Matching Composed Semantic Web Services at Publishing Time

Andreas Friesen, Michael Altenhofen

{andreas.friesen, michael.altenhofen}@sap.com
SAP Research, CEC Karlsruhe, SAP AG, Vincenz-Prießnitz-Str. 1,
D-76131 Karlsruhe, Germany

Abstract. This paper describes an algorithm optimizing the discovery process for composed semantic web services. The algorithm can be used to improve discovery of appropriate component services at invocation time. It performs semantic matchmaking of goals of a composed service to appropriate component services at publishing time. The semantic discovery problem at invocation time is therefore reduced to a selection problem from a list of available (already discovered) component services matching a goal of the composed service.

1. Introduction

In the last few years a new approach to design (integrate) software applications, called Service-oriented Architecture (SOA), arose and became very popular. The main idea of that approach is to build applications through composition of loosely coupled components called services. This approach promises to increase interoperability and to reduce integration costs.

Today, a standardized technological realization of SOA, called Web Services, is available and becomes more and more widespread. Web Services consist of three core technologies: SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language) and UDDI (Universal Description, Discovery and Integration). SOAP describes an interoperable XML-based message exchange protocol that allows communication with a Web Service over the Internet [6]. WSDL describes the operations (interface) of a Web Service [5]. UDDI allows the service providers to publish and service requesters to discover Web Services [7]. There are numerous emerging additional technologies on top of Web Services dealing with workflows, transactions, security and so on (W3C: http://www.w3c.org, OASIS: http://www.oasis-open.org). However, the above technologies support only manual integration of the Web Services into composite applications. This is because their semantics are not explicitly described in a formalized way and therefore can be understood only by humans but not by machines.

In order to enable automated usage of Web Services several competing initiatives are developing infrastructures, so-called Semantic Web Services, which combine Semantic Web and Web Services technologies (e.g., W3C Semantic Web: http://www.w3.org/2001/sw/, SWWS: http://www.swsi.org/, SDK: http://www.sdkcluster.org.)

Web Service usage comprises many aspects of Web Services, e.g. publication, discovery, selection, composition, mediation, negotiation, invocation, monitoring, compensation and recovery. Within this big picture, the paper describes an optimization of semantic discovery in the context of composed web services.

2. Atomic vs. Composed Web Services

Services published to a registry can be divided into two classes: atomic services and composed services. An atomic service does not use any further services to perform its functionality. Opposed to that, a composed service relies on a set of composed or atomic services. We call this set „component services“ of the composed service.

However, a composed service making an ultimate decision about the set of its component services at design time will probably become non-optimized or even unable to perform its task for the following reasons:

- New, more optimal component services may have been added to the registry in the meantime.
- The properties of the existing services may change (become more/less optimal or usable/unusable for the composed service).
- Existing services may be removed from the registry.

In order to protect itself from becoming less optimal or unable to perform its tasks a composed service has to:

- Discover component services in order to detect new, changed or removed services
- Select from the discovered set of available services the services used for the current execution of the service according to some optimization algorithm

A composed service discovers a component service using some goal. Since a composed service also has to be published to a registry, its goals used to find appropriate component services are known to it at publishing time and do not change as long as the service does not change or has been removed. However, at the time being the goals of a composite service used to discover its component services are hidden in its internal business logic (also known as orchestration) and can therefore only be used by the service itself.

The above considerations show that a composite service, in order to keep its functionality optimized, has to discover its component services again and again either at each invocation or at some other time intervals specified by its internal logic.

3. Semantic Discovery

Semantic discovery of Web Services means semantic reasoning over a knowledge base where a goal describes the required web service capability as input. Semantic discovery adds accuracy to the search results in comparison to traditional Web Service discovery techniques, which are based on syntactical searches over keywords contained in the web service descriptions [8], [9]. The additional accuracy of a match is expensive in terms of required computational power. The expensiveness of semantic matchmaking has several aspects influencing the design of the Semantic Web Services infrastructure in different ways. Besides the quality of the semantic match, we consider the following two aspects of outmost importance:

- Response time
- Scalability

Response time specifies how long semantic discovery takes to find a set of web services matching a goal. This criterion can restrict or even prohibit service usage in certain scenarios, where a Web Service has to be discovered and directly invoked and the total time for discovery and invocation has to be short. For instance, a composed service may semantically discover required component services at runtime. If potential service requesters of the composed service expect short response times this service becomes unusable to them even if the functionality offered by the service meets their needs.

Scalability specifies how many semantic discovery requests can be processed within a given unit of time. If a composed service identifies its component services using time-consuming semantic discovery techniques each time it is invoked, discovery may become a bottleneck.

A practical approach to the above problems is to execute semantic discovery only if necessary, e.g. by buffering results from former searches. An even better approach would be to do semantic matchmaking or some sub-steps of this process ahead of the current discovery request. Since goals and service capabilities rely on the same ontological concepts, it is possible to match a service capability to the ontological concepts even if the goal is not known at that point in time. In [1] an OWL-S/UDDI matchmaker is proposed that performs reasoning at publishing time in order to find matches of different quality (exact, subsumption, plug-in, fail) between capabilities and the ontological concepts they rely on. The term „matchmaking“ is often used as a synonym for semantic discovery. The matches found are stored as lists (identifying the different qualities of the match) attached to the ontological concepts. The matching of goals and capabilities is therefore reduced to computing intersections between capabilities lists of the concepts used in the goal. A goal is described, e.g., in:

- OWL-S through inputs and outputs [2]
- WSMO through postconditions and effects [3]

4. Optimization Approach

We propose to make the goals of the composed service used to discover component services part of its public description (public to a registry, but not necessarily to other clients of the registry). This would allow the application of a new discovery algorithm, which reduces the efforts to discover component services at invocation time significantly. In fact, using the algorithm described below, the discovery process during the service invocation will take a constant time in order to find a set of services matching the goal, so the overall process identifying component services takes a constant time for semantic discovery plus the time for the selection from a list of services matching the goal. (The selection time remains dependent on an internal algorithm of the composed service.)

Before introducing the algorithm let's consider what is the additional value of having goals explicitly published in the web service description (e.g., as a list). If the goals are known, the discovery process can be executed by a registry at publishing time and the list of the available services matching the goal can be stored (linked in some way to the goal) in the registry. This means that, if the service sends a discovery query with the goal to a registry, the registry first tries to find the goal and returns in case of success the list of services linked to it. Only if the goal could not be found, the registry starts the semantic discovery in the "traditional way". This does not yet solve the problem of detecting new, changed and removed services possibly impacting the functionality of the composed service, since the service could make the discovery once and store it for the further use internally (outside of the registry). So what is needed is a way to update the list of available services linked to the goal if something changes (e.g., a new service is added to the registry, or an existing service is changed or removed).

The following observation shows how the required behavior can be achieved. A goal presents in fact a partial description of the service capability, e.g. in WSMO a goal is described through postconditions and effects while a service capability is described by preconditions, assumptions, postconditions and effects. This means, that if a registry can find matching service capabilities using a goal then it can also find matching goals using service capabilities.

5. Optimization Algorithm

The optimization algorithm requires some changes (extensions) on the Web Services registry structure, which are introduced in the next subsection.

5.1 Foundations

The key abstractions of a registry used for semantic discovery can be described by the following key abstractions illustrated in Figure 1:

- Discovery Service
- Publishing Service
- Knowledge Base
- Goal-Capability Association Storage (GCA-Storage)

Discovery Service provides an interface allowing execution of semantic search queries against the Knowledge Base.

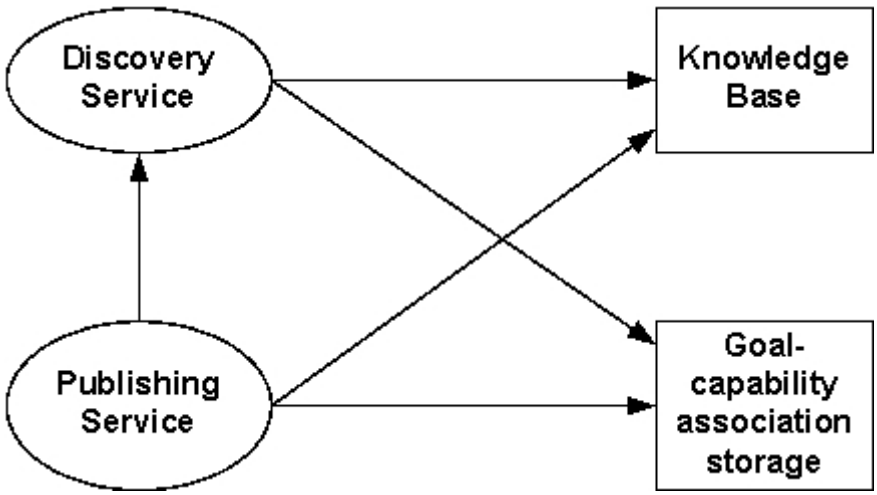

Figure 1 Key abstractions of a registry

Publishing Service provides an interface to add new services or to update or remove existing services from the knowledge base of the registry.

The published services are stored in the Knowledge Base used for semantic reasoning.

The Web Service Description (WSD) of a composed service is modified (extended) in a way that the goals a composed service uses to find component services are described explicitly in its Web Service description.
We introduce a GCA-Storage able to store associations (links) between explicitly described goals in the WSD of a composed service and the WSDs of potential component services (containing matching service capabilities). The GCA-Storage is illustrated in Figure 2.

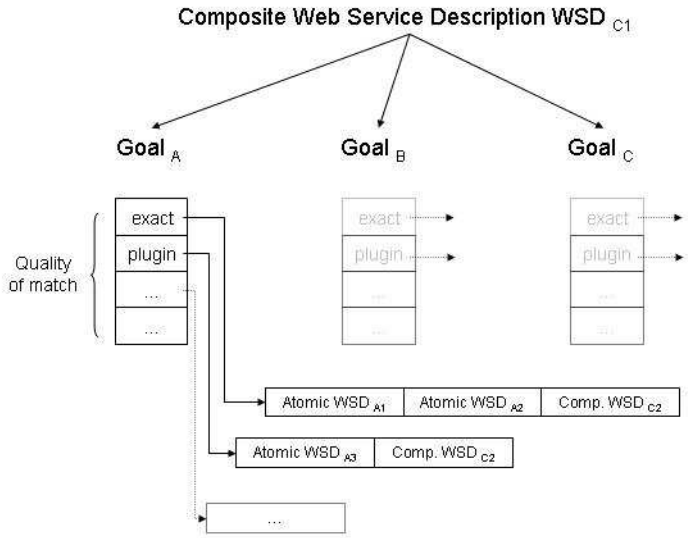

Figure 2 Structure of goal-capability association storage

The Publishing Service is modified in a way allowing the operations described in the algorithm below.

The Discovery Service is modified in a way such that not only a goal can be used to find service capabilities but also a service capability can be used as a search criterion in order to find goals explicitly stored in the directory.

5.2 Matching Goals and Service Capabilities at Publishing Time

The following algorithm performs semantic discovery at publishing time and keeps the goal-service association storage consistent for any changes of goals or service capabilities published to the registry.

```
publishService(wsd){
choose (wsd){
 newAtomic:     perform InsertNewAtomic;
 updateAtomic:  perform UpdateAtomic;
 deleteAtomic:  perform DeleteAtomic;
 newComposed:   perform InsertNewComposed,
InsertNewAtomic;
 updateComposed: perform UpdateComposed, UpdateAtomic;
 deleteComposed: perform DeleteComposed, DeleteAtomic;
 }
}
```

InsertNewComposed: Extract goal descriptions from the service capability description of the composed service and use the discovery service to find matching service capabilities. Store the lists of found service capabilities (together with the goals) to the GCA-Storage, i.e. associate them with the matching goals.

UpdateComposed: Remove old associations (between goals and service capabilities) from the GCA-Storage, find and store new associations in the GCA-Storage (Further optimization possible, e.g. compare the new and old WSD version to find differences between old and new goals. If a goal has not changed no action is required, i.e. no reasoning is required and the already existing association remains valid.)

DeleteComposed: Remove associations for the goals contained in the WSD from the GCA-Storage

InsertNewAtomic: Use discovery service with service capability as search criterion in order to find matching goals and store found matches to the GCA-Storage.

UpdateAtomic: Delete old associations from GCA-Storage. Use discovery service to find new matches and store them in the GCA-Storage.

DeleteAtomic: Remove associations for the service capability contained in the WSD from the GSA-Storage.

Note that a composed service can act as an atomic service (component service) for other composed services. Therefore, after an operation for a composed service has been performed, an associated operation for the atomic service must follow (associating herewith its service capability with goals of some other services).

6. A Business Scenario

In this scenario the "Vehicle- & Insurance-selling Market Place DE" provides a user-friendly interface to vehicle buyers (allowing specification of the type and the preferred location of a vehicle seller as well as an appropriate insurance) and aggregates the resources of the vehicle sellers, regional vehicle-selling market places and insurances to reach a broad variety of offered vehicle types, quantity of offered vehicles, broad location coverage and appropriate insurances (making its service attractive to the vehicle buyers). The example will help us to demonstrate:

- the difference between semantic discovery and traditional keyword-based discovery approaches concerning the quality of match
- the reduction of semantic discovery requests of the composed web services because of the introduced optimization algorithm

The vehicle sellers (in the example “Munich Luxury Cars” and “Dresden Motorcycles”) are bound to a location, provide a restricted number of vehicle types and have a restricted number of vehicles on offer.

The regional vehicle selling market places (in the example “Bavarian Car Market Place” and “Cars and Motorcycles Saxony”) aggregate offers of vehicle sellers providing a broad variety and quantity of offered vehicles for a distinct region.

The vehicle sellers are interested to reach as many potential vehicle buyers as possible and therefore to be associated with as many vehicles selling market places as possible. Hence, a vehicle seller has to specify its service as precise as possible in order to achieve high rankings (as exact match as possible) in the discovery and selection processes of the vehicle-selling market places.

Also the vehicle insurances are interested to be connected to as many market places as possible. Also for the insurances applies, to get high priority they have to exactly specify the vehicle types, conditions, regional restrictions concerning vehicle registration, etc.

“Vehicle & Insurance Market Place.de” has the following goal: *“Buy vehicle in Germany. Buy vehicle insurance for a vehicle registered in Germany”*.

“Bavarian Car Market Place” has the goal: *“Buy car in Bavaria, Germany. Buy car insurance for a car registered in Bavaria, Germany”* and provides service capability: *“Sell Car in Bavaria, Germany. Sell insurance for a car registered in Bavaria Germany”*.

“Cars & Motorcycles Saxony”: *“Buy car or motorcycle in Saxony, Germany”* and provides service capability: *“Sell car or motorcycle in Saxony, Germany”*.

“Bavarian Car Insurance” has the service capability: *“Sell insurance for a car registered in Bavaria, Germany”*.

“Direct Vehicle Insurances.de” has the service capability: *“Sell insurance for a vehicle registered in Germany”*.

“Munich Luxury Cars” has the service capability: *“Sell luxury car in Munich, Bavaria, Germany”*.

“Dresden Motorcycles” has the service capability: *“Sell motorcycle in Dresden, Saxony, Germany”*.

Let’s assume, that none of the above services is published (Figure 3 illustrates the business scenario example after the steps described in the following have been executed.). Vehicle types are described by the vehicle ontology (vo) and locations by the location ontology (lo) (Figure 4 adumbrates these two ontologies).

