HDS: a Software Framework for the Realization of Pervasive Applications

AGOSTINO POGGI
Dipartimento di Ingegneria dell'Informazione
Università degli Studi di Parma
Viale G.P. Usberti 181A, Parma, 43100
ITALY
agostino.poggi@unipr.it http://www.ce.unipr.it/people/poggi

Abstract: - Nowadays pervasive computing is one of the most active research fields because it promises the creation of environments where computing and communication devices are gracefully integrated with users so that applications can provide largely invisible support for tasks performed by users. This paper presents a software framework, called HDS (Heterogeneous Distributed System), that tries to simplify the realization of pervasive applications by merging the client-server and the peer-to-peer paradigms and by implementing all the interactions among the processes of a system through the exchange of typed messages and the use of composition filters for driving and dynamically adapting the behavior of the system. Typed messages and computational filters are the elements that mainly characterize such a software framework. In fact, typed messages can be considered an object-oriented "implementation" of the types of message defined by an agent communication language and so they are means that make HDS a suitable software framework both for the realization of multi-agent systems and for the reuse of multi-agent model and techniques in non-agent based systems. Composition filters drive and adapt the behavior of a system by acting on the exchange of messages. In fact, on the one hand, composition filters can constrain the exchange of messages (e.g., they can block the sending/reception of some messages to/from some processes), they can modify the flow of messages (e.g., they can redirect some messages to another destination) and they can manipulate messages (e.g., they can encrypt and descript messages). On the other hand, processes can dynamically add and remove some composition filters to adapt the behavior of a system to any hardware and software new configuration and to any new user requirement.

Key-Words: - Typed messages, Composition filters, Software framework, Pervasive systems, Multi-agent systems, Java.

1 Introduction

Nowadays pervasive computing is one of the most active research fields because it promises the creation of environments where computing and communication devices are gracefully integrated with users so that applications can provide largely invisible support for tasks performed by users [20,21].

Several works discussed about the features that make a software infrastructure suitable for the realization of such environments (see, for example, [10,12,14,18]). The list of such features is very long and includes: adaptation, context awareness, distribution, interoperability, invisibility, mobility and scalability.

These features are not independent among them and, in particular, adaptation can be considered a mandatory requirement for the realization of pervasive applications that exhibit the previous features. In fact, adaptation allows to overcome the intrinsically dynamic nature of pervasive environments where users, devices and software components can dynamically enter, leave or move and where users can at any time require support for new tasks [6,9,15,20,22,23].

In this paper, we present a software framework, called HDS (Heterogeneous Distributed System) whose goal is to simplify the realization of distributed and adaptive applications by taking advantage of typed messages and message filters. The next section introduces the HDS software framework. Section three presents the three models that concur to the definition of the architecture of a HDS application. Section four describes typed messages communication support. Section five gives some notes on the implementation of the software framework.. Section six introduces the

reasons for which HDS is suitable for the realization of pervasive applications. Section seven discusses about the experimentation of such a framework. Finally, section eight concludes the paper sketching some future research directions.

2 HDS

HDS (Heterogeneous Distributed System) is a software framework that has the goal of simplifying the realization of distributed applications by merging the client-server and the peer-to-peer paradigms and by implementing all the interactions among all the processes of a system through the exchange of messages.

This software framework allows the realization of systems based on two types of processes: actors and servers. Actors have their own thread of execution and perform tasks interacting, if necessary, with other processes through synchronous and asynchronous messages. Servers perform tasks on request of other processes by composing, if necessary, the services offered by other processes through synchronous messages. Moreover, while both servers and actors may directly take advantage of the services provided by other kinds of application, only the servers can provide services to external applications by simply providing one or more public interfaces.

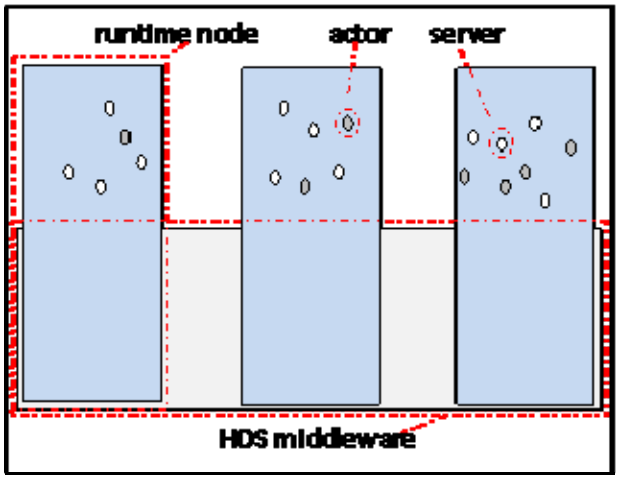

Fig. 1. The architecture of an HDS distributed system.

Actors and servers can be distributed on a (heterogeneous) network of computational nodes (from now called runtime nodes) for the realization of different kinds of application (see Figure 1). In particular, actors and servers are grouped into some runtime nodes that realize a platform. An application can be obtained by combining some preexistent applications by realizing a federation.

3 Model

The software architecture of a HDS application can be described through the three different models:

- the concurrency model, that describes how the processes of a runtime node can interact and share resources.
- the runtime model, that describes the services available for managing the processes of an application.
- the distribution model, that describes how the processes of different runtime nodes can communicate.

3.1 Concurrency Model

The concurrency model is based on seven main elements: process, description, description selector, mailer, message, content and message filter.

A process is a computational unit able to performs one or more tasks taking, if necessary, advantage of the tasks provided by other processes. To facilitate the cooperation among processes, a process can advertize itself making available to the other processes its description. The default information contained in a description is represented by the process identifier and the process type; however, a process may introduce some additional information in its description.

A process can be either an actor or a server. An actor is an active process that can have a proactive behavior and so can start the execution of some tasks without the request of other processes. A server is a passive process that is only able to perform tasks in response of the request of other processes.

A process can interact with the other processes through the exchange of messages based on one of the following three types of communication:

- synchronous communication, the process sends a message to another process and waits for its answer;
- asynchronous communication, the process sends a message to another process, performs some actions and then waits for its answer;
- one-way communication, the process sends a message to another process, but it does not wait for an answer.

In particular, while an actor can start all the three previous types of communication with all the other processes, a server can only respond to the requests

of the other processes in case serving them composing the services provided by other processes through synchronous communications. Moreover, a server can respond to a request through more than one answer (e.g., when it acts as broker in a publisher subscriber system) and can forward a request to another server for its execution.

A process has also the ability of discovering the other processes of the application. In fact, it can both get the identifiers of the other mailers of the systems and check if an identifier is bound to another mailer of the system taking advantage of the registry service provided by HDS middleware.

Moreover, a process can take advantage of some special objects, called description selectors, for requiring the listing of specific subsets of mailer identifiers. In fact, a description selector allows the definition of some constraints on the information maintained by the process descriptions (e.g., the process must be of a specific type, the process identifier must have a specific prefix and the process must be located in a specific runtime node) and the registry service is able to apply their constraints on the information of the registered descriptions for building the required subsets of identifiers.

A process does not exchange directly messages with the other processes, but delegates this duty to a mailer. In fact, a mailer provides a complete management of the messages of a process: it receives messages from the mailers of the other processes, maintains them up to the process requests theirs processing and, finally, sends messages to the mailers of the other processes.

In a similar way to a process, a mailer can be either an actor mailer or a server mailer. Of course, it depends on the fact that, as described above, an actor and a server can assume a different set of roles in message exchanging.

A message contains the typical information used for exchanging data on the net, i.e., some fields representing the header information, and a special object, called content, that contains the data to be exchanged. In particular, the content object is used for defining the semantics of messages (e.g., if the content is an instance of the Ping class, then the message represents a ping request and if the content is an instance of the Result class, then the message contains the result of a previous request).

Normally, a mailer can communicate with all the other mailers and the sending of messages does not involve any operation that is not related to deliver messages to the destination; however, the presence of message filters can modify the normal delivery of messages.

A message filter is a composition filter [5] whose primary scope is to define the constraints on the reception/sending of messages; however, it can also be used for manipulating messages (e.g., their encryption and decryption) and for the implementation of replication and logging services.

Each mailer has two lists of message filters: the ones of the first list, called input message filters, are applied to the input messages and the others, called output message filters, are applied to the output messages (figure 2 shows the flow of the messages from the input message filters to the output message filters). When a new message arrives or must be sent, the message filters of the appropriate list are applied to it in sequence until a message filter fails; therefore, such a message is stored in the input queue or is sent only if all the message filters have success.

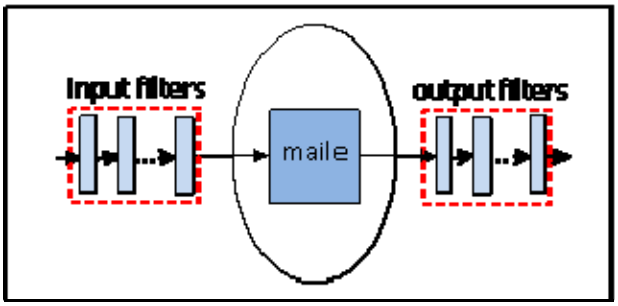

Fig. 2. Flow of the messages from the input to the output message filters.

Message filters are not only used for customizing the reception and the sending of messages, but are also used by the processes for asking their mailer for the input messages they need for completing their current task. In fact, as described above, a message filter allows to define the constraints that are necessary to identify a specific message and a mailer is able to use it for selecting the first message in the input queue that satisfies its constraints (e.g., the reply to a message sent by the process, a message sent by a specific process and a message with a specific kind of content).

3.2 Runtime Model

The runtime model defines the basic services provided by the middleware to the processes of an application. This model is based on four main elements: registry, processer, filterer and porter.

