Exposing agents as web services: a case study using JADE and SPADE

Henrique Donâncio[1], **Arthur Casals**[2], **Anarosa A. F. Brandão**[2]

[1]Instituto de Matemática e Estatística - Universidade de São Paulo (IME-USP)
R. do Matão, 1010 - 05508-090 - São Paulo - SP - Brazil

[2]Escola Politécnica - Universidade de São Paulo (EPUSP)
Av. Prof. Luciano Gualberto, 158 - trav. 3 - 05508-900 - São Paulo - SP - Brazil

donancio@ime.usp.br, {arthur.casals,anarosa.brandao}@usp.br

Abstract. *Agents are autonomous software components that can be combined into multiagent systems (MAS). They possess capabilities related to distributed systems. For this reason, agents have been studied along with web services and Web technologies in Service-Oriented Architectures (SOA). With the advent of new Web-related paradigms, the need of reviewing existing work in this area arises. In this paper we briefly review some of the existing work relating agents and web services. For this purpose, we perform a comparison between two existing multiagent platforms (JADE and SPADE) considering their ability to support agents as web services. This comparison is done using an existing implementation of a simple MAS in JADE and its re-implementation in SPADE.*

1. Introduction

Agents are autonomous entities that can be organized in communities and work together to solve problems of different complexity degrees [Ferber 1999]. As such, multiagent system (MAS) are systems composed of multiple agents that interact among themselves in a single environment [Russell and Norvig 2003]. Also, due to their inherent interactive capabilities, agents can be used as a paradigm when designing complex distributed systems [Wooldridge 2009]. Due to their inherent interoperability, each agent can act to maximize the expected output of the system and to adapt to unexpected contingencies [Jennings 2000].

Part of the research related to MAS and complex systems involves integrating agents and web services [Jennings et al. 1998, Lieberman et al. 1995, Etzioni 1996, Ardissono et al. 1999, Greenwood and Calisti 2004]. Designing inter-operable complex systems involves not only creating systems with distributed capabilities, but also systems capable of using existing communication protocols and mechanisms in order to share and reuse knowledge.

In particular, the idea of using agents *as* web services appeared in the early 2000s [Hendler 2001, Huhns 2002]. Subsequent work on agents exposed as web services was also related to web-based MAS [Muldoon 2007, Thiele et al. 2009, Tapia et al. 2009]. However, while SOAP web services are still largely used to implement SOA systems [Keen et al. 2004], new web paradigms use different technologies, from hypermedia-controlled RESTful web services (HATEOAS [Alarcon et al. 2010]) to real-time protocols (such as WebSockets [1]) and event-driven architecture [Michelson 2006]

[1]https://tools.ietf.org/html/rfc6455

elements.

Our research is focused on inter-operable agents capable of using Internet communication protocols. For this reason, we need to explore how existing MAS platforms can be used in conjunction with technologies related to the existing web paradigms. Different MAS platforms were developed in the last years [Kravari and Bassiliades 2015], with different levels of compliance with inter-operable systems development standards. Since the BDI agent architecture (explained below) is of particular interest for our research, we focused our comparison in MAS platforms that support this architecture. Also, due to the fact that we are interested in inter-operable agents, we explicitly analyze the communication language used by agents in the compared platforms.

This paper is organized as follows: in Section 2, we provide some background on web services, intelligent agents, and agent communication languages. Section 3 contains some of the existing work related to using agents in conjunction with Web-related technologies. In Section 4, we briefly present some multiagent platforms as well as the criteria used to choose them. These platforms are compare in Section 5. In addition, Section 6 is dedicated to illustrate how an agent can be deployed as a web service using two of the evaluated platforms. Finally, Section 7 discusses and concludes this paper with some perspectives for future work.

2. Background

In this section we present some concepts that will be used along this work. First we provide a brief description of web services, followed by an overview on intelligent agents (with emphasis on one particular architecture). After that we provide an overview on languages used by the agents to interact between themselves, referred to as *agent communication languages*.

2.1. Web services and SOA

A web service can be described as a software designed to interact with existing applications, enabling different applications to interact between themselves using Web technologies and protocols [Alonso et al. 2004]. It can be *discoverable* through the use of a directory service, and it can be *described* in terms of message formats, communication protocols, and data types it uses. Web services embody the concepts of a service-oriented architecture (SOA)[1] [Erl 2005], so they can be used in conjunction to obtain the functionalities of an equivalent larger system.