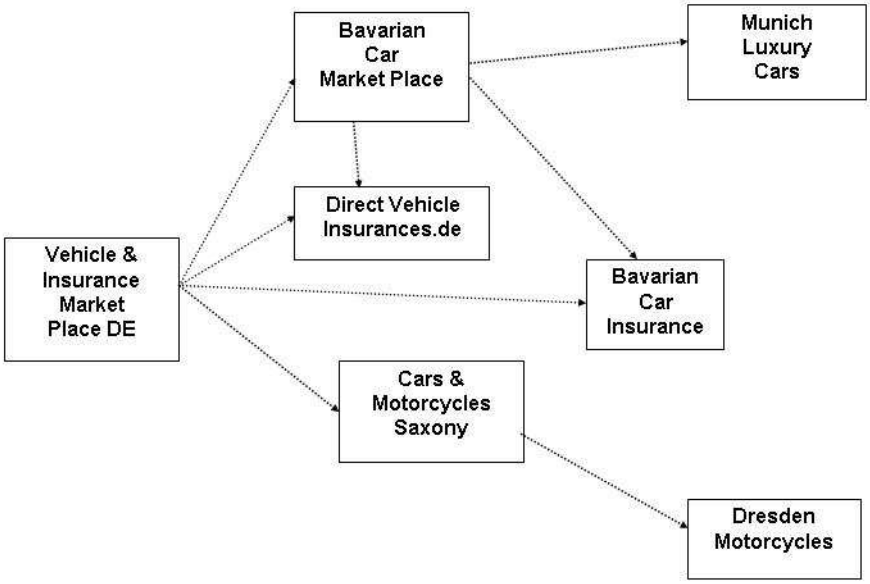

Figure 3 Composed vehicle market places

Step 1: The insurance services are published. Since they are atomic the publishing algorithm performs InsertNewAtomic. However, since no services with goals requiring vehicle insurance are found in the registry at this point of time the insurance services are not associated with any goals in the GCA-Storage.

Step 2: The "Vehicle and Insurance Market Place.de" service is published. This service is composite so the algorithm performs InsertNewComposed and stores the associations to the both insurance services in the GCA-Storage (see Figure 2 for details how the associations are stored). Subsequent InsertNewAtomic finds no goals which can be associated with its service capability so no new associations are stored to the GCA-Storage.

Step 3: The Saxony and Bavarian Market Places are published. InsertNewComposed finds no vehicle sellers. Bavarian Market Place is associated with Bavarian Car Insurance ("exact match") and with Direct Vehicle Insurances.de ("subsume"). InsertNewAtomic stores associations of "plug-in" quality to "Vehicle and Insurance Market Place.de" for both regional market places.

Step 4: Finaly, the Munich and Dresden dealers are published. InsertNewAtomic provides associations to their regional market places respectively.

The "Vehicle and Insurance Market Place.de" is now enabled to sell Luxury Cars in Bavaria and Motorcycles in Saxony and can sell appropriate insurances.

Let's assume "Dresden Motorcycles" extends its offer and changes its capability to "Sell car and motorcycle in Dresden, Saxony, Germany". UpdateAtomic updates the association with the Saxony Market Place enabling the "Vehicle and Insurance Market Place.de" to sell not only motorcycles but also different types of cars in Sachsen.

