

AGENTSERVICE: A FRAMEWORK FOR DISTRIBUTED MULTI-AGENT SYSTEM DEVELOPMENT

C. Vecchiola, A. Grosso, A. Passadore, and A. Boccalatte
University of Genoa, Via Opera Pia 13, 16145 Genoa, Italy
{christian, passa, agrosso, nino}@dist.unige.it

Abstract – Today, software systems are complex, dynamic, distributed, and evolve over time. Multi-agent systems are a powerful abstraction to model such systems. Agent programming frameworks support software engineers in transforming these abstractions into concrete distributed applications. Even though there is wide availability of tools for multi-agent development, only few of them constitute valid support from a software engineering point of view. AgentService provides a reliable, robust, and modular software infrastructure for multi-agent system development. In this paper, we present the innovative features of the framework; including: the agent model, the scheduling engine, and the runtime environment. These components are discussed in detail and compared with the solution adopted by other frameworks.

Key Words – Agent Oriented Software Engineering, Multi-agent Systems, Software Frameworks

1. Introduction

Agent-based computing encompasses abstractions, methodologies, and tools that characterize the agent-oriented approach. It adopts the concepts of agent and multi-agent system to express software systems. A software agent is a software artifact situated in some environment that is capable of flexible autonomous action in order to meet its design objectives [1]. Multi-agent systems (MASs) are aggregations of socially able agents that cooperate, compete, and negotiate in order to satisfy their goals. Whereas such abstractions are definitively powerful for modeling distributed, dynamic, and open software systems, the corresponding implementations do not often express the same properties.

Agent programming frameworks try to address this lack of expression. They provide software engineers with a model for developing software agents, a run-time environment for their execution, and a set of tools supporting the entire life cycle of agent-based software projects: from development to maintenance. AgentService is a software framework developed to support software engineers in designing and developing multi-agent systems. The core features of the framework are its innovative agent model and the platform run-time environment. The agent model allows for real multi-tasking activity for the agents and is suitable for implementing different agent architectures. The runtime platform provides a flexible and modular software environment able to evolve and adapt to different application contexts.

In this paper, we first review the state of the art of multi-agent system development. Then we describe the framework by highlighting the three most important or innovative aspects of the framework: the agent

model, the scheduling engine, and the support for distributed multi-agent system. A comparative analysis of the discussed features of AgentService with the ones offered by other frameworks is presented and conclusions follow.

2. State of the Art

Agent programming frameworks represent an Software Engineering outcome for agent-based computing. They provide a structured and comprehensive approach encompassing methodologies and tools for modeling, developing, and maintaining multi-agent system.

Although there is wide availability¹ of software tools for multi-agent system development, only few tools have attracted the attention of the academic community. Among these, very few can be considered frameworks. Important projects, such as FIPA-OS [2], are no longer maintained or focused only on a particular aspect of MASs, for example the Tracy toolkit [3] and TuCSon [4]. Among those which can be considered agent-programming frameworks, only three were widely used, are still active, developed, and integrated with new features. They are: JADE (Java Agent Development Environment) [5] and its extensions, AgentFactory [6], and JACK [7].

JADE is probably the most popular and widely used agent programming framework in the agent research community. It provides a management GUI, RAD features, ontology management, and support for agent mobility. JADE agents are software artifacts consisting of by multiple activities called behaviors. Developers simply program behaviors and compose them into agents. JADE has been further extended by other projects. LEAP [8] is a lightweight version of JADE for portable devices. JADEX [9] provides BDI (Believe Desire Intention) support for JADE. WADE [10] is the latest evolution of JADE and allows the development of multi-agent systems in terms of system logics according to the workflow metaphor.

AgentFactory is a modular and extensible framework that provides native support for designing and developing BDI agents. These are a combination of agent programs expressed by using multi-modal temporal logic and a set of *actuators* and *perceptors* through which agents interact with the runtime environment. AgentFactory strongly emphasizes the design and the development phase and comes with a rich set of vis-

¹ Featured and authoritative websites, such as AgentLink (<http://www.agentlink.org>), provide a comprehensive list of resources for multi agent system development.

ual tools for leveraging the modeling process. In particular it is worth noticing the Protocol Tool, which allows users to design interaction protocols by using AUML [11].

