaboration decomposes that task in two subtasks: Source-retrieval and Aggregation. The scene for the operational description of this problem decomposer includes three roles: Information-search, Source-retrieval and Aggregation.

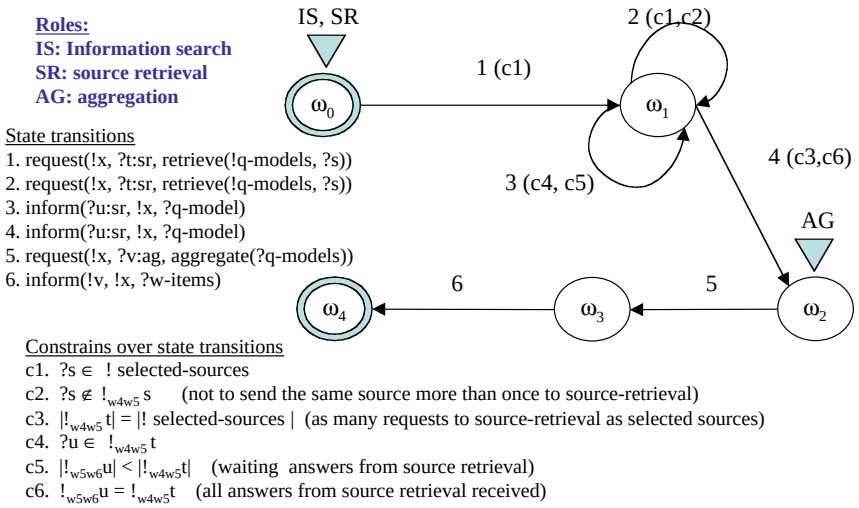


Fig. 11. Example of a scene using ISLANDER

The scene begins with two agents playing the roles Information-search (IS) and Source-retrieval (SR). First state is reached when IS sends a first request to SR with a set of query models and the source to be queried. The first state (w1) is hold while the IS agent sends the requests for all the sources (c3), until all the requests have been processed and the answers sent back to the requester (c6). An agent playing the Aggregation (AG) role enters the scene at the second state (w2), that changes to the third when the IS sends all the information to be aggregated to AG. The scene ends when AG sends the result of the aggregation process to IS.

7.2 Registering/Deregistering agent capabilities

This section describes the activities needed to let the Librarian be aware and up-to-date of the agents available in the system at any moment, and the capabilities they are equipped with.

A new concept called *capability* is defined in the *Team-ontology* (figure 12) allowing agents to register its capabilities. Following the mapping scheme presented in §7.1, a capability is described with a PSM. Therefore, a capability has a*name* that that is the name of a problem-solving method, while the *description* is an string encoding the UPML description of the PSM using one of the existing web-syntax formats for UPML: XML and RDF (this is indicated by the slot *encoding*.

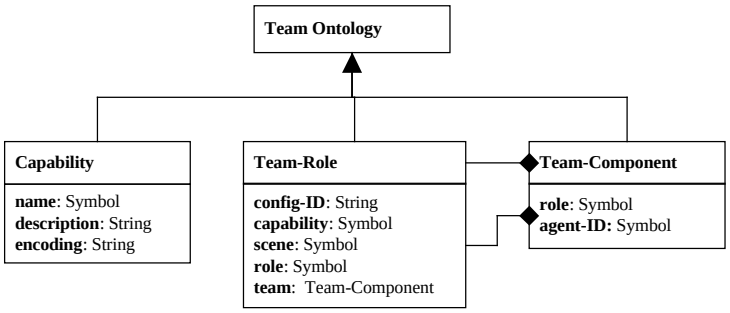


Fig. 12. Team ontology

The registering scene is very simple. When a PSA enters the agent platform, it sends a "register" message with the list of capabilities it is equipped with. The librarian builds a table with the bindings between capabilities and agents, in order to answer questions about which agents are equipped with a particular capability (PSM), as described in the next section.

Deregistering is easier than register because the librarian keeps the existing binding between capabilities and agents. When a PSA will leave the platform it sends a "deregister" message to the librarian, and the librarian erases all the bindings between that agent and the capabilities it has registered.

