GENERIC FRAMEWORK FOR AGENT ADAPTABILITY AND UTILIZATION IN A VIRTUAL ORGANIZATION— PRELIMINARY CONSIDERATIONS

Maria Ganzha, Maciej Gawinecki, Michal Szymczak, Grzegorz Frackowiak
Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
Marcin Paprzycki
Systems Research Institute, Polish Academy of Sciences,
Warsaw Management Academy, Warsaw, Poland
{*Maciej.Gawinecki,Maria.Ganzha,Marcin.Paprzycki*}*@ibspan.waw.pl*

Myon-Woong Park, Yo-Sub Han, Y.T. Sohn
Korea Institute of Science and Technology, Seoul, Korea
myon@kist.re.kr

Keywords: virtual organization, generic adaptability framework, software agents, agent system

Abstract: In our work we consider resource management in a virtual organization. The proposed approach is based on utilization of ontologies to represent structure of the organization and its domain of operations, and software agents that support workers in fulfilling various roles within the organization. In this paper we consider processes involved in a generic *Virtual Organization Agent* taking up / changing / adapting its functions to, a specific role.

1 INTRODUCTION

Currently we are in the process of developing a system with the main goal of supporting resource (information in particular) management in a *Virtual Organization* (*VO*, (Barnatt, 1995; Bleeker, 1998; Dunbar, 2001; Goldman et al., 1995; Warner and Witzel, 2005)). In such organization workers access various *resources* to complete their tasks/projects. Obviously, access to resources should be *adaptive* (change with tasks, and evolve as tasks/projects evolve) and *personalized* (individual workers require access to different resources depending on their roles in the project and/or the organization). For instance, assume that two workers are a part of a team that is to design an implement a knowledge management portal. Obviously, worker who is designing and implementing the back-end part of the system that is based on an Oracle database, needs different resources than her colleague preparing a Web-based front-end for the Internet Explorer. However, a decision to extend the front-end to support also Firefox and Opera browsers will result in the second worker needing additional resources to complete her job (e.g. she may need training modules on working with Firefox and Opera). Separately, if a worker from being a "coder" is moved to undertake a role of a "software tester," she will need different resources to fulfill her new role (e.g. she may need to extend her knowledge about testing tools used in the organization). Furthermore, she will also send different reports to a different supervisor.

In this context, current trends in information management suggest that one of promising approaches to develop a flexible and robust support for resource management in an organization is through utilization of ontologies and semantic reasoning on the one hand (Fensel, 2003), and software agents on the other (Jennings and Wooldridge, 2002). We have decided to accept these assumptions and attempt at developing a generic approach to adaptive resource provisioning in a virtual organization. Obviously, we acknowledge that these assumptions are not without critics, but we believe that our work is also a step toward establishing actual benefits and limitations of the proposed approach. To this effect, in our initial paper ((Ganzha et al., 2007b)) we have analyzed processes involved when a project is introduced into an organization. Next, in (Ganzha et al., 2007a), we have considered roles played within an organization by various entities (humans and agents) identified in (Ganzha et al., 2007b). This allowed us to conceptualize which roles can be played by (a) software agents alone, (b) humans, and (c) by human-agent teams. We have also discussed agent interactions and introduced a sample application, a *Duty Trip Support*, to illustrate how the proposed top level system design can be utilized in practice. Proceeding further, in (Szymczak et al., 2008) we have proposed a top level overview of ontologies to be used in the system (with the key concept of a *generic resource*). Furthermore, we have showed how these basic ontological constructs can be integrated with travel ontologies developed in the *Travel Support System* ((Gawinecki et al., 2005)) to work together within the *Duty Trip Support* application. Finally, in (Frackowiak et al., 2008) we introduced our approach to the way that resource closeness is going to be established (laying ground for semantic reasoning that is to be one of core functionalities of the system).

As already indicated, one of the key ideas behind the proposed system is that each worker is supported by her/his *Personal Agent* (*PA*). It should be obvious that to effectively support their *User*s, their *PA*s have to be adaptable. For instance when Jill is to lead a project (takes up a role of a *Project Manager*, (Ganzha et al., 2007a)) her *Personal Agent* must be able to support her in this new role (e.g. to facilitate flow of documents / reports within the team that is realizing a given project). Furthermore, as the project (or the *VO*) evolves, Jill's duties can change (e.g. project grows considerably and a number of workers and mid-managers are introduced to handle the size of the job). In such situation her *PA* has to adapt itself (or be adapted) to these changes. Obviously, we could assume that each agent is created with a "complete knowledge" to support user in any role, but this would be a terrible waste of resources. Furthermore, this would generate a security nightmare as agents would be potentially capable of performing actions that are not allowed for a given position in an organization (e.g. human resource functions, which are located under the umbrella of an *Organization Provisioning Manager* role, allow access to data that should not be accessible to most workers). Therefore, it is better that a generic *Personal Agent* does not have such capabilities. Finally, note that changes in the organizational structure would still require agent adaptation.