JACK Intelligent Agents is a commercial solution delivering complete support for agent design, implementation, deployment, and monitoring. It supports the development of systems composed by BDI agents and teams of agents with a rich set of graphical tools. The interesting feature of JACK is the JACK Agent Language, which is an extension of the Java programming language allowing users to describe agent oriented systems.

In conclusion, it is important to mention—even though not active anymore—the ZEUS toolkit [12]. ZEUS has probably been one of the first software solutions introducing the concept of framework for agent development and it was used as software infrastructure in industry by British Telecom (BT).

3. AgentService Framework

AgentService tackles the problem of multi-agent system development by providing a robust and flexible software infrastructure to implement dynamic multi-agent systems. It focuses on allowing designers and developers to easily compose a wide range of multi-agent systems with minimal implementation costs. The distinctive features of the framework are:

- A simple and intuitive agent model based on the concept of state and activities;
- A highly customizable runtime environment for agents execution characterized by an innovative scheduling engine and a modular platform;
- A complete support for distributed multi-agent system development;

In the following subsections, we describe how AgentService provides an effective and flexible implementation of the pillars of agent based computing that are the agent and the multi-agent systems abstractions. In the next section we then describe how these features have been extended to support distributed multi-agent systems.

3.1 Agent Model and Implementation

Autonomy, multi-tasking, and dynamic state characterize software agents. AgentService provides a software model able to support and easily implement these three features.

The definition of new type of agent—called agent template—is a composition of knowledge objects and behavior objects. Knowledge objects are simple data classes exposing a set of correlated fields that make up a self-consistent unit of knowledge. Behavior objects implement the logic of a specific agent task or an agent role. While knowledge objects identify the state of the software agent, behavior objects identify the concurrent activities performed by the agent.

Agent instances are dynamically created upon user request and the runtime environment creates the agent instances by inspecting the agent template. Each of the agent instances has its own thread of control that takes care of the task execution and of the agent life cycle; other software components, except the platform AMS, cannot control or interfere the execution of agent instances.

This model enforces autonomy because the framework does not allow method invocation on agent instances from software components that are not the platform itself. At the same time support multi-tasking and dynamic state as a result of the aggregate and concurrent execution of behavior objects updating and changing the state of the agent.

3.2 Runtime Environment

The runtime environment is the software infrastructure supporting agent execution, interaction, and management. In the AgentService framework the runtime environment consisting of the agent platform, which is the container of software agents and provides the basic services to them, and by the scheduling engine, which is the system component that takes care of agent execution.

3.2.1 Platform Architecture

The agent platform acts as virtual machine for agents: it controls their instantiation, schedules their activities, provides them communication and localization services, and maintains their state. At the same time it also constitutes the main access point to the MAS since it provides access to all the services offered by the multi-agent system.

Fig. 1 illustrates the components constituting the agent platform. The system has been designed following the FIPA specifications² and by focusing on the aspects that make the platform a robust software infrastructure in terms of security, modularity, openness, and customization.

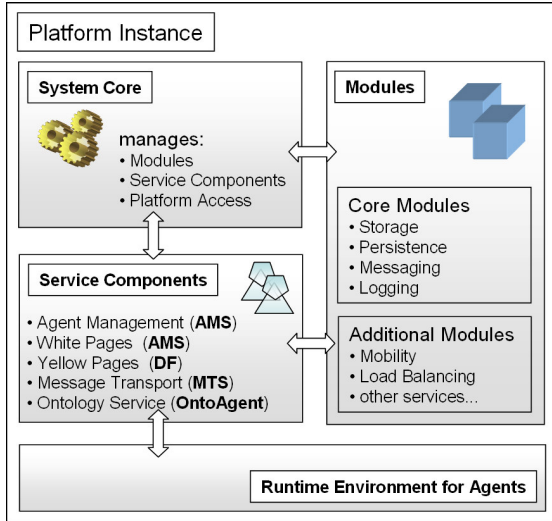


Figure 1: platform architecture.

The System Core is the skeleton of the platform and it is responsible for activating and configuring the platform. It loads all the modules installed in the system and then delegates the control to the FIPA Service Components.

