Semantic Service Management for Enabling Adaptive and Evolving Processes

Johannes Fähndrich, Tobias Küster, and Nils Masuch

DAI-Labor
Technische Universität Berlin
Berlin, Germany
e-mail: {firstname.lastname}@dai-labor.de

***Abstract*—With the rise of new paradigms like the Internet of Things, where thousands of devices and services of different providers are to be connected to complex processes, service-oriented approaches come to the fore. However, current solutions still lack of comprehensive methodologies how to dynamically manage and combine services to fulfil the given goals. In this paper we present a semantic-based service management methodology that enables the semantic description of services and provides an automatic service discovery and composition solution at design- and runtime. Furthermore, we present development tools that support the usage of semantic web technologies and we describe an execution environment where the approach is embedded. We conclude with an evaluation scenario from an e-mobility research project.**

Keywords–Semantic Service Enhancement; Semantic Service Matching; Automated Service Composition; Model Transformation; BPMN Processes; OWL-S; Semantic Service Descriptions

I. INTRODUCTION

The ever increasing digitalization of our societies leads to a vast amount of new possibilities, but also challenges. In the meantime, many companies, administrations and devices share their data or functionalities with others via application programming interfaces (APIs) or services respectively. Examples are the smart home or the transportation domain. In the first case, many different devices, such as smart meters and household appliances are addressable and can be regulated remotely. In the second case, the market is being extended by new services, such as car-sharing, bike-sharing and ride-sharing offers, which are provided digitally and where the user can find, reserve and unlock the most appropriate one via an API. Furthermore, the vehicles themselves can be configured via services and the environment is also becoming more digitized (charging-stations, parking spots, traffic analysis services, etc.). And even more sophisticated is the approach of the Internet of Things (IoT) which intends to connect services across domain borders.

However, in all cases there are some huge challenges that have to be overcome in order to exploit their potential. At first there is the requirement of finding a service. Different approaches like Universal Description, Discovery and Integration (UDDI) have been proposed, but none really has made it into the market. Second, there is the need for interoperability. Since a homogeneous data environment in open, extensible platforms is unrealistic, automated mapping solutions between models or ontologies respectively are one potential approach. And finally, due to the increasing amount of services, there is a strong requirement for automatic interpretation of services and their composition to value-added functionalities.

Especially for the last challenge, semantic technologies are an appropriate approach by providing structured data to machines. However, this does not come without a price. The management overhead can be immense especially for developers not familiar with semantic technologies. Therefore it was our goal to develop a semantic-based service management methodology that considers the whole life-cycle of semantic services including more sophisticated algorithms for automation. More concretely, we provide development tools for model transformation, for the semantic description of services and their deployment in order to set up a service. Furthermore, we propose how to find and match services at design-time and how to easily integrate them either to Java code or into a Business Process Model and Notation (BPMN) editor. Based upon that we developed comprehensive matching and service composition techniques that can be used both at design-time and at runtime.

This paper is structured as follows: In Section II, we present the different components that constitute our approach to semantic service engineering, and in Section III we show how those components are combined to form a holistic development method for semantic services and their composition. In Section IV, we demonstrate how the different components and the method have been applied in a research project in the e-mobility domain. Finally, we present some related approaches in Section V before we conclude in Section VI.

II. COMPONENTS

First, we will describe the several components that make up our approach to semantic service engineering. We subdivided this section into three parts: First, we will have a look at the fundamental aspects, i.e., the semantic service matcher and planner, and describe their behaviour in detail. Then, we introduce different tools that help in the development of semantic services and in their aggregation and orchestration to complex, value added processes. Finally, we present the execution environment, making use of a multi-agent framework while at the same time being fully interoperable with existing Web Services Description Language (WSDL) and Representational State Transfer (REST) services.

A. Semantic Service Core

Since the beginning of research in semantic service matching, matchmakers have matured in precision and recall [1]. Thus, the focus of service matching has shifted to the integration of non-functional parameters and formal modelling of system properties. The development on the Service Matcher that had the best Normalised Discounted Cumulative Gain

(NDCG) value in the last S3 contest in 2012 [1], called $SeMa^2$, has been focused on formalising and distributing the architecture of $SeMa^2$ and enabling a learning mechanism to customise the matching results to a given domain. We start this section by describing how we modelled the architecture, the matching probability, its aggregation and which parameters for the learning can be extracted. For an even more detailed discussion about $SeMa^2$, we refer to [2].

1) Architecture of a modern Service Matcher: The service matching task can be broken down into subtasks like matching the inputs of the request and the advertisement, or comparing their textual descriptions. In the $SeMa^2$ architecture each of these subtasks has been explicitly encapsulated in a so called *expert* which can be distributed following the agent paradigm.

