Semantic Service Manager – Enabling Semantic Web Technologies in Multi-Agent Systems

Nils Masuch[†], Christian Kuster[‡], and Sahin Albayrak[†‡]

[†]DAI-Labor, Technische Universität Berlin, Germany
Email: {firstname.lastname}@dai-labor.de
[‡]German-Turkish Advanced ICT Research Centre, Berlin, Germany
Email: {firstname.lastname}@gt-arc.com

Abstract: Computer systems are becoming more and more ubiquitous. Many of today's application areas do not fit for monolithic approaches, but for distributed intelligent systems. The multi-agent paradigm is an appropriate and therefore popular approach for building these systems. To interpret messages and provided services the agents need a common ontology. Solutions to integrate standards like OWL-S allowing for using semantic web services into agent frameworks have already been presented in recent years. However, they mostly focus on service matchmaking, neglecting the design process. As a result, semantic web services are not used in actual agent frameworks. In this paper, we introduce the Semantic Service Manager, a tool that enables for implementing semantic web services in the multi-agent framework JIAC V and therefore enables the dynamic integration of new services into complex systems.

1 Introduction

Due to the increasing complexity in today's software systems the distribution of their respective components has become a key attribute for successful software evolution. The reasons for that are manifold, however there are certain characteristics that are clearly advantageous compared to monolithic approaches. The modular software structure offers the possibility to easily reuse developed components in other systems. Further, the processor and memory load can be distributed over several hardware entities, in specific architectures even dynamically. Finally, the main advantage is the flexibility of a distributed system. The modular software structure namely allows to exchange specific modules by others, either by modifying the invocation code or by using an adaptable software solution that dynamically searches for appropriate alternatives.

However, an essential requirement that comes with distributed software is interoperability. On a basic level, this means that there have to be defined protocols on how information is exchanged. Going a level higher, the next step would be to have a common data type representation in order to be able to process the data. These can either be simple or complex data types. However, if we want to build intelligent systems, that are able to automatically understand the meaning of exchanged information and due to that being able to decide what next steps to do, we have the need of complex ontology representations and rules

that allow for detailed reasoning.

A popular approach for developing distributed systems is the multi-agent paradigm, in which multiple so-called software agents are either working collaboratively on a common goal or can be interpreted as opponents each of them trying to fulfil its own goals. Since one software agent usually is not able to solve a problem, the agents interact with each other and must thereby ideally behave autonomously and adaptable. However, this raises the challenge of understanding the intentions and functionalities of the other agents. As described before, the agents must have a common ontology and rule understanding for that. In current approaches, most of the agent functionalities are defined in proprietary languages making extensibility and interaction with other entities than software agents of a specific framework difficult.

The service-oriented computing (SOC) community developed well-established standards to describe and invoke functionalities, such as WSDL/SOAP and REST services. But furthermore, solutions for the semantic description of software services have been proposed in the last years. Amongst others, these are OWL-S [BHL+04], WSMO [WSM] and SAWSDL[1], which extend usual information by preconditions and effects and define input and output information in a more expressive ontology language, such as the Web Ontology Language (OWL) [MvH04].

Integrating standards like OWL-S or WSMO into the agent community were sighted theoretically but often discarded in practice. One of the reason is that many of the standards come with a complex syntactical structure and are therefore not easy to handle and an appropriate tool support is missing that is supporting the developers of multi-agent systems (MAS) by describing their functionalities semantically. It is our conviction that the integration of semantic web technologies into MAS will only be successful, if there is a comprehensive support. Therefore, in this paper we propose a software engineering component called Semantic Service Manager (SSM) that provides semi-automatic semantic annotation of agent functionalities for the multi-agent framework JIAC V [LKK+13]. The component enhances the services by the semantic service description ontology OWL-S, which is based on OWL and provides rule support in the Semantic Web Rule Language (SWRL) [HPSB+04]. Further, we show that with this tool support, we are able to develop software solutions, where agents can search for appropriate services with the help of a semantic service matchmaker, making a first step towards adaptable software agents that are also able to integrate external services that are using the same service expression language.

