Software Engineering for Service-Oriented MAS

Emilia Garcia, Adriana Giret, and Vicente Botti

Department of Information Systems and Computation, Technical University of Valencia, Camino de Vera, Valencia, Spain
{mgarcia,agiret,vbotti}@dsic.upv.es

Abstract. Nowadays, service-oriented architectures (SOA) and multiagent systems (MAS) are two increasingly important technologies. Despite the differences in technology, SOA and MAS have some similar objectives and their integration produces systems with more flexibility, functionality and interoperability. Their integration creates new requirements and special methods and tools are necessary to develop systems that integrate both technologies. This paper analyzes the most important issues for developing Service-oriented MAS. Furthermore, some methods and tools to develop this kind of systems are analyzed to show how current approaches solve the problem of the integration between agents and services.

Keywords: Multiagent systems, service-oriented architectures, software engineering, development tools.

1 Introduction

Nowadays, SOA and MAS are two increasingly important technologies. The objectives of both architectures share some similarities, i.e., both of them try to create distributed and flexible systems that are composed of loosely-coupled entities which interact with each other.

Despite these similarities, there are major differences in their technology. Services have interface standards and exchange protocols that are completely different from agent communication languages and protocols. This is why they cannot interact with each other directly.

This is a problem which needs to be solved, but some studies [15] have investigated this issue and show that the integration of agents and services produces attractive benefits. Services have a well-defined infrastructure and interoperability whereas agent technology aims to provide intelligent and social capabilities (trust, reputation, engagement, etc) for applications. Therefore, the integration of agents and services improves the flexibility, interoperability and functionality of the system.

Nevertheless, most agent software engineering techniques do not consider integration with services, nor do service software engineering techniques consider integration with agents.

There are some works that address the integration between agents and services [8]. They define frameworks and provide tools for developing systems in

M. Klusch, M. Pechoucek, and A. Polleres (Eds.): CIA 2008, LNAI 5180, pp. 86–100, 2008.

which agents and services are integrated. Each of them has its own integration mechanism, communication language, and even its own service and agent concepts.

In order to define the software engineering issues for developing Service-oriented MAS, a detailed study of the state of the art of software engineering for agents, services and service-oriented agent systems is made and briefly summarized in Section 2. Section 3 defines a list of the most important software engineering requirements for developing Service-oriented MAS. Furthermore, some software engineering tools for developing systems that integrate agents and services have been analyzed to define how current approaches and tools solve the problem of the integration between agents and services. A description and a brief analysis of some selected tools and frameworks is presented in Section 4. Finally, Section 5 presents some conclusions and future work.

2 Background

This section is divided in three parts. Firstly, the state of the art of software engineering for MAS is briefly described. Secondly, the state of the art of software engineering for SOA is summarized. Finally, the state of the art of software engineering for service-oriented agent systems is analyzed.

2.1 Agent-Oriented Software Engineering

MAS are complex systems with a distributed nature, where each element is autonomous, reactive, proactive and social. This complexity makes the use of techniques and tools to support the development process necessary. Agent-oriented software engineering is based on traditional software engineering, but it takes into account the specific features of the agent technology. On the market today there are a great number of methodologies, development environments, modeling languages, debugging tools and platforms that deal with the development process of a multiagent system [16].

Some well-known tools and methodologies include AUML [21], Jade [1] , JACK [9], Gaia [23], Tropos [7].

The two main drawbacks to software engineering tools of this kind are the gap between modeling and platforms, and the lack of automatic and complete translation between the models and the executable code [18].

Agents usually use specific technology to represent ontologies, protocols and content languages. This makes the interaction with other types of systems difficult and sometimes necessitates the use of intermediary elements.

2.2 Service-Oriented System Engineering

It is based on traditional system engineering, but it must take into account specific characteristics of SOA systems. System engineering needs to be collaborative because SOA applications are often collaborative. Service consumers,

service brokers and service providers collaborate to invoke, search, register and provide services. Systems may be composed at runtime using existing services so many SOA engineering tasks need to be done on the fly at runtime.

Services are oriented to being reused, so it is very interesting for them to have platform-independent modeling techniques.

Recently some initial results have been proposed [20,12]. These works have mainly concentrated on developing a methodology for service-oriented engineering and design-time models. In the first line of research, works have concentrated on how to provide sufficient principles and guidelines to specify, construct, refine, and customize highly volatile business processes choreographed from a set of internal and external Web services [22,13]. In the second line of research, some works have concentrated on developing design-time models using goal-oriented requirement analysis techniques [17,14].

2.3 Service-Oriented MAS Engineering

Agents and services have clear differences, such as the different representational encodings and technologies, for example. Despite these differences though, they have similar goals. Both technologies build flexible and distributed systems. Some researchers have studied the benefits of mixing these technologies [15]. They state that using agents and services can provide systems with more flexibility, functionality and interoperability. Many works on this subject are focused on the interaction mechanism between agents and services [11], but there is no work that provides a complete description of which are the software engineering requirements for developing this kind of systems. A detailed list of the software engineering features to take into account in the Service-oriented MAS development is presented in Section 3. Furthermore, there are few development environments and tools that actually offer facilities for developing agents that interact with services. Some of these tools are described and analyzed in Section 4.

3 Software Engineering Requirements

The aim of this section is to make an overall analysis of the needs that arise when developing Service-oriented MAS, relying more on the concepts than in a specific terminology. In the specialized literature, there are a great number of different perspectives about the integration between agents and services; there are even different conceptions of what an agent and a service are. Because of this, the identification of a set of independent, orthogonal features which completely characterize the Service-oriented MAS development process seems unfeasible.

The selection of these issues is based on the background briefly described in Section 2 and the study of current frameworks and techniques for Service-oriented MAS engineering that is summarized in Section 4.

These issues are classified into four categories: (1) Integration between agents and services; (2) Development issues; (3) Multiagent systems; (4) Service-oriented architectures.

3.1 Integration between Agents and Services

Firstly, it is important to analyze the way in which the relationship between agents and services is considered. Drawing from the published literature and, as described in [5], there are three different ways: (1) Some approaches hold that there is no conceptual distinction between agents and services. For them, both are active building blocks in a loosely-coupled architecture, and there is only an engineering problem of creating overall systems behaviors from active components [24]. (2) Other approaches hold that agents and services can communicate in a bidirectional way. They have to provide a mapping between the language protocol used in the multiagent system and the service language protocol and vice versa [2,25]. (3) Finally, other approaches hold that the communication is only useful in one direction, i.e., agents invoke services but not vice versa. In this view, agents are responsible for the application, and they use services or composite services as resources to achieve their objectives [4].

Furthermore, agents and services have different **communication standards**. Agents usually use FIPA ACL messages. Services are described with WSDL descriptions and use SOAP as the communication mechanism. However, not all the approaches follow the standards. The use of different technologies in services and agents and the **mechanism to match** them should be evaluated. Another point to consider is if the approach offers the possibility of dynamically **publishing and discovering services**. Some approaches even discuss the possibility of interacting with **external services** that may or may not be registered in the system.

3.2 Development Issues

This section analyzes the general features that a method and a development environment for developing Service-oriented MAS should have. These features are extracted from traditional software engineering and are grouped into three categories:

3.2.1 Software Engineering Support

As described in Section 2, the development of a system that integrates agents and services is a complex task which greatly benefits from the adoption of software engineering techniques. Firstly, the **application domain** should be considered. Some approaches are supposed to address systems from any domain and other are oriented to specific domains. The **model-central element** is another key feature; it defines the initial point and the perspective of the modeling process. Offering a **methodology** that involves agents and services and that also take into account their interaction greatly helps the developer to go from the initial information to the final implementation. There are lots of methodology features that can be analyzed, but the most important are which parts of the development process are covered and whether development guidelines are provided. Furthermore, a methodology component that can process a user behavioral description of desired functionality and recommends that the behaviour should be implemented via a service or an agent, is a very useful feature for developing

this kind of systems. The **modeling language** specifies the type of notation used for modeling. It can be formal or informal and may or may not use graphical elements. The notation should be precise, complete and clear. The systems are usually very complex, so it is very useful if the language modeling allows to model at different abstraction level. A very interesting feature is to offer mechanisms to specify semantics of model element extensions using formal methods, such as OCL.

3.2.2 Technical Issues

There are many approaches that only analyze the development process in a theoretical way. They define methods, but they do not offer **tools** or development environments to model, design and implement systems. Thus, one evaluation criterion is whether the approach offers tools and which parts of the **development process** are covered by them. These tools can be evaluated based on many criteria, but the most important are the requirements to set up and run it, the functionality offered, the ease of use and their scalability.

3.2.3 Implementation Issues

The most advanced development environments integrate the modeling, design and implementation processes in the same tool. They even offer tools that **automatically generate parts of the code** from the models. These characteristics are very desirable because they reduce the implementation time and the number of implementation errors. In the development process of systems that integrate agents and services, the **translation** from one descriptive language to another is very useful. This requires that the development environment has to offer a mechanism to automatically translate from one standard to another.

3.3 Multiagent Systems

There are many works that analyze the features that should provide the methodologies and the development environments for multiagent systems [6,18]. They describe a great number of criteria, but, in this paper, only the most important criteria for the specific case where agents interact with services are considered.

The multiagent system approach should have an **agent architecture** that carry out the fundamental **properties** of agents (autonomy, reactivity, sociality and proactiveness). The **content language** used for the communication mechanism and whether or not the architecture is FIPA compliant are very important features because they are strongly related with the necessary mechanism to interact with and to integrate services and agents.

3.4 Service-Oriented Architectures

As explained in Section 2.2, there are some features that should be taken into account for developing systems that use service-oriented technology.

The service architecture should be modeled **independently of a specific platform** to obtain more flexibility and reusability. The **platform-dependent**

characteristic also need to be specified to implement the system. Service-oriented systems are usually composed of many **standards** (See 2.2). Therefore, the development environment should provide facilities to implement and check the correctness of these standards. Another feature to take into account is the way **service descriptions** are implemented. Service specification provides a means for defining complete service specifications that include behavioral rules in addition to static interfaces, operations, preconditions, post conditions, and constraints. Service specifications are not architecturally neutral. Services can be **composed** to achieve more complex functionality. The mechanism to specify how services are composed from other services in the models and how the composition is translated to an executable code is another important feature.

4 Frameworks and Techniques

A brief description and analysis of some frameworks and techniques for modeling, designing and implementing systems of this kind is given below. This selection was made due to the relevance of the researchers and companies responsible for these tools and the fact that they cover a range of pertinent issues and supporting technologies that are primarily focused on the integration of agents and services.

- **The Nuin agent platform.** Nuin [4][5] is an open-source Java implementation which is a combination of a belief-desire-intention (BDI) agent platform and semantic web techniques. It provides an abstract service boundary to add custom behaviours to the agent. This abstract service boundary also provides a natural basis for extending the internal agent services in order to include external web services.
- **JASE.** It [3] is a Java-based Agent-oriented and Service-oriented Environment for deploying dynamic distributed systems. It defines a service-agent programming model, which is a combination of two concepts in the field of distributed computing: the concept of services and the concept of mobile agents. In JASE, mobile agents are used to support applications, and service interface agents are used to wrap services.
- **The Agent Modeling Language (AML).** AML [19] is a semi-formal visual modeling language for specifying, modeling and documenting systems that incorporate features drawn from multiagent systems theory. AML also supports the modeling of services and their interaction with agents.
- **The framework for Rapid Prototyping of SOA.** This framework is proposed by Zinnikus in [25]. It is built around a Model-Driven Development methodology that is used for transforming high-level specifications of SOA into executable artefacts, both for Web Services (WSDL files) and for BDI agents. It follows the OMG Model-Driven Architecture (MDA) approach and defines a Platform-Independent Model (PIM) for SOA (PIM4SOA) and Platform-Specific Models (PSMs) for describing Web services (XSD and WSDL), JACK BDI agents and BPEL processes. This framework is composed of three parts: a modeling part, a service part and an autonomous

agent part. The modeling part is concerned with applying Model-Driven Development (MDD) techniques and tools to the design of SOAs. It defines models and transformations that are specific to the concepts used for SOAs, such as Web Service descriptions and plans for autonomous agents. The service part provides a highly flexible communication platform for Web services. The autonomous agent part deals with designing and enacting service compositions as well as performing mediation, negotiation and brokering in both SOAs.

- **Jade web services integration gateway (WSIG).** WSIG [2] is a Jade add-on that provides support for bidirectional invocation of Web services from Jade agents, and Jade agent services from Web services clients.

4.1 Tools Analysis

In this section, some of the features of these frameworks are related to show how current approaches develop this kind of systems, but the goal of this section is not to make an extensive and complete analysis of these approaches. Figures 1, 2 and 3 summarize this analysis highlighting the parts that are not covered by the analyzed tool.

4.2 Integration between Agents and Services

The Nuin approach considers that interaction between agents and services is only useful in one direction. Agents primarily are responsible for mediating between user goals and the available strategies and plans. Agents invoke atomic or composite external web services as necessary. An initial approach at integration of semantic web capabilities is to include the use of RDF/OWL as a knowledge representation, and the ability to use RDQL to query RDF-based knowledge stores. A future goal for the Nuin approach is to add support for the direct use of semantic web service descriptions in OWL-S. Nuin assumes that an appropriate binding to the abstract service is defined, but the interaction between Nuin agents and web services has not yet been implemented (see Figure 1).

In JASE, the general idea of service is that the application is separate from the resources needed to fulfill a task; these resources are modeled by services, which are independent of the application. JASE models services as agents. A service interface agent encapsulates a local resource. Each service interface agent consists of two parts: a service agent and a service interface. A service interface acts as a front-end interface for the other agents in the system to communicate with the service agent it represents. A service agent is a specialized service, which can be realized in the form of software or hardware. JASE uses XML to describe both service descriptions and agent queries, so no gateway is necessary.

In AML, services are encapsulated blocks of functionality that the entities can offer to perform upon request. AML is only a modeling language, so it represents a bidirectional interaction between agents and services. However it does not consider the technological differences between agents and services. The services publication and their discovery are not considered, either.

	Integration between agents and services					
	Integration type	Standards		Interaction mechanism	Publish services	External Services
		Agents	Services			
Nuin	Agents invoke services	Nuinscript , OWL, RDF	WSDL	not implemented yet	use external UDDI registers	can use them
Jase	Model services as agents	XML	XML	not need (use the same standards and protocols)	provide Service Server	can not use
AML	not covered	not covered	not covered	not covered	not covered	only internal services
WSIG	Bidirectional	ACL	WSU stack	covered completely	not covered	not covered
Zinnikus	Bidirectional	not specified	WSDL	covered completely	yes	can use them

Fig. 1. Integration issues

WSIG is a gateway that offers automatic, bidirectional operation allowing both FIPA compliant agent services and Web services to be registered with it. Agent services and web services can thereby publish their service descriptions to consumers outside their normal operational domain. The gateway can then intercept calls to these registered services allowing agents to invoke Web services and vice versa by transforming message encodings and creating service access endpoints. All invocation-related interactions between the gateway and agents use ACL encoded FIPA-Request and FIPAInform performatives. All Web services use the standard WSU stack (WSDL, SOAP and UDDI).

The Zinnikus approach extends the JACK agent framework for Web Services in order to provide a goal-oriented service composition and execution module within a SOA. Following the MDA approach, at design time a modeller specifies a set of plans (PSM level) that constitute the workflow library of the agents. Web service calls are integrated as steps into plans. Service providers are mapped to JACK agents/teams. The parts of the PIM that define the processes involved are mapped to agent/team plans and correlated events, whereas the parts that define the interfaces are mapped to the modules that provide the client- and server-side code for the JACK agent platform. Johnson and Lyndon are tools of the service part of the framework and allow the communication between external and internal web services and agents. The Johnson tool is responsible for invoking web services and receiving calls issued by Web service clients. The Lyndon tool takes WSDL files as input and configures Johnson tool to play either the role of service provider, service consumer or service proxy for the service described by the WSDL file analyzed.

