Kontext-basierte Personalisierung von Web Services

Markus Keidl Stefan Seltzsam Christof König Alfons Kemper

Universität Passau
D-94030 Passau
⟨keidl|seltzsam|koenig|kemper⟩@db.fmi.uni-passau.de

Abstract: Web Services finden zunehmend im Business-To-Consumer-Bereich Verwendung. In diesem Umfeld ist der Kreis der Benutzer heterogen und sehr groß. Jeder hat seine eigene Vorstellung davon, wie der Web Service mit ihm interagieren soll. Web Services müssen deshalb personalisierbar und möglichst flexibel sein, um auf Benutzer in der gewünschten Weise reagieren zu können. In diesem Beitrag präsentieren wir Technologien, die die Entwicklung von derartigen Web Services unterstützen. Mit der dynamischen Dienstauswahl haben Web Services die Möglichkeit, Dienste zur Laufzeit basierend auf einer technischen Spezifikation der gewünschten Dienste auszuwählen und aufzurufen. Mit Vorgaben kann die dynamische Dienstauswahl beeinflusst werden. Vorgaben können direkt beim Aufruf von Diensten angegeben werden oder im Kontext eines Dienstes enthalten sein. Die Verwendung von Kontexten ermöglicht Web Services, Benutzern eine angepasste und personalisierte Version ihrer selbst zur Verfügung zu stellen. Wir präsentieren diese Technologien im Rahmen des ServiceGlobe-Systems, einer Dienstplattform für die flexible und ausfalltolerante Ausführung von Web Services.

1 Einleitung

Web Services haben sich als effizientes Mittel zur Integration von heterogenen Anwendungen im B2B-Bereich (Business-To-Business-Bereich) etabliert. Mit Web Service, im Folgenden auch Dienst genannt, bezeichnen wir eine autonome Softwarekomponente, die durch eine eindeutige URI identifiziert wird und mit den Internet-Standards XML, SOAP oder HTTP angesprochen werden kann. Zunehmend finden Web Services auch im B2C-Bereich (Business-To-Consumer-Bereich) Verwendung. In diesem Umfeld ist der Nutzerkreis heterogener. Er besteht nicht aus einigen Unternehmen, die gegebenenfalls miteinander kooperieren, sondern aus einer großen Menge von Benutzern, jeder mit anderen Vorstellungen davon, wie der Dienst auf ihn reagieren soll. Web Services müssen deshalb personalisierbar und flexibel sein, um mit den Benutzern in der gewünschten Weise interagieren zu können.

In diesem Beitrag präsentieren wir mit der dynamischen Dienstauswahl eine Technologie, die die Entwicklung von personalisierbaren und flexiblen Web Services unterstützt. Sie ermöglicht es Web Services, andere Dienste zur Laufzeit basierend auf einer technischen Spezifikation der gewünschten Dienste auszuwählen und aufzurufen, und stellt damit eine Abstraktion von den tatsächlichen Diensten dar. Mit Vorgaben kann die dynamische

Dienstauswahl beeinflusst werden. Dienste können damit beim Aufruf z. B. anhand ihrer Metadaten ausgewählt werden. Vorgaben können zum einen direkt beim Aufruf von Diensten angegeben werden, sie können aber auch im Kontext eines Dienstes enthalten sein. Unter Kontext verstehen wir dabei Informationen, die ein Web Service benötigt, um Benutzern eine angepasste und personalisierte Version seiner selbst zur Verfügung zu stellen. Die Integration von Vorgaben für die dynamische Dienstauswahl in die Kontexte von Web Services ermöglicht die Personalisierung dieser Web Services, indem die Vorgaben von der Dienstplattform automatisch verwendet werden. Damit kann ein Benutzer einen Web Service z. B. dazu veranlassen, nur kostenlose Dienste aufzurufen.

Wir präsentieren diese Technologien im Rahmen des ServiceGlobe-Systems, einer Dienstplattform für die flexible und ausfalltolerante Ausführung von Web Services. ServiceGlobe unterstützt mobilen Code, d. h. Dienste können zur Laufzeit auf beliebige Internet-Server, die an das ServiceGlobe-System angeschlossen sind, verteilt und dort instanziiert werden. Funktionalität, wie sie für Dienstplattformen Standard ist, z. B. SOAP-basierte Kommunikation, ein Transaktionssystem sowie ein Sicherheitssystem [SBK01], wird ebenfalls unterstützt. Diese Bereiche werden bereits von existierenden Technologien abgedeckt und stellen deshalb nicht den Fokus dieses Beitrags dar. Die Verwendung von ServiceGlobe ermöglicht die Integration der vorgestellten Technologien direkt in die Dienstplattform sowie die Ausnutzung spezifischer ServiceGlobe-Technologien.

Der große Erfolg von Web Services spiegelt sich im kommerziellen Bereich in einer Reihe von Dienstplattformen und Produkten wider, z. B. Sun ONE [Sun], Microsoft .NET [NET] und HP Web Services Platform [WSP]. Diese Produkte und Plattformen basieren auf den bekannten Web-Standards XML [BPSMM00], SOAP [BEK+00], UDDI [UDD00] und WSDL [CCMW01] und bieten Werkzeuge an, um existierende Anwendungen schnell und unkompliziert als Dienste anbieten zu können. Daneben existieren Forschungsplattformen wie ServiceGlobe [KSSK02, KSK02] und SELF-SERV [BDSN02], die sich gezielt auf einzelne Aspekte der Entwicklung und Ausführung von Web Services konzentrieren. Im SELF-SERV-System werden z. B. Dienste mit gleichartigen Schnittstellen zu so genannten Service Communities zusammengefasst. Es wird allerdings nicht näher auf Strategien zur Auswahl von Diensten aus diesen Service Communities eingegangen, wie dies in ServiceGlobe mit der dynamischen Dienstauswahl der Fall ist. Den Schwerpunkt in SELF-SERV bildet vielmehr die Komposition von Web Services mit Hilfe von Statecharts. Weitere Sprachen zur Komposition von Web Services sind IBM's WSFL (Web Services Flow Language) [Ley01], Microsoft's XLang [Tha01] und HP's WSCL (Web Services Conversation Language) [BBB+01]. Compaq's Web Language [Mar99] (ehemals WebL) wurde hauptsächlich zum Laden, Parsen und Generieren von HTML- und XML-Inhalten entwickelt. XL [FK01] dient nicht nur der Komposition von Diensten, sondern unterstützt nach Art einer Programmiersprache auch die Programmierung von Web Services.

Das WWW-Konsortium hat mit [HO02] einen ersten Entwurf vorgelegt, in dem wichtige Anforderungen für Web-Service-Architekturen vorgestellt werden. Hier finden sich z. B. auch Ansätze für eine dynamische Dienstauswahl und die Notwendigkeit von Dienst-Metadaten. Microsoft's Passport [Pas] oder das Liberty Alliance Project [Pro] stellen erste Ansätze für die Nutzung von Kontextinformationen für Web Services dar, wobei zur Zeit nur die Identität eines Benutzers berücksichtigt wird.

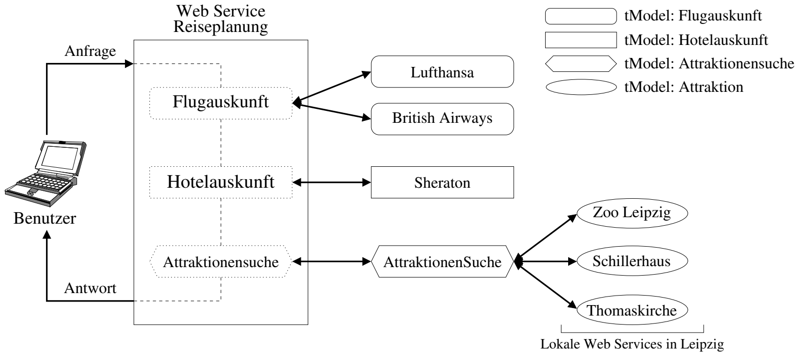

Abbildung 1: Anwendungsszenario: Marktplatz für Reiseagenturen

Der Rest dieses Beitrags ist wie folgt gegliedert: Abschnitt 2 beschreibt ein Anwendungsszenario, das als durchgängiges Beispiel verwendet wird. Im Anschluss präsentiert Abschnitt 3 einen Überblick über die Dienstplattform ServiceGlobe. Abschnitt 4 erläutert das Konzept der dynamischen Dienstauswahl und wie Vorgaben eingesetzt werden können, um die Dienstauswahl zu beeinflussen. Abschnitt 5 beschreibt den Einsatz von Kontext für Web Services im ServiceGlobe-System. Abschnitt 6 beschließt diesen Beitrag mit einer Zusammenfassung.