The rest of the paper is structured as follows. In chapter 2 we will give a detailed overview about multi-agent frameworks and their support for integration of semantic web technologies and discuss the current research status in the service oriented computing community. In chapter 3, we present our implementation, the SSM, and its integrated features. Chapter 4 describes how this tool can be used for the development of adaptable and dynamic multi-agent systems by showing a use case in the context of the multi-agent framework JIAC V. Finally we conclude and describe the next steps of our work.

[1] http://www.w3.org/2002/ws/sawsdl/

2 Related Work

Although the approach to combine the concept of semantic web services with agent technologies is becoming a more and more important research topic, the agent frameworks that are mostly used today don't offer any or inadequate support. In practice, extensions to known frameworks or framework-independent approaches are found.

Frameworks like *Jason* [BH04] or *3APL* [HDBVdHM99] are geared towards research and focus on cognitive concepts, especially planning. Communication is merely realized through speech acts to exchange knowledge between agents. The agents in these frameworks do not have the possibility to provide services to other agents.

Even though the agent framework *JACK* [Win05] is a more pragmatic and successful approach that is focused on industrial requirements, planning is still the main concept as it concentrates on the implementation of the BDI (Belief Desire Intention) architecture. Here, agents can communicate over simple speech acts by sending serialized Java objects. With a plug-in JACK is capable of FIPA speech acts, but as the authors do not see the demand for FIPA-compliance in industrial projects it isn't further integrated into the framework. There is no way to advertise services. JACK offers a plug-in that adds the concept of teams to the framework, allowing the agents that are grouped in teams to delegate tasks to sub teams. Using another plug-in agents can be accessed via the Java Servlet API to enable the implementation of web applications.

JADE (Java Agent DEvelopment Framework) [BPR99] as another often used agent framework employs services that are provided by agents. In every JADE system there is an agent called DF (Directory Facilitator) that holds the so-called yellow pages. This is a directory that contains provided services. These services are described by values such as *name*, *language*, *ontology* and self-defined properties of the service, therefore they cannot hold semantic content. The search in the running system is based on these values. However, extensions to JADE have emerged that focus on semantic web services. *SEMMAS (SEMantic web services and Multi-Agent System)* [GSMBVGFB08] is a framework based on the JADE platform that combines semantic web services and MAS. For the semantic annotations OWL-S, an ontology in OWL describing semantic web services, is used, and an external tool as editor. The authors focus on the service matchmaking, so the semantic markup of services is disregarded. *JADEX (JADE eXtension)* [BLP03] as another extension to JADE tries to bring JADE closer to the industry and integrates the BDI model into JADE. JADEX has a different approach for the implementation of services, namely *Contract Oriented Programming*. Services are Java classes that implement interfaces. The methods of these interfaces can be annotated with preconditions and effects, though, these expressions are limited to the parameter and results of the methods. JADEX provides rich tool support throughout the whole agent life cycle, but service descriptions have to be written in Java code. This poses a cumbersome design process. Furthermore, ontologies have to be transformed into JavaBeans with external editors or manually generated from scratch, decoupling them from the actual ontologies. This makes the use of JADEX inflexible. Nevertheless, JADEX is very close to the approach presented in this paper.

An agent-framework-independent approach is shown by McIlraith, Son and Zeng [MSZ01].

In their paper they use the agent programming language ConGolog [GLL00] in combination with DAML for semantic markup of web services. They use wrapper functions to translate the information given by web services at the html layer forth to DAML and back. This means there is no explicit support of the development of the services itself.