Modules are the design elements used to extend, customize, and tune the agent platform in order to meet the requirements of a specific scenario. Core modules implement fundamental system services that are required by the platform to bootstrap. They provide: a repository for agent definitions, a persistence service, a communication infrastructure, and logging. Additional modules constitute the means to extend the features of the multi-agent system and to seamlessly integrate the new services into the software platform. It is possible to transparently configure and add new modules without changing the platform code base. This feature has been extensively used during the development in order to provide different implementations of core modules and the integration of new services such as the mobility infrastructure. The case of the messaging module is particularly interesting: it has been possible to provide different implementations of the communication infrastructure (Microsoft Message Queue (MSMQ), in process local queues, Microsoft ASP.NET Web Services, and .NET Remoting) with absolutely no changes to the platform or the agent

² FIPA-SPECS, (2002) [Online Documents] Available at: <http://www.fipa.org/repository/standardspecs.html>.

model. This simple example demonstrates how AgentService can be easily tuned to support different scenarios with different performance or security requirements.

The FIPA Service Components are a set of mandatory services that should be available to software agents and implemented as software agents. AgentService is compliant with the FIPA abstract specifications and provides an implementation of the Agent Management System (AMS), the Directory Facilitator (DF), and the Message Transport System (MTS). In order to support effective communication between agents, the framework provides an implementation of the ontology service by means of the Ontology Agent (OntoAgent). This optional component, together with a complete object model for representing, manipulating, and reasoning about ontologies, allows agents to speculate about concepts and to interoperate with other implementations of the FIPA compliant multi-agent platform. Being the maintainer of the knowledge base of the multi-agent system, the Ontology Agent simplifies the interaction of different communities of agents that are aggregated together in the same platform. Agents belonging to different communities can query the Directory Facilitator to discover which agents are hosted in the platform, which services they offer, and which software ontology they support. Then, the Ontology Agent can be queried to obtain a representation of the ontology and discover similarities between different ontologies.

3.2.2 *Scheduling Engine*

The scheduling engine is the most innovative feature of the framework. It controls the execution of software agents and it is directly responsible for the performance, the robustness, and the security of the platform. The scheduling engine is the result of the coordinated activity of three components: *Agent Factory*, *Agent Scheduler*, and *Behavior Scheduler*.

The Agent Factory is responsible for the agent instances creation and it is one controlling their isolation level. Common implementations of these components in other frameworks rely on the Agent-Process option—that assigns an operative systems process to each of the agent instances—or the more common Agent-Thread(s) option—that assigns one or more threads to each of the agent instances. The scheduling engine takes advantage of one of the unique features of the Common Language Infrastructure (CLI)³: Application Domains. Application Domains are lightweight processes that reside within a single operative sys-

³ The Common Language Infrastructure is the specification of the virtual execution system and of the runtime environment implemented within the .NET and Mono frameworks.

tem process and share the resource of the same CLI execution engine. They can be easily started and stopped as threads and provide customization features in terms of security and execution isolation. In order to improve the ability to tune the model and overcome possible performance bottlenecks, the scheduling engine can be configured with different types of Agent Factories:

- *LightAgentFactory*: this factory creates all the agent instances within a single Application Domain;
- *AppDomainAgentFactory*: this factory creates a different Application Domain for each software agent;
- *AppDomainPolicyAgentFactory*: this factory makes use of configurable policies to partition the number of agent instances into different Application Domains.

This solution provides the highest flexibility and can easily fit a wide range of scenarios by means of policies. Moreover, it also provides an easy way to differentiate security profiles when needed.

The Agent Scheduler is responsible for controlling the life cycle of agents and automatically configuring the required Behavior Scheduler for each of the agent instances. There are three basic behavior schedulers offered by the framework:

- *DefaultBehaviourScheduler*: maps each behavior object to a different thread. This is the most demanding scheduler in terms of system resources.
- *ThreadPoolBehaviourScheduler*: uses a configurable thread pool for scheduling behavior objects. This scheduler provides a good performance on the average case.
- *CompositeBehaviourScheduler*: used when behavior objects support interleaved execution.

The support given at run-time for behavior execution also comprises a set of APIs that are accessible to agents and behavior objects. They allow users to control behavior execution, creation, and the implementation of complex synchronization patterns such as configurable barriers.

