FUSION@, A SOA-Based Multi-Agent Architecture

Dante I. Tapia, Sara Rodríguez, Javier Bajo, Juan M. Corchado

Departamento Informática y Automática

Universidad de Salamanca

Plaza de la Merced s/n, 37008, Salamanca, Spain

{dantetapia; srg; jbajope; corchado}@usal.es

Abstract. This paper presents a multi-agent architecture which facilitates the integration of distributed services and applications to optimize the construction of multi-agent systems. The architecture proposes a new and easier method to develop distributed multi-agent systems, where applications and services can communicate in a distributed way, even from mobile devices, independent of a specific programming language or operating system. The core of the architecture is a group of deliberative agents acting as controllers and administrators for all applications and services. The functionalities of the agents are not inside their structure, but modelled as services. This approach provides a higher ability to recover from errors and a better flexibility to change the agents' behaviour at execution time.

Keywords: Multi-Agent Systems, Services Oriented Architectures, Distributed Computing.

1 Introduction

The continuous development of software and systems requires creating increasingly complex and flexible applications, so there is a trend toward reusing resources and share compatible platforms or architectures. In some cases, applications require similar functionalities already implemented into other systems which are not always compatible. At this point, developers can face this problem through two options: reuse functionalities already implemented into other systems; or redeploy the capabilities required, which means more time for development, although this is the easiest and safest option in most cases. While the first option is more adequate in the long run, the second one is most chosen by developers, which leads to have replicated functionalities as well as greater difficulty in migrating systems and applications. Moreover, the absence of a strategy for integrating applications generates multiple points of failure that can affect the systems' performance. This is a poorly scalable and flexible model with reduced response to change, in which applications are designed from the outset as independent software islands.

This paper describes a *Flexible User and ServIces Oriented multi-ageNt Architecture* (FUSION@). One of the most important characteristics is the use of intelligent agents as the main components in employing a service oriented approach, focusing on distributing the majority of the systems' functionalities into remote and local services and applications. The architecture proposes a new and easier method of building distributed multi-agent systems, where the functionalities of the systems are not integrated into the structure of the agents, rather they are modelled as distributed services and applications which are invoked by the agents acting as controllers and coordinators.

Agents have a set of characteristics, such as autonomy, reasoning, reactivity, social abilities, pro-activity, mobility, organization, etc. which allow them to cover several needs for dynamic environments, especially ubiquitous communication and computing and adaptable interfaces. Agent and multi-agent systems have been successfully applied to several scenarios, such as education, culture, entertainment, medicine, robotics, etc. [6], [15]. The characteristics of the agents make them appropriate for developing dynamic and distributed systems as they possess the capability of adapting themselves to the users and environmental characteristics [8]. The continuous advancement in mobile computing makes it possible to obtain information about the context and also to react physically to it in more innovative ways [8]. The agents in this architecture are based on the deliberative Belief, Desire, Intention (BDI) model [9], [3], [12], where the agents' internal structure and capabilities are based on mental aptitudes, using beliefs, desires and intentions [7]. Nevertheless, complex systems need higher adaptation, learning and autonomy levels than pure BDI model [3]. This is achieved in FUSION@ by modelling the agents' characteristics to provide them with mechanisms that allow solving complex problems and autonomous learning.

FUSION@ has been designed to facilitate the development of distributed multi-agent systems. Agents have the ability to dynamically adapt their behaviour at execution time. FUSION@ provides an advanced flexibility and customization to easily add, modify or remove applications or services on demand, independently of the programming language. It also formalizes the integration of applications, services, communications and agents.

In the next section, the specific problem description that essentially motivated the development of FUSION@ is presented. Section 3 describes the main characteristics of this architecture and briefly explains some of its components. Section 4 presents the results and conclusions obtained.

2 Problem Description and Background

Excessive centralization of services negatively affects the systems' functionalities, overcharging or limiting their capabilities. Classical functional architectures are characterized by trying to find modularity and a structure oriented to the system itself. Modern functional architectures like Service-Oriented Architecture (SOA) consider integration and performance aspects that must be taken into account when functionalities are created outside the system. These architectures are aimed at the interoperability between different systems, distribution of resources, and the lack of dependency of programming languages [5]. Services are linked by means of standard communication protocols that must be used by applications in order to share resources in the services network [1]. The compatibility and management of messages that the services generate to provide their functionalities is an important and complex element in any of these approaches.

One of the most prevalent alternatives to these architectures is agents and multi-agent systems technology which can help to distribute resources and reduce the central unit tasks [1]. A distributed agents-based architecture provides more flexible ways to move functions to where actions are needed, thus obtaining better responses at execution time, autonomy, services continuity, and superior levels of flexibility and scalability than centralized architectures [4]. Additionally, the programming effort is reduced because it is only necessary to specify global objectives so that agents cooperate in solving problems and reaching specific goals, thus giving the systems the ability to generate knowledge and experience. Unfortunately, the difficulty in developing a multi-agent architecture is higher and because there are no specialized programming tools to develop agents, the programmer needs to type a lot of code to create services and clients [14]. It is also necessary to have a more complex system analysis and design, which implies more time to reach the implementation stage. Moreover, the system control is reduced because the agents need more autonomy to solve complex problems. The development of agents is an essential piece in the analysis of data from distributed sensors and gives those sensors the ability to work together and analyze complex situations, thus achieving high levels of interaction with humans [11].

Agent and multi-agent systems combine classical and modern functional architecture aspects. Multi-agent systems are structured by taking into account the modularity in the system, and by reuse, integration and performance. Nevertheless, integration is not always achieved because of the incompatibility among the agents' platforms. The integration and interoperability of agents and multi-agent systems with SOA and Web Services approaches has been recently explored [1]. Some developments are centred on communication between these models, while others are centred on the integration of distributed services, especially Web Services, into the structure of the agents. Bonino da Silva, et al. [2] propose merging multi-agent techniques with semantic web services to enable dynamic, context-aware service composition. They focus on SOA in designing a multi-agent service composition as an intelligent control layer, where agents discover services and adapt their behaviour and capabilities according to semantic service descriptions. Ricci et al. [13] have developed a java-based framework to create SOA and Web Services compliant applications, which are modelled as agents. Communication between agents and services is performed by using what they call “artifacts” and WSDL (Web Service Definition Language). Shafiq et al. [16] propose a gateway that allows interoperability between Web Services and multi-agent systems. This gateway is an agent that integrates Foundation for Intelligent Physical Agents (FIPA) and The World Wide Web Consortium (W3C) specifications, translating Agent Communication Language (ACL), SOAP and WSDL messages, and combines both directories from agents’ platforms and web services. Li et al. [10] propose a similar approach, but focus on the representation of services. They use SOAP and WSDL messages to interact with agents. Walton [18] presents a technique to build multi-agent systems using Web Services, defining a language to represent the dialogs among agents. There are also frameworks, such as Sun’s Jini and IBM’s WebSphere, which provide several tools to develop SOA-based systems. Jini uses Java technology to develop distributed and adaptive systems over dynamic environments. Rigole et al. [14] have used Jini to create agents on demand into a home automation system, where each agent is defined as a service in the network. WebSphere provides tools for several operating systems and programming languages. However, the systems developed using these frameworks are not open at all because the framework is closed and services and applications must be programmed using a specific programming language that support their respective proprietary APIs.

Although these developments provide an adequate background for developing distributed multi-agent systems integrating a service oriented approach, most of them are in early stages of development, so it is not possible to actually know their potential in real scenarios. FUSION@ has an advantage regarding development because we have already implemented it into a real scenario. In addition, FUSION@ not only provides communication and integration between distributed agents, services and applications; it also proposes a new method to facilitate the development of distributed multi-agent systems by means of modelling the functionalities of the agents and the systems as services. Another feature in this architecture is security, which is managed by the agents. All communications must take place via the agents, so services cannot share their resources unless the agents allow it. Besides, services defined for each system must always be available, so they are not shared with other systems unless it is specified.

3 FUSION@, A SOA-based Multi-Agent Architecture

FUSION@ is a modular multi-agent architecture, where services and applications are managed and controlled by deliberative BDI (Belief, Desire, Intention) agents [9], [3], [12]. Deliberative BDI agents are able to cooperate, propose solutions on very dynamic environments, and face real problems, even when they have a limited description of the problem and few resources available. These agents depend on beliefs, desires, intentions and plan representations to solve problems [7]. Deliberative BDI agents are the core of FUSION@. There are different kinds of agents in the architecture, each one with specific roles, capabilities and characteristics. This fact facilitates the flexibility of the architecture in incorporating new agents. As can be seen on Figure 1, FUSION@ defines four basic blocks which provide all the functionalities of the architecture.

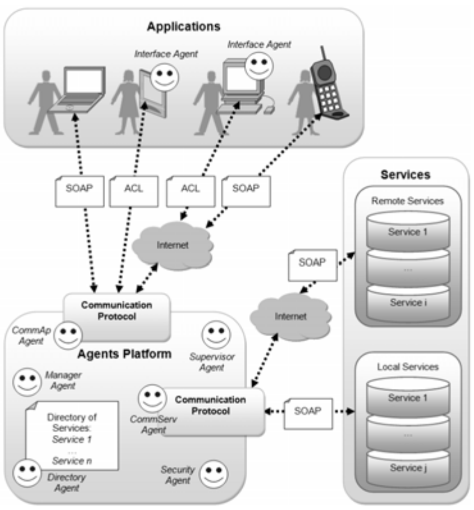

Fig. 1. FUSION@ model

- Applications. These represent all the programs that can be used to exploit the system functionalities. They can be executed locally or remotely, even on mobile devices with limited processing capabilities, because computing tasks are largely delegated to the agents and services.
- Agents Platform. This is the core of FUSION@, integrating a set of agents, each one with special characteristics and behaviour. An important feature in this architecture is that the agents act as controllers and administrators for all applications and services, managing the adequate functioning of the system, from services, applications, communication and performance to reasoning and decision-making.
- Services. They are the bulk of the functionalities of the system at the processing, delivery and information acquisition levels. Services are designed to be invoked locally or remotely. Services can be organized as local services, web services, or even as individual stand alone services.
- Communication Protocol. This allows applications and services to communicate directly with the agents platform. The protocol is completely open and independent of any programming language. This protocol is based on SOAP specification to capture all messages between the platform and the services and applications [5]. All external communications follow the same protocol, while

the communication among agents in the platform follows the FIPA Agent Communication Language (ACL) specification. Applications can make use of agents platforms to communicate directly (using FIPA ACL specification) with the agents in FUSION@, so while the communication protocol is not needed in all instances, it is absolutely required for all services.

There are pre-defined agents which provide the basic functionalities of FUSION@:

- CommApp Agent. This agent is responsible for all communications between applications and the platform. It manages the incoming requests from the applications to be processed by services. It also manages responses from services (via the platform) to applications. All messages are sent to Security Agent for their structure and syntax to be analyzed.
- CommServ Agent. It is responsible for all communications between services and the platform. The functionalities are similar to CommApp Agent but backwards. Manager Agent signals to CommServ Agent which service must be invoked. All messages are sent to Security Agent for their structure and syntax to be analyzed. This agent also periodically checks the status of all services to know if they are idle, busy, or crashed.
- Directory Agent. It manages the list of services that can be used by the system. For security reasons [17], FUSION@ does not include a service discovery mechanism, so applications must use only the services listed in the platform. However, services can be added, erased or modified dynamically. There is information that is constantly being modified: the service performance (average time to respond to requests), the number of executions, and the quality of the service (QoS). This last data is very important, as it assigns a value between 0 and 1 to all services. All new services have a quality of service (QoS) value set to 1. This value decreases when the service fails (e.g. service crashes, no service found, etc.) or has a subpar performance compared to similar past executions. QoS is increased each time the service efficiently processes the tasks assigned.
- Supervisor Agent. This agent supervises the correct functioning of the other agents in the system. Supervisor Agent periodically verifies the status of all agents registered in the architecture by sending ping messages. If there is no response, the Supervisor agent kills the agent and creates another instance of that agent.
- Security Agent. This agent analyzes the structure and syntax of all incoming and outgoing messages. If a message is not correct, the Security Agent informs the corresponding agent (CommApp or CommServ) that the message cannot be delivered. This agent also directs the problem to the Directory Agent, which modifies the QoS of the service where the message was sent.
- Manager Agent. Decides which agent must be called by taking into account the QoS and users preferences. Users can explicitly invoke a service, or can let the Manager Agent decide which service is best to accomplish the requested task. This agent also checks if services are working properly. It requests the Comm-

Serv Agent to send ping messages to each service on a regular basis. If a service does not respond, CommServ informs Manager Agent, which tries to find an alternate service, and informs the Directory Agent to modify the respective QoS.

- Interface Agent. This kind of agent was designed to be embedded in users' applications. Interface agents communicate directly with the agents in FUSION@ so there is no need to employ the communication protocol, rather the FIPA ACL specification. The requests are sent directly to the Security Agent, which analyzes the requests and sends them to the Manager Agent. These agents must be simple enough to allow them to be executed on mobile devices, such as cell phones or PDAs. All high demand processes must be delegated to services.

FUSION@ is an open architecture that allows developers to modify the structure of these agents, so that agents are not defined in a static manner. Developers can add new agent types or extend the existing ones to conform to their projects needs. However, most of the agents' functionalities should be modelled as services, releasing them from tasks that could be performed by services. Services represent all functionalities that the architecture offers to users and uses itself. As previously mentioned, services can be invoked locally or remotely. All information related to services is stored into a directory which the platform uses in order to invoke them, i.e., the services. This directory is flexible and adaptable, so services can be modified, added or eliminated dynamically. Services are always on "listening mode" to receive any request from the platform. It is necessary to establish a permanent connection with the platform using sockets. Every service must have a permanent listening port open in order to receive requests from the platform. Services are requested by users through applications, but all requests are managed by the platform, not directly by applications. When the platform requests a service, the CommServ Agent sends an XML message to the specific service. The message is received by the service and creates a new thread to perform the task. The new thread has an associated socket which maintains communication open to the platform until the task is finished and the result is sent back to the platform. This method provides services the capability of managing multiple and simultaneous tasks, so services must be programmed to allow multi-threading. However, there could be situations where multi-tasks will not be permitted, for instance high demanding processes where multiple executions could significantly reduce the services performance. In these cases, the Manager Agent asks the CommServ Agent to consult the status of the service, which informs the platform that it is busy and cannot accept other requests until finished. The platform must then seek another service that can handle the request, or wait for the service to be idle. To add a new service, it is necessary to manually store its information into the directory list managed by the Directory Agent. Then, CommServ Agent sends a ping message to the service. The service responds to the ping message and the service is added to the platform. A service can be virtually any program that performs a specific task and shares its resources with the platform. These programs can provide methods to access data bases, manage connections, analyze data, get information from external devices (e.g. sensors, readers, screens, etc.), publish information, or even make use of other services. Developers have are free to use any programming language. The only requirement is that they must follow the communication protocol based on transactions of XML (SOAP) messages.

4 Results and Conclusions

Several tests have been done to demonstrate if the FUSION@ approach is appropriate to distribute resources and optimize the performance of multi-agent systems. Most of these tests basically consist on the comparison of two simple configurations (System A and System B) with the same functionalities. These systems are specifically designed to schedule a set of tasks using a planning mechanism [6]. System A integrates this mechanism into a deliberative BDI agent, while System B implements FUSION@, modelling the planning mechanism as a service.

