SEMANTIC DISCOVERY OPTIMIZATION
Matching composed semantic web services at publishing time

Andreas Friesen, Michael Altenhofen

SAP Research, CEC Karlsruhe, Vincenz-Prießnitz-Str. 1, D-76131, 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) (Box, 2000), WSDL (Web Service Description Language) (Christensen, 2001) and UDDI (Universal Description, Discovery and Integration) (Bellwood, 2002). 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 SW: 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/COMPOSED 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 (Keller, 2004), (Motik, 2004). 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 (Paolucci, 2004). Besides the accuracy of a semantic match, also the following two aspects have to be considered:

- 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 (Paolucci, 2004) 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, intersection, fail*) between capabilities and the ontological concepts they rely on. *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 semantically described, e.g., in:

- OWL-S through inputs and outputs (OWL-S, 2003)
- WSMO through postconditions and effects (Roman, 2004)

4 OPTIMISATION 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 allows achieving the required behavior. 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 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.

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.

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.
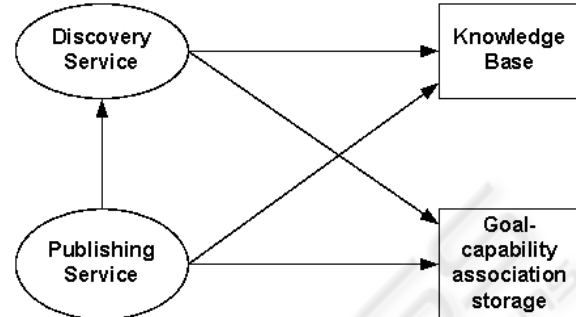

Figure 1: Key abstractions of a registry

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.
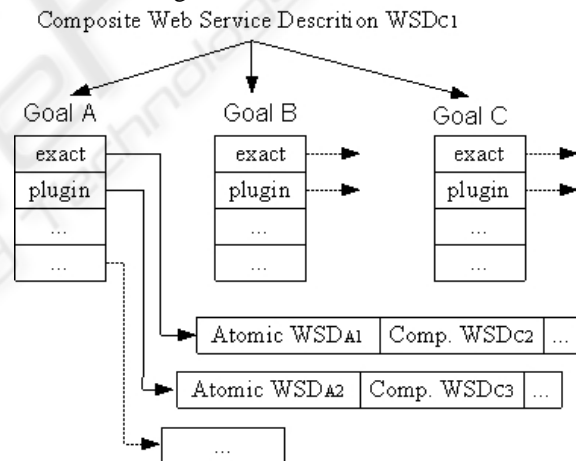

Figure 2: Goal-capability association storage

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.

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 and use the discovery service to find matching service capabilities. Store the lists of found service capabilities to the GCA-Storage (associating 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 the 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).

The proposed optimization approach can be used in different business scenarios. Of special interest are scenarios with composed services aggregating functionality of supplying component services, e.g., in the example of a Virtual Travel Agency (Stollberg. 2004). A composed service in such scenario can profit from the introduced optimization procedure if it can express its (possibly dynamically changing) requirements on the resources (it expects to be provided by component services) as a set of semantically described goal descriptions.

6 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

REFERENCES

Bellwood, T., 2002. Bellwood, T. et all; UDDI Version 2.04 API Specification, July 2002, http://uddi.org/

Box, D., 2000. Box, D. et all; SOAP 1.1, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/

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

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

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

OWL-S, 2003. http://www.daml.org/services/owl-s/1.0/owl-s.html

Paolucci, M., 2004. N. Srinivasan, M. Paolucci and K. Sycara, "Adding OWL-S to UDDI, implementation and throughput", SWSWPC 2004, 6-9, 2004,

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

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