The entire scheduling engine can be customized according to specific needs, and third parties can easily integrate their implementation into the platform without changing the system code base but by simply editing configuration files.

3.2.3 *Agent Execution*

The process of creating a new agent instance requires its definition to be present in the platform. This operation is accomplished by uploading the libraries containing the agent template definition into the platform while it is running. Users can dynamically instantiate new agents by indicating the type of template, the

name of the agent, and an optional configuration file. The System Core delegates this task to the configured agent factory that: (i) retrieves the required template definition from the storage, (ii) creates a new agent instance according to its specific implementation and policy, (iii) configures such instance with the agent template, and (iv) sets the required behavior scheduler. The instance is then returned to the System Core that initializes it, adds it to the agent scheduler, and binds it to the runtime of the platform.

During execution, the aggregate activity of behavior objects defines the status of the agent. The behavior scheduler takes care of the concurrent execution of behavior objects. Moreover, specific APIs are provided in the behavior base class to access the knowledge objects in order to avoid race conditions and spurious read and writes. These APIs ensure exclusive access to the shared resources at execution time. In addition the framework provides a tool for statically detecting the possibility of race conditions by simply examining the agent template source code.

4. Distributed Multi-agent System Support in AgentService

We can define a distributed multi-agent system as a multi-agent system composed of software agents running on different nodes of a network. Hence, the area of distributed agent computing is the area in which distributed systems enable, or facilitate, multi-agent systems, and multi-agent systems are a special kind of distributed application.

There could be different degrees of distribution in terms of communication, code mobility, and interoperation. A multi-agent system supports distributed communication when agents interact with other agents located on different platforms by exchanging messages. Support for code mobility implies the ability to move software agents from one platform to another one. Whereas the code mobility and communication are restricted to agent platforms built with the same technology, support for interoperation allows agents implemented with different technologies to interact. The interoperation is generally limited to communication and can be achieved by relying on standards such as FIPA-ACL⁴. AgentService supports the first two degrees of distribution and by implementing FIPA-ACL is able to interoperate with JADE platforms.

⁴ FIPA ACL Message Structure Specification, FIPA, [Online Document] Available at: <http://www.fipa.org/specs/fipa00061/index.html>.

4.1 Inter-platform communication

AgentService provides a communication service based on message exchange and that is transparent to the physical location of the agents. In order to send a message it is only necessary to know the agent name and the platform address where the agent resides. It is then responsibility of the messaging module to deliver the message. The framework provides two solutions for inter platform communication: the former is implemented by using Web Services, the latter uses the .NET Remoting.

The solution based on Web Service technology is tightly coupled with the Messaging Module. Agent messages travel within SOAP messages and have to be serialized in XML format. The adoption of Web Service technology ensures standards compliance for network communication; it allows AgentService agents to easily interact with external software components and vice versa.

The solution based on .NET Remoting has been recently included in the standard version of the Messaging Module. The module filters the messages exchanged by agents and dispatches those directed to federated platforms simply by checking the address of the recipient. In order to receive messages from the other platform it exposes an interface that Remoting clients can invoke to send messages. In respect to the previous solution based on Web Services, this one is more practical and compact, because it does not require the use of external services as the Web Server. On the other hand, because of the use of Remoting, it reduces the possibilities of interoperation between different technologies.

Message exchange is a fundamental operation for software agents; hence, we setup a test to evaluate the performance of the Remoting Messaging Module. The test measures the round trip time of a ping message exchanged between two agents. We first performed the test among agents within the same platform (intra platform communication), and then among agents located on different platforms on the same local area network.

Fig. 2 shows how the average round trip time increases when the number of agents exchanging the messages increases. Both of the two experiments show an exponential growth of the round trip time as the number of software agents hosted increases. Whereas in the first test the performance is acceptable even for two hundred agents, in the second test the round trip time increases from one to two orders of magnitude. Since the overhead introduced by the Remoting infrastructure is present in both of the two cases the reason of this decrease is probably due to the network. Nonetheless, the medium time round trip time in the worst

case for the inter-platform communication—which is about four seconds for two hundred of agents—is still an acceptable value.