2 Anwendungsszenario

Im Rahmen dieses Beitrags verwenden wir einen Marktplatz für Reiseagenturen als durchgängiges Anwendungsszenario. Der Marktplatz offeriert Reiseagenturen, aber auch privaten Kunden, die Möglichkeit, komplette Reisen zu planen und zu buchen, inklusive Flug, Hotel, Sehenswürdigkeiten usw. Die Teilnehmer des Marktplatzes (Fluglinien, Hotelketten, etc.) bieten ihre Dienstleistungen als Web Services an. Die Integration der Teilnehmer erfolgt mit Hilfe eines lokalen UDDI-Verzeichnisses, in das die Web Services aller Teilnehmer eingetragen werden. Darin sind Dienste mit der gleichen Semantik einer gemeinsamen technischen Spezifikation zugeordnet, einem sogenannten *tModel* (Abkürzung für „technical model"). Das tModel als technische Spezifikation enthält eine Klassifikation der Funktionalität des Dienstes und eine formale Beschreibung seiner Schnittstelle. Weitere Informationen zu UDDI und tModels findet man z. B. in [RV02].

Der Marktplatz stellt eine Reihe von eigenen Diensten zur Verfügung, die Dienste unterschiedlicher Teilnehmer kombinieren, um damit eine erweiterte Funktionalität und weitergehende Informationen anbieten zu können. Ein Beispiel ist der Dienst *Reiseplanung*, der die automatische Planung einer Urlaubs- oder Dienstreise ermöglicht. Abbildung 1 zeigt eine exemplarische Ausführung des Dienstes für die Planung einer Reise zur BTW-Konferenz 2003 nach Leipzig. Die Ausführung des Dienstes gliedert sich in drei Phasen:[1] Nach dem Erhalt einer Anfrage werden zuerst mehrere Flugauskunftsdienste par-

[1]Das Beispiel betrachtet nur die Suche nach den gewünschten Informationen zu Flügen, Hotels usw. Details zu Buchungen sowie weitere Schritte, die ein vollständiger Dienst zur Reiseplanung durchführen müsste, wurden aus Gründen der Übersichtlichkeit weggelassen.

allel nach geeigneten Flügen befragt. Dabei wird ausgenutzt, dass in UDDI Dienste mit der gleichen Semantik einem gemeinsamen tModel zugeordnet sind. Im Beispiel sind alle Flugauskunftsdienste (*Lufthansa* und *British-Airways*) dem tModel `Flugauskunft` zugeordnet. Der Dienst *Reiseplanung* gibt nur an, dass Dienste des tModels `Flugauskunft` aufgerufen werden sollen; die Dienstplattform setzt dies mit Hilfe von UDDI in den Aufruf der tatsächlichen Dienste um. Dieser Vorgang wird im Folgenden als dynamische Dienstauswahl bezeichnet. In Abbildung 1 wird die dynamische Dienstauswahl dadurch verdeutlicht, dass das aufgerufene tModel und die tatsächlich aufgerufenen Dienste durch dieselben Symbole dargestellt werden. Beim Aufruf des tModels wird das Symbol mit einer gepunkteten Linie gezeichnet, bei den aufgerufenen Diensten mit einer durchgehenden Linie. In der zweiten Phase sucht der Dienst *Reiseplanung* nach passenden Übernachtungsmöglichkeiten. Dazu wird ein Aufruf des tModels `Hotelauskunft` ausgeführt, woraufhin die Dienstplattform passende Dienste auswählt und anspricht. In der letzten Phase sucht der Dienst nach Sehenswürdigkeiten vor Ort. Beim Aufruf des tModels `Attraktionensuche` wählt die Dienstplattform den Dienst *AttraktionenSuche* aus, der wie *Reiseplanung* vom Marktplatz zur Verfügung gestellt wird, und spricht ihn an, um von ihm die gewünschte Liste an Sehenswürdigkeiten zu erhalten. Dieser Dienst wiederum führt selbst einen tModel-Aufruf aus (tModel `Attraktion`, aus Gründen der Übersichtlichkeit nicht gezeigt) und spricht lokale Dienste für Sehenswürdigkeiten in und um Leipzig an, um nähere Informationen, z. B. aktuelle Programme oder Öffnungszeiten, einzuholen. In den folgenden Abschnitten werden gezielt verschiedene Punkte dieses Anwendungsszenarios angesprochen und konkretisiert.

3 Die Dienstplattform ServiceGlobe

ServiceGlobe ist eine verteilte, erweiterbare Dienstplattform für die flexible und robuste Ausführung von Web Services. Es ist vollständig in Java 2 implementiert und basiert auf den Standards XML, SOAP, UDDI und WSDL. In diesem Abschnitt werden die grundlegenden Komponenten dieser Plattform präsentiert. In ServiceGlobe werden zwei Arten von Diensten unterschieden: externe und interne Dienste (siehe Abbildung 2).

Externe Dienste sind Dienste wie sie zur Zeit im Internet eingesetzt werden. Sie werden nicht von ServiceGlobe selbst bereitgestellt. Diese Dienste sind normalerweise stationär, d. h. sie werden auf einem bestimmten, festen Rechner ausgeführt. Sie sind auf Basis unterschiedlicher Systeme entwickelt worden und haben verschiedene Aufrufschnittstellen. Um diese Dienste unabhängig von ihrer tatsächlichen Aufrufschnittstelle, z. B. HTML-Formularen oder RPC (Remote Procedure Call), integrieren zu können, verwendet ServiceGlobe *Adaptoren*. Diese setzen interne Anfragen auf die externe Schnittstelle um und umgekehrt. Damit ist auch der Zugriff auf beliebige Anwendungen, z. B. ERP-Systeme (Enterprise Resource Planning), möglich. Externe Dienste können somit genau wie interne Dienste benutzt werden.

Interne Dienste sind Dienste, die auf der Basis der ServiceGlobe-API implementiert wurden. Zu ihrer Entwicklung können auch spezialisierte Programmiersprachen verwendet werden, die dann in native ServiceGlobe-Dienste übersetzt werden müssen. Die Kommunikation interner ServiceGlobe-Dienste wird mit Hilfe von SOAP abgewickelt.

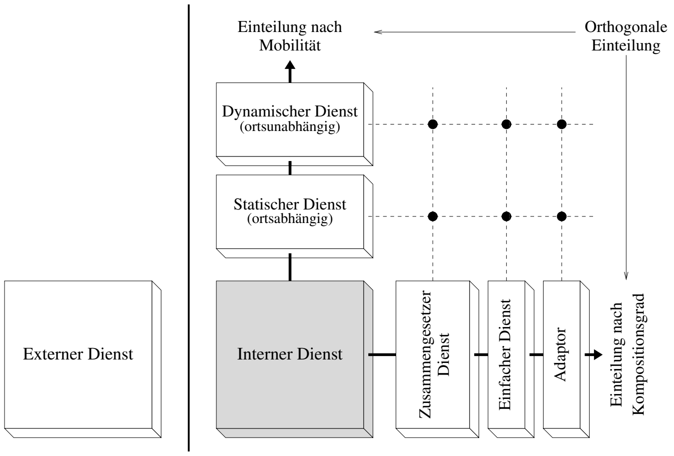

Abbildung 2: Einteilung von Diensten in ServiceGlobe

Dienste erhalten ein einzelnes XML-Dokument als Eingabeparameter und sie generieren ein einzelnes XML-Dokument als Ergebnis. Es werden zwei Arten von internen Diensten unterschieden: *dynamische* und *statische* Dienste. Statische Dienste sind *ortsabhängig*, d. h. sie können nicht dynamisch auf beliebigen ServiceGlobe-Rechnern ausgeführt werden. Gründe dafür können sein, dass sie Zugriff auf gewisse lokale Ressourcen, z. B. ein DBMS zum Speichern von Daten, oder bestimmte Rechte, z. B. Zugriff auf das Dateisystem, benötigten, die nur auf bestimmten, festen Rechnern verfügbar sind. Diese Einschränkungen verhindern die Ausführung von statischen Diensten auf beliebigen ServiceGlobe-Rechnern. Im Gegensatz dazu sind dynamische Dienste *ortsunabhängig*. Sie sind zustandslos, d. h. der interne Zustand eines solchen Dienstes wird nach der Abarbeitung einer Anfrage gelöscht. Sie benötigen keine speziellen Ressourcen oder Rechte und können deshalb auf jedem beliebigen ServiceGlobe-Rechner ausgeführt werden.

Eine orthogonale Einteilung von internen Diensten unterscheidet *Adaptoren*, *einfache Dienste* und *zusammengesetzte Dienste*. Adaptoren wurden bereits definiert. Einfache Dienste sind interne Dienste, die keinen anderen Dienst benutzen. Zusammengesetzte Dienste sind höherwertige Dienste, die aus anderen Diensten zusammengebaut werden. Sie basieren sozusagen auf diesen anderen Diensten, weswegen diese auch als *Basisdienste* bezeichnet werden. Natürlich kann auch ein zusammengesetzter Dienst wiederum als Basisdienst eines anderen zusammengesetzten Dienstes verwendet werden.

Interne Dienste werden auf *Service-Hosts* ausgeführt. Dabei handelt es sich um Rechner, auf denen die ServiceGlobe-Laufzeitumgebung läuft. Die internen Dienste von ServiceGlobe sind mobiler Code. Ihr ausführbarer Code wird, wenn notwendig, von sogenannten *Code-Repositories* auf einen Service-Host geladen. Ein UDDI-Verzeichnis wird verwendet, um Code-Repositories zu finden, von denen ein bestimmter Dienst geladen werden kann. Service-Hosts bieten einen SOAP-Dienst namens *Dynamic Runtime Loading* an, um beliebige dynamische Dienste auszuführen. Dadurch ist die Menge an verfügbaren Diensten nicht fest, sondern sie kann zur Laufzeit von jedem Teilnehmer des

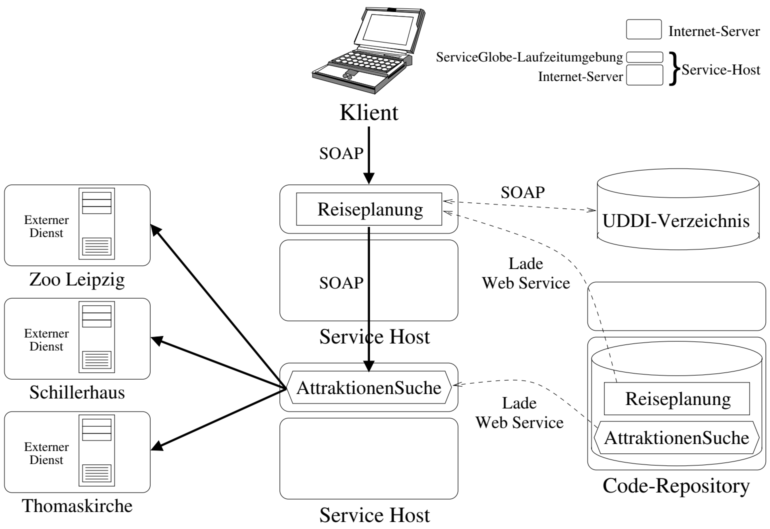

Abbildung 3: Architektur des ServiceGlobe-Systems

ServiceGlobe-Systems erweitert werden. Wenn interne Dienste die notwendigen Rechte besitzen, können sie die Ressourcen eines Service-Hosts benutzen, z. B. lokale Datenbanken. Diese Rechte sind Teil des Sicherheitssystems von ServiceGlobe, das auf [SBK01] basiert, und sie werden autonom von den Administratoren der Service-Hosts verwaltet. Das Sicherheitssystem schützt Service-Hosts auch vor Diensten, die als mobiler Code zur Laufzeit geladen werden und bei denen es sich potenziell um gefährliche Angreifer handeln kann.

Das Laden von Diensten zur Laufzeit ermöglicht die Verteilung von dynamischen Diensten zu beliebigen Service-Hosts. Dadurch ergeben sich für ServiceGlobe eine Reihe von Optimierungsmöglichkeiten: Mehrere Instanzen eines dynamischen Dienstes können aus Gründen der Lastbalancierung und Parallelisierung auf verschiedenen Rechnern ausgeführt werden. Dynamische Dienste können auf Service-Hosts instanziiert werden, die eine möglichst optimale Umgebung für die Ausführung besitzen, z. B. einen schnellen Prozessor, ausreichend Hauptspeicher oder eine schnelle Netzwerkanbindung. Außerdem trägt die Verteilung von dynamischen Diensten zur einer robusten Ausführung bei, da nicht verfügbare Service-Hosts dynamisch durch verfügbare ersetzt werden können.

Abbildung 3 zeigt die wesentlichen Komponenten des ServiceGlobe-Systems und wie sie miteinander interagieren (basierend auf dem Anwendungsszenario aus Abschnitt 2): *Reiseplanung* und *AttraktionenSuche* sind beides dynamische Dienste des Marktplatzes. Die Dienste *Zoo Leipzig*, *Schillerhaus* und *Thomaskirche* sind externe Dienste (Adaptoren wurden in der Abbildung weggelassen). Zuerst sendet der Klient eine SOAP-Anfrage an einen Service-Host des Marktplatzes, den Dienst *Reiseplanung* auszuführen. Der Dienst wird vom Code-Repository geladen (falls er nicht bereits gepuffert wird) und auf dem Service-Host instanziiert. *Reiseplanung* greift während seiner Ausführung auf den dynamischen Dienst *AttraktionenSuche* zu. Mit Hilfe von UDDI wird zuerst ein passender

Service-Host gesucht – im Beispiel ein Service-Host in oder um Leipzig – und der Dienst anschließend im Auftrag des Dienstes *Reiseplanung* auf diesem instanziiert. Der Dienst *AttraktionenSuche* benutzt die drei externen Dienste, um die gewünschten Informationen über Sehenswürdigkeiten zu erhalten.

4 Dynamische Dienstauswahl

Zusammengesetzte Dienste rufen andere Dienste normalerweise auf, indem sie die URL oder den Zugriffspunkt des aufzurufenden Dienstes angeben. Im Gegensatz dazu wird bei der dynamischen Dienstauswahl nur eine technische Spezifikation des Dienstes angegeben, der aufgerufen werden soll. Die Dienstplattform wählt daraufhin geeignete Dienste aus, gegebenenfalls unter Einsatz eines Dienstverzeichnisses wie UDDI, und ruft sie auf. Zusätzlich zur technischen Spezifikation können verschiedene Arten von Vorgaben übergeben werden, um die dynamische Dienstauswahl zu beeinflussen. Das Konzept der dynamischen Dienstauswahl hat drei wichtige Vorteile:

- In verteilten Systemen sind hohe Verfügbarkeit und Fehlertoleranz wichtige Ziele. Bei der Verwendung von zusammengesetzten Diensten kann es passieren, dass einige Basisdienste nicht erreichbar sind, z. B. aufgrund einer Netzwerkpartitionierung, nicht verfügbarer Rechner oder dem Absturz eines Basisdienstes.[2] Aus diesem Grund ist die Verwendung von festen Zugriffspunkten in zusammengesetzten Diensten nicht wünschenswert.
- Die Verwendung von Vorgaben zur Beeinflussung der dynamischen Dienstauswahl hilft bei der Entwicklung von flexiblen Web Services. Wie weiter unten gezeigt wird, ist es nicht notwendig bestimmte Eigenschaften der aufzurufenden Dienste in den zusammengesetzten Dienst selbst zu codieren. Die Eigenschaften werden stattdessen als (deklarative) Vorgaben spezifiziert und dem Dienst zur Laufzeit übergeben. Neu entwickelte Vorgaben können verwendet werden, ohne den Dienst ändern oder neu compilieren zu müssen.
- Dynamische Dienste werden zur Laufzeit instanziiert. Zusammen mit der dynamischen Dienstauswahl ergeben sich eine Vielzahl von Optimierungsmöglichkeiten. Zum Beispiel kann mit Vorgaben festgelegt werden, dass dynamische Dienste im lokalen LAN instanziiert werden müssen oder innerhalb einer festgelegten Menge von Service-Hosts.

Im Folgenden wird die dynamische Dienstauswahl genauer erklärt. Dabei wird UDDI als Dienstverzeichnis verwendet, vor allem weil es sich dabei um den De-Facto-Standard für diese Art von Verzeichnis handelt und weil es die benötigte Funktionalität besitzt.

In UDDI wird jeder Dienst einem tModel zugeordnet.[3] Das tModel als technische Spezifikation enthält eine Klassifikation der Funktionalität des Dienstes und eine formale Be-

[2] Ähnliche Probleme, wenn auch im Zusammenhang mit Web-Scripting-Sprachen, wurden in [CD99a] untersucht.

[3] Tatsächlich kann ein Dienst sogar mehreren tModels zugeordnet werden. Der Aufruf mehrerer tModels unterscheidet sich aber konzeptuell nicht wesentlich vom Aufruf eines tModels und wird deshalb im Folgenden nicht weiter betrachtet.

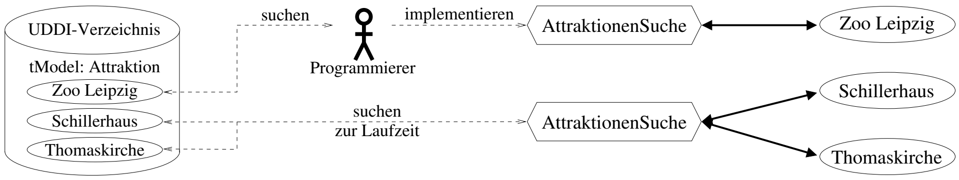

Abbildung 4: Ein Beispiel für dynamische Dienstauswahl

schreibung seiner Schnittstelle. Ein Dienst ist somit eine *Implementierung* oder *Instanz* seines tModels. Statt also in einem Web Service explizit einen Zugriffspunkt anzugeben, wird das tModel angegeben oder „aufgerufen". Dadurch legt man die Funktionalität des Dienstes fest, der aufgerufen werden soll, anstatt eines tatsächlichen Dienstes. Abbildung 4 zeigt ein Beispiel: Dem tModel `Attraktion` sind drei Dienste zugeordnet: *Zoo Leipzig*, *Schillerhaus* und *Thomaskirche*. Angenommen, ein Programmierer will einen neuen Dienst namens *AttraktionenSuche* implementieren, der einen dem tModel `Attraktion` zugeordneten Dienst aufrufen soll. Normalerweise wird der Programmierer im UDDI-Verzeichnis nach einem passenden Dienst suchen, z. B. *Zoo Leipzig*, und dessen Zugriffspunkt im neuen Dienst verwenden. Mit dynamischer Dienstauswahl wird der Programmierer hingegen den unteren der beiden *AttraktionenSuche*-Dienste implementieren. Dieser wird keinen fest codierten Zugriffspunkt enthalten, sondern einen Aufruf des tModels `Attraktion`. Zur Laufzeit wird die Dienstplattform im Auftrag des Dienstes das UDDI-Verzeichnis nach einem geeigneten Dienst durchsuchen und ihn aufrufen (oder sogar mehrere geeignete Dienste, wie in Abbildung 4 gezeigt).

4.1 Vorgaben

Die Vorgaben zur Beeinflussung der dynamischen Dienstauswahl für den Aufruf eines tModels können auf zwei Arten übergeben werden: innerhalb des Kontextes des Dienstes oder direkt beim Aufruf. Auf die erste Art wird in Abschnitt 5 genauer eingegangen. Werden Vorgaben auf die zweite Art übergeben bedeutet dies, dass der Programmierer diese Vorgaben bei der Entwicklung des Dienstes festgelegt hat und dass sie im Code des Dienstes gespeichert sind. Sie können nachträglich nicht mehr geändert werden und sind für jede Ausführung des Dienstes gleich. Eine Änderung ist nur durch eine erneute Compilierung des Dienstes möglich.

4.1.1 Präferenzen und Einschränkungen

Es gibt zwei unterschiedliche Arten von Vorgaben: *Präferenzen* und *Einschränkungen*.[4] Einschränkungen müssen erfüllt werden, wohingegen Präferenzen erfüllt werden sollten. Präferenzen sind also Soll-Vorgaben, d. h. die Dienstplattform wird bei der dynamischen Dienstauswahl als erstes solche Dienste aufrufen, die die Präferenzen erfüllen. Sollte es zu wenig passende Dienste geben, dürfen auch alternative Dienste aufgerufen werden, die die Präferenzen nicht erfüllen. Einschränkungen hingegen sind Muss-Vorgaben, d. h. ein

[4]Eine ähnliche Unterscheidung bei Bedingungen in SQL-Anfragen in hard constraints und soft constraints findet sich in [Kie02].

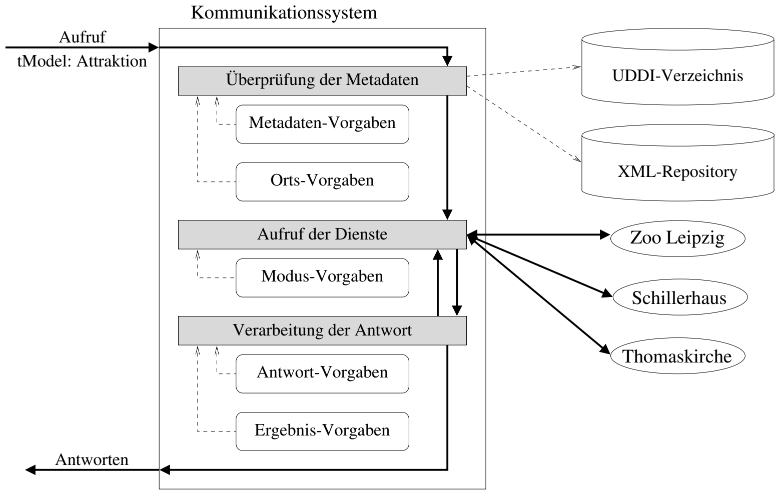

Abbildung 5: Ablauf der dynamischen Dienstauswahl

aufgerufener Dienst muss diese Vorgaben erfüllen. Sind zu wenig passende Dienste vorhanden, können keine alternativen Dienste aufgerufen werden.

4.1.2 Vorgabetypen

Neben der Einteilung in Präferenzen und Einschränkung werden orthogonal dazu fünf verschiedene Typen von Vorgaben unterschieden: Metadaten-Vorgaben, Orts-Vorgaben, Modus-Vorgaben, Antwort-Vorgaben und Ergebnis-Vorgaben. Jeder Vorgabentypen beeinflusst eine bestimmte Phase der dynamischen Dienstauswahl (siehe Abbildung 5). Die Vorgaben eines Typs können wiederum in Präferenzen und Einschränkungen unterschieden werden; bei Modus- und Ergebnis-Vorgaben sind Präferenzen allerdings nicht sinnvoll. Eine Ausnahme bei den Vorgabetypen nehmen die Orts-Vorgaben ein. Sie beeinflussen nicht nur die Auswahl und den Aufruf von Diensten, sondern sie können auch die Auswahl der Service-Hosts beeinflussen, auf denen dynamische Dienste gestartet werden.

Metadaten-Vorgaben Metadaten-Vorgaben (Metadata Constraints) werden vor dem Aufruf von Diensten als Filter auf alle Dienste angewendet, die von UDDI zurückgegeben werden, wenn eine Dienstplattform nach allen zu einem tModel passenden Diensten fragt (wie in Abbildung 5 dargestellt). Metadaten-Vorgaben sind im Grunde XPath-Anfragen [CD99b], die auf die Metadaten eines Dienstes angewendet werden. Ist die Anfrage erfolgreich, kann der Dienst weiterverwendet werden, ansonsten wird er verworfen. Die Metadaten eines einzelnen Dienstes bestehen zum einen aus den Informationen, die in UDDI über ihn zur Verfügung stehen. Das sind neben den bindingTemplate-, businessService- und businessEntity-Einträgen, die auf den Dienst verweisen, auch die tModel-Einträge, die der Dienst referenziert. Dazu kommen zusätzliche Metadaten, die in anderen Metadaten-Repositories gespeichert sind, z. B. [KKKK01, KKKK02b, KKKK02a]. Diese zusätzlichen Metadaten enthalten Informationen, die nicht in UDDI zu finden sind. Beispiele sind die Kosten für den Aufruf des Dienstes oder das vom Dienst unterstützte Transaktionsprotokoll. Die Metadaten über einen Dienst werden in einem virtuellen XML-Dokument zusammengefasst, auf das die XPath-Anfragen der Metadaten-Vorgaben angewendet werden. Die folgende Metadaten-Präferenz kann beispielsweise verwendet werden, um bevorzugt Dienste der Hotelkette Sheraton auszuwählen:

```
<metadataPreference>
  /businessEntity/name="Sheraton"
</metadataPreference>
```

Genaugenommen werden damit Dienste mit einem businessEntity-Eintrag bevorzugt, dessen Unterelement `name` den Wert `Sheraton` besitzt. Andere Dienste werden nur im Notfall ausgewählt. Die folgende Einschränkung legt fest, dass die Dienstplattform nur kostenlose Dienste auswählen darf (und keinen einzigen kostenpflichtigen Dienst):

```
<metadataCondition>
  /ServiceMetadata/CostsPerCall="0"
</metadataCondition>
```

Orts-Vorgaben Orts-Vorgaben (Location Constraints) werden dazu verwendet, Vorgaben bezüglich des Ausführungsorts eines Dienstes, d. h. des Service-Hosts, zu machen, sowohl für statische als auch für dynamische Dienste. Bei statischen Diensten ermöglicht eine Orts-Vorgabe die Auswahl anhand des Dienststandorts. Bei dynamischen Diensten bewirkt sie, dass diese (im Falle einer Präferenz) bevorzugt oder (im Falle einer Einschränkung) strikt an dem angegebenen Standort instanziiert und ausgeführt werden. Die Informationen über den Standort von Diensten und Service-Hosts erhält die Dienstplattform aus den entsprechenden Metadaten aus dem UDDI-Verzeichnis (siehe Abbildung 5). Der gewünschte Standort kann auf zwei Arten festgelegt werden: basierend auf der Netzwerkadresse des Rechners, auf dem der Dienst läuft oder laufen soll, oder geographisch. Die Unterscheidung in Orts-Vorgaben für statische Dienste und Orts-Vorgaben für dynamische Dienste hat den Vorteil, dass beim Aufruf eines tModels statische und dynamische Dienste unterschiedlich behandelt werden können. Ansonsten wäre es nicht möglich den Standort von statischen Diensten unabhängig vom Standort der Service-Hosts zu wählen, auf denen die dynamischen Dienste ausgeführt werden sollen.

