Adaptability in an Agent-based Virtual Organization—Towards Implementation

G. Frąckowiak[1], M. Ganzha[1], M. Paprzycki[1], M. Szymczak[1], Y.-.S. Han[2], and M-.W. Park[2]

[1] Systems Research Institute Polish Academy of Sciences, Warsaw, Poland, Maria.Ganzha@ibspan.waw.pl

[2] Korea Institute of Science and Technology, Seoul, Korea myon@kist.re.kr

Abstract. Ability of an organization to adapt to change is one of its important features. When a real-world organization is transformed into a virtual one, with a help of software agents and ontologies, it is important to specify how adaptability can be achieved. In our earlier work we have conceptualized, on a general level, adaptability in an agent-based virtual organization. The aim of this chapter is to discuss how agent adaptability can be implemented.

Key words: software agents, virtual organization, agent systems, agent adaptability, ontologies

1 Introduction

Organizational adaptability to various changes is one of important issues in the world of business (see, for instance [13]. In our recent work ([10, 11, 14, 16]) we have argued that emergent software technologies such as software agents [18] and ontologies [2] could be the base of mapping a real-world organization into a virtual one. We have thus proposed a system in which: (i) organizational structure, consists of specific "roles" and interactions between them, and is represented by software agents and their interactions [10]; while (ii) domain knowledge, resource profiles (representing organizational semantics) and resource matching are ontologically represented and operated on using various forms of semantic reasoning [16]. Second, we have argued that as the real-world organization changes, not only its ontology has to be adjusted, but also "mechanisms of interaction" within its agent-based "representation". Obviously, this concerns not only changes in the the organizational structure itself, but also has to materialize as a response to task changes carried out by the organization (not only changes within specific projects, but also changes in the project portfolio), as well as changing *interests*, *needs* and *skills* of employees.

In our earlier work ([6, 9]) we have discussed in general terms processes involved in both human resource and non-human resource adaptability. One of the important issues was the fact that in addition to changes in the ontology of

the organization, software agents that play the key role in supporting workers, have to be adaptable as well. Therefore, the aim of this paper is to extend our earlier results and look in more detail into the question: what will it take for Jade agents (our current platform of choice; [3]) to be adaptable. First is to be able to *generate on demand* agents with needed functionalities to fulfill specified roles. Second is to *modify them* in response to changes in the organization and/or the environment it operates in. To this effect we, first, briefly describe our system. We follow with a discussion how agent adaptability can be actually implemented.

Before proceeding, let us make a few comments. First, note that while our approach to agent adaptability is in part responding to the way that Jade agents operate, results presented here generalize naturally to other FIPA-compliant agent platforms ([1]). Second, work presented here is an extension of results presented in [9, 6]. Third, it is assumed that readers possess basic knowledge about software agents and the way they are implemented in modern agent environments, like Jade ([3]).

2 System overview

The main function of the system under development is to provide users (employees) an infrastructure that will help to fulfill their roles within the organization. Here, the key concepts are utilization of software agents and ontologies. In the proposed system, software agents exist, first, as independent entities, e.g. a *Task Monitoring Agent*, which tracks progress of a specified task, and undertakes appropriate actions in case of any delays. Note that roles that can be fulfilled by software agents alone vary from organization to organization and depend on its specific needs (see, also [14]). Second, every employee has an associated *Personal Agent* (*PA*). This agent has two main functions: (a) it is the interface between the *Employee* and the system (allowing her to utilize all of its functions), and (b) it supports *Employee* in all *roles* that (s)he is to play within the organization. In other words, an agent is integral part of system but also a bridge between the user and the system. It is worthy mentioning, that this notion of a *Personal Agent* follows the general idea put forward by P. Maes [12]. We can easily envision that a "work *PA*" is a part of a "complete *PA*" which supports *User* in all facets of life.

Let us now briefly summarize main features of the proposed system. First, we assume that work carried out within the organization is project-driven (however, the notion of the project is very broad and includes change of a transmission belt in a Ford Mondeo, as well as managing a team of researchers working on a grant-based project). Therefore, it can be stated that all employee activities are focused on tasks leading to completion of a project. After analysis of project-driven real-world organizations, key roles were identified and we represent them in the form of an AML Social Model diagram, in Figure 1.