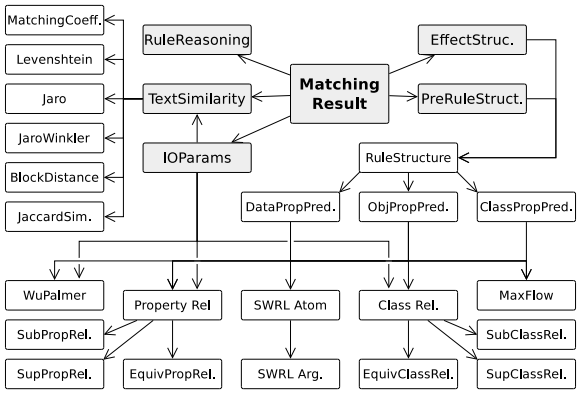

Figure 1. Expert System of the $SeMa^2$. High-level experts are composed of low-level experts, all contributing to the Matching Result.

As shown in Figure 1, the $SeMa^2$ consists of 28 different experts, which are dependent from each other (edges of the graph). The "Matching Result" represents the overall result of a matching request. It is also defined as an expert as it aggregates the results from the opinions of four types of experts: the text similarity expert, comparing the textual descriptions of a service, the in- and output parameter expert looking at the parameters and results of the services, the effect structure expert evaluating the similarity of effects, and the rule reasoning expert which evaluates whether the precondition and effect rules are satisfied with the same parameters. Each of those experts uses other experts to help forming its opinion, expressing the matching score of one aspect of a service. Thus, each expert encapsulates such a scoring method, which can be reused by multiple experts or extended with new scoring as the architecture evolves.

2) Probabilistic model of opinion: The different opinions of the experts are formalised by utilising the results of Morris [3], as probabilities $p_i(R, A)$. As an expert i observes aspects of a request R and advertisement A and calculates their distance. We can abstract this opinion as $p_i(\Theta|d)$ where Θ is the subject of interest and d are the observations. $p_i(\Theta|d)$ could be interpreted as a degree of belief of Θ observing data d. For more details see [2].

To aggregate the opinions of the different experts, an Opinion Pool is used. Here, a weighted mean of the opinions is created, for which we chose a weighted arithmetic mean called linear opinion pool [4] in a previous work [2]. This arithmetic mean has been generalised by Genest [5] to be able to use weights in the interval $[-1, 1]$ in a more general class of linear opinion pools. With this formalisation, the quality of the different aspects can be weighted during the aggregation. Choosing those weights is done during the learning phase.

3) Learning Semantic Service Matcher: Selecting weights for each experts instances ($SeMa^2$ for now has 128 experts instances), we do not only assess the performance of the expert, but also the quality of the description of the service, the ontologies of the domain and if present specific description aspects of a domain. These interdependencies are the reason why we are unable to learn the performance of an expert in general and reuse the weights for other matching domains.

For the learning, $SeMa^2$ implements different standard learning mechanisms, reaching from genetic algorithms implemented with the Watchmaker Framework [6] to simulated annealing [7]. For the statistical evaluation the Semantic Web Service Matchmaker Evaluation Environment (SME2) tool [8] is used, calculating the NDCG of each expert and adapting its weight according to the optimisation strategy used during the learning. As a drawback, this ability to adapt to the domain makes an offline learning phase necessary, where a test collection of example services needs to be defined, including a relevance rating for the training set of service to be used by the SME2 tool.

4) Semantic Services Planner: The ability to automatically compose services to reach a given goal is called service planning [2]. The service planner based on the $SeMa^2$ utilises the service matcher for three tasks: first, to reason about effects and preconditions to find applicable service. Second, to reason on parameter selection for grounding the services and third, to apply the execution of a service to reach a new state.

Name: ServicePlan
Input: S_{start}, S_{goal}, Services **Output:** Service Composition

1: $path \leftarrow []$
2: $Closed \leftarrow \emptyset$
3: $Open \leftarrow \{S_{start}\}$
4: **while** $s \leftarrow StateSearch.next(Open)$ **do**
5: **if** $s \notin Closed$ **then**
6: **if** $s = S_{goal}$ **then**
7: **return** $reconstructPath(path + [s])$
8: **end if**
9: $grounded \leftarrow ServiceSearch.UsefulServices(s)$
10: **if** $grounded \neq \emptyset$ **then**
11: $succ \leftarrow \{execute(s,\ g) \mid g \in grounded\}$
12: $Open \leftarrow Open \cup succ \setminus Closed$
13: $path \leftarrow path + [s]$
14: **end if**
15: $Closed \leftarrow Closed \cup \{s\}$
16: **end if**
17: **end while**
18: **return** failure

Figure 2. Service Planner algorithm

The algorithm in Figure 2 describes a standard planning approach applied to service planning. Here, the contribution is a planning in the service world without translating the service to the Planning Domain Description Language (PDDL) or similar to solve the planning problem.