A registry is a runtime service that allows the discovery of the processes of the application. In fact, a registry provides the binding and unbinding of the processes with their identifiers, the listing of the identifiers of the processes and the retrieval of a

special object, called reference, on the basis of the process identifier.

A reference is a proxy of the process that makes transparent the communication respect to the location of the process. Therefore, when a process wants to send a message to another process, it must obtain the reference to the other process and then use it for sending the message.

A processer is a runtime service that has the duty of creating new processes in the local runtime node. Of course, an important side effect of the creation of a process is the creation of the related mailer. The creation is performed on the basis of the qualified name of the class implementing the process, a list initialization parameters.

The lists of message filters cannot be directly modified by the processes, but they can do it taking advantage of a filterer. A filterer is a runtime service that allows the creation and modification of the lists of message filters associated with the processes of the local runtime node. Therefore, a process can use such a service for managing the lists of its message filters, but also for modifying the lists of message filters associated with the other processes of the local runtime node.

Finally, a porter is a runtime service that has the duty of creating some special objects, called ports, that allows an external application to use the services implemented by a server of the local runtime node. In particular, a port is a wrapper that encapsulates a server for limiting the access to the functionalities of the process by masquerading the use of some its services and by adding some constraints on the use of some other its services.

3.3 Distribution Model

The distribution model has the goal of defining the software infrastructure that allows the communication of a runtime node with the other nodes of an application possibly through different types of communication supports, guaranteeing a transparent communication among their processes. This model is based on three kinds of element: distributor, connector and connection.

A distributor has the duty of managing the connections with the other runtime nodes of the application. This distributor manages connections that can be realized with different kinds of communication technology through the use of different connectors (see Figure 3). Moreover, a pair of runtime nodes can be connected through different connections.

A connector is a connections handler that manages the connections of a runtime node with a specific communication technology allowing the exchange of messages between the processes of the accessible runtime nodes that support such a communication technology.

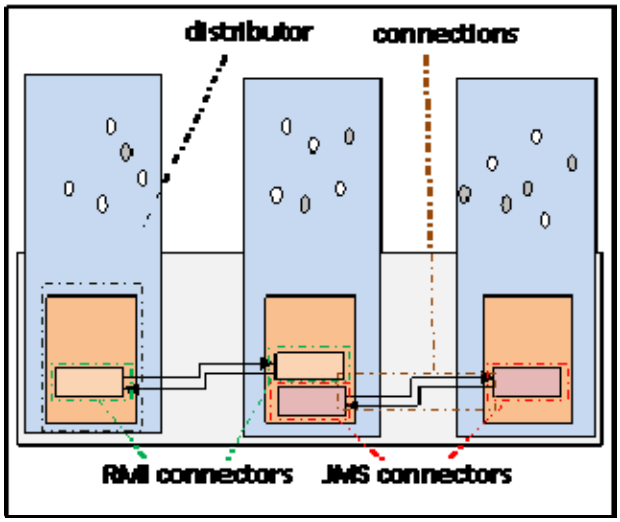

Fig. 3. An HDS application based on three runtime nodes connected through RMI and JMS technologies.

A connection is a mono-directional communication channel that provides the communication between the processes of two runtime nodes through the use of remote references. In particular, a connection provides a remote lookup service offering the listing of the remote processes and the access to their remote references..

4 Communication

Processes interact through messages that can be considered typed because of their content. In fact, the content of a message is represent by an object whose type indicates the use that the receiver should make of the content of the message.

Typed messages allow to couple the meaning of the communicative acts of an ACL with the meaning of the concepts expressed by a domain ontology. In fact, a first hierarchy of Java interfaces can allow to split the messages on the basis of their ACL communicative act, i.e., their performatives, and then other interfaces and concrete classes can specialize such interfaces for defining the ontology of a specific application domain, i.e., the messages ex-changed during the execution of the tasks of such an application domain.

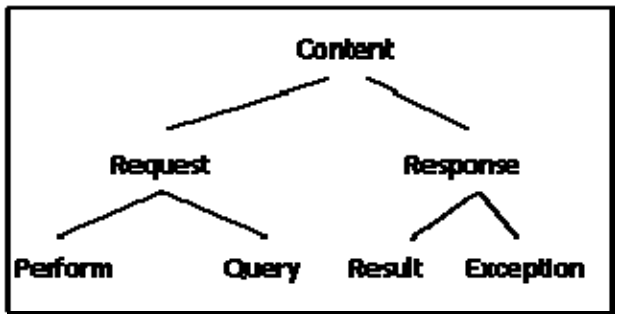

Fig. 4. Classification of the content of the messages derived from ACL communicative acts.

The communication module partially takes advantage of the classification of ACL messages on the basis of their performatives by refining the concept of message content through a hierarchy of Java interfaces. However, messages are not classified on the basis of the complete list of the communicative acts provided by an ACL (e.g., KQML [10] or FIPA ACL [11], but: i) messages are split into request and response messages, distinguishing the messages starting and interaction from the ones that reply to them (in this case, the term "request" has a different meaning of the typical ACL related term), ii) request messages are split into perform and query messages, and iii) response messages are split into result and exception messages.