Here, we can see the general hierarchical management structure that can be applied to almost every standard real-world organization. Structure of the organization consists of *Departments* and *Teams*. Each *Team* has at least one *Team*

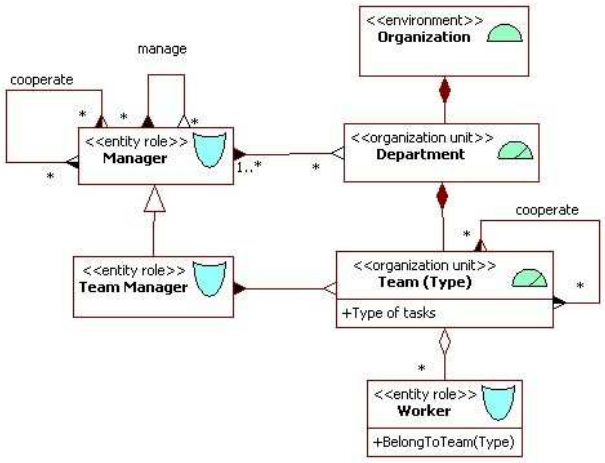

Fig. 1. AML Social Model of an organization

Manager, while each *Manager* may: (1) manage a team, (2) supervise managers of lower level (in this way a recursive hierarchical structure of the organization is represented), or (3) cooperate with other managers on the same level (e.g. when teams collaborate, or when the CFO and the CIO have to collaborate to introduce a new CRM platform to the organization). Note that: (a) *Organization* is an "environment" for *Departments*, *Managers*, *Teams* and *Workers*; (b) *Organization* cannot exist without at least one *Team*; (c) it is possible for a *Team* to consist only of a *Manager*—without any *Workers* (e.g. this could represent the case of self-employment). In Figure 1 we also depict the *Worker* who can be a member of any of the teams (obviously at a given stage (s)he is going to be a member of one team.)

To illustrate how the proposed conceptualization can be instantiated, in Figure 2 we present example of the real-world organization; a University represented also as an AML Social Diagram.

Here, a number of specific entities have been represented. First, we can see the hierarchical and cooperative structure of University management (entity role *Manager*, right top corner of the figure). The *University Worker Team* organizational unit represents all workers of the *University*. Since the University consists of *Departments*, we can see also the *Department Worker Team* organizational unit, which represents all workers of a *Department*. The *Department* consists of a number of teams. We have considered a large *Department* where we can find the *Management Team* (e.g. consisting of Department Chair and Associate Chairs), *Teaching Team* (comprising all Teaching Faculty), *Technical Team* (consisting of IT support personnel as well as laboratory personnel), *Research Team* (consisting of grant-based all post-graduate and graduate associates), and *Assistant Team* (consisting of one or more Secretaries). Finally, we can see a *Worker*, who belongs to one or more teams.

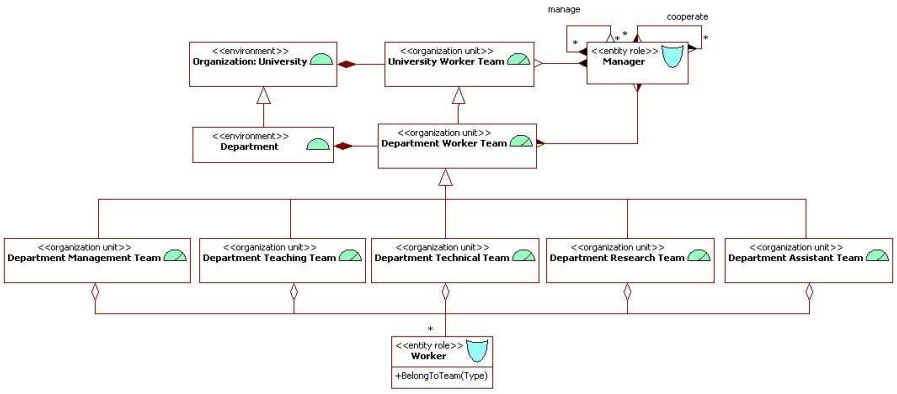

Fig. 2. University; AML Social Model

To complete the picture, in Figure 3 we present an AML Mental Diagram of the *Department*. We present this diagram first, to move from the real-life organization depicted in Figure 2, to the virtual organization, where we talk about specific roles and software agents that support *Employees* in fulfilling them. Second, as it introduces key entities involved in agent adaptability. Finally, as roles identified there will be used in examples across the paper.