7.3 Finding capable agents

The goal of this scene is for the PA to find out which problem-solving agents are equipped with the capabilities required by a UPML configuration (obtained by the broker, as explained in §6). This scene involves the PA, that holds the configuration of the application, and the librarian, that keeps the bindings between capabilities and agents.

For each problem-solving method in the configuration the PA queries the librarian which agents in the platform have registered that PSM. A query to the librarian is a "request" message with the command "Find agents" and the name of the PSM required. The Librarian acts like a yellow pages service in this scene, matching the PSM required by the PA to the capabilities registered and stored in the agent-capability binding table, and answers the PA with the list of all the capable agents.

7.4 Selecting agents for the team

This scene describes the process of selecting one and only one agent for each capability and each task (notice that the same capability could be required for different subtasks of the problem) required to operationalize a configuration.

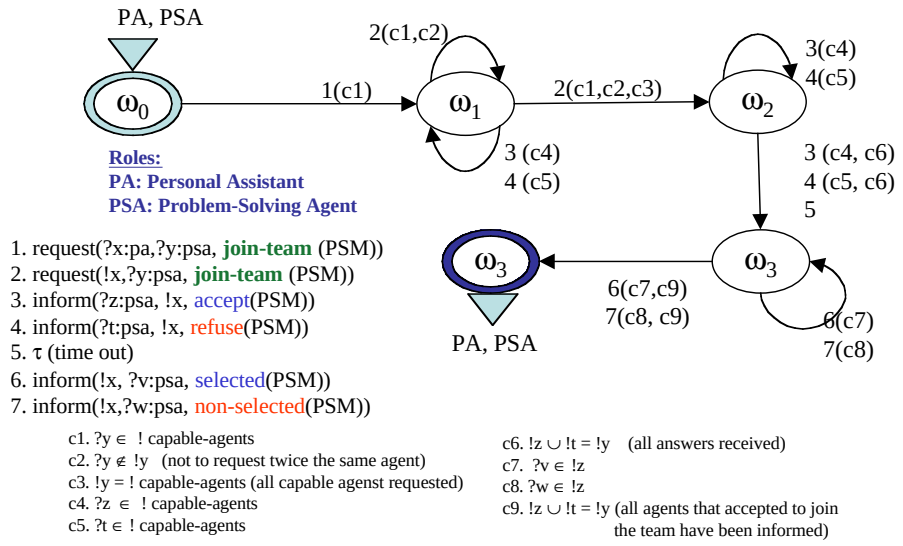


Fig. 13. Selecting team members

This interaction is carried on in a similar way to the *contract-net* protocol. For each pair of task and problem-solving method in the configuration, the PA requests all the capable agents (the process of finding capable agents is described in above) if they accept to join a team of agents to solve a part of the problem using the required capability.

Problem-solving agents can decide autonomously on accepting or refusing that request (see figure 13). The PA waits until all the available agents have answered or a time out is reached, then, the PA decide between alternative agents for the same task which one to select as a member of the team.

Different algorithms and strategies can be used to select team members. A simple, but meaningful one, is to select the agent that answered first to the join-team request, as this is an indicator of its availability and a low cost of communicating with that agent. But the framework presented here is suitable for a wide range of selection strategies that can be embodied in the PA.

After selecting the team members, both selected and non-selected agents are informed of that condition with an "accept" or "refuse" message. The selected agents are `committed`[1] to form part of the team. The process of instructing them to form the team is addressed below, in the next section.

7.5 Instructing agents to form a team

This scene describes how to instruct the agents selected to form a team about the task to be carried on as part of the team (the role to play), the capability

[1] Commitments are specified as normative rules of the electronic institution, see ?

