AN AGENT-BASED COMPOSITION MODEL FOR SEMANTIC MICROSERVICES

Ivan Salvadori, Alexis Huf and Frank Siqueira
Department of Informatics and Statistics, Federal University of Santa Catarina, Brazil

ABSTRACT

Microservices are replacing the traditional monolithic Web applications by splitting them into multiple small and independent services that work together. Focused on doing one thing with excellence, microservices may be composed to provide more complex capabilities upon a given domain. This paper presents a composition model, aimed at composing semantic microservices for achieving more valuable outcomes. The proposed model is aligned with microservices architectural principles, such as independence of development, high cohesion, loose coupling, organizational alignment, composability, and so on. This paper also presents a reference architecture and a case study for the proposed model.

KEYWORDS

Semantic Web Services; Agents; Microservices; Composition

1. INTRODUCTION

The adoption of microservices is in continued expansion, and traditional monolithic applications are giving way to highly cohesive and loosely coupled sets of services (Newman, 2015). Microservices are designed to provide solutions upon a bounded context within a well-defined domain, resulting in multiple components that communicate and operate together. While some concepts are similar to SOA (Service Oriented Architecture), the focus on loose coupling precludes integration approaches based on central architectural entities, such as the service bus approach, where business logic may be concentrated on a single not cohesive component and highly coupled component. Despite the advantages such as technological decoupling, scaling, easier deployment, and better reuse, the adoption of microservices results in a more complex ecosystem with fragmented functionality, which requires additional communication and cooperation efforts.

Several service composition approaches can be found in the literature. However, there are few proposals that primarily address the microservices architecture. Stubbs et al. (2015) and Florio (2015) have proposed solutions that are specifically focused on microservices. However, their main goal is service monitoring, instead of service composition. Charif and Sabouret (2013) and Kumar (2012) adopt a multi-agent approach for service composition. These solutions could be applied to microservices, however composition does not take into account semantic descriptions neither aspects specific to microservices architecture, such as the distribution of functionality previously present on a single service.

This work presents a composition model for semantic microservices. In this model, microservices functionalities are modeled in terms of concrete desires, in which a given microservice is responsible for implementing operations that result in an outcome aligned with a predefined concept in a domain ontology. The proposed model makes use of agents capable of establishing a communication and negotiation process fulfilling abstract and more expressive desires using the simple concrete desires provided by microservices. This work also presents a reference architecture for the proposed model, which is defined at a higher level and independent from the technological stack employed to develop microservices.

The remainder of this paper is organized as follows: Section 2 summarizes the main concepts related to semantic microservices composition. Section 3 describes related research efforts found in the literature. Section 4 presents the proposed composition model for semantic microservices. A reference architecture is presented in Section 5. An illustrative case study is presented in Section 6. Finally, the conclusions and future work are presented in Section 7.

2. BACKGROUND

According to Newman (2015), microservices are small independent services. These services must be deployed as different artifacts, ideally on different machines and must work together by communicating across a network. There is no widely accepted measure of how small a microservice should be, however some rules of thumb are listed by Newman, such as keeping things that change for the same reason together, and separating things that change for different reasons. By splitting up a monolithic application into several microservices, deployment and maintenance are facilitated. It also makes easier to reach resilience and scalability, since there is no central point of failure, and due to the possibility of scaling only the more demanded microservices. According to Newman (2015), each microservice should be developed and maintained by a single team. This is an important characteristic that makes microservices independent from one another and from developers as well.

Considering that all microservices implement simple tasks, in order to execute more complex tasks, with more valuable outcome, microservices must be composed. According to McIlraith et al. (2001), automatic Web service composition involves the automatic selection, composition and interoperation of services to perform a given task that none of the available services is able to entirely perform. To allow such composition, a richer description of the services is necessary. Semantic Web Services are described with the use of semantic web technologies, allowing the development of complex composition algorithms that take into account the functionalities of services.