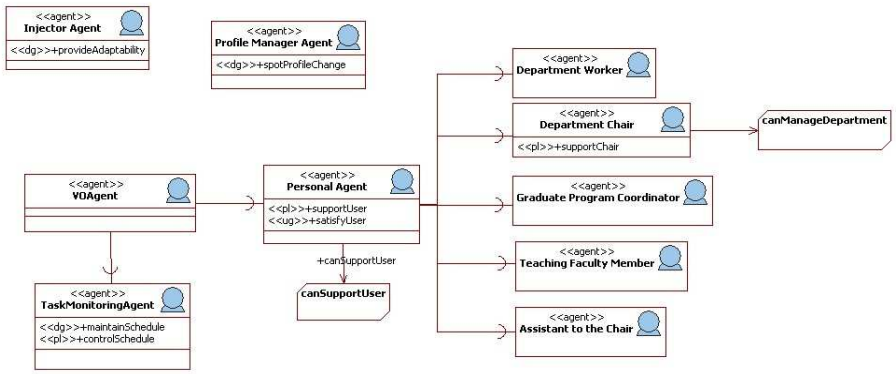

Fig. 3. University Department; AML Mental Diagram

In Figure 3 we introduce the *VOAgent* which is the one of the fundamental concepts of our system. We conceptualize the *VOAgent* as skeleton agent, which can be extended with various functionalities. Those extensions allow the *VOAgent* to support *Employees* in playing specific roles in the organization. The *VOAgent* can be "transformed" into any other agent (see [6] for a similar diagram that presents on a higher/generic level other roles that the *VOAgent* can be

transformed into). Here, let us observe first that the *VOAgent* can be transformed into an, above mentioned, *Task Monitoring Agent* (*TMA*). This agent is an independent entity in our system and does not support any *Employee*. Next, we can see that the *VOAgent* can be transformed into a *Personal Agent* (*PA*). The *Personal Agent* provides the basic support of an *Employee*. Note that the *PA* is not associated with any specific role within an organization. As such, it is a generic role that is associated with every worker in the organization. For instance, every *Employee* of the *University* represented as a member of the *University Worker Team* in Figure 2, would have a *Personal Agent* associated with her/him.

In Figure 3 we have identified a few sample roles that exist in a typical large *University Department*: *Department Worker*—a basic role associated with every worker of the *Department*, *Department Chair*, *Graduate Program Coordinator*, *Teaching Faculty Member*, and *Assistant to the Chair*. Note that in smaller *Universities* some teams identified in Figure 2 may not be present, while some roles introduced here may be played by a single person (e.g. the *Department Chair* who is also a *Graduate Program Coordinator*).

Finally, Figure 3 includes auxiliary agents like *Injector Agent* or *Profile Manager Agent* which play crucial role in agent adaptability and will be described later. With this background we can look into processes involved in extending the *VOAgent* to allow it to play required roles.

3 Configuring Generic Agents

3.1 Overview of agent adaptability

Before we proceed, let us note that our approach to agent adaptability follows ideas of Tuan Tu and collaborators, from their project *DynamiCS*. For instance, in [17] it was discussed how e-commerce agents can be dynamically assembled from separate components (i.e. communication module, protocol module and strategy module) to address the requirements of the e-commerce environment (to be able to participate in unknown in advance form of price negotiations). While technical details of our approach differ, we follow the same general approach of dynamically (re)assembling agents and adapting their behavior by (re)configuring the set of "modules" that a given agent consists of. In this context let us introduce an initial understanding of the notion of a *module*. Let us thus say that a *module* is an object that encapsulates appropriate knowledge and behaviors required for an agent to instantiate a specific functionality. For instance, a *Department Management Module* will group behaviors and knowledge that allow the *Personal Agent* extended by such module to interact with the system and support a member of the *Department Management Team* in completing *Department Management*-related tasks. Specifically, we that such module will contain all necessary knowledge and behaviors to help the *Department Chair* in managing duty trips of *Department Workers* (see, [7] for a detailed description of duty trip support).

To start discussion of agent adaptability, in Figure 3.1, we present the use case diagram of processes involved in (re)configuring agents. This Figure should be looked into together with Figure 3.

