Realizing Distributed and Pervasive Information Management Systems with HDS

Federico Bergenti[*] and Agostino Poggi[+]

[*]Dipartimento di Matematica, Università degli Studi di Parma
Viale U. P. Usberti 53/A, 43100 Parma, Italy

[+]Dipartimento di Ingegneria dell'Informazione, Università degli Studi di Parma
Viale U. P. Usberti 181/A, 43100 Parma, Italy

{federico.bergenti,agostino.poggi}@unipr.it

Abstract. This paper focuses on realizing distributed and pervasive information management systems using the novel tool HDS (Heterogeneous Distributed System). First, we frame the presented research in the scope of information agent technology by reviewing relevant work on the subject and by stating the notable role that communication plays in such a field. Then, we introduce HDS and we provide details on its functionality and internals. Finally, we present the RAIS (Remote Assistant for Information Sharing) pervasive information management system and we show how it has been recently retargeted to use HDS as its main communication and distribution vehicle.

1 Introduction

The role of agent technology in the field of information management systems is remarkable mainly because *(i)* it provides solid abstractions and models that help the study and realization of such systems; and *(ii)* it supports their concrete implementation with mature tools that targets large-scale, decentralized and pervasive systems. The important role of agent technology for this kind of systems is also witnessed by the recent introduction of the idea of *intelligent information agent* to support the realization of decentralized and scalable information management systems.

A closer look to the relationship between agent technology and information management systems reveals that the agent paradigm has been intensively applied in the realization of such systems for taking advantage of various features of agents. For example, we see agents used in information management systems as a means for supporting communication, coordination and even presentation of information to users (see, e.g. [7]).

In Section 2 we review relevant work on the use of agent technology for information management systems and we stress the role of agents in communication and in support of decentralized, scalable distribution. Agents are ideal tools for implementing software systems with advanced features in terms of decentralized, scalable distribution and therefore they are a key ingredient in next generation information management systems where the amount of managed information is going to overwhelm any centralized approach.

In this scenario of agent-based information management systems capable of effectively coping with unforeseen information overload, we present *HDS (Heterogeneous Distributed System)*, a software framework that simplifies the realization of decentralized applications by merging the client-server and the peer-to-peer paradigms and we discuss its uses for the realization of flexible and pervasive information management systems. Section 3 introduces HDS and it provides an overview of its features and internals.

In order to exemplify the role of HDS in the realization of decentralized information management systems, Section 5 describes *RAIS (Remote Assistant for Information Sharing)*, a peer-to-peer multi-agent system supporting the sharing of information among a community of users connected through the Web. RAIS offers a search facility similar to Web search engines, but it avoids the burden of publishing the information on the Web while providing warranties of controlled and dynamic access to the information. The use of agents in such a system is very important because it simplifies the realization of three core RAIS services: information filtering, information publishing and management of networks of reputation.

Finally, the paper concludes with Section 6 that summarizes the results and sketches some future research directions.

2 Intelligent Information Agents

Information agent technology [5] is a keyword that arose in the research on software agents around ten years ago to provide concrete responses to the challenges that the Internet scale posed to information management systems. While information agent technology does not bring any new feature to information management systems, it makes them capable of coping with the unforeseen amount of information sources that the open and dynamic nature of the Internet brings, and with the issues that it inevitably implies. This is the reason why information agent technology is inherently interdisciplinary and why it needs to fuse a number of approaches, methods and tools that originated in different research community with diverse aims into a coherent view, e.g., advanced database and knowledge-base systems, distributed information systems, and human-computer interaction (see, for example, [8]).

The main objective of information agent technology is to supply methods and tools for the effective realization of *intelligent information agents*, i.e., software entities capable of providing value-added interfaces to multiple, heterogeneous information sources that are possibly spread across a network. The type, size and topology of the network are irrelevant and intelligent information agents promote a scalable and decentralized approach that copes with networks of information sources ranging from local Intranets to the whole Internet.

Together with specific features discussed below, intelligent information agents inherit some characterizing features of software agents. The very basic feature that they take over from software agents is that they are not simply a concrete development technology; rather they entail a set of abstractions and metaphors that are not necessarily related to how they are concretely realized. This is the reason why we can talk about intelligent software agents while modelling an information management system at different stages of the development cycle and with different levels of detail. We say that the main features that characterize intelligent information agents are the skills that they have and their functional and non-functional properties, rather than the agent-oriented technique used in their realization.

However, skilled intelligent information agents exhibit their autonomous behaviour by means of a synergic combination of: *(i)* pro-activity, i.e., they are goal-directed rather than statically programmed and they can anticipate future information needs while bringing about their goals; *(ii)* timely reactivity, i.e., they keep information sources under surveillance and they timely react to relevant changes; and *(iii)* social behaviour, i.e., they share goals among groups and they compete and/or cooperate in order to fulfil them.

The main tasks that intelligent information agents perform are intended to anticipate future queries from users and/or other agents and to maintain personalized views of information sources in order to facilitate information fruition. Such tasks require equipping intelligent information agents with notable skills, e.g., retrieve, analyze and fuse information from heterogeneous sources into coherent views that anticipate future information needs. Moreover, most advanced intelligent information agents are also in charge of providing personalized user experiences and they have advanced skills in presenting their views of information to users and other agents.

Many systems of intelligent information agents have been realized and deployed both in academic and in industrial settings and intelligent information agents are nowadays best practices for large-scale, decentralized information management systems in industrial control, Internet search, personal assistance, network management, games, and many others application areas (see, e.g., [8]).

The literature proposes a classification of such a wide variety of systems [6] on the basis of the actual skills and on the characteristics of agents as follows:

1. Cooperative: if agents share goals and can cooperatively bring about such goals, possibly exploiting cooperation as a means of optimization;
2. Adaptive: if agents' behaviour can change according to changes in the work environment, which comprises information sources, and/or other agents;
3. Rational: if agents' behaviour is designed to explicitly maximize a measurable utility, which can possibly depend on the behaviour of other agents; and
4. Mobile: if agents travel in the network to get close to information sources in order to, e.g., minimize the amount of data transmitted over the communication network or balance the workload across the network.

The sketched classification supports the identification of key skills that characterize intelligent information agents, as shown in Figure 1. Not all agents exhibit all such skills and not all skills are needed to turn an agent into an intelligent information agent, but such a schematic representation shows the main topics we need to get in touch when dealing with advanced intelligent information agents.

Among the skills shown in Figure 1, the agent research community have been working hardly on the communication skills because *(i)* they are the very basic vehicle for decentralized cooperation, and *(ii)* because they are needed to support open, decentralized multi-agent systems as some standardization body envisaged [5]. The research on communication skills provided solid results, e.g., JADE [2,3], that are nowadays considered consolidated baselines for comparisons. Next section presents a step ahead in this area of research, i.e., a novel tool to support distribution and communication in a multi-agent system that capitalizes and extends the decennial research on JADE.

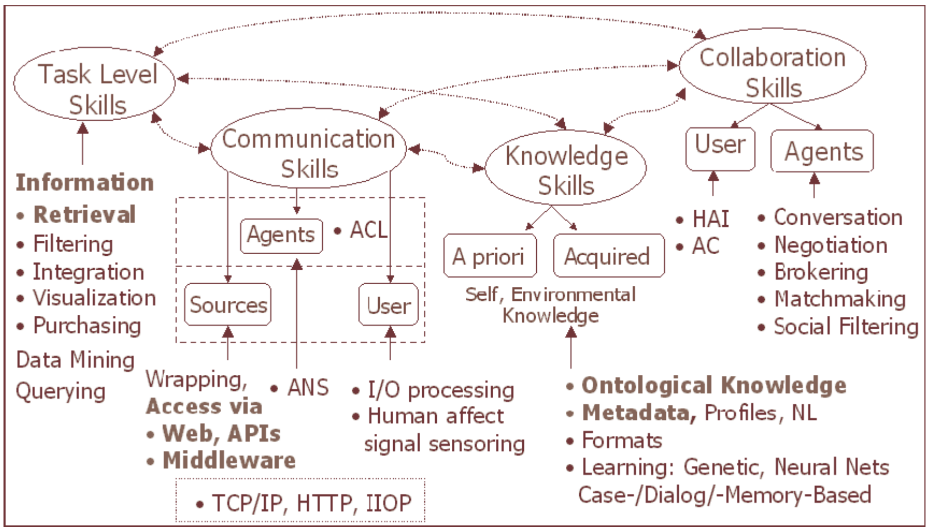

Fig. 1. Basic skills of an information agent (from [7]).

3 HDS

HDS (Heterogeneous Distributed System) is a software framework that aims at 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 upon request from other processes by composing (if necessary) the services offered by other processes through synchronous messages. While both servers and actors may directly take advantage of the services provided by other kinds of application, only the servers provides services to external applications by simply offering public interfaces (one or even more).

Actors and servers can be distributed on a heterogeneous network of computational nodes, the so called *runtime nodes*, for the realization of different kinds of application (see Figure 2). In particular, actors and servers are grouped into runtime nodes that realize a platform. An application can be obtained by combining pre-existent applications by realizing a federation.

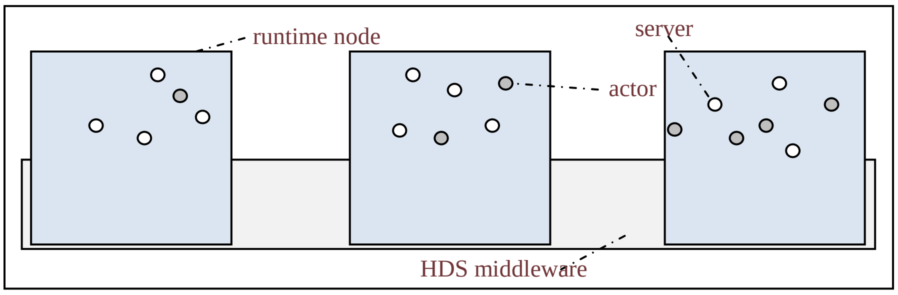

Fig. 2. The architecture of an HDS distributed system.

3.1 Software Architecture Model

The software architecture of an HDS application can be described through the following four different models: *(i)* the *concurrency model*, that describes how the processes of a runtime node can interact and share resource; *(ii)* the *runtime model*, that describes the services available for managing the processes of an application; *(iii)* the *distribution model*, that describes how the processes of different runtime nodes can communicate; and *(iv)* the *communication model* that provides a simple model for defining the data structures that are used as contents of messages.

The concurrency model is used for both defining the processes that realize an application and for supporting their interaction. This model is based on four main elements: *process*, *selector*, *message* and *filter*.

A process is a computational unit able to perform one or more tasks taking advantage, if necessary, of the tasks provided by other processes. To facilitate the cooperation among processes, a process advertises itself by making its description available to the other processes. 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 a process that can have a proactive behaviour and so it can start the execution of tasks without any request from other processes. A server is a process that is only able to perform tasks in response of the request of other processes.

A process interacts 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 action 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 only respond to the requests of other processes in case serving them by 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 interaction) and it can forward a request to another server.

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

Moreover, a process can take advantage of some special objects, called *selectors*, to discover the identifiers of the other processes of the application. A process can ask the runtime registry service for: *(i)* the identifiers of the runtime nodes of the application; *(ii)* the identifiers of the processes running on a runtime node of the application; and *(iii)* the identifiers of the subset of the processes of a runtime node that provide some specific feature. In fact, a selector allows the definition of some constraints on the features of the processes of the application, e.g., the process must be of a specific type or the process must be located in a specific runtime node, and the runtime registry service is able to apply such constraints on the registered processes. Selectors are also used by the processes for extracting from their input message queue the message they need for completing the current task. In this case, the selector defines some constraints on the features that are necessary for identifying the waited message, e.g., the reply to a message sent by the process or a message with a specific kind of content.

A message contains the typical information used for exchanging data on the network, 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 process can communicate with all the other processes and the act of sending messages does not involve any operation that is not related to the delivery of messages to the destination; however, the presence of one or more filters can modify the normal delivery of messages.

A filter is a *composition filter* [4] whose primary aim 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 process has two lists of filters: the ones of the first list, called *input filters*, are applied to the input messages and the others, called *output filters*, are applied to the output messages. Figure 3 shows the flow of the messages from the input filters to the output filters. When a new message arrives or must be sent, the filters of the appropriate list are applied to it in sequence; such a message is stored in the input queue or it is sent only if all the filters succeed.

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*, *dispatcher* and *filterer*.

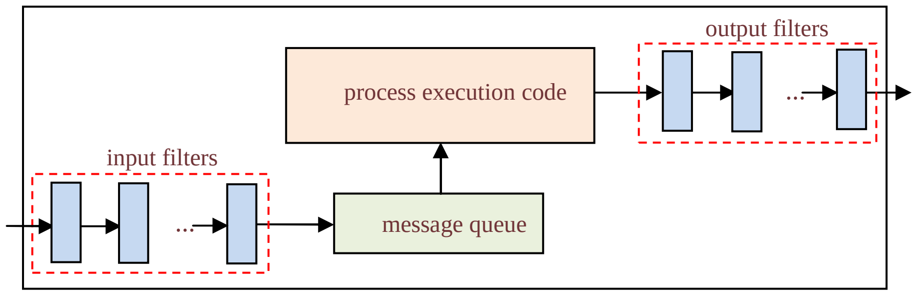

Fig. 3. Flow of the messages inside an HDS process.

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 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 processer is a runtime service that has the duty of creating new processes in the local runtime node. The creation is performed on the basis of the qualified name of the class implementing the process and a list of initialization parameters.

A dispatcher is a runtime service that allows the exchange of messages among both local and remote processes. In fact, it maps local identifiers to the corresponding process message queue and remote identifiers to the corresponding connection.

The lists of message filters cannot be directly modified by the processes, but they can do it by 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, and also for modifying the lists of message filters associated with the other processes of the local runtime node.

The distribution model has the goal of defining the software infrastructure that enables 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. The distributor manages connections that can be realized with different kinds of communication technology through the use of different connectors (see Figure 4). Moreover, a pair of runtime nodes can be connected through different connections.

A connector is a connection 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.

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.

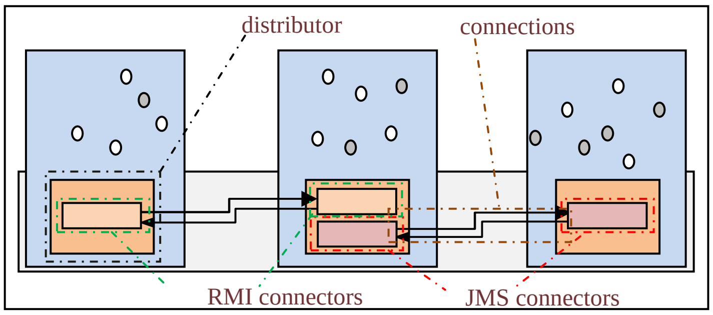

Fig. 4. An application based on three runtime nodes connected through RMI and JMS technologies.

The communication model has the goal of specifying the messages necessary for realizing the interaction protocols used in an application. In particular, it provides a very simple model for defining the data structures that are shared by the processes of an application and then used as contents of messages. This model is based on two elements: *entity* and *action*. An action is an entity describing a task that can be executed by a process and that might be used by an actor for the execution of such a task to another process.

3.2 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, i.e., 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 [11] and JMS [10] 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)* actor and server classes are used for the implementation of the processes involved in the application; *(ii)* description selector classes are used for the discovery of the processes involved in common tasks; *(iii)* message filter classes are used for customizing the communication among the processes; *(iv)* typed messages are used in the interaction among the processes; and *(v)* artefacts (i.e., Java classes and/or configuration files) are used for the deployment of the runtime nodes, of the communication channels among runtime nodes, and for the start-up of the initial sets of processes and message filters.

Such a second module contains: *(i)* 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 messages useful for realizing typical communication protocols used in distributed applications; and *(iii)* a set of software components (realized through an abstract class called `AbstractRunner`) that allow the bootstrap of the runtime nodes of an application and their initialization. With respect to messages and the related communication protocols, the software framework, taking advantage of the communication model, provides the basic actions and entities for realizing application dependent client-server protocols, for supporting group communication, for realizing the publish-subscribe protocol, and for implementing FIPA ACL performatives and interaction protocols [5].

4 Using HDS for the Realization of Pervasive Information Systems

One of the main reasons why we designed and implemented HDS is to have a software framework that couples flexible and performable distributed software components with the coordination techniques provided by multi-agent systems, for realizing the sharing of information within virtual communities and organizations. Such a need derives from the fact that we have been working on the sharing of information and on collaboration within virtual communities and organizations for about ten years taking advantaged of JADE [2,3], probably the most well known and used multi-agent development framework. The use of such a framework and of agent interoperability standards, i.e., FIPA, allowed us to quickly realize very advanced prototypes, but the simple addition of few new features required a lot of time spent in implementing the behaviours of agents, and long validation trials. Moreover, the performance of the realized systems was often poor. Therefore, when a first stable implementation of the HDS software framework became available, we started to work on the implementation a peer-to-peer system that supports the sharing of information and the collaboration among the members of virtual communities by redesigning and extending the last system we realized with JADE. Such a system, known as *RAIS (Remote Assistant for Information Sharing)* [13], is a peer-to-peer multi-agent system supporting the sharing of information among a community of users connected through the Internet.

RAIS offers search facility similar to Web search engines, but it avoids the burden of publishing the information on the Web and it guaranties a controlled and dynamic access to information. The use of agents in such a system is very important because it simplifies the realization of the three main services: *(i)* the filtering of the information coming from different users on the basis of the previous experience of the local user; *(ii)* the pushing of the new information that can be of possible interest for a user; and *(iii)* the delegation of access capabilities on the basis of a network of reputation built by the agents on the community of users. RAIS is composed of a dynamic set of agent platforms connected through the Internet.

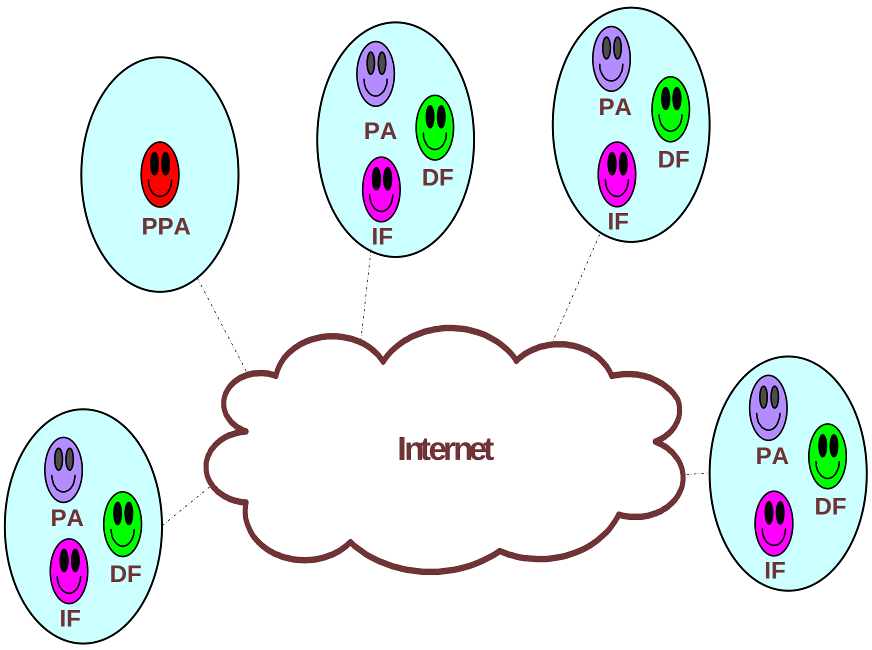

Fig. 5. Architecture of the JADE-based RAIS system.

Each agent platform acts as a peer of the system and it is based on three kinds of agents: a Personal Assistant (PA), an Information Finder (IF) and a Directory Facilitator (DF). Another agent, called Personal Proxy Assistant (PPA), allows a user to access her/his agent platform from a remote system. Figure 5 shows the architecture of the JADE-based RAIS system.

Given the JADE-based architecture of the first implementation of the RAIS system, we designed a new release of the system maintaining a similar architecture, but *(i)* using a more efficient way to exchange data, which is based on the use of messages via the HDS communication model; and *(ii)* ensuring system flexibility and evolution through the use of HDS filters. Figure 6 shows the architecture of the HDS-based RAIS system.

In the JADE-based implementation of the RAIS system we spent a lot of work when we tried to add new services. For example, this happened when we added the possibility to exchange documents through the use of encrypted messages and to allow the agent of a platform to exchange information with the agents of other platforms without requiring the authorization of their users on the basis of the reputation of such agents. The management of these two services have been introduced in the new implementation of the RAIS system by adding in each platform a new agent, called Trust and Security Manager (TSM), that monitors the exchange of messages for building the reputation profiles of the other platforms/users and provides an graphical user interface to its user to view and modify the reputation profiles of the known platform, for defining the access rules for the information she/he like to share with other users and for identifying for which pieces of information an encrypted transmission is required.

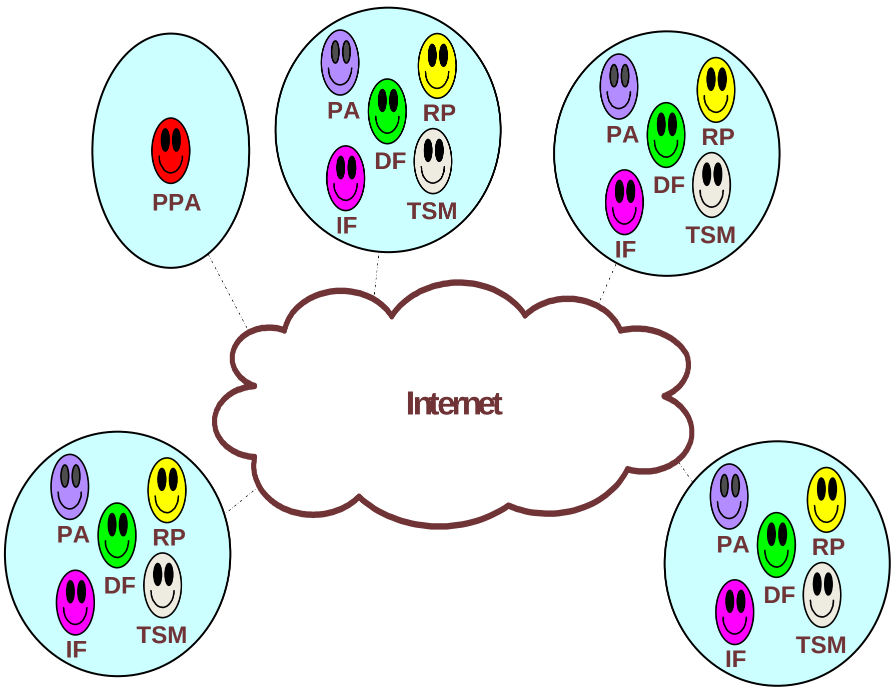

Fig. 6. Architecture of the HDS-based RAIS system.

The introduction of such a type of agent did not require any modification in the other agents of the system. In fact, a TSM agent: *(i)* acts on the other agents of its platform by adding them filters for modifying the flow of messages from/to the agents of the other platforms; and *(ii)* negotiates with the TSM agents of the other platform when an operation (e.g., the exchange of encrypted messages) requires the introduction of some filters in the agents of the other platforms.

Moreover, in addition to the new TSM agent, we introduced a new agent, called Request Propagator (RP), able to propagate a request received by the IF agent of its platform to the IF agents of some other platforms. Also in this case, the introduction of this type of agent did not require any modification to the other agents of the systems. The duty of an RP agent is to propagate a request if it believes that some other platform might have information for satisfying such a request. To do it, a RP agent builds an information profile for each known platform by adding to its PA agent a filter that captures a copy of each message exchanged by the PA agent with the IF of the other platforms.

5 Conclusions

This paper presents a novel tool to develop large-scale, decentralized information management systems. Such a tool, namely HDS, provides effective support for the concrete realization of the communication skills of intelligent information agents, which are among the most important skills that characterize an agent of such a kind. After a brief introduction to intelligent information agents, which serve as a background theme for this research, we presented the structure of HDS and some of its features. Then, we showed how HDS was applied to support the distribution needs of the RAIS system, a real-world information management system previously built on top of JADE.

