Fachseminararbeit · Wintersemester 2011/2012
Fachbereich Design Informatik Medien · Hochschule RheinMain
Bachelor-Studiengang Medieninformatik

Konzepte und Standards zur domänenübergreifenden Integration von komplexen Webanwendungen

Markus Tacker

https://github.com/tacker/fachseminar/

22. Dezember 2011

Abstract

Die Suche nach Konzepten für die dynamische Bindung von Webservice ist die Motivation für diese Fachseminararbeit. Vorraussetzung dafür ist, dass Webservice semantisch beschrieben werden, erst so ist eine automatische Dienstvermittlung zur Laufzeit möglich. Wie Dostal und Jeckle aber in [4, S.55] beschreiben, wurden aktuell verbreitete Standards zur Anbindung von Webservice wie z.B. die *WSDL* aber im Hinblick auf die Anbindung von *Services* mit einer konkreten *API* entwickelt — sie beschreiben den syntaktischen Rahmen einer Schnittstelle, das zu Grunde liegende Wissen über das Domänenkonzept, also die *Semantik*, wird nicht festgehalten. Nach einer Einführung in das Thema in Abschnitt 1, in dem der aktuellen Stand der Entwicklung beschrieben wird und die daraus resultierende Hindernisse bei der dynamischen Bindung von Webservice erläutert werden, werden in Abschnitt 2 die theoretischen Aspekte und wie man mit Hilfe von *Ontologien* Domänenkonzepte semantisch beschreibbar machen kann vorgestellt. Mit der *SAWSDL* liefert das *W3C* den Entwurf eines Standards für diese Aufgabe, der in Abschnittes 2.2 beschrieben wird. Abschnitt 3 stellt mögliche Lösungsansätze für die Implentierung einer *SOA* vor, deren Dienste zur Laufzeit gebunden werden. Im Fazit in Abschnitt 4 werden die aufgezeigten Konzepte auf ihre Anwendbarkeit auf *komplexen Webanwendungen* analysiert.

Inhaltsverzeichnis

1 Einleitung

In diesem Abschnitt wird das grundlegende Problem bei der Anbindung *webbasierter Anwendungen* erläutert.

Webbasierter Anwendungen zeichnet aus, dass sie komplexe Arbeitsabläufe abbilden — die Anwendung wird dadurch *zustandsbehaftet*. Als Beispiel für diese Art von webbasierten Diensten werde ich in dieser Seminararbeit eine Online-Zeiterfassung (*mite*)[1] betrachten.

Angenommen, ein IT-Unternehmen setzt in seinem Intranet eine Projektverwaltungssoftware ein, in der alle Mitarbeiter auf alle Projekte, an denen sie arbeiten, zugriff haben. In diesem Intranet werden auch die Aufgaben zu den einzelnen Projekten verwaltet, sowie die zugeordneten Kostenstellen. Die Software bietet auch eine Funktion zur Zeiterfassung an, diese ist aber sehr unkomfortabel und wird von den Mitarbeitern deswegen nicht gerne verwendet. Das Unternehmen entscheidet sich nun, *mite* zur Zeiterfassung einzusetzen. Die Funktionalität von *mite* kann für sich alleinstehend verwendet werden. Hierzu werden über die Website die nötigen Businessdaten von *mite* (Benutzer, Leistungen[2], Projekte, Kunden und Zeiten) angelegt. Diese Daten werden bei *mite* gespeichert. Mit der öffentlichen *API*[3] von *mite* ist es aber auch möglich, alle Businessdaten „von außen“ zu bearbeiten. Um die Verwendung für die eigenen Mitarbeiter so komfortabel wie möglich zu machen, und die erfassten Zeiten automatisch den eigenen Projekten zuordnen zu können, implementiert das Unternehmen in der Intranet-Software ein Mapping zwischen seinen eigenen Businessdaten und denen von *mite*. Mit Hilfe das Mappings können beide System parallel verwendet

[1] http://mite.yo.lk/
[2] beschreibt eine Tätigkeit, z.B. Programmierung, mit einem Stundensatz
[3] http://mite.yo.lk/api/index.html

werden und es ist sichergestellt, dass die Datenbestände beider Seiten synchronisiert sind. Entscheidet sich das Unternehmen aber zu einem späteren Zeitpunkt, einen anderen oder weiteren Anbieter zur Zeiterfassung einzusetzen muss diese Anbindung erneut implementiert werden.

Hier wird das Problem offensichtlich: Die bisherige Vorgehensweise, Webservice individuell an eine Software zu binden ist nicht flexibel. Es fehlte eine Beschreibung der Schnittstelle, die derart gestaltet ist, dass die Vermittlung zwischen den verschiedenen Domänenkonzepten, das Mapping, automatisch erfolgen kann.

Der Wunsch nach solch einer flexiblen Architektur manifestiert sich in der *Service Oriented Architecture* (*SOA*). Die Idee, Informationen und Funktionen von Software mit Hilfe von Webservice zu verwenden, fußt auf dem durch die Konzepte zur *SOA* eingeleiteten Paradigmenwechsel zu modularen Systemen mit loser Kopplung. In einer *SOA* werden Anwendungen nicht mehr monolithisch aufgebaut, sondern in kleinere, in sich geschlossene Komponenten unterteilt. Diese kommunizieren miteinander in einem Intra- oder einem Extranet über ihre öffentliche Schnittstellen, der sogenannten *Application Programming Interface* (*API*). Diese Kapselung von Diensten hat auch zum Ziel, eine möglichst hohe Kohäsion innerhalb eines Systems zu ermöglichen — Quellcode soll, wenn möglich, nur einmal geplant, entworfen und geschrieben werden, und im ganzen System verwendet werden, woraus im Endergebnis weniger Code, eine höhere Standardisierung und damit letztendlich niedrigere Kosten resultieren [7].

Spätestens mit der neueren Entwicklung des Internets zum *Web 2.0* wurde diese Idee auch auf das Internet übertragen. Inzwischen ist es die Regel, dass webbasierte Anwendungen einen Teil ihrer Daten und Funktionen mit Hilfe von Schnittstellen „nach außen“ publizieren. Die Kommunikation mit diesen Schnittstellen ist dabei auf Protokollebene standardisiert. Nach Elyacoubi, Belouadha und Roudies in [6] sind etablierte Standards für Webservices der ersten Generation wie *Web Services Description Language 2.0* (*WSDL*), *Simple Object Access Protocol* (*SOAP*) und *Universal Description, Discovery and Integration* (*UDDI*) primär unter dem Aspekt entwickelt worden, einen einfachen Weg zur Verteilung und Wiederverwertung von Services in einer *SOA* zu etablieren. Mithilfe dieser Standards lassen sich zu einem Problem passende Dienste in der *UDDI* finden, die *WSDL* kann dann dazu verwendet werden, Quellcode(-Fragmente) zu generieren und mit *SOAP* kann schließlich mit ihnen kommuniziert werden. Es fehlt ihnen aber die Möglichkeit, das Auffinden *discovery*), Veröffentlichen (*publication*), Zusammenstellen (*composition*), und Aufrufen (*invocation*) automatisch vorzunehmen. [6, S.653]

Um ein wirklich modulare Architektur zu erzeugen, muss diese in der Lage sein, die zur Erledigung einer Aufgabe nötigen Abläufe aus beliebigen, passenden Diensten zusammensetzen zu können — erst so wird eine Architektur Fehlertolerant und kann auf das entfallen und hinzukommen von Diensten schnell reagieren. Voraussetzung hierfür ist, dass die Semantik eines Webservice maschinenlesbar zur Verfügung gestellt wird, damit eine automatische Vermittlung zwischen einer Aufgabe und zur Verfügung stehenden Webservice erfolgen kann.

Im Hinblick auf ökonomische Aspekte kann es sogar von Vorteil sein, die parallele Verwendung mehrere Dienste der gleichen Art zu ermöglichen. Patel beschreibt z.B. in [11, S.29], dass Unternehmensintranets oft zu unspezifisch für die individuellen Bedürfnisse eines ein-

zelnen Mitarbeiters sind. Im unserem Intranet-Beispiel könnte das Unternehmen seinen Mitarbeitern die Zeiterfassung mit *mite* ermöglichen, aber auch alternativ mit z.B. *TimeNote*[4]. Auf den ersten Blick erscheint diese Heterogenität kontraproduktiv, das Unternehmen eröffnet seinen Mitarbeitern aber so die Möglichkeit, das Werkzeug für die gegebene Aufgabe „Zeiterfassung“ zu verwenden, dass ihren Vorlieben am ehesten entspricht, und kann dadurch die Akzeptanz dafür steigern. Neben der Möglichkeit der Wahl, erhält man so auch automatisch Redundanz. Masak hat in [10, S.236ff] diesen Gedanken auf eine größere Ebene übertragen und kommt zu dem Schluss, dass es in einem digitalen Ökosystem, wie das Internet eines ist, notwendig ist, Redundanz auf allen Ebenen einzuführen, und durch eine abstrahiertes Mapping eine ad-hoc Komposition von Services zu ermöglichen, da es durch die schiere Größe unmöglich ist alle Aspekte des Systems systematisch zu erfassen.

2 Semantische Webservice

Wie im vorigen Abschnitt beschrieben wurde, fehlt der bisherigen Beschreibung von Webservices der semantische Aspekt. Unter *semantischen Webservice* versteht man solche Webservice , deren Beschreibung neben der konkreten technischen Aspekten zur Anbindung auch Information zur abgebildeten „Welt“ enthalten. In diesem Abschnitt erläutere ich deren Grundlagen.

2.1 Semantik

Die *Semantik* (griechisch, „Bezeichnung“) beschreibt das Wesen von Dingen und ermöglicht die Interpretation und Übertragung von Konzepten auf konkrete Begebenheiten. Semantik ist die Grundlage jeglicher Kommunikation und umgibt uns überall. Bereits in jungen Jahren lernen wir, dass ein über einem Weg hängender Kasten, aus dem uns ein Licht rot anstrahlt eine *bestimmte* Bedeutung hat. Nach einiger Zeit verbinden wir damit intuitiv: „Halt, hier geht es nicht weiter.“ Wichtig ist allerdings, dass die scheinbar eindeutige Verbindung zwischen der Farbe „Rot“ und dem Konzept „Nicht weiter gehen!“ kontextabhängig ist. Begegnet uns ein leuchtendes Rot auf einem Apfel, wissen wir, dass das Obst frisch und genießbar ist. Die Bedeutung „Halt!“ der Farbe Rot verdreht sich in diesem Kontext in das Gegenteil: „Iss mich!“.

Wie schon auf Seite 3 beschrieben, ist Voraussetzung für eine Service-Infrastruktur mit loser Kopplung, dass die Bedeutung der Aufgabe, die mit dem Webservice abgebildet wird automatisch ermittelt werden kann.

Beschreibt man einen Webservice z.B. mittels der *WSDL*, legt man damit lediglich den Syntax für die vom Webservice verarbeiteten Anfragen fest. Die Bedeutung der Funktionalität und der übertragenen Daten erschließt sich daraus nicht. Sie entsteht lediglich in der Interpretation der Benutzer des Dienstes.

In Listing 1 auf Seite 14 findet sich eine *WSDL* für der Webservice „PeopleAsk“[5], mit dem

[4] http://www.timenote.de/
[5] http://peopleask.ooz.ie/

sich die aktuell in Google gestellten Fragen abrufen lassen. Beschrieben werden die Entitäten *GetQuestionsAbout* mit dem Attribut *query* und *GetQuestionsAboutResponse*, das eine Liste mit Strings ist. Aus dem Dokument geht jedoch nicht hervor, dass eigentlich *Suchanfragen* einer *Suchmaschine* zurückgegeben werden — dieses Wissen entsteht aus Informationen, die nur außerhalb der Schnittstellenbeschreibung zugänglich sind.

Es fehlt also eine Komponente, die dem reinen Akt der Datenübertragung ein inhaltlichen Beschreibung, hinzufügt und das zudem noch in maschinenlesbarer Form. In der Informatik sind das *Ontologien*.

Ontologien sind die *Spezifikation eines Konzepts*. *Spezifikation* bedeutet dabei eine formale und deklarative Repräsentation, die damit automatisch maschinenlesbar ist und Missverständnisse ausschließt. Ein *Konzept* ist die abstrakte und vereinfachte Sicht der für das Konzept relevanten Umgebung. Ontologien beschreiben aus der Sicht des Dienstanbieters die Zusammenhänge in der Umgebung, auf die durch den Webservice implizit zugegriffen wird. In [3, S.31] liefert Devedžić zum bessern Verständnis dieses Bildnis: Möchte eine Person über Dinge aus der Domäne *D* mit der Sprache *L* sprechen, beschreiben Ontologien die Dinge, von denen angenommen wird, dass sie in *D* als Konzepte, Beziehungen und Eigenschaften von *L* existieren.

2.2 Semantische Beschreibung von Webservice

Mit den *Semantic Annotations for WSDL*[6] (*SAWSDL*) hat das *World Wide Web Consortium*[7] (*W3C*) im Jahr 2007 den Entwurf eines Standard vorgelegt, der es ermöglicht Informationen zu diesen Ontologien maschinenlesbar, als Teil einer *WSDL*-Datei, auszuliefern.

SAWSDL ist dabei unabhängig von einem semantischen Konzept und liefert nur den Rahmen um andere semantische Frameworks in *WSDL* zu integrieren — es wird lediglich vorausgesetzt, dass diese Konzepte anhand von URIs identifierziert werden können [8, S.61]. Eine Mögliche Technik zur semantischen Beschreibung bietet die *OWL 2 Web Ontology Language*[8] (*OWL 2*), die auf dem *Resource Description Framework*[9] (*RDF*) basiert. In *RDF* werden dabei die Entitäten (in *RDF* „Ressourcen“ genannt) mit ihren Attributen und Beziehungen untereinander syntaktisch beschrieben. In *RDF* fehlt aber die Möglichkeit, die Beziehungen von Eigenschaften zu beschreiben. Zum Beispiel besitzt ein Buch das Attribut *Autor*. Dass damit aber eine weitere Ressource gemeint ist (eine *Person* mit der Rolle *Autor*) lässt sich in einem *RDF*-Dokument nicht hinterlegen. Mit der *RDF Vocabulary Description Language: RDF Schema*[10] (*RDFS*) wurde deswegen die Möglichkeit geschaffen, Gruppen zusammengehöriger Ressourcen und ihrer Beziehung untereinander zu beschreiben. Mit *OWL 2* ist es schließlich möglich, Ontologien in Form von Klassen, Eigenschaften, Instanzen und Operationen zu beschreiben.Abbildung 1 auf Seite 6 liefert einen Überblick über den Zusammenhang der vorgestellten Technologien.

In *WSDL* werden Webservice auf einer syntaktischen Ebene beschrieben. Es wird festgelegt,

[6] http://www.w3.org/TR/sawsdl/
[7] http://www.w3.org/
[8] http://www.w3.org/TR/owl2-primer/
[9] http://www.w3.org/TR/rdf-primer/
[10] http://www.w3.org/TR/rdf-schema/

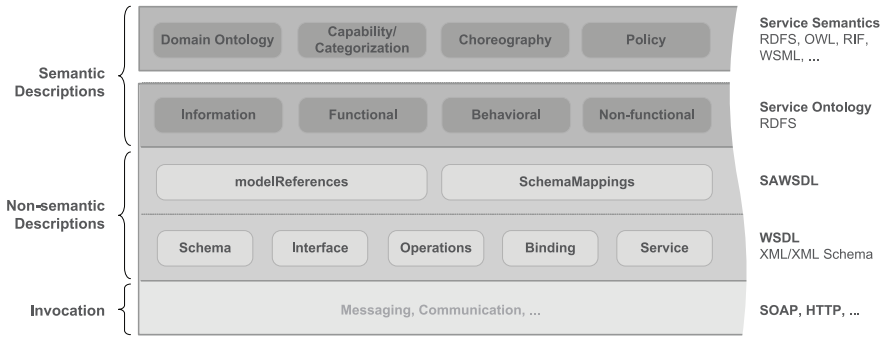

Abbildung 1: *Extended Web service specification stack* [8, S.63]. Die Hauptbeschreibungssprache *WSDL* ist eng an die darunterliegenden Kommunikationsprotokolle gekoppelt. *SAWSDL* verbindet als Schicht darüber *WSDL* mit den übergeordneten semantischen Informationen, wobei die Service-Ontologien die allgemeinen Aspekte von Webservice beschreiben und die Service-Semantik die domainspezifischen Aspekte formuliert.

wie die auszutauschenden Nachrichten *aussehen* und an welchen Endpunkten der *API* diese zum Einsatz kommen, nicht jedoch was sie *bedeuten*. In *WSDL* werden die abstrakten Elemente *Element Declaration*, *Type Definition* und *Interface* verwendet, um einen Webservice allgemein zu beschreiben. In diesen Elementen spielen die technischen Einzelheiten, wie z.B. das verwendete Protokoll, keine Rolle. Ein *Type* entspricht dabei einem Objekt aus der Domäne, ein *Element* beschreibt ein Attribut dieses Objekts. Ein *Interface* beschreibt die Operationen und deren Parameter, die von der Schnittstelle unterstützt werden. In Listing 2 auf Seite 15 findet sich ein einfaches Beispiel einer *WSDL*-Datei. Abbildung 2 auf Seite 7, die das Datenmodell *WSDL*-Datei zeigt, liefert einen Überblick über die Elemente in *WSDL*[11].

SAWSDL führt nun für diese drei Elemente zusätzlich Attribute ein, um deren semantische Bedeutung zu beschreiben [8, S.62ff]:

- `modelReference` definiert eine Beziehung zwischen einem der definierten Komponente in der *WSDL* und einem Objekt im semantischen Modell. Diese Attribut kann auf jedes *WSDL*- oder XML-Schema-Element angewendet werden. Der Wert des Attributes ist dabei eine oder mehrere URIs, die auf ein semantisches Modell verweisen.

- Die Attribute `liftingSchemaMapping` und `loweringSchemaMapping`, die auf Typedefinitionen definiert werden können, spezifizieren das Mapping zwischen semantischen Daten (z.B. *RDF* und XML) sowie umgekehrt. Hierbei ist es auch möglich, mehrere Mappings je Typ zu definieren, um verschiedene Repräsentation je Kontext zu ermöglichen. Die *lifting*- und *lowering*-Transformationen sind nützlich, wenn von einem semantischen Client aus mit einem Webservice kommunziert wird. Für eine Anfrage werden dann die semantischen Daten in das Anfrage-Format des Client durch *lowering* transformiert, die Antwort wird dann durch *lifting* wieder in ein semantisches Format konvertiert. Dieses Verfahren kommt auch bei der Verwendung einer gemeinsamen Ontologie zum Einsatz — ein automatischer Vermittler kann da-

[11] Quelle: http://www.w3.org/TR/wsdl20-primer/

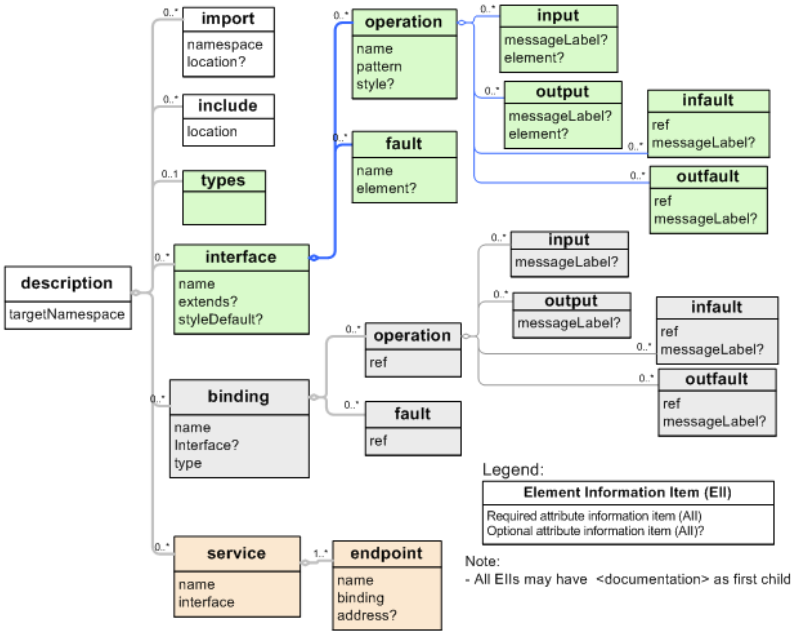

Abbildung 2: Das Datenmodel von *WSDL*

bei die Daten zwischen zwei Endpunkten mit den *Lifting*-Informationen des Anfragers und den *Lowering*-Informationen des Empfängers vermitteln. Sprachen für das `liftingSchemaMapping` sind z.B. *XSLT* oder *XQuery*, für das `loweringSchemaMapping` *SPARQL* (eine Abfrage-Sprache für *RDF*) gefolgt von *XSLT* oder *XQuery*.

Abbildung 3 auf Seite 8 liefert eine Übersicht über die Integrationspunkte von *SAWSDL* in *WSDL*.

2.3 RESTful Webservices

Auch wenn dass *W3C* in seinen Standards und Entwürfen keine Vorgaben zur eigentlichen Realisierung der Kommunikation mit Webservices macht, sind Standards wie *WSDL* auf die Anforderungen einer *SOAP*-basierten Kommunikation ausgelegt. *SOAP*-basierte Webservice haben aber nur wenig mit den Strukturen des Webs gemeinsam, dort ist die vorherrschende Sprache *HTTP* bei dem Ressourcen durch URIs repräsentiert werden und Aktionen durch Verben wie `GET` und `POST`. Im Gegensatz dazu versteht *SOAP* einen Webservice als die *API* einer Software, auf der Methode aufgerufen werden, wobei es sich zusätzlich noch um die Serialisierung und Deserialisierung der Nachrichten kümmert. Die Einfachheit einer *RESTful API* bietet sich also gerade dann an, wenn der zugrunde liegende Dienst sowieso schon HTTP-Anfragen verarbeiten muss. Aus diesem Grund erfreuen sich *RESTful APIs* vor allem bei Schnittstellen zu Webservices einer großen Beliebheit [12, S.18].

Auch die in Abschnitt 2 vorgestellte *API* von *mite* ist *RESTful*. Möchte man nun aber solch eine *API* semantisch beschreiben steht man vor dem Problem, dass es mit einer *WSDL* nicht möglich ist einen *RESTful API* korrekt zu beschreiben. Dies liegt vor allem daran, dass das

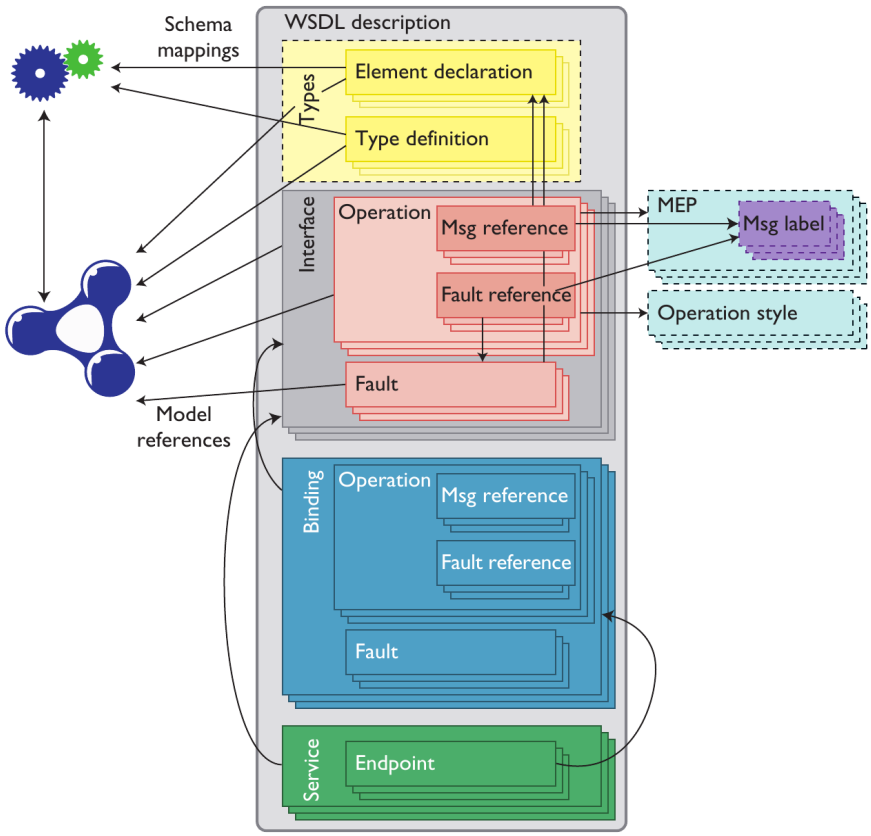

Abbildung 3: *WSDL* mit *SAWSDL*. Diese Abbildung zeigt die *WSDL*-Kompüonenten und die zugehörigen *SAWSDL*-Erweiterungen, die zu Konzepten zu semantischen Spezifikation der Domain oder zum Data-Mapping verweisen. [8, S.61]

HTTP-Binding in *WSDL* 1.1 zum einen nur `GET` und `POST` offiziell Unterstützt (in *Representational state transfer* (*REST*) sind zusätzlich mindestens noch `PUT` und `DELETE` verwendet und generell die Verwendung eigener Methoden gestattet) und ein *port type* (der Endpunkt einer API-Methode) bis zur vier verschiedene Operation pro Endpunkt definiert (Schreiben [*One-way*], Schreiben & Lesen [*Request-response*], Lesen [*Solicit-response*] und Benachrichtigung [*Notification*]), diese aber mit der gleichen Methode (die im *binding* festgelegt wird).

WSDL und damit auch *SAWSDL* sind also ungeeignet um eine *RESTful API* semantisch zu beschreiben, mit den Veröffentlichungen von Xuan Shi in [12] und Maleshkova, Kopecký und Pedrinaci in [9] stehen aber zwei Entwürfe zur Verfügung, die eine Lösung für dieses Problem bieten, deren Vorstellung aber in dieser Seminararbeit nicht behandelt wird.

2.4 *SAWSDL* verwenden

Der Standardisierungsprozess des *W3C* fordert von allen Entwürfen für neue Standards, dass diese auf ihrer vollständige Implementierbarkeit überprüft werden müssen, bevor sie

letztendlich zur Empfehlung erhoben werden dürfen. Jedes Feature der Spezifikation muss funktional in mindestens einer Implementierung vorliegen und idealerweise zwischen zwei Implementierungen interoperabel sein. Damit eine Arbeitsgruppe, also diejenigen die neue Standards entwickeln, zu einer Empfehlung kommen kann, muss diese Gruppe einen Implementierungsreport vorlegen. Im Bericht der *SAWSDL*-Arbeitsgruppe[12] finden sich Implementierungen in mehreren Bereichen des Standards.

Direkte Implementierungen von *SAWSDL* sind Parser-*API*s, die die Erweiterungen anderen Anwendungen und Werkzeugen zur Verfügung stellen, mit denen *WSDL*-Dokumente semantisch beschrieben werden können. So wurde die *Woden API für WSDL 2.0*[13] und die *WSDL4J API für WSDL 1.1*[14] so erweiterte, dass diese *SAWSDL*-Informationen auslesen können. Auf diesen Bibliotheken bauen viele Jaba-basierte Werkzeuge auf, wie z.B. der Apache Webservice Stack *Axis 2*[15]. Auch zwei GUI-Werkzeuge, mit denen *WSDL*-Dokumente semantisch beschrieben werden können, unterstützen den Standard: *Radiant*[16] von der *University of Georgia* und das *Web Service Modeling Ontology Studio*[17] von *Ontotext*. Da *SAWSDL* wie beschrieben eine Spezifikation zum Hinzufügen von Semantiken ist, liegt sein Wert vor allem in den Anwendungen, die von diesen Semantiken gebrauch machen. Im Implementierungsreport werden mehrere solche Anwendungen erwähnt. Insbesonderes kann man *Semantic Markup for Web Services* (OWL-S) und *Web Service Modeling Ontology* (WSMO), die zu den gängigsten Frameworks für semantische Webservices zählen, mit *SAWSDL* in *WSDL*-Dokumente integrieren. Mit *Lumina*[18] von der *University of Georgia* können mit *SAWSDL* beschriebene Webservice „entdeckt“ werden und die *Semantic Tools for Web Services*[19] von IBM sind in der Lage, semantische Daten zu verknüpfen und vermitteln. In *SAWSDL* ist es sogar möglich, semantische Beschreibungen für die *Business Process Execution Language for Web Services*[20] (BPEL4WS) zu hinterlegen, ein Anwendungsfall der ursprünglich nicht von der Arbeitsgruppe berücksichtigt wurde. [8, S.63]

Auch für die *Eclipse IDE* gibt es mit *Jena*[21] ein Plug-In dass bei der Entwicklung von semantischen Webservices in Java unterstützt.

Wie in diesem Abschnitt gezeigt wurde, sind also die theoretischen Grundlagen und praktischen Hilfsmittel zur semantischen Beschreibung von Webservice vorhanden.

3 Dynamische Verwendung von semantischen Webservice

Ziel einer Lösung muss es also sein, dass man Web Services, die zur Entwicklungszeit noch unbekannt sind, dynamisch binden kann. In diesem Abschnitt stelle ich Lösungsansätze vor, in denen die dynamische Bindung von semantischen Webservice möglich ist.

[12] http://www.w3.org/2002/ws/sawsdl/CR/
[13] http://ws.apache.org/woden/
[14] http://sourceforge.net/projects/wsdl4j/
[15] http://axis.apache.org/axis2/java/core/
[16] http://lsdis.cs.uga.edu/projects/meteor-s/downloads/index.php?page=1
[17] http://www.wsmostudio.org/
[18] http://lsdis.cs.uga.edu/projects/meteor-s/downloads/Lumina/
[19] http://lists.w3.org/Archives/Public/public-ws-semann/2006Oct/0030.html
[20] http://www.ibm.com/developerworks/library/specification/ws-bpel/
[21] http://jena.sourceforge.net/

3.1 Lösungsansätze

Wie Dostal und Jeckle in [5, S.61] ausführen, werden in den üblichen Beschreibungen zum Ablauf in einem Web-Service-Szenario *WSDL*-Dokumente hauptsächlich als Eingabe für einen Generator beschrieben, mit dessen Hilfe die programmierspezifischen Implementierungen erzeugt werden. Dies erfolgt allerdings in der Regel zur Entwicklungszeit der Anwendung und nicht zu deren Laufzeit. Zwar ist es bei Sprach- und Ausführungsumgebungen wie Java heute technisch durchaus möglich, auch zur Ausführungszeit Klassen der Anwendung hinzuzufügen und auf diesem Weg das oben beschriebene Implementierungsszenario von der Entwicklungs- in die Laufzeit zu verlagern. Allerdings würde ein solcher Ansatz eine Reihe von Nachteilen bzw. Riksiken bergen. Das gewichtigste Argument gegen den Genierungsansatz ist zweifelsfrei im Bereich Sicherheit angesiedelt. Das Einbinden von nicht getestetem Code bietet geschickten Angreifer ein *el Dorada* von Möglichkeiten, potentiell gefährliche Programmsequenzen in die Anwendung ein zu schmuggeln. Statt des generativen Ansatzes eignet sich daher ein Framework, das mit Hilfe der Informationen in einem *WSDL*-Dokument eine *SOAP*-Kommunikation durchführen kann, ohne dazu Codegenerierungen durchführen zu müssen, besser. Für Java-Anwendungen existiert dazu unter anderem das *Web Services Invocation Framework*[22] (*WSIF*) der Apache Group. Entsprechend den Elementen einer WSDL-Beschreibung sind innerhalb des *WSIF* Klassen definiert, die mittels der *WSDL*-Eingabe parametrisiert werden. Damit ist es möglich jeden denkbaren Web Service „spontan“ zu nutzen und so tatsächlich dynamisches Verhalten der Anwendungen in Web-Service-Szenarien zu erreichen. Abbildung 4 auf Seite 10 zeigt schematisch die Ablaufe innerhalb einer solchen Lösung.

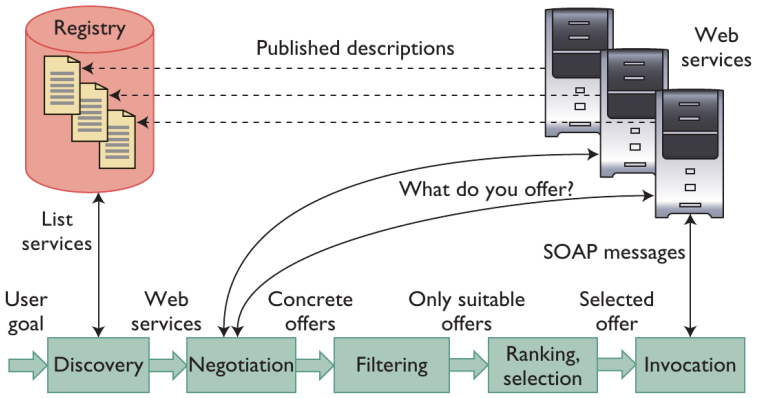

Abbildung 4: *Abläufe in einem System für semantische Webservices* [8, S.64]. Alle Abläufe, beginnend mit dem Auffinden von Webservices die zu einer Aufgabe passen bis zum Aufrufen des ausgewählten Webservice , können mit semantischen Technologien automatisiert werden.

[22] http://ws.apache.org/wsif/

3.2 Implementierungen

Im Rahmen des *Automated Service Brokering in Serive-oriented Architectures* (*ADDO*)-Projektes[23] and der Universität Kassel, dass sich zum Ziel gesetzt hat, einen automatischen Algorithmus zur qualitätsberücksichtigenden Service-Aufindung und ein Framework zur automatischen Serviceintegration und -verwaltung zu entwickeln haben Bleul, Zapf und Geihs in [1, S.410ff] eine Archtitektur entworfen, die in der Lage ist, die angesprochenen Anforderungen zu erfüllen. Die vorgestellte Architektur enthält einen *Service Broker*, an dem sich semantische Services registrieren und der in der Lage ist automatisch Services aufzufinden. Ein *Service Container* überwacht die Services und deren Integration in das System. Die Architektur kann Services auch zur Laufzeit austauschen und sich so selbst zu „heilen“. Anbieter von Services müssen in diesem System die semantische und syntaktische Beschreibung selber Erstellen und beim *Service Broker* registrieren — sie müssen sich auch um die Deregistrierung kümmern, sollte der Dienst nicht mehr zur Verfügung stehen [1, S.416].

Auch eine Gruppe von Wissenschaftlern aus Italien hat mit dem *C-Cube Framework*[2] einen ähnlichen Entwurf vorgelegt. Das System besteht, ähnlich wie das *ADDO*-Projekt aus Komponenten zur Verwaltung der aktiven Services (namentlich *Service, Service Description, Service Monitoring*) zum Binden und Ausführen von Services (hier *Service Execution*). Das System ist in der Lage automatisch auf Basis semantischer Beschreibungen nach zu einer Anfrage passenden Diensten zu suchen, diese zu Binden und auszuführen [2, S.4].

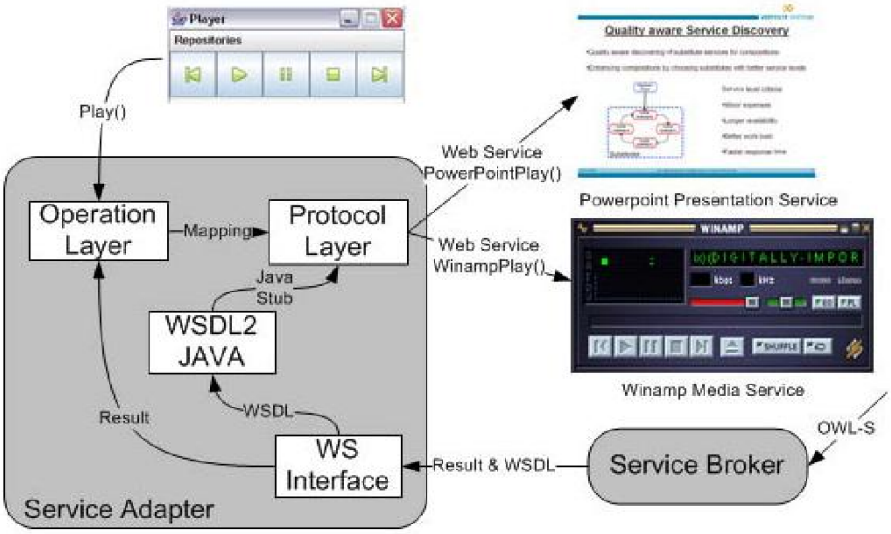

Abbildung 5: *Referenz-Implementation des* ADDO-*Projektes* [1, S.418].

Das *ADDO*-Projekt hat zur Verdeutlichung der Zusammenarbeit der vorgestellten Komponenten eine einfache Beispiel-Anwendung implementiert (siehe Abbildung 5 auf Seite 11). Die Anwendung ist ein einfaches Multimedia-Player-Frontend, das in Java implementiert ist. Der Player erfordert einen Service, der die Operationen *Play*, *Stop* und *NextTitle* zur Verfügung stellt, sowie optional nach zusätzlich *Pause* oder *PreviousTitle*. Es stehen zwei Systeme zur Verfügung, die diese Operationen anbieten: ein *PowerPoint*-Laptop, der an einen Beamer

[23] http://www.vs.uni-kassel.de/ADDO/index.html

angeschlossen ist, sowie ein *WinAmp*-Medien-Player, der an einer Stereo-Anlage angeschlossen ist. Beide System sind über einen eigenen Webservice ansprechbar und implementieren mindestens die Operationen *Play*, *Stop* und *NextTitle*. Ihre semantische Beschreibung liegt als OWL-S im *Service Broker* vor. Der *Service Container* ist als Java-Bibliothek implementiert und steht in der Anwendung als Instanz zur Verfügung. Ziel ist es nun, einen generischen Medien-Player an die in der aktuellen Umgebung verfügbaren Operationen zu binden. Sobald der *Service Container* von der Anwendung aufgerufen wird, löst die bis jetzