

# Information resource management in an agent-based virtual organization—initial implementation

Maria Ganzha<sup>1,2</sup>, Adam Omelczuk<sup>4</sup>, Marcin Paprzycki<sup>1,3</sup>, and Mateusz Wypysiak<sup>4</sup>

<sup>1</sup> Systems Research Institute Polish Academy of Sciences,  
Warsaw, Poland

Maria.Ganzha, Marcin.Paprzycki@ibspan.waw.pl

<sup>2</sup> University of Gdansk, Gdansk, Poland

<sup>3</sup> Warsaw Management Academy, Warsaw, Poland

<sup>4</sup> Warsaw University of Technology, Poland  
omelczuka,wypysiakm@student.mini.pw.edu.pl

**Abstract.** In this paper we describe an early stage prototype of a system for information resource management in an agent-based virtual organization. We focus our work on support for human resource adaptability (e.g. knowledge, skills) as a tool for assisting project managers. In the proposed approach, a virtual organization is functionalized in terms of roles played by agents, while organization structure and information flow are represented in terms of agent-agent and agent-human interactions. Finally, all resources (e.g. workers, skills, training resources, etc.) are ontologically demarcated, and information is semantically processed. Discussion of organizational needs is followed by the outline of the system. Finally, basic capabilities of the implemented prototype are illustrated.

**Keywords:** agent systems, ontologies, virtual organization, resource management, Information Resources, e-learning, adaptability.

## 1. Introduction

Development of new ICT technologies often influences provisioning of information needed to support workers in an organization. Among recent promising technologies, one can identify software agents [22], as well as ontologies and semantic data processing [13]. In this work, we consider how an organization can combine them to provide workers with the needed *Information Resources (IR)*<sup>5</sup>. In the proposed approach a real-world organization is modeled in terms of identified roles played by various entities within it. These roles are then represented by software agents. Furthermore, the organizational hierarchy is mapped into (hierarchical) relationships between agents, while the information

---

<sup>5</sup> Let us remark that the abbreviation *IR* stands for *Information Resource*, not for *Information Retrieval*, as it is often used elsewhere

flow within an organization is conceptualized in terms of agent-agent (and, possibly, agent-human) message exchanges. In this way we follow, conceptually, the main tenets of the Gaia methodology for agent system development [36]. Furthermore, in the proposed approach, all resources are ontologically demarcated (e.g. workers, articles, courses, books, projects, tasks, etc.), and information is semantically processed. Although in this paper, we focus only on issues involved in delivery of a class of *Information Resources*, presented results can be naturally extended to involve other resources, by modifying the ontology and the communication patterns. Note that, first, the proposed approach is in line with the vision for development of systems combining agents and semantic data processing, outlined in the seminal paper by J. Hendler [20]. Second, it follows the guidelines for agent system development proposed in [27]. There, a well presented argument supports the claim that to achieve progress in use of agent system in the real world, various approaches have to be tried and practically experimented with, to be able to gain the necessary experience.

Let us start with an overview of organizational adaptability that provides the use cases for the system prototype.

### 1.1. Adaptability in an organization—brief overview

Let us consider a *Virtual Organization* (VO; [5, 9, 10, 14, 19, 34]), where workers need to access various *Information Resources* to complete their tasks/projects. Obviously, access to resources should be (a) *adaptive*, matching the specific projects that the workers are involved in at a given moment, and changing with the project (as it evolves), and (b) *personalized*, different workers, depending on their knowledge, experience, and assigned task(s), require access to different resources (and their needs also change over time).

For instance, assume that two workers (W1 and W2) are a part of a team designing and implementing a knowledge management portal, combined with a dedicated mobile application. Here, worker W1 who is designing and implementing the back-end of the system (which is based on the Oracle database) needs different resources than her colleague (W2) preparing a dedicated front-end application geared for mobile operating systems (iOS, Android, and Windows Phone). However, because the new version of the Windows Phone system differs considerably from its previous versions, worker W2 needs additional resources to complete her job (e.g. she may need training modules related to working with the new Windows Phone API). Separately, if a graphics / Web interface specialist (worker W3) is added to the team, to create an application layout for the mobile devices (with much smaller screens), he may also need extra *IRs* to complete his task (e.g. he may need to extend his knowledge about most common screen resolution on mobile devices, including some technical details, like screen contrasts, or just an information about layout usability for the small screens).

In this example, the following situations of interest can be identified:

- Decision, made by the management of the organization, about acceptance / rejection of a project is based on availability of workers with given skills, whose schedule is such that they can participate in the project.
- Some workers, who have been selected to participate in the project, need extra training (access to *IR*'s) to be ready when the project starts (worker W2).
- During the project, some workers need extra training (access to *IR*'s) to improve their skills (worker W3).

Note that, in this work, we use terms *training* and *Information Resources* rather broadly (and somewhat informally). Specifically, by *training* we mean access to various types of *IR*'s that can improve knowledge of the worker, while the term *Information Resource* includes both e-learning modules and publications.

The above listed situations represent cases of, broadly understood, *adaptability* taking place within a system (for more details, see [8, 18]). They can be divided into two orthogonal groups. First, we can observe *institutional* and *individual* adaptability. Here, if workers with appropriate skills (and/or schedule) cannot be found, they have to be hired and/or trained, representing the general case of *institutional adaptability*. Obviously, an organization may select to not to adapt, and reject a possible project. At the same time, worker who is being trained, or provided with an *IR* (e.g. a book) learns, and this represents the case of *individual adaptability* (i.e. her/his skills change). Second, we can distinguish *reactive* and *proactive* adaptability. Here, the *reactive adaptability* means that the *IR* provisioning occurs in response to a specific situation (e.g. during a project a need for extra resource(s) is recognized; as in the case of worker W3, above). At the same time, if we assume that training of worker W2 takes place before the project starts (the organization recognizes the need to train its developers in the use of the new API of the operating system that they will work with), this would represent the case of *proactive adaptability*.

Let us observe that, from the point of view of functionalities planned for the prototype, the proactive and the reactive adaptability resemble each other very closely. As soon as it is established that there exists a gap between the *needed* and the *existing* skills, the system should proceed in exactly the same way. First, the gap has to be assessed, on a case-by-case basis. Second, appropriate *IR*s (assumed to help closing this gap) should be found and delivered to the worker. Therefore, at this stage, we have concentrated our attention on the reactive scenario (leaving the proactive one for the future). Let us therefore describe, in some details, the two use cases that follow from the above general example, and provide foundation for the development (and testing) of the system prototype.

## 1.2. Use case scenarios

**Scenario 1: Project Acceptance Decision.** Let us assume that a new project is considered by an organization. Let us also assume that, after an initial analysis, this project is specified through: (a) set of required tasks, (b) set of skills

required by workers to complete these tasks, and (c) time constraints on each task. To be able to decide whether to accept the project, these requirements are matched against human resources available in the organization. Here, the following constraints are considered: (i) “schedule-availability” of workers within the organization, (ii) their individual skills, and (iii) possibility to hire extra workers (if needed).

After the initial analysis, project specification is send to the selected *Project Manager (PM)* who will assess whether it is possible to complete it. For each task, the *PM* seeks *Workers* with appropriate skills and schedule. In our current approach, such assessment is based on *PM* ↔ *Worker* interactions. Here, *Workers* assess their own skills and schedule against the proposed task(s) and send responses to the *PM*. In the case when the *Worker* would need additional knowledge to accomplish a given task, (s)he looks for suitable *IR(s)*. Depending on their availability and time needed to acquire knowledge, a positive or a negative response is send to the *PM*.

Upon receiving responses, the *PM* checks if all tasks can be covered by the available *Workers*. When some tasks cannot be assigned to the *Workers* available in the organization, the *PM* asks the *Human Resource Manager (HRM)* to try to hire employees with needed skills and availability. If such *Worker(s)* can be hired, the *HRM* “sends them” to the *PM*. When all tasks within the project have assigned *Workers*, the *PM* accepts the project. Next, she confirms assigned tasks to the selected *Workers*. In the case when one or more tasks are left without assigned *Worker*, the *PM* is forced to reject the project.

**Scenario 2: Employee Support.** Let us now assume that a team is working on a project. At some moment during that time, one of *Workers* (e.g. W3 in the example above) faces a problem that she never encountered before. Obviously, she starts to look for information to solve this problem. She can look for someone that was dealing with a similar problem (by asking fellow *Workers*—W1 and W2, posting request on mailing list/forum, or search in the Internet). There, she may find solutions that are too complex for her, because they assumed that the person that will use them possessed higher level of knowledge. Other resources can, in turn, cover the issue on such an introductory level that they will be a waste of her time (they will cover topics that she is already familiar with). Thus, the need for personalized support for each *Worker* arises. The system should be able to provide *Workers* with needed *IR(s)* as a reaction to their needs.

Note that, the proactive scenario is very similar to the one described above. The only difference is in the fact that the decision to expand knowledge arises (proactively) on the basis of predicted future needs. Thus the differentiation between them belongs to a different (meta) level of the worker support system.

### 1.3. Related work

Information management with utilization of agent technology is not a new idea. There are many published works related to this topic (see, for instance, [8–10,

14, 17, 18, 31]), but most of them focus only on theoretical discussion how such systems can benefit from agent technology, and provide the initial design of such system (possibly including some suggestions concerning its implementation). Specifically, they discuss how organization can be transformed to the form that actually can be used in agent systems, with ontologically demarcated *Information Resources*. They consider how to map the needed resources to the missing skills. For instance, in [30] authors' discuss various methods of ontological matchmaking (understood as matching instances within a single ontology) and propose their own approach. However, this novel approach has not been tested in a realistic application.

In [25] authors present a platform that was designed during the EU project "Platform for Organizationally Mobile Public Employees" (project *Pellucid*). Main goal of this project was to create adaptable platform for assisting organizationally mobile employees. Such platform was to improve organization effectiveness and efficiency by formalizing, recording, and storing information about experience and knowledge, and allowing easy access to such data in a mobile environment. The proposed system was to combine software agents and semantic data processing. Unfortunately, as in case of so many other similar projects, after the end of EU-funding the promising research has stopped.

Another approach worthy mentioning was presented in [24]. In his thesis, author discusses use of ontology based knowledge representation in multi-agent systems. Author focuses on the issue how to create stronger connection between those two with utilization of the semantic web. Again, the research does not seem to be continued after completion of the thesis.

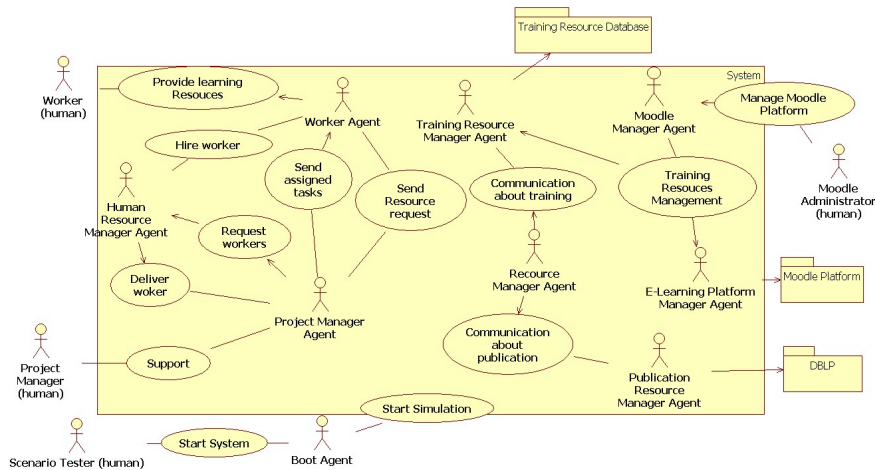
In Poland, between 2005 and 2007, there existed (sponsored by the Polish government) project called WKUP (eng. Virtual Public Service Consultant; [6]). The goal of this project was to create interactive, personalized platform to support citizens in finding assistance how to deal with tasks required by various administrative processes. The support was to be based on ontological representation of knowledge about the Polish legal system. Specifically, the WKUP system was to be able to understand questions in natural language, use ontologies describing domain of public administration, and be able to cooperate with a semantic registry of public services based on the UDDI (Universal Description Discovery and Integration) specification. Even though the initial prototype has been developed around 2007, the project has been later abandoned [12].

Overall, while many publications consider the idea of combining software agents and semantic data processing to support workers in a *Virtual Organization*, in most cases only initial stages of system design have been completed, and promising prototypes have been abandoned (e.g. due to the lack of continued funding).

## 2. System overview

As stated above, the aim of our work was to develop a prototype system supporting autonomous *IR* provisioning in a *Virtual Organization*. This system should

automatically detect *Worker* needs, and attempt at provisioning targeted *IRs*. One of key assumptions is that the system will be agent-based, and will use semantic data processing. Following the ideas discussed in [31, 17], we have decided that each *Worker* in the organization will be supported by her/his personal *Worker Agent*, which will represent her/his interests in the system. The *Worker Agent* is a form of a *Personal Agent*, idea of which was proposed for the first time by P. Maes in [26]. Furthermore, a number of auxiliary, agents will be added to the system. These agents will facilitate roles that either can be fulfilled autonomously (without need for human intervention), or will emulate roles that, in an actual organization, are typically fulfilled by humans supported by their *Worker Agents*. Based on these assumptions, in the context of the above presented example, and the two use case scenarios (that define the scope of our initial work), in Fig. 1 we present the AML use case of the system (for more details about AML, see [11]).



**Fig. 1.** AML Use case—functionality of the system

Let us now describe most important features of the proposed system in some detail.

### 2.1. Agents in the system

Let us start with a brief discussion of agents and their roles (see, Fig. 1). For the initial system prototype, supporting use cases described in section 1.2, we have designed and implemented the following agents:

- *Personal Agent (PA)*; an artificial meta-name for a group of agents controlled by human user. An instance of such agent is provided to each *Worker*

in the VO. The PA supports: *Workers* in a form of *Worker Agent (WA)*; *Project Managers*, as the *Project Manager Agents*; and *E-learning System Administrators* as the *Moodle Administrator Agents*.

- *Resource Agent (RA)*; auxiliary autonomous agents that can play the following roles in the system:
  - *Human Resource Manager Agent (HMRA)*, which emulates the situation in which a human HRM would be supported by an extended version of the WA (prepared to support the HRM activities); in our system, due to its limited scope, this role is emulated (completed autonomously).
  - *Publication Resources Manager Agent PRMA*, which is interfacing with the Digital Bibliography and Library Project (DBLP; [32]),
  - *Training Resource Manager Agent (TRMA)*, which is responsible for providing the system-side interface to the e-learning platform,
  - *E-learning Platform Manager Agent (EPMA)*, which provides the platform-side interface to the e-learning platform (here, it is assumed that multiple e-learning platforms are located “somewhere within the Internet” and thus both sides—system and platform—have to be represented by separate agents – rather than having a single platform-side agent that would be contacted directly from the system).

Let us summarize agents and their roles in the form of the AML Role Diagram, in Fig. 2.

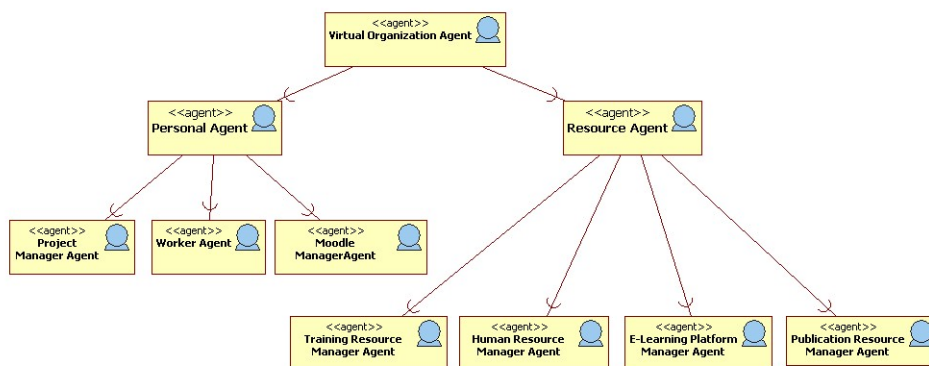


Fig. 2. Agent's roles hierarchy

## 2.2. Information Resources

Let us now consider the *Information Resources* that are needed to complete the use case scenarios described in section 1.2. In the current system prototype we have introduced two types of resources: (i) *training resources*, and

(ii) *publication resources*. The *training resources* comprise online courses supplied for the *Worker* training. The *publication resources* are publications that provide the needed information.

As stated above, in addition to software agents, we will use semantic data processing. Therefore, both types of *IRs* have been demarcated using a simplistic ontology, which was designed to support selected functionalities of the prototype. Let us now look in some detail into the use of ontology in our system.

### 2.3. Ontology in the system

When designing our prototype we were faced with decisions concerning development and use of ontologies. For instance, we could have used the existing ontologies, e.g. the Dublin Core [1] for demarcation of publication resources, and adapt the *VO ontology* found in [33, 29, 23] to describe the organization. However, first, this would still leave us with the need to develop an ontology of training resources. Second, combining the Dublin Core, the VO ontology and the training resource ontology is a research task in its own right, likely resulting in a rather large and complex combined ontology. Third, we were not able to locate a mainstream e-learning platform that would be ontology enabled. Overall, focusing on ontologies, would postpone implementation of the system, which would contradict our main goal—development and experimentation with a working prototype (see, also [27]). Therefore, we have decided to develop our own simplistic ontology and use it in the prototype. In Fig. 3 we present the overall structure of this ontology.

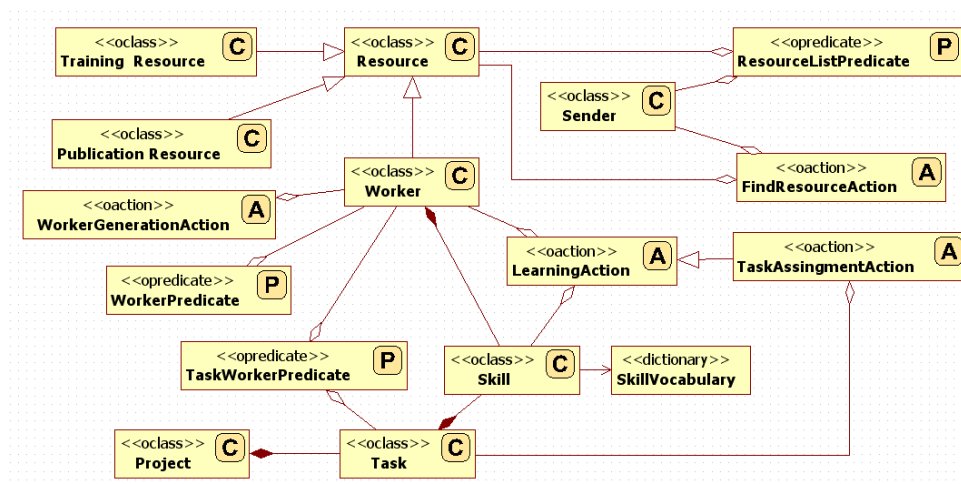


Fig. 3. Ontology design (C - Concept, P - Predicate, A - Action)



Here, the central concept is the *Resource*. For the prototype, we have decided to focus on the field of information technology, as conceptualized by the Association for Computing Machinery Computing Classification System [7]. The *Skill* concept consists of two parts. The name of the discipline, and a value from the interval (0, 100), which indicates the skill-level. Note that the *Skill* class is the only one used in both the *Resource* and the *Worker* modules (see, section 3).

The *Resource* concept describes the *Information Resources* as well as *Human Resource*. It is used mostly in communication between the *Worker Agents* and the *Resource Module*. It contains the main *Resource* class, the implementation of the *Concept* interface, and three child classes: the *Training Resource*, the *Publication Resource* and the *Worker*. The *Worker* class stores the information about *Worker's* skills (implemented as the *Skill concept*). The *Skill* concept establishes connection between the *Worker* and the *Task* concepts, as well as between the *Worker* concept and the *LearningAction* action.

In case of concepts *TrainingInformation* and *PublicationResource* these classes store specific information about those *IRs*, like author, title and publisher, for the *publication resources*, or names, localization and cost for participating in the course, for the *Training Resources*. In this structure, thanks to the inheritance, subclasses could use the *Resource* class in all messages for all *IRs*. Note that the only place where there is a need to distinguish between the different *IRs* is when the *Worker Agent* displays found resources to the *Worker*. In other words, within the system both forms of *Information Resources* are processed exactly in the same way, only in the user interface they are distinguished.

There exists also an artificial class called *Sender*. It was created because, despite the fact that the FIPA message envelope contains information about the sender of the message, when the request for resources from the *Worker Agent* is forwarded (by the *Resource Manager Agent*) to the *Training Resource Manager Agent* (or to the *Publication Resource Manager Agent*), the sender of the request has to be preserved in order to be able to return the found resources to the proper *Worker Agent*. Such requests are implemented using the *AgentAction* class *FindResourceAction*. This class contains information about the sender, and a list of topics that should be covered by the returned resources. This list is stored as a list of *Resource objects*, with the skill field assigned. As an answer to the request, represented by the *FindResourcesAction*, the *Resource Module* sends the *ResourceListPredicate* class, which implements the *Predicate* interface. This class is used as a facade for the list of found *Resources*. This is done, because the JADE framework does not allow sending the *Concept* implementations directly. They have to be wrapped in an *Action* or a *Predicate*.

The second part of the ontology is used to describe the *Worker*. This module focuses on the project, by describing *Workers*, tasks within project, skills possessed by the *Workers* and required for tasks, the project itself and actions that can be performed. Those objects are used mostly in communication between the *Worker Agent* and the *Project Manager Agent*. The two main, separated, concepts are the *Project* and the *Worker* classes. The first describes the actual project, in terms of name, list of tasks (understood as a "to do" list;

e.g. creating a database, or designing application layout for a mobile device), available time (here, we have decided to use the standard man-hours), and the information if this project is to be considered to be within the scope of a proactive or a reactive scenario (i.e. if it is a project for the future, or if it is a project that the organization has committed to). The *Worker* class stores basic information about the employee, like: first name, last name, list of skills of the represented person. This section contains also two implemented *Predicates*, called the *WorkerPrediacte* and the *TaskWorkerPredicate*. The first predicate is used in the transport of the *Worker* concept (for the same reasons as mentioned earlier for the *ResourceListPredicate*). The *TaskWorkerPredicate* has a more complex job to do. It contains the *Task* and the *Worker* concepts, to be used in the communication process, and parameters defining how well skills of a given *Worker* match skills required for completing a specific *Task*. Here, we use simple metrics, based on the assessment of how many skills were on the exactly right level, how many above, and how many below the required skill level (for a given task). Additionally, the system stores the number of skill levels that are above the requirements, and number of those skills that did not fulfill the requirements.