The search used is defined in the function *StateSearch.next(Open)*. Depending on the implementation of the state search, the next state to be extended is selected. Here an A^* or equivalent algorithm can be used. In each state s that will be extended next, the selection of the services and their grounding is formalised in the function *ServiceSearch.UsefulServices(s)*. Here a set of grounded services is selected, which define the transition to the following open states. The state transition function is given by *execute(s, g)*, where the output and the effect of a service are integrated into the given state s. This is a theoretical execution, since the execution at runtime includes backtracking and a context sensing mechanism to sense the effect of a service. After extending a multitude of nodes during the search of the state space, the function *reconstructPath(path)* reduces the path from the goal to the start state to a minimal call of services.

The complexity of algorithm 2 depends on the implementation of the state search and state pruning mechanism, being the heuristic which selects useful services, including the complexity of the service matcher used. In general, the worst case complexity of such an algorithm is exponential [9, p.72].

By planning on services we accept a number of challenges:

- *Service Grounding* checks all parameters of services to be executed next and creates all combinations of individuals that fit those parameters. These combinations lead to multiple (possibly infinite) grounded services out of one service description. Here the challenge lies in the selection of continuous parameters.
- *Output Integration* into the state poses a challenge since it is not clear how a service without effect can influence the state. One example of such services are information providing services, which are not world altering services [10]. Thus, here we create an assertion of the class of the output, creating an appropriate individual, equivalent to the "AgentKnows" of Doherty et al. [11].
- *Semantic Web Rule Language built-ins (SWRLb)* are mathematical extensions like "greater than", string manipulations or description of time. Additionally, lists are modelled in SWRLb but are not supported by reasoners like Pellet [12].
- *Semantic Web Rule Language XML Concrete Syntax (SWRLx)* is an extension to the Semantic Web Rule Language (SWRL) allowing to model individual creation, creation of classes and properties. This is vital to the service planning, because service execution might create individuals or classes, which can not be modelled without SWRLx built-ins.

B. Development Tools

The method for semantic service management and development makes use of two development tools, which are both implemented as Eclipse plugins [13] and thus can seamlessly be integrated into the developer's usual workflow.

1) Semantic Service and Ontology Manager: In order to be able to integrate intelligent planning algorithms, the environment has to come up with the necessary infrastructure. One essential requirement in this respect is the semantic description of functionalities or services. Since current standards such as the Web Ontology Language for Web Services (OWL-S) [14] are not easy to describe from scratch, we developed a plug-in called Semantic Service Manager (SSM) [15], providing a set of features supporting a semi-automatic description of services. The core of SSM is an Ontology Manager, which enables the developer to include and utilize Web Ontology Language (OWL) ontologies for the application in semantic service descriptions. However, since many development approaches use other languages to specify the domain of concern, such as the Eclipse Modeling Framework (EMF), the Ontology Manager also provides a transformation process from EMF to OWL.

Based on the Ontology Manager the developer is then able to describe the service according to name, description, input and output parameters and finally preconditions and effects. The latter ones can be described via the Semantic Web Rule Language (SWRL) and for this purpose SSM comes with a syntax highlighting editor and structure parser. The description can then be utilized in different ways. Either it can be deployed to a semantic service repository (see Section II-C), it can be sent to a BPMN process (see next paragraph), or it can be linked to a service of the multi-agent framework JIAC V (Java-based Intelligent Agent Componentware, version 5) [16]. With these options at hand, the developer can easily connect semantic descriptions to services and is able to deploy them immediately.

The second purpose of the SSM is the search and utilization of existing and running services within a distributed environment. Therefore the SSM provides a *Service Discovery View* where the developer can define (incomplete) parameters of a service and search the platform directory using the $SeMa^2$ matcher. The developer can also adapt the weightings of the different matching techniques used. After selecting one of the services they can either be pushed to the *Visual Service Design Tool* (VSDT) to use it within a BPMN process, or a code inclusion function can be triggered that inserts the service call code into the open Java window.

2) Visual Service Design Tool: While basic services are usually implemented in the form of Java classes or equivalent, for service compositions business process modelling notations have proven useful. Using the VSDT, existing semantic services can be orchestrated to complex processes [17] using the BPMN notation [18].

The VSDT integrates with the Semantic Service Manager view in the way that services from the SSM can be imported into the VSDT. With a single click in the User Interface (UI), an according service description is added to the currently opened VSDT process, together with data types representing the different ontology concepts. That service can then be used in a service task and combined with other services to a complex process.

Next, those processes can be exported to executable languages such as BPEL (Business Process Execution Language) processes [17] or JIAC agent behaviours [19], being the execution environment used in this approach. In the case of JIAC agents, VSDT processes can either be compiled to JIAC beans, encapsulating an accordant behaviour, or they can be interpreted directly. In this work, we will focus on the interpreting approach, as further described in the following section.

C. Execution Environment

The services are executed as part of a JIAC multi-agent system. This way, each service is running on an individual agent, providing an adequate level of modularity and encapsulation. The environment also provides interfaces to other types of (web) services, such as SOAP (Simple Object Access Protocol) and REST, which can be integrated transparently with JIAC.

1) JIAC V Multi-agent Framework: The execution environment is based on *JIAC V*, a multi-agent framework also incorporating many aspects of service-oriented architectures [16]. The agents are situated on agent nodes (runtime containers). Each agent's behaviours and capabilities are defined in several agent beans, providing different general and application-specific functions (see Figure 3).

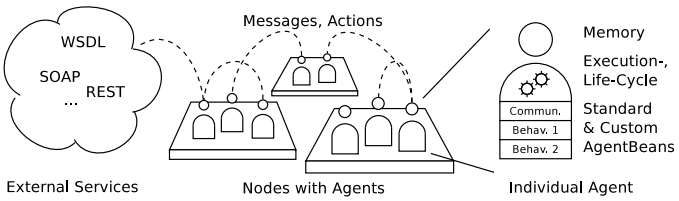

Figure 3. Components of a JIAC multi-agent system and individual agents (adapted from [20])

Complementary to message-based communication, one of the core mechanics of JIAC agents is to expose *actions*. Depending on its scope, an action can be found and used by other components of the same agent, by other agents on the same node, or by any agent on the network. Each JIAC agent node has a directory of known agents and actions, both on the same node as well as on other nodes, that can be used for querying and finding specific agents and actions using according templates. Given just the name, or the inputs and outputs of an action, the directory will find and return an action that matches that template (if such an action exists), which can then be used for creating an according intention.

For integration with other services, the WSDL- and REST-services integration beans can be used. Those components do both have two effects: First, all the JIAC actions accessible via the directory will be exposed to the outside world as according WSDL or REST services, respectively, and second, additional JIAC actions will be created and exposed, representing each of the WSDL and REST services known to those beans. Thus, JIAC agents can seamlessly and transparently be integrated with both, REST and WSDL services.

Integrating the semantic service matcher into JIAC was very natural and straightforward. Whenever a semantic service template (as opposed to a plain JIAC action template) is passed to the directory, the directory will delegate it to the semantic service matcher bean, which will return the best matching service. To the agent invoking the service, it is fully transparent whether it is a standard JIAC action or a semantic service.

In order to utilize the service matching and planning functionalities within the JIAC environment it was necessary to extend the existing action model for agents by means of a semantic service description model. The model is oriented towards the OWL-S standard dividing information into Profile, Process and Grounding parts. The latter can either reference JIAC action information but it can also define WSDL or REST attributes.

Loosely coupled to the JIAC environment is the Semantic Service Repository. Each platform can host multiple of these repositories, where the developer can deploy and manage its service descriptions. As this seems pretty static on first sight, we included a mechanism in which only in cases when a service is running and recognised by the directory, the linked service descriptions are considered for matching.

2) JIAC based BPMN Interpreter: One of several application-independent components for JIAC agents is the process interpreter bean, enabling the agent to interpret and execute BPMN processes created with the VSDT.

The process interpreter bean is composed of three layers: First, the *process interpreter bean* itself provides actions for adding processes to be interpreted and for managing already running processes. Also, it acts as an interface to the agents, providing functionality for sending and receiving messages and invoking other actions from within the BPMN processes. Finally, it exposes all the processes (that have an according start event) as actions so they can be used by other agents.

Whenever a process diagram is added to the process engine bean for interpretation, an *interpreter runtime* is created, which is responsible for each process spawned from this process diagram. It keeps track of events and creates a new instance of that process whenever an event corresponding to the respective start event occurs. Different volatile *process instances* are responsible for running the processes spawned by the runtime, executing the different activities and keeping track of the current state of the process, i.e., which activities are ready for execution, as well as the values of the different process variables.

Making use of JIAC's communication and service infrastructure, the interpreted processes can automatically make use of other JIAC actions, and – if the respective proxy beans are present – of WSDL and REST services. If the semantic service matcher is installed in the node, it is automatically used for finding services according to the templates used in the processes. The current state of the interpreter bean – the active runtimes, their respective process instances, and their internal states – can be monitored using a simple UI, also providing an interface for manually starting processes and for the processes to interact with the user, e.g., for BPMN user tasks, or for querying missing service parameters.

III. Methodology for Semantic Service Development

In the following, we will sketch a process of how the different components introduced in the last section are used together to form a methodology of semantic service engineering. At its base, the method is similar to other software- and service engineering methodologies, but combines those with requirements for and contributions of semantic services. An overview of the methodology is shown in Figure 4, using the BPMN notation, and highlighting how the different components are used in the stages of the process. In the following, we will describe the different steps in more detail.

A. Ontology Engineering

The first step in creating semantic services is to model the ontology that will be used for describing the service's inputs,

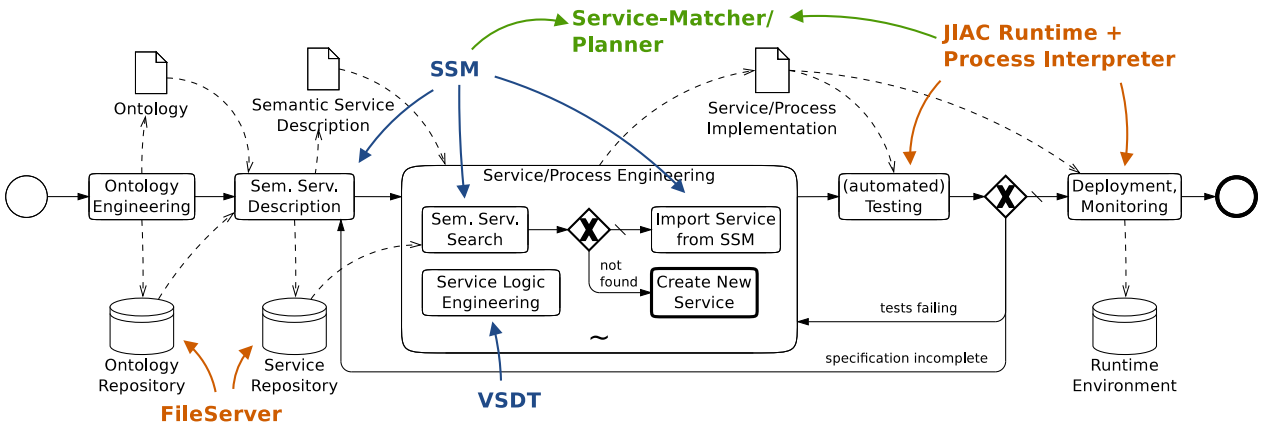

Figure 4. Semantic Service Management and Development Process, as a BPMN process, and associated components: Green: Semantic Service Core; Blue: Development Tools; Red: Execution Environment.

outputs, precondition and effect, if any. This is particularly important, since one of the main motivations for semantic services is for those services to be easily findable, reusable, and composable with other services; thus, whenever possible it should be the aim to reuse, or, if necessary, extend existing ontologies, instead of creating new ones. This step is also concerned with mapping the ontological concepts, for example described in OWL, to a representation that is closer to the service implementation, e.g., Java classes (or vice versa, starting with Java classes and generating according OWL ontologies).

The new or modified ontologies are then uploaded to a server hosting a repository of known ontologies, so they can be used in the next step, as well as in other services. There is no specific tool for this step in our method. Ontologies can be created, e.g., with Protégé,[21] or generated from existing Java classes or EMF models [22].

B. Creating Semantic Service Description

Next is the creation of the semantic service description itself, defining the "contract" of the service. Of course, this step is not particular for semantic services, but is a common practice for all of service- and software engineering. The major difference is that besides name, textual description, input and output parameter, also the preconditions and effects of a service can be defined. Especially the latter, which in our approach can be described with the semantic rule language SWRL, extend the attributes of a service in a way that matching or planning processes can deduce its purpose and its formal prerequisites. However, as describing semantic terms can be challenging, we paid attention to provide a user-friendly editor with syntax-highlighting, auto-completion and validation parser. Currently missing, but contemplated is the integration of several QoS attributes, making the selection of services also sensitive to non-functional aspects.

The new service description is uploaded to a service repository, adding it to the list of services usable by the semantic service matcher and planner. In our method, the SSM tool is used for creating the service descriptions using OWL-S. Existing ontologies can be browsed (but not edited) for selecting concepts for input and output, while preconditions and effects are specified using SWRL. The finalized service description can then be deployed to the repository and an accordant stub for the service implementation can be generated.

C. Service- and Process Engineering

The bulk of the service development process is occupied with engineering the service's implementation. While the service's method declaration can be generated from the semantic service description, its body has to be implemented by a developer. Here, we can differentiate two main activities: Identifying and integrating existing services, and developing the logic that combines those services to a new service, or process, with added value.

There are three ways how services can be searched, identified, and imported into the currently developed process, using the SSM tool:

- The service can be searched for, using a semantic service template, and the service best matching the template is integrated into the current service.
- In case no single service satisfies the template, the semantic service planner can be used to automatically find a service composition that, as a whole, matches the template; the individual services of that composition are then integrated into the current service in the appropriate sequence.
- Instead of searching services at design time, the template that would be used for matching the service can itself be integrated into the current service, deferring the search and matching process to runtime.

Of course, there is also a fourth case: That no service or service composition can be found that fulfils the template. In this case, a new service has to be created, thus starting the service development process again.

The service logic can be created in two ways: Either in the form of a Java method, or, using the VSDT, as a BPMN process, which is later either mapped to Java (JIAC agent beans) or interpreted directly. Which one to choose mainly depends on the ratio of service reuse to "original" service logic: In case the new service is mainly a composition of existing basic services, they can very well be modelled visually as business processes, but if they contain complex calculations or make extensive use of third-party libraries (that are not

available as services), then implementing the services in plain Java is the better choice.

D. Testing the Implementation against the Specification

The last step before deployment is testing, to ensure that the services' implementations comply with their semantic descriptions. Of course, testing plays a well-established role in software engineering and is not particular to semantic service development. However, the presence of formal semantic descriptions impose both an obligation and an opportunity for (automated) unit testing.

On the one hand, while even a regular function or service that does not comply with its documentation is always a nuisance, a semantic service that violates its stated effect could threaten the functionality of the entire system it is embedded in, as automated planners will rely on that information. On the other hand, since the intended behaviour of the service has already been specified in its precondition and effect, writing the actual tests becomes very straightforward.

While this is currently not implemented in our approach, it would also be possible to automatically generate unit tests from the semantic service description, particularly the service's preconditions and according effects. For this, the input parameters can be generated, setting all attributes that are not specified in the precondition randomly; then, the expected output can be inferred from the service's effect, thus testing the actual result of the service invocation against the expected value.

In case the service does not comply with the tests (i.e., with its stated preconditions and effects), the usual course of action is, of course, to fix the service. However, in some cases this may also expose flaws in the service's input, output, precondition and effect (IOPE) descriptions. In this case, the process has to backtrack and update the semantic service description and adapt or extend the service's implementation accordingly.

E. Deployment and Runtime Monitoring

The final step is to deploy the new service to the runtime environment. Depending on whether the service has been implemented directly as a Java class (e.g., a JIAC agent bean exposing an accordant action), or in the form of a BPMN process diagram orchestrating different existing services, the deployment process is slightly different.

- In case the service has been implemented directly in Java and is meant to be a basic service to be used as a building block for other services, it is best to create a new agent exclusively for that service and to deploy it to the runtime server.
- In case of a service composition created as a BPMN process, the process diagram can be deployed to an already running process interpreter agent. This way, deployment and undeployment is very dynamic, and the interpreter also provides basic capabilities for runtime monitoring and user interaction. Alternatively, the process can also be automatically translated to Java code and deployed as in the above mentioned case.

In both cases, the services are deployed to the JIAC runtime environment and can be invoked as actions, and searched for using the semantic service matcher. Using the WSDL and REST integration beans, the services will also be exposed as WSDL or REST services, respectively, and can transparently use other services available in those formats.

IV. The EMD Use Case

In the project EMD (Extendable and adaptive E-Mobility Services), a use case within the transportation domain was constructed to demonstrate the use of the developed tools, the methodology, and the basic services, as depicted in Figure 5. Starting from the fundamental services, like reading the scheduled meetings from a calendar, getting the state of charge of a vehicle, scheduling the charging, or searching charging stations, this example constructs an adaptive extended service with the goal of supporting home visiting nurses during their daily rounds. Here, the idea is that if the state of charge of an electric car does not suffice for the whole schedule of visits for the day, the application will provide an alternative means of transportation for just enough time to charge the vehicle. This will lead the nurse to a few appointments via an intermodal route (including public transport, car or bike sharing vehicles, walking, taxis or any mobility service available). Finally, she returns to her vehicle when the vehicle's charging process has finished.

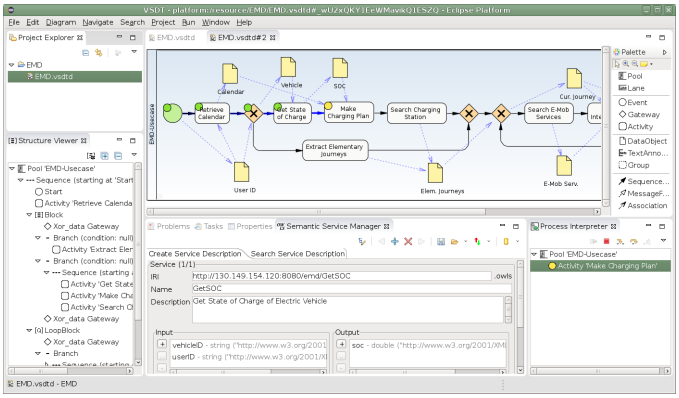

Figure 5. Example usecase for an extendible, adaptive mobility service. Top: VSDT editor showing process diagram; bottom: SSM view.

The search of the mobility providing services is the adaptive part of the service: Depending on the available services and the context of the user (e.g., carrying heavy equipment) the appropriate services are given to an intermodal routing service. Integrating the $SeMa^2$ service matcher into the service (shown as the "Search E-Mob Service") component, allows us to dynamically change the services which are used for the intermodal routing.

Describing all services semantically using the SSM allows the $SeMa^2$ to find fitting services at runtime. The semantic description of the services is done with the entities of the domain ontology created during the project. Since the whole process should not be composed automatically but using the VSDT editor to describe the interconnection of the service, the $SeMa^2$ is also part of the design process, finding existing service to be integrated into the developed process.

The service planner can then be used to create a first service composition which can be adapted using the VSDT. During the development the process can be debugged during its execution via the built-in BPMN process interpreter of the VSDT. This allows the inspection of the exchanged messages

and the developer has the ability to add or remove specific services or to integrate transformation of the exchanged data formats.

V. Related Work

The foundation for an effective management of semantic services lies in a well-elaborated formalism for the semantic description of services. In this respect, a lot of research has been done and has led to a variety of approaches, some of them being lightweight, others coming up with a complete framework solution. The authors in [23] give a detailed overview about existing solutions, such as OWL-S, Semantic Annotations for WSDL and XML Schema (SAWSDL) or the Web Service Modeling Ontology (WSMO). Since the composition of services, which is an essential part of our framework, highly relies on formal expressions for preconditions and effects, we analysed OWL-S as the most suitable one embedding the rule language SWRL.

A comprehensive overview about approaches building upon these formalisms, such as development tools, matching and planning algorithms and the execution environment, is not feasible for this publication, therefore we refer to our related publications for this. Within the remaining section we shortly focus on other projects that present concepts for semantic service management.

The project Mercury [24] focused on automatic service discovery based on the user context. For the semantic description of the services different standards such as OWL-S and SAWSDL are supported. The project provides tool support for the user requests and enables the developer to insert the found services into a process chain. In contrast to our approach, Mercury can consider different semantic service formats, however, the user request does not allow to describe formal preconditions or effects, but is solely focusing on key words. Klan et al. [25] propose relevant requirements to an efficient semantic service management with respect to service matching. Furthermore, they define an evaluation methodology rating the results provided by semantic matchers according to user requirements. Within the project DIANE [26] a new service description model has been proposed that especially focuses on the expression of state transitions after service invocations. Furthermore, it provides the instance-based description of service requests, which makes the reasoning process more realistic. In contrast to that, Karastoyanova et al. [27] focus on the problem of semantic service management from a BPMN process perspective, providing an architecture for Semantic Business Process Management. They cover the whole lifecycle using the WSMO technology for the semantic part. However, what is missing in this work is an approach for automated service composition at runtime. The Framework PORSCE II [28] uses AI planning techniques like LPG-td [29] which implements a graph plan algorithm, to automatically create service composition, with the drawback of transforming the composition problem to a standard PDDL planning problem and thus losing the expressiveness of OWL and SWRL.

VI. Conclusion

In this paper, we presented a semantic service management methodology that covers the phases of service description modelling and deployment as well as service discovery and composition, both at design-time and at runtime, in order to generate adaptive and flexible systems in service-oriented environments. Most of the phases are supported by tools reducing the development effort. Furthermore, we presented an inclusion mechanism into a BPMN process environment called VSDT, where service templates can be specified within the process structure and dynamically matched to concrete services at runtime. Our whole approach has been included into a real environment, where services from the electric mobility domain have been linked to complex processes.

In the future, we will address the challenges for service composition that were formulated in Section II-A4. Among others, we intend to complete the integration of the specification for the description language SWRL and to facilitate the semantic annotation of services by further elaborating our tools. In doing so, we see a good opportunity to foster the usage of service planning approaches in real world applications.

Acknowledgements

This work is funded by the German Federal Ministry of Economic Affairs and Energy under the funding reference number 16SBB007A.

References

[1] M. Klusch, U. Küster, A. Leger, D. Martin, and M. Paolucci, "5^{th} International Semantic Service Selection Contest - Performance Evaluation of Semantic Service Matchmakers," Nov. 2012, last access: 2016/03/07. [Online]. Available: http://www-ags.dfki.uni-sb.de/~klusch/s3/s3c-2012-summary-report.pdf

[2] J. Fähndrich, N. Masuch, H. Yildirim, and S. Albayrak, "Towards Automated Service Matchmaking and Planning for Multi-Agent Systems with OWL-S – Approach and Challenges," in Service-Oriented Computing – ICSOC 2013 Workshops. Cham: Springer International Publishing, 2014, pp. 240–247.

[3] P. A. Morris, "Combining expert judgments: A Bayesian approach," Management Science, vol. 23, no. 7, 1977, pp. 679–693.

[4] M. Stone, "The opinion pool," The Annals of Mathematical Statistics, vol. 32, no. 4, 1961, pp. 1339–1342.

[5] C. Genest, "Pooling operators with the marginalization property," The Canadian Journal of Statistics/La Revue Canadienne de Statistique, vol. 12, no. 2, 1984, pp. 153–163.

[6] D. Dyer. Watchmaker Framework. Last access: 2016/05/11. [Online]. Available: http://watchmaker.uncommons.org/ (2006)

[7] W. L. Goffe, G. D. Ferrier, and J. Rogers, "Global optimization of statistical functions with simulated annealing," Journal of Econometrics, vol. 60, no. 1-2, Jan. 1994, pp. 65–99.

[8] M. Klusch and P. Kapahnke. The Semantic Web Service Matchmaker Evaluation Environment (SME2). Last access: 2016/05/11. [Online]. Available: http://projects.semwebcentral.org/projects/sme2/ (2008)

[9] M. Ghallab, D. S. Nau, and P. Traverso, Automated Planning: Theory & Practice, D. E. M. Penrose, Ed. Morgan Kaufmann, 2008.

[10] H. Saboohi and S. A. Kareem, "A resemblance study of test collections for world-altering semantic web services," in Int. MultiConf. of Engineers and Computer Scientists (IMECS), vol. I, 2011, pp. 716–720.

[11] P. Doherty, W. Lukaszewicz, and A. Szalas, "Efficient Reasoning Using the Local Closed-World Assumption," in Agents and Computational Autonomy. Springer Berlin Heidelberg, Jan. 2003, pp. 49–58.

[12] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical owl-dl reasoner," Web Semant., vol. 5, no. 2, Jun. 2007, pp. 51–53. [Online]. Available: http://dx.doi.org/10.1016/j.websem.2007.03.004

[13] E. Foundation. Eclipse. Last access: 2016/05/11. [Online]. Available: http://www.eclipse.org/ (2016)

[14] D. Martin et al., "OWL-S: Semantic Markup for Web Services," Website, Tech. Rep., Nov. 2004. [Online]. Available: http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[15] N. Masuch, C. Kuster, and S. Albayrak, “Semantic service manager–enabling semantic web technologies in multi-agent systems,” in Proceedings of the Joint Workshops on Semantic Web and Big Data Technologies, INFORMATIK 2014, Stuttgart, Germany, 2014, 2014, pp. 499–510.

[16] M. Lützenberger, T. Konnerth, and T. Küster, “Programming of multi-agent applications with JIAC,” in Industrial Agents – Emerging Applications of Software Agents in Industry, P. Leitão and S. Karnouskos, Eds. Elsevier, 2015, pp. 381–400.

[17] T. Küster and A. Heßler, “Towards transformations from BPMN to heterogeneous systems,” in Business Process Management Workshops, ser. LNBIP, D. Ardagna, M. Mecella, and J. Yang, Eds. Springer Berlin Heidelberg, 2009, vol. 17, pp. 200–211.

[18] OMG, “Business process model and notation (BPMN) version 2.0,” Object Management Group, Specification formal/2011-01-03, 2011.

[19] T. Küster, M. Lützenberger, and S. Albayrak, “A formal description of a mapping from business processes to agents,” in Engineering Multi-Agent Systems, ser. LNAI, M. Baldoni, L. Baresi, and M. Dastani, Eds. Springer International Publishing, 2015, vol. 9318, pp. 153–170.

[20] T. Küster, A. Heßler, and S. Albayrak, “Towards process-oriented modelling and creation of multi-agent systems,” in Engineering Multi-Agent Systems, ser. LNAI, F. Dalpiaz, J. Dix, and M. B. van Riemsdijk, Eds. Springer International Publishing, 2014, vol. 8758, pp. 163–180.

[21] Stanford. Protégé. Last access: 2016/05/11. [Online]. Available: http://protege.stanford.edu/ (2016)

[22] E. Foundation. Eclipse Modeling Framework (EMF). Last access: 2016/05/11. [Online]. Available: https://eclipse.org/modeling/emf/ (2016)

[23] A. Barros and D. Oberle, Handbook of Service Description: USDL and Its Methods. Springer Publishing Company, Incorporated, 2012.

[24] K. Opasjumruskit, J. Expósito, B. König-Ries, A. Nauerz, and M. Welsch, “Service discovery with personal awareness in smart environments,” Creating Personal, Social, and Urban Awareness through Pervasive Computing, 2013, pp. 86–107.

[25] F. Klan and B. König-Ries, “A user-centered methodology for the evaluation of (semantic) web service discovery and selection,” in Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14). ACM, 2014, p. 18.

[26] U. Küster, B. König-Ries, M. Klein, and M. Stern, “Diane: A matchmaking-centered framework for automated service discovery, composition, binding, and invocation on the web,” International Journal of Electronic Commerce, vol. 12, no. 2, 2007, pp. 41–68.

[27] D. Karastoyanova et al., “A reference architecture for semantic business process management systems,” in Multikonferenz Wirtschaftsinformatik, 2008, pp. 1727–1738.

[28] O. Hatzi, D. Vrakas, and N. Bassiliades, “The PORSCE II framework: Using AI planning for automated semantic web service composition,” Knowledge Engineering Review, vol. 28, no. 02, 2013, pp. 137–156.

[29] A. Gerevini, A. Saetti, I. Serina, and P. Toninelli, “LPG-TD: a fully automated planner for PDDL2.2 domains,” in Proc. of 14th Int. Conf. on Automated Planning and Scheduling (ICAPS-04) International Planning Competition abstracts, 2004.