Summarizing what has been said thus far, the aim of this paper is to: (a) specify how an agent that is to fulfill a given role is created, and (b) discuss how agent adaptability can be achieved. To this effect we proceed as follows. We start with an overview of the system (including basic entities and their roles). Next, we present a top level overview of processes involved in creating an agent with a given role and adapting role of an existing agent. We follow with a detailed description of the infrastructure for agent adaptability.

2 PROPOSED SYSTEM

As noted above, it is often claimed that the best approach to resource representation and personalized information delivery is ontological demarcation and semantic reasoning (Fensel, 2003). Furthermore, it can be observed that representation and management of a resource flow in an organization can be achieved as a result of a two-step process. First, roles of members of a real-world organization are specified, and second represented as an agent-based *Virtual Organization*, where each person is represented by her/his *Personal Agent* (*PA*), which can support different roles in different situations. Additionally, auxiliary agents facilitate functioning of the system. Note that this approach is grounded not only in general agent notions (see, for instance, (Jennings and Wooldridge, 2002)), but also in role-oriented agent system development methodologies (e.g. Gaia (Wooldridge et al., 2000), or Prometheus (pro,)). In the latter case the problem space is defined in terms of (1) roles that are to be fulfilled, and (2) interactions between entities playing these roles. Next, each role is fulfilled by a single agent, or is further divided into a number of cooperating agents (see, also, (Jennings, 2001)). As it will be seen, both situations materialize in our system. In Figure 1 we represent high level view of the proposed system through its project-oriented use case diagram.

This use case diagram allows us to discuss briefly what happens when a new project is proposed, and in this context to identify major roles within the organization. Note that in what follows we discuss the use case through *roles* which may be fulfilled by any number of agents and/or humans (unless explicitly stated, entities identified here should *not* be understood as agents). To handle the proposed project, a *Personal Agent* (representing a selected *User*) undertakes a role of a *Project Manager*(*PM*) and orders the *Analysis Manager* (*AM*), to analyze the proposal and create document(s) supporting the decision whether to accept the job or not. If the job is accepted the *PM* creates a *Project Schedule* (based on analysis of available and needed resources; a resource itself). In our work we assume that every *PM* has knowledge about *some* resources in the *VO*. This knowledge is a part of adaptation of a *PA* to the role of the *PM*, while its extent varies and is established within the ontology of the organization. As a result, available resources are reserved (a *Resource Reservation* document is created; a resource itself). If the project requires additional resources (not known, by the *PM*, to be available within an organization) the *PM* contacts the *Organization Provisioning Manager* (*OPM*) and requests them. The *OPM* knows all resources within