To send requests, three implementations of the *Action* interface are used. The *WorkerGenerationAction* is used by the *Project Manager Agent*, when it decides that a new employee has to be hired. It contains the *Task* concept, because such decision is made when some of the tasks, in the specific project, are not assigned (and cannot be assigned to the currently employed workers). Thus, a new *Worker* should be hired to complete this task. The remaining two actions are connected by the base-child class relation. The base class, called the *LearningAction*, stores the list of *Skills* levels of which the recipient (*Worker Agent*) must improve, and/or list of brand new *Skill(s)* to be acquired. The child class is called *TaskAssignmentAction*, and stores the *Task* that is assigned to this *Worker Agent*. We decided to create such relation, because, sometimes, when the *Project Manager Agent* assigns a *Task* to a particular *Worker Agent*, a given *Worker* must learn something new or increase knowledge in one of already possessed *Skills*, while at other times learning activities are required without assigning any task.

### 3. Implementation details

As a result of the technical requirements analysis, completed in [28], we have selected the following technologies to implement the initial system prototype.

- All agents within the system are written in Java using the JADE framework [2], with addition of the log4j framework [3], used for the error logging purpose. During the actual implementation process, we used Java version 1.6.0.22. The runtime environment was provided by the standard Oracle Java Virtual Machine.
- For storing data used to describe the *training resources*, the MySQL database engine was used.

- As the e-learning platform we have selected the Moodle Platform [4]. It is an open source Learning Management System. While Moodle is written in PHP we decided not to connect with it through its web interface, but created a dedicated agent connected directly to the Moodle database. Note that this design decision follows the basic principles of agent system design, where software agents provide the basic abstraction for design modularity and component encapsulation (see, also [21]).
- The MySQL server and the Apache server, required to host the Moodle platform, were provided by the XAMPP version 1.7.3.
- Information retrieved from the *Training Resource Database* and the DBLP, for further use by different agents, was described using ontology created using the Ontology Bean convention (see, section 2.3 for more details).

Lest us make a comment concerning the last decision. One of the weaknesses of the JADE agent platform is its limited ability to deal with ontologies. As a matter of fact, at the time of working on our prototype the only reasonable way to combine JADE agents and ontologies was by representing ontologies as Java classes. Therefore, we have decided to implement our simplistic ontology directly into Java classes using the Ontology Beans (without the OWL demarcation). It is only now that we have developed a JADE add-on that allows agents communicate using OWL snippets (see, [35] for more details). This will allow us to actually use OWL ontologies directly, in the next release of the system.

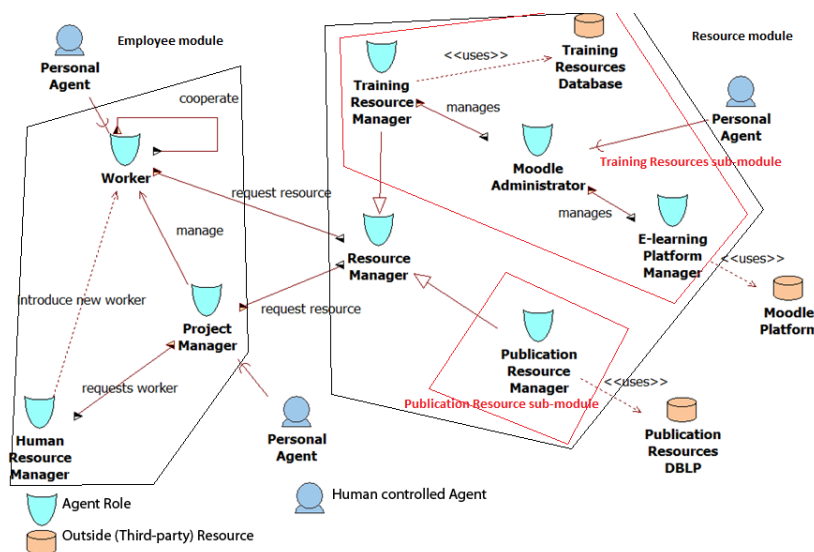


Fig. 4. System modules design

Let us now look in more details into the overall structure of the system and the implemented modules, which have been represented in Fig. 4.

### 3.1. Resource Module

Let us start with the *Resource Module*. It deals with all aspects of searching and obtaining different *Information Resources*, including managing databases, and e-learning platforms. It was divided into the following sub-modules: the *Publication Resource* module, the *Training Resource* module, and the *Connector*.

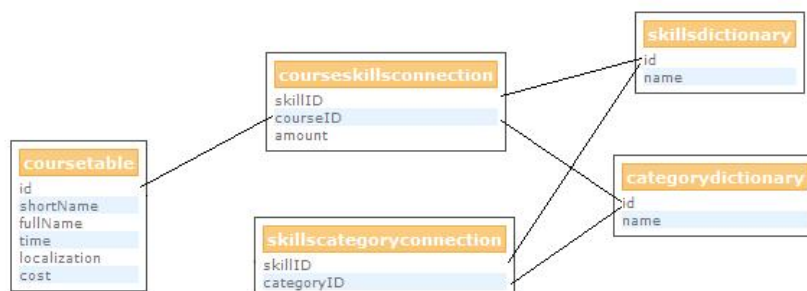
Here, the *Connector* consists of the *Resource Manager Agent*, which connects the *Resource Module* and the *Employee Module*. On the setup it starts two cyclic behaviors that are responsible for periodically refreshing the list of available *Publication/Training Resource Manager Agents*. This is achieved by asking the JADE Directory Facilitator (*DF*) for the list of such agents. All requests from the *Worker Agent* (to search for the *IRs*), received by the *Resource Manager Agent*, are copied into identical messages send to the *Training Resource* module and the *Publication Resource* module. Each of these messages contains added information about the original sender. Next, the *Resource Manager Agent* awaits responses from all specialized *Resource Managers* and merges answers received from them into a single *ResourceList* object. Here the inheritance relation between the *Resource* and the *PublicationResource*, as well as the *TrainingResource*, is used. The resulting list is forwarded to the proper *Worker Agent* (identified through the *Sender* object included in the message). In this way, the *Employee Module* is not (and does not have to be) aware about possible sources of *Information Resources* (which, therefore, can be dynamically added/removed, with only localized changes in the systems).

The *Training Resource* module is responsible for managing content of the e-learning platform—in our case the Moodle Platform—and searching for courses matching the requested topics. It is composed of three agents: (i) *E-learning Platform Manager Agent*, (ii) *Training Resource Manager Agent* and, human controlled, (iii) *Moodle Administrator Agent*. The *E-learning Platform Manager Agent* is directly connected to the database of the Moodle Platform and is responsible for translating requests received from the *Moodle Administrator Agent* to the SQL queries, and executing them. On the startup, in addition of the standard setup, this agent establishes connection with the database (using data retrieved from a file). Such solution allows developers to create different configuration for different Moodle Platforms by starting multiple instances of the same agent. Note that, during system development, we used the standard Java JDBC method to interface the database.

The *Moodle Administrator Agent* is a user-driven agent. It has two cyclic behaviors. First, it is responsible for updating the list of available Moodle Platforms (by periodically asking the JADE Directory Facilitator about agents that advertise themselves as agents interfacing the Moodle Platforms). The second cyclic behaviour looks for existing *Training Resources Manager Agents*. The remaining behaviors are triggered by user actions and deal with: (1) creating a new Moodle User Account, (2) selecting a Moodle User Account, (3) updating a

Moodle User Account, (4) deleting a Moodle User Account, (5) creating a new Moodle Course, (6) selecting a Moodle Course, (7) updating a Moodle Course, and (8) deleting a Moodle Course.

The last agent of the *Training Resource* module is the *Training Resource Manager Agent*. On setup, it connects to an auxiliary course database, created to speed-up the searching process and to lower the cost of communication between the system and the Moodle Platform. This course database allows also for a single *Training Resources Manager Agent* to handle more than one e-learning platform and/or different types of them. It contains description of available courses in a format corresponding to the ontology of training resources (see, Fig. 5, for more details).



**Fig. 5.** Training Resource Database

When searching for the *Training Resources* concerning a given subject (identified by the *Skill* name) the *Training Resource Manager Agent* looks in the *Skill-Dictionary* table for the ID of such *Skill* and then creates a query that will return all *IR* that cover the selected topic(s) and has the *Skill* level above the received requirement. This means that, after attending a specific e-course, worker should obtain the corresponding knowledge. All courses found as a result of the query, are send back to the *Resource Manager* (without any knowledge to whom they will be forwarded to). Next, the *Resource Manager Agent* forwards them to the requesting *Worker Agent*.

In the system prototype, the second source of the *Information Resources* is the *Publication Resource* module. Currently, it contains only one agent, which is responsible for communicating with the DBLP database. However, in the future, it is planned (and it is a natural extension of the proposed system design) to add additional agents that will facilitate information from other / additional sources. Here, the keywords (*Skill* names) received from the *Worker Agent* are transformed into an appropriate query-string, and executed on the DBLP interface. The received response is parsed from the HTML code, and filtered using the *Skill level* parameter (to select these resources that are actually needed). Next,

it is stored in the *ResourceList*. If required, the *ResourceList* may be trimmed to the requested length (note that the query to the DBLP—as well as to any other data source—may return hundreds of responses). Next, the *ResourceList* is send back to the requesting *Worker Agent*.

### 3.2. Employee module

The second specialized module is the *Employee Module*, which is composed of three agents: (a) *Project Manager Agent*, (b) *Worker Agent*, and (c) *Human Resource Manager Agent*. The last agent is completely autonomous, while the first two require human interactions (at least in the current design of the system).

The *Human Resource Manager Agent* emulates functions of a human *HRM*, supported by an appropriate *Worker Agent*. Its only role is to fetch (from a file) a list of “not hired workers” and, when requested, search for a workers with needed skills. This is to emulate situation when the *PM* finds out that she needs to hire extra workers to complete the project and requests help from the *HRM*.

The *Worker Agent* supports the *Worker* and represents her/him in the system. It stores the *Worker* profile and, if appropriate, the *Task* objects representing tasks currently assigned to its owner. This is the only agent (in this module) that communicates with the *Resource Manager Agent*, to ask for *Information Resources*. This behavior can be triggered manually by the *Worker*, or autonomously during the project management process. It is facilitated by creating a list of needed *Skills*. This list is send to one, or more, *Resource Manager Agents*. On demand, this agent can send the list of *Skills* of its *Worker* (owner), to the *Project Manager Agent*, so that it can be used in the task-worker matching process (using data as actual as possible). Furthermore, the *WA* responds to the queries from the *Project Manager Agent*, asking which tasks, stored in the *Project* predicate object, can be completed by the *Worker*. Second, it accepts the information that its *Worker* was assigned a task, and appropriately modifies her/his profile (availability schedule).

Note that, in the case when a “not hired” employee receives a message containing an information that a task was assigned to it, it changes its registration type, in the JADE Directory Facilitator, from “not hired” to “hired.”

The last agent in this module is the *Project Manager Agent*. As a matter of fact, this is a *PA* extended with the project management related capabilities (see, also [15, 17, 16]). This agent is responsible, first, for communication between the “outside world” and the system. Here, it accepts the *Project* objects, representing projects to be dealt with. It also communicates with the *Human Resource Manager Agent* and the *Worker Agents* (representing potential project workers). On the startup, it initiates two cyclic behaviors responsible for updating the list of available *Human Resource Manager Agents*, and the list of hired *Worker Agents*. This is done by cyclically asking the JADE *DF* for the current list of such agents. Currently, the *Project Manager Agent* handles project management from the moment of receiving a new project, to sending assigned tasks to the selected *Worker Agents*, or to declining its completion (due to the lack of resources / workers).

### 3.3. Matchmaking in the system

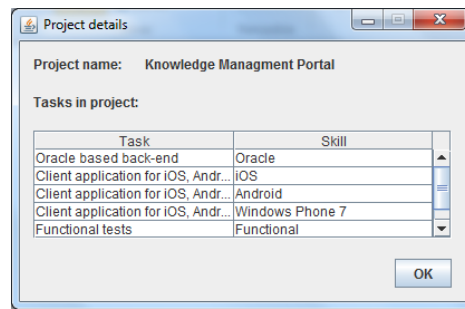
While in [30] a complex method for ontological matchmaking was proposed, here we decided for a more simplistic approach. As stated above (sections 2.3, and 3), we have deliberately elected to apply a very simple, Java class-based ontology; rather than a full blown OWL-based one (similar to that discussed in [17]). This being the case, at current stage of system design, applying complex ontological matchmaking would make no sense.

The implemented matchmaking aims at minimizing the gap between the needed and the existing skills. It assigns *Workers* to *Tasks* by picking an employee with the skill-level at least as high as that required by the task. The assessment of the skill-level is done independently by each *Worker Agent*, and is coordinated by the *Project Manager Agent*. The *Workers* that do not match the needed *Skill* set, can be (self)assigned to the training activities that should remove the gap and allow them to complete the task. In the process of establishing if the *Worker* can complete the task, the current schedule of the *Worker*, the time needed to accomplish the task, and (if necessary) the time of completion of the needed training activities are jointly taken into account. Matchmaking is also used by the *Human Resource Manager Agent* to find the best *Workers* to be hired, in the case that a project needs them. Agents that use the matchmaking engine include: the *Project Manager Agent*, the *Worker Agent*, and the *Human Resource Manager Agent*.

## 4. Experimental evaluation

Let us now illustrate work of our prototype, using a sample scenario that matches the first use case, discussed in section 1.2. Let us assume that the execution environment contains the followings units: single instances of *Project Manager Agent*, *Resource Manager Agent*, *Publication Resources Manager Agent*, *Training Resource Manager Agent*, *Human Resource Manager Agent*, and five *Worker Agents*, three employees (WA1, WA2, WA3), with the following skills: Oracle back-end developer, front-end mobile platforms developer, and graphics / web designer. Furthermore, we have two workers in the pool managed by the *Human Resource Manager Agent* (WA4 and WA5), both graphics / web designers. Here, recall that the *Human Resource Manager Agent* emulates the process of hiring additional employees. The *Publication Resources Manager Agent* and the *Training Resource Manager Agent* are connected to their respective sources of *Information Resources* (DBLP and Moodle). Furthermore, the *Training Resource Manager Agent* is connected with the *Training Resource Database* storing information about current course offerings. The sample project under consideration matches the example presented in section 1.1. It consists of four tasks: T1—implementation of a back-end that uses an Oracle database, T2—implementation of a mobile application for the iOS, Android and Windows Phone, T3—performing functional tests, and T4 - creation of the layout of the graphic interface for the new application.

Let us now assume that the WA1 and the WA2 have skills connected with the task T1 (and that the WA1 is a better match—more skills satisfy task requirements); however, the WA2 also can be assigned to complete task T2. Furthermore, worker WA3 may execute either task T3 or task T4, but with a skills preference favoring execution of task T3. Finally, both workers WA4 and WA5 have skills corresponding to tasks T3 and T4 (however, WA5 is a better match for either one of them). The fact that a worker has skills connected to the task does not mean that all of the required skills are present. This can also mean that some of needed skills are not at the required level, or that some of them are missing (see, also Fig. 6). We will now discuss what is happening when the project is introduced to the system, from the “point of view” of each major agent / module.



The screenshot shows a window titled "Project details" with a close button. Inside, the "Project name" is "Knowledge Management Portal". Below it, the "Tasks in project:" section contains a table with two columns: "Task" and "Skill".

Task	Skill
Oracle based back-end	Oracle
Client application for iOS, Andr...	iOS
Client application for iOS, Andr...	Android
Client application for iOS, Andr...	Windows Phone 7
Functional tests	Functional

An "OK" button is located at the bottom right of the dialog box.

Fig. 6. Skills connected to tasks—without skills levels

#### 4.1. Project Manager Agent

From the perspective of the *Project Manager Agent*, the scenario begins when it receives a message containing an ontologically described project to manage. As its first activity, the *Project Manager Agent* refreshes the list of workers (*Worker Agents*) available in the organization and, in the considered example, finds out that these are WA1, WA2 and WA3. Now it can send a call for proposals to all of them, seeking task executors. Upon reception of responses (let us omit the case of non-responsive WAs) the *Project Manager Agent* matches proposals to tasks, to select the best *Worker Agent* for each of them. In our situation, after the matching is completed, the T1 will be preliminarily assigned to the WA1, T2 to the WA2 and T3 to the WA3; while task T4 will be without a worker assigned to it. Now, the *Project Manager Agent* will send a message to the *Human Resource Manager Agent*, with a request to hire a new employee that will match the requirements of task T4. After obtaining information that worker WA5 should be hired, the *Project Manager Agent* can change the preliminary assignment of tasks to the final one (by re-evaluating the available *Workers* vis-a-vis the required tasks), and send to appropriate *Worker Agents* their assigned



tasks—T1 to WA1, T2 to WA2, T3 to WA5 and T4 to WA3. In this moment this scenario is completed for the *Project Manager Agent*; see, also Fig. 4.1.

Task	Worker
Oracle based back-end	Ava Roberts
Client application for iOS, And...	Emily Phillips
Functional tests	Landon Phillips
Interface creation	Landon Garden

Fig. 7. Task-worker matching

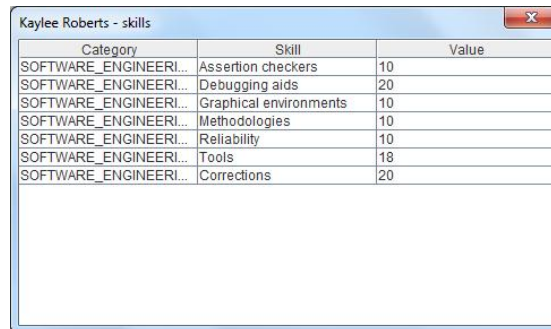
#### 4.2. Worker Agent

Now let's consider the *Worker Agent* perspective (here, we will use the WA2 as the example). Since all *Worker Agents* execute the same behaviors, the specific case of W2 represents the general *Worker Agent* behavior. Here, our discussion will need a more detailed set of assumptions. Let us, therefore, assume that the WA2 has the following skills (called *Worker Skills* WS; see, Fig. 8): WS1, iOS skill at level 53, WS2, Android skill at level 61, WS3, Oracle skill at level 67. The WA2 can, of course, have also other skills, but in this example they will be irrelevant, so we omit them. The specification of the T1 states that the needed skills are (*Task Skills* TS, where  $S_n$  in  $WS_n$  and  $T_m S_n$  denote the same skill  $n$ ): T1S3 at level 70; and T2: T2S1 at level 69, T2S2 at level 60 and T2S4, Windows Phone skill, at level 34.

For each *Worker Agent* this scenario starts when it receives a message with a request to perform the individual task matching process. This message can be sent by the *Project Manager Agent* or the *Human Resource Manager Agent*. However, this is inconsequential for the actions undertaken by the *Worker Agent*. Each task will be checked against the *Worker* skills. In this stage the *Worker Agent* can observe that it matches all skills needed for task T1, but it needs to improve one skill by a small amount ( $T1S3 - WS3 = -3$ ). In the case of task T2, one new skill has to be acquired (S4) and one has to be improved by a significant amount (16). To check if it is possible to acquire/improve skills, the *Worker Agent* will communicate with the *Resource Manager Agent* seeking resources for topics related to S1, S3 and S4. After obtaining the information about appropriate resources, the *Worker Agent* will create suggestions, which tasks can be assigned to it. In our case, both the T1 (see, Fig. 9) and the T2 (see, Fig. 10) will be acceptable. Now, the *Worker Agent* replies to the *Project Manager Agent* with its suggestions, and waits for further requests. This scenario can end with

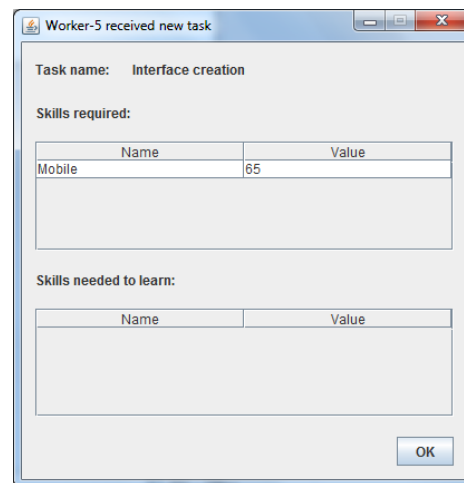
Maria Ganzha et al.

no task assigned to the worker (here, WA4), or with a reception of a message containing assigned task(s), and (if required) list of found *IRs* (see, Fig. 11).



Category	Skill	Value
SOFTWARE_ENGINEERI...	Assertion checkers	10
SOFTWARE_ENGINEERI...	Debugging aids	20
SOFTWARE_ENGINEERI...	Graphical environments	10
SOFTWARE_ENGINEERI...	Methodologies	10
SOFTWARE_ENGINEERI...	Reliability	10
SOFTWARE_ENGINEERI...	Tools	18
SOFTWARE_ENGINEERI...	Corrections	20

Fig. 8. Worker skills with levels



Worker-5 received new task

Task name: Interface creation

Skills required:

Name	Value
Mobile	65

Skills needed to learn:

Name	Value
------	-------

OK

Fig. 9. Task assigned to worker with none learning activities to perform

#### 4.3. Human Resource Manager Agent

For the *Human Resource Manager Agent* the scenario starts when it receives a message-request, from the *Project Manager Agent*, to hire a new employee that could work on the task T4. Work of the *Human Resource Manager Agent* consists of comparing skills required for the T4 to skills provided by workers

## Information Resource management in a VO

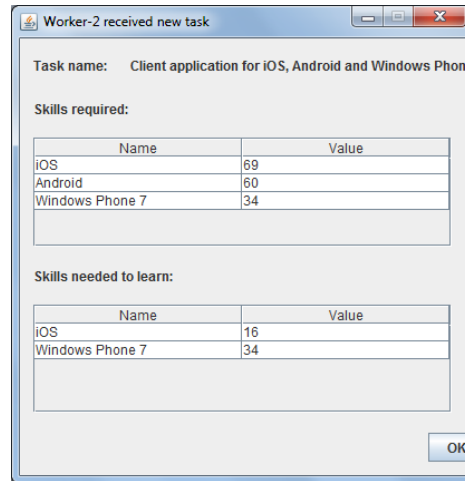


Fig. 10. Task assigned to worker with learning activities

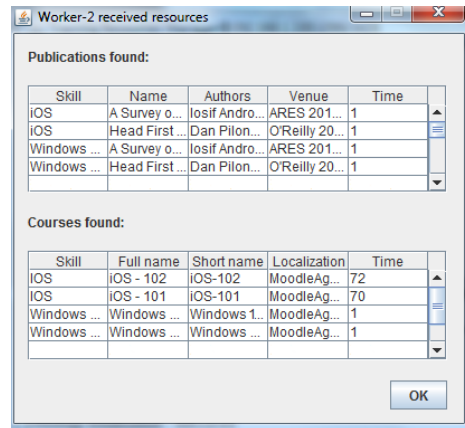


Fig. 11. Worker with found resources

WA4 and WA5. Upon completing such comparison, it decides that WA5 is more suitable for this task. Therefore, the *Human Resource Manager Agent* sends a message with the WA5 contact information and its skills to the *Project Manager Agent*.

#### 4.4. Resource Module

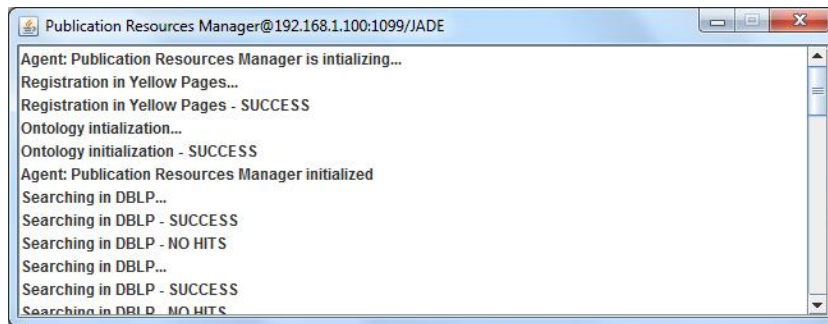
Now let us see how this scenario looks from the perspective of the *Resource Module*. First, the *Resource Manager Agent* is contacted by a *Worker Agent* (here, agent representing worker WA2) with a request to search for resources that can improve its skills S1, S3 and S4. It forwards this message to the *Training Resource Manager Agent* and to the *Publication Resources Manager Agent*. Contact information for both agents is obtained from the JADE Directory Facilitator. The dispatched message contains the list of *Skills* and an added information that it was the WA2 that originated the query. After obtaining both answers (again, we omit the case of non-responsive entities) the *Resource Manager Agent* forwards the combined *ResourceList* back to the WA2.

When processing the request, both the *Training Resource Manager Agent* and the *Publication Resources Manager Agent* work in a very similar way. Here, the *Training Resource Manager Agent*, after obtaining request to search for courses concerning S1, S3 and S4, will search the course database. Suppose that for skills S1 and S3 there was only a single course available for each of them, but for skill S4 there were 6. Now the *Training Resource Manager Agent* will trim this number, and selects only a predefined number of courses, say two. Selection of those two courses is based on the skill level, duration, and cost of each one of them (see, Fig. 13). The *ResponseList* (consisting of courses applicable for each of the three skills) will be send back to the *Resource Manager Agent*. At the same time, the *Publication Resources Manager Agent* will perform similar actions, but instead of searching the course database it will build a query that will be passed to the DBLP search engine. Next it will parse the obtained answer (see, Fig. 12). Finally, the resulting (possibly trimmed) list of publications will be send back to the WA2, as the *ResponseList*.

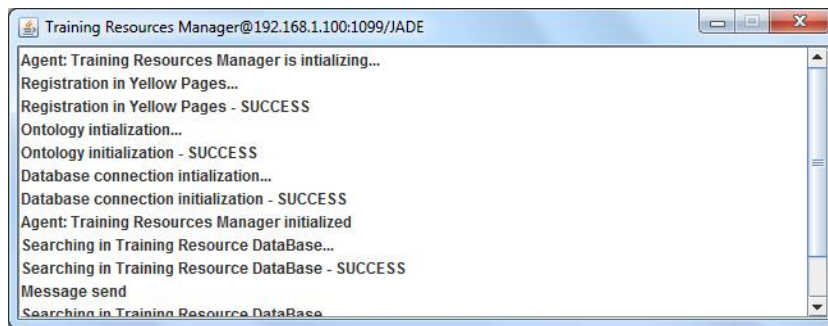
Note that this scenario captures characteristics of both use cases described in section 1.2. Specifically, the individual support case is subsumed by the above scenario. The *Worker Agent* could act not only in response to the message from the *Project Manager Agent*. It could also work either on request of its owner, or autonomously (proactively or reactively); in all cases it would be searching for the needed resources. Thus the use case 1.2, would simply start from a message send from the *Worker Agent* to the *Resource Manager Agent*.

Let us complete our description by the depiction of an actual communication between the above described JADE agents. This communication was captured in a running system using the JADE Sniffer Agent, and is represented in Fig. 14.

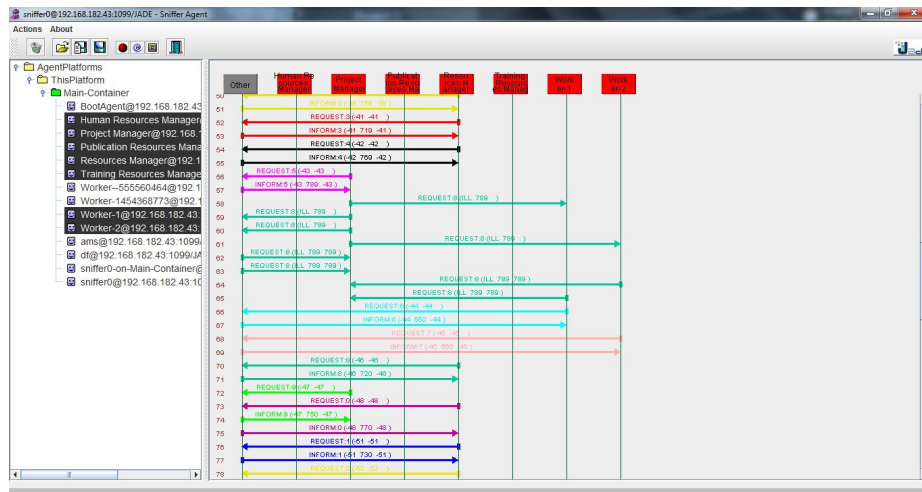
## Information Resource management in a VO



**Fig. 12.** Searching in Digital Bibliography and Library Project—none resources found



**Fig. 13.** Searching in Training Resources Database—resources found



**Fig. 14.** Communication process—messages sent in the system—captured by JADE Sniffer Agent

## 5. Concluding remarks

The aim of the presented work was to describe a prototype of an agent-based *Information Resources* provisioning system, facilitating support for workers in a virtual organization. The implemented prototype consists of agents directly supporting workers and project managers (interacting with humans), as well as autonomous agents that facilitate services needed for functioning of the system. The implemented prototype has been interfaced with the DBLP library (as an example of publication resource provisioning), and with the Moodle platform (as an example of e-learning content provisioning).

While the system works with ontologically demarcated resources and semantic information processing, we have selected a restricting ontological representation, based on Java classes. This decision was a deliberate one and was caused by factors outlined in sections 2.3 and 3.

In view of known shortcomings of the initial prototype, we plan to proceed as follows. First, we will evaluate lessons learned during system development and testing. Second, we will replace Java class-based ontology by an OWL-based one, and adapt all needed information processing and communication procedures. Here, we will use our JADE add-on that allows JADE agents to communicate by sending OWL snippets (without compiling ontologies into Java classes, see [35]). We will also introduce a newly designed front-end based on the Play framework (for more details, see [35]). Furthermore, we will expand the list of sources of *Information Resources* (both publication and training), to make the system truly heterogeneous. We will report on our progress in subsequent publications.

## References

1. Dublin core webpage. <http://dublincore.org/>
2. Java Agent DEvelopment framework. <http://jade.tilab.com/>
3. Log4JADE Agent-based Logging Service. <http://log4jade.sourceforge.net/>
4. Moodle. <http://moodle.org/>
5. <http://www.businessdictionary.com/definition/virtual-organization.html> (2008)
6. <http://www.mwi.pl/badania-i-innowacje/projekty/wkup.html> (2011)
7. The acm computing classification system. <http://www.acm.org/about/class/ccs98-html>
8. Badica, C., Popescu, E., Frackowiak, G., Ganzha, M., Paprzycki, M., Szymczak, M., Park, M.W.: On human resource adaptability in an agent-based virtual organization. In: N.T. Nguyen, R.K. (ed.) *New Challenges in Applied Intelligence Technologies. Studies in Computational Intelligence*, vol. 134, pp. 111–120. Springer, Heidelberg, Germany (2008)
9. Barnatt, C.: Office space, cyberspace and virtual organization. *Journal of General Management* 20(4), 78–92 (1995)
10. Bleeker, S.: *The Virtual Organization*. Sage Thousand Oaks (CA) (1998)

11. Cervenka, R., Trencansky, I.: *The Agent Modeling Language - AML*. Birkhuser Basel (2007)
12. Czerniejewski, B.: *Personal Communication*
13. Dieter, F.: *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, New York (2003)
14. Dunbar, R.: *Virtual Organizing*, pp. 6709–6717. Thomson Learning, London (2001)
15. Frackowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M., Szymczak, M., Bădică, C., Han, Y.S., Park, M.W.: *Adaptability in an agent-based virtual organization*. *International Journal Accounting, Auditing and Performance Evaluation* (2008), in press
16. Frackowiak, G., Ganzha, M., Paprzycki, M., Szymczak, M., Han, Y.S., Park, M.W.: *Adaptability in an agent-based virtual organization—towards implementation*. In: *WEBIST (Selected Papers)*. pp. 27–39 (2008)
17. Ganzha, M., Gawinecki, M., Szymczak, M., Frackowiak, G., Paprzycki, M., Park, M.W., Han, Y.S., Sohn, Y.: *Generic framework for agent adaptability and utilization in a virtual organization—preliminary considerations*. In: Cordeiro, J., et al. (eds.) *Proceedings of the 2008 WEBIST conference*. pp. IS–17–IS–25. INSTICC Press (2008)
18. Ganzha, M., Paprzycki, M., Gawinecki, M., Szymczak, M., Frackowiak, G., Badica, C., Popescu, E., Park, M.W.: *Adaptive information provisioning in an agent-based virtual organization—preliminary considerations*. In: Nguyen, N. (ed.) *Proceedings of the SYNASC Conference*. LNAI, vol. 4953, pp. 235–241. IEEE Press, Los Alamitos, CA (2007)
19. Goldman, S., Nagel, R., Preiss, K.: *Agile Competitors and Virtual Organizations*. Van Nostrand Reinhold, New York (1995)
20. Hendler, J.: *Agents and the semantic web*. *IEEE Intelligent Systems* 16(2), 30–37
21. Jennings, N.: *An agent-based approach for building complex software systems*. *Commun. ACM* 44(4), 35–41 (2001)
22. Jennings, N., Wooldridge, M.: *Agent technology: foundations, applications, and markets*. Springer (1998)
23. Kim, H., Fox, M., Gruninger, M.: *An ontology for quality management—enabling quality problem identification and tracing*. *BT Technology Journal* 17(4), 131–140 (1999)
24. Laclavik, M.: *Ontology and agent based approach for knowledge management*
25. Laclavik, M., Balogh, Z., Hluchy, L., Nguyen, G., Budinska, I., Dang, T.: *Pellucid agent architecture for administration based processes*
26. Maes, P.: *Agents that reduce work and information overload*. *Commun. ACM* 37(7), 30–40 (1994)
27. Nwana, H.S., Ndumu, D.T.: *Software agents: an overview*. In: *Knowledge Engineering Review*, vol. 11, pp. 205–244. Cambridge University Press (1999)
28. Omelczuk, A., Wypysiak, M.: *Managing human resource adaptability in an agent-based virtual organization* (2011)
29. <http://ontoweb.aifb.uni-karlsruhe.de/Ontology/index.html>
30. Rhee, S.K., Lee, J., Park, M.W., Szymczak, M., Frackowiak, G., Ganzha, M., Paprzycki, M.: *Measuring semantic closeness of ontologically demarcated resources*. *Fundam. Inform.* 96(4), 395–418 (2009)
31. Szymczak, M., Frackowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M., Park, M.W.: *Resource management in an agent-based virtual organization—introducing a task into the system*. In: *Proceedings of the MaSeB Workshop*. pp. 458–462. IEEE CS Press, Los Alamitos, CA (2007)
32. <http://dblp.uni-trier.de/>

Maria Ganzha et al.

33. <http://www.eil.utoronto.ca/enterprise-modelling/index.html>
34. Warner, M., Witzel, M.: Zarzadzanie organizacja wirtualna. Oficyna Ekonomiczna (2005)
35. Wasielewska, K., Drozdowicz, M., Szmaja, P., Paprzycki, M., Ganzha, M., Lirkov, I., Petcu, D., Badica, C.: Agents in grid system—design and implementation. Springer (2011), to appear
36. Wooldridge, M., Jennings, N.R., Kinny, D.: The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems* 3(3), 285–312 (2000)

**Maria Ganzha** obtained M.S. and her Ph.D. in Applied Mathematics from the Moscow State University, Moscow, Russia in 1987 and 1991 respectively. Her initial research interests were in the area of differential equations, solving mixed wave equations in space with disappearing obstacles in particular, currently she works in the areas of software engineering, distributed computing and agent systems in particular. She has published more than 100 research papers and is on editorial boards of 5 journals and a book series and was invited to Program Committees of over 100 conferences.

**Adam Omelczuk** is an MS student at the Warsaw University of Technology, where he is applying agent-oriented programming paradigm to the personalized information delivery process. His research interests focus on use of ontologies and semantic data processing.

**Marcin Paprzycki** (Senior Member of the IEEE, Senior Member of the ACM, Senior Fulbright Lecturer, IEEE CS Distinguished Visitor) has received his M.S. Degree in 1986 from Adam Mickiewicz University in Poznan, Poland, his Ph.D. in 1990 from Southern Methodist University in Dallas, Texas and his Doctor of Science Degree from Bulgarian Academy of Sciences in 2008. His initial research interests were in high performance computing and parallel computing, high performance linear algebra in particular. Over time they evolved toward distributed systems and Internet-based computing; in particular, agent systems. He has published more than 350 research papers and was invited to Program Committees of over 400 international conferences. He is on editorial boards of 14 journals and a book series.

**Mateusz Wypysiak** is an MS student at the Warsaw University of Technology, where he is applying agent-oriented programming paradigm to the information delivery process. His research interests focus on practical aspects of delivering of personalized information.

*Received: January 17, 2012; Accepted: May 25, 2012*