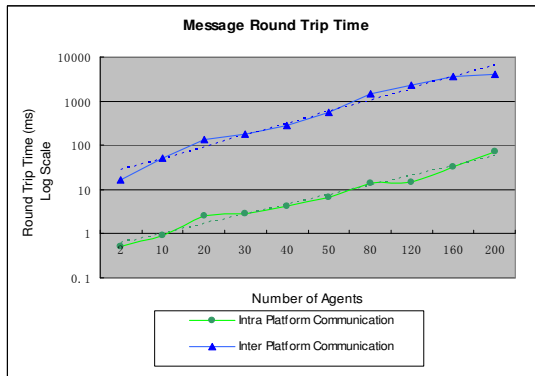


Figure 2: message round trip time.

4.2 Agent Mobility Infrastructure

AgentService supports mobility of agents by implementing a customizable weak mobility model [13]. The main idea is to exploit the adopted agent model and to move just the agent runtime state between platforms. Within the target platform the agent activities can be restarted by taking advantage of the persisted agent state. In addition, the framework provides developers with an entry point, the *Resume* method, for customizing the agent resumption process.

The migration of one agent can be initiated by the agent itself, decided by the platform administrator, or automatically activated by load balancing algorithms.

The execution of an agent in a remote runtime environment requires the information defining the agent state and the libraries containing the agent definition. Hence, moving an agent among AgentService installations requires ensuring the presence on the target platform of the template defining the agent and moving its runtime state. The agent runtime state is defined by collection of its knowledge objects and the list of the current state of its behavior objects. This information is the same that is required to restore the agent instance after a platform restart on the local node. Moreover, mobility requires an additional security infrastructure since unknown and potentially harmful code is executed on a remote host. AgentService addresses this issue by relying on the security model provided by the CLI for what concerns code verification, low level execution authorization. Moreover it also provides a set of customizable policies allowing platform

administrators to restrict carefully define the agents that are allowed to migrate and being hosted on the remote node.

We implemented the mobility infrastructure as an additional model and we integrated with the pre-existing system with minimal efforts.

4.3 Interoperability

Interoperability is obtained by supporting FIPA-ACL and developing a complete and full featured object model for software ontologies. AgentService has its own format for messages exchanged between agents: these messages can convey any kind of serializable CLI object. This solution does not allow a wide inter-operation since most of the other programming frameworks are based on the Java technology. By supporting FIPA-ACL and creating a specific message body for FIPA messages it is possible to interoperate with all the platforms able to understand FIPA ACL. Moreover, AgentService provides a specific version of the Messaging Module able to dialog with JADE platforms and exchanging messages based on FIPA-ACL with them. This feature, together with a complete implementation of ontology support, allows quickly prototyping agents that are FIPA-ACL aware and to develop real applications that spawn over different agent programming frameworks.

5. Additional Tools

AgentService support software engineers with a collection of tools that is mainly focused on leveraging the design and the implementation phases of a multi-agent system. Of a particular interest are the agent designer and the interaction designer.

The agent designer allows developers to graphically define a new agent template together with the knowledge and behavior objects. It automatically generates the code corresponding to the visual representation of the agent and keeps the two views synchronized.

The interaction designer is used to define interaction protocol between two agents. It allows developers to graphically compose the state machine defining the interaction protocol—that can be expressed by means of ontology libraries—and automatically generates the code implementing the state machine.

The two designers are part of a plug-in that can be integrated into the Visual Studio .NET development environment and allows users to use the IDE to create multi-agent systems that target the AgentService framework.

6. Discussion

In order to compare AgentService with the other works presented, we have to discuss the agent model, the runtime infrastructure supporting the multi-agent system, and the tools available in the framework.

In comparing the agent models it is interesting to evaluate the choices made to support autonomy and multi-behavioral execution. Autonomy is poorly supported in AgentFactory since other agents can obtain a reference to other agents and control their behavior. Moreover multi-tasking is completely absent. JADE exposes the same weaknesses regarding autonomy and supports the parallel execution of behaviors by exposing execution interleaving to developers. Conversely, JACK provides a good support for either the autonomous execution of agents or behavior parallelism. AgentService supports autonomy by design and ensures that agent instances can only be directly controlled by the platform infrastructure. At runtime it is possible to isolate the execution of one agent thus enforcing its autonomy. Moreover, the parallel execution of behaviors is completely transparent to developers and delegated to the runtime engine, unless complicated synchronization patterns and a finer control on behavior execution is needed. Even in this case, the framework provides some useful APIs that simplify the work of developers.