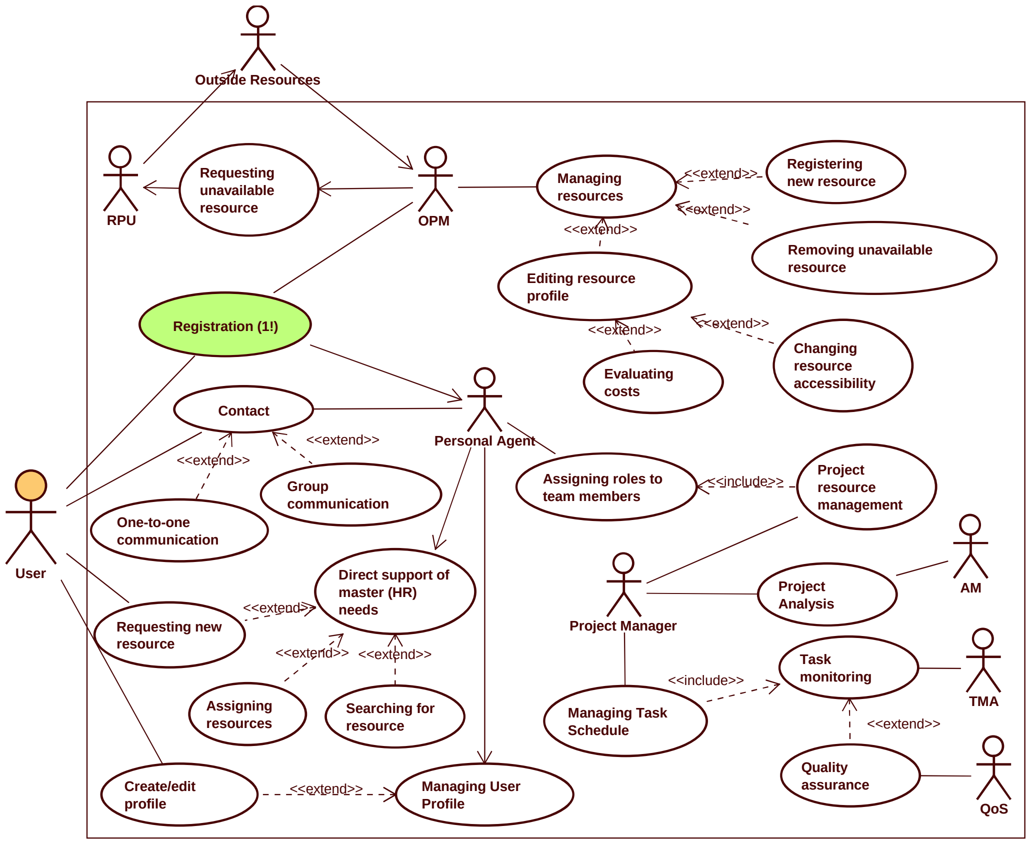

Figure 1: Project-focused use case of the system

the organization and can either find what is needed internally, or request that appropriate resources be found outside of the organization. This latter task is the role of the *Resource Procurement Unit* (*RPU*). In the use case we can see also the *Task Monitoring Agent*, which is associated with each task specified within the *Project Schedule*. Finally, the *Quality of Service* role is responsible for checking quality of each completed task. Let us now extract from the use case presented in Figure 1 the *Personal Agent* and roles it can undertake. This allows us to conceptualize the system from a different perspective, resulting in the use case diagram presented in Figure 2. Here, we focus on the fact that each *User* (worker in the organization) is represented in the system by her/his *Personal Agent* (*PA*). In the simplest case, the *PA* is only providing rudimentary support for the *User* (its role is not *extended*). Note that such rudimentary support could be also understood as support of some (organization specific) *core functionalities* available to all workers within the organization; e.g. meeting

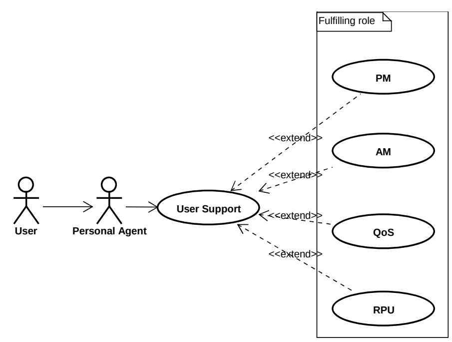

Figure 2: Role-focused use case of the system

scheduling, calendaring, e-mail sorting and filtering, grant announcement reception, etc. (functions similar to these have been described, by P. Maes in (Maes, 1994), as crucial for *Personal Agents*). However, the *PA* has to be also extendable to support other roles (see above and (Ganzha et al., 2007a)) that the *User* has to fulfill. Note that we assume that in most cases the role of the *Task Monitoring Agent* can be fulfilled by a software agent alone (and thus such an extension is not represented in Figure 2), while the remaining roles may require involvement of a human (whether this is the case or not depends on the operation mode of the specific *VO*). Therefore, the generic *PA* needs to be extendedable to support the *User* in fulfilling roles of: *Project Manager*, *Analysis Manager*, *Organization Provisioning Manager*, *Resource Provisioning Unit*, and *Quality of Service*. The aim of the remaining parts of this paper is to present how functionalities of a generic agent can be extended to support *any* role to be fulfilled in an organization.