Multi-agent systems have a similar definition. Horling and Lesser (2004) define such systems as an aggregation of agents that perform simple tasks in order to fulfill more complex ones. Therefore, multi-agent systems provide a useful paradigm for splitting complex problems into sub-problems that can be solved by the computational elements of the system. In addition, Wooldridge (2009) highlight the potential of such systems for dealing with distributed issues. When a monolithic application is refactored into microservices, the goal is that each microservice implements only a set of highly cohesive simple tasks and is able to operate independently from other microservices, both characteristics also present in agents.

3. RELATED WORK

Stubbs et al. (2015) present a decentralized solution for communication and service discovery in Docker containers with embedded agents. These agents are able to monitor child containers and to emit custom events, in addition to membership messages to inform that containers joined or left the system or suffered a failure. With respect to discovery, membership events allow only simple tagging of services, therefore they do not provide all the required information for service composition.

Florio (2015) proposes a self-adaptive infrastructure based on multi-agent systems for managing multiple microservices deployed into Docker containers. In order to allow the system to scale up, and to improve availability and performance, agents that run a self-adaptive loop are attached to each Docker manager, and coordinate their actions through communication.

Charif and Sabouret (2013) propose a dynamic, decentralized and autonomous approach with four steps for service composition. The first step consists in the service consumer defining its needs, which may be a request for data or for an action to be executed. The second step consists in extracting keywords from the consumer request and in selecting a service based on these keywords. In the third step, services establish a communication process to undertake the consumer request. In this step, agents decompose the request into several feasible sub-tasks. Finally, in the fourth step, the selected services are activated and their responses are forwarded to a mediator capable of verifying that the result is correct. A drawback is that all communication is performed through synchronous messages, therefore the consumer has to wait for the answer, otherwise the process is lost.

Kumar (2012) presents an approach for service selection and composition based on multi-agent systems that adopts the Foundation for Intelligent Physical Agents (FIPA) communication model. In this approach, agents establish a negotiation based on multiple attributes. A dedicated coordinator agent is responsible for service composition. Multiple attributes, such as time consumption, reputation, among others, are combined into a utility value. Negotiation terminates when this utility value reaches a predefined threshold.

Only the first two of these works Stubbs et al. (2015), Florio (2015) have microservices as their primary target architecture, but they do not tackle the service composition problem. On the other hand, the works that leverage multi-agent systems for service composition, while applicable, do not target microservices architectures. These works also do not perform functional composition based on semantic description of services.

4. A COMPOSITION MODEL FOR SEMANTIC MICROSERVICES

Semantic microservices provide semantic descriptions of their capabilities and also manipulate data enriched semantically. In both cases, a RDF syntax such as RDF XML, Turtle, or JSON-LD, among others, is used. The proposed model is based on agents that are able to fulfill one or more desires, each one is applied to a single information element. To fulfill such desires, an agent may directly use a microservice, whose implementation is aligned with the desires and information elements, or it may coordinate other agents in order to fulfill his assigned desire, therefore composing the functionality of the available microservices.

Desires are used to represent a consistent unit of functionality applied to a domain information element. Such functionality is modeled by the application and may be modeled as fine-grained CRUD (Create, Read, Update and Delete) operations on specific domain classes, or as coarse-grained domain processes to be dynamically orchestrated. The granularity and number of desires to be provided by a single microservice is largely determined by the application domain. The goal in designing microservices, however, should be on minimizing both of these aspects while respecting the Single Responsibility Principle and taking into consideration the business boundaries as explained in (Newman, 2015).

In this model, desires are ontology individuals of a desire class, applied to an information element, which is also a class. A summarization of the ontology used to describe desires is shown in Figure 1. For example, suppose that in a public security domain, a client has the desire to obtain the criminal record of a person. In this case "get" would be modeled as a desire, and microservices providing the *GetInformation* (*CriminalRecord*) desire would provide functionality to this end. Desires document both customer needs and microservices, and are used to govern the microservice composition approach as described in this section.

```
usa:Desire a owl:Class .
usa:CompositeDesire a owl:Class.
usa:InformationElement a owl:Class .

usa:informationElement
    a owl:ObjecProperty;   rdfs:domain usa:Desire;            rdfs:range usa:informationElement.
usa:relatedDesire
    a owl:ObjecProperty;   rdfs:domain usa:CompositeDesire;   rdfs:range usa:Desire .
```

