Towards a Flexible Development Framework for Multi-Agent Systems

Agostino Poggi
Dipartimento di Ingegneria dell'Informazione
University of Parma
Parma, Italy
agostino.poggi@unipr.it

Abstract—**In this paper, we present a software framework, called HDS (Heterogeneous Distributed System), that tries to simplify the realization of distributed applications and, in particular, of multi-agent systems, by: i) abstracting the use of different technologies for the realization of distributed applications on networks of heterogeneous devices connected through a set of different communication transport protocols, ii) merging the client-server and the peer-to-peer paradigms, and iii) implementing all the interactions among the processes of a system through the exchange of typed messages that allow the implementation of a large set of communication protocols and languages.**

Keywords - software framework; layered framework; typed message, composition filter.

I. Introduction

In the early 1990s, Multi-Agent Systems (MAS) were put forward as a promising paradigm for the realization of complex distributed systems. Over the years, MAS researchers have developed a wide body of models, techniques and methodologies for developing complex distributed systems, have realized several effective software development tools, and have contributed to the realization of several successful applications. However, even if today's software systems are more and more characterized by a distributed and multi-actor nature, that lends itself to be modeled and realized taking advantage of MAS techniques and technologies, very few space in software development is given to the use of such techniques and technologies.

It is due to several reasons [1][2][3][4][5]. One of the most important reasons is that the large part of software developers have few knowledge about MAS technologies and solutions: it is mainly because of the lack of references to the results of MAS research outside the MAS community. Moreover, even when there is a good knowledge about MAS, software developers do not evaluate the possibility of their use because: i) they believe that multi-agent approaches are not technically superior to traditional approaches (i.e., there are not problems where a MAS approach cannot be replaced by a non-agent approach), and ii) they consider MAS approaches too sophisticated and hard to understand and to be used outside the research community.

Therefore, it is possible to state that the MAS community has yet to demonstrate the significant benefits of using agent-oriented approaches to solve complex problems, but also that some efforts should be done for facilitating the use and the integration of MAS technologies and solutions in mainstream software development technologies and solutions.

In particular, MAS developers should avoid to consider MAS solutions a "panacea" for all the kinds of system. Therefore, a MAS should be realized only when the components of a system must express the typical features (i.e., proactiveness, sociality and goal) that distinguish a software agent from another software component.

The case when a new MAS is realized and must be embedded and integrated with an existing software environment, where other software systems are running and interact with each other, deserves particular attention. In this case, MAS developers usually try to propose, as most effective integration solution, the "agentification" of the other software systems even if this kind of integration, besides requiring the burden of encapsulating the interface of a non-agent based system with a code that generates and processes messages of the agent communication language (ACL) used by the MAS, often might not guarantee the level of performance, security and transactionality required in such a software environment. It is because the majority of the MAS community has still the belief that agents are the best means for supporting software interoperability. This belief comes from the first important works on MAS [6][7] and was partially supported by the specifications for agent interoperability defined by FIPA [8]; however, in the last ten years the advent and the success of Web services and service-oriented architecture have shown that what software customers and developers want is a solution that simply renew the traditional ways of integrating systems, avoiding any solution that might be considered too complex or that might reduce the quality of the system.

In this paper, we present a software framework, called HDS (Heterogeneous Distributed System), whose goal is to simplify the realization of distributed applications taking advantage of multi-agent model and techniques and providing an easy way for the integration between MAS and non-agent based systems. The next section describes the main features of the HDS software framework. Section three discusses about the experimentation of such a framework for the realization of both

MAS and non-agent based systems. Finally section four concludes the paper sketching some future research directions.

II. 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 the processes of a system through the exchange of typed messages. In particular, HDS provides both proactive and reactive processes (respectively called actors and servers) and an application can be distributed on a (heterogeneous) network of computational nodes (from now on called runtime nodes).

The HDS software framework model is based on two layers called, respectively; concurrency and runtime layers. While the first layer defines the elements that an application developer must directly use for realizing applications, the second layers, besides providing the services that enable the creation of the elements of the concurrency layer and their interaction, abstracts the use of different technologies for realizing distributed applications on nets of heterogeneous devices connected through a set of different communication transport protocols.

A. Concurrency Layer

The concurrency layer is based on six main elements: process, description, selector, message, content and 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. Usually a description contains the process identifier, the process type and the data that have been used for its initialization; however, a process may introduce some additional information in its description.

A process can be either an actor or a server. An actor is a 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 reactive 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: i) synchronous communication, the process sends a message to another process and waits for its answer; ii) asynchronous communication, the process sends a message to another process, performs some actions and then waits for its answer and iii) 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 either respond to the requests of the other processes or can delegate the execution of such requests to some other processes.

Taking advantage of the registry service provided by the runtime layer, a process has also the ability of discovering the other processes of the application. In particular, a process can: i) check if an identifier is bound to a process of the application, ii) get the identifiers of the other processes of its runtime node, and iii) get the identifiers of the processes of the application whose description satisfies some constraints. The last capability is possible, because, a process can create a special type of object, called selector, that define 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 or suffix), then the process sends such a selector to the registry service provided by the runtime layer, and the registry service applies the constraints defined by the selector on the information of the registered process descriptions and sends to the processes the identifiers of the processes that satisfy the constraints defined by the selector.

As we wrote above, processes interact with each other through the exchange of messages. 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). In particular, the message model defined by the concurrency layer allows the implementation of a large set of communication protocols and languages. In fact, the traditional client-server protocol can be realizing associating the request and response data to the message content element and the most known agent communion language, i.e., KQML and FIPA ACL [9], can be realized by using the content element for the representation of ACL messages.

Normally, a process can interact with all the other processes of the application and the sending of messages does not involve any operation that is not related to the delivery of messages to the destination; however, the presence of message filters can modify the normal delivery of messages. A message filter is a composition filter [10] 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.

The runtime layer associates two lists of message filters with each process: 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. When a new message arrives or must be sent, the relative message filters 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.

B. Runtime Layer

The main goal of the runtime layer is to allow the use of different technologies for realizing distributed applications on nets of heterogeneous devices connected through a set of different communication transport protocols, but abstracting such technologies through a tiny API towards the concurrency layer. In particular, the runtime layer defines a set of services that can be used by the concurrency layer and a set of interfaces that must be implemented for integrating a new technology and using it for providing the services provided by the runtime layer.

The main element of the runtime layer is the reference. A reference is a proxy of the process that makes transparent the communication respect to the location of the process and the technologies connecting the reference with its process. The duty of a reference is to allow the insertion of messages in the queue of its process. Therefore, when a process wants to send a message to another process, it must obtain the reference to such a process and then use it for putting the message in the input queue of the other process.

The access to the reference of a process is possible through the use of the registry service. This service is provided by the runtime layer to the processes of an application and allows: the binding and unbinding of the processes with their identifiers, description, and references, the retrieval of a reference on the basis of the process identifier and the listing of sets of identifiers of the processes of an application by using, if necessary, some selectors..

The runtime layer has also the duty of creating processes and their related references. In fact, it provides a factory service that allows a process of an application to create other processes by proving to the runtime layer the qualified name of the class implementing the process and its initialization list.

Finally, the runtime layer provides a filtering service that allow the management of the list of message filters associated with the processes of an application. In fact a process cannot modify any list of message filters. Therefore, taking advantage of the filterer service, a process can modify the lists of its message filters, but can also drive the behavior of some other processes by modifying their message filter lists.

C. Implementation

The HDS software framework has been realized taking advantage of the Java programming language. While the runtime layer has been implemented for providing the remote delivery of messages through both Java RMI [11] and JMS [12] communication technologies, the concurrency layer provides: i) a message implementation, ii) four abstract classes that implement the application independent parts of actors, servers, selectors and filters, and iii) a set of abstract and concrete content classes useful for realizing the typical communication protocols used in distributed applications. In particular, HDS provides a client-server implementation of all the protocols used by processes for accessing to the services of the runtime layer and a complete implementation of the FIPA ACL coupled with an abstract implementation of the "roles" involved in the FIPA interaction protocols. Moreover, for simplified the deployment of application, current HDS implementation provides a software tool that allows the deployment of applications through the use of a set of configuration files.

III. 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 MAS, and ii) makes easy the reuse of the typical models and techniques, that are exhibited in MAS (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 MAS started with the development of an abstract implementation of the BDI agent architecture [13], 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 MAS 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 goal, 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 without the use of the BDI agents that, however, take advantages of the typical multi-agent interaction protocols, 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 than the ones without BDI agents, but spending a lot of time in their implementation.

IV. CONCLUSIONS

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 [14][15][16], a software framework that can be considered one of the most known and used software framework for the developing of MAS. 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 MAS 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 MAS community. This work may be important not only for enriching other theories and technologies with some aspects of MAS theories and technologies, but also for providing new opportunities for the diffusion of both the knowledge and use of MAS 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.

Current and future research activities are 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 [17] and by PCOM [18], ii) the automatic creation of the Java classes representing the typed messages from OWL ontologies taking advantage of the O3L software library [19], and iii) the extension of the software framework with a high-performance software library to support the communication between remote processes, i.e., MINA [20].

REFERENCES

[1] V. Maříka and J. Lažanský. Industrial applications of agent technologies. Control Engineering Practice, 15(11):1364-1380, 2007.

[2] L. Braubach, A. Pokahr and W. Lamersdorf. A Universal Criteria Catalog for Evaluation of Heterogeneous Agent Development Artifacts. In Proc. of the Sixth Int.Workshop "From Agent Theory to Agent Implementation" (AT2AI-6), pp. 19-28, Estoril, Portugal, 2008.

[3] J. McKean, H. Shortery, M. Luckz, P. McBurneyx and S. Willmott. Technology diffusion: analysing the diffusion of agent technologies, Autonomous Agents and Multi-Agent Systems, 17(2):372-396, 2008.

[4] S. A. DeLoach. Moving multi-agent systems from research to practice. Agent-Oriented Software Engineering, 3(4):378-382, 2009.

[5] D. Weyns, A. Helleboogh and T. Holvoet. How to get multi-agent systems accepted in industry? Agent-Oriented Software Engineering, 3(4):383-390, 2009.

[6] M. R. Genesereth and S. P. Ketchpel. Software Agenta, Communications of ACM, 37(7):48-63, 1994.

[7] J. M. Bradshaw. An introduction to software agents. in, Jeffrey M. Bradshaw, Ed, Software Agents, pp. 3-46, MIT Press, Cambridge, MA, 1997.

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

[9] Y. Labrou, T. Finin, and Y. Peng. Agent Communication Languages: The Current Landscape. IEEE Intelligent Systems, 14(2):45-52. 1999.

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

[11] E. Pitt and K. McNiff. Java.rmi: the Remote Method Invocation Guide. Addison-Wesley, 2001.

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

[13] A. S. Rao and M. P. Georgeff: BDI Agents: From Theory to Practice. In Proc. of the First International Conference on Multiagent Systems, pp. 312-319, San Francisco, CA, 1995.

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

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

[16] JADE. Available from http://jade.tilab.com.

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

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

[19] A. Poggi. Developing Ontology Based Applications with O3L. WSEAS Trans. on Computers 8(8):1286-1295, 2009.

[20] MINA. Available from: http://mina.apache.org.