Independent from the research in agent technologies various research is done on semantic web services with the goal to combine both the semantic web [BLHL+01] and web services [BHM+04] to enable automatic service discovery by, e. g. agents. One concept to collect web services in a registry is UDDI (Universal Description, Discovery and Integration) [Coa00]. Although UDDI has not established itself as a global registry for web services as planned, the research in semantifying web services in UDDI shows up-to-date problems. Today, most web services only provide information about their functionality at a technical level via WSDL, but no semantics. Specifications like OWL-S were developed to fill this gap. Paolucci et al. [PKPS02] enrich UDDI with DAML-S, a predecessor to OWL-S. In their paper they show a mapping from DAML-S Profiles to UDDI records. Furthermore, they describe a matchmaker component. However, the approach doesn't provide any concept for the design and implementation of the semantic web services. Srinivasan et al. [SPS05] and Talantikite, Aissani and Boudjlida [TAB09] use OWL-S to semantify the web services in UDDI. Jaeger et al. [JRgL+05] show another approach, but their work only focuses on improving the matching algorithm.

Most of the presented papers merely focus on service matchmaking, but forget the design process, being just as important. But some focus on the semantic markup. The *OWL-S editor* [EDM+05] is a plug-in for Protégé, an open source ontology editor. With it a complete creation and editing of OWL-S descriptions is possible. Furthermore, the service behind the description can be tested via a GUI. Drawbacks of the OWL-S editor are that it cannot handle multiple ontologies, because of limitations of Protégé. There is no connection to a framework, meaning that the editor lacks usability. The authors also do not see the benefits of a Java-to-OWL transformation. They argue that most commonly the service is developed before the implementation in code. Another editor for OWL-S is the *OWL-S IDE* [SPS06]. It is a plug-in for Eclipse and, in contrary to the OWL-S editor, supports the generation of OWL-S skeletons out of Java code. However, this generation is limited to basic types due to the missing support of ontologies. Furthermore, it doesn't support preconditions nor effects.

As a result it can be stated that, though the agent community has gaining interest in the use of automatic discovery of services, most agent frameworks do not provide this functionality. Extensions to these frameworks as well as contributions from the SOA (service-oriented architecture) community show promising approaches, but merely focus on service matchmaking and mostly ignore the need for tools to implement semantic services. Existing tools are independent from frameworks, meaning a cumbersome work flow. However, adequate tools that integrate into the design process are essential for the acceptance in the industry.

3 The Semantic Service Manager

At the moment agent actions in JIAC V are implemented in plain Java code with the possibility to publish them under a name, simply represented by a string. This publication can be done via an annotation applied to the method that implements the action or via an explicit registration in the service directory of the agent node, the runtime environment for the agents. Hence, services can only be searched by their name, and even more, must be known by the agent at compile time, because such a representation doesn't provide any semantic information. The agent has no ability to understand actions provided in the system and is therefore not autonomous nor adaptive at all.

The *Semantic Service Manager (SSM)* is a tool aiming at the development process of agent-oriented software to enable the semi-automatic annotation of agent actions. The main goal of SSM is to semantify JIAC V actions so that they can be searched by semantic properties. This leads to MAS where agents are capable of adaptively using services. SSM focuses on the design of these semantic services and therefore provides a tool to create new services from scratch as well as add semantic information to existing JIAC V actions. Further, services can be loaded from an online source or a local file or be imported from an open Java file containing one or more agent actions. OWL-S is used as format for the service description since it is a standardised specification to describe semantic services that is based on OWL. The tool is capable of managing multiple services at once. An ontology management component holding loaded ontologies eases handling parameter types. To formulate preconditions and effects OWL-S allows for using different established rule languages. SSM implements a parser that supports SWRL. Finally, the semantic services can be saved as an OWL-S file. The designed services can later be searched by a semantic service matchmaker, which is not part of this paper. In the following sub sections the key concepts are explained in detail.

SSM is implemented as a plug-in for Eclipse and provides a view that integrates into the workbench. Figure 1 shows the workbench of an Eclipse instance with SSM view on the left and an open Java file on the right. The service description in SSM corresponds to the method in the Java file. The transformation from Java methods to service descriptions is further described in section 3.5. The view contains fields for the OWL-S file location, the name and the description of the service. Furthermore, the input and output parameters and preconditions can be chosen and effects for these parameters can be formulated. For the handling of ontologies *OWL API*[2] in version 3.4.4 is used.