4.3 Development Issues

4.3.1 Software Engineering Support

In all the studied tools, the application domain is general as shown in Figure 2. Even though JASE models can describe any kinds of open and global dynamic distributed systems, it is specialized in mobile agents.

The central element of the model in all cases is the agent, except for the Zinnikus approach. This approach defines first a platform-independent model for services (PIM4SOA) and later platform-specific models that have agents as central elements.

Nuin and JASE do not provide or use any methodology, nor any modeling language.

AML is a modeling language that is specified as an extension to UML 2.0 in accordance with major OMG modeling frameworks (MDA, MOF, UML, and OCL). It proposes 11 diagrams that extend UML 2.0, to model all the MAS features and interactions with services graphically. The notation is clear, complete, precise and understandable. AML offers the possibility to model at different abstraction level. ADEM is the agent methodology proposed for AML researchers but there is no public available detailed documentation of ADEM.

WSIG is a transparent gateway to translate communication standards, so it cannot be analyzed with these features.

As explained in Section 4, the Zinnikus approach is built around a Model-Driven Development methodology that transforms high-level specifications of a SOA into executable artefacts, both for web services (WSDL files) and for BDI agents. The modeling part of the framework and more specifically, the MDD framework defines the metamodels used to specify SOAs. It also provides modeling guidelines, model transformation and generation support for execution artefacts such as WSDL files and BDI plans. It also supports importing existing WSDL files into the SOA models. All these models are represented graphically.

4.3.2 Technical Issues

Nuin and JASE do not provide any development environment to implement applications. Nuin models core BDI agent architectures on AgentSpeak(L) and PRS, allowing agent designers to specify and implement agents using programming abstractions that correspond closely with the terms commonly used in intelligent agent theories. It is written in Java, and requires JDK 1.4 or later. Jade and Jena libraries are also necessary for a full functionality.

JASE is also implemented in Java. Similar to Nuin, JASE provides programming abstraction libraries.

AML is supported by tree case tools: Rational Rose 2003, Enterprise Architect 4.0 and StarUML. The AML implementation consists of UML profile support for AML, a set of modeling utilities (specialized element specification dialogs, model consistency checker, etc.), and forward-engineering tools for TAPI, the commercial-agent platform of Whitestein Technologies AG.

The WSIG requires JADE v3.3 platform to run, and the following third party technologies are available on the system: Jakarta Tomcat, Apache jUDDI, mySQL, MySQL Conector/J.

The Zinnikus approach provides tool support for the MDD framework. It has been developed as a set of plugins for Rational Software Modeller (RSM) (IBM Rational Software). RSM is a UML 2.0 compliant modeling tool from IBM based on the Eclipse modeling environment. All models and metamodels were implemented using the EMF Core (Ecore) metamodel. Model transformations have been implemented using the model transformation capabilities of the RSM/Eclipse platform. Also [25] provides Johnson tool, Lyndon tool, WSDL Analyzer (a tool for detecting similarities at a structural level between WSDL

	Development issues								
	Software Engineering Support				Technical issues		Implementation issues		
	Application domain	Model-central element	Methodology	Modeling language			Automatic generation code for:		
					Offer tools	Development process covered	Agents	Services	Standards translation
Nuin	General	Agents	not covered	not covered	Programming abstraction libraries	Implementation (only programming help)	api	not covered	not implemented yet
Jase	General	Mobile agents	not covered	not covered	Programming abstraction libraries	Implementation (only programming help)	api	api	no needs
AML	General	Agents	ADEM no public available	Yes, Informal, Graphic	yes	model, design, implementation (not tested)	Jade generation (not tested)	no	no
WSIG	not covered	not covered	not covered	not covered	not covered	not covered	not covered	not covered	complete and transparent
Zinnikus	General	Services /Agents	Services /Agents	Yes, Informal, Graphic	yes	complete but poor at implementation	no	no	no

Fig. 2. Analysis of the development issues

descriptions of Web services and generating the corresponding mappings) and RDF store (which stores both design-time information and runtime information as RDF files for the purpose of monitoring).

