Semantic Service Management and Orchestration for Adaptive and Evolving Processes

Johannes Fähndrich, Tobias Küster, and Nils Masuch

DAI-Labor, Technische Universität Berlin
Ernst-Reuter-Platz 7, 10587 Berlin, Germany
e-mail: {johannes.faehndrich, tobias.kuester, nils.masuch}@dai-labor.de

***Abstract*—Introducing adaptiveness into service compositions allows for a next generation of services, which adapt to their context of use and posses self-* properties like self-healing and self-configuring. However, the development of adaptive and flexible services is challenging and lacks tool support. For this next step in service development there are a multitude of requirements to be met: a service discovery needs to keep the available services up-to-date, a semantic layer needs special development resources to introduce interoperability, ontologies need to be managed and merged, a service selection mechanism has to find the fitting service for a context out of a vast amount of service advertisements, and runtime components need to surveil the execution of such a service composition. In this paper, we review multiple projects, in which those components have been subject to research, and we present our own approach of a semi-automatic development methodology for adaptive service compositions and finally discuss future challenges.**

Keywords–Semantic Service Matching; Automated Service Composition; BPMN Processes; OWL-S.

I. INTRODUCTION

In recent years, the increasing digitalization of our societies has led to a vast amount of new possibilities. 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, new services are provided digitally, such as car-sharing offers, where the user can find, reserve and unlock a nearby car via an API, and many of the actual cars or charging stations are accessible via services, as well. Those are just two examples of new services, leading towards a sophisticated *Internet of Things* (IoT), which is often eagerly anticipated to connect services across domain borders.

By introducing adaptiveness into service compositions, those services can be dynamically combined and orchestrated, allowing for a next generation of services, which adapt to their context of use and posses self-* properties like self-healing and self-configuring. However, there are some significant challenges that have to be overcome in order to exploit their potential. To begin with, there is the requirement of finding an appropriate service in the first place. 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 understanding 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.

In this article, which is an updated and extended version of a paper previously presented at The Eleventh International Conference on Internet and Web Applications and Services (ICIW 2016) [1], it is 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 an editor for the Business Process Model and Notation (BPMN). Based upon that we developed comprehensive matching and service composition techniques that can be used both at design-time and at runtime.

The remainder of this paper is structured as follows: At first, we motivate our case in Section II. In the following sections, we introduce the core components used for matching and planning (Section III), as well as development tools (Section IV) and a runtime environment (Section V), in which those are used. In each of those sections, we will also contrast our approach and contribution to the respective state-of-the-art. Then, in Section VI, we combine the core components, tools, and runtime to a comprehensive development method for semantic service engineering, and show how it was applied in research projects (Section VII), before we finally wrap up and conclude in Section VIII.

II. MOTIVATION

Proper service management is highly important to build reliable and reusable software systems. Service management can be divided into several phases, whereas each of them contains requirements that are not fully met so far. As illustrated in Figure 1, the service management starts with the most fundamental part, the service engineering. Under the term service engineering we subsume the concrete specification of a service, which also contains the embedment of existing services into a new process. At this point, the challenge of finding such a service comes into play for the first time. When we think about platforms containing hundreds or thousands of

services, a comprehensive search, matching, and even planning mechanism is necessary. In our case, we try to address this challenge by providing a semantic-based service matching mechanism, which will be described in Section III. Furthermore, this feature has to be embedded into a development environment. Many approaches that provide service matching and planning stick to very specific planning languages for the whole process definition, which makes it hard to use them for daily problems. Therefore, in this paper it is a clear focus to provide an approach that relies on a specification language that is commonly used and expressive enough. After having specified the process of the service its proper declaration has to be processed. Currently, this step is neglected since most of the service management approaches do not rely on a service declaration that can be analysed and interpreted by machines. In future scenarios however, each service, even a value-added one that is already using other services, should be made available to other instances in order to enable efficient service composition. In Section IV we will provide an approach how to semantically enhance services based on a semi-automatic process. After the declaration the service has to be tested and where required the service engineering has to be launched again in order to adapt the service process. There are already lots of interpreter components that enable the step-through and validation of a process. However, an interpreter component that is integrating service matching, planning and service selection features during testing is – to the best of our knowledge – not available so far. We will present our approach for this in Section IV, as well. After a successful testing phase the developer has to deploy the service in order to make it available. Here the important issue is to transform the specified process from the Service Engineering phase into an executable language. The more transformations the tool chain supports the more flexible the service can be launched. In our case we provide different target languages as well as direct interpretation, which we describe in Section V.

Furthermore, deployed processes currently are not very flexible in practice. In our approach we aim to support the dynamic integration of services or even service chains at runtime. This leads to a highly adaptive service that can select services according to functionality and Quality-of-Service parameters. The approach of adaptive service selection at runtime is also discussed in Section III.

The described adaptability is also important for the last phase of the service management lifecycle: the service monitoring. While executing the service, another component has to surveil the status of the service. Think about a situation when a street network routing service that is being used for a multimodal routing service is immediately out of order and not usable any more. Usually, some error message might occur, that will be sent to a maintenance person that has to check why the service can not be invoked any more and has to decide how to proceed next. It is our goal to simplify this by using the before described adaptive service approach. In our concept the service composition itself is searching for an alternative service within the platform based on the semantic description of the missing service.

In summary it can be stated that in each of the service management phases there are still open challenges to fulfil in order to provide an adaptable service management methodology, that can be used for actual problems. In the following we will describe our concept of adaptive service selection and planning.

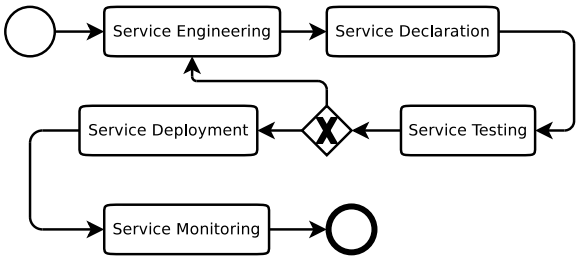

Figure 1. Basic service management lifecycle.

III. Core Components

In this section, we will have a look at the core components used for matching semantic services and for planning with those services. While those components are used in the development tools and in the runtime environment that we will introduce in the next sections, they are self-contained and can be used independently from the rest, e.g., in a different context.

A. Semantic Service Matcher

Since the beginning of research in semantic service matching, matchmakers have matured in precision and recall [2]. 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 *Semantic Service Selection* contest (S3) in 2012 [2], 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. In the following we first describe the approach of $SeMa^2$ and then discuss the current state-of-the art for service matchmaking.

1) Approach: Within this section we describe 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 [3].

a) 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.

As shown in Figure 2, the $SeMa^2$ consists of 33 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 six 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; the rule reasoning expert, which evaluates whether the precondition and effect rules are satisfied with the same parameters; as well as the rule structure expert, and the marker passing expert, which connects the describing ontologies through ontology matching.

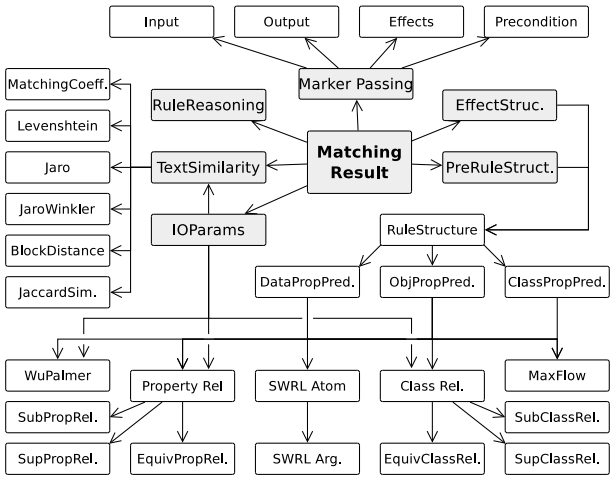

Figure 2. Expert System of the $SeMa^2$. High-level experts are composed of low-level experts, all contributing to the Matching Result.

Each expert creates an opinion about the matching of two services. These opinions are aggregated by experts which use other experts to create their opinion. Six types of experts form the top level of the aggregation. Five of those experts analyse structural and logical relatedness of the two services. The sixth expert is a semantic expert. It analyses the semantic similarity between the different aspects of the two services. For its opinion it created a semantic graph through lexical decomposition and passes markers along the edges of this graph to measure the semantic distance from, e.g., the input of one service to another. The interested reader is referred to [4] for more details. 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. The opinions of the experts are weighted due to their performance in an offline learning phase. For more details please find [3].

b) Probabilistic model of opinion: The different opinions of the experts are formalised by utilising the results of Morris [5], 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 [3].

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 [6] in a previous work [3]. This arithmetic mean has been generalised by Genest [7] 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. This selection of weights for the experts enable the matcher to adapt to the specifics of the semantic service descriptions of a special domain.

c) Learning Semantic Service Matcher: Selecting weights for each experts instances ($SeMa^2$ for now has 133 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 [8] to simulated annealing [9]. For the statistical evaluation the Semantic Web Service Matchmaker Evaluation Environment (SME2) tool [10] 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.

2) State-of-the-Art: In order to optimise the result of the service matching a learning phase can be introduced to adjust the parameters of a service matchmaker to the properties of the domain. The parameters to learn depend on the service matchmaker and thus its flexibility depends on the parameters that can be observed.

In *Klusch et al.* [11] the authors introduced a formal model of defining weights for the aggregation of different similarity measures with the names w_w – similarity and w_s – structural similarity measure. The aggregation method has been learned using a Support Vector Machine (SVM) approach based on training data. The matchmaker component that invokes this approach is designed to match SA-WSDL services (Semantic Annotations for Web Service Description Language).

Klusch and *Kapahnke* [12] introduce another learning service matchmaker by extending the approach of a prior work [13] for OWL-S service descriptions (Web Ontology Language for Web-services). Here, matching results of different matching types are aggregated using a weighted mean. The authors introduce different types of matching results that are weighted. Firstly, approximated logical matching, which is divided into approximated logical plug-in and subsumed-by matching. Secondly, non-logic-based approximated matching, which are text and structural semantic similarity-based signature matching. The weights of this aggregation are also learned using a SVM. This supervised learning approach is replicated in our work, but with a different learning algorithm. The relevance set that is used to rank the matching results are reused with a genetic algorithm and a hill-climbing search.

Gmati et al. [14] use an architecture similar to ours, which uses parameters as weights to combine the results of the different matching components. This idea was published earlier in [3] with an additional learning component for the introduced weights.

To the best of our knowledge there exist only these approaches that utilise machine-learning techniques in order to cope with the challenge of aggregating service matchmaking techniques. For future research, service matchers will also have to be extended with the capability of comparing QoS parameters and Service Level Agreements of services.

B. Semantic Services Planner

During the creation of a service composition, automation might help the developer to save time. The capability of general purpose planners to create solutions for complex problems is sometimes seen with conflicting opinions: The creation of a plan is resource intense and can produce use case specific plans, which are rarely reusable. On the other hand, they can find solution humans are not capable of finding. This is due the the fact that a human is insufficiently effective when scanning through the vast amount of available services and comprehend their contextual usage.

This leads to the question when to use a general purpose planner during the creation of service compositions. The answer is flexible as it is unspecific: The planner can be used to the extend the developer needs its help. If the developer wants to prove feasibility before creating a hand-made service composition, then the whole service composition can be planned to get an idea of the available services. If the overall service composition is already designed, the inclusion of new services might entail other services, which can be found through planning. This can help a developer to chose between the fit of a service into its compositions. At the end the suggestion of single services in a context of a service composition (between a service layer $i-1$ and another $i+1$) can be based on QoS parameters as well.

This allows a specification of the adaptive parts of a service composition, since the parts that are provided by a planner can be changed during runtime. By removing the grounding of a service, the so-created service template needs to be filled with a service instance available at runtime, which enforces adaptation. In this way, a compromise between development time and adaptiveness can be found.

1) Approach: The ability to automatically compose services to reach a given goal is called service planning [3]. 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.

The algorithm in Figure 3 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.

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

```
1:  path ← [ ]
2:  Closed ← ∅
3:  Open ← {S_start}
4:  while s ← StateSearch.next(Open) do
5:      if s ∉ Closed then
6:          if s = S_goal then
7:              return reconstructPath(path + [s])
8:          end if
9:          grounded ← ServiceSearch.UsefulServices(s)
10:         if grounded ≠ ∅ then
11:             succ ← {execute(s, g) | g ∈ grounded}
12:             Open ← Open ∪ succ \ Closed
13:             path ← path + [s]
14:         end if
15:         Closed ← Closed ∪ {s}
16:     end if
17: end while
18: return failure
```

Figure 3. Service Planner algorithm.

The complexity of the algorithm depends on the implementation of the state search and state pruning mechanism, being the heuristic that selects useful services, including the complexity of the service matcher used. In general, the worst case complexity of such an algorithm is exponential [15, 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 [16]. Thus, here we create an assertion of the class of the output, creating an appropriate individual, equivalent to the "AgentKnows" of Doherty et al. [17].
- *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 [18].
- *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 that can not be modelled without SWRLx built-ins.

2) State-of-the-Art: There is a multitude of related work in using planning techniques for service composition. Rodriguez et al. [19] analyse three parts of Service-Oriented Architecture (SOA) in connection to artificial intelligence planning: i) service discovery techniques, ii) service composition systems,

and iii) service development tools. Even though Rodriguez et al. noticed that the use of QoS parameters for the selection of a service in a service composition is "a significant research problem" [19] their analysis of the state of the art does not reflect QoS parameters. Markou and Refanidis focus on non-deterministic planning approaches for automated service composition where the approaches are analysed for their heuristics but only in the sense of using them or not [20]. Zou et al. [21], [22], [23] focus more on efficiency of the planning techniques for the automated service composition but neglect to see the importance of semantic information given by semantic web service descriptions. This leads them to look at approaches translating the service composition problem into a PDDL planning domain and with that they lose every semantic description available before. Transforming facts described in OWL (Web Ontology Language) and, e.g., SWRL into a PDDL domain leaves with no basis for ontology matching if the same fact is formulated in different ways.

We argue that the performance of a general purpose planner depends on the heuristic used during the search for a path from start to goal state. This leads us to the conclusion to analyse the state of the art for factors neglected by other surveys, e.g., how a service description is used to create heuristics for the used planning technique.

We start out with Rodriguez-Mier et al. [24] who neglect non-functional parameters as well, but they describe a general heuristic that is used in an A^* algorithm. This heuristic combines the amount of already executed services in a backward search: $h(n) = distance(S_n, S_{goal})$, where S_n is the current state and S_{goal} is the goal state. In addition the cost of a state S_n is the number of services still needed to reach the start state. This is denoted with $c(n)$. This is combined to the heuristic function $f(n) = h(n) + c(n)$. This heuristic has the drawback that it only is applicable after a solution has been found. Thus it can be used to select the best path, from start to goal state, but it can not be used during the planning itself.

Meyer and Weske [25] as well count the number of service executions as a heuristic for their planning mechanism. They restrict this heuristic further, arguing that it is only an upper limit since the plan could include parallel executions and thus the amount of service execution steps can be further reduced.

Hoffman and Nebel [26] use a relaxed plan heuristic by removing the deletions out of the effect of a service. This can be done in PDDL since there are only additions and deletions of facts. Such a heuristic becomes research worthy if the semantics of the problem looked at comply to the open world assumption. This is because we can not decide if a left-out fact is true or not. Further effects could conflict each other because they are coming from different conceptualisations, e.g., different ontologies describing the same or opposite fact.

Klusch et al. [27] use the heuristic of Hoffman and Nebel [26] where a breadth first search is used if services have the same heuristic value.

Bonet and Geffner [28] build a heuristic by evaluating each fact of the goal. Here a fact f_g of the goal has an estimated cost of the length of a minimal path through the planning state space from the initial state. This leads to a heuristic:

$$h(s) = \sum_{f_g \in S_{goal}} g_s(f_g) \quad (1)$$

where

$$g_s(f_g) = \begin{cases} 0 & \text{if } f_g \in S \\ \max\limits_{s \in Services} [1 + g_s(s.pre)] & \text{else} \end{cases} \quad (2)$$

Here $Services$ is the set of all available services and $s.pre$ is the set of preconditions of the service s. The maximum in Equation (2) is chosen because it forms an admissible heuristic [28].

Mediratta and Srivastrava [29] introduce the heuristic of Hoffman and Nebel [26] to an AND-OR graph as a plan. Here the cost of each OR path through the graph is selected with a minimum. For AND-connected nodes in the Graph the cost function is summed up.

Fanjiang and Syu [30] use genetic algorithms for the service composition, which does not contain any heuristics at all. This is the same with Lécué et al. [31] who use reasoning in OWL-DL (Web Ontology Language Description Logic) for the selection of fitting services but no heuristics.

Regarding this state of the art we suggest that heuristics are part of the future work in this domain [32]. This is based on the performance issues described in the evaluation of the here analysed papers. Furthermore, the heuristic can be used during design time, to suggest services to the developer as part of a semi-automatic service composition framework. During design time, the actual execution and parametrisation of the service is left to the developer. Thus, a heuristic does not need to select purely executable services for a given state.

IV. Development Components

The method for semantic service management and development makes use of two development tools, which are both implemented as Eclipse plugins [33] and thus can seamlessly be integrated into the developer's usual workflow.

The overall architecture of the proposed development components is shown in Figure 4. We differentiate between the **runtime**, in which a service can be grounded to its real parameters and execution environment, and the **design time**, in which there are fewer resource constrains for the planning and an expert is able to fine-tune the service composition in a BPMN editor.

Here, black arrows describe the information flow, e.g., the service matcher gets the service descriptions out of a service repository and supplies the state-space-planner with fitting services for a current state.

a) Design Time: At design time, the developer is supported by a **BPMN editor** to create service compositions. This BPMN editor uses **semantic annotations** to describe the functional and non-functional aspects of a service. These descriptions are used by a planning component to build service compositions (a sequence of services) to suggest to the user while he is searching for reusable service for his composition. This **state space planner** uses a **service matcher** to identify useful services out of a service repository.

The result of this process is a BPMN description of a service composition that can be deployed into a runtime. This description can be quite abstract as we will describe in Section IV-B. This abstraction allows for flexibility and thus for an adaptation of the composition at runtime.

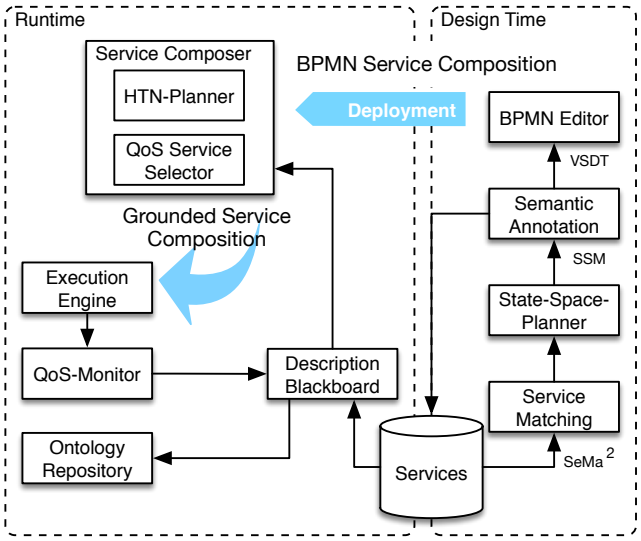

Figure 4. Architecture of the development components.

b) Runtime: At runtime, the given service composition is concretised. This is done since the resource consumption of the general purpose planning is too high to be used at runtime. To keep the introduced flexibility, the **service composer** uses a **HTN-planner** (Hierarchical Task Network) to select between alternative sub-plans. This is thought as a first principle planner where plans are selected from a plan library [34]. As a one-step plan, a service call is the atomic entity that can be replaced. This service composer can replace unavailable services, or use a **QoS service selector** to optimise the service composition to some criteria. With ever more service compositions available and thus more alternatives of sub-plans to search from, this service composer becomes a fast planner with domain specific, optimised service composition.

Another task of the service composer is the selection of unknown parameters. Those are called the service grounding. This might be, e.g., the resolution of a display device or the rendered models, which are unknown during design time. Furthermore, if a template of a service is part of the composition, the concrete service instance needs to be chosen, before the composition can be executed. Again this selection can be based on the QoS parameters of the services.

The resulting grounded service composition is passed to an **execution engine**. This execution engine reports QoS parameters back to the QoS-Monitoring, which in parts then ranks the services used to learn their quality parameters for future references.

The **service blackboard** describes the available services and their QoS behaviour. The service blackboard thus does restrict the accessibility of services theoretically available during design time and practically executable during runtime – there might be political, organisational, or financial reasons as to by whom services can be accessed – and lets the service composer choose from alternatives.

All in all, this separation of runtime and design-time is a trade-off between complexity and adaptability. Since the automated creation of a service composition from scratch is too expensive, the adaptation of existing plans might lose on adaptiveness, but renders the system profitable through reusability of plans.

In the following we describe the two involved development components, namely the Semantic Service and Ontology Manager (SSM) and the Visual Service Design Tool (VSDT) in more detail.

A. *Semantic Service and Ontology Manager*

The description of the Semantic Service and Ontology Manager is divided into an approach section and a short state-of-the art section related to the semantic annotation of services.

1) Approach: 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 OWL-S [35] are not easy to describe from scratch, we developed a plug-in called Semantic Service Manager (SSM) [36], providing a set of features supporting a semi-automatic description of services. The core of SSM is an Ontology Manager (see Figure 5), which enables the developer to include and utilize 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.

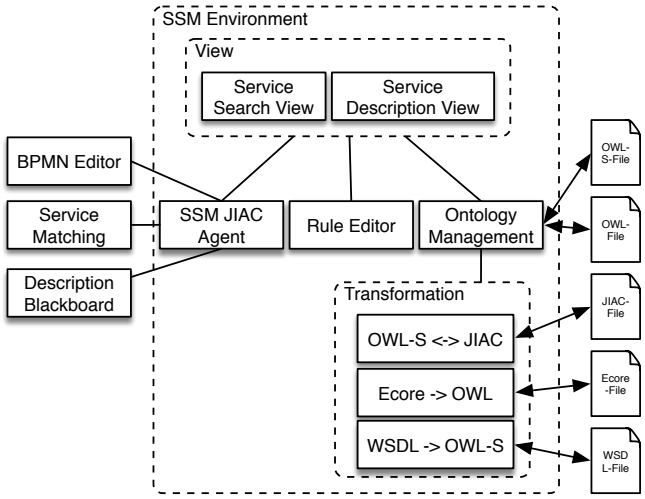

Figure 5. Components of the Semantic Service Manager.

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 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 V), it can be sent to a BPMN process (see the next paragraph), or it can be linked to a service of the multi-agent framework JIAC V (Java-based Intelligent Agent Componentware, version 5) [37]. 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 (see Figure 7), or a code inclusion function can be triggered that inserts the service call code into the open Java window.

The different ways how the SSM can be used for describing or searching services or service templates and for importing them into a complex process are shown in Figure 6.

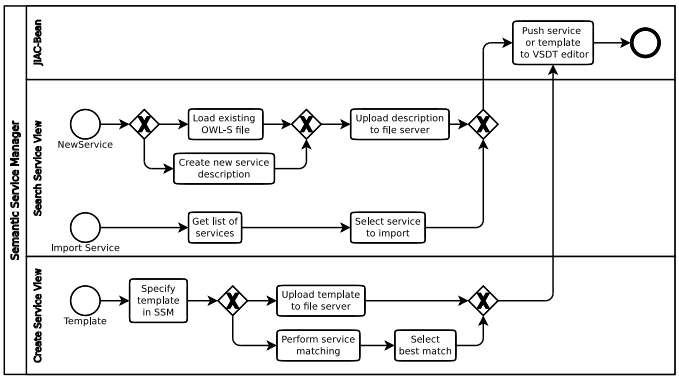

Figure 6. Different ways of creating or searching services with the SSM and using them in complex processes.

2) State-of-the-Art: Many of the works in the context of semantic service management merely focus on service matchmaking, but forget the design process, although being just as important. However, some focus on the semantic markup. The *OWL-S editor* [38] 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 graphical user interface. 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* [39]. It is a plug-in for Eclipse and, 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 does not support preconditions nor effects.

B. Visual Service Design Tool

This section starts with a detailed description of the Visual Service Design Tool followed by a state-of-the-art paragraph related to process modelling.

1) Approach: While basic services are usually implemented in the form of Java classes or equivalent, for service compositions the Business Process Model and Notation (BPMN) [40] has proven useful. Using the VSDT, existing semantic services can be imported from the SSM and orchestrated to complex processes using the BPMN notation (Figure 7).

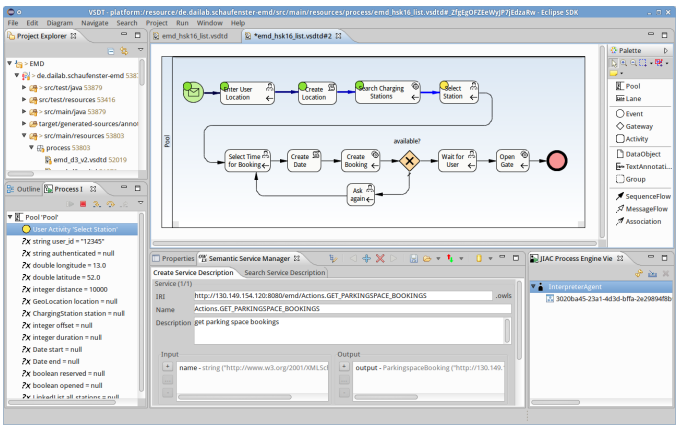

Figure 7. Semantic service development tools with example usecase. Top: VSDT editor showing process diagram; bottom: SSM view.

The VSDT is based on the Eclipse Graphical Modelling Framework (GMF) and provides a rich visual editor for BPMN processes [41]. It also provides means for process validation, simulation/debugging, and export features to different executable languages.

The BPMN editor integrates with the Semantic Service Manager view in the way that services from the SSM can be imported into the VSDT. Via a function in the User Interface (UI), an accordant 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 control flow, short scripts, and other services to a complex process. Instead of an actual service, the same approach can also be used for importing a service template into the VSDT process, which will then be matched to an actual service at runtime.

Accordingly, the BPMN service model used in the VSDT had to be extended to allow for semantic information. While the BPMN specification only accounts for Web service implementations – both for *service-* and for *send-* and *receive*-tasks – we extended the model to allow for the implementation to be either a Service or a Message Channel, according to the more diverse means of interaction in JIAC, and in multi-agent systems in general. Also, while the service description can still be used for Web services, it supports additional attributes for the service's preconditions and effects, e.g., in the form of SWRL expressions, and whether the service refers to an actual service or a service template (Figure 8).

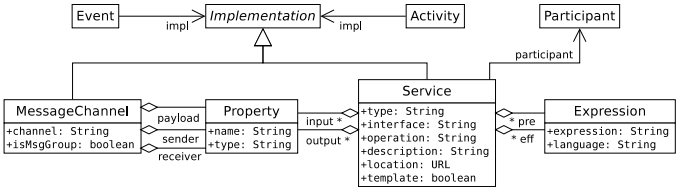

Figure 8. Extended Message- and Service-model used in VSDT BPMN editor.

Next, those processes can be exported to executable languages such as BPEL (Business Process Execution Language)

processes [41] or JIAC agent behaviours [42], [43], the latter being the execution environment used in this approach, which will be discussed in detail in Section V. 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.

2) State-of-the-Art: BPMN [40] can be used for describing services and service orchestrations in particular on a high level of abstraction. BPMN provides a rich syntax for modelling both the internal processes as well as the interactions of the system, and can thus be seen as a combination of Activity Diagrams and Sequence Diagrams of the Unified Modelling Language (UML). Further, while the process diagrams are easily understandable, the underlying formal model provides the attributes necessary to describe readily-executable programs.

BPMN is being used for modelling and generating service-oriented systems in a number of other works and can be seen as a de-facto standard for this task. Besides the mapping from BPMN to BPEL that is included in the specification itself [40, Chapter 14], alternative mappings have been proposed, e.g., by Ouyang et al. [44] and Mendling et al. [45]. Today, many process management systems can also execute the BPMN diagrams directly.

Besides those well-established paths, there are also approaches using BPMN for modelling agents and multi-agent systems. For instance, in GPMN, Jander et al. [46] combine BPMN processes with goal-hierarchies for Jadex agents equipped with BPMN interpreters. In WADE [47], on the other hand, a proprietary notation similar to BPMN is used, and the processes are transformed to executable code for JADE (Java Agent Development Framework).

Concerning the integration of semantics into BPMN, Barnickel et al. extended the Oryx BPMN editor with ontology matching capabilities, using OWL-DL [48], but to the best of the authors' knowledge there are no approaches towards integrating semantic service matching into BPMN or accordant process engines.

V. Runtime Components

In this section we will discuss the different components of the runtime 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 WSDL (Web Service Description Language), SOAP (Simple Object Access Protocol), and REST (Representational State Transfer), which can be integrated transparently with JIAC.

A. Multi-agent Framework

The execution environment is based on *JIAC V* (www.jiac.de), a multi-agent framework also incorporating many aspects of service-oriented architectures [37]. The agents are situated on agent nodes (runtime containers).

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, by any agent on the network, or exposed as a webservice to be used by different applications. 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 according to 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 accordant intention.

Each agent's behaviours and capabilities are defined in several agent beans, providing different general and application-specific functions (see Figure 9).

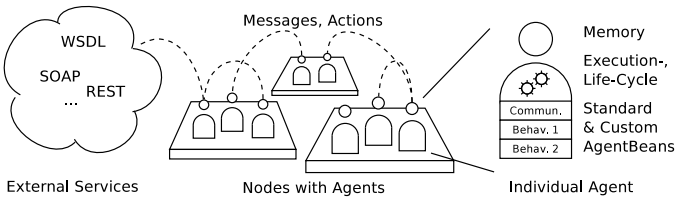

Figure 9. Components of a JIAC multi-agent system and individual agents (adapted from [49]).

Besides providing actions for others to use, agent beans can also implement periodic behaviour, or behaviours to be executed when the state of the agent changes (e.g., when it is starting or stopping). Also, they can attach observers to the agent's memory to react, e.g., to incoming messages or to changes in the environment. Finally, several application-independent beans can be added to the agent or the agent node as a whole, to provide certain functionalities, such as communication, persistence, migration, or reactive behaviour.

Integrating the semantic service matcher into JIAC was very natural and straightforward. The $SeMa^2$ itself has been wrapped into a JIAC agent node bean, i.e., there is one instance of the matcher for each individual node, shared by all agents on that node, hooking into the *directory* running on that node. Whenever a semantic service template (as opposed to a plain JIAC action template) is passed to the directory for service lookup, 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 the found capability is a standard JIAC action or a semantic service.

In order to utilise 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 or it can define WSDL or REST attributes.

B. WSDL and REST Web Service Integration

For interfacing with other services, the WSDL- and REST-services integration beans can be used. Those two components do both have the following two effects:

- all the JIAC actions accessible via the directory that have the 'webservice' scope will be exposed to the outside world as accordant WSDL or REST services, respectively,

- additional JIAC actions will be created and exposed, representing each of the WSDL and REST services known to those beans.

The respective input and output data types (e.g., XML schemas in the case of WSDL web services) are mapped to corresponding Java classes, and vice versa. The created web services are hosted by the same agent node using an integrated Jetty server. Thus, JIAC agents can seamlessly and transparently be integrated with both, REST and WSDL services.

C. Semantic Service Repository

Each of the semantic services is associated with a URI resource, holding their actual semantic descriptions in the form of an OWL-S document. While in theory each of the agents (or corresponding entities in a different runtime) could host their respective service descriptions themselves, this approach is not optimal, as the URI might change depending on where the agent is running. Instead, a central *Semantic Service Repository* is used for hosting the different service descriptions and their relevant ontologies, each being identified by a unique and invariant URI.

The service repository has been realised as a JIAC agent node, encapsulating a Jetty web server and providing a number of actions for deploying, searching, and fetching service descriptions. It also supports multiple filters, e.g., for only showing services that are currently running. Service descriptions can be deployed to the repository either statically, using the SSM tool, or dynamically whenever an agent providing the respective service starts. The service repository automatically parses the service descriptions and adapts all the internal URI references, e.g., to the descriptions of the services' inputs and outputs within the same document, to its current server address, where those resources are stored.

Each JIAC service, that is backed by a semantic service description, has an attribute `semanticURI` referring to the corresponding OWL-S resources. When the $SeMa^2$ matcher is invoked from within JIAC, the runtime will fetch the service descriptions, pass those to the matcher, receive the result, and finally return the action whose `semanticURI` corresponds to that matching result.

Currently, we are working towards distributing the service repository, to improve scalability for large numbers of services and service requests, as well as the ability to automatically assess the Quality-of-Service (QoS) of the invoked services, e.g., latency and time-to-complete.

D. 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 [42].

The process interpreter bean is composed of three layers (Figure 10): 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 accordant start event) as actions so they can be used by other agents.

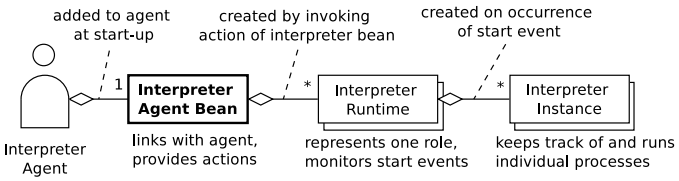

Figure 10. Layered architecture of BPMN Interpreter Bean. [42]

Whenever a BPMN diagram is added to the process engine bean for interpretation, an *interpreter runtime* is created, which is responsible for each process spawned from this diagram. It keeps track of start events and creates a new instance of that process whenever an event corresponding to the respective process start events occurs. Several volatile *process instances* are responsible for running the individual processes spawned by the runtime, evaluating conditions and assignments, 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. Depending on the type of the activities, different actions are taken, e.g., sending or receiving a message, invoking another service, executing some short script, or interacting with the user.

After one or more processes have been deployed to the interpreter – either at start-up or using the above-mentioned actions – in each step of the *interpreter bean*'s execution cycle, each *interpreter runtime* will advance each of its associated *interpreter runtimes* by one step, which in turn each execute each activity that is currently in a *ready* or *active* state.

Employing JIAC's communication and service infrastructure, the interpreted processes can discover and 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.

While this UI is intended for developers, a similar generic UI can also be used for invoking the services and interacting with the user in a more end-user friendly way, as we will show in the following.

E. Smart Personal Assistant and UI Renderer

The *Smart Personal Assistant* (SPA) is a UI framework for quickly developing adaptive, multimodal user interfaces for services [50], and is used in a number of research projects. While the usual SPA UI is manually created for the service at hand and styled to fit the design of the respective project, a special *Renderer UI* has been created, allowing to start any service, and also providing callbacks for user interaction triggered by the invoked service. Similar to the interpreter monitoring UI, input fields for querying the service parameters and for displaying the output are automatically derived from the respective classes, using the Java Reflection API.

While those generic, automatically generated UIs do not look as polished as the manually crafted ones, they allow

for quickly prototyping new complex services with rich user interaction and for integrating them into the user's workflow.

All of the here described components are open source and available for download from www.jiac.de.

VI. Methodology

In the following, we will sketch a process of how the different components introduced in the last three sections 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 11, 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, 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 [51], or generated from existing Java classes or EMF models [52].

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 a new instance of the service development process.

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. As a middle way, it is also possible to integrate short snippets of Java code into a BPMN process, using *script* tasks.

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

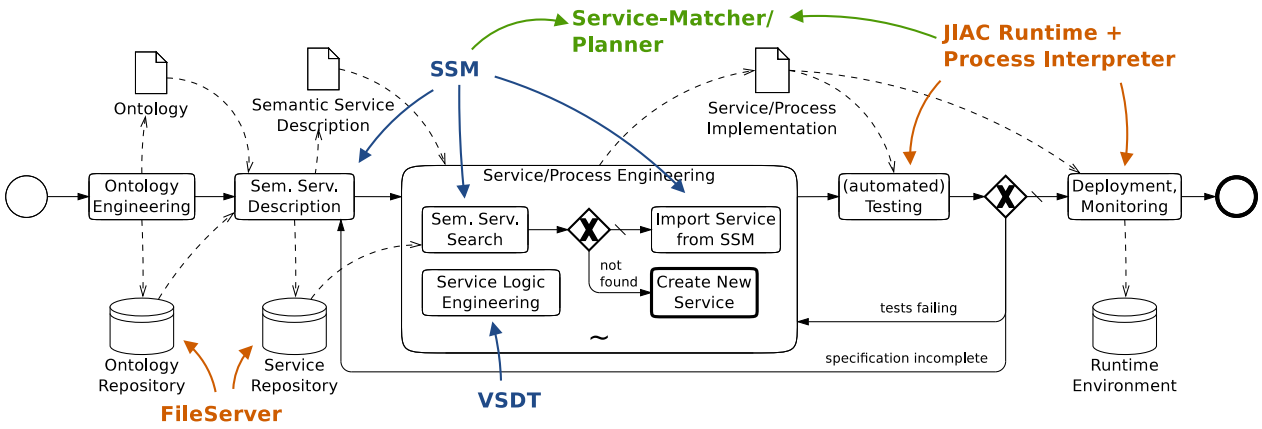

Figure 11. Semantic Service Management and Development Process, as a BPMN process, and associated components: Green: Semantic Service Core; Blue: Development Tools; Red: Execution Environment. The central activity is an *ad-hoc* subprocess, executing the embedded tasks as needed, in no fixed order.

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 pre-conditions 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.

VII. Semantic Service Management in Practice

In this section, we will explore how the semantic service engineering method discussed in this paper can be applied in practice. For this, we will have a look at two scenarios: First, we describe how the service matcher and the development tools have been used in a recently completed research project in the e-mobility domain. Then, we continue to describe one of our current projects, in which we are extending our approach for the *augmented reality* domain.

A. Semantic Services for E-Mobility

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 seen in Figure 7. The process was created using the VSDT editor, orchestrating services from the SSM. The finished process is deployed to the JIAC BPMN interpreter for execution and the SeMa^2 is used for service matching at runtime.

Our first scenario combines different basic services for searching for charging stations, reserving parking spaces and charging slots, as well as access control to the same. First, the process queries the user's information, particularly w.r.t. her current location, and available subscriptions for car sharing and parking space providers. It then uses the location to find charging stations that are close by using services from charging station provides for whom the user has a subscription. Those charging stations are then presented in a list for the user to choose from, using the UI renderer, and the user is asked for the time of reservation. The corresponding charging station reservation service is matched and the booking is made, if that time slot is not already taken, or the user is asked again. Finally, as soon as the user signals that she arrived at the location, another service is matched and invoked to handle the access control, if any.

In this example, the SeMa^2 can be used for finding relevant services for searching and reserving charging stations, depending on the user's subscriptions. For this, the `User` object is

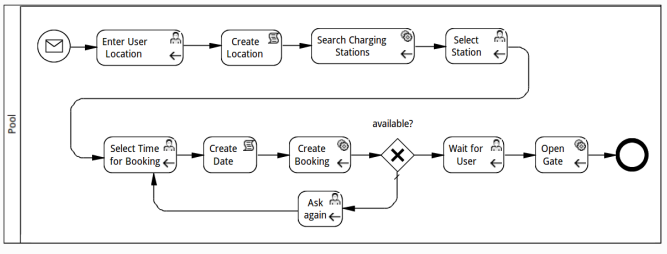

Figure 12. Service orchestration in the e-mobility use-case.

passed as an input parameter or as a context object to the service matcher, and the different services, having a precondition like `hasSubscription(?user, "Provider X")`, can then be selected by the semantic service matcher. This matching can be either be done a-priori, at design time, using the SSM for finding a set of matching services for one specific provider, or it can be deferred to runtime, making the selected services depending on the current user.

Of course, this required the relevant services not only to be semantically annotated, but to be so using the same domain model, or ontology, as used in the rest of the system. Since this is usually not the case, particularly when dealing with service provided by third parties (such as e-mobility providers) some of those services have to be wrapped accordingly.

B. Semantic Service Management for Augmented Reality

Another technical domain where semantic service management has the potential to lead to a boost of development is the area of augmented reality (AR) services. In the project AcRoSS (Augmented-Reality-based Product-Service Systems, more information available at: www.across-ar.de) we aim to set up a library for AR-services that can be used together with the software components presented in this paper in order to develop problem tailored solutions for small and medium-sized enterprises, for whom it is currently extremely hard and expensive to develop such specific solutions.

However, services for AR-glasses do have very strict requirements that have to be met. For instance, the hardware on the devices is currently still limited. Therefore, there has to be a very efficient concept for adaptive service processes. Furthermore, in many scenarios the glasses that will be used will be offline, meaning that the matching procedure and the services have to be located on the device itself. These issues as well as service specific aspects, like the quality of object recognition services, lead to the need for a service management concept that takes Quality-of-Service aspects into account.

One scenario within the AcRoSS project is about maintenance and repair of exposure machines, which are used for the creation of printing plates for the newspaper industry. The maintenance task includes the cleaning of exposure rolls, which means that they have to be taken out of the machine, maintained and correctly set into place again afterwards. Although this task sounds simple in first place, it is quite error-prone, since the rolls can also be set into place with the wrong direction or at the wrong place within the machine. The same challenge holds for even more complicated repair tasks. Currently, very experienced employees have to do these tasks at the client side, which is expensive. Using AR, the employee will be supported by services that recognize each part of the machine, search for related manuals in the backend, guide the employee what to do with the component and also check and display the machine's status. In order not to redevelop each process again and again for every machine, the process will be designed in an adaptive way, meaning that specific services like the retrieval of manuals will be selected dynamically as well as the request for the machine's status. Furthermore, at design time the developer will be able to select object recognition services via given Quality-of-Service parameters.

At the time this paper has been written the project was in the specification phase. A thorough evaluation will be done and published at a later point in time.

VIII. Conclusion

In this article, we presented an approach for semantically matching services and for combining those services to complex plans, both at design-time and at runtime, as well as a set of development tools and an accordant runtime environment for generating adaptive and flexible systems in service-oriented environments. Those planning components, development tools and runtime have been integrated into a methodology for semantic service management and engineering, covering all phases from semantic service description and service development up to testing and runtime monitoring. In this approach, semantic services are orchestrated in adaptive business processes, based on BPMN, where service templates can be specified within the process and dynamically matched to concrete services at runtime. This method has successfully been applied, among others, in a research project in the e-mobility sector. Currently, the same approach is adapted and extended for services in the augmented reality domain.

In the future, we plan to address a number of challenges related to automated service composition and planning [32]. Among others, we want to extend service matching and planning by taking quality-of-service aspects into account. Also, we want to investigate the use of heuristics for more efficient planning, to foster the usefulness of service planning in real-world applications.

Acknowledgements

This work is based on projects funded by the German Federal Ministry of Economic Affairs and Energy under the funding reference numbers 16SBB007A and 01MD16016F.

References

[1] J. Fähndrich, T. Küster, and N. Masuch, "Semantic service management for enabling adaptive and evolving processes," in Proc. of 11th Int. Conf. on Internet and Web Applications and Services (ICIW 2016), Valencia, Spain, May 22–26 2016, pp. 46–53.

[2] 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: 2016/11/28. [Online]. Available: http://www-ags.dfki.uni-sb.de/~klusch/s3/s3c-2012-summary-report.pdf

[3] 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, ser. Lecture Notes in Computer Science, A. Lomuscio, S. Nepal, F. Patrizi, B. Benatallah, and I. Brandi, Eds. Springer International Publishing, 2014, vol. 8377, pp. 240–247.

[4] J. Fähndrich, S. Weber, and S. Ahrndt, "Design and Use of a Semantic Similarity Measure for Interoperability Among Agents," in Multiagent System Technologies. Springer International Publishing, 2016, pp. 41–57.

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

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

[7] 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.

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

[9] 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.

[10] M. Klusch and P. Kapahnke. The Semantic Web Service Matchmaker Evaluation Environment (SME2). Last access: 2016/11/28. [Online]. Available: http://projects.semwebcentral.org/projects/sme2/ (2008)

[11] M. Klusch, P. Kapahnke, and I. Zinnikus, "SAWSDL-MX2: A machine-learning approach for integrating semantic web service matchmaking variants," in 2009 IEEE International Conference on Web Services (ICWS), IEEE Computer Society. IEEE, 2009, pp. 335–342.