It was done because from our experience and from the analysis of several real application scenarios and software implementations, what can be important (or at least can simplify the implementation) in some kinds of application is to be able to distinguish: i) the requests that require the reading and the sending of data (i.e., query requests) from the ones that require the execution of actions that can also modify the state of the process (i.e., perform requests), and ii) the positive and negative responses. Figure 4 shows the hierarchy of interfaces classifying message content on the basis of ACL communicative acts.

5 Implementation

The HDS software framework has been realized taking advantage of the Java programming language. The application architecture model has been defined through the use of Java interfaces and its implementation has been divided in two modules.

The first module contains the software components that define the software infrastructure and that are not directly used by the developer, that is, all the software components necessary for managing the lifecycle of processes, the local and remote delivery of messages and their filtering. In particular, the remote delivery of messages has been provided through both Java RMI [16] and JMS [13] communication technologies.

The second module contains both the software components that application developers extend, implement or at least use in their code, and the software components that help them in the deployment and execution of the realized applications. The identification of such software components can be easily done by analyzing what application developers need to realize: i) the actor and server classes used for the implementation of the processes involved in the application, ii) the description selector classes used for the discovery of the processes involved in common tasks, iii) the message filter classes used for customizing the communication among the processes, iv) the typed messages used in the interaction among the processes, and v) the artifacts (i.e., Java classes and/or configuration files) for the deployment of the runtime nodes and of the communication channels among runtime nodes, and for the startup of the initial sets of processes and message filters.

It implies that such a module needs to contain; i) some software components for simplifying the realization of actors, servers, description selectors and message filters (realized through four abstract classes called AbstractActor, AbstractServer, AbstractSelector and AbstractFilter), ii) a set of abstract and concrete typed messages useful for realizing the typical communication protocols used in distributed applications, and ii) a software tool that allows the deployment of a HDS software application through the use of a set of configuration files (realized through a concrete class called Launcher).

In regard to type messages and the related communication protocols, the software framework provides the basis interfaces and classes for realizing application dependent client-server protocols and the basic interfaces and classes for supporting the interaction among processes through the use of communication language derived by the agent communication language (ACL) defined in the FIPA specifications [8]. This communication language differs from the ACL defined by FIPA specifications in the fact that the content object that defines a typed message groups together the

meaning of the performative, content and ontology of a FIPA ACL message.

Moreover, the features provided by typed messages make them suitable for implementing the usual coordination and negotiation techniques expressed by multi-agent systems and for using such techniques also outside of a multi-agent system and without any (or with a limited) knowledge about agent communication languages and ontologies.

In particular, the software framework provides an abstract implementation of some interaction protocols (i.e. the English and Dutch auction protocols, the Contracted Net and the iterated Contract Net protocols and the brokering and the recruiting protocols) that derive from the interaction protocols defined in the FIPA specifications [8]. This implementation replaces the ACL messages with typed messages and delegates to the application developer only the duty of writing the code for processing the content of the messages, selecting the messages to be sent and building their content.

For example, the abstract implementation of the iterated Contract Net protocol is based on two abstract classes, that describe the two roles involved in the protocol, i.e., the initiator and the participant, and an interface, called Contract, used in the content of the exchanged messages for maintaining the information about both the task to be executed and the bids of the participants.

The abstract class that represent the initiator role defines three main methods; the first method sends a "offer" message to the list of processes acting as participants. The second method is an abstract method whose implementation must select the participant to which send either an "accept" or another "offer" message. Finally, the third method is an abstract method whose implementation must process the message containing the results of the execution of the required task.

The abstract class, that represent the participant role, defines two main methods. The first method is an abstract method whose implementation must decide to propose a bit for the task described by the "offer" message or to refuse it. The second message must decide to execute the task and then must send the information about the results of its execution.

Therefore, the used of the iterated Contract Net protocol inside an application requires: i) the definition of concrete class implementing the Contract interface, ii) the definition of a concrete class that extend the initiator abstract class implementing the methods for accepting, refusing or sending an updated contract and for processing the result received by the participant(s) to which the contract(s) have been assigned, and iii) the definition of at least a concrete class that extend the participant abstract class implementing the methods for accepting or refusing an offer and for performing the task associated with the contract.