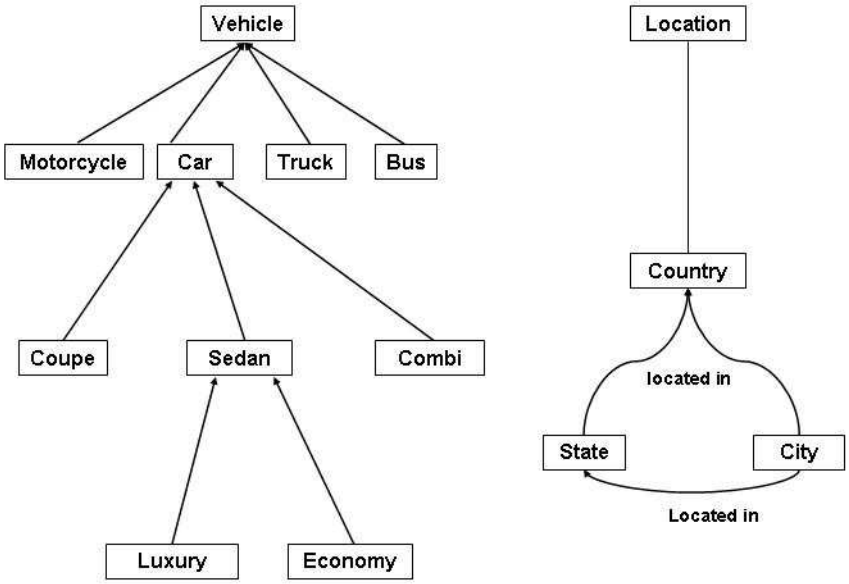

Figure 4 Vehicle and location ontologies

Let's assume "Direct Vehicle Insurances.de" removes its service. DeleteAtomic removes associations to "Bavarian Market Place" and "Vehicle and Insurance Market Place.de" from GCA-Storage. This implies that now these market places can sell insurances only for cars registered in Bavaria.

Usually, a goal is used to match service capabilities. In the introduced algorithm also goals extracted from service capabilities of "atomic services" are used to match goals of composed services. In this case the quality of match is different. Let's assume a vehicle buyer has a goal to buy a car in Bavaria: "*Buy a car in Bavaria, Germany"*.

From the viewpoint of the car buyer the service capability of "Munich Luxury Cars" has the matching quality "plug-in" since it sells only luxury cars and not all types of cars as required by the goal.

Let's assume the goal is already published. The car dealer publishes its "Munich Luxury Cars" service with the above described service capability. The publishing service extracts, let us say in WSMO, the postcondition and effect from the service capability description and builds a goal. (It is not difficult to extract the goal from the service capability, e.g., in WSMO the postconditions and effects describing a goal are also a part of a service capability description.)

With the extracted goal the publishing service requests the discovery service to match it against the goals published with descriptions of composed services. In this example the quality of match will be "subsumption". However, since in this case the direction of the match was "service capability -> goal" and not "goal -> service capability". This means, that the found goal subsumes the service capability and therefore from the viewpoint of that goal the service capability of the luxury car dealer is a "plug-in".

7. Conclusion

The introduced approach demonstrates that at least for composed services in applicable business scenarios the semantic matchmaking between goals and service capabilities can be executed at the publishing time instead of invocation time. This approach reduces also the total number of semantic queries due to buffering of existing matches in the goal-service association storage. The introduced algorithm ensures that the goal-service association storage can be kept consistent for any changes of goals or service capabilities, if they are published to the registry.

Acknowledgement: The work is partially funded by the European Commission under the project DIP.

References

[1] N. Srinivasan, M. Paolucci and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput." First International Workshop on Semantic Web Services and Web Process Composition **(SWSWPC 2004) 6-9, 2004, San Diego, California, USA

[2] OWL-S, http://www.daml.org/services/owl-s/1.0/owl-s.html

[3] D. Roman, H. Lausen, U. Keller, et all, Web Service Modeling Ontology (WSMO), http://www.wsmo.org/2004/d2/v1.0/20040920/

[4] M. Stollberg, R. Lara, et all, WSMO Use Case "Virtual Travel Agency", http://www.wsmo.org/2004/d3/d3.3/v0.1/

[5] Christensen, E. et all; Web Service Description Language (WSDL) 1.1, March 2001, http://www.w3.org/TR/wsdl

[6] Box, D. et all; Simple Object Access Protocol (SOAP) 1.1, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[7] Bellwood, T. et all; UDDI Version 2.04 API Specification, July 2002, http://uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm

[8] U. Keller, R. Lara, A. Polleres, "WSMO Web Service Discovery", http://www.wsmo.org/2004/d5/d5.1/v0.1/20041112/

[9] S. Grimm, B. Motik, C. Preist, „Variance in e-Business Service Discovery", ISWC 2004, http://www.fzi.de/wim/publikationen.php?id=1238