HDS is a software framework designed for the realization of any kind of distributed system. Some of its functionalities derive from the one offered by JADE; such a 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 of the MAS community. HDS 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.

The experimentations on the RAIS system provided interesting results in terms of the effort and the skills needed to implement new features in the systems. In particular, the migration of the system to HDS allowed the fast addition of two kinds of agents with no modifications to the other kinds of agents. This allowed to quickly accomplishing the task of integrating these new kinds of agents and it also required no further testing and validation to the consolidated parts of the system. Unfortunately, such a good result holds when additional agents are only interested in the communication and it may not be so easy when additional agents have functional dependencies upon existing agents. Nonetheless, we believe that the simple APIs of HDS would simplify the development of generic additional agents in comparison to the effort needed to develop them with JADE.

The research presented in this paper has various lines of future development. Current and future research activities on HDS are dedicated, besides to continue the experimentation and validation in the realization of collaborative services for social network, to the improvement of the HDS software framework itself. 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 [9] and by PCOM [1], *(ii)* the automatic creation of the Java classes representing the typed messages from OWL ontologies taking advantage of the O3L software library [12], and *(iii)* the extension of the software framework with a high-performance software library to support the communication between remote processes, i.e., MINA (*http://mina.apache.org*).

Finally, we intend to take advantage of the experience of porting RAIS to HDS by identifying guidelines and a structured methodology to support developers in similar migrations.

Acknowledgments

This work is partially supported by the Italian Ministry MIUR (Ministero dell’Istruzione, dell’Università e della Ricerca).

References

1. Becker, C., Hante, M., Schiele, G. and Rotheemel, K. PCOM – A component system for pervasive computing. In *Procs. 2nd IEEE Conf. Pervasive Computing and Communications* (PerCom 2004), Orlando, FL, 67-76, 2004.
2. Bellifemine, F. Poggi, A., Rimassa, G. Developing multi agent systems with a FIPA-compliant agent framework. *Software - Practice & Experience*, 31, 103-128 (2001)
3. Bellifemine, F., Caire, G., Poggi, A., Rimassa, G. JADE: a Software Framework for Developing Multi-Agent Applications. Lessons Learned. *Information and Software Technology*, 50, 10-21 (2008)
4. Bergmans, L., Aksit, M. Composing crosscutting concerns using composition filters. *Communications of the ACM*, 44(10):51—57, ACM Press (2001)
5. FIPA Specifications, *http://www.fipa.org* (2000)
6. Klusch, M. *Intelligent Information Agents*, Springer, Berlin (1999)
7. Klusch, M. Information Agent Technology for the Internet: A survey. *Data & Knowledge Engineering*, 36(3):337-372, Elsevier Science B.V. (2001)
8. Klusch, M., Bergamaschi, S., Edwards, P., and Petta, P. (eds.) *Intelligent Information Agents: The AgentLink Perspective*, Lecture Notes in Computer Science, 2586, Springer, Berlin (2003)
9. Kumar, M., Shirazi, B. A., Das, S. L., Sung, B. Y., Levine, D., and Singhal, M. PICO: A Middleware Framework for Pervasive Computing. *IEEE Pervasive Computing*, 2(3):72-79, IEEE Computer Society Press, 2003.
10. Monson-Haefel, R., Chappell, D. *Java Message Service*. O’Reilly & Associates (2000)
11. Pitt, E. McNiff, K. *java.rmi: The Remote Method Invocation Guide*. Addison-Wesley (2001)
12. Poggi, A. Developing Ontology Based Applications with O3L. *WSEAS Trans. on Computers*, 8(8):1286-1295, 2009.
13. Poggi, A., Tomaiuolo, M. Integrating Peer-to-Peer and Multi-Agent Technologies for the Realization of Content Sharing Applications. In Vargiu, E., Soru, A. (eds.), *Information Retrieval and Mining in Distributed Environments*. pp. 93-107, Springer, Berlin, Germany (2010)