Adaptability in an agent-based virtual organisation

G. Frąckowiak, M. Ganzha*, M. Gawinecki, M. Paprzycki and M. Szymczak

Systems Research Institute Polish Academy of Sciences,
Warsaw, Poland
E-mail: Grzegorz.Frackowiak@ibspan.waw.pl
E-mail: Maria.Ganzha@ibspan.waw.pl
E-mail: Maciej.Gawinecki@ibspan.waw.pl
E-mail: Marcin.Paprzycki@ibspan.waw.pl
E-mail: Michal.Szymczak@ibspan.waw.pl
*Corresponding author

C. Bădică

Software Engineering Department
University of Craiova
Bvd. Decebal 107, Craiova, RO-200440, Romania
E-mail: badica_costin@software.ucv.ro

Yo-Sub Han and Myon-Woong Park

Korea Institute of Science and Technology
Seoul, Korea
E-mail: emmous@kist.re.kr
E-mail: myon@kist.re.kr

Abstract: In this paper we consider the adaptability in an agent-based Virtual Organisation (VO). After introducing the system and the need for adaptive behaviours, human resource adaptability in particular, we discuss how the goal of 'agent adaptation' can be implemented. Specifically, we propose how agent behaviours can be added into/removed from/replaced in an agent skeleton, and, in this way, agent functionality modified.

Keywords: agent systems; virtual organisation; electronic learning; e-learning; adaptability.

Reference to this paper should be made as follows: Frąckowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M., Szymczak, M., Bădică, C., Han, Y-S., Park, M-W. (xxxx) 'Adaptability in an agent-based virtual organisation', *Int. J. Agent-Oriented Software Engineering*, Vol. X, No. Y, pp.000–000.

Biographical notes: Grzegorz Frąckowiak is a Software Developer at a leading Polish IT company and a researcher at the Polish Academy of Sciences in Warsaw, Poland. His main interests are agent technologies, ontologies and mobile technologies. Since 2007 he has participated in a Polish-Korean

project, the main purpose of which is to create an agent-based virtual organisation. During this time he coauthored more than ten articles based on the project results.

Maria Ganzha obtained her MS and her PhD in Applied Mathematics from 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 70 research papers, is on the editorial boards of 5 journals and a book series, and was invited to the programme committees of over 40 conferences.

Maciej Gawinecki received his MS degree from Adam Mickiewicz University in Poznań, Poland, in 2005. Between 2006 and 2008, he worked at the Polish Academy of Sciences as a Researcher in the E-CAP project. He is currently a PhD student at the University of Modena and Reggio Emilia, Modena, Italy. His research interests are in agent-based computing, ontologies, semantic processing and internet computing.

Marcin Paprzycki (Senior Member of the IEEE and Senior Fulbright Lecturer) received his MS degree from Adam Mickiewicz University in Poznań, Poland in 1986 and his PhD from Southern Methodist University in Dallas, Texas, USA in 1990. His initial research interests were in high-performance computing and parallel computing, high-performance linear algebra in particular. Over time they evolved towards distributed systems and internet-based computing, in particular, agent systems. He has delivered more than 100 invited presentations at conferences and seminars and has published more than 250 research papers. He was also invited to the programme committees of over 300 international conferences and is a member of the editorial boards of 15 journals and a book series.

Michał Szymczak is researching semantic network solutions for personalised information provisioning in the System Research Institute of the Polish Academy of Sciences (SRI PAS). He graduated with an MS degree in Computer Science from the Adam Mickiewicz University in Poznań, Poland in 2006. Since then he has been involved in online services design and project management in one of the largest polish IT companies, Comarch S.A. Since 2007 he has been designing a semantic network core for the joint Korean-Polish Adaptive and Personalized Information Provisioning project on behalf of SRI PAS.

Costin Bădică is a Professor at the Department of Software Engineering, University of Craiova, Romania. From 2001 to 2002 he worked as a Postdoctoral Researcher at the Department of Computer Science, King's College London, UK, on the application of formal representation and reasoning to business processes. He authored and coauthored about 90 articles in conference proceedings, journals and book chapters, 1 monograph and 5 textbooks. His publications in the last five years are related to applications of multi-agent systems, information extraction from the web and formal modelling of business processes. Prof. Costin Bădică is a co-initiator of the Intelligent Distributed Computing (IDC) series of symposia. He is also serving as a member of the editorial board of some international journals and he has co-organised and participated in the programme committees of many international conferences and workshops.

Yo-Sub Han is a Senior Research Scientist at the Korea Institute of Science and Technology. He received his PhD from the Hong Kong University of Science and Technology in 2005. His research interests are intelligent Human-Computer Interaction (HCI) and formal language theory.

Myon-Woong Park has been working on intelligent software for design and manufacturing since his PhD programme at the University of Manchester, UK, which was completed in 1987. His research interests have been widened recently to more general and futuristic applications such as intelligent office, e-learning and the smart home. The issue of the environment during the decision-making process in design and manufacturing was also added to his research area due to the positioning of the organisation he belongs to. He currently serves as a Principal Research Scientist of the Intelligence and Interaction Research Center and also a Professor for Human-Computer Interaction (HCI) and Intelligent Robotics of Korea Institute of Science and Technology.

1 Introduction

In some recent works (Ganzha *et al.*, 2007a–b; Szymczak *et al.*, 2007; 2008), we have argued that adaptability is an important feature needed in a system supporting workers in a Virtual Organisation (VO). To this effect we have claimed, first, that emergent software technologies such as the software agents Wooldridge (2002) and the ontologies SW (2008) should be the base around which the mapping between a real organisation and a virtual one should be conceptualised. Thus we have proposed a system in which:

- the organisational structure, consisting of specific 'roles' and the interactions between them, is represented by software agents and their interactions
- domain knowledge, resource profiles (representing organisational semantics) and resource matching are approached utilising ontologies and semantic reasoning.

Second, we have showed that as the real organisation changes, not only its ontology has to be adjusted, but also the 'mechanisms of interaction' within its agent-based 'representation'. Obviously, this concerns not only changes in the the organisational structure itself, but also the need to respond to the evolution of the projects carried by the organisation, as well as the changing *interests*, *needs* and *skills* of employees. Thus, we conjecture that adaptability within the organisation can be divided into:

1 *system adaptability*, obtained through:
   - adaptation within the 'structure' of the agent system
   - adapting the resource profiles
2 *human resources adaptability*, achieved through (e-)learning (training activities).

Thus far, first, we have outlined the processes involved when a task/project is introduced into an organisation (discussed from the point of view of resource management) (Szymczak *et al.*, 2007); second, in Ganzha *et al.* (2007a), we approached the proposed system from the point of view of the roles played by various entities identified in

Szymczak *et al.* (2007); while in Szymczak *et al.* (2008), we outlined how ontologies are going to be used in the proposed system. This allowed us to conceptualise, in Ganzha *et al.* (2007a), which roles can be played by:

- software agents alone
- by human(s)
- by human-agent team(s).

Finally, in Frąckowiak *et al.* (2008), we introduced our approach to the way that resource closeness is to be established (laying the ground for semantic reasoning, which is to be one of the core functionalities of the system). Additionally, in Ganzha *et al.* (2007b) and Bădică *et al.* (2008), we started discussions concerning the training activities within a *VO*.

The aim of this paper is to discuss both forms of adaptability in the system. In this context we first outline our general approach to agent adaptability. We follow this with a brief case study of human resource adaptability provided through (e-)training. Finally, we propose how agent adaptability can be actually implemented. It should be noted that, in this paper, we extend and modify the material presented in Ganzha *et al.* (2007b), Bădică *et al.* (2008) and Ganzha *et al.* (2008).

2 System overview

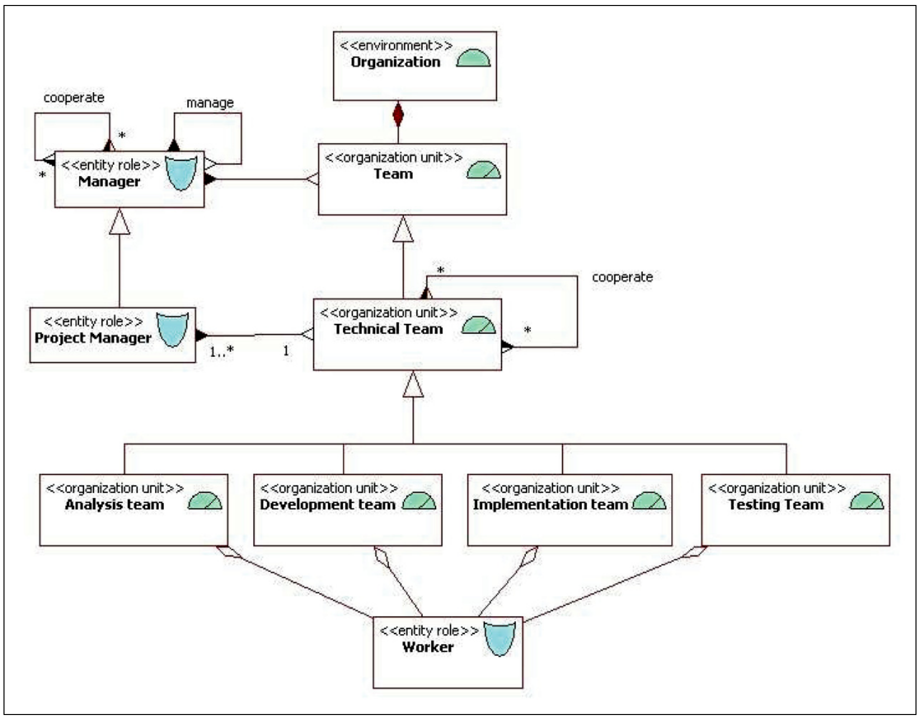
Let us start by briefly summarising the main features of the system. Every employee has an associated *Personal Agent* (*PA*). This agent has two main functions:

1 It is the interface between the *User* and the system.

2 It supports its owner in all the *roles* that (s)he is going to play within the organisation.

In our system, we assume that work carried out within the organisation is project-driven (however, the notion of the project is very broad and includes the installation of cable TV as well as the design and implementation of an intranet-based information system for a corporation). Therefore, it can be stated that all user activities are performed in order to fulfil project requirements. After analysis of project-driven real-world organisations, several roles were identified (see the Use Case diagram presented in Ganzha *et al.* (2007a)). However, further analysis revealed that these roles can be further compacted and represented in the form of an Agent Modelling Language (AML) Social Diagram, in Figure 1.

Figure 1 AML Social Model of an organisation (see online version for colours)



Here, we can see the general hierarchical management structure that can be applied to typical real-world organisations (an 'IT-related organisation' in the example, but this is easily generalised). The structure of the organisation consists of teams. Each team has at least one manager, who can:

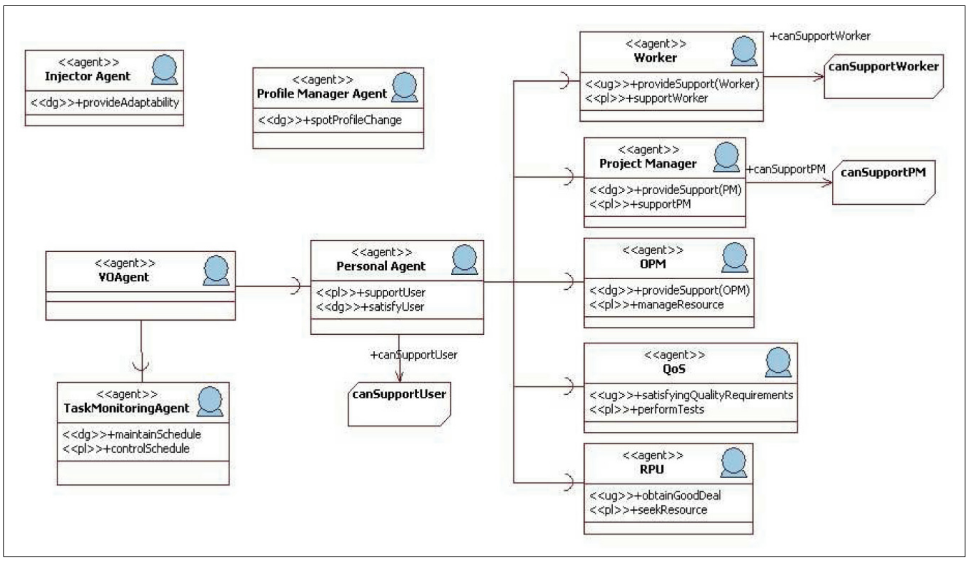
- manage a team
- supervise managers of lower levels (in this way a hierarchical structure of the organisation is represented)
- cooperate with other managers on the same level (*e.g.*, in the case of team collaboration).

Since AML is a relatively new notation (see Cervenka *et al.* (2007) for more details), let us note that a link with a white triangle at one end and a black triangle at the other means sub-super relationships, while a link with bi-colour triangles represents peer-to-peer relations. We also use the UML notation (AML is an extension of UML) and, thus, a white rhomboidal link represents aggregation, and a dark rhomboidal link denotes composition. The *Technical Team* is the team that is working on a project and is a specific instance of a *Team.* Obviously, in the organisation other teams are likely to exist – instances of the generic *Team* concept (*e.g.*, workers of the *Human Resource Department*). In Figure 1 the *Technical Team* is associated with a software-type project, but the specific roles of its members (possibly subteams) can be easily generalised to any project. In Figure 1 we also depict the *Worker*, who is a member of one of the teams. Note that:

- the *Organisation* is an ‘environment’ for *Managers*, *Teams* and *Workers*
- the *Organisation* cannot exist without at least one *Team*
- it is possible for a *Team* to consist only of a *Manager* – without any *Workers* (*e.g.*, the case of self-employment).

In Figure 1 we have concentrated on the ‘managerial hierarchy’ of a real-world organisation and depicted only two major roles: a *Worker* and a *Manager*. However, when such an organisation is mapped into an agent-based virtual organisation, additional roles can be identified. To this effect, in Szymczak *et al.* (2007), we analysed the processes involved in a project (task) being introduced to the organisation. This allowed us to identify additional roles related to resource management (rather than direct project realisation). These roles involve software agents and have been summarised, in the form of an AML Role Diagram, in Figure 2.

Figure 2 AML Role Diagram of the system (see online version for colours)



In this figure we conceptually move from the real-world organisation to the agent-based virtual organisation. We can see the *VOAgent*, the basic agent skeleton, which is transformed into either agents that are self-contained in their roles (they play these roles autonomously, without human involvement), *e.g.*, the *Task Monitoring Agent*, or agents that support employees in fulfilling specific roles in the organisation. Transformations of the *VOAgent* involve the *Injector Agent* and the *Profile Manager Agent*. Their actions are described in more detail in Section 4. Here, let us state that they monitor the state of the organisation and in appropriate moments provide selected agents with a list of ‘modules’ that they have to modify (add, replace or remove). These *modules* are understood as sets of behaviours and knowledge that support a given functionality. If the *VOAgent* is to support an *Employee*, then first it loads a set of core modules (the same for all employees) as well as modules related to a given *Employee* (*e.g.*, her *Personal Profile*), and becomes a *Personal Agent* (*PA*), with the role of supporting its *User*. Next, depending on the specific role that the *User* is to play in the real organisation, the *PA* loads additional modules that allow it to support its *User* in that particular role. Note that only in the case of *Worker* and *Project Manager* supporting agents have we identified a set of their beliefs related to the support of their *Users* in these roles. In the remaining cases we observe that roles such as *OPM*, *QoS* or *RPU* typically involve worker-teams and thus complex beliefs. Let us summarise the basic functionalities of the roles identified in Figure 2:

- The *Worker* is a default role of any human resource (employee) in the organisation.
- The *Project Manager* (*PM*) is a role that is associated with a project proposal when it is submitted to the organisation. Its main duties cover the formulation of project requirements; if the project is accepted, the formulation of the project schedule; the assignment of resources to project activities; supervising the project's progress and assuring its completion.
- The *Organisation Provisioning Manager* (*OPM*) is responsible for managing the resources of the organisation (it is assumed to have access to complete information about all resources in the organisation).
- The *Resource Procurement Unit* (*RPU*) represents an interface between the organisation and the 'outside world'. Its role is to seek and potentially deliver the resources requested by the *OPM.*
- The *Task Monitoring Agent* (*TMA*) is responsible for monitoring an assigned task according to its schedule and informing the *PM* about its completion or of any problems.
- The *Quality of Service* (*QoS*) management unit is responsible for the quality control of tasks completed by the workers.

Since each *Employee* is represented in the system by her/his *PA*, in its basic role, the *PA* provides rudimentary support for his/her functioning in the organisation. Such support involves (organisation-specific) *core functionalities* available to all workers within the organisation; *e.g.*, meeting scheduling, e-mail sorting and filtering, searching for resources (such as training modules), knowing whom to call in case of an emergency. In this way we follow the notion of the *PA* as conceptualised in Maes (1994). However, the *PA* has to also be extendable to support specific roles that the *User* has to fulfil. Note also that we assume that, in most cases, the role of the *TMA* can be fulfilled by a software agent alone, while the remaining roles may require the involvement of a human (whether this is the case or not depends on the operation mode of the specific *VO*) and only in this context do they appear in Figure 2. In other words, it is possible that a role of the *RPU* is to be fulfilled by an agent alone. In this case the *RPU* would be 'moved' (within Figure 2) to the same status/location as the *TMA*. Overall, the *PA* needs to be able to support the *Employee* in fulfilling the roles of *Worker*, *PM*, *OPM*, *RPU* and *QoS* (or any other (sub) roles specific for a given organisation, which may be specific instances of the basic roles identified in Figure 2). Note that we assume here that the role of a *Worker* is a default extension of the core functionalities of the *PA*. It involves, among others, 'placing' the *Employee* within the organisation, for instance, by providing the *PA* with information about which specific team the *User* belongs to and who is his/her direct supervisor. In this way, becoming a *PM* is not an extension of the role of the *Worker*, but an extension of the functionalities of the *PA* that has to support the *Employee* in that role.

3 Human-resource adaptability

Let us now turn our attention to adaptability in the system. We start by recognising the fact that, as time passes, not only human resources change 'on their own', *e.g.*, due to participation in projects, their knowledge expands. It is also possible that their capabilities have to be modified and/or additional knowledge served, *e.g.*, to successfully participate in a newly contracted project. Therefore let us look into human resource adaptability and start with defining a training task.

3.1 Conceptualising training tasks

In our work, we understand *training* in the context of vocational or practical skills and refer to it as *workplace learning* Training (Wikipedia) (2008). Clearly, in this case training tasks can and should be closely related to organisational projects and approached keeping in mind three issues:

1. *timing*, *i.e.*, when training should be started (possibly also: when it should end)
2. *goals*, *i.e.*, what should be the goals of each specific training activity
3. *trainees*, *i.e.*, who should be enrolled in a given training task.

Furthermore, *timing* is crucial for distinguishing between *reactive* and *proactive* training activities. To introduce the important features of both approaches, let us consider three possible situations:

1. *reactive approach: first case – project level.* When a new project is introduced into the organisation, the analysis process (see Ganzha *et al.* (2007b) for more details) may indicate that, to be able to accept it, selected employees need to be enrolled into training activities (carried out within the project bud get and time) to upgrade their skills and remove the gap between the project requirements and the skills present in the organisation.

   Note that the training decision depends on the following factors:

   - current level of competence of available resources
   - the competence increment, which represents the gap between available and required competencies for the job
   - project constraints.

   Furthermore, the introduction of the training tasks may also require an update of the *Project Schedule* (or even schedules and resource assignments of other running projects; but this scenario is out of the scope of this paper) to accommodate the new training activities within the project timing and costs (Ganzha *et al.*, 2007b).

2 *reactive approach: second case – individual or group level.* While the project is running, either the *PA* of an employee or the manager of that project (its *PM*) may decide to enrol an employee or group of employees in *ad hoc* training (possibly of small granularity) to acquire specialised knowledge increments to solve specific problems.

3 *proactive approach*. This occurs when the 'management', analysing current market conditions, the history of the interactions between the organisation and the external environment, specific regulations, expected projects, *etc.*, decides to enrol selected employees in training task(s).

We will now elaborate these three cases in the framework of a software and services company (however, they generalise naturally to other business areas). Let us consider an example of a customer requesting creation of an intranet and a company knowledge portal.

3.2 Reactive approach

Quite often (*e.g.*, in the case of Information Technology (IT) projects), the decision to start a project is taken even if there is no perfect match between the competencies of the available human resources and the needs of the required tasks (the match has to be just 'good enough'; see Frąckowiak *et al.* (2008) for more details). As a result, human resource adaptability issues may arise during the unfolding of the project (*e.g.*, finding tips on how to implement the Jade agent's mobility between platforms). In this case, the programmer informs her/his *PA* about the missing information that she/he needs in order to complete her/his task. It is the job of the *PA* to provide its *User* with the needed resource – either nonhuman (a manual, a tutorial, a book excerpt, *etc.*) or human (a peer who possesses the needed information and is able to share it). Here, we assume that every *PA* has the modules needed to perform such a search (within the resource space that it has access to, as granted through appropriate privilege profile(s)) (Szymczak *et al.*, 2008). The search may involve contacting the *OPM*, as well as other *PAs.* The ability to query other *PA*s is also restricted through the access profiles defining the organisational structure (obviously, it is very unlikely that a programmer will have direct access to the Chief Information Officer (CIO) of the company). This scenario is similar to well-known cases of collaborative filtering (see, for instance, Montaner *et al.*, 2003).

In the case of searching for nonhuman resources, we deal with *agent knowledge adaptability.* Specifically, knowledge of the *PA* will be updated with information about the location of the needed resource(s). Moreover, if the needed resource is a *tutorial* (or some other form of e-training), we have to provide the *User* not only with the information that an (*Advanced Jade Mobility*) e-learning module is available, but also with help to interface with it. This function is obtained through the adaptability of the *PA*. Specifically, the training material is associated with the interface module (provided by the system or, more likely, by the training material supplier). As a result the *Injector Agent* (see Figure 2, above, and the detailed discussion in Section 4) will provide the *PA* with the module supporting the needed functionality. Furthermore, the completion of training results in an update of the *Human Resource Profile* of the *User* and the removal (from the *PA)* of the no longer needed interface to the training module. This latter operation is needed to keep the *PA* clean of spurious modules and thus minimise resource utilisation.

In the case when the same help request appears from different programmers (note that each such request is stored in the *project log*), the *PM* (that analyses the *project log*) might consider ordering an *ad hoc* training on the topic, involving a selected group within the team (or the whole team). Here, a set of issues related to group training needs to be considered (*e.g.*, dealing with varying learning styles), but they are outside of the scope of this paper. The interested reader may consult Ganzha *et al.* (2007b) and Bădică *et al.* (2008) and the references presented there for more details.

A similar situation takes place when the *QoS* module reports that a task has not been carried out correctly by one or more team members (all reports from the *QoS* are also collected in the *project log*). Analysing the *project log*, the *PM* may decide that a just-in-time training is needed for one or more team members to improve their skills and reduce the number of incorrectly completed tasks.

Once more, group training involves providing selected *PAs* with information about the location of training materials and the appropriate interface module(s), as well as updating *User* profiles and the removal of any unnecessary module(s) upon training completion.

3.3 Proactive approach

Let us now consider the situation where a new project request is received and, for various reasons (which might include, among others, that required resources are missing and/or the requested expertise and competences are unavailable), it is determined that it should be rejected. Moreover, assuming that a situation like this is repeated, the management is faced with deciding whether to:

1 continuously reject similar project proposals (while there is a clear market interest)

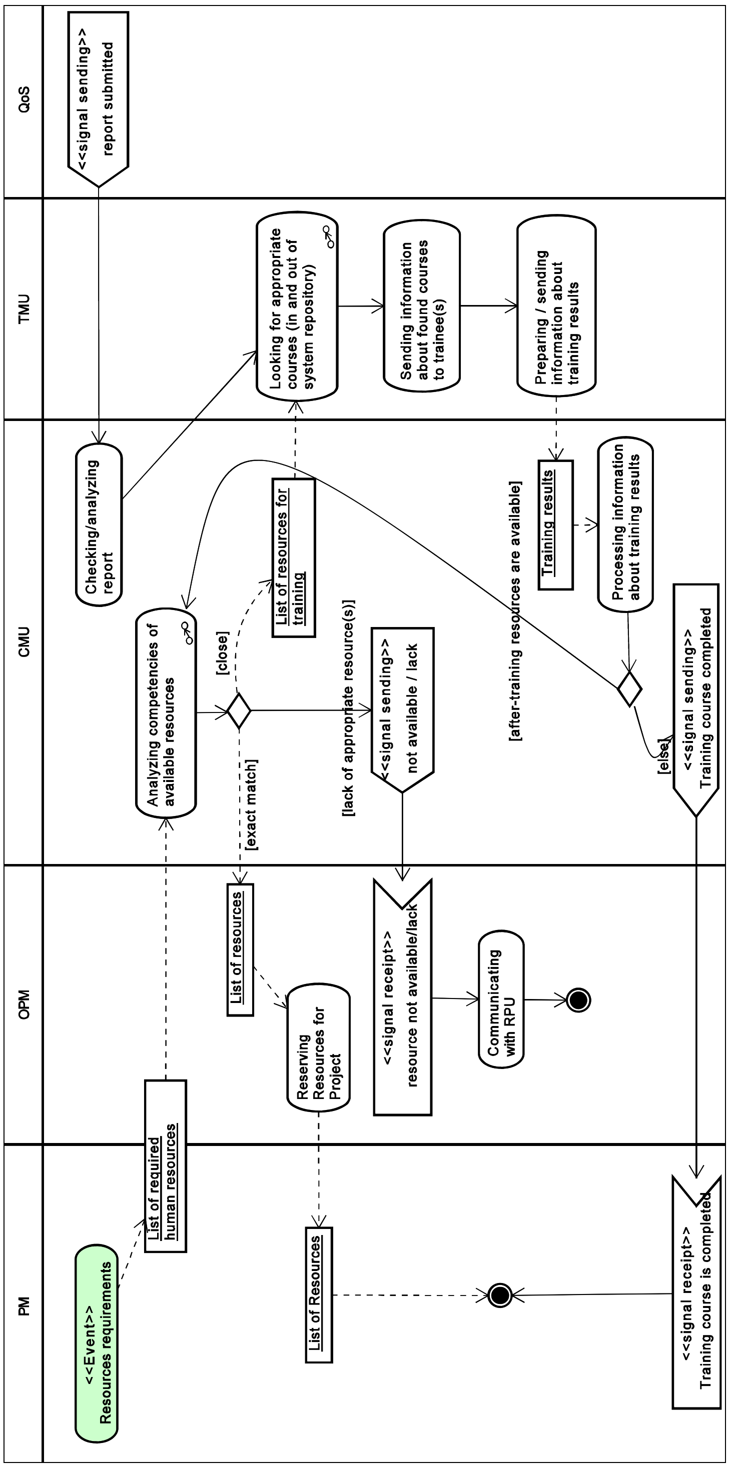
2 hire new staff

3 proactively involve the available human resources in training tasks.

While situations (2) and (3) are instances of human resource adaptation at the organisational level, clearly only option (3) is directly within the scope of this paper. It should also be noted that other scenarios pertinent to the proactive approach include the following:

- the organisational management expects a certain set of projects to materialise within short- or mid-range perspective
- an expansion or a change in direction of the organisation
- more generally, long-term and semi-long-term goals and strategies of the organisation
- a merger and/or acquisition, which will require synchronisation between the skills of workers from two (or more) different companies.

Figure 3 Interactions of the *CMU* and the *TMU* with other units in the system (see online version for colours)


QoS
TMU
CMU
OPM
PM
<<signal sending>> report submitted
Checking/analyzing report
Looking for appropriate courses (in and out of system repository)
Sending information about found courses to trainee(s)
Preparing / sending information about training results
Training results
Processing information about training results
[after-training resources are available]
[else]
<<signal sending>> Training course completed
List of resources for training
Analyzing competencies of available resources
[close]
[exact match]
[lack of appropriate resource(s)]
<<signal sending>> not available / lack
List of resources
Reserving Resources for Project
<<signal receipt>> resource not available/lack
Communicating with RPU
List of required human resources
<<Event>> Resources requirements
List of Resources
<<signal receipt>> Training course is completed

Upon further reflection, we see that while the reactive approach involves mainly decisions at the project level, the proactive approach involves mostly decisions at the higher organisational level. Separately, we observe that the granularity of training tasks (and consequently costs, time and effort) in the reactive approach should be expected to be substantially smaller than in the case of the proactive approach. For example, proactive training can include such resource-consuming tasks as continuing professional education, initial training for new employees (*e.g.*, 'school to work transition'), coaching and motivational seminars, and group/team-building activities. Note that the *PA*, which has access to the *User* calendar, allows the scheduling of proactive training in such a way that it will not collide with other professional activities. In this way we can observe how the *PA* can actually assist its *User* through intelligent training scheduling.

3.4 Competence and Training Management Units as training facilitators

Based on the material presented thus far, as well as on ideas found in related works (Schmidt and Kunzmann, 2006; Tzelepis and Stephanides, 2006), two specialised units are to be added to the proposed system (note that, following Szymczak *et al.*, 2007), we use the term 'unit' for each entity in the system; and associate specific roles with each one of them):

1 *Competence Management Unit (CMU)* – responsible for the management of competencies

2 *Training Management Unit* (*TMU)* – responsible for the management of training activities.

In what follows, we outline the main functionalities of these units and their interactions with other units existing in the system, as captured in Figure 3.

3.5 Competence Management Unit

The *CMU* is responsible for the management of competencies within the organisation. The representation of competencies will utilise the competence ontology described in Biesalski and Abecker (2005), HR-XML Consortium (2008) and Schmidt and Kunzmann (2006), and the associated reasoning mechanisms proposed in Mochol *et al.* (2007).

The functionalities of the *CMU* comprise:

- the management of individual competencies of all human resources in the organisation; this requires the ability to represent, record and update competencies at an individual level
- the provisioning of a global view of competencies available at the organisational level; this facility is required, for example, to be able to asses if the organisation has competencies 'good enough' to accept a given project
- qualitative and quantitative reasoning about matchings between available and required competencies; this functionality is needed to help decide whether to hire new staff (Bizer *et al.*, 2005; Mochol *et al.*, 2007), assign human resources to tasks or enrol human resources in training.

Note that the *PM* and the *OPM* have to interact with the *CMU* during the process of fixing the problem of missing resources. Furthermore, the *CMU* will utilise information from the *QoS* unit, which assesses the work done by individuals and teams (each time a task is completed, the *QoS* checks the result). This being the case, the *QoS* can provide the *CMU* with information which tasks have been successfully or unsuccessfully completed. This information, in turn, can be used to assess which individuals or teams need extra training (*i.e.*, training needs can be assessed directly on the basis of on-the-job performance).

3.6 Training Management Unit

The introduction of the *TMU* is motivated by the need for a specialised unit that is capable of formulating training goals for employees engaged in training activities, based on the contextual conditions that resulted in training being requested at various levels within the organisation: individual, group, project and organisation.

Following Tzelepis and Stephanides (2006), the main functionalities of the *TMU* are defined as follows:

a the identification of training goals by analysing individual, project and business needs, available competencies and contextual conditions when the training occurs, *i.e.*, reactive (both cases) or proactive approach

b the selection of learning objects and a learning strategy (this process also includes injecting the *PA*(s) of trained personnel with interface modules and thus involves the *IA* and its functionalities), and defining initial training data

c monitoring the training process and management of learning outcomes, which involves updating *Human Resource Profiles* and removing unnecessary interface modules.

Note that function (a) requires interaction with the unit responsible for deciding the actual assignment of the training task (responsibility of the *PM* or other higher-level authority) and with the *CMU* to evaluate the gap between the existing and required knowledge. Function (b) requires interaction with the *RPU* in case a suitable learning object could not be located at the level of the *TMU*. Function (c) involves the *PA*(s) of trained personnel.

Obviously, the work of the *TMU* involves interaction with the actual training units (the structure and functioning of which are out of the scope of this paper). However, we can specify that the role of the *TMU* is to provide input specifying:

- who needs training
- which area needs to be trained
- what training method should be applied
- when training should take place.

The output of the training unit is the certification of the completed training and an assessment of trainee(s), which will be sent to the *CMU*, and to appropriate *PA*s to update profile(s) of trainee(s).

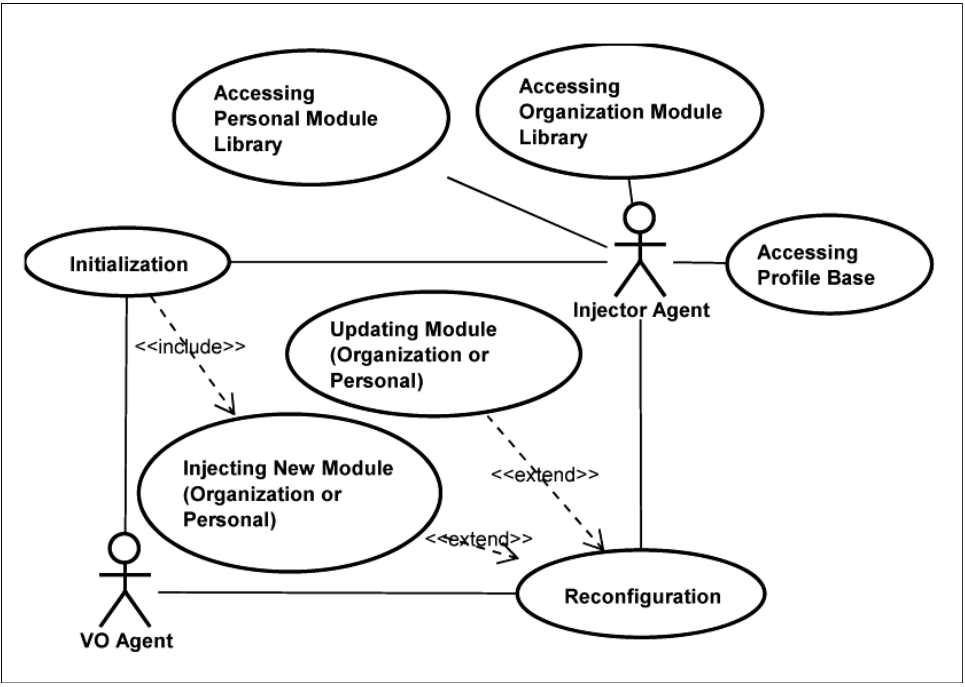
4 Configuring generic agents

Outlining the processes involved in human resource adaptability showed an important role to be played by agent adaptability when new modules have to be added to, or removed from, the *PA*. Let us thus direct our attention to this aspect of adaptability in our system.

4.1 Overview of agent adaptability

We start by presenting the use case diagram of the processes involved in (re)configuring agents (see Figure 4, which should be looked at together with Figure 2). Before we proceed, let us note that our approach follows the ideas put forward by Tu and collaborators in the project *DynamiCS* (Tu *et al.*, 1999). For instance, Tu *et al.* (1999) discussed how e-commerce agents that are to participate in various forms of negotiations can be dynamically assembled from separate modules (communication module, protocol module and strategy module). While the technical details of our approach differ, we directly follow the same general approach of dynamically assembling agents and adapting their behaviour by reconfiguring the set of modules that a given agent consists of.

Figure 4 Functionality of the Injector Agent


Accessing Personal Module Library
Accessing Organization Module Library
Initialization
Accessing Profile Base
Injector Agent
<<include>>
Updating Module (Organization or Personal)
Injecting New Module (Organization or Personal)
<<extend>>
<<extend>>
Reconfiguration
VO Agent

In Figure 4 we can see the *Initialisation* process through which the generic (skeleton) *VOAgent* is created. In this way *any* agent in the organisation is instantiated (a future *PA* supporting an *Employee*, or an autonomous agent, *e.g.*, a *TMA*) as a skeleton which has no 'knowledge' and/or behaviours associated with it. In the case of Jade agents (Jade, 2008), which is our platform of choice, this can be viewed as the simplest instantiation of the *jade.core. Agent* class. The *VOAgent* is extended with components and behaviours supporting its interactions with the *Injector Agent.* These interactions include adding modules, updating modules, removing modules and the agent's knowledge management. The set of behaviours supporting this functionality is called the *Injection Interface.*

This skeleton agent is then 'operated on' by the *Injector Agent*, which has access to *Module Factories* and a *Profile Base.* The *Module Factories* perform the following functions: First, they create module(s) that facilitate the *core functions* of all (*User*-supporting) agents, as well as their extended functionalities. For instance, the calendar-managing module(s) is(are) most likely to be associated with all *PA*s (all workers can be expected to perform certain functions within certain deadlines), while modules supporting intelligent internet search will not be necessary for janitors and waiters in a restaurant (who do not have any reason to search for data on the internet) and thus will be provided only to selected employees in support of their role as *Worker.* Second, *Module Factories* create modules related to specific roles supported by the agent (*e.g.*, in the case of the role of a *QoS* team member, modules that allow the *User* who is supported by his/her *PA* to correctly apply testing procedures to a specific task). Finally, they instantiate the modules necessary for the functioning of autonomous agents (*e.g.*, the *TMA*).

The journal uses lowercase "i" for "internet". Thus we have retained the current appearance.

The *Profile Base* contains information about all the profiles (associated with all the roles identified within the organisation) and is used to appropriately select modules to be added to the *VOAgent*; *e.g.*, for a *PA* a complete list of core modules and personal profile(s) that have to be combined to assemble such an agent for a given *User.*

We can also observe that the *Injector Agent* takes part not only in agent *initialisation*, but also in agent *reconfiguration*, while reconfiguration (agent functionality adaptation) can take three forms:

1 adding a new module

2 removing a module

3 updating (replacing) a module.

Here, let us note that knowledge can be passed to the agent not only while loading modules. In the case of changing some data in the *Data Model* (see, below), with this data being used by some modules, it is possible to update only the knowledge of an agent. It is a special form of agent reconfiguration.

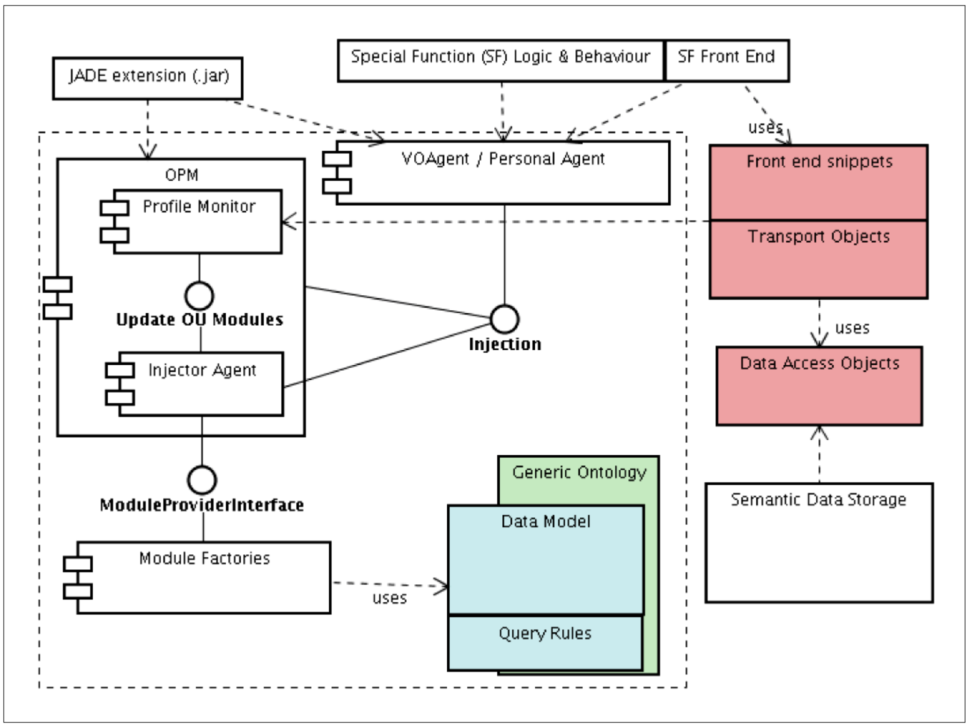
As an example, imagine an *Employee* who is a *Researcher* (which is a specific instantiation of the role of *Worker*). His *PA* will have to be loaded with modules that allow it to support him in fulfilling this role; thus let us call the resulting agent a *Researcher Agent.* The organisational profile of the *Employee* contains information about unit(s) in the organisation to which he belongs (*e.g.*, the *Nanotechnology Unit*, see Szymczak *et al.* (2008)). Knowledge about the modules required for an agent supporting a *Researcher* is stored in the *Profile Base* and is extracted by the *Injector Agent.* Therefore, when a new *PA* is assembled from a *VOAgent, Researcher Modules* (*e.g.*, modules that interface with *Grant Announcement* and *Duty Trip Support* functionalities; Frąckowiak *et al.*, 2008) will be injected into the 'clean' *PA*, thus extending its role. However, when the *Employee* moves from a different department, modules will be added, removed and/or replaced within an existing *PA* (a case of agent adaptation). For instance, if the *Researcher* works as the *Division Head*, it would have access to the personal data of other *Researchers* in the *Division.* Such access should no longer be allowed to the *Researcher* who is not a *Division Head*, and thus the modules supporting it should be removed from his *PA*. To envision an instance of a module, consider the fact that one of the tasks of a *Division Director* is to approve the *Duty Trips* of *Division Employees.* Therefore, one of the *Division Director Modules* allows it to perform this task. In such a module, information is 'stored' as a set of agent behaviours. Note that this example assumes that an infrastructure for data/profile change notification is utilised in the system. However, we do not intend to discuss this issue, as it is out of the scope of this contribution.

Since this description has been presented at a rather high level of abstraction, let us now look in more detail at how these processes can be realised in practice.

4.2 Details of agent adaptability

To discuss how agent creation and adaptation is achieved, we have conceptualised it in the form of a component diagram in Figure 5. This diagram combines the generic framework and system artefacts which are specific to the organisation in which the system is run. In the context of this paper, we are particularly interested in what is happening within the dashed-line rectangle, which delineates the core of the proposed approach.

Figure 5 Component diagram (see online version for colours)



Let us start our description by noting that the *OPM* is actually an umbrella role that is fulfilled by a number of entities (some of them by agents alone, while some involve *Employee*(s) supported by their *PA*(s)). In Ganzha *et al.* (2007a), we argued that travel-recommending functions belong to the *OPM.* Similarly, searching the organisation for a C++ coder available between 18 January and 3 November 2009 is also its role (fulfilled by a different (sub)entity than that involved in travel support; see also Szymczak *et al.*, 2007). Here, within the *OPM*, we distinguish two, earlier mentioned, entities directly related to the support of agent adaptability. First, the *Injector Agent* (*IA*), which is responsible for assembling an agent. The *VOAgent* is modified (through the Injection Interface) by the *IA* in the case of agent initialisation. This modification can result in the creation of a *PA* or an autonomous agent (*e.g.*, a *QoS* agent in an organisation in which the *QoS* role is fulfilled by an agent alone; see also Szymczak *et al.*, 2008). In Figure 5 the *VOAgent* is represented after it has been transformed into the *PA*, but the same process applies to autonomous agents. The *PA* is extended (with functionalities selected according to the specific profile) to allow it to support its *User* in fulfilling a given role. Together with the *IA*, we also see the *Profile Monitor Agent* (*PMA*). The role of the *PMA* is to monitor changes in the data model and to inform the *IA* that a particular profile was updated (this is pertinent to both *User*-supporting and autonomous agents). The *IA* communicates also with the *Module Provider Interface,* which associates modules with module factories (stored in *Module Factories)* and creates instances of modules for the requested resource (*e.g.*, the *Employee* fulfilling a given role).

In the figure, we also represent the *Generic Data Model* and the *Generic Query Model*, which are ontologies that define universal concepts for any organisation in which we might wish to implement the proposed system. These concepts include human resources, nonhuman resources, profiles, profile access privileges, organisation units, module configurations, tasks, matching types and matching relations (see also Szymczak *et al.*, 2008). Both these generic ontologies can be reused and specified by organisation-specific data and query models. They are also used to generate classes that implement the behaviours of specific modules.

Let us stress that we view all entities and their relations represented within the dashed rectangle as a *generic framework* that will materialise in most organisations.

Considering the organisation-specific elements of the system (elements that will differ between organisations and are represented outside of the generic framework), crucial roles are played by the *Organisation-Specific Data Model* and the *Organisation Specific Query Model.* Both these ontologies reuse the *Generic Ontology*, which is a part of the framework, in order to represent data structures and matching scenarios which are pertinent to the organisation. Based on the organisation-specific ontologies, their instances can be created, stored and queried through the *Semantic Data Storage*, which is an infrastructure for manipulating and storing semantically demarcated data. For the time being, to support these functionalities, we intend to utilise the Jena (2008) persistence layer. However, we are well aware of the fact that currently existing semantic data storage and querying software is far from being efficient. As a result, in the future we may select a different persistence technology.

Finally, *Special Function-related* 'boxes' represent specific applications that the system is to deal with. Examples of such functions would be the *Duty Trip Support* (see Szymczak *et al.*, 2008) and the *Grant Announcement* (see, Frąckowiak *et al.*, 2008). Both these functions involve interactions between the *OPM* and the *PA*.

Modules, when approached from the low end of the software stack, are Java objects, which are composed of a map of objects which represent the knowledge part of modules, and of agent behaviour descriptions that contain the info about behaviours that support a module-specific communication model (or models) and functionality (functionalities). There is one universal module class. The instance of this class should be seen more like a description of things that should be loaded in order to support the agent in a specific role. The object of a module class is created by a factory, which is another dedicated-module class. The instance of this module factory class is connected with the organisation data model. As we mentioned before, all factories are stored in the *Module Factories.* Their interface should expose the method which creates a module instance for every user.

Finally, as described above, each module instance might be composed of the list of different knowledge objects and behaviour descriptions. Currently, we assume that each module will be an implementation of the universal module class. The object of this class can be easily loaded by an agent. As a result the agent will load all behaviours that allow it to become an autonomous agent, or a *PA*, or to support the user in a specific role.

Module functionalities may require accessing organisation-specific data which is stored in the *Organisation Semantic Storage.* Hence, the API for accessing semantically demarcated data is required. Let us note that methods of accessing data are not the key issue here. For instance, this can be done in various ways through an appropriately created *Data Source Gateway Agent.* Currently, we assume that any system unit responsible for connecting with the *Semantic Storage* will be utilising *Jastor*-based (2008) *Data Access Objects* designed to be the system API for accessing data in the *Semantic Storage.* Therefore, we can focus our attention on the infrastructure which allows agents to communicate 'about' organisation-specific data. To this effect we plan to utilise *Transport Objects* as a medium for communicating data (including, but not limited to, requesting and returning data). *Transport Objects* (*TO*s) are *Plain Old Java Objects* (POJO, 2008) which represent data stored in the *Semantic Storage* and can be passed in ACL messages between system agents. Developers who will prepare agent modules can use such *Transport Object* classes as an API.

To briefly illustrate processes that are based on the relations depicted in Figure 5, let us assume that there is a *Seller* role in a given organisation. This role involves two behaviours. The first behaviour is responsible for sending offers to customers, while the second listens to their responses and drafts the initial proposals. Both behaviours use a common set of data, which is a list of customers. Now, imagine that we want to create an agent playing the role of a *Seller Agent.* Our task is to initialise the *VOAgent*, add *core modules* to it to turn it into the *PA* and then inject it with the *Seller Module(s)* (this could be a single module or a collection of modules – some of which are shared among multiple roles). In order to inject a module, we have to prepare it first. The *IA* obtains the name of the module factory to provide the *VOAgent* with modules that can extend it to become a *PA.* When the *PA* is created, the *Injector Agent* accesses the *Profile Library* and obtains information about the role(s) of a given *Employee* which is(are) to be supported by its *PA,* as well as a list of modules that have to be associated with each of its roles. Next it contacts the *Module Provider Interface* and obtains a list of classes implementing particular module factories. These factories allow the *IA* to create module instances for specific roles. In our example the module factory will set the name and the version of the module object, the list of *Employee* clients (retrieved from the *Data Model* specifically for the given *Employee)* and also add descriptions of behaviours (sending offers and collecting answers) to the module object. Descriptions of behaviours contain information about behaviours' classes and about additional (third party) libraries which should be added to the agent classpath. These Java objects can then be injected by the *PA*, turning it into a *Seller Agent.*

Now, let us use a different example, and observe what happens when the *Worker of an Implementation Team* (see Figure 1) is promoted to become a *Project Manager* and her *PA* has to be modified to support her in the new role. As a result of the promotion, the organisation profile of the *Employee* (the *Human Resource Profile*; see Szymczak *et al.*, 2008) is adjusted. This information becomes known to the *Profile Monitoring Agent*, which in turn informs the *IA*. The *IA* accesses the *Profile Library* and obtains information on the collection of modules that should constitute the *PA* that can support the *User* in the role of *Project Manager*. Next, it contacts the *Module Provider Interface* to obtain information which classes the factory will create modules that need to be injected/replaced in the *PA* (this list may also include classes that have to be removed). On the basis of the obtained list, the *IA* modifies the *PA*. Observe that, taking into account our current assessment of the capabilities of the Jade platform, we tend to believe that the simplest approach to implement this process would be to instantiate a completely new *PA*, with all the necessary modules injected and then to remove the old *PA*. However, note that, in general, replacing modules in a 'working system' is a rather complicated issue as it involves, among others, dealing with behaviours that are 'in progress' and the fact that exchanging even a single module is likely to concern a number of agents spread across the system. Therefore, we will investigate this issue further.

Thus far we have concentrated our attention on the situation when the change concerns a single *PA* that has to be adapted to support its *User* in a new role. A different scenario takes place when change occurs within the organisation. For instance, let us assume that a new post of a *VP for Institutional Advancement* is created changing a number of interdependencies between individuals and organisational units. These changes materialise in the ontology of the organisation and are propagated across appropriate classes, behaviours, modules and profiles. The *PMA* catches information about these changes and informs the *IA*. As a result, the *IA* has to perform *all* the necessary updates (of all affected agents). Note that, as indicated above, the *OPM* (and thus the *PMA*) has knowledge of all resources in the organisation. Thus it has access to the profiles of all agents (including all *PA*s). This being the case, it is capable of providing the *IA* with a complete list of agents that need to be modified, and even a list of specific modifications. However, this approach puts a heavy burden on the *OPM.* The other possibility of adapting multiple profiles is that the *PMA* prepares a list of affected modules and the *IA* broadcasts this to agents in the *VO* and asks these agents that require change(s) to identify themselves. Here the burden is put on the communication infrastructure. We will investigate further the efficiency of various possible means of implementing multi-agent adaptability.

4.3 Example of agent adaptability

Let us now consider an extended example that will allow us to see how the proposed approach will work in somewhat more realistic settings. Let us assume that the system is implemented in an *East Asia Science Institute* and an employee of that institute, Dr. Jackie Sang, got promoted from *Researcher* to *Division Head Officer.* As suggested above, as a result, not only does his profile change, but also the range of duties and competences. Obviously, the initial profile change involves some human action (someone issues a document specifying that Dr. Sang got promoted and this document is then sent to the *Human Resources* of the institute to be processed). However, we assume that as soon as the decision to promote Dr. Sang is inserted into the computer system of the institute, the remaining processes should be completed autonomously by the infrastructure we are developing. It should be obvious that the *PA* of Dr. Sang has to be updated to support his actions as the *Head of the Division.* For example, such update may involve adding capabilities to preview, and accept or reject the *Duty Trip* applications of division employees (see Ganzha *et al.*, 2007a).

The style followed is to lowercase the initial letter if it is not a proper noun. Thus we have retained lowercase "i" for "institute".

Obviously, regardless of his current position in the institute, Dr. Sang has to be represented within the system as a human resource (Szymczak *et al.*, 2008). In Figure 6, we show a snippet of his organisational profile, which specifies his position in the organisation.

As we can see, Dr. Jackie Sang is a member of the *Food Sciences Division* and is one of the *Division Head Officers* in the institute. Since he is a *Division Head Officer* and a member of the *Food Sciences Division*, it follows that he is a *Head* of that *Division.* Here, obviously we implicitly assume a certain model of the organisation, which is explicitly elaborated in its ontology. Knowing Dr. Sang's new role in the organisation (*Division Head*), the *IA* can infuse modules which allow the *PA* of Dr. Sang to perform certain actions required by his new role. The listing in Figure 6 presents a sample of a configuration of an organisation unit module assignment and module factories class localisation.

Specifically, we can learn here that all the *PA*s of the members of the *Food Sciences Division* organisation unit are infused with modules identified as *:AM_Apply-ForDutyTrip*, *:AM_SubmitDTReport* and *:AM_ViewInterestingGAs.* Analogically, *PA*s of *Users* who play the role of *Head Officers* are infused with the following modules: *:AM_ManageDTApplications*, *:AM_ValidateDTReports* and *:AM_FilterDTReports.* Also, in the sample of the configuration, the following *OPM* modules are listed: *:AM_GA_Support* and *:AM_DT_Support.* Modules *:AM_ApplyForDutyTrip*, *:AM_SubmitDTReport*, *:AM_ManageDTApplications*, *:AM_ValidateDTReports* and *:AM_FilterDTReports* are all examples of the *DT PA Module* in the component diagram, while the *:AM_ViewInterestingGAs* module is an example of the *GA PA Module.* Finally, the *:AM_DT_Support* and the *:AM_GA_Support* are examples of the *DT OPM Module* and the *GA OPM Module.*

Figure 6 Sample human resource profile

```
:JS_HumanResource a vo_onto:HumanResource ;
      vo_onto:hasProfile :JS_HRProfile ,
                             JS_OrgProfile.

:JS_HRProfile a vo_onto:HRProfile;
         vo_onto:belongsTo :JS_HumanResource;
         vo_onto:name ''Jackie Sang''^^xsd:string.

:JS_OrgProfile a vo_onto:OrganizationProfile;
         vo_onto:belongsTo :JS_HumanResource;
         vo_onto:isInOrgUnits
             :IST_DivisionMember_FoodSciences ,
             :IST_DivisionHeadOfficers.
:IST_DivisionMember_FoodSciences a vo_onto:OrganizationUnit ;
  vo_onto:assignedModules :AM_ApplyForDutyTrip,
         :AM_SubmitDTReport, :AM_ViewInterestingGAs.
         :IST_DivisionHeadOfficers a vo_onto:OrganizationUnit ;
  vo_onto:assignedModules :AM_ManageDTApplications ,
         :AM_FilterDTReports , :AM_ValidateDTReports.
:IST_DT_OPM a a vo_onto:OrganizationUnit ;
  vo_onto:assignedModules :AM_DT_Support, :AM_GA_Support.
:AM_ApplyForDutyTrip a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
           ''org.ist.apip.modules.dt.Appication''^^xsd:string.
:AM_SubmitDTReport a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
           ''org.ist.apip.modules.dt.ReportSubmission''^^xsd:string.
:AM_ManageDTApplications a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.dt.AppicationManagement''^^xsd:string.
:AM_FilterDTReports a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.dt.ReportFiltering''^^xsd:string.
:AM_ValidateDTReports a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.dt.ReportValidation''^^xsd:string.
:AM_ViewInterestingGAs a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.ga.ViewAnnouncements''^^xsd:string.
:AM_DT_Support a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.dt.OpmDtSupport''^^xsd:string.
:AM_GA_Support a vo_onto:AgentModule;
  vo_onto:moduleFactoryLocation
         ''org.ist.apip.modules.ga.OpmGaSupport''^^xsd:string.
```

Again, these modules are instances of an organisation-specific Java class composed of properties and behaviours which support certain functionalities. For instance, the *:AM_ApplyForDutyTrip* module includes behaviours which enable one to post a *Duty Trip* application. Naturally, it may also cache previous *Duty Trip Reports*, some user-specific configuration of this module or some draft applications. On the other hand, the *:AM_ManageDTApplications* module consists of behaviours which allow one to query the semantic storage for all open *Duty Trip Applications* and allow *Division Head Officers* to reject or accept duty trip applications. The *:AM_SubmitDTReport* module is designed to provide *Division Workers* with a functionality which enables them to edit, submit and modify reports of their duty trips. *Division Head Officers* are able to browse and validate these reports utilising implementations of the *:AM_FilterDTReport* and the *:AM_ValidateDTReport* modules. For a system which requires employees to be provided with recent information about interesting grant opportunities (see Frąckowiak *et al.*, 2008), the module identified as *:AM_ViewInterestingGAs* was designed. It allows employees to be informed about grant announcements which may be of interest to them. The localisation of each module factory is described in the module definition. For instance, the factory class of the *:AM_ApplyForDutyTrip* module is named *tripstorg.ist.apip.modules.dt.Application.*

After the organisation profile of Dr. Jackie Sang (his profile as a human resource) is updated due to the fact that he was promoted, the *Profile Monitor Agent*, which is aware of all changes that take place in the data model, informs the *Injector Agent* that that particular profile was updated, and the modules of Dr. Sang's *PA* have to be updated with modules specific to the *IST_DivisionHeadOfficers* organisation unit.

An important functionality of the *IA* is to recognise modules which are necessary for supporting particular roles (of members of a particular organisation unit) and locating these module factories in Java libraries. In our example, after all necessary modules are located, a list of appropriate Java class names is retrieved by the *IA*. Next, it infuses the appropriate *PA* with the modules created with the use of module factories. After the update procedure (initialised by inserting information into Dr. Sang's profile) is completed, his *PA* provides not only *Duty Trip Support* functions allowed for *Division Researchers*, which are realised by modules *:AM_ApplyForDutyTrip* and *:AM_SubmitDTReport*, and grant announcement functions for *Division Researchers* delivered by the *:AM_ViewInterestingGAs* module. In addition, his *PA* is now capable of performing duty trip support actions that are reserved for *Division Head Officers*, which are realised by the modules *:AM_ManageDTApplications* and *:AM_FilterDTReports.* Managing access rights to *Duty Trip Applications* and *Duty Trip Reports* (also resources with their own profiles; see Szymczak *et al.*, 2008) will be realised through their profile privileges.

As mentioned before, change in profile may result in the removal of some unnecessary modules. Imagine the situation in which, before being promoted, Dr. Sang was also a member of the *Crops_Research_Division.* After promotion, his profile has changed and he is no longer working in the *Crops_ Research_ Division.* That means that his *PA* should no longer include modules required by the *Crops_Research_ Division.* The *IA* informs Dr. Sang's *PA* that he has to remove modules which 'belong to' members of *Crops_Research_Division.* By removing all behaviours connected with given module, the agent stops to support *User* in a given role.

5 Concluding remarks

In this paper we considered the adaptability in an agent-based virtual organisation. Specifically, we concentrated our attention on adapting software agents to the changing structure of the organisation, to the changes in projects carried out by the organisation and to address human resource adaptation. We addressed these issues both on the formal (AML and UML diagrams) and practical levels (specifying how the proposed approach is actually going to be implemented). Finally, we have specified a number of areas that need to be investigated to obtain an efficient implementation of the proposed framework. We will report our progress in addressing these questions and implementing the proposed framework in subsequent papers.

Acknowledgement

Work of G. Frąckowiak, M. Ganzha, M. Gawinecki, M. Paprzycki, M. Szymczak, Y-S. Han and M-W. Park has been supported in part by the KIST-SRI PAS 'Agent Technology for Adaptive Information Provisioning' grant.

References

Bădică, C., Popescu, E., Frąckowiak, G., Ganzha, M., Paprzycki, M., Szymczak, M. and Park, M-W. (2008) 'On human resource adaptability in an agent-based virtual organization', in N.T. Nguyen and R. Katarzyniak (Eds.) *New Challenges in Applied Intelligence Technologies, Studies in Computational Intelligence*, Springer, Vol. 134, pp.111–120.

Biesalski, E. and Abecker, A. (2005) 'Human Resource Management with ontologies', *Wissensmanagement. Professional Knowledge Management, Third Biennial Conference*, Kaiserslautern, Germany, Revised selected papers, LNAI 3782, Springer, pp.499–507.

Bizer, C., Heese, R., Mochol, M., Oldakowski, R., Tolksdorf, R. and Eckstein, R. (2005) 'The impact of semantic web technologies on job recruitment processes', *Proc. International Conference Wirtschaftsinformatik*, Bamberg, Germany.

Cervenka, R. and Trencansky, I. (2007) *AML: The Agent Modelling Language*, Birkhäuser.

Frąckowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M., Szymczak, M., Myon-Woong, P. and Yo-Sub, H. (2008) 'On resource profiling and matching in an agent-based virtual organization', *Proceedings of the 2008 ICAISC Conference*, Springer.

Ganzha, M., Gawinecki, M., Szymczak, M., Frąckowiak, G. and Paprzycki, M. (2008) 'Generic framework for agent adaptability and utilization in a virtual organization – preliminary considerations', *Proc. of the 2008 WEBIST Conference*.

Ganzha, M., Paprzycki, M., Gawinecki, M., Szymczak, M., Frąckowiak, G., Bădică, C., Popescu, E. and Park, M-W. (2007a) 'Adaptive information provisioning in an agent-based virtual organization – preliminary considerations', *Proceedings of the SYNASC'07 Conference*, IEEE CS Press, pp.235–241.

Ganzha, M., Paprzycki, M., Popescu, E., Bădică, C. and Gawinecki, M. (2007b) 'Agent-based adaptive learning provisioning in a virtual organization', *Advances in Intelligent Web Mastering. Proc.AWIC'2007*, *Advances in Soft Computing*, Fontainebleu, France: Springer, Vol. 43, pp.25–40.

HR-XML Consortium (2008) http://www.hr-xml.org/.

Jastor – Typesafe (2008) 'Ontology driven RDF access from java', http://jastor.sourceforge.net/.

Java Agent DEvelopment Framework (Jade) (2008) http://jade.tilab.com/.

Jena (2008) Jena Semantic Web Framework, http://jena.sourceforge.net/.

Maes, P. (1994) 'Agents that reduce work and information overload', *Commun. ACM*, ACM Press, Vol. 37, No. 7, pp.30–40.

Mochol, M., Wache, H. and Nixon, L. (2007) 'Improving the accuracy of job search with semantic techniques', *Business Information Systems*, LNCS 4439, Springer, pp.301–313.

Montaner, M., López, V.P. and de la Rosa, J.L. (2003) 'A taxonomy of recommender agents on the internet', *Artif. Intell. Rev.*, Vol. 19, No. 4, pp.285–330.

Page, I., Jacob, T. and Chern, E. (1993) 'Fast algorithms for distributed resource allocation', *IEEE Transactions on Parallel and Distributed Systems*, February, Vol. 4, No. 2, pp.188–197.

Plain Old Java Object (POJO) (2008) http://www.martinfowler.com/bliki/POJO.html.

Schmidt, A. and Kunzmann, C. (2006) 'Towards a human resource development ontology for combining competence management and technology-enhanced workplace learning', *OTM Workshops (2). On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops*, LNCS 4278, Springer, pp.1078–1087.

Semantic Web (SW) (2008) http://www.w3.org/2001/sw/.

Szymczak, M., Frąckowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M. and Park, M. (2007) 'Resource management in an agent-based virtual organization – introducing a task into the system', *Proceedings of the MaSeB Workshop*, Los Alamitos, CA: IEEE CS Press, pp.458–462.

Szymczak, M., Frąckowiak, G., Gawinecki, M., Ganzha, M., Paprzycki, M., Park, M-W., Han, Y-S. and Sohn, Y-T. (2008) 'Adaptive information provisioning in an agent-based virtual organization – ontologies in the system', in N.T. Nguyen (Ed.) *Proceedings of the AMSTA-KES Conference*, LNAI 4953, Heidelberg, Germany: Springer, pp.271–280.

Training (Wikipedia) (2008) http://en.wikipedia.org/wiki/Training.

Tu, M.T., Griffel, F., Merz, M. and Lamersdorf, W. (1999) 'A plug-in architecture providing dynamic negotiation capabilities for mobile agents', in K. Rothermel and F. Hohl (Eds.) *Proceedings MA'98: Mobile Agents*, LNCS 1477, Springer-Verlag, pp.222–236.

Tzelepis, S. and Stephanides, G. (2006) 'A conceptual model for developing a personalized adaptive elearning system in a business environment', *Current Developments in Technology-Assisted Education*, Formatex Publishing House, Vol. 3, pp.2003–2006.

Wooldridge, M. (2002) *An Introduction to MultiAgent Systems*, John Wiley & Sons.