3.1 Ontology Management

As OWL-S is based on OWL to reference the parameters and the relationships between them SSM provides an ontology management component. Using this component the user can browse through ontologies and see the different entities that are defined in them. For every entity properties that are defined on it are shown. Entities can then be selected

[2] http://owlapi.sourceforge.net/

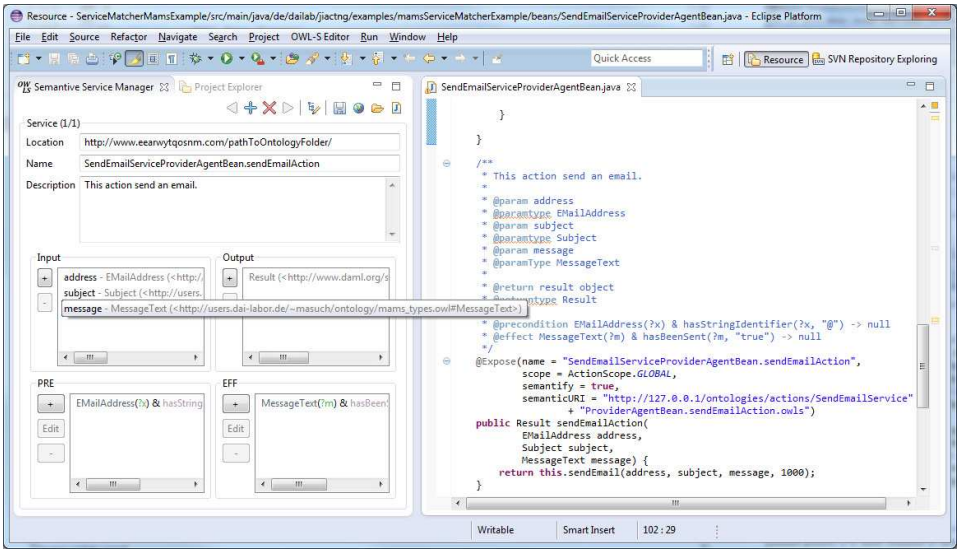

Figure 1: Semantic Service Manager: the service description (on the left) is derived from the Java method (on the right).

and used as parameters. To increase the usability and avoid unreferenced parameters they can only be selected from loaded ontologies. Figure 1 shows selected parameters for input and output. Here, the input parameter *message* refers to the ontology class *http://users.dai-labor.de/~masuch/ontology/mams_types.owl#MessageText*. New Ontologies are loaded into SSM either from an online or offline source and then are displayed in this component. Furthermore, ontologies that are defined in another format like Ecore (see 3.2) can be imported, too. The ontology storage is persistent and, thus, providing a database for OWL entities.

3.2 Ecore2OWL Transformation

In software engineering domain models are often used to describe a problem area. An open-source framework that has prevailed is the *Eclipse Modeling Framework (EMF)*[3]. In EMF domain models are built using the Ecore meta model, and the domain models are therefore called Ecore models. To integrate the possibility to use Ecore models SSM provides a *Ecore2OWL* transformation. Starting with an Ecore model rules are applied to the the model elements to transform them into elements of an OWL model. The transformation is realized using *ATL Transformation Language (ATL)*)[4] in version 3.4.0 and based upon the implementation of Guillaume Hillairet[5]. The transformation produces a persis-

[3] http://www.eclipse.org/modeling/emf
[4] http://www.eclipse.org/atl/
[5] http://perso.univ-lr.fr/ghillair/projects/ecore2owl.zip

tent local OWL file that is loaded automatically into the ontology management component.

3.3 Semantic Service Description

To hold the information of the defined semantic services SSM implements a data structure *Semantic Service Description*, which abstracts from the format of the OWL-S specification and embodies only necessary fields to specify services. Then in the export process this service description is transformed into a valid OWL-S file. In SSM a service description consists of the following: A name, a human readable description and a location where the OWL-S file is stored. This location has to be a valid URL pointing to the ontology. Furthermore, it contains the input- and output parameters as well as preconditions and effects.