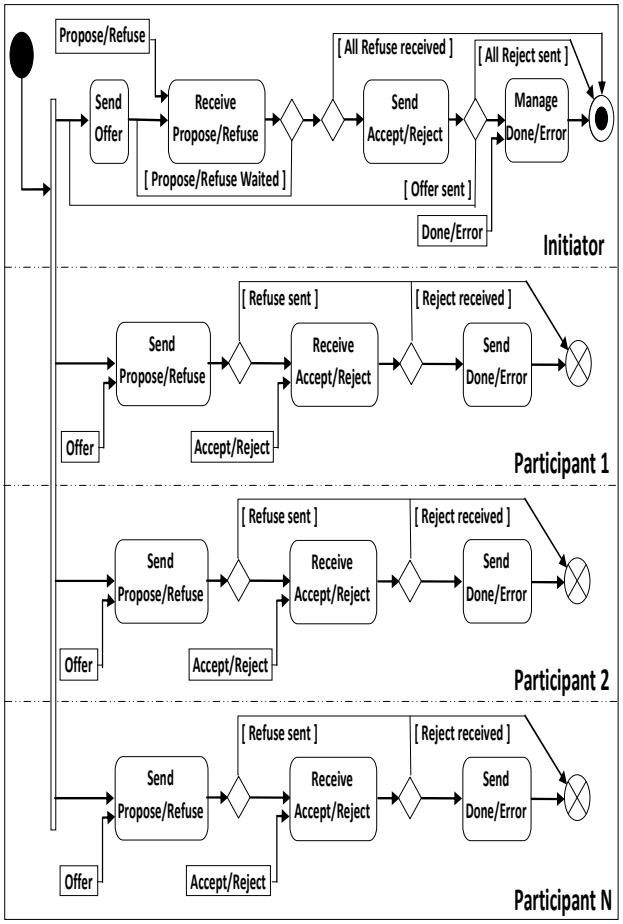

Fig. 5. An UML activity diagram showing a description of the typed messages based iterated Contract Net protocol.

For example, the abstract implementation of the iterated Contract Net protocol is based on two abstract classes, that describe the two roles involved in the protocol, i.e., the initiator and the participant, and an interface, called Contract, used in the content of the exchanged messages for maintaining the information about the task to be executed and the bids of the participants.

The abstract class, that represent the initiator role, defines three main methods; the first method sends a "offer" message to the list of processes received as input parameter. The second method is an abstract method whose implementation must

select the participant to which send either an "accept" or another "offer" message. Finally, the third method is an abstract method whose implementation must process the message containing the results of the execution of the required task.

The abstract class, that represent the participant role, defines two main methods. The first method is an abstract method whose implementation must decide to propose a bit for the task described by the "offer" message or to refuse it. The second message must decide to execute the task and then must send the information about the results of its execution.

Figure 5 shows a UML activity diagram that presents a partial description of the behavior of the iterated Contract Net protocol described above. This partial description allows to simplify the graphical representation of the behavior of the protocol removing all the arcs connecting the input messages to an activity with the source of the message: it is possible because, of course all the input messages of the initiator activities come from the participants and all the input messages of the activities of a participant come from the initiator.

6 HDS and Pervasive Adaptation

Providing adaptation in a pervasive environment is a challenging task as the adaptation concern affects multiple elements (devices, services, etc.) in the environment. The problem is further compounded by the fact that the elements are geographically distributed and in many instances there is no central node controlling the operation of the pervasive environment [13,26].

HDS integrates multi-agent systems and aspect-oriented techniques for the realization of adaptive and pervasive applications. In fact, multi-agent systems are based on autonomous software entities that can interact with their environment, and therefore they adapt well to the dynamic nature of pervasive applications [6,7,12,21,31,34]. Aspect-oriented techniques [8,14] are considered a suitable means for the realization of complex applications that are composed of different interleaved cross-cutting concerns (properties or areas of interest such as energy consumption, fault tolerance, and security) and then they are indicated for providing adaptation in pervasive environments given that adaptation largely affects the other features (concerns) of such environments [26,13].

HDS is not a software framework specialized for the development of multi-agent systems, however, its actors and its typed messages allow both the development of specific software components for the realization of multi-agent systems (e.g., agent oriented architectures and abstract implementations of agent interaction protocols) and the use of agent-based techniques and models in other kinds of software systems.

In particular, actors can provide the autonomous and proactive behavior that is fundamental to realize systems that relieve users of the burden of the manual reconfiguration operations needed when such systems must be adapted to new situations. To be able to perform such a reconfiguration, actors need to cooperate among them; it can be done through the use of typed messages for implementing all the negotiation protocols that are necessary for the coordination of the reconfiguration activities.

In principle, the combination of actors and typed messages may be sufficient for realizing effective adaptive and pervasive applications, however, in practice, the corresponding performance and the development costs may be too high to be used for real adaptive and pervasive applications. Therefore, actors and typed messages are not used to adapt the behavior of an application, but are used for driving the adaptation of the application by modifying the sets of input and output message filters associated with the processes involved in the application.

Messages filters allow the adaptation of the behavior of an application with a limited computational overhead. In fact, besides to be used for implementing the typical services of an aspect oriented support, i.e., security, persistence and logging, message filters can be used for adapting the application to any hardware and software modification and for personalizing the application to the users that are using it (e.g., an input and the corresponding output message filter can adapt the interface between a client and a server, a set of messages filters, distributed on a net of processes, can route a request towards the most appropriate process (or user) that can serve the request or can combine the tasks of different processes to provide new kinds of service).

As introduced Above, message filters are able to adapt an application with a limited overhead, this is possible because each message filter usually realizes a very simple task (e.g., it forwards or transforms a message) and because their management is very simple (i.e., a message filter

can either propagate a message to the next filter of the list or stop the propagation of the message). However, message filters can be considered the suitable "bricks" for adapting an application, but they need some additional software supports that allow to move the application among different configurations of the message filters that are suitable for guaranteeing a correct behavior of the application in all the possible states of the environment.

Therefore, message filters are only used for guaranteeing the appropriate behavior of a pervasive application in a specific state of the environment and delegate the dynamic adaptation of the application to a set of actors. Moreover, as introduced above, the use of the actor for the dynamic adaptation of an application may introduce important overhead on the performance of the application when the reconfiguration of the message filters requires complex reasoning and negotiation tasks. However, it should be an exception and not the norm because the state of a pervasive environment evolves through a sequences of changes that usually in few cases cause important modifications in the environment and then the reconfiguration overhead is usually limited

7 Experimentation

A first experimentation of the HDS software framework has been and is still now done and is oriented to demonstrate that: i) such a software framework is suitable to realize complex applications and multi-agent systems, and ii) makes easy the reuse of the typical models and techniques, that are exhibited in multi-agent systems (i.e., the interaction protocols), in other kinds of software systems.

The experimentation of the software framework as means for realizing complex system consists in the development of an environment for the provision of collaborative services for social networks and, in particular, for supporting the sharing of information among users.

The current release of such an environment allows the interaction among users that are connected through heterogeneous networks (e.g., traditional wired and wireless computer networks and GSM and UMTS phone networks), devices (e.g., personal computers and smart phones), software (e.g., users can interact either through a Web portal or through specialized application, and users can have at their disposal applications that are able to either visualize and modify or only visualize documents) and rights (e.g., some users have not the right of performing a subset of the actions that are possible in the environment). The experimentation is still in the first phase, but has been already sufficient to realize that the integration between actors, servers and message filters is a profitable solution for the realization of adaptive and pervasive applications.

The experimentation of the use of multi-agent typical models and techniques and of the use of such a software framework for realizing multi-agent systems started with the development of an abstract implementation of the BDI agent architecture [19], and then continued on the parallel development of two prototypes of a market place application: both the prototypes are based on the HDS implementation of FIPA interaction protocols, but only the first realize the prototype as a multi-agent system by taking advantage of realized BDI agent abstract architecture.

The abstract BDI agent architecture was realized in few time by simply extending the abstract actor class with: i) a working memory, where maintaining the beliefs, desires and intentions of the agent, ii) a plan library, where maintaining the set of predefined plans and iii) an interpreter able to select the plan that must be used to achieve the current intension, and then able to execute it. Therefore, starting from this abstract implementation, a concrete BDI agent can be obtained by providing the code for initializing and updating the working memory and for filling the plan library.

After the realization of such abstract agent architecture, we start the parallel development of the prototypes of a market place where agents can buy and sell goods through the use of English and Dutch auctions.

This experimentation was done by two groups of master students: the students of the first group had a good knowledge about artificial intelligence and agent-based systems, because they followed two courses on those topics, and the students of the second group had few knowledge about such topics, because they did not follow any related course.

In few words, the results of the experimentation were that: all the students had not difficulties to obtain a good implementation of the version of the application with-out the use of the BDI agents, but while the students without any knowledge about artificial intelligence and agent-based system had great difficulties for realizing the version of the

application based on the use of BDI agents and realized some prototypes that do not completely exploit the characteristic of BDI agents, the other students did it obtaining more flexible prototypes in respect to the ones without BDI agents, but spending a lot of time in their implementation.

8 Conclusion

This paper presented a software framework, called HDS, that has the goal of simplifying the realization of distributed applications by merging the client-server and the peer-to-peer paradigms and by implementing the interactions among all the processes of a system through the exchange of typed messages.

HDS is implemented by using the Java language and its use simplify the realization of systems in heterogeneous environments where computers, mobile and sensor devices must cooperate for the execution of tasks. Moreover, the possibility of using different communication protocols for the exchange of messages between the processes of different computational nodes of an application opens the way for a multi-language implementation of the HDS framework allowing the integration of hardware and software platforms that do not provide a Java support.

HDS can be considered a software framework for the realization of any kind of distributed system. Some of its functionalities derive from the one offered by JADE [3,4], a software framework that can be considered one of the most known and used software framework for the developing of multi-agent systems. This derivation does not depend only on the fact that some of the people involved in the development of the HDS software framework were involved in the development of JADE too, but because HDS tries to propose a new view of multi-agent systems where the respect of the FIPA specifications are not considered mandatory and ACL messages can be expressed in a way that is more usable by software developers outside the multi-agent system community. This work may be important not only for enriching other theories and technologies with some aspects of multi-agent system theories and technologies, but also for providing new opportunities for the diffusion of both the knowledge and use of multi-agent system theory and technologies.

HDS is a suitable software framework for the realization of pervasive applications. Some of its features introduced above (i.e., the java implementation, the possibility of using different communication protocols and the possibility a multi-language implementation) are fit for such kinds of application. However, the combination of multi-agent and aspect-oriented techniques might be one of the best solutions for providing an appropriate adaptation level in a pervasive application. In fact, this solution allows to couple the power of multi-agent based solutions with the simplicity of compositional filters solutions guaranteeing both a good adaptation to the evolution of the environment and a limited overhead to the performances of the applications.

Currently we are using HDS for the realization of: i) an agent based social network and ii) a Java framework for prototyping and evaluating distributed constraint satisfaction algorithms.

In the first application, HDS is used for providing an agent based support layer for the interaction among users in the social network In particular, agents are associated with the users and agents can proactively act on the behalf of their users by taking advantage the information contained in their profile. In fact, agents have two main roles: i) they mediate the access to the profile information of their users (allowing or refusing queries from other agents) ii) they use information in the profile in order to discover new friendships and acquaintances on their owner's behalf.

In the second application, HDS is used for implementing the agents that concur to the resolution of a problem through the use of a distributed constraint satisfaction algorithms. Of course, such an application takes advantage of both type messages and messages filters: coordination algorithms among agents are realized through the exchange of typed messages and message filters are used for the instrumentation of the code enabling performance measurement, debugging and fine tuning of the algorithms without any modification to their code.

Future research activities will be dedicated, besides to continue the experimentation and validation of the HDS software framework in the realization of collaborative services for social network, to the improvement of the HDS software framework. In particular, current activities are dedicated to: i) the implementation of more sophisticated adaptation services based on message filters taking advantages of the solutions presented by PICO [11] and by PCOM [2], ii) the automatic

creation of the Java classes representing the typed messages from OWL ontologies taking advantage of the O3L software library [17], and iii) the extension of the software framework with a high-performance software library to support the communication between remote processes, i.e., MINA [1].

References:

[1] Apache Foundation. MINA software. Web site. Available from: http://mina.apache.org.

[2] Becker, C., Hante, M., Schiele, G., and Rotheemel, K. PCOM - a component system for pervasive computing. In Proceedings of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom 2004), Orlando, FL, 67-76, 2004.

[3] Bellifemine, F., Poggi, A., and Rimassa, G. Developing multi agent systems with a FIPA-compliant agent framework. Software Practice & Experience, 31:103-128, 2001.

[4] Bellifemine, F., Caire, G., Poggi, A., and Rimassa, G.. JADE: a Software Framework for Developing Multi-Agent Applications. Lessons Learned. Information and Software Technology Journal, 50:10-21, 2008.

[5] Bergmans, L., and Aksit, M. Composing crosscutting concerns using composition filters. Communications of ACM, 44(10):51-57, 2001.

[6] Cabri, G., Ferrari, L., and Zambonelli, F. 2004. Role-Based Approaches for Engineering Interactions in Large-Scale Multi-agent Systems. In C. Lucena, A. Garcia, A. Romanovsky, J. Castro, P. Alencar eds. Software Engineering for Multi-Agent Systems II, Lecture Notes in Computer Science, 2940, pp. 243-263, Springer-Verlag, Berlin, Germany.

[7] Chakraborty, D., Takahashi, H., Suganuma, T., Takeda, A., Kitagata, G., Hashimoto, K., and Shiratori, N. An adaptive context aware communication system for ubiquitous environment based on overlay network. In Proceedings of the 12th WSEAS International Conference on Computers N. E. Mastorakis, V. Mladenov, Z. Bojkovic, D. Simian, S. Kartalopoulos, A. Varonides, C. Udriste, E. Kindler, S. Narayanan, J. L. Mauri, H. Parsiani, and K. L. Man, Eds. Recent Advances In Computer Engineering, pp. 832-837, World Scientific and Engineering Academy and Society (WSEAS), Stevens Point, Wisconsin, 2008.

[8] Cheng, S., Garlan, D., Schmeri, B.R., Sousa, J.P., Spitznagel, B., Steenkiste, P., and Hu, N. Software Architecture-Based Adaptation for Pervasive Systems. In H. Schmeck, T. Ungerer, and L.C. Wolf, Eds. Trends in Network and Pervasive Computing, Lecture Notes In Computer Science, 2299, pp. 67-82, Springer-Verlag, London, U.K., 2002.

[9] Filman, R., Elrad, T., Clarke, S., and Aksit, M. Aspect-Oriented Software Development. Addison-Wesley, 2004.

[10] Finin, T., Fritzson, R., McKay, D. and McEntire, R. KQML as an agent communication language. In Proceedings of the 3rd International Conference on information and Knowledge Management, pp. 456-463, Gaithersburg, MD, 1994.

[11] FIPA Consortium. FIPA Specifications. Available from http://www.fipa.org.

[12] Fok, C., Roman, G., and Lu, C. Agilla: A mobile agent middleware for self-adaptive wireless sensor networks. ACM Transactions on Autonomous and Adaptive Systems, 4(3):1-26, 2009.

[13] Funk, C., Schultheis, A., Linnhoff-Popien, C., Mitic, J., and Kuhmunch, C. Adaptation of Composite Services in Pervasive Computing Environments. In Proceedings of IEEE International Conference on Pervasive Services, Istanbul, Turkey, 242-249, 2007.

[14] Fuentes, L., Gamez, N., and Sanchez, P. Aspect-oriented design and implementation of context-aware pervasive applications. Innovations in Systems and Software Engineering, 5(1):79-93, 2009.

[15] Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., and Irwin, J. Aspect-oriented programming. In M, Aksit and S. Matsuoka eds. Proceedings of the European Conference on Object-Oriented Programming (ECOOP). Lecture Notes in Computer Science, 1241, pp. 220-242, Springer-Verlag, Berlin, Germany, 1997.

[16] Kindberg, T., and Fox, A. System Software for Ubiquitous Computing. IEEE Pervasive Computing, 1(1):70-81, 2002.

[17] 17]Kumar, M., Shirazi, B.A., Das, S.K., Sung, B.Y., Levine, D., and Singhal, M. PICO: A Middleware Framework for Pervasive Computing. IEEE Pervasive Computing, 2(3):72-79, 2003.

[18] Kumar, M., and Zambonelli, F. Middleware for pervasive computing. Pervasive Mobile Computing, 3(4):329-331, 2007.

[19] Monson-Haefel, R. and Chappell, D. Java Message Service. O'Reilly & Associates, 2000.

[20] Niemelä, E., and Latvakoski, J. 2004. Survey of requirements and solutions for ubiquitous

software. In Proceedings of the 3rd International Conference on Mobile and Ubiquitous Multimedia, College Park, MD, 71-78, 2004.
[21] Porekar, J., Dolinar, K., and Jerman-Blazic, B. Middleware for Privacy Protection of Ambient Intelligence and Pervasive Systems. WSEAS Transactions on Information Science & Applications, 3(4):633-639, 2007.
[22] Pham, H., Paluska, J.M., Saif, U., Stawarz, C., Terman, C., and Ward, S. A dynamic platform for runtime adaptation. Pervasive and Mobile Computing, 5(6):676-696, 2009.
[23] Platon, E., Mamei, M., Sabouret, N., Honiden, S., and Parunak, H.V. Mechanisms for environments in multi-agent systems: Survey and opportunities. Autonomous Agents and Multi-Agent Systems, 14(1):31-47, 2007.
[24] Pitt, E. McNiff, K. Java.rmi: the Remote Method Invocation Guide. Addison-Wesley, 2001.
[25] Poggi, A. Developing Ontology Based Applications with O3L. WSEAS Transactions on Computers, 8(8):1286-1295, 2009
[26] Raatikainen, K., Chrisrensen, H.B., and Nakajima, T. Application requirements for middleware for mobile and pervasive systems. ACM SIGMOBILE - Mobile Computing and Communications Review. 6(4):16-24, 2002.
[27] Rao, A.S., and Georgeff, M.P. BDI Agents: From Theory to Practice. In Proceedings of the First International Conference on Multiagent Systems, pp. 312-319, San Francisco, CA, 1995.
[28] Rashid A., and Kortuemì, G. Adaptation as an Aspect in Pervasive Computing. In Proceedings of the Workshop on Building Software for Pervasive Computing at the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSA 2004), Vancouver, Canada, 2004. Available from: http://www.ics.uci.edu/~lopes/bspc04-documents/RashidKorteum.pdf.
[29] Saha, D., and Mukherjee, A. Pervasive Computing: A Paradigm for the 21st Century. Computer 36(3):25-31, 2003.
[30] Satyanarayanam, M. Pervasive Computing Vision and Challenges. IEEE Personal Communications, 6(8):10-17, 2001.
[31] Sousa, J. P., Poladian, V., Garlan, D., Schmeri, B., and Shaw, M. Task-based adaptation for ubiquitous computing. IEEE Transactions on Systems, Man, and Cybernetics, 36(3):328-340, 2006.
[32] Soylu, A., De Causmacker, P, and Desmet, P. Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. Journal of Software, 4(9):992-1013, 2009.
[33] Tesauro G., Chess, D.M., Walsh, W.E., Das, R., Segal, A., Whalley, I., Kephart, J. O., and White, S.R. A Multi-Agent Systems Approach to Autonomic Computing. In Proceedings of the 3rd international Joint Conference on Autonomous Agents and Multiagent Systems, pp. 464-471, New York, NY, 2004.
[34] Wu, Q. and Wang, D. An adaptive requirement framework for SCUDW are middleware in ubiquitous computing. WSEAS Transactions on Computer, 8(1):163-173, 2009.