2.2. Intelligent agents

As mentioned before, agents are autonomous entities able to work together to solve determined problems. Agents can exist in both physical and virtual environments, and they are capable of proactively interacting with other agents. Such interactions can involve cooperation, negotiation, or coordination, and each agent is capable of creating their own goals and taking individual actions in order to satisfy them [Wooldridge 2009]. As a consequence, a MAS composed of goal-oriented agents is intrinsically adaptable, and can be used to solve problems in many different domains [Pěchouček and Mařík 2008].

[1]http://www.opengroup.org/subjectareas/soa

Intelligent agents are capable of reasoning, deciding which action to perform according (i) to the available information and (ii) to their consequences in the environment [Wooldridge 2009]. The belief-desire-intention (BDI) architecture [Rao and Georgeff 1991] is one of the software architectures used to model and implement intelligent agents, and it is based on the human practical reasoning model [Bratman 1987]. A BDI agent uses the concepts of belief, desire, and intention in a means-ends reasoning process, and its actions are organized in an execution plan built on top of (i) what the agent believes to be true, and (ii) what the agent desires to achieve as a goal [Konolige and Nilsson 1980].

2.3. Agent Communication Languages

An agent communication language (ACL) establishes a structure for the message exchanging between agents, both in type and meaning. However, agents don't simply exchange messages; they actually engage in *conversations* [Labrou 2001]. A conversation can be seen as a pre-arranged protocol or message exchanging pattern, oriented towards a specific task or objective.

We will limit the scope of the ACLs to the two most consistently used [Li and Kokar 2013, Kamdar et al. 2018]: FIPA-ACL and KQML. Both languages are based on the speech acts theory [Searle 1969] and are composed of different *performatives* [1]. They can be understood as sentences that describe and influence an environment at the same time. As such, the messages exchanged between agents can represent actions or communicative acts. Despite being relatively old, these standards are still being used to this date [Răileanu et al. 2018, Blos et al. 2018].

The FIPA protocol [2] is an effort to established guidelines to make platforms interoperable. FIPA-ACL is composed of 22 communication performatives. The most common performatives are:

- `inform`: The sender informs the receiver that a given proposition is true;
- `request`: The sender requests that the receiver execute some action;
- `agree`: The sender agrees to take some action, possibly in the future;
- `not understood`: The sender (eg, *i*) informs the recipient (eg, j) that it perceived the action performed by *j*, but did not understand it;
- `refuse`: The sender refuses to execute a particular action, explaining the reason for the refusal.

While FIPA-ACL only establishes interaction protocols between agents, there are also FIPA specifications for agent management used in conjunction with the communication standards. The basic components of these specifications are (i) the Agent Platform, where each agent has a unique identifier called AID (FIPA Agent Identifier); (ii) the Agent Management System (AMS), responsible for managing the Agent Platform; and (iii) the Directory Facilitator (DF), that allows the agents to publish services in a discoverable manner.

Similarly, the KQML language is also composed of communication performatives. It possesses three different layers:

[1]https://stanford.library.sydney.edu.au/entries/speech-acts/
[2]http://www.fipa.org/

- **Content layer:** where the actual content of the message resides, in the computer's own representation language;
- **Communication layer:** used to encode lower level communication parameters, such as the identity of both the sender and the recipient of the message;
- **Message layer:** provides the performatives used in the communication process, finding all possible interactions with KQML-compliant agents.

KQML also describes a special class of agents named *facilitators*. The *facilitator* is responsible for performing multiple communication services, such as maintaining a registry of services, routing messages to these services based on content, and providing mediation between communication parts. Differently from FIPA-ACL, these specifications are contained in the ACL definition, and do not depend on additional standards.

Despite being similar in multiple aspects, there are some characteristics particular to one or another ACL that make their use domain-dependent. We will not detail all the differences between the two ACLs at this point. These differences, however, can be crucial in choosing one or another ACL according to a specific set of requirements.