Die folgende Orts-Vorgabe legt fest, dass ausgewählte Dienste in einem Umkreis von 50 Kilometern um Leipzig (ISO-3166-Kürzel DE-SN-LEJ) liegen müssen bzw. dort instanziiert werden sollen. Das Attribut `serviceType` legt fest, dass es sich um eine Orts-Vorgabe sowohl für statische als auch dynamische Dienste handelt. Alternative Werte wären `Static` für statische Dienste und `Dynamic` für dynamische Dienste.

```
<locationCondition addressType="Geographical" serviceType="All">
  <center>DE-SN-LEJ</center>
  <maxDistance>50km</maxDistance>
</locationCondition>
```

Modus-Vorgaben Eine Dienstplattform ist nicht darauf beschränkt beim Aufruf eines tModels nur eine Instanz des tModels auszuwählen; es können auch mehrere Instanzen ausgewählt werden. Somit wird der Aufruf eines tModels ersetzt durch den Aufruf eines oder mehrerer Dienste. Modus-Vorgaben (Mode Constraints) ermöglichen es, die Anzahl der Dienste festzulegen, die bei einem tModel-Aufruf angesprochen werden sollen. Wie Abbildung 5 zeigt, sind Modus-Vorgaben entscheidend für den Aufruf der Dienste und die Verarbeitung ihrer Antworten. Aufgrund der Modus-Vorgaben wird entschieden, ob nach der Verarbeitung einer Antwort ein alternativer Dienst aufgerufen werden soll, ob alle Antworten als Ergebnis zurückgegeben werden sollen oder ob auf weitere Antworten gewartet werden muss. Die folgenden Modi werden beim Aufruf eines tModels unterschieden: One-Modus, Some-Modus und All-Modus.[5]

Im *One-Modus* wird nur eine Instanz des tModels aufgerufen. Im Falle eines Fehlers, z. B. wenn der Dienst zeitweilig nicht verfügbar ist, wird ein alternativer Dienst aufgerufen. Im *Some-Modus* wird eine Teilmenge der von UDDI zurückgelieferten Dienste aufgerufen. Die Größe der Menge, d. h. die Anzahl der aufzurufenden Dienste, wird als Parameter angegeben (als Prozentangabe oder absolut). Dienste, die nicht antworten, werden durch alternative Dienste ersetzt, solange bis die geforderte Anzahl von Diensten erfolgreich geantwortet hat oder keine alternativen Dienste mehr verfügbar sind. Im *All-Modus* werden alle zurückgelieferten Dienste aufgerufen. Bei Fehlern können natürlich keine alternativen Dienste aufgerufen werden.

Das folgende Beispiel zeigt eine Some-Modus-Vorgabe, die festlegt, dass fünf Prozent der verfügbaren Dienste aufgerufen werden sollen:

```
<modeCondition modeType="Some" number="5" numberType="Percentage" />
```

Antwort-Vorgaben Antwort-Vorgaben (Reply Constraints) werden ausgewertet, wenn eine Antwort von einem aufgerufenen Dienst eintrifft (siehe Abbildung 5). Eine Antwort wird nur an den aufrufenden Dienst zurückgegeben, wenn sie alle für sie relevanten Antwort-Vorgaben erfüllt, ansonsten wird sie verworfen. Es gibt zwei Arten von Antwort-Vorgaben: Eigenschafts-Vorgaben und Selektions-Vorgaben.

Eine *Eigenschafts-Vorgabe* (Property Constraint) erlaubt die Auswahl von Antworten basierend auf einer oder mehrerer, fest definierter Eigenschaften der Antwort. Die Eigenschaften müssen entweder von der Dienstplattform oder aber vom aufgerufenen Dienst zur Verfügung gestellt werden. Ein Dienst erreicht dies dadurch, dass er entsprechende XML-Elemente in seine Antwort einfügt. ServiceGlobe unterstützt folgende Eigenschaften: Verschlüsselung, Signatur und Alter der Daten. Mit den ersten beiden kann überprüft werden, ob eine Antwort verschlüsselt bzw. signiert ist und von wem sie signiert ist. Die letzte Eigenschaft kann verwendet werden, um das Alter der gelieferten Daten zu überprüfen. Wichtig ist dies zum Beispiel, wenn die Antwort aus einem Cache stammt, man aber ausreichend aktuelle Daten haben möchte. Die folgende Vorgabe würde eine Dienstplattform dazu veranlassen, nur Antworten zurückzugeben, die vom angegebenen Subjekt signiert sind:

```
<propertyCondition>
  <signature>
    <signatureDN>CN=Customer, O=Universität Passau, C=DE</signatureDN>
  </signature>
</propertyCondition>
```

[5]Die Modi entsprechen den Kommunikationsformen Unicast, Multicast und Broadcast in Netzwerken

Selektions-Vorgaben (Selection Constraints) erlauben die Anwendung eines beliebigen XPath-Ausdrucks auf die Antwort eines Dienstes, inklusive ihrer SOAP-spezifischen Teile, wie z. B. den SOAP-Header. Verschlüsselte Teile der Nachricht können natürlich nicht abgefragt werden. Selektions-Vorgaben, die auf verschlüsselte Teile Bezug nehmen, gelten als nicht erfüllt, was bei Einschränkungen dazu führt, dass die Antwort nicht an den aufrufenden Dienst zurückgegeben wird. Die folgende Vorgabe stellt beispielsweise sicher, dass alle Antworten im SOAP-Body ein Unterelement `Hotel` mit einem Attribut `class` und dem Wert `FirstClass` enthalten:

```
<selectionCondition>
  /Envelope/Body//Hotel/@class="FirstClass"
</selectionCondition>
```

Ergebnis-Vorgaben Der fünfte Vorgabentyp sind Ergebnis-Vorgaben (Result Constraints). Anders als Antwort-Vorgaben betreffen sie keine einzelne Antwort, sondern alle bisher eingetroffenen Antworten. Es gibt zwei Arten von Ergebnis-Vorgaben: Timeout-Vorgaben und FirstN-Vorgaben.

Mit einer *Timeout-Vorgabe* (Timeout Constraint) kann eine maximale Wartezeit für Antworten von aufgerufenen Diensten festgelegt werden. Nach Ablauf dieser Zeitspanne werden alle noch ausstehenden Dienste abgebrochen und die bis zu diesem Zeitpunkt erhaltenen Antworten an den aufrufenden Dienst zurückgegeben. Diese müssen natürlich alle anderen relevanten Vorgaben erfüllen, z. B. Antwort-Vorgaben. Folgende Vorgabe ist ein einfaches Beispiel für eine Timeout-Vorgabe:

```
<timeoutCondition value="100" valueUnit="Seconds"/>
```

FirstN-Angaben ermöglichen es, den Aufruf eines tModels nach dem Erhalt einer vorher festgelegten Anzahl von Antworten zu beenden. Der aufrufende Dienst erhält nur diese Antworten zurück. Aufrufe an Dienste, die bis zu diesem Zeitpunkt noch nicht geantwortet haben, werden abgebrochen. Die Anzahl an abzuwartenden Antworten kann entweder absolut (Attribut `numberType="Absolute"`) oder prozentual (`numberType="Percentage"`), abhängig von der Anzahl der initial aufgerufenen Dienste, festgelegt werden. Die folgende Vorgabe würde den Aufruf eines tModels nach dem Eintreffen von zehn Prozent der Antworten beenden:

```
<firstNCondition number="10" numberType="Percentage" />
```

4.2 Kombination von Vorgaben

Um die dynamische Dienstauswahl auf komplexere Weise zu beeinflussen, ist die Kombination mehrerer Vorgaben notwendig. Will man z. B. die Auswahl so beeinflussen, dass nur Dienste einer vorgegebenen Firma aufgerufen werden (Metadaten-Vorgabe) und außerdem maximal 10 Sekunden auf Antworten gewartet wird (Timeout-Vorgabe), benötigt man zwei konjunktiv verknüpfte Vorgaben (AND-Operator).

In ServiceGlobe können Vorgaben mit Hilfe der Operatoren AND und OR kombiniert werden.[6] Für jeden der Operatoren gibt es ein entsprechendes XML-Element: `andGroup`

[6] An der Integration des Operators NOT wird zur Zeit gearbeitet.

```
<orGroup>
  <andGroup>
    <metadataCondition>
      /ServiceMetadata/ServiceType="Dynamic"
    </metadataCondition>
    <locationPreference serviceType="Dynamic"
                        addressType="Geographical">
      <pattern>DE-SN-LEJ</pattern>
    </locationPreference>
  </andGroup>
  <locationPreference serviceType="All" addressType="Geographical">
    <pattern>DE-*-*</pattern>
  </locationPreference>
</orGroup>
```

Abbildung 6: Kombination von Vorgaben

für den AND-Operator und `orGroup` für den OR-Operator. Jeder Operator ermöglicht die Kombination von Vorgaben, sogenannten *atomaren Termen*, und bereits kombinierten Vorgaben, auch *kombinierte Terme* genannt, zu einem neuen (kombinierten) Term. Das bedeutet, dass die Operatoren ineinander geschachtelt werden können. Abbildung 6 zeigt ein Beispiel: die OR-Kombination von zwei Termen. Der erste Term sagt aus, dass nur dynamische Dienste ausgewählt und vorzugsweise auf Service-Hosts in Leipzig instanziiert werden sollen.[7] Der zweite Term legt fest, dass beliebige Dienste in Deutschland bevorzugt ausgewählt werden sollen.

Durch die Kombination von Vorgaben kann es dazu kommen, dass Konflikte entstehen, die die Erfüllung aller Vorgaben verhindern, wie folgendes Beispiel zeigt:

```
<orGroup>
  <metadataCondition>
    /businessEntity/name="Sheraton"
  </metadataCondition>
  <timeoutCondition value="100" valueUnit="Seconds"/>
</orGroup>
```

Bei der Auswertung dieser kombinierten Vorgabe hat die Dienstplattform zwei Möglichkeiten: Sie kann zum einen *nur Dienste des Unternehmens Sheraton* ansprechen und warten bis alle geantwortet haben. In diesem Fall erfüllt sie nur die erste Vorgabe. Oder sie kann zum anderen *alle Dienste* des aufgerufenen tModels ansprechen. Im Falle eines Timeouts muss die Dienstplattform dann entscheiden, ob sie alle bisher erhaltenen Antworten zurückgibt und damit nur die zweite Vorgabe erfüllt oder ob sie den Timeout ignoriert, auf alle Antworten der Sheraton-Dienste wartet und damit nur die erste Vorgabe erfüllt. In diesem Fall hat sie allerdings initial unnötig viele Dienste angesprochen. Im Allgemeinen kann die Dienstplattform also nicht beide Vorgaben gleichzeitig erfüllen.

4.3 Vorgabenkonflikte und Konfliktauflösung

Die Kombination von Vorgaben kann zu verschiedenen Konflikten führen. Ein *Widerspruch* tritt auf, wenn zwei oder mehr Vorgaben angegeben werden, die nicht alle zu-

[7] Mit dem Unterelement `pattern` der Orts-Vorgabe wird ein Muster definiert, zu dem die geographischen Informationen der Service-Hosts passen müssen. Das Zeichen `*` agiert als Wildcard.

```
<dssConstraints>
  <locationPreference srcKey="1" addressType="Geographical">
    <pattern>DE-*-*</pattern>
  </locationPreference>
</dssConstraints>
<dssConstraintsSources>
  <source srcKey="1">
    <URI>http://tempuri.org/sg/constraints</URI>
    <priority>2</priority>
  </source>
</dssConstraintsSources>
```

Abbildung 7: Vorgaben mit Prioritäten und Quellenangaben

gleich erfüllt werden können. Ein Beispiel dafür sind zwei Modus-Einschränkungen, die unterschiedliche Aufrufmodi fordern. Ein weiteres Beispiel sind zwei Metadaten-Einschränkungen, bei denen die eine die Verwendung von dynamischen Diensten fordert, während die andere statische Dienste verlangt. Widersprüche können nur zwischen Einschränkungen auftreten, da Präferenzen im Falle eines Widerspruchs mit einer anderen Vorgabe immer fallen gelassen werden können. Die zweite Art von Konflikt wurde bereits im vorhergehenden Abschnitt erklärt. Bei ihr ist es nicht immer möglich alle Vorgaben zu erfüllen. Die Dienstplattform muss in diesem Fall entscheiden, welche Vorgaben erfüllt werden sollen und welche nicht. Hierbei handelt es sich um eine Frage der Optimierung: Einerseits soll die Dienstplattform möglichst nur so viele Dienste ansprechen wie unbedingt nötig, andererseits sollen so viele Vorgaben erfüllt werden wie möglich.

Die Ursache für derartige Konflikte können Fehler bei der Zusammenstellung der Vorgaben sein. Hauptsächlich werden solche Konflikte aber auftreten, wenn für einen tModel-Aufruf Vorgaben aus unterschiedlichen Quellen verwendet werden, z. B. vom Dienst selbst oder aus seinem Kontext. Eine Dienstplattform muss Konflikte auflösen oder die Abarbeitung der Anfrage abbrechen. Der Rest dieses Abschnitts erläutert, wie eine Dienstplattform Konflikte auflösen und das Abbrechen von Anfragen vermeiden kann.

Die Auflösung von Konflikten basiert auf Prioritäten. Jedem Term wird eine Priorität von 0 (Minimum) bis ∞ (Maximum) zugeordnet. Eine implizite Priorisierung ist durch die Reihenfolge der Terme in ihrer XML-Repräsentation gegeben. Je später ein Term im XML-Dokument definiert wird, desto geringer ist seine Priorität. Eine explizite Priorisierung kann auf zwei Arten durchgeführt werden: Zum einen kann jedem Term mit dem Attribut `priority` eine explizite Priorität zugeordnet werden. Zum anderen kann mit dem Attribut `srcKey` eine Referenz auf die Quelle eines Terms angegeben werden. Den Quellen – identifiziert durch eine URI – werden in einem gesonderten Abschnitt des Kontextes Prioritäten zugeordnet. Diese werden dann auf die jeweiligen Terme angewendet. Die Angabe einer Quelle wirkt sich – bei kombinierten Termen – rekursiv auf alle enthaltenen Teilterme aus, solange bis gegebenenfalls eine weitere Quellenangabe folgt. Abbildung 7 zeigt ein Beispiel. Besitzen Terme die gleiche explizite Priorität, entscheidet ihre implizite Priorität, d. h. die Reihenfolge, in der sie angegeben wurden.

4.4 Auswertung von Vorgaben

Der folgende Abschnitt beschreibt die Vorgehensweise von ServiceGlobe bei der Auswertung von Vorgaben beim Aufruf eines tModels. Sie besteht aus zwei Schritten: Zuerst werden alle für den tModel-Aufruf relevanten Vorgaben zusammengefasst und Konflikte aufgelöst. Anschließend werden den Vorgaben entsprechende Dienste ausgewählt und aufgerufen und ihre Antworten verarbeitet.