3 CONFIGURING GENERIC AGENTS

3.1 Overview of agent adaptability

Let us now present the use case diagram of processes involved in (re)configuring agents in the system. In Figure 3 we introduce a new (auxiliary) agent, the *Injector Agent*, responsible for infusing a generic agent created within an organization (a *VO Agent*) with appropriate *modules* that allow it to facilitate the correct set of functionalities. For instance, in the case of creation of a basic *PA* we have to fit it with modules supporting the above described *core functions*, while in the case of a *User* represented by a given *PA* being promoted to a role of a *Project Manager*, with modules supporting that function. Here the notion of a *module* is understood as a set of behaviors and knowledge that support a given functionality. Before we proceed to describe the content of Figure 3 in more detail, let us note that our approach follows ideas behind the proposal put forward by Tuan Tu and collaborators in project *DynamiCS* (see, (dyn,)). For instance in (Tu et al., 1999), Tu and colleagues have 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 technical details of our approach differ, we clearly follow the same general approach of dynamically assembling agents and adapting their behavior by reconfiguring the set of modules that a given agent consists of. In Figure 3 we can see first, the *Initialization* process through which the generic *VO Agent* skeleton is created. In this way *any* agent in the organization is instantiated (a *Personal Agent*, or an auxiliary agent) as an skeleton which has no "knowledge" and/or behaviors associated with it. In the case of Jade agents (Jade,), which is our platform of choice, this can be viewed as the simplest instantiation of the *jade.core.Agent* class. This skeleton agent is then "operated on" by the *Injector Agent*, which has access to (1) an *Organization Module Library*, (2) a *Personal Module Library*, and (3) a *Profile Base*. The *Organization Module Library* stores modules (consisting of knowledge and behaviors) related to specific roles played in an organization (e.g. in the case of the role of *Analysis Manager* modules providing access to financial policies of an organization; these modules allow the *AM* to correctly analyze and predict cost of the proposed project). This library is to contain also all modules necessary for functioning of auxiliary (not related directly to *User* support) agents (e.g. the *Task Monitoring Agent*). The *Personal Module Library* contains modules that facilitate *core functions* of all (*User*-supporting) agents, as well as their extended functionalities. For instance, calendar managing modules are most likely to be associated with all *PA*'s (all workers can be expected to perform certain functions on certain days), while modules supporting intelligent 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). Finally, the *Profile Base* contains information about all profiles (associated with all roles identified within the organization) and is used to appropriately select modules to configure the generic *VO Agent*; e.g. for a basic *PA* a complete list of core modules that have to be combined to assemble such agent.

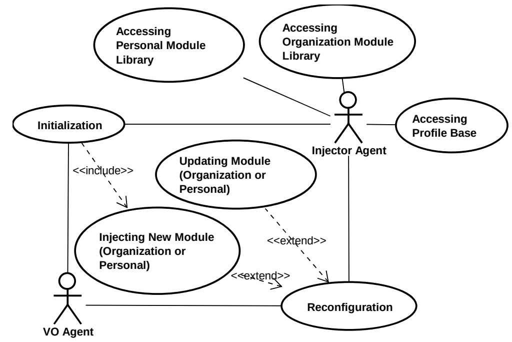

Figure 3: Functionality of the Injector Agent

Here, we can also observe that the *Injector Agent* takes part not only in agent *initialization*, but also in agent *reconfiguration*. Specifically, reconfiguration (agent functionality adaptability) comes in two forms: (a) adding a new module, and (b) updating a module (this involves also complete removal of an obsolete module that is no longer needed). As an example, imagine a worker (*User*) who is a *Seller*. His *Personal Agent* will have to support him in fulfilling this role; thus let us call such agent a *Seller Agent*. The organizational profile of the *User* contains information about units in the organization to which he belongs (including the *Sales Unit*, see (Szymczak et al., 2008)). Knowledge about modules required for an agent supporting a *Seller* is stored in the *Profile Base* and is extracted by the *Injector Agent*. Therefore, upon creation of a new *PA* (or when the worker moves to the *Sales Unit*), *Seller Modules* will be injected into either the skeleton *VO Agent* (agent initialization), or into an existing *PA* (agent adaptation). In the latter case it is also possible that some modules will be removed from the *PA*. Assume, for instance, that the *User* worked in the *Accounting Unit* and had access to some financial data. Obviously such access should not longer be allowed to the *User* who does not work in *Accounting* and thus modules supporting it should be removed from his *PA*. To envision an instance of a module consider the fact that one of tasks of a *Seller* is to report results of his activities to selected entities within the organization (e.g. his direct supervisor). Therefore, one of *Seller Module*s contains information about specific report(s) and their recipients. Within the same, or another module support for report preparation may be included.

Since this description has been presented at a rather high level of abstraction, let us now look into more details as to how these processes can be realized in practice.

3.2 Details 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 4. This diagram combines the generic framework, system artifacts which are specific to the organization in which the system is run, and the specific example of internal functionalities of an organizations, which are realized by entities with names starting with *DT* (for the *Duty Trip Support*; see (Szymczak et al., 2008)) and *GA* (for the *Grant Announcement*; this functionality allows the *PA* to select which grant announcements should be pulled from the repository and presented to its *User*). In the context of this paper we are particularly interested in what is happening within the dash-line rectangle, which delineates the core of the proposed approach.

Let us start our description by noting that, as has been implicitly suggested above, the *OPM* (*Organization Provisioning Manager*) is actually an umbrella role that is fulfilled by a number of entities. In (Ganzha et al., 2007a) we have argued that travel recommending functions belong to the *OPM*. Similarly, searching the organization for a C++ coder available between January 17th and August 23rd is also its role (fulfilled by a different entity than that involved in travel support; see, also (Ganzha et al., 2007b)). Here, within the *OPM* we distinguish two entities directly related to support of agent adaptability. First, the already mentioned *Injector Agent* (*IA*), which is responsible for assembling an agent with appropriate modules that allow it to support its *User*. Together with the *IA* we see also 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. The *IA* communicates also with the *Class Localizer* which has a role of associating modules with behaviors and thus classes that implement them. Note also the *VO Agent* that is modified (the *Injection* relation) by the *IA* in the case of agent initialization (in our system, it is achieved by extending the JADE base agent class: *jade.core.Agent*). However, we can also assume that this agent is already a *Personal Agent* that is to be adapted (also through the *Injection* relation). Finally, we represent the *Generic Data Model* and the *Generic Query Model*, which are 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 (Szymczak et al., 2008)). Both these generic ontologies can be reused and specified by organization specific data models and query models. They are also used to generate classes that implement behaviors of specific modules and that are supplied to the *IA* by the *Class Localizer*. Let us stress here, again, that we view all entities and their relations represented within the dashed rectangle as a *generic framework* that will materialize in any organization.

Considering the organization specific elements of the system (elements that will differ between organizations and are represented outside of the dashed border 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