At the moment, preconditions can only be made about the input parameters, and effects must refer to the output parameters. This behaviour could be extended to a more general range of logic statements, though, possible benefits have yet to be elaborated.

3.4 SWRL Rule Parser

In OWL-S it is possible to formulate preconditions and effects for the services through SWRL. The language provides different concrete syntaxes to formulate rules as well as logic. SSM being a tool designed for human users, it features the informal so-called *Human Readable Syntax*. In that syntax, rules are of the form $antecedent \Rightarrow consequent$ where antecedent as well as consequent are conjunctions of properties written $p_1 \wedge \ldots \wedge p_n$. Variables in $p_{1\ldots n}$ are denoted by a leading question mark. In figure 1 the list for preconditions contains the rule $EMailAddress(?x) \wedge hasStringIdentifier(?x, \text{“@”}) \Rightarrow null$. Here, the consequent equals *null* meaning that there is no further logic to be reasoned. The example says that the variable x being of type *EMailAddress* has to include the string “@” as precondition. This requirement depicts a very simple check if the email address is valid.

For entering the SWRL rules a text dialogue can be used. Only properties that are defined for the input- and output parameters are available, avoiding errors due to chosen properties that are not allowed in the current context. The rules are checked by a parser that is implemented using *ANTLR*[6] in version 4.1.

3.5 Java2OWL-S Transformation

Although in a process planned from the beginning as a semantic service it is more common to develop the service before implementing the actual code, for us it has shown that in practice the development process often is reversed. For this reason we think that a se-

[6] http://www.antlr.org/

mantic annotation for already existing services is necessary for a complete support of the development process.

To keep the Java2OWL-S transformation simple, the Java file has to be opened in the editor (see figure 1). By starting the transformation all methods in the Java file that are annotated with "@Expose" are identified as JIAC V actions and therefore are selectable in a dialogue. For every selected method a separate service description is generated. As in Java annotations can have a list of key-value pairs in the transformation process it is checked whether a OWLS-S file already exists (*semanticURI* in figure 1). If this file exists and can be found it will be loaded into SSM, otherwise the service description will be generated from the Java method. For this purpose the types of the parameters and the return type are matched against the available classes from the ontology management component. The name is extracted from the Java annotation, and the preconditions and effects are imported from the Javadoc tags "@precondition" and "@effect" and checked by the SWRL parser. When changes are saved in the imported service description the Java method gets updated as well.

This Java2OWL-S transformation is generic enough to be used for every Java method. However, in our implementation it is limited to methods annotated with "@Expose" since in JIAC V agent actions are marked this way.

4 SSM in Practice

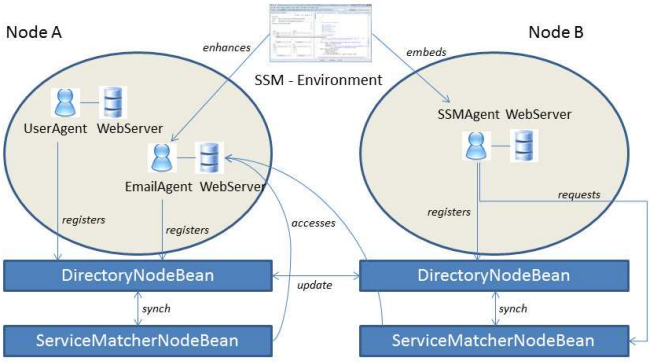

Figure 2: The SSM embedded in the JIAC V environment

4.1 Integration in JIAC V Methodology

One of the goals of SSM was its support and integration into the development methodology of the multi-agent framework JIAC V [LKK+13]. The framework itself comes up with a powerful discovery and messaging infrastructure, which allows to build up dis-