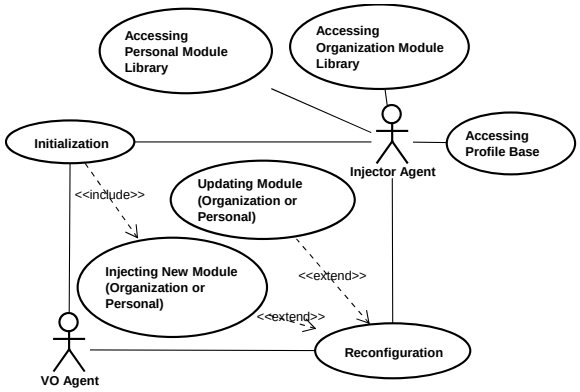

Fig. 4. Functionality of the *Injector Agent*—use case diagram

Here, we can see high-level conceptualization of agent initialization and reconfiguration. Note, that almost every agent in the system (besides some auxiliary agents like the *Injector Agent*) is going to be initialized in the same way. First, the *VOAgent* is going to be created. This agent is able to cooperate with the *Injector Agent* in order to load required modules and knowledge. Subsequent stages of agent initialization include providing it with appropriate modules that allow it to extend itself with functionality required to play (a) specific role(s) (the *Injecting New Modules* function). The process of reconfiguration also involves cooperation between the *IA* and the *VOAgent* (the *Updating Module* function). Note that in the case of agent initialization we can assume that such agent will be able to self-load needed modules. As we will see later, this is not the case when already loaded modules have to be modified/updated.

In order to provide the *VOAgent* with the needed modules the *Injector Agent* has access to:

– *Module Factories*—entities containing factories of every module available in the system (see also the component diagram in Figure 5 for more details). This includes (1) factories of core modules (*Personal Module Library*) which are associated with all functions of a *Personal Agent* (e.g. a *Calendar Managing Module*), (2) specific modules (*Organization Module Library*) created in order to support agent in roles identified in the organization (e.g. *Faculty Evaluation Module* provided to support role of the *Department Chair*), and (3) autonomous agent modules (e.g. *Checking Completion of the Task Module* provided for the *Task Monitoring Agent*).

– *Profile Base*, which stores profiles (i.e. lists of required modules) associated with each role identified within the organization. This information is used to select modules required by a *Personal Agent* supporting a *Department Worker* in fulfilling a specific role.

The *Injector Agent* is involved not only in agent initialization but also in agent reconfiguration. Agent reconfiguration takes place in the following situations:

– One or more profiles in the *Profile Base* have changed and as a result some modules must be added to or removed from an agent supporting functionality specified by such profile(s). Adding a module means that a new functionality is added to the agent (e.g. it will be now able to interface with the new Wiki system installed to manage knowledge in the *University*). Removal of a module means that the agent will no longer support some functionalities (e.g. access to an obsolete *University* blackboard system will be removed).
– The organization modifies some procedures and as a result modules are updated. For instance, a new post of *Associate Chair for Departmental Development* is created and thus selected *Department Workers* will have to report to this new *Associate Chair*. As a result *Personal Agents* of these *Workers* (that support them in their roles) have to have modules involved in communication/dependency structure modified. This process involves removal of the old version of the (*Communication Module*) and loading of new one.
– Agent reconfiguration can also take place in situation when only some part of agent knowledge has to be replaced.

As an example, imagine a *Department Worker* who is a *Professor* in *Department of Biology* (which is a specific instantiation of a role of the *Department Worker*). His *Personal Agent* will have to be loaded with modules that allow it to support her in fulfilling this role; let us name the resulting agent a *Professor Agent*. The organizational profile of the *Department Worker* contains information about unit(s) in the organization to which he belongs (e.g. the *Department of Biology*; see, also [16]). Knowledge about modules required for an agent supporting a *Professor* is stored in the *Profile Base* and can be accessed/extracted by the *Injector Agent*. Therefore, when a new *Professor* is hired by the *University*, first a *PA* is assembled by on the basis of a *VOAgent*. This involves loading it with standard *PA* modules; e.g. module that allows access to the *University* intranet. In the second step of the assembly, *Professor Modules* (e.g. modules that interface with the *Grant Announcement* and the *Duty Trip Support* functionalities; see, [8]) are injected into thus created *PA*, extending its role to support the *Department Worker*. However, when the *Professor* "changes its position within the structure of the organization", some modules are likely going to be added, removed and/or replaced within an already existing *PA*; a case of agent adaptation. For instance, if the *Professor* worked as the *Department Chair*, she had access to personal data of other *Department Workers* in her *Department*. Such access should not longer be allowed to the *Professor* who is not a *Department Chair*, and thus modules supporting it should be removed from her

Personal Agent. Note that this example assumed that a specific infrastructure for data/profile change notification exists in the system. However, here we do not intend to discuss this issue, as it is out of scope of this paper.

3.2 General framework of agent adaptability

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

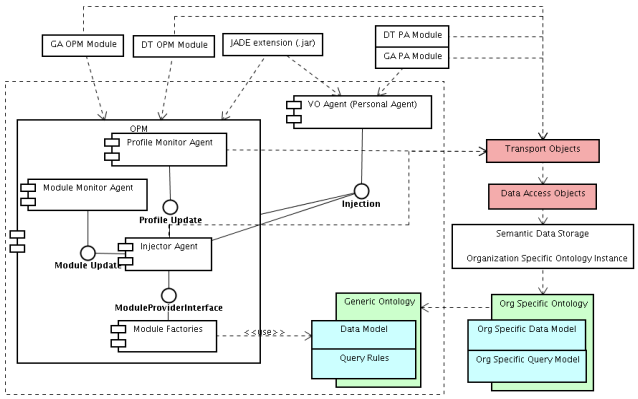

Fig. 5. Component Diagram of agent adaptability

Let us start our description by recalling from [6] that the *OPM* (*Organization Provisioning Manager*) is an umbrella role that is fulfilled by a number of entities (some of them are agents alone, while some of involve *Employee*(s) supported by their *PA*(s)). For instance, in [10] we have argued that travel recommending functions belong to the *OPM*. Similar claim can be made about the *Grant Announcement* application described in [15]. Finally, searching within the *University* for a classroom available during the Spring 2009 semester every Thursday between 2PM and 4PM is also its role (fulfilled by a different (sub)entity within the *OPM*; see, also [14]). Here, we show that agent adaptability, being the case resource management, is also one of the roles of the *OPM*. Therefore, the above described *Injector Agent* (*IA*), and the *Profile Monitor Agent* (*PMA*) are also "a part" of the *OPM*. The role of the *PMA* is to monitor changes in the data model and to inform the *IA* that a particular profile was updated. Finally, the *Module Monitor Agent* informs the *IA* about new modules or new modules versions introduced into the organization. As a result the *IA* has to reconfigure agents that play a roles connected with those modules. Obviously, any form of

(re)configuration is pertinent to both *User*-supporting and autonomous agents, as both of them are created and maintained with the help of the *IA*.

The *IA* communicates also with the *Module Provider Interface*, which associates modules with module factories (stored in the *Module Factories*) and creates instances of modules for the requested resource (e.g. the *Department Worker* fulfilling a given role).

In Figure 5 the *VOAgent* is represented after it has been already transformed into the *PA* (but everything discussed here applies also to cases involving autonomous agents). The *PA* is extended (with functionalities selected according to the specific profile) to support the *Department Worker* in fulfilling a given role. This is achieved by the *IA* through the *Injection Interface*.

In the figure we also represent the *Generic Data Model* and the *Generic Query Model* using ontologies which define concepts universal for any organization in which we could wish to implement the proposed system. These concepts include: human resource, non-human resource, profile, profile access privileges, organization units, module configuration, task, matching types and matching relations (see also [7, 16]). Both these generic ontologies can be reused and specified by organization specific data and query models. They are also used to generate classes that implement behaviors of specific modules.

Let us stress, again, that we view all entities and their relations represented within the dashed rectangle as a *generic framework* that will materialize in most organizations (not only the *University*, which is the focus of this paper).

Considering the organization specific elements of the system (elements that will differ between organizations and are represented outside of the generic framework), crucial roles are played by the *Organization Specific Data Model* and the *Organization 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 organization. Based on the organization 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 utilize the Jena ([4]) 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. Such decision is going to be based mainly on experimental work involving various existing technologies (similar to that described in [5]).

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, [16]) and the *Grant Announcement* (see, [8]). Both these functions involve interactions between the *OPM* and the *Personal Agent*. Note that while these functions have been described in the context of a generic *Research Institute*, they fit very well in the *University*-based example presented here.

3.3 Implementing agent adaptability

Let us now take a closer look at some crucial, from the point of view of implementing agent adaptability, components of our system. Before proceeding let us note that solutions discussed here are on the basis of our current state of knowledge. It is therefore possible that as we proceed with implementation we may find them lacking in important respects and thus in need of adjusting them.

As mentioned above, from implementation point of view the *VOAgent* is an extension of the `jade.core.Agent` class. The extension must be made in order to provide following functionalities:

- working with modules, in particular, adding, replacing, removing, and registering them
- working with behaviors, monitor them, controlling, adding, removing
- providing access to the *Shared Object Map*, which is an map of objects shared by working behaviors

Note that module loading, removing and replacing will involve an additional ontology, which we name the *Module Ontology*. When fully developed this ontology will contain terms like *LOAD_MODULE*, *REMOVE_MODULE*, *SHOW_MODULE_LIST*, *UPDATE_MODULE* and will be utilized directly by JADE agents for agent assembly and modifications.

Now we can also define more precisely the concept of a module. Each module is an instance of a single *universal module class*. This class contains:

- Module name and version.
- List of behavior descriptions that should be loaded in order to support a specific functionality. This list is constant for every agent using a specific module. We assume that modules group exactly the same behaviors. Description should contain all data necessary to load the behavior.
- List of objects that should placed in agent's *Shared Object Map*. This list will differ between agents because data used by behaviors will depend on the specific profile utilized by an agent.

We also predict some other properties needed within modules, which however do not belong to this level of abstraction. Such properties could be: date of module creation, sequence number, signature of module creating entity, additional data necessary for module loading, etc.

As an example imagine an instance of the Module class—a *Department Worker Module*—prepared for a *Biology Department Worker*. The name set for this module is *Department_Worker*, the version(let assume that it is not the first one) is 3.0. The list of behavior descriptions contains only one behavior which allows user to interact with other *Department Workers* (specifically, it allows the *PA* that represents a given *Worker* to interact with *PA*s representing other *Workers*). Of course, the real module will contain also other behaviors. Knowledge part of this module contains name of department which is *Biology* and list of other *Department Workers* (again, it is list of *PA*s representing other *Workers*).

Now, the instance of the *Department Worker Module* prepared for a different *University Employee*, but from other *Department* (let us say *Chemistry*) will be slightly different. The module name and version won't be different. Also the list of behaviors won't change. The difference will be in the knowledge included in this specific instance. The name of department will be *Chemistry* and because it is a different *Department*, the list of *Workers* will also be different.

An important issue which we have to deal with during agent creation or update is to supply it with definitions of new classes e.g. new behaviors classes, new ontologies, etc. Before loading of any knowledge part or behavior (which are instances of some classes) we have to inform agent about localization of all required classes. Therefore, information about all required classes has to be included into agents' classpath. Currently we assume that each module will contain information about localization of all required classes. However, we acknowledge that class loading is somewhat more complex problem requiring further investigation. For instance, it is also possible that class localization will not be included in the module but there will be some action, performed by the *Injection Agent*, preparing agent for module inclusion.

Note also that behaviors included in modules cannot be default Jade behaviors. We presume that in order to provide agent in full behavior monitoring and control function we have to extend them with names and versions. In other words, agents have to be self-aware as to which versions of which specific behaviors they are build out of.

Now let us extend described thus far concepts and discuss somewhat more complicated issues and some real-life examples of their utility.

4 Examples and further considerations

Let as assume that there is a *Department Worker* in the *University* who belongs to *Department Technical Team*. He plays the role of *Technical Support* and his duties include installing software, taking care of hardware problems, preparing auditoriums for lectures, etc. All behaviors supporting this *Department Worker* in fulfilling role of a *Technical Support* will have to be included in his *Personal Agent* in the form of a *Department Technical Support Module*. Functionality of this module will help him with incoming requests, reporting his activities, ordering materials (e.g. toner for printers) from university warehouses, etc. This module consists of behaviors supporting, among others, the above mentioned functions, as well as the necessary data, e.g. list of other members of the *Department Technical Team*. Now, let us imagine that we want to create a *VOAgent* and turn it into an extended *PA*, which supports the *Department Worker* in fulfilling the *Technical Support* role.

To achieve this goal, we have to inject the *PA* it with core modules that support the primary role of a *Department Worker* and, of course, include also the *Technical Support Module*. This module is prepared by the *Technical Support Factory* (an instance of a *Module Factory* from Figure 5). In order to inject necessary modules we have to prepare them first. First, the *Injector Agent* obtains

names of one or more *Module Factories* that will provide the *VOAgent* with modules that extend it to become a *PA*. When the *Personal Agent* is fully assembled, the *IA* accesses the *Profile Library* and obtains information about role(s) of a given *Department Worker* which is(are) to be supported by its *PA*, as well as a list of modules that have to be associated with each of these roles. In our case this is the *Technical Support* role and a list of modules that constitute the complete support for this role. Next, the *IA* contacts the *Module Provider Interface* and obtains the list of classes implementing particular *Module Factories*. These *Factories* allow the *IA* to create instances of modules for (the) specific role(s).