3. Related work

Integrating agents and the web services has been both proposed and experimented on since the late 1990s [Jennings et al. 1998, Lieberman et al. 1995, Etzioni 1996, Ardissono et al. 1999, Greenwood and Calisti 2004]. In particular, the idea of using agents *as* web services appeared in early 2000s [Hendler 2001, Huhns 2002]. Using BDI agents as web services was also proposed around that time by Dickinson and Wooldridge [Dickinson and Wooldridge 2005].

Using agents in conjunction with web services was further explores with platforms such as CArtAgO-WS [Ricci et al. 2010], which allowed the development of SOA applications populated by agents taking into account both the concepts of artifacts - "objects" or services that could be used by the agents - and the use of agent architectures (including BDI). Most of the existing work in this field, however, considers the Internet environment mostly as a means for communicating - which is from where our research motivation is originated.

Another related work [Radhakrishnan et al. 2018] presents a comparative performance study between SPADE and JADE, with focus on security in the cloud environment. The compared platforms were chosen due to reasons similar to our own, with the addition of considering knowledge transferring capabilities with the use of ontologies.

In our research, we would like to explore in detail how different Internet technologies could be used by agents and MAS - and not only for communication. Exposing agents as web services is an intuitive manner of taking advantage of the communication protocols already in place. However, the Web can be also used as a distributed environment *per se*. The existing interoperability provided by the SOA architecture can also be used for distributed reasoning and planning, for example, allowing a MAS to be fully implemented and deployed using technologies not necessarily particular to agents or MAS.

For this reason, it is important first to understand how the existing multiagent platforms could be used in this context, and what were their restrictions or limitations regarding the integration with the Web. The objective of the present work is to revisit web-based agents considering the current state of the Internet. We focus particularly on agents deployed as web services and MAS that use them. For this purpose, we compared different existing multiagent platforms in terms of capabilities and web-related capabilities. This comparison is detailed in the next paragraphs.

4. Multiagent platforms

In order to compare existing multiagent platforms, our first criteria was to choose which ones were open-source and were actively used by the multiagent community. In addition, we also consider the platforms recently used in the annual Multiagent Programming Context event [1] to select the following ones to compare: JaCaMo [2], JIAC V [3], JADE [4], SPADE [5], and SARL [6].

Since the BDI architecture is of particular interest for our research and JIAC and SARL do not provide support for BDI agents, we did not include them in the comparison presented in this work. Both JaCaMo and SPADE provide native support for BDI agents, and JADE uses a complementary layer called BDI4JADE [7]. In terms of implementation, however, JADE and SPADE provide a communication middleware for agents. JaCamo, on the other hand, is a meta-framework for Multi-Agent Oriented Programming (MAOP) implemented in Java. It is composed of three different frameworks, and due to its nature it could be modified to be used on top of JADE or other communication middleware. For this reason, we focused our analysis in the JADE and SPADE platforms.

Before presenting the evaluated platforms, we will describe the communication standards they use. Understanding these standards is not crucial to comprehend the platforms or their evaluation. It is important, however, to understand how they take advantage of the Internet environment from a lower-level perspective. Such understanding is related to the motivation behind the present work, and it will play a steering role in our future research.

4.1. JADE

JADE is a distributed middleware written in JAVA that requires a minimal knowledge of agent theory to implement FIPA-compliant agents. Each agent in JADE is hosted by a *Container*. While a host can contain other Containers, ultimately there is a *Main Container* on top of them all, responsible for providing the essential FIPA-ACL functionalities for the agents.

JADE uses the FIPA protocol not only for agent communication, but also for management. The FIPA protocol specifies an agent management infrastructure (named *Agent Platform*) and a service publishing platform (named *Directory Facilitator*). Using JADE allows the identification of every agent through the use of an unique identifier (AID) in a distributed environment. The Directory Facilitator, on the other hand, allows the agents to publish services that are visible to other agents.