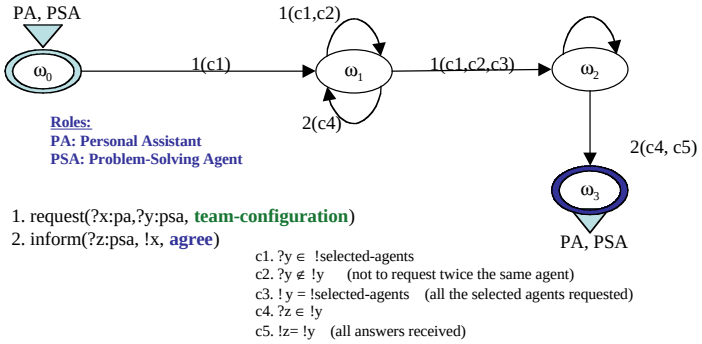


Fig. 14. Instructing selected team members

required for that task (a PSM), and the other members of the team to cooperate with.

Since selected team members have been committed to form part of the team, they are waiting for the instructions to participate in that team. For each task in the configuration, the PA sends an "inform" message to the selected agent for that task, including all the necessary information encapsulated by the concept *Team-role*, that is described in the *Team-ontology* (see figure 12).

A *Team-role* consist on the following elements: A unique identifier of the team (as there is a team for each configuration of the application, we use the identifier of the configuration for the team), the name of the scene, the name of the required capability (a PSM) and the role to play in the scene (the Task to be solved). If the required capability corresponds to a problem decomposer, then the *team-role* fills also the slot *team*: a set of *team-component*, which specifies the agents playing the other roles in the scene.

A *team-component* (see figure 12) is a pair consisting in a role name R and an agent identifier A, meaning that the role R in the scene describing the operational description (corresponding to a subtask in the problem decomposer) will be played by agent A. The agent receiving a *Team-role* specification does not need to know which capability is required of other team components, because this information is sent to that agents separately, with another team-role instance.

8 Conclusions

We have presented a framework to develop configurable information agents by integrating a knowledge modelling architecture (UPML) and agent-mediated institutions. The knowledge modeling architecture aims at maintaining the independence between the domain (domain knowledge and ontology) and the information agents (with separate ontologies), while agent-mediated institutions give a formalism for representing interaction protocols in multi-agent systems.

We have described WIM, a configurable agent-based application developed upon the proposed framework. Agents in WIM annotate their capabilities with a domain-independent ontology—the same by which PSMs are described. The WIM application is built by linking the "knowledge requirements" specified in UPML to specific domain resources using bridges —examples shown are the MeSH thesaurus and the EBM models.

Describing agent capabilities is an active field of research, as this is an important issue to achieve interoperability in open agent architectures. The most widely adopted approach to solve the interoperability problem is the combined use of middle agents [5] and agent capabilities description languages (ACDL). An example of an ACDL is LARKS, and ACDL that is being included in the RETSINA multi-agent infrastructure [19] to be used within the matchmaking process. [2]. Our framework shares some ideas and characteristics of other ACDLs, but there are some major differences. We will expose here the main contributions of our work to this field:

(i) Our framework is based on a well defined architecture for describing software components from a knowledge modelling perspective (UPML).

(ii) UPML allows to describe problem-solving methods and tasks, but also domains, keeping the independency between them. This is achieved by using bridges to connect components described with its own ontologies.

(iii) We have described a general framework for developing reusable problem-solving libraries that allows to build configurable applications. Furthermore, an application built following this framework can be dynamically extended or modified with new competencies, just registering new agent capabilities to the librarian.

(iv) Other particularity of UPML is that it is neutral about the content language, thus giving developers a great flexibility. Different content languages will support different "matching" techniques between requirements (a problem specification) and services provided (agent capabilities)

(v) An aspect of special relevance is the notion of brokering. We have presented a particular brokering algorithm based on CBR over attributes described with UPML and feature-terms as the content language. Our framework allows to use different brokers and personal assistants in the same platform at the same time. The only requirement is that they should follow the scene called Problem specification and brokering, that is explained in section §6.

(vi) Moreover, we have distinguished between library and application. A library is a repository of UPML descriptions of agent capabilities as problem-solving methods. A new application is built selecting a collection of agents and developing the bridges required to link library-registered agents with domain resources and domain ontologies. A configurable application is a collection of components that are selected (brokering) and connected (team formation) to solve a particular problem.