In our example the *Module Factory* will prepare instance of the module class, which contains all data and behaviors required for the given module. As mentioned before, the *Module Factory* will prepare data that includes, among others, the list of other team members (retrieved from the *Data Model* specifically for the given *Department Worker*). The *Module Factory* will also add descriptions of behaviors (e.g. for dealing with requests, interacting with supply department, etc.) to the module object. Currently, we assume that descriptions of behaviors contain information about behavior's classes and about additional (3'rd party) libraries which should be added to the agent classpath. These Java objects can then be self-injected by the *PA*, turning it into *Technical Support Agent*.

Let us use a different example, and observe what happens when the *Department Worker* (see Figure 3) is promoted to become a *Department Chair* and her *PA* has to be modified to support her in the new role. As a result of the promotion, the organization profile of the *Department Worker* (the *Human Resource Profile*; see [16]) is adjusted. This information becomes known to the *Profile Monitoring Agent*, which in turn informs the *IA* about this fact. The *IA* accesses the *Profile Library* and obtains a complete list of modules that should constitute the *PA* that can support the *Department Worker* in the role of *Department Chair*; and contacts the *Module Provider Interface* to obtain information which classes factory will create modules that need to be injected into the *PA*. On the basis of thus obtained list, the *IA* will modify the *PA*.

Let us now focus on another complex issue. Let us consider, again, the *Technical Support Module*, which provides set of behaviors and knowledge that allow the *PA* to support a *Department Worker* in the role of *Technical Support*. Every change in real-life organization procedure(s) must also affect behaviors of the *Technical Support Agent*. Imagine that before an organizational change members of the *Technical Support* were allowed to exchange requests (as long as they were completed in time) without approval of the *Technical Team Manager*. After the change, members of the *Technical Support Team* are not allowed to exchange requests. All exchanges have to be approved by the *Technical Support Team Manager*. This change affects not only the *Technical Support Team Members* but also several other entities including, for instance, the *Technical Support Team Manager* (and thus their appropriate *Personal Agents*). As a matter of fact, every entity, which takes part in this scenario will have to accommodate the new procedure. This requires reconfiguration of agents representing affected entities. New versions of behaviors and modules must be introduced into the

system, and this requires update of appropriate *Module Factories*. New libraries with behavior definitions and module factories have to be stored. Next, the *Injector Agent* must help install new modules with new behaviors to every agent, which role requires using just updated modules.

While injecting new modules is rather easy to achieve (agents can self-inject with additional modules), module updating is a more complex problem. Let us observe that:

- when we introduce new modules we have to be sure that every agent in the system will "instantaneously" start working with the same version of the module; situation in which agents try to communicate with each other while utilizing incompatible procedures/messages/protocols can result in a disaster
- update cannot occur in the middle of a conversation/transaction between any affected agents; as a matter of fact, agents cannot switch behavior version (kill older version and load a new one) if the current one is a part of a still working process.

Combining these two observations makes it easy to see why module update is a very complex issue and may even lead to the need of complete system shutdown. It is only in this case when we can for certain assure that no transaction is in progress and that no agent-version incompatibility will occur. We will investigate this issue in more details, with an attempt at reducing the impact of module updating on the functioning of the system.

5 Concluding Remarks

In this paper we have considered adaptability in an agent-based virtual organization. Specifically, we have concentrated our attention on issues involved in implementation of agent adaptability, while using an example of a *University* to illustrate potential solution and open research questions. We are in the process of implementing the proposed solution and will report on our progress in subsequent publications.

ACKNOWLEDGMENT

Work supported in part by the KIST-SRI PAS "Agent Technology for Adaptive Information Provisioning" grant.

References

1. The foundation of intelligent physical agent (fipa). `http://fipa.org/`.
2. Semantic web. `http://www.w3.org/2001/sw/`.
3. Jade—java agent development framework. TILab, 2008. `http://jade.tilab.com/`.