A task is a java object that contains a set of parameters (TaskId, MinTime, MaxTime, ScheduleTime, UserId, etc.). ScheduleTime is the time in which a specific task must be accomplished, although the priority level of other tasks needing to be accomplished at the same time is factored in. The planning mechanism increases or decreases ScheduleTime and MaxTime according to the priority of the task: ScheduleTime=ScheduleTime-5min*TaskPriority and MaxTime=MaxTime+5min*TaskPriority

To generate a new plan (i.e. scheduling), an automatic routine sends a request to the agent. In System A, the agent processes the request and executes the planning mechanism. On the other hand, System B makes use of FUSION@, so the request is processed by the Manager Agent which decides to use the planner service (i.e. the planning mechanism modelled as a service). The platform invokes the planner service which receives the message and starts to generate a new plan. Then, the solution is sent to the platform which delivers the new plan to the corresponding agent. Table 1 shows an example of the results delivered by the planning mechanism for both systems.

Table 1. Example of the results delivered by the planning mechanism

Time	Activity
19:21	Exercise
20:17	Walk
22:00	Dinner

An agenda is a set of non organized tasks that must be scheduled by means of the planning mechanism or the planner service. There were 30 defined agendas each with 50 tasks. Tasks had different priorities and orders on each agenda. Tests were carried out on 7 different test groups, with 1, 5, 10, 15, 20, 25 and 30 simultaneous agendas to be processed by the planning mechanism. 50 runs for each test group were performed, all of them on machines with equal characteristics. Several data have been obtained from these tests, focusing on the average time to accomplish the plans, the number of crashed agents, and the number of crashed services. For System B five planner services with exactly the same characteristics were replicated.

Figure 2 shows the average time needed by both systems to generate the paths for a fixed number of simultaneous agendas. System A was unable to handle 15 simultaneous agendas and time increased to infinite because it was impossible to perform those requests. However, System B had 5 replicated services available, so the workflow was distributed, and allowed the system to complete the plans for 30 simultaneous agendas. Another important data is that although the System A performed slightly faster when processing a single agenda, performance was constantly reduced when new simultaneous agendas were added. This fact demonstrates that the overall performance of System B is better when handling distributed and simultaneous tasks (e.g. agendas), instead of single tasks.

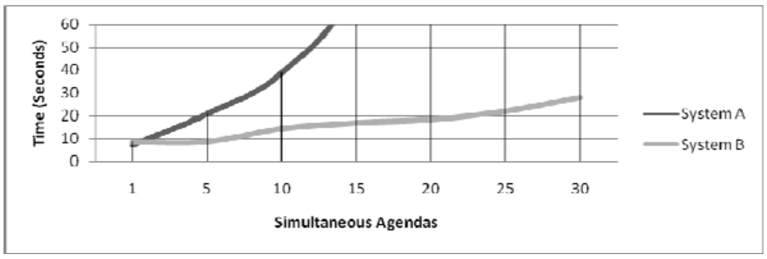

Fig. 2. Time needed for both systems to schedule simultaneous agendas

The architecture presented in this paper proposes an alternative where agents are based on the BDI (Belief, Desire, Intention) model and act as controllers and coordinators. FUSION@ exploits the agents' characteristics to provide a robust, flexible, modular and adaptable solution that covers most of the requirements of a wide diversity of projects. All functionalities, including those of the agents, are modelled as distributed services and applications. By means of the agents, the systems are able to modify their behaviour and functionalities at execution time. Developers can create their own functionalities with no dependency on any specific programming language or operating system.

Results demonstrate that FUSION@ is adequate for distributing composite services and optimizing performance for multi-agent systems. Future work consists on applying this architecture into composite multi-agent systems, as well as extending the experiments to obtain more decisive data.

Acknowledgements. This work has been partially supported by the TIN2006-14630-C03-03 and the IMSERSO 137/07 projects. Special thanks to Tulecom for the technology provided and the know-how supported.