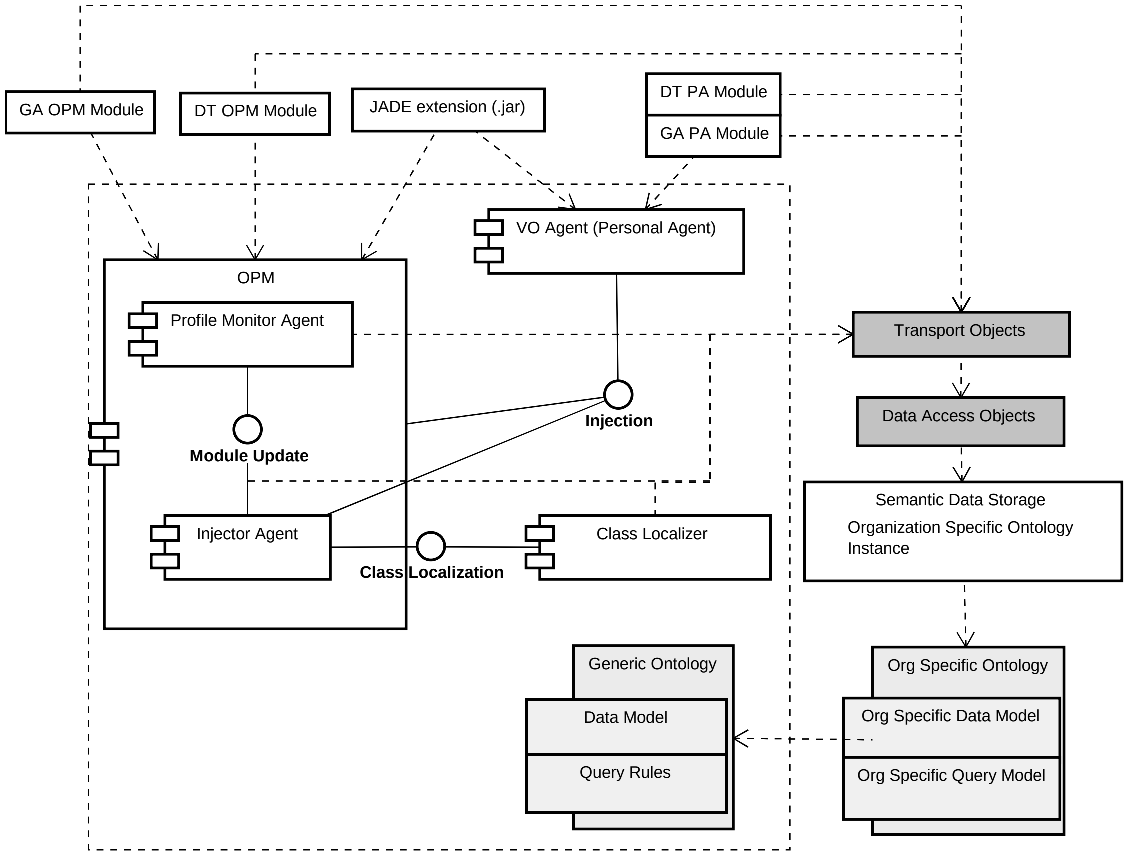

Figure 4: Component Diagram

demarcated data. For the time being we intend to utilize the Jena (Jena—RDF persistency engine,) persistence layer and one of the *PostgreSQL* or *MySQL* database engines, however this part of the system may need to be tuned or redesigned in the future (we are well aware of the fact that currently existing semantic data storage and querying software is far from being efficient). Modules, when approached from the low-end of the software stack, are Java classes, which might be composed of class properties which represent knowledge part of modules, and of JADE agent behaviors which support module-specific communication model and functionalities. These Java classes can be injected into *VO Agent*s.

Module functionalities may require accessing organization specific data which is stored in the *Organization 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, as it can be done in various ways (through a *Gateway Agent*, for instance). Currently, we assume that any system unit responsible for connecting with the *Semantic Storage* will be utilizing *Jastor* based (Jastor—Ontology Driven RDF Access from Java,) *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" organization specific data. To this effect we plan to utilize *Transport Objects* as a medium for communicating data (including, but not limited to requesting and returning data). *TO*s are *Plain Old Java Object* (POJO: Wikipedia, ; POJO,) 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.

Let us now combine what we have just described utilizing the component diagram, with the information depicted in Figure 3, and describe processes involved in agent initialization and modification. Here, we focus our attention on *User*-supporting *Personal*

Agents, but described processes remain the same also for auxiliary agents. When the *VO Agent* is instantiated, the *Injector Agent* accesses the *Profile Library* and obtains information about role(s) of a given *User* 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 *Class Localizer* and obtains a list of classes implementing particular modules. As we described above, each module might be composed of various properties and behaviors. Currently we assume that each module will be implemented as a single class, but it is also possible that module's properties and behaviors could be scattered among more than a single Java class. These classes are then injected by the *IA* into the *VO Agent* skeleton; resulting in a creation of a *Personal Agent*. Now, let us observe what happens when the *User* is moved from one position (Researcher) to another position (Division Head). As a result, the organization profile of the *User* (human resource profile) 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 what is the collection of modules that should constitute the *PA* that can support the *User* in the role of Division Head; and contacts the *Class Localizer* to obtain information which classes need to be injected/replaced in the *PA* (this list may also include classes that have to be removed). On the basis of thus obtained list, the *IA* modifies the *PA*. Observe that, taking into account our current assessment of 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, we will investigate this issue further. Note also that thus far we have considered 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 organization. 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 organizational units. These changes materialize in the ontology of the organization and are propagated across appropriate classes, behaviors, modules and profiles. The *PMA* catches information about these changes and informs the *IA*. As a result the *IA* has to perform *all* necessary updates (of all affected agents). Note that, as indicated above, the *OPM* (and thus the *PMA*) has knowledge of all resources in the organization. Thus it has access to 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 that require change identify themselves. Here the burden is put on the communication infrastructure. We will investigate further the efficiency of various possible means implementing multi-agent adaptability.

3.3 Example

Let us now consider an extended example that will allow us to see how the proposed approach will work in a somewhat more realistic environment. Let us assume that the system is implemented in an East Asia Science Institute and an employee of that institute, dr Ha Yoong Cha, got promoted from the Researcher to the Division Head Officer. As it was specified 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 Cha got promoted and this document is then send to Human Resources of the Institute to be processed). However, we assume that as soon as the decision to promote dr Ha Yoong Cha is inserted into the computer system of the Institute, the remaining processes should be done automatically. As obvious from the above, the *PA* of dr Cha has to be updated to support his actions as the Head of the Division. For example, such update may involve adding capabilities for preview, and acceptance or rejection of all Duty Trip applications of division employees (see also (Ganzha et al., 2007a) for the processes involved in Duty Trip application and approval).

Obviously, regardless of his current position in the Institute, dr Ha Yoong Cha has to be represented within the system as a human resource (Szymczak et al., 2008). In what follows we show a snippet of his organizational profile, which specifies his position in the organization:

```
:JD_HumanResource a vo_onto:HumanResource;
    vo_onto:hasProfile :JD_HRProfile,
                       JD_OrgProfile.

:JD_HRProfile a vo_onto:HRProfile;
      vo_onto:belongsTo :JD_HumanResource;
      vo_onto:name ''Ha Yoong Cha''^^xsd:string.

:JD_OrgProfile a vo_onto:OrganizationProfile;
      vo_onto:belongsTo :JD_HumanResource;
      vo_onto:isInOrgUnits
        :IST_DivisionMember_NanoTechnology,
        :IST_DivisionHeadOfficers.
```

```
:IST_DivisionMember_NanoTechnology 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 .
: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:moduleLocation ``org.ist.apip.modules.dt.Appication''^^xsd:string .
:AM_SubmitDTReport a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.dt.ReportSubmission''^^xsd:string .
:AM_ManageDTApplications a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.dt.AppicationManagement''^^xsd:string .
:AM_FilterDTReports a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.dt.ReportFiltering''^^xsd:string .
:AM_ViewInterestingGAs a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.ga.ViewAnnouncements''^^xsd:string .
:AM_DT_Support a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.dt.OpmDtSupport''^^xsd:string .
:AM_GA_Support a vo_onto:AgentModule ;
        vo_onto:moduleLocation ``org.ist.apip.modules.ga.OpmGaSupport''^^xsd:string .
```

Figure 5: Sample configuration file

As we can see, dr Ha Yoong Cha is a member of the Nano-Technology Division and is one of the Division Head Officers in the Institute. Since he is a Division Head Officer and a member of the Nano Technology Division it follows that he is a Head of that Division. Here, obviously we implicitly assume a certain model of the organization, which is explicitly elaborated in its ontology. Knowing dr Cha's new role in the organization (Division Head) the *IA* can infuse modules which allow the *PA* of dr Cha to perform certain actions. The listing in Figure 5 presents a sample of a configuration of organization unit module assignment and module class localization.

Specifically, we can learn here that all *PA*s of members of the Nano Technology Division organization unit are infused with modules identified as: *AM_ApplyForDutyTrip*, *AM_SubmitDTReport*, *AM_ViewInterestingGAs*. Analogically, *PA*s of *Users* who play roles of Head Officers are infused with the following modules: *AM_ManageDTApplications* and *AM_FilterDTReports*. Also, in the snippet of the configuration, the following *OPM* modules are listed *AM_GA_Support* and *AM_DT_Support*. Again, these modules are organization specific Java classes composed of properties and behaviors which support certain functionalities. For instance, the *AM_ApplyForDutyTrip* module includes behaviors which enable 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 behaviors which allow to query the semantic storage for all open Duty Trip applications and inform Division workers about accepting or rejecting application by Division Head Officers. Localization of each module is described in the module definition. For instance, class of the *AM_ApplyForDutyTrip* module is named *org.ist.apip.modules.dt.Appication*. Modules *AM_ApplyForDutyTrip*, *AM_SubmitDTReport*, *AM_ManageDTApplications* 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*.

After the organization profile of dr. Ha Yoong Cha is updated due to the fact that he was promoted (his profile as a human resource), the *Profile Monitor Agent*, which is aware of all changes that take place in the data model, informs the *Injector Agent* that particular profile was updated and modules of dr. Cha's *PA* have to be updated with modules specific to *IST_DivisionHeadOfficers* organization unit. An important functionality of the *IA* is to recognize modules which are necessary for supporting particular roles (of members of a particular organization unit) and locating these modules in Java libraries. In our example, after all necessary modules are located and list of appropriate Java class names is retrieved by the *IA*, it infuses the appropriate *PA* with the modules located at the *org.ist.apip.modules.dt.AppicationManagement* and the *org.ist.apip.modules.dt.ReportFiltering*.