Figure 1. Summarized µSA (µService Agents) ontology, in Turtle syntax

There are three types of agents in this model: service, composing and client agents. A service agent is associated with a single microservice and has the purpose of representing the microservice in bidding processes. The service agent extracts all *concrete desires* from the microservice semantic description and registers them as provided desires. The composing agent, on the other hand, provides a single desire, which is described by an ontology as a *composite desire*. Composite desires are satisfied by fulfilling all documented related desires, in an order determined by the inputs and outputs of the selected agents and by the data provided to the composing agent. The third type of agent, a *client agent*, represents the client and coordinates the bidding process. This type of agent does not fulfill any desire directly.

Desires can be classified as either *concrete* or *abstract*, the former being a desire that can be completely fulfilled by a single microservice, and the later are those that cannot. In addition to composite desires, abstract desires include those for which only desires involving subclasses of the information element are offered by agents in the system. For this type of abstract desire, the information element hierarchy provides the information necessary for composing the available microservices. Both composition types are non-exclusive and may be combined in any order.

Since abstract desires provide more valuable results to service consumers, they demand more implementation effort in terms of functionality, development and time. Concrete desires, however, are easier to implement, require less code and may be developed and maintained by a single team. The characteristics of concrete desires are similar to the microservices principles, which make microservices suitable for developing this type of desire. However, it is important to provide to service consumers more valuable results

that can be achieved by combining concrete desires. In terms of services, the process of combining desires is referred to as microservices composition. Hence, the proposed model aims to produce a microservices composition method capable of producing more valuable results that were not available previously.

Hierarchical composition is performed as part of the bidding process, where the client agent broadcasts the bidding of the target desire to all desire-providing agents. Each agent replies with a bid specifying how it participates in fulfilling the desire and a set of non-functional parameters. In order to fulfill a desire $d(e)$, the set of desires offered during the bidding that are to be fulfilled is given by $\{d(e_i)$ s.t. $e_i \in E\}$, where $E = \{e_1,e_2,...,e_n\}$ and for all $e_i \in E$, e_i is equivalent to or subclass of e, and there is no $e_j \in E$ s.t. e_j is a subclass of e_i . For each $d(e_i)$ to be fulfilled, there may be more than one agent providing the desire, and in such situation, the agents are considered to be in competition. Competition is solved by using non-functional parameters of the providing agents, prevailing only the agent with the best parameters.

Considering the second type of composition, from the composite desire definition the agent knows that fulfilling the set of related desires, in any order, is both necessary and sufficient to fulfill the composite desire. Knowing this, the composing agent is responsible for bidding the fulfillment of each related desire, as previously described, and for selecting providers for each of them. Given all selected providers, the agent must then compute a possible workflow, taking into consideration the input/output descriptions provided by the selected agents. If, given the input data provided by the client, the workflow cannot be built, the failure is forwarded to the client with an explanation of the missing data.

In both composition types, the data produced by selected agents must be combined by the coordinating agent before it is returned to its client (another agent or the end user). Such combination is also necessary when a workflow computed by a composing agent is executed, and outputs obtained in previous steps need to be used as inputs on the next steps. If RDF is used ubiquitously by the semantic microservices, the underlying data model allows such combination to be trivially performed as a union of the RDF graphs. In fact, if the microservice architecture is obtained by decomposing a monolith that happens to consume and produce RDF, the only change on the client would be the need to specify his desire.

An application of the desire metaphor and this composition model is offering action-driven services on top of data-driven microservices. Data-driven microservices manage information elements and allow only basic operations, such as those specified by the HTTP *GET*, *PUT*, *POST* and *DELETE* verbs. These basic operations enforce basic business rules related to data validation and resources state. Domain specific actions can be defined as combinations of these domain-independent operations on information elements, and the complexity of combining these operations to achieve a business goal can be abstracted. Furthermore, the workflow for accomplishing the business goals is dynamically constructed from the current available microservices.

5. REFERENCE ARCHITECTURE

A reference architecture for the application of the composition model previously presented is shown in Figure 2. The architecture includes two infrastructural (Richards, 2015) microservices and the three previously mentioned agent types. The infrastructural Common Knowledge microservice stores all the information necessary for reasoning about desires and information elements, as well as the definition of the composite desires. The conversation repository acts as a bridge between the consumers from the external environment and the agents, by providing an HTTP interface to a message-based bidirectional channel between the consumer and a Client Agent allocated for that conversation. Conversations allow a prompt response to the client, which must not remain blocked while the desire is fulfilled.

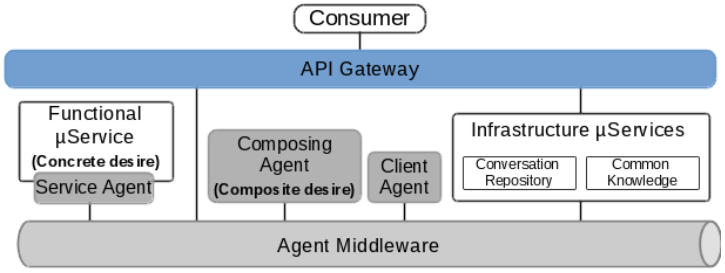

Figure 2. The reference architecture

The API Gateway assigns an idle client agent selected from a pool for each conversation. Both the desire and the provided input data given by the external consumer are forwarded to the client agent, which starts the bidding process (see section 4). Once the bidding process is complete, the client agent requests the fulfillment of the desire to all selected agents. Finally, the client agent combines the outputs received from the selected agents and forwards them to the client through the conversation repository.

It is mandatory that a service agent be able to understand the functionality exposed by the service it represents. While high-level functionality is modeled by desires and information elements, invocation details are also necessary, and can be described using existing service description languages. In the same way that microservices can be implemented using different technologies, no particular description language is recommended for use with the reference architecture. One example of such language is WSMO-Lite, which can be used to describe both SOAP and REST services (Roman et al., 2014).

Microservice environments also have the characteristic of high dynamism, with new microservices being made available and in some cases even competing with existing services. An agent middleware provides infrastructural support for agent discovery and for the implementation of negotiation protocols such as the one proposed in Section 4.

6. CASE STUDY

This section describes the application of the proposed composition model through a simplified case study based on an a system managing criminal, financial and immigration information about persons. This example demonstrates the details on how the composition model can be applied to a microservice architecture. It also shows how microservices are selected, composed and activated in order to fulfill a given desire.

Composition is driven by an ontology that describes the domain elements and desires that can be fulfilled in such domain. Figure 3 shows the domain ontology modeled for the case study. The entities on the left define the information elements managed by the system, while the entities to the right are desire classes that could be satisfied by microservices. Instances of the desire classes must be applied to information elements. For example, to register a criminal record, the desire instance would be *Register*(*CriminalRecord*).

Microservices implementations are aligned with the desires defined in the domain ontology, which means that a microservice should implement one or more functions that lead to the complete fulfillment of a desire. Note that the desire abstraction need not be present in the microservice implementation, but the service agent must be able to map a desire to a set of operations on the microservice.

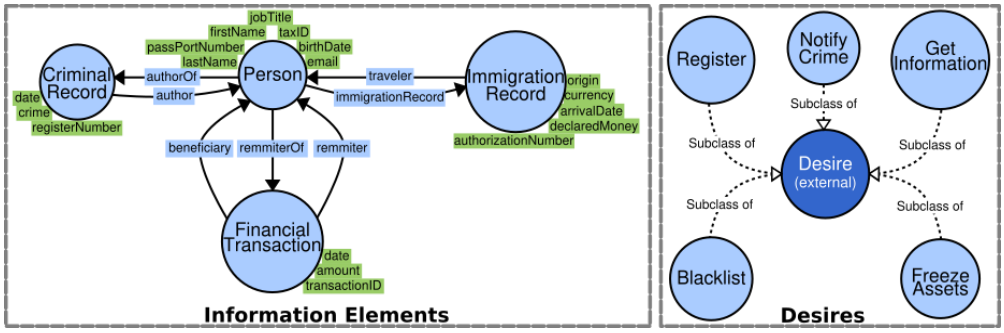

Figure 3. The case study domain ontology

The components of the case study are shown in Figure 4. The criminal records, immigration (travel) records, financial transactions and personal data were generated using the Mockaroo tool. In this example, for all information elements but Person, data is fragmented. This fragmentation simulates the fact that as actual data would be under responsibility of different organizations, different structures and technologies would be required to access this data, therefore different teams would each develop a different microservice for each data source. To avoid unnecessary complexity, instead of countries, only four regions (South America, North America, Asia and Europe), and two law enforcement agencies (Interpol and FBI) were considered. The symbol * beside information elements of the provided desires in Figure 4 represents that the microservice provides the desire not on the listed information element, but on a subclass of it.

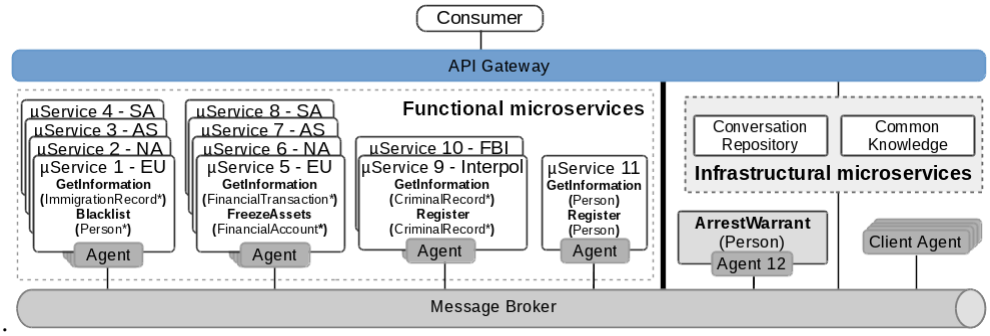

Figure 4. The use case components in the reference architecture

Figure 5 shows what a consumer would send to his client agent through the conversation repository in order to fulfill the *GetInformation*(*CriminalRecord*) desire on a person with a given tax ID. During the bidding process, only the agents associated with microservices 9 and 10 will place bids. Since there is no bid of greater value than these two, and these microservices are not in a competition state (as both offered subclasses of *CriminalRecord*), both will be invoked in order to fulfill the desire. The client agent that coordinated the bidding process will receive the output from both microservices and combine the two RDF graphs into a single graph that constitutes the response for the desire.

```
[ a secd:GetInformation; usa:informationElement sec:CriminalRecord ].
[ sec:taxID "733-11716-531-23" ].
```

Figure 5. Example of a desire to get Criminal Records from a person's tax ID

```
[   a secd:NotifyCrime
    usa:informationElement sec:CriminalRecord ;
    usa:relatedDesire
         [ a sec:Register; usa:informationElement sec:CriminalRecord ] ,
         [ a sec:GetInformation; usa:informationElement sec:Person ] ,
         [ a secd:Blacklist; usa:informationElement sec:Person ] ,
         [ a secd:FreezeAssets; usa:informationElement sec:Person ]       ] .
```

Figure 6. Example definition of a composite desire.

An example of a composite desire is *NotifyCrime*(*CriminalRecord*), defined in Figure 6. During the bidding process, the only agent to place a bid will be Agent 12. From the definition, the agent knows it has to fulfill four desires, but the order is not specified. Agent 12 coordinates a bidding process for each of the four related desires, and with all selected agents in hand, it must match the documented inputs and outputs with the agent input in order to establish a workflow.

During the negotiation that leads to the workflow definition, the agents do not exchange input/output documentations, but instead negotiate by exchanging sample data and validation results. The workflow is assembled as a sequence of layers, where the outputs of a layer are combined with the inputs to form the inputs of the next layer. The first layer includes all selected agents that accepted the data provided by the client as valid input. The *i*-th layer is composed by all agents that were not chosen for previous layers that accepted the merged inputs and outputs of the ($i-1$)-th layer as their input. Execution of the workflow follows the same logic as the workflow definition, except that actual data from the fulfillment of desires is used.