[1]https://multiagentcontest.org/
[2]http://jacamo.sourceforge.net/
[3]http://www.jiac.de/agent-frameworks/jiac-v/
[4]http://jade.tilab.com/
[5]https://pypi.org/project/SPADE/ - version 2.*
[6]http://www.sarl.io/
[7]http://www.inf.ufrgs.br/prosoft/bdi4jade/

The message delivering mechanism used by JADE depends on an agent's location, and it is optimized to minimize the delivery time of messages. If two communicating agents reside in the same host, JADE use an internal set of message transport protocols. Also, if both agents share the same container, the message will not be serialized, but cloned, and the new object reference is passed to the receiving agent. Otherwise, communication between different containers uses JAVA RMI (Remote Method Invocation) [1]. A study of the performance of the message transport layer system of the JADE platform is presented in [Cortese et al. 2002], where it was shown that the JADE internal communication protocol was more efficient for intra-platform communication.

4.2. SPADE

SPADE (Smart Python multi-Agent Development Environment) is an agent framework written in Python, and it is also FIPA-compliant. The BDI architecture used by SPADE is based on a distributed schema: plans used by the agents are represented in the form of services. Therefore, instead of possessing a library with multiple plans, the agents' actions are defined by services published in the Directory Facilitator. In that sense, a plan becomes a composition of offered services that are accessed by the agent so it can accomplish its intentions.

Similarly to JADE, SPADE implements the agent management infrastructure specified by the FIPA protocol. However, it uses a different message delivering mechanism - mostly due to differences in the programming language used in the implementation (Python [2]). It is important to notice that in this work we used SPADE version 2.*, which is built using Python version 2.* .

5. Multiagent Platforms Comparison

We established the comparison criteria between JADE and SPADE 2 from a Web-related perspective. For this reason, we focused on (i) communication protocols and associated characteristics; (ii) organization representation; (iii) support to content language; and (iv) support to ontologies. The latter is especially important since ontologies can be used to model and consume knowledge in an inter-operable manner [Bechhofer 2009]. The results of the comparison can be found in Table 1:

Table 1. Comparison: JADE and SPADE

	JADE	SPADE
BDI	✓	✓
Directory Facilitator (DF)	✓	✓
Agent Management System (AMS)	✓	✓
FIPA-ACL	✓	✓
KQML		
Multiplatform	✓	✓
Explicit organization representation		
Ontologies	✓	

[1]http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html
[2]https://www.python.org/

It is important to mention that SPADE can use ontologies according to the FIPA-ACL specifications, which state that ontologies can be passed as parameters within messages. However, SPADE does not possesses functionalities related to creating and manipulating ontologies within its own environment. JADE, on the other hand, possesses such functionalities - despite the fact that JADE-based ontologies cannot be reused since they are modeled as Java classes.

It is also important to notice that JADE and SPADE possess multiple points in common - especially considering the adherence to the FIPA protocol. While these specifications are not necessarily mandatory for integrating agents and the Web, they are certainly important in scenarios involving a huge number of services. The DF functions as a yellow pages for services, cataloging and listing all web services that can be used by the agents. The AMS, on the other hand, is crucial for the study of message passing and process management within the MAS platforms. The MAS architecture and some aspects of the implementation are detailed in the next section.

6. Deploying agents as web services

As part of our study process, we modeled a simple MAS that used agents deployed as web services. The objective of this step was to verify how this could be built and deployed using the current technologies and tools. As mentioned before, we chose JADE and SPADE as the target multiagent platforms. Part of the reason behind this choice was related to the framework's web-related functionalities previously described. While the modeled MAS is not complex (and we will not describe it in detail in the present work), we intend to explore and evolve this implementation as our research advances.

6.1. JADE

JADE agents have already been used in conjunction with web services [Nguyen and Kowalczyk 2007a, Nguyen and Kowalczyk 2007b, Liu et al. 2006]. The JADE platform uses an add-on called WSIG[1] for this purpose, which handles all requests coming from the Web and sending them to the MAS.

Also, a JADE agent must possess an *Ontology* and a set of *Actions* so that its capabilities can be exposed as web services. Ontologies define the vocabulary and semantics used in the communication between JADE agents. Actions are objects that correspond to a request: once a request is made, it results in an action object containing the request elements that is sent to the JADE agent.

6.2. SPADE

SPADE does not possess a specifically created component or add-on to expose agents as Web Services. For this reason, we used a Python-based library called Werkzeug [2] that works similarly to JADE's WSIG. Using this library made it possible to manipulate requests via the Web and send them to the agent platform.