The runtime infrastructure is of great importance for multi-agent system development, since it determines the services available to agents. This is one of the strengths of AgentService. The flexibility and modular architecture of the AgentService platform is unique when compared to the solution provided by the other frameworks. In particular, the scheduling engine has been designed by considering security, flexibility, and efficiency. Moreover, the simplicity through which the run-time system can be extended, customized, and tuned is straightforward and constitutes a distinctive feature. The presence of a runtime infrastructure strongly varies in the other frameworks. JADE provides a full featured, FIPA compliant agent platform. AgentFactory includes a minimal implementation of a FIPA compliant agent platform and supports agent migration but is no more than an agent container. JACK is not compliant with the FIPA specifications for the abstract architecture. The concept of software platform is almost implicit and not well defined. These three frameworks do not provide the flexibility offered by AgentService.

The support provided by means of designers and additional tools is of a great value in a framework since it simplifies and enhances all the phases of the software life cycle. All of the frameworks discussed in this paper provide software engineers with designers, monitoring facilities, and some RAD tools. AgentService is mainly oriented to support the design and the implementation phases while the support for monitoring and maintenance is not mature yet.

Feature	AgentService	JADE	LEAP	Agent Factory	JACK
Technology	CLI (.NET)	Java	Java	Java	Java
FIPA Service Components	V	V	V	V	
FIPA Compliance	(working)	V	V	V	
Agent Model	Knowledge-Behaviour	Behaviour	Behaviour	BDI	BDI
Native BDI				V	V
Advanced Scheduling	V				
Mobility	V	V	V	V	V
Interaction Designer	V			V	V
Runtime Tools	(partial)	V	V	V	V
Ontology Support	V	V	V	V	V
Language Support	(experimental)			V	V
Methodology Support		V	V	V	V
GUI Support	V	V	V	V	V

Figure 3: framework features comparison

Fig. 3 provides a summary of features between AgentService, JACK, JADE (and LEAP), and AgentFactory. The table lists some of the most representative features ranging from architectural aspects of the frameworks, compliance with FIPA specifications, presence of CASE tools, and methodologies. As emerges from the comparison AgentService is the only solution based on the .NET technology and this makes the framework unique for what concerns its scheduling engine. Another element of distinction is its intrinsic modularity that makes it really suitable for dynamic and evolvable software environment. Nonetheless, the framework still lacks in providing an advanced support to the end user by means of designers and GUIs for monitoring multi-agent systems. This is principally due to its age, which is relatively young if compared to other frameworks.

7. Conclusions

In this paper, we have presented AgentService. The main design goal of this framework was to simplify the development process of a multi-agent system and to provide a good support for high quality software production. This is reflected in the design choices adopted in designing the core components of the system: the agent model and the runtime infrastructure supporting the multi-agent system. The agent model and the

scheduling engine are the key elements that allow the implementation of real autonomous, multi-behavioral, reactive agents. These two components are integrated into a robust, highly customizable runtime environment allowing the creation of multi-agent systems that can be easily distributed and that can evolve over time. To the best of our knowledge there is no other framework that exposes such degree of customization and flexibility. We believe that this is a really important feature for complex software systems, which normally evolve their composition and their structure over time.

As emerged by the comparison with other frameworks, AgentService does not support all the phases of the software life cycle with high level tools. This is a lack we intend to cover. In particular, it is our intention to provide a better support for the analysis phase by providing the integration with the most known analysis methodologies [14] and development processes, in particular Tropos [15].

8. References

- [1] M.J. Wooldridge, Intelligent Agents, in G. Weiss (Ed.), *Multiagent systems: a modern approach to distributed artificial intelligence* (Cambridge, MA: MIT Press, 1999), 27-71.
- [2] S. Poslad, P. Buckle, & R. Hadingham, The FIPA-OS agent platform: open source for open standards, *Proc. of PAAM2000*, Manchester, UK, 2000.
- [3] P. Braun, P. W.R. Rossak, *Mobile agents - Basic concepts, mobility models, and the Tracy Toolkit* (San Francisco, CA: Morgan Kaufmann Publishers, 2005).
- [4] A. Omicini, & F. Zambonelli, The TuCSon coordination model for mobile information agents, *Proc. 1st Workshop on Innovative Internet Information Systems*, Pisa, Italy, 1998.
- [5] F. Bellifemine, G. Caire, & D. Greenwood, *Developing Multi-agent Systems with JADE*, (Chichester, UK: Wiley, 2007).
- [6] R. Collier, G.M.P. O'Hare, T. Lowen, & C. Rooney, Beyond prototyping in the factory of agents', *Proc. 3rd Central and Eastern European Conf. on Multi-Agent Systems - CEEMAS'03*, Prague, Czech Republic, 2003.
- [7] R. Evertsz, M. Fletcher, R. Jones, J. Jarvis, J. Brusey, & S. Dance, Implementing industrial multi-agent Systems using JACK, in *Programming multi-agent systems*, (Heidelberg, Germany: Springer Berlin, 2004), 18-48.
- [8] M. Berger, S. Rusitschka, D. Toropov, M. Watzke, & M. Schlichte, Porting distributed agent-middleware to small mobile devices, *Proc. Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices*, Bologna, Italy, 2002.
- [9] A. Pokahr, L. Braubach, & W. Lamersdorf, Jadex, A BDI Reasoning Engine, in R. Bordini, M. Dastani, J. Dix and A. El Fallah Seghrouchni (Eds), *Multi-agent programming* (New York, NY: Springer Science+Business Media Inc, 2005), 149-174.
- [10] G. Caire, D. Gotta, & M. Banzi, WADE, A software platform to develop mission critical applications exploiting agents and workflows, *Proc. 7th International Conference on Autonomous Agents and Multiagents Systems AAMAS 2008*, Estoril, Portugal, 2008.
- [11] J. Odell, H.V.D. Parunak, & B. Bauer, Extending UML for agents. *Proc. Agent-Oriented Information Systems (AOIS) Workshop, 17th National Conference on Artificial Intelligence (AAAI)*, 2000.
- [12] H.S. Nwana, D.T Ndumu, L.C. Lee, & J.C. Collis, ZEUS: a toolkit and approach for building distributed multi-agent systems, *Applied Artificial Intelligence Journal*, 18, 1999, 129 - 186.
- [13] G. Cabri, L. Leonardi, & F. Zambonelli, Weak and strong mobility in mobile agent applications, *Proc. 2nd International Conference and Exhibition on The Practical Application of Java (PA JAVA 2000)*, Manchester, UK, 2000.

- [14] M.F. Wood & S.A. DeLoach, An overview of the Multiagent systems engineering methodology, *Proc. 1st International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, 2000.
- [15] A. Perini, P. Bresciani, P. Giorgini, F. Giunchiglia, & J. Mylopoulos, Tropos: an agent-oriented software development methodology, *Autonomous Agents and Multi-agent System*, 8(3) 2004, 203-236.

9. Biographies

Christian Vecchiola is a post doctoral fellow at the Computer Science and Software Engineering Department, The University of Melbourne, Australia. He currently works on distributed evolutionary algorithms design and implementation. He received his Ph.D. in Computer Science from the University of Genova in 2007 with a dissertation on applying the agent oriented paradigm to the design and the implementation of evolvable software systems. Dr. Vecchiola, together with Dr. Grosso, is one of the chief designers of the AgentService programming framework. His research interests include agent-oriented software engineering, programming language design and compiler implementation, and distributed evolutionary algorithms.

Alberto Grosso received his Ph.D. in Computer Science from the University of Genova in 2006. He currently works at DIST and his primary research interests are in the field of multi-agent systems, multi-robot systems, and agent oriented software engineering.

Andrea Passadore took a Master Degree in 2006 in Computer Science. At now, he is a PhD Student at the University of Genova. His main interests are in the field of multi-agent systems, semantic web, and SOA. He is involved in the AgentService development team and coordinates different projects and applications, with the main goal to improve the platform usability and performances.

Antonio Boccalatte graduated in Electronic Engineering at the University of Genova (1976). In 1987 won a chair at the University of Genova, and actually is Professor of «Data Base Management Systems» at the Faculty of Engineering. He is author of more than 70 scientific papers presented at international congresses or published on international journals. He is involved in research activity on System Architecture and Artificial Intelligence.