References

1. Ardissono, L., Petrone, G. and Segnan, M. 2004. A conversational approach to the interaction with Web Services. Computational Intelligence, Blackwell Publishing. Vol. 20. pp. 693-709.
2. Bonino da Silva, L.O, Ramparany, F., Dockhorn, P., Vink, P., Etter, R. and Broens, T. 2007. A Service Architecture for Context Awareness and Reaction Provisioning. IEEE Congress on Services (Services 2007). pp. 25-32
3. Bratman, M.E., Israel, D. and Pollack, M.E. 1988. Plans and resource-bounded practical reasoning. Computational Intelligence, Blackwell Publishing. Vol. 4. pp. 349-355.
4. Camarinha-Matos, L.M. and Afsarmanesh, H. 2007. A Comprehensive Modeling Framework for Collaborative Networked Organizations. Journal of Intelligent Manufacturing, Springer Netherlands. Vol. 18(5). pp. 529-542.
5. Cerami, E. 2002. Web Services Essentials Distributed Applications with XML-RPC, SOAP, UDDI & WSDL. O'Reilly & Associates, Inc. 1st Edition.
6. Corchado, J.M., Bajo, J., De Paz, Y. and Tapia, D.I. 2008. Intelligent Environment for Monitoring Alzheimer Patients, Agent Technology for Health Care. Decision Support Systems, Eslevier, Amsterdam, Netherlands. In press.
7. Georgeff, M. and Rao, A. 1998. Rational software agents: from theory to practice. Agent Technology: Foundations, Applications, and Markets, N.R. Jennings and M.J. Wooldridge (Eds), Springer-Verlag, New York, USA.
8. Jayaputera, G.T., Zaslavsky, A.B. and Loke, S.W. 2007. Enabling run-time composition and support for heterogeneous pervasive multi-agent systems. Journal of Systems and Software. Vol. 80(12). pp. 2039-2062.
9. Jennings, N.R. and Wooldridge M. 1995. Applying agent technology. Applied Artificial Intelligence, Taylor & Francis. Vol. 9(4). pp. 351-361.
10. Li, Y., Shen, W. and Ghenniwa, H. 2004. Agent-Based Web Services Framework and Development Environment. Computational Intelligence, Blackwell Publishing. Vol. 20(4). pp. 678-692.
11. Pecora, F. and Cesta, A. 2007. Dcop for smart homes: A case study. Computational Intelligence, Backwell Publishing. Vol. 23(4). pp. 395-419.
12. Pokahr, A., Braubach, L. and Lamersdorf, W. 2003. Jadex: Implementing a BDI-Infrastructure for JADE Agents. In EXP - in search of innovation (Special Issue on JADE), Department of Informatics, University of Hamburg, Germany. pp. 76-85.
13. Ricci, A., Buda, C. and Zaghini, N. 2007. An agent-oriented programming model for SOA & web services. In 5th IEEE International Conference on Industrial Informatics (INDIN'07), Vienna, Austria. pp. 1059-1064.
14. Rigole, P., Holvoet, T. and Berbers, Y. 2002. Using Jini to Integrate Home Automation in a Distributed Software-System. In Revised Papers From the 4th international Workshop on Distributed Communities on the Web (April 03-05, 2002). J. Plaice, P. G. Kropf, P. Schulthess, and J. Slonim (Eds). Lecture Notes In Computer Science. Springer-Verlag, London. Vol. 2468. pp. 291-304.
15. Schön, B., O'Hare, G.M.P., Duffy, B.R., Martin, A.N. and Bradley, J.F. 2005. Agent Assistance for 3D World Navigation. Lecture Notes in Computer Science, Springer. Vol. 1. pp. 499-499.
16. Shafiq, M.O., Ding, Y. and Fensel, D. 2006. Bridging Multi Agent Systems and Web Services: towards interoperability between Software Agents and Semantic Web Services. In Proceedings of the 10th IEEE International Enterprise Distributed Object Computing Conference (EDOC'06). IEEE Computer Society, Washington, DC. pp. 85-96.
17. Snidaro, L. and Foresti, G.L. 2007. Knowledge representation for ambient security. Expert Systems, Blackwell Publishing. Vol. 24(5). pp. 321-333.
18. Walton, C. 2006. Agency and the Semantic Web. Oxford University Press, Inc.