4.3.3 Implementation Issues

As explained in the above sections, JASE and Nuin do not offer any modeling mechanism, so the translation between models and code is not considered.

The CASE tools provided by AML are supposed to be able to translate agent models into Jade code [10], but these plugins are not publicly available.

The WSIG v0.4 supports the standard WSU stack and FIPA acl/sl0 communication. The functionality of WSIG is translated between these two standards. WSIG automatically registers UDDI web service registers in the Jade platform DF and viceversa, i.e., WSIG registers all agent services of the DF in the WSIG UDDI repository. WSIG is transparent, so the programmer does not need to add any code in agents or services. An agent calls a web service as to another agent and viceversa.

The Zinnikus approach provides model-to-model transformation services that allows the transformation of PIM4SOA models into underlying PSMs such as XSD, WSDL, JACK BDI agents or BPEL. However, it does not specify if JACK agents code is generated automatically from these models.

4.4 Multiagent Systems

Nuin agents are BDI agents that are reactive, autonomous, proactive and social.

JASE is not a BDI architecture. It provides a mechanism for developing agents with all the basic properties described in Section 3 as well as with the ability to migrate.

AML allows agents to be modeled with all the basic properties.

WSIG is a transparent gateway to translate communication standards, so it cannot be analyzed with these features (see Figure 3).

	Multiagent systems			Service-oriented architectures			
	Agent Architecture	FIPA compliant	Basic properties	Platform independent	Platform dependent	Service specification	Service composition
Nuin	BDI	yes	yes	not covered	not covered	not covered	not covered
Jase	no BDI	no	yes	not covered	not covered	yes	no
AML	not covered	not covered	yes	yes	no	yes	yes (model)
WSIG	not covered	not covered	not covered	not covered	not covered	not covered	not covered
Zinnikus	BDI	no	yes	yes	yes	yes	yes

Fig. 3. Agents and services issues

The Zinnikus approach extends the JACK agent framework for Web Services (JACK4WS) following the BDI model. The agents have all the basic properties. JACK is not FIPA compliant. JACK agents are not bound to any specific agent communications language. Nothing prevents the adoption of high-level symbolic protocols such as KQML or FIPA Agent Communication Language (ACL).

4.5 Service-Oriented Architectures

The objective of Nuin is not to implement services but to implement agents that can invoke services. Therefore, it cannot be analyzed with these features (see Figure 3).

JASE uses XML to describe both service descriptions and the mobile agent's queries. A service in JASE is a mechanism to encapsulate a local resource. JASE does not consider service composition.

AML does not cover most operational semantics, which is often dependent on a specific execution model given by an applied theory or deployment; it offers platform-independent models. The AML support for modeling services comprises (1) the means for the specification of the functionality of a service and the way a service can be accessed (service specification and service protocol), (2) the means for the specification of what entities provide/use services (service provision, service usage, and serviced property), and (if applicable) by what means (serviced port). They are modeled in AML in terms of service specifications, service provisionings and service usages. AML supports OCL.

WSIG is a transparent gateway to translate communication standards, so it cannot be analyzed with these features.

In the Zinnikus approach, service providers are mapped to JACK agents/ teams; the processes involved are mapped to agent/team plans, and interfaces are mapped to the modules that provide the client- and server-side code for the JACK agent platform. Thus service specification and interaction points are well-defined. The service composition is analyzed and implemented at the agent level, i.e., a composition is a collaboration between agents that are service providers.

4.6 Discussion

Some authors [5] say that if services are able to invoke agents, this would violate the autonomy of the invoked agent, thereby turning the agent into just

another service. From our perspective, it is important to differentiate between agents and services. We agree that it is useful for agents to be able to invoke services. However, we disagree with the idea that if a service can invoke an agent, the agent must expose pre-determined or deterministic behaviours. Nowadays, service technology offers the possibility to register and deregister services dynamically and there is no reason why an agent cannot change the behavior of its published services depending on its own goals and situation. For this reason, we think that bidirectional integration is more complete and useful.

Organizations like FIPA or OMG are working to establish standards, and most approaches follow them. It is very important to obtain open environments where services and agents implemented by different companies with different technologies can interact.

As discussed in Section 2, developing systems that integrate agents and services is a complex task and the use of methodologies is useful. Nonetheless, there are few methodologies that take into account both technologies. The Zinnikus approach presents a methodology in [25] which states that the methodology is complete and that guidelines are offered. However there is no more documentation and they do not offer public downloads.

Figure 2 shows that these approaches do not offer tools that cover the entire development process. Also, there are few techniques for automatic code generation and they are not sufficient. These are very important research lines that are still open.

The possibility of offering platform-independent model services is very useful for develop service-oriented multiagent systems. It allows high-level modeling where the technology used is not specified. Complete approaches for developing systems that integrate services and agents should offer this possibility as well as automatic transformations to platform-dependent models.

From the list of features presented in Section 3 and the analysis of the results of Section 4.1 we can summarize the most important requirements for developing Service-oriented MAS in the following list: (1) a methodology that involves agents and services, and takes into account their integration; (2) a modeling language that allows the definition of the specific characteristics of agents, and services as well as the specification of their integration; (3) a tool that covers the entire development process, i.e., one that supports the methodology and the modeling language used and offers implementation facilities such as automatic code generation; (4) a gateway that allows the interaction between agents and services despite the differences in their technology and standards, which should provide mechanisms for publishing and invoking services; (5) an agent platform that integrate both technologies transparently.

After this analysis it can be observed that there is currently no complete tool or framework that covers the entire development process of service-oriented multiagent systems. In order to develop systems of this kind, a software engineer has to merge a set of different methods, modeling language, tools, etc.

5 Conclusions and Future Work

Multiagent systems and service-oriented architectures are two approaches with similar goals but major differences in their technology. Both are hot research and industrial topics, and integration between agents and services is very beneficial as it generates more complete, flexible and interoperable systems with greater functionality.

The use of software engineering principles, methods and techniques in the entire development cycle of multiagent systems and service-oriented architectures is both interesting and necessary in many cases. In the same way, the development of systems that integrate the two technologies requires methodologies and development environments that take into account the specific characteristics of agents, services and their integration.

In this paper we have put forward a comprehensive list of the most important software engineering issues in the development of service-oriented MAS systems. These issues were defined, based on a detailed study of state of the art development methods and taking into account the new characteristics which arise when agents and services are integrated.

These issues are used to analyze several approaches. From this study we can conclude that in order to develop systems of this kind, it is currently necessary to merge a set of different methods and tools. There is no software engineering tool that covers the entire development process of these systems, and so this is an open line of research.

The highlighting of the fundamental development issues put forward in this study is an attempt to identify the new requirements imposed to the development process of Service-oriented MAS. Moreover, this requirement list can help to improve current state of the art methods, tools and platforms in order to develop these kinds of systems correctly.

At the same time, the ideas presented in this study can be used as a starting point from which to develop a complete framework for service-oriented MAS, in which agents and services are smoothly integrated, drawing on the advantages of both approaches.

We plan to continue along this line of work in the future, completing the list of requirement, evaluating state of the art methods, tools and platforms in order to define a ranking list of development tools for these kinds of systems.

Acknowledgements

This work is partially supported by the TIN2006-14630-C03-01, PAID-06-07/3191 projects and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

1. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). John Wiley & Sons, Chichester (2007)
2. Board, J.: Jade web services integration gateway (wsig) guide (2005)

3. Chunlin, L., Layuan, L.: An agent-oriented and service-oriented environment for deploying dynamic distributed systems. Computer Standards and Interfaces 24, 323–336 (2002)
4. Dickinson, I.: Nuin: the jena agent framework (2004), http://www.nuin.org
5. Dickinson, I., Wooldridge, M.: Agents are not (just) web services: investigating bdi agents and web services. In: Proc. SOCABE 2005 (2005)
6. Eiter, T., Mascardi, V.: Comparing environments for developing software agents. AI Commun. 15(4), 169–197 (2002)
7. Giorgini, P., Mylopoulos, J., Perini, A., Susi, A.: The tropos metamodel and its use. Informatical journal (2005)
8. Greenwood, D., Lyell, M., Mallya, A., Suguri, H.: The ieee fipa approach to integrating software agents and web services. In: Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Industrial Track (2007)
9. Jack agent platform (2008), http://www.agent-software.com/shared/products/index.html
10. Kostic, M.: Code generation from AML Implementation into CASE tools and support for existing agent platforms. PhD thesis (2006)
11. Marco Mari, M.T., Poggi, A., Turci, P.: Enhancing multi-agent systems with peer-to-peer and service-oriented technologies. In: Sixth International Workshop From Agent Theory to Agent Implementation (AT2AI-6) (2008)
12. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: 05462 service-oriented computing: A research roadmap. In: Service Oriented Computing (SOC) (2006)
13. Papazoglou, M.P., van den Heuvel, W.: Business process development lifecycle methodology. Communications of ACM (to appear, 2006)
14. Penserini, L., Perini, A., Susi, A., Mylopoulos, J.: From stakeholder needs to service requirements specifications. Technical report, itc-irst, Automated Reasoning Systems (2006)
15. Singh, M.P., Huhns, M.N.: Service-Oriented Computing Semantics, Processes, Agents. John Wisley and Sons Ltd. (2005)
16. Rafael, M.D., Bordini, H., Winikoff, M.: Current issues in multi-agent systems development (invited paper). In: Post-proceedings of the Seventh Annual International Workshop on Engineering Societies in the Agents World, pp. 38–61 (2007)
17. Rolland, C., Souveyet, C., Kraeim, N.: An intentional view of service-oriented computing. Revue Ingnierie des Systmes dÍnformation (ISI),RSTI (Revue des Sciences et Technologies de lÍnformation)- ISI 13(1), 107–137 (2008)
18. Sudeikat, J., Braubach, L., Pokahr, A., Lamersdorf, W.: Evaluation of agent-oriented software methodologies examination of the gap between modeling and platform (revised selected papers). AOSE-2004 at AAMAS 2004 (2005)
19. Trencansky, I., Cervenka, R.: Agent modelling language (AML): A comprehensive approach to modelling mas. Informatica 29(4), 391–400 (2005)
20. Tsai, W.-T., Wei, X., Paul, R., Chung, J.-Y., Huang, Q., Chen, Y.: Service-oriented system engineering (SOSE) and its applications to embedded system development. In: AOSE 2002 (2007); Revised Papers and Invited Contributions
21. A. UML. Agent uml (2008), http://www.auml.org
22. Witwicki, S.J., Durfee, E.H.: Commitment-based service coordination. In: Kowalczyk, R., Huhns, M., Klusch, M., Maamar, Z., Vo, Q.B. (eds.) Service-Oriented Computing: Agents, Semantics, and Engineering. LNCS, vol. 5006. Springer, Heidelberg (2008)

23. Zambonelli, F., Jennings, N.R., Wooldridge, M.: Developing multiagent systems: The gaia methodology. ACM Trans. Softw. Eng. Methodol. 12(3), 317–370 (2003)
24. Zhu, H., Shan, L.: Agent-oriented modelling and specification of web services. In: Proceedings of the 10th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, vol. 00, pp. 152–159 (2005) (ISBN-ISSN:1530-1443 , 0-7695-2347-1)
25. Zinnikus, I., Benguria, G., Elvester, B., Fischer, K., Vayssire, J.: A model driven approach to agent-based service-oriented architectures. In: Fischer, K., Timm, I.J., André, E., Zhong, N. (eds.) MATES 2005. LNCS (LNAI), vol. 4196, pp. 110–122. Springer, Heidelberg (2006)