In this prototype, the input and output documentation of the agents for fulfilling a desire is given by SPIN (http://spinrdf.org/) constraints. Inputs are documented with either ASK or CONSTRUCT rules, while outputs are documented with ASK queries. This restriction on the output documentation is necessary so that the service agents are able to produce example output data. Service agents must extract the desires and construct the associated SPIN documentation from the microservices semantic description.

The adopted agent middleware was JADE (Java Agent Development Framework) and the microservices were implemented using Spring Boot. Due to the small scale of the environment, a single agent platform was sufficient to host all agents. The processing overhead, not including communication overhead introduced by the composition algorithm, was evaluated for both desires discussed in this section on an Intel i5 with 3.10GHz and OpenJDK 1.8.0_02-b14. The execution details and results are available in a public repository at https://dx.doi.org/10.6084/m9.figshare.3856155. Each test sample involves configuring the whole

environment, but the same JVM instance is reused. In the following conclusions, the first sample is discarded due to lazy initializations.

For the desire shown in Figure 5, the average overhead is 2.016 ± 0.109 milliseconds at 99% confidence. The desire shown in Figure 6 requires the evaluation of SPIN input constraints 16 times and the results for 200 runs is shown in Figure 7. The jitter in the first samples is due to the JVM Just-in-time compiler and such effects are not observed if the JVM is set to interpret-only mode (-Xint). The coefficient of variation (σ/μ) for total time in Figure 7 is 0.360 ms, however, with -Xint JVM option, it is only 0.0219 even with an average of 249.183 ± 0.729 milliseconds. For both desires, SPIN constraints checking is the most expensive operation, amounting to 62.188% ± 0.409% for the desire in Figure 5 and 61.702% ± 0.169% for the desire in Figure 6, both at 99% confidence.

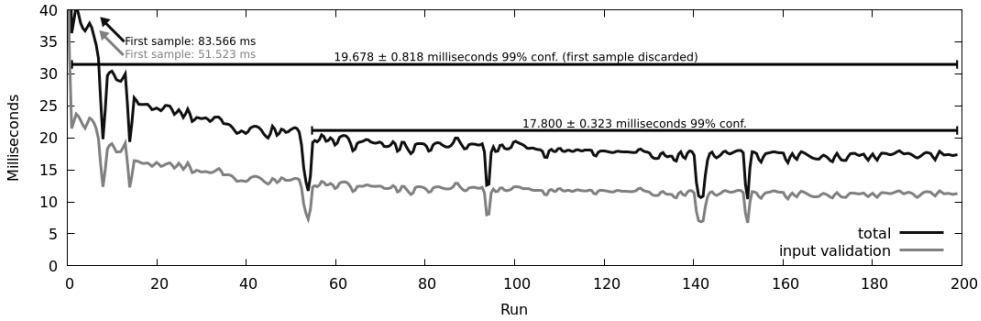

Figure 7. Processing overhead times for fulfilling a *NotifyCrime* desire

7. DISCUSSION

Similar to Kumar (2012), the service composition model described in Section 4 relies on a coordinator. However, instead of minimizing a utility value, agents cooperate to directly decompose a task identified by a desire applied to an information element. In comparison with Charif and Sabouret (2013), the proposed composition model defines user requests using the same modeling as is used to document the microservices, and does not relies on keyword extraction nor on a mediator to verify results.

With respect to how the composition is achieved, there is a distinction between planning techniques that consider pre- and post-conditions, such as Klusch et al. (2005), and the more numerous techniques that use a graph of services connected by their I/O parameters, such as Rodrigez-Mier et al. (2016). The abstraction of an *information element* corresponds to an input and a *desire*, at a large granularity, corresponds to a combination of outputs, pre- and post-conditions. This large granularity has two features that justify its adoption for microservice architectures. (1) It documents functionality at suitable level for decomposition into microservices. (2) Composite desires can be defined from other desires, which is a middle ground between common planning-based service composition and process templates. Therefore the composition model fits better for microservice architectures than traditional composition methods.

The main difference of the proposed composition model from existing service composition techniques is the direct targeting of the approach to microservice architectures. The fragmentation of functionality previously present on a monolithic application into several cohesive and decoupled microservices naturally creates the need for composition of services. The abstraction of a *desire* applied to an *information element* is intended to describe units of functionality offered by the microservices. It is also used for specification of composition requests eases evolution of the architecture by allowing microservices to be replaced by agents.

8. CONCLUSIONS AND FUTURE WORK

This paper presented a composition model and a reference architecture for discovering and composing microservices, supported by a multi-agent system. In our proposal, microservices can be easily and

dynamically added to the system, as the agents that are associated to them have no distinctive traits when started. Composition is guided by agents that fulfill a desire, as would a microservice. The adopted metaphor of desires is strongly related to the microservices architecture and also serves as a guideline during decomposition of a monolith into microservices.

Microservices are a recent architectural model that aims to reduce coupling and to build highly scalable and reliable systems. While container technologies, such as Docker, provide rudimentary service discovery capabilities, bridging customer needs to the offered microservices is still a problem to be solved. In our work we address this problem with service composition. Unlike BPEL engines, common in more traditional SOA environments, the process to fulfill the user needs is dynamically discovered and executed by agents.

More complex compositions are likely to arise when the available services are not described by a single ontology. Third party services and ontologies introduce not only heterogeneity, but also bring numerous additional possibilities for composite processes. In future work, the proposed architecture will be adapted and applied to such more complex and heterogeneous environment.

REFERENCES

Yasmine Charif and Nicolas Sabouret, 2013. Dynamic service composition enabled by introspective agent coordination. *Autonomous Agents and Multi-Agent Systems*, vol. 26 issue 1 pp. 54–85. ISSN 1387-2532. doi: 10.1007/s10458-011-9182-5.

Luca Florio, 2015. Decentralized self-adaptation in large-scale distributed systems. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, pp. 1022–1025, New York, New York, USA, 2015. ACM Press. ISBN 9781450336758. doi: 10.1145/2786805.2803192

Bryan Horling and Victor Lesser, 2004. A survey of multi-agent organizational paradigms. *The Knowledge Engineering Review*, vol. 19 issue 4, pp. 281–316. ISSN 1469-8005. doi: 10.1017/S0269888905000317.

Sandeep Kumar, 2012. Agent-based semantic web service selection and composition. *Agent-Based Semantic Web Service Composition*, SpringerBriefs in Electrical and Computer Engineering, pp. 15–25. Springer New York, 2012. ISBN 978-1-4614-4662-0. doi: 10.1007/978-1-4614-4663-7_3.

Karl Matthias and Sean P Kane, 2015. *Docker: Up and Running*. O'Reilly Media, Inc., Gravenstein Highway North, Sebastopol, CA 95472, USA.. ISBN 9781491917572.

Matthias Klusch et al, 2005. Semantic web service composition planning with OWLS-Xplan *AAAI Fall Symposium on Semantic Web and Agents*, Vol. 5.

S.a. McIlraith, et al, 2001. Semantic Web services. *IEEE Intelligent Systems*, vol. 16 issue 2 pp. 46–53. ISSN 1541-1672. doi: 10.1109/5254.920599.

Sam Newman, 2015. *Building Microservices*. O'Reilly Media, Gravenstein Highway North, Sebastopol, CA. USA. ISBN 9781491950357.

Rodriguez-Mier P, 2016. An Integrated Semantic Web Service Discovery and Composition Framework. *IEEE Transactions on Services Computing*, vol. 9 issue 4 pp. 537-550. ISSN 1939-1374. doi: 10.1109/TSC.2015.2402679.

Mark Richards, 2015. *Microservices vs. Service-Oriented Architecture*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. O'Reilly. ISBN 9781491952429.

Dumitru Roman, et al,2014. WSMO-Lite and hRESTS: Lightweight semantic annotations for Web services and RESTful APIs. *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 31: pp. 39–58. ISSN 15708268.

Joe Stubbs, et al, 2015. Distributed Systems of Microservices Using Docker and Serfnode. *7th International Workshop on Science Gateways*, Budapest, Hungary, pp. 34–39. ISBN 978-1-4673-7459-0. doi: 10.1109/IWSG.2015.16.

Michael Wooldridge, 2009. *An Introduction to MultiAgent Systems*. Wiley Publishing, 2nd edition. ISBN 0470519460, 9780470519462.