Vorverarbeitung der Vorgaben

Zuerst werden alle für einen tModel-Aufruf relevanten Vorgaben aus den verschiedenen Quellen konjunktiv zusammengefasst. Die resultierende, kombinierte Vorgabe wird in ihre disjunktive Normalform umgewandelt. Das Ergebnis ist eine kombinierte Vorgabe der folgenden Form:

$$(c_{11} \wedge \ldots \wedge c_{1i_1}) \vee (c_{21} \wedge \ldots \wedge c_{2i_2}) \vee \ldots \vee (c_{n1} \wedge \ldots \wedge c_{ni_n})$$

mit $\forall k \in \{1, \ldots, n\}, j \in \{1, \ldots, i_k\} : c_{kj} \in \textit{Vorgaben}$, d. h. die c_{kj} sind atomare Terme. Die Vorgaben in jedem *AND-Term*, d. h. einem Term, der nur $\wedge$-Operatoren enthält, werden anhand ihres Auswertungszeitpunkts sortiert. Die Reihenfolge lautet: Metadaten-Vorgaben, Orts-Vorgaben, Modus-Vorgaben, Antwort-Vorgaben und Ergebnis-Vorgaben. Die AND-Terme der obigen kombinierten Vorgabe seien bereits so geordnet. Damit gilt:

$$\begin{aligned}
\forall k \in \{1, \ldots, n\} \quad : \quad & c_{k1}, \ldots, c_{km_k} \in \textit{MetadatenVorgaben} \\
& c_{km_k+1}, \ldots, c_{kl_k} \in \textit{OrtsVorgaben} \\
& c_{kl_k+1}, \ldots, c_{kp_k} \in \textit{ModusVorgaben} \\
& c_{kp_k+1}, \ldots, c_{kr_k} \in \textit{AntwortVorgaben} \\
& c_{kr_k+1}, \ldots, c_{ki_k} \in \textit{ErgebnisVorgaben}
\end{aligned}$$

Die Menge der Vorgaben eines Typs für einen einzelnen AND-Term kann natürlich leer sein. Jeder AND-Term darf maximal eine Modus-Vorgabe enthalten, d. h. $\forall k \in \{1, \ldots, n\} : l_k + 1 = p_k$, ansonsten existiert ein Konflikt. Die Auflösung von Konflikten geschieht dadurch, dass aus allen miteinander in Konflikt stehenden Vorgaben diejenige mit der höchsten Priorität ausgewählt wird. Alle anderen Vorgaben werden entfernt. Damit ein Dienst die kombinierte Vorgabe erfüllt, muss er mindestens einen der AND-Terme erfüllen, d. h. die Metadaten des Dienstes, der Modus seines Aufrufs und seine Antwort müssen den Vorgaben des AND-Terms entsprechen.

Aufruf der Web Services

Im nächsten Schritt werden von UDDI Informationen über alle Dienste angefordert, die das aufgerufene tModel implementieren. Diese Dienste werden in einer Liste geordnet. Im Folgenden bewirken Einschränkungen, dass Dienste, die eine Einschränkung nicht erfüllen, verworfen werden. Präferenzen führen dazu, dass die Dienste anhand der erfüllten Präferenzen und deren Prioritäten geordnet werden. Je mehr Präferenzen ein Dienst erfüllt und je höher die Prioritäten der Präferenzen sind, desto weiter vorne steht er in der Liste.

Anschließend werden zuerst Metadaten-Vorgaben angewendet, dann Orts-Vorgaben. Bei Orts-Vorgaben für dynamische Dienste werden zuerst alle Service-Hosts vom UDDI-Verzeichnis angefordert und entsprechend den Orts-Vorgaben gefiltert. Jeder dynamische Dienst wird dann einem Service-Host aus der Liste zugeteilt. Anschließend werden die Dienste aufgerufen, wobei eine Modus-Vorgabe bestimmt, wieviele Dienste initial aufgerufen werden und ob gegebenenfalls alternative Dienste angesprochen werden müssen. Die Antworten der Dienste werden mit Hilfe der Antwort-Vorgaben gefiltert und wiederum führen Präferenzen zu einer Sortierung der Antworten. Sind alle Ergebnis-Vorgaben erfüllt, werden die bisher gesammelten Antworten unter Berücksichtigung ihrer Sortierung und der Ergebnis-Vorgaben zurückgegeben.

Bei der Auswertung von kombinierten Vorgaben mit mehreren AND-Termen wird momentan der AND-Term mit der höchsten Priorität[8] ausgewählt. In Zukunft werden im Rahmen einer Optimierungskomponente verschiedene Optimierungskriterien berücksichtigt werden und die Auswahl wird darauf basieren.

5 Web-Service-Kontext

Kontext bzw. die in ihm enthaltenen Informationen sind ein effizientes Mittel, um personalisierbare und flexible Web Services zu entwickeln. Gerade die Integration von Vorgaben in den Kontext ist dabei hilfreich. Der folgende Abschnitt beschreibt, wie Kontext in das ServiceGlobe-System integriert ist und welche Informationen er enthalten kann.

5.1 Integration in Dienstplattformen

Kontextinformationen sind nicht Teil der direkten Eingabedaten eines Dienstes; sie stellen zusätzliche Informationen dar, die nicht vorhanden sein müssen. Die Integration des Kontextes erfolgt transparent, d. h. Dienste müssen sich nicht aktiv darum kümmern Kontextinformationen zu verarbeiten, sie in ausgehende Nachrichten einzufügen oder aus eingehenden Nachrichten auszulesen. Dies wird von der Dienstplattform erledigt. Kontextinformationen können je nach Typ von der Dienstplattform oder vom aufgerufenen Dienst ausgewertet werden. Soweit möglich sollten sie von der Dienstplattform automatisch ausgewertet werden, da Dienste nur Kontextinformationen berücksichtigen können, die bei ihrer Implementierung bekannt waren und integriert wurden. Kontextinformationen, die weder von der Dienstplattform noch vom aufgerufenen Dienst unterstützt werden, werden ignoriert. Insbesondere können natürlich existierende Dienste Kontext ohne weiteres ignorieren.

Aus Gründen der Transparenz werden Kontextinformationen beim Aufruf eines Dienstes nicht als Teil des SOAP-Bodies der Nachricht übertragen – dieser enthält die direkten Eingabedaten des Dienstes – sondern als Teil des SOAP-Headers. Dienstplattformen, die nicht mit Kontextinformationen umgehen können, können diesen Teil des Header ignorieren.

[8] Die Priorität eines AND-Terms ist (momentan) das Maximum der Prioritäten seiner Vorgaben.

5.2 Arten von Kontextinformationen

Der folgende Abschnitt beschreibt die verschiedenen Arten von Informationen, die der Kontext eines Web Services in ServiceGlobe enthalten kann.

Kontakt- und Klientinformationen

Kontaktinformationen enthalten Daten über die Identität des Benutzers, z. B. Name, Adresse, Email-Adresse, soweit der Benutzer bereit ist diese anzugeben. Benutzer haben auch die Möglichkeit, Pseudonyme anzugeben oder anonym zu bleiben. Die Daten werden direkt aus dem Klienten entnommen, auf dem der Benutzer die initiale Anfrage gestellt hat. Außerdem können die Kontaktinformationen Daten über Personalisierungsdienste, z. B. das Liberty Alliance Project [Pro] oder Microsoft Passport [Pas], enthalten, die von den Web Services oder der Dienstplattform zur automatischen Identifikation des Benutzers verwendet werden können.

Klientinformationen enthalten Daten zur Hard-/Software des Klienten sowie seinen Standort. Der Standort umfasst nicht nur die geographische Angabe der Position des Benutzers sondern auch semantische Informationen darüber. Der lokalisierte Kontext kann für Optimierungen bei der Dienstausführung und zur Optimierung von UDDI-Anfragen verwendet werden. Informationen über den Klienten werden verwendet, um den Umfang der Antwort des Web Service anzupassen. Je nach Klient können z. B. Bilder entweder direkt oder als Verweis in die Antwort eingebunden werden. Verweise sind sinnvoll, wenn der Benutzer an einem PDA mit einer langsamen oder teuren Netzverbindung arbeitet. Auch die Größe der Bilder kann an den Klienten angepasst werden. Klientinformationen ermöglichen es zudem, in die Antwort die passenden Stylesheets einzubinden, die es dem Klienten ermöglichen, die XML-Antwort entsprechend dem Ausgabemedium zu formatieren, z. B. in HTML-Seiten oder WAP-Seiten.

Vorgaben für die dynamische Dienstauswahl

In Abschnitt 4.1 wurden verschiedene Vorgaben vorgestellt, die bei der dynamischen Dienstauswahl verwendet werden können. Die Angabe von Vorgaben beim Aufruf eines tModels erfordert, dass die Vorgaben im Code des Dienstes enthalten sind und deshalb bereits bei der Compilierung des Dienstes festgelegt wurden. Statt direkt beim Aufruf können Vorgaben auch in den Kontext eines Dienstes eingefügt werden. Beim Aufruf eines tModels bestimmt die Dienstplattform alle relevanten Vorgaben aus dem Kontext und verwendet diese – zusätzlich zu direkt angegebenen Vorgaben. Dadurch ergibt sich eine zusätzliche Flexibilität bei der dynamischen Dienstauswahl, wodurch eine automatische Anpassung der Dienste an die Wünsche der Benutzer, z. B. die Verwendung der aktuell billigsten Dienste, möglich wird. Durch die Einbindung von Vorgaben in den Kontext kann außerdem eine Optimierung der Ausführung der angesprochenen Dienste erreicht werden.

Ein Dienst wird im Allgemeinen mehr als ein tModel aufrufen. Folglich wird der Kontext eines Dienstes Vorgaben für mehr als einen Aufruf enthalten und für die Vorgaben muss deshalb festgelegt werden, für welchen Aufruf sie verwendet werden sollen. Aus diesem

```
<Context xmlns="http://sg.fmi.uni-passau.de/context">
  <dssConstraints>
    <metadataCondition tModel="Reiseplanung">
      /ServiceMetadata/CostsPerCall="0"
    </metadataCondition>
    <metadataCondition tModel="Attraktionensuche">
      /ServiceMetadata/ServiceType="Dynamic"
    </metadataCondition>
    <locationPreference tModel="Attraktionensuche" serviceType="Dynamic"
                        addressType="Geographical">
      <pattern>DE-SN-LEJ</pattern>
    </locationPreference>
    <locationCondition tModel="Attraktion" addressType="Geographical">
      <center>DE-SN-LEJ</center>
      <maxDistance>50km</maxDistance>
    </locationCondition>
    <propertyCondition tModel="Attraktion">
      <maxAgeOfData>1d</maxAgeOfData>
    </propertyCondition>
  </dssConstraints>
</Context>
```

Abbildung 8: Vorgaben im Kontext eines Dienstes

Grund kann bei jeder Vorgabe das tModel (mit dem Attribut `tModel`) bzw. der Schlüssel des tModels (Attribut `tModelKey`) angegeben werden, bei dessen Aufruf die Vorgabe verwendet werden soll. Wird nichts derartiges angegeben, gilt die Vorgabe für alle Aufrufe. Abbildung 8 zeigt ein Beispiel für Vorgaben mit Angabe des gültigen tModels. Auch AND- und OR-Kombinationen können mit den Attributen `tModel` bzw. `tModelKey` einem tModel-Aufruf zugeordnet werden. In einem solchen Fall sind alle in dem Term definierten Vorgaben dem angegebenen tModel-Aufruf zugeordnet. Es ist offensichtlich, dass eine Kombination von Vorgaben, die unterschiedlichen tModel-Aufrufen zugeordnet sind, nicht sinnvoll ist.

Vorgaben im Kontext eines Dienstes können aus mehreren unterschiedlichen Quellen stammen: vom Benutzer, von Dienstplattformen oder von Diensten in der Aufrufkette. Der Klient eines Benutzers hat die Möglichkeit, Vorgaben beim Abschicken einer Anfrage automatisch in den Kontext einzufügen. Diese Vorgaben kann der Benutzer direkt angegeben haben, der Klient kann sie im Laufe der Zeit angesammelt haben – ähnlich wie Bookmarks bei Web-Browsern – oder die Vorgaben werden vom Arbeitgeber des Benutzers vorgegeben. Eine Dienstplattform kann beim Aufruf eines lokalen Dienstes eine Menge von lokalen Standardvorgaben in den Kontext des aufgerufenen Dienstes einfügen. Schließlich kann auch ein Dienst beim Aufruf eines Basisdienstes Vorgaben in die ausgehende Nachricht einfügen. Der aufgerufene Dienst kann beim Aufruf von eigenen Basisdiensten diese Vorgaben weitergeben und gegebenenfalls auch selbst neue einfügen.

6 Zusammenfassung

Wir haben in diesem Beitrag Technologien präsentiert, die in ihrer Kombination die Entwicklung von personalisierbaren und flexiblen Web Services erleichtern. Die dynamische

Dienstauswahl ermöglicht die Auswahl und den Aufruf von Web Services zur Laufzeit, basierend auf einer technischen Spezifikation der gewünschten Web Services. Sie kann durch verschiedene Vorgaben beeinflusst werden, die direkt beim Aufruf der Dienste oder in den Kontexten der Dienste angegeben werden können. Der Einsatz von Kontexten ermöglicht es personalisierbare Web Services zu entwickeln, die mit ihren Benutzern in der von ihnen gewünschten Weise interagieren. Neben Vorgaben kann der Kontext eines Dienstes auch Informationen zum Benutzer oder der Klientumgebung enthalten.

Wir haben diese Technologien im Rahmen des ServiceGlobe-Systems präsentiert, einer Dienstplattform für die flexible und robuste Ausführung von Web Services. Neben den genannten Technologien unterstützt ServiceGlobe mobilen Code, wodurch eine Verteilung der Web Services zur Laufzeit hin zu beliebigen Service-Hosts im Netz möglich wird.

Literaturverzeichnis

[BBB+01] A. Banerji, C. Bartolini, D. Beringer, V. Chopella, K. Govindarajan, A. Karp, H. Kuno, M. Lemon, G. Pogossiants, S. Sharma, and S. Williams. Web Services Conversation Language (WSCL). `http://www.e-speak.hp.com/media/wscl_5_16_01.pdf`, 2001. Hewlett-Packard.

[BDSN02] B. Benatallah, M. Dumas, Q. Z. Sheng, and A. H. H. Ngu. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proc. of the 18th Intl. Conference on Data Engineering (ICDE)*, pages 297–308, 2002.

[BEK+00] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. `http://www.w3.org/TR/SOAP`, 2000. W3C Note.

[BPSMM00] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible Markup Language (XML) 1.0 (Second Edition). `http://www.w3.org/TR/REC-xml`, 2000. W3C Recommendation.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. `http://www.w3.org/TR/wsdl`, 2001. W3C Note.

[CD99a] L. Cardelli and R. Davies. Service Combinators for Web Computing. *IEEE Trans. Software Eng.*, 25(3):309–316, 1999.

[CD99b] J. Clark and S. DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath`, 1999. W3C Recommendation.

[FK01] D. Florescu and D. Kossmann. An XML Programming Language for Web Service Specification and Composition. *IEEE Data Engineering Bulletin*, 24(2):48–56, 2001.

[HO02] H. Haas and D. Orchard. Web Services Architecture Usage Scenarios. `http://www.w3.org/TR/ws-arch-scenarios`, 2002. W3C Working Draft.

[Kie02] W. Kießling. Foundations of Preferences in Database Systems. In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[KKKK01] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. Verteilte Metadatenverwaltung für die Anfragebearbeitung auf Internet-Datenquellen. In *Proc. GI Conf. on Database Systems for Office, Engineering, and Scientific Applications (BTW)*, pages 107–126, 2001.

[KKKK02a] M. Keidl, A. Kemper, D. Kossmann, and A. Kreutz. Verteilte Metadatenverwaltung und Anfragebearbeitung für Internet-Datenquellen. *Informatik Forschung und Entwicklung*, 17(3):123–134, 2002.

[KKKK02b] M. Keidl, A. Kreutz, A. Kemper, and D. Kossmann. A Publish & Subscribe Architecture for Distributed Metadata Management. In *Proc. of the 18th Intl. Conference on Data Engineering (ICDE)*, pages 309–320, 2002.

[KSK02] M. Keidl, S. Seltzsam, and A. Kemper. Flexible and Reliable Web Service Execution. In *Proc. of the 1st Workshop on Entwicklung von Anwendungen auf der Basis der XML Web-Service Technologie*, pages 17–30, 2002.

[KSSK02] M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services across the Internet (Demonstration). In *Proc. of the Conf. on Very Large Data Bases (VLDB)*, pages 1047–1050, 2002.

[Ley01] F. Leymann. Web Services Flow Language (WSFL 1.0). `http://www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, 2001. IBM.

[Mar99] H. Marais. Compaq's Web Language. `http://www.research.compaq.com/SRC/WebL/WebL.pdf`, 1999. Compaq.

[NET] Microsoft .NET. `http://www.microsoft.com/net`.

[Pas] Microsoft Passport. `http://www.passport.com`.

[Pro] Liberty Alliance Project. `http://www.projectliberty.org`.

[RV02] E. Rahm and G. Vossen, editors. *Web & Datenbanken: Konzepte, Architekturen, Anwendungen*. dpunkt-Verlag, 2002.

[SBK01] S. Seltzsam, S. Börzsönyi, and A. Kemper. Security for Distributed E-Service Composition. In *Proc. of the 2nd Intl. Workshop on Technologies for E-Services (TES)*, volume 2193 of *Lecture Notes in Computer Science (LNCS)*, pages 147–162, 2001.

[Sun] Sun Open Net Environment (Sun ONE). `http://www.sun.com/sunone`.

[Tha01] S. Thatte. XLANG: Web Services for Business Process Design. `http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm`, 2001. Microsoft.

[UDD00] Universal Description, Discovery and Integration (UDDI) Technical White Paper. `http://www.uddi.org`, 2000. White Paper.

[WSP] HP Web Services Platform. `http://www.hp.com/go/webservices`.