tributed systems even over network boundaries. In JIAC V, a distributed system is called platform. Each platform can contain multiple nodes (as seen in figure 2), which can run on several server. A node itself can host multiple agents, which can represent certain roles and provide different services. The system is flexible in a way that new agents can be deployed at runtime and are immediately able to communicate with other agents. Furthermore, nodes can implement so-called *Node Beans*, which provide functionalities for all agents on the node. Figure 2 shows a system with two nodes. Both nodes do provide two *Node Beans*, namely the *Directory Node Bean* and the *Service Matcher Node Bean*. The first one is responsible for the advertisement of services between the nodes and the second one is an implementation of an OWL-S based Service Matcher component called $SeMa^2$ [MHB+12]. In the figure, Node A contains two agents, the *UserAgent* and the *EMailAgent*. When an agent is being started it registers all its actions at the *Directory Node Bean*. If the action does contain a semantic service description it is being hosted by the agent on a web server and the address is communicated to the *Service Matcher Node Bean*. The Service Matcher then accesses this description and stores it in its repository of available services. On the other hand, the *Directory Node Bean* and the *Service Matcher Node Bean* of the other node are also being informed about the new services.

The SSM can now support the developer in two ways. On the one hand it can directly help annotating an action at design time. In this case the agent is not running and the service description is annotated as described in the previous chapter. The agent will load this description into its web server at start-up and will be available for the Semantic Service Matcher component. On the other hand, the SSM shall support the developer by finding an existing service that can be invoked. This option, we are currently working on, embeds its own agent, the SSMAgent. As the agent also has access to the service matcher component, which synchronises with all the other nodes via the *Directory Node Bean*, it can search for specific input, output, precondition or effect parameter. The Service Matcher will list all possible solutions and the developer can select the most appropriate one for integration.

4.2 Use Case for Semantic Service Descriptions for Agents

The overview of the SSM tool in figure 1 shows the semantic annotation of an email service of the *EMailAgent* shown in figure 2. The service has a precondition described in SWRL checking whether the mail address is valid. Further is has the effect that after invocation in successful cases the message will be sent to the recipient. Now, if the developer annotates its email service, another agent, for example the *UserAgent*, can either search via the SSM for a service at design time that delivers messages or it is also possible to integrate a search template into the code of the *UserAgent*. In this case the Semantic Service Matchmaker can reason on these information and as soon as the email service is online the service can be matched and invoked by the *UserAgent*.

5 Conclusion

Although the agent community has already revealed semantic web services as a promising concept to enable agents to understand intentions and functionalities of other agents, due to the lack of appropriate tools the potential of this concept is not used. The Semantic Service Manager (SSM) tries to fill this gap and and aims at the development of semantic web services in the agent framework JIAC V. It enables designing new services as well as semantifying existing agent actions. With the standard OWL-S actions can be described using ontologies and moreover preconditions and effects can be formulated in SWRL.

SSM is a promising tool for developing semantic web services in JIAC V as it allows for using ontologies in the context of agent actions to describe preconditions and effects at all. But even though the tool support for the design process already seems solid, many issues are undissolved. Currently, SSM lacks the distribution of the generated service descriptions in JIAC V. The concept to store and publish the service descriptions, as proposed in Section 4.1, has to be further elaborated and then implemented. An open question is here how to realize a lightweight web server component that is able to provide the OWL-S file.

Future work on SSM, besides the distribution of semantic service descriptions, can be the integration of the semantic service matchmaker at design time. This allows the developer to enter a service description into SSM and then let it search for services that match that description in running instances of JIAC V MAS. Additionally all running services can be visualised like in the ontology management component. This two solutions ease the design and gearing of new semantic services. Further transformations from WSDL to OWL-S and XSD to OWL are also possible next steps to achieve a greater support of different formats. Our goal is to combine the various different formats into OWL respectively OWL-S.

Acknowledgment

This work is partially funded by the German Federal Ministry of Education and Research under the funding reference numbers 01IS12049A and 01IS12049B.

References

[BH04] Rafael H Bordini and Jomi F Hübner. A Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net, 2004.

[BHL+04] Mark Burstein, Jerry Hobbs, Ora Lassila, Drew Mcdermott, Sheila Mcilraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. Website, November 2004.

[BHM+04] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard. Web services architecture. 2004.

[BLHL+01] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[BLP03] Lars Braubach, Winfried Lamersdorf, and Alexander Pokahr. Jadex: Implementing a BDI-Infrastructure for JADE Agents, 2003.

[BPR99] Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE–A FIPA-compliant agent framework. In *Proceedings of PAAM*, volume 99, page 33. London, 1999.

[Coa00] UDDI Coalition. UDDI Technical White Paper, 2000.

[EDM+05] Daniel Elenius, Grit Denker, David Martin, Fred Gilham, John Khouri, Shahin Sadaati, and Rukman Senanayake. The OWL-S editor–a development tool for semantic web services. In *The Semantic Web: Research and Applications*, pages 78–92. Springer, 2005.

[GLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector J. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(12):109 – 169, 2000.

[GSMBVGFB08] Francisco García-Sánchez, Rodrigo Martínez-Béjar, Rafael Valencia-García, and Jesualdo T. Fernández-Breis. Knowledge Technologies-Based Multi-Agent System for Semantic Web Services Environments. In Leszek Rutkowski, Ryszard Tadeusiewicz, LotfiA. Zadeh, and JacekM. Zurada, editors, *Artificial Intelligence and Soft Computing ICAISC 2008*, volume 5097 of *Lecture Notes in Computer Science*, pages 1222–1233. Springer Berlin Heidelberg, 2008.

[HDBVdHM99] Koen V Hindriks, Frank S De Boer, Wiebe Van der Hoek, and John-Jules Ch Meyer. Agent programming in 3APL. *Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[HPSB+04] Ian Horrocks, Peter F Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, Mike Dean, et al. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member submission*, 21:79, 2004.

[JRgL+05] Michael C. Jaeger, Gregor Rojec-goldmann, Christoph Liebetruth, Gero Mühl, and Kurt Geihs. Ranked Matching for Service Descriptions using OWL-S, 2005.

[LKK+13] Marco Lützenberger, Tobias Küster, Thomas Konnerth, Alexander Thiele, Nils Masuch, Axel Heßler, Michael Burkhardt, Jakob Tonn, Silvan Kaiser, and Jan Keiser. Engineering Industrial Multi-Agent Systems — The JIAC V Approach. In Massimo Cossentino, Amal El Fallah Seghrouchni, and Michael Winikoff, editors, *Proceedings of the 1st International Workshop on Engineering Multi-Agent Systems (EMAS 2013)*, pages 160–175, 2013.

[MHB+12] Nils Masuch, Benjamin Hirsch, Michael Burkhardt, Axel Heßler, and Sahin Albayrak. *Semantic Web Services: Advancement through Evaluation*, chapter SeMa^2 – a Hybrid Semantic Service Matching Approach. Springer-Verlag, Berlin / Heidelberg, May 2012.

[MSZ01] S.A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web services. *Intelligent Systems, IEEE*, 16(2):46–53, Mar 2001.

[MvH04] Deborah McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. Technical report, W3C, 2004. http://www.w3.org/TR/owl-features/.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia Sycara. Importing the Semantic Web in UDDI. In *Web Services, E-Business, and the Semantic Web*. Springer, 2002.

[SPS05] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. An efficient algorithm for OWL-S based semantic search in UDDI. In *Semantic Web Services and Web Process Composition*, pages 96–110. Springer, 2005.

[SPS06] Naveen Srinivasan, Massimo Paolucci, and Katia Sycara. Semantic Web Service Discovery in the OWL-S IDE. In *Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 06*, HICSS '06, pages 109.2–, Washington, DC, USA, 2006. IEEE Computer Society.

[TAB09] Hassina Nacer Talantikite, Djamil Aissani, and Nacer Boudjlida. Semantic annotations for web services discovery and composition. *Computer Standards & Interfaces*, 31(6):1108 – 1117, 2009.

[Win05] Michael Winikoff. JACK(TM) intelligent agents: An industrial strength platform. In *Multi-Agent Programming*, pages 175–193. Springer, 2005.

[WSM] Web Service Modeling Ontology (WSMO). http://www.w3.org/Submission/WSMO/.