[2] `http://www.cs.cmu.edu/ softagents/interop.html`

(vii) Finally, ISLANDER, the formalism used to describe agent communication scenes provides our framework with a graphic editor and computational support for verifying agents follow specification.

References

1. Chema Abasolo, Josep-Lluis Arcos, Eva Armengol, Ramon Lopez de Mantaras, and Enric Plaza. Component matching. Technical report, IIIA-CSIC, 2001. IBROW3 Deliverable 3.1.
2. Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.
3. Jaime Carbonell. ISMIS 2000 invited talk, 2000.
4. Sudarshan Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS project: Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan*, pages 7–18, Tokyo, Japan, 1994.
5. K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proc. of the 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan, 1997.
6. M. Esteva, J. A. Rodriguez, C. Sierra, P. Garcia, and J. L. Arcos. On the formal specifications of electronic institutions. In *Agent-mediated Electronic commerce. The European AgentLink Perspective, LNAI 1991:126-147*, 2001.
7. A.R. Feinstein and R.I. Horwitz. Problems in the evidence of evidence-based medicine. *American Journal of Medicine*, 103:529–535, 1997.
8. D. Fensel, V. Benjamins, S. Decker, M. Gaspari, R. Groenboom, W. Grosso, M. Musen, E. Motta, E. Plaza, G. Schreiber, S. Studer, and B. Wielinga. The component model of UPML in a nutshell. In *Proc. First Working IFIP Conference on Software Architecture WICSA*, San Antonio, Texas, 1999.
9. D. Fensel, C.A. Knoblock, N. Kushmerick, and M.C. Rousset. Workshop on intelligent information integration (iii99), 1999.
10. Dieter Fensel, V. Richard Benjamins, Enrico Motta, and Bob J. Wielinga. UPML: A framework for knowledge system reuse. In *IJCAI*, pages 16–23, 1999.
11. D. Garland and D. Perry. Special issue on software architectures. *EEE Transactions on Software Engineering*, 1995.
12. Michael R. Genesereth, Arthur M. Keller, and Oliver M. Duschka. Infomaster: an information integration system. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 539–542, Tucson, USA, 1997.
13. Mario Gomez and Chema Abasolo. Improving meta-search by using query-weighting and numerical aggregation operators. In *Proc. 9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, 2002.
14. Matthias Klusch, editor. *Intelligent Information Agents*. Springer, 1999.
15. Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Query-answering algorithms for information agents. In *Proceedings of the AAAI*, Portlan, USA, 1996.
16. E. Motta. *Reusable components for Knowledge Modelling*, volume 53 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 1999.
17. Marian H. Nodine, William Bohrer, and Anne H. H. Ngu. Semantic brokering over dynamic heterogeneous data sources in infosleuth. In *ICDE*, pages 358–365, 1999.

18. International Foundation on Cooperative Information Systems. Second international conference on cooperative information systems, 1994.
19. K. Sycara, M. Paolucci, M. Van Velsen, , and J.A. Giampapa. The retsina mas infrastructure. Technical report, Robotics Institute, Carnegie Mellon University, 2001. tech. report CMU-RI-TR-01-05.
20. Vicen Torra. Weighted owa operators for synthesis of information. In *Proc. 5th IEEE Inter. Conference on Fuzzy Systems*, pages 966–971, New Orleans, USA, 1996.
21. Gio Wiederhold. Mediators in the architecture of future information systems. *Computer Magazine of the Computer Group News of the IEEE Computer Group Society*, 1992.
22. Gio Wiederhold. Intelligent integration of information. In *Proceedings of ACM SIGMOD Conference on Management of Data*, pages 434–437, Washington DC, USA, 1993.
23. G. De Michelis E. Dubois M. Jarke F. Matthes J. Mylopoulos K. Pohl J. Schmidt C. Woo and E. Yu. Cooperative information systems: A manifesto. In *Proc. 4th Intl. Conf. on Cooperative Information Systems*, Brussels, Belgium, 1996.