Since exposing agents as web services is not an existing functionality in SPADE, it also does not have a established set of rules for *Ontologies* and *Actions*, present in the

[1]https://jade.tilab.com/doc/tutorials/WSIG_Guide.pdf
[2]http://werkzeug.pocoo.org/

JADE platform. Its messaging mechanism, however, allows matching specific actions with messages received through the use of a labeling mechanism: inbound messages are checked for this label and processed accordingly, resulting in different actions depending on the label.

6.3. Implementation

In order to perform the proposed comparison, we felt the need to use a single MAS model implemented using the two different MAS platforms. For this reason, we used SPADE to re-implemented an existing MAS that was originally implemented in JADE, called Smart Agenda [Casals et al. 2018] (see Figure 1). This MAS functions as an agent-based personal assistant, and it will be explained in the next paragraphs.

At this point, we would like to mention that the original MAS implementation depends on intrinsic multiagent platform components and functionalities (as seen below). For this reason we chose to use each of the platforms' functionalities as extensively as we could, as a manner of effectively comparing their built-in capabilities.

6.4. MAS Smart Agenda Requirements and Architecture

The original Smart Agenda MAS is a web-based, agent-based personal assistant. It is deployed on the web, and all interactions between the user and the system are performed through the use of a one-page web application. After creating an account in the system, the user can use it to schedule new events or to modify existing ones, with the option of including other registered users in these events. It works pretty much as a web calendar application, with the difference that it uses intelligent agents not only for event coordination among multiple users, but also for its built-in recommendation system [Casals et al. 2018].

The recommendation system used by Smart Agenda takes into account user preferences (i.e. accommodation, transportation means, and periods of the day in which events should be scheduled) for the scheduling of any new events. It also allows the automatic re-scheduling of existing events in the case of any unforeseen circumstances, without the intervention of the user. A video detailing the operation and use of SmartAgenda (made available by the original authors of the system) can be found at https://bit.ly/Emas18Demo
.

From the MAS perspective, the system uses three different types of agents: *Coordinator*, *Manager* and *Agenda*. The Coordinator agent is responsible for handling all requests between the user and the system. In order to make the system scalable, every Manager is responsible only for a limited number or users. Every request made by an users is forwarded to a correspondent Manager agent, which is responsible for forward to respective Agenda agent. The Agenda checking if existing any restrictions at the moment. If everything is OK, then proceeds with the event scheduling according to the user's preferences. The Manager agent also plays a role in inter-mediating communications and scheduling conflicts when events involving more than one user are scheduled.

7. Discussion

In this work, we revisited some of the existing work on web-based agents and MAS. We also presented two different open source multiagent platforms: JADE and SPADE.

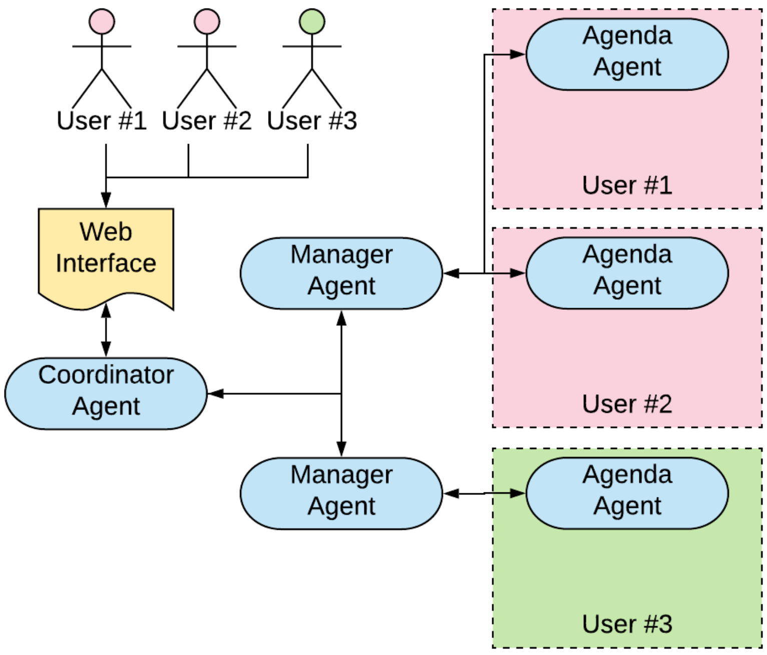

Figure 1. Implementation architecture for the Smart Agenda agents - adapted from [Casals et al. 2018]

Our objective was to compare them according to their support to developing agents to be exposed as web-services. The comparison was presented and a simple MAS was implemented using JADE and SPADE, with agents deployed as web services. Although not being complex, the idea behind the implementation was to create an MAS that could be further evolved and explored during the course of our research.