After the update procedure initialized by inserting information to dr Cha's profile is completed, his *PA* provides not only duty trip support functions allowed for Division Researchers, which are realized 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 reserved for Division Head Officers, which are realized by 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 realized through their profile privileges.

4 CONCLUDING REMARKS

The aim of this paper was to present our approach to agent adaptation within a virtual organization. We have started by summarizing our semantically driven approach to resource management. This allowed us to present use case based formalizations of processes involved in a project being introduced into an organization, and of roles that are to be played by *Personal Agent*s that is to support *User*s/workers. Next we have discussed (and represented in the form of use case and component diagrams) how a generic agent is dynamically assembled to support one of required roles, and how it is further adapted to individual and organizational changes. Across the paper we have identified a number of technical issues that need to be addressed. Currently, we are in the process of implementing the framework depicted in the component diagram and will report on our progress in subsequent reports.

REFERENCES

Dynamics—dynamically configurable software. `http://vsis-www.informatik.uni-hamburg.de/members/info.php/7`.

The prometheus methodology. `http://www.cs.rmit.edu.au/agents/SAC2/methodology.html`.

Barnatt, C. (1995). *Journal of General Management*, 20(4):78–92.

Bleeker, S. (1998). *The Virtual Organization*. Sage Thousand Oaks (CA).

Dunbar, R. (2001). *Virtual Organizing*, pages 6709–6717. Thomson Learning, London.

Fensel, D. (2003). *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*. Springer-Verlag, New York.

Frackowiak, G., Ganzha, M., Gawinecki, M., Paprzycki, M., Szymczak, M., and Park, M.-W. (2008). On resource profiling and matching in an agent-based virtual organization. In *Proceedings of the 2008 ICAISC Conference*. submitted for publication.

Ganzha, M., Paprzycki, M., Gawinecki, M., Szymczak, M., Frackowiak, G., Bădică, C., Popescu, E., and Park, M. (2007a). Adaptive information provisioning in an agent-based virtual organization—preliminary considerations. In *Proceedings of the SYNASC'2007 Conference*, Los Alamitos, CA. IEEE CS Press. in press.

Ganzha, M., Paprzycki, M., Gawinecki, M., Szymczak, M., Frackowiak, G., and Park, M. (2007b). Resource management in an agent-based virtual organization—introducing a task into the system. In *Proceedings of the IAT Conference*, Los Alamitos, CA. IEEE CS Press.

Gawinecki, M., Gordon, M., Nguyen, N. T., Paprzycki, M., and Szymczak, M. (2005). Rdf demarcated resources in an agent based travel support system. In et. al., M. G., editor, *Informatics and Effectiveness of Systems*, pages 303–310, Katowice. PTI Press.

Goldman, S., Nagel, R., and Preiss, K. (1995). pages 219–240. Van Nostrand Reinhold, New York.

Jade. `http://jade.tilab.com/`.

Jastor—Ontology Driven RDF Access from Java. `http://jastor.sourceforge.net/`.

Jena—RDF persistency engine. `http://jena.sourceforge.net/`.

Jennings, N. R. (2001). An agent-based approach for building complex software systems. *Commun. ACM*, 44(4):35–41.

Jennings, N. R. and Wooldridge, M. J., editors (2002). *Agent Technology: Foundations, Applications, and Markets*. Springer.

Maes, P. (1994). Agents that reduce work and information overload. *Commun. ACM*, 37(7):30–40.

POJO. `http://www.martinfowler.com/bliki/POJO.html`.

POJO: Wikipedia. `http://en.wikipedia.org/wiki/POJO`.

Szymczak, M., Frackowiak, G., Gawinecki, M., Ganzha, M., Paprzycki, M., Park, M.-W., Han, Y.-S., and Sohn, Y. (2008). Adaptive information provisioning in an agent-based virtual organization—ontologies in the system. In *Proceedings of the 2008 AMSTA-KES Conference*. to appear.

Tu, M., Griffel, F., Merz, M., and Lamersdorf, W. (1999). A plug-in architecture providing dynamic negotiation capabilities for mobile agents. In Kurt Rothermel, F. H., editor, *Proceedings MA'98: Mobile Agents*, LNCS 1477, pages 222–236. Springer-Verlag.

Warner, M. and Witzel, M. (2005). *Zarzadzanie organizacja wirtualna*. Oficyna Ekonomiczna.

Wooldridge, M., Jennings, N. R., and Kinny, D. (2000). The gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312.