[12] M. Klusch and P. Kapahnke, "The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection," vol. 15, Sep. 2012, pp. 1–14.

[13] M. Klusch, B. Fries, and K. Sycara, "OWLS-MX: A hybrid semantic web service matchmaker for OWL-S services," Web Semantics: Science, Services and Agents on the World Wide Web, vol. 7, no. 2, Apr. 2009, pp. 121–133.

[14] F. E. Gmati, N. Y. Ayadi, A. Bahri, S. Chakhar, and A. Ishizaka, "A framework for parameterized semantic matchmaking and ranking of web services," in Proc. of 12th Int. Conf. on Web Information Systems and Technologies, 2016, pp. 54–65.

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

[16] 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.

[17] 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.

[18] 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.

[19] G. Rodríguez, Á. Soria, and M. Campo, "Artificial intelligence in service-oriented software design," Engineering Applications of Artificial Intelligence, vol. 53, no. C, Aug. 2016, pp. 86–104.

[20] G. Markou and I. Refanidis, "Non-deterministic planning methods for automated web service composition," Artif. Intell. Research, vol. 5, no. 1, 2016, p. 14.

[21] G. Zou, Y. Gan, Y. Chen, and B. Zhang, "Dynamic composition of Web services using efficient planners in large-scale service repository," Knowledge-Based Systems, vol. 62, May 2014, pp. 98–112.

[22] G. Zou et al., "QoS-aware dynamic composition of Web services using numerical temporal planning," 2012.

[23] G. Zou, Y. Chen, Y. Xu, R. Huang, and Y. Xiang, "Towards automated choreographing of web services using planning," AAAI, 2012.

[24] P. Rodriguez-Mier, M. Mucientes, and M. Lama, "Automatic web service composition with a heuristic-based search algorithm," in 2011 IEEE International Conference on Web Services (ICWS). IEEE, 2011, pp. 81–88.

[25] H. Meyer and M. Weske, "Automated Service Composition Using Heuristic Search," in Agents and Computational Autonomy. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 81–96.

[26] J. Hoffmann and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," Journal of Artificial Intelligence Research, vol. 14, no. 1, 2001.

[27] M. Klusch and A. Gerber, "Fast Composition Planning of OWL-S Services and Application," in Proc. of European Conference on Web Services (ECOWS '06). IEEE, 2006, pp. 181–190.

[28] B. Bonet and H. Geffner, "Planning as heuristic search," Artificial Intelligence, vol. 129, no. 1-2, Jun. 2001, pp. 5–33.

[29] A. Mediratta and B. Srivastava, "Applying planning in composition of web services with a user-driven contingent planner," IBM Research, 2006.

[30] Y.-Y. FanJiang and Y. Syu, "Semantic-based automatic service composition with functional and non-functional requirements in design time: A genetic algorithm approach," Information and Software Technology, vol. 56, no. 3, Mar. 2014, pp. 352–373.

[31] F. Lécué, A. Léger, and A. Delteil, "DL Reasoning and AI Planning for Web Service Composition," in 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology. IEEE, 2008, pp. 445–453.

[32] M. Lützenberger, T. Küster, N. Masuch, and J. Fähndrich, "Multi-agent systems in practice – when research meets reality," in Proc. of 15th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2016), J. Thangarajah, K. Tuyls, C. Jonker, and S. Marsella, Eds. Singapore: IFAAMAS, May 2016, pp. 796–805.

[33] Eclipse Foundation. Eclipse. Last access: 2016/11/28. [Online]. Available: http://www.eclipse.org/ (2016)

[34] L. de Silva, S. Sardiña, and L. Padgham, "First Principles Planning in BDI Systems," in Proccedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009), C. Sierra, K. S. Decker, and J. S. Sichman, Eds. Budapest, Hungary: IFAAMAS, May 2009, pp. 1105–1112.

[35] 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/

[36] 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, pp. 499–510.

[37] 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.

[38] D. Elenius et al., "The OWL-S editor – a development tool for semantic web services," in The Semantic Web: Research and Applications. Springer, 2005, pp. 78–92.

[39] N. Srinivasan, M. Paolucci, and K. Sycara, "Semantic web service discovery in the OWL-S IDE," in Proceedings of the 39th Annual Hawaii International Conference on System Sciences - Volume 06, ser. HICSS '06. Washington, DC, USA: IEEE Computer Society, 2006.

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

[41] 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.

[42] T. Küster, A. Heßler, and S. Albayrak, "Process-oriented modelling, creation, and interpretation of multi-agent systems," International Journal of Agent-Oriented Software Engineering, 2016, to appear.

[43] 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.

[44] C. Ouyang, M. Dumas, W. M. P. van der Aalst, A. H. M. ter Hofstede, and J. Mendling, "From business process models to process-oriented software systems," ACM Transactions on Software Engineering and Methodology, vol. 19, no. 1, August 2009, pp. 1–37.

[45] J. Mendling, K. B. Lassen, and U. Zdun, "On the transformation of control flow between block-oriented and graph-oriented process modelling languages," International Journal of Business Process Integration and Management (IJBPIM), vol. 3, no. 2, 2008, pp. 96–108.

[46] K. Jander, L. Braubach, A. Pokahr, W. Lamersdorf, and K. Wack, "Goal-oriented processes with GPMN," International Journal on Artificial Intelligence Tools, vol. 20, no. 6, 2011, pp. 1021–1041.

[47] F. Bergenti, G. Caire, and D. Gotta, "Interactive workflows with WADE," 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, vol. 0, 2012, pp. 10–15.

[48] N. Barnickel, J. Böttcher, and A. Paschke, "Semantic mediation of information flow in cross-organizational business process modeling," in Proc. of 5th Int. Workshop on Semantic Business Process Management SBPM 2010, held in conjunction with the European Semantic Web Conference (ESWC 2010), Heraklion, Greece, May 2010, pp. 21–28.

[49] 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.

[50] N. Braun, R. Cissée, and S. Albayrak, "An agent-based approach to user-initiated semantic service interconnection," in Service-Oriented Computing: Agents, Semantics, and Engineering: AAMAS 2007 International Workshop, SOCASE 2007, Honolulu, HI, USA, May 14, 2007. Proceedings, J. Huang et al., Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 49–62.

[51] Stanford. Protégé. Last access: 2016/11/28. [Online]. Available: http://protege.stanford.edu/ (2016)

[52] Eclipse Foundation. Eclipse Modeling Framework (EMF). Last access: 2016/11/28. [Online]. Available: https://eclipse.org/modeling/emf/ (2016)