While the MAS architecture was simple enough to be modeled using a traditional BDI architecture, we found a few difficulties related to the implementation - mostly related to exposing the agents as web services. This is interesting because at this point we would expect that the libraries would be mature enough to allow a seamless implementation. Instead, we found problems varying from incomplete documentation to message passing between agents. This is somewhat worrisome considering that the first implementations exposing agents as web services (and specifically using JADE) are more than a decade old.

As a development ecosystem, Python features a variety of well-structured APIs (Application Programming Interface) as RESTful web services, thus providing multiple inter-operable functions that could be used by Python-based agent platforms. However, we encountered multiple difficulties while trying to establish agent communication within SPADE using web standards, even when using an auxiliary third-party library (Werkzeug). In comparison, we were able to do the same with the WSIG add-on for JADE, despite the fact that both JADE and SPADE are considered to be mature multiagent platforms. While this situation raises some concerns regarding the SPADE platform, it also validates our decision to evaluate MAS platforms considering simple SOAP web services, instead of using newer technologies. We intend to explore this point further in future work.

This work is meant to be a stepping stone to our research interests. Our long-term objective is to study and explore MAS platforms from both the modeling and the implementation perspective, considering both existing agent-oriented software engineering methodologies and new Web paradigms.

At this point, it was meant only to verify possible bottlenecks in the chosen MAS platforms related to the technologies involved (which we successfully achieved). We also intend to evolve and consolidate both implementations, so they can be complex enough to be analyzed in terms of performance, scalability, and robustness. Once we have a consolidated model implemented in different MAS platforms, we will be able to establish a common baseline for future performance and scalability comparisons.

Nevertheless, we could use these results to define an architecture to delivery, for instance, web-bots that can chat with web users to assist them in several ways.

8. Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001. This work is partially supported by ANEEL's Research and Development (R&D) Program.

References

Alarcon, R., Wilde, E., and Bellido, J. (2010). Hypermedia-driven restful service composition. In *International Conference on Service-Oriented Computing*, pages 111–120. Springer.

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Distributed Information Systems*, pages 3–27. Springer Berlin Heidelberg, Berlin, Heidelberg.

Ardissono, L., Barbero, C., Goy, A., and Petrone, G. (1999). An agent architecture for personalized web stores. In *Proceedings of the third annual conference on Autonomous Agents*, pages 182–189. ACM.

Bechhofer, S. (2009). Owl: Web ontology language. In *Encyclopedia of database systems*, pages 2008–2009. Springer.

Blos, M. F., da Silva, R. M., and Wee, H.-M. (2018). A framework for designing supply chain disruptions management considering productive systems and carrier viewpoints. *International Journal of Production Research*, pages 1–17.

Bratman, M. (1987). *Intention, plans, and practical reason*, volume 10. Harvard University Press Cambridge, MA.

Casals, A., Seghrouchni, A. E. F., Negroni, O., and Othmani, A. (2018). Exposing agents as web services in jade. In *International Workshop on Engineering Multi-Agent Systems*. Springer.

Cortese, E., Quarta, F., Vitaglione, G., and Vrba, P. (2002). Scalability and performance of jade message transport system. In *AAMAS workshop on agentcities, Bologna*, volume 16, page 28.

Dickinson, I. and Wooldridge, M. (2005). Agents are not (just) web services: considering bdi agents and web services. In *Proceedings of the 2005 Workshop on Service-Oriented Computing and Agent-Based Engineering (SOCABE'2005), Utrecht, The Netherlands*.

Erl, T. (2005). *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall, 1 edition.

Etzioni, O. (1996). Moving up the information food chain: Deploying softbots on the world wide web. In *Proceedings of the national conference on artificial intelligence*, pages 1322–1326.

Ferber, J. (1999). *Multi-agent systems: an introduction to distributed artificial intelligence*, volume 1. Addison-Wesley Reading.

Greenwood, D. and Calisti, M. (2004). Engineering web service-agent integration. In *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, volume 2, pages 1918–1925. IEEE.

Hendler, J. (2001). Agents and the semantic web. *IEEE Intelligent systems*, 16(2):30–37.

Huhns, M. N. (2002). Agents as web services. *IEEE Internet computing*, 6(4):93–95.

Jennings, N. R. (2000). On agent-based software engineering. *Artificial intelligence*, 117(2):277–296.

Jennings, N. R., Sycara, K., and Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous agents and multi-agent systems*, 1(1):7–38.

Kamdar, R., Paliwal, P., and Kumar, Y. (2018). A state of art review on various aspects of multi-agent system. *Journal of Circuits, Systems and Computers*, page 1830006.

Keen, M., Acharya, A., Bishop, S., Hopkins, A., Milinski, S., Nott, C., Robinson, R., Adams, J., and Verschueren, P. (2004). Patterns: Implementing an soa using an enterprise service bus. *IBM Redbooks*, 336:20–28.

Konolige, K. and Nilsson, N. J. (1980). Multiple-agent planning systems. In *AAAI*, volume 80, pages 138–142.

Kravari, K. and Bassiliades, N. (2015). A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11.

Labrou, Y. (2001). Standardizing agent communication. In *ECCAI Advanced Course on Artificial Intelligence*, pages 74–97. Springer.

Li, S. and Kokar, M. M. (2013). *Agent Communication Language*, pages 37–44. Springer New York, New York, NY.

Lieberman, H. et al. (1995). Letizia: An agent that assists web browsing. *IJCAI (1)*, 1995:924–929.

Liu, S., Küngas, P., and Matskin, M. (2006). Agent-based web service composition with jade and jxta. In *SWWS*, volume 6, pages 110–116.

Michelson, B. M. (2006). Event-driven architecture overview. *Patricia Seybold Group*, 2(12):10–1571.

Muldoon, C. (2007). *An agent framework for ubiquitous services*. PhD thesis, Citeseer.

Nguyen, X. T. and Kowalczyk, R. (2007a). Ws2jade: Integrating web service with jade agents. In Huang, J., Kowalczyk, R., Maamar, Z., Martin, D., Müller, I., Stoutenburg, S., and Sycara, K. P., editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 147–159, Berlin, Heidelberg. Springer Berlin Heidelberg.

Nguyen, X. T. and Kowalczyk, R. (2007b). Ws2jade: Integrating web service with jade agents. In *International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 147–159. Springer.

Pěchouček, M. and Mařík, V. (2008). Industrial deployment of multi-agent technologies: review and selected case studies. *Autonomous Agents and Multi-Agent Systems*, 17(3):397–431.

Radhakrishnan, G., Chithambaram.V, and K.L., S. (2018). Comparative Study of JADE and SPADE Multi Agent System.

Răileanu, S., Anton, F. D., Borangiu, T., and Anton, S. (2018). Design of high availability manufacturing resource agents using jade framework and cloud replication. In *Service Orientation in Holonic and Multi-Agent Manufacturing*, pages 201–215. Springer.

Rao, A. S. and Georgeff, M. P. (1991). Modeling rational agents within a BDI-architecture. In Allen, J., Fikes, R., and Sandewall, E., editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 473–484. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

Ricci, A., Denti, E., and Piunti, M. (2010). A platform for developing soa/ws applications as open and heterogeneous multi-agent systems. *Multiagent and Grid Systems*, 6(2):105–132.

Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

Searle, J. R. (1969). *Speech acts: An essay in the philosophy of language*, volume 626. Cambridge university press.

Tapia, D. I., Rodríguez, S., Bajo, J., and Corchado, J. M. (2009). Fusion@, a soa-based multi-agent architecture. In *International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008)*, pages 99–107. Springer.

Thiele, A., Kaiser, S., Konnerth, T., and Hirsch, B. (2009). Mams service framework. In *International Workshop on Service-Oriented Computing: Agents, Semantics, and Engineering*, pages 126–142. Springer.

Wooldridge, M. (2009). *An introduction to multiagent systems*. John Wiley & Sons.