4. Jena—a semantic framework for java. `http://jena.sourceforge.net`, 2008.
5. K. Chmiel, D. Tomiak, M. Gawinecki, P. Karczmarek, M. Szymczak, and M. Paprzycki. Testing the efficiency of jade agent platform. In *ISPDC '04: Proceedings of the Third International Symposium on Parallel and Distributed Computing/Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC/HeteroPar'04)*, pages 49–56, Washington, DC, USA, 2004. IEEE Computer Society.
6. G. Frąckowiak, M. Ganzha, M. Gawinecki, M. Paprzycki, M. Szymczak, C. Bădică, Y.-S. Han, and M.-W. Park. Adaptability in an agent-based virtual organization. *Internetional Journal Accounting, Auditing and Performance Evaluation*, 2008. in press.
7. G. Frackowiak, M. Ganzha, M. Gawinecki, M. Paprzycki, M. Szymczak, M.-W. Park, and Y.-S. Han. *Considering Resource Management in Agent-Based Virtual Organization*. Studies in Computational Intelligence. Springer, Heidelberg, Germany, 2008. in press.
8. G. Frackowiak, M. Ganzha, M. Gawinecki, M. Paprzycki, M. Szymczak, M.-W. Park, and Y.-S. Han. On resource profiling and matching in an agent-based virtual organization. In *Proceedings of the ICAISC'2008 conference*, LNCS. Springer, 2008. in press.
9. M. Ganzha, M. Gawinecki, M. Szymczak, G. Frackowiak, M. Paprzycki, M.-W. Park, Y.-S. Han, and Y. Sohn. Generic framework for agent adaptability and utilization in a virtual organization—preliminary considerations. In J. Cordeiro et al., editors, *Proceedings of the 2008 WEBIST conference*, pages IS–17–IS–25. INSTICC Press, 2008. to appear.
10. M. Ganzha, M. Paprzycki, M. Gawinecki, M. Szymczak, G. Frackowiak, C. Badica, E. Popescu, and M.-W. Park. Adaptive information provisioning in an agent-based virtual organization—preliminary considerations. In N. Nguyen, editor, *Proceedings of the SYNASC Conference*, volume 4953 of *LNAI*, pages 235–241, Los Alamitos, CA, 2007. IEEE Press.
11. M. Ganzha, M. Paprzycki, E. Popescu, C. Bădică, and M. Gawinecki. Agent-based adaptive learning provisioning in a virtual organization. In *Advances in Intelligent Web Mastering. Proc.AWIC'2007*, number 43 in Advances in Soft Computing, pages 25–40, Fontainebleu, France, 2007. Springer.
12. P. Maes, R. Guttman, and A. Moukas. Agents that buy and sell: Transforming commerce as we know it. 42(3):81–91, 1999.
13. Y. Malhotra. Role of information technology in managing organizational change and organizational interdependence. `http://www.kmbook.com/change/`, 1993.
14. M. Szymczak, G. Frąckowiak, M. Ganzha, M. Gawinecki, M. Paprzycki, and M.-W. Park. Resource management in an agent-based virtual organization—introducing a task into the system. In *Proceedings of the MaSeB Workshop*, pages 458–462, Los Alamitos, CA, 2007. IEEE CS Press.
15. M. Szymczak, G. Frackowiak, M. Ganzha, M. Paprzycki, M.-W. Park, Y.-S. Han, Y. T. Sohn, J. Lee, and J. K. Kim. Infrastructure for ontological resource matching in a virtual organization. In N. Nguyen and R. Katarzyniak, editors, *Proceedings of the IDC Conference*, volume 134 of *Studies in Computational Intelligence*, pages 111–120, Heidelberg, Germany, 2008. Springer.
16. M. Szymczak, G. Frackowiak, M. Gawinecki, M. Ganzha, M. Paprzycki, M.-W. Park, Y.-S. Han, and Y. Sohn. Adaptive information provisioning in an agent-based virtual organization—ontologies in the system. In N. Nguyen, editor, *Proceedings of the AMSTA-KES Conference*, volume 4953 of *LNAI*, pages 271–280, Heidelberg, Germany, 2008. Springer.

17. M. Tu, F. Griffel, M. Merz, and W. Lamersdorf. A plug-in architecture providing dynamic negotiation capabilities for mobile agents. In K. Rothermel and F. Hohl, editors, *Proceedings MA'98: Mobile Agents*, volume 1477 of *LNCS*, pages 222–236. Springer-Verlag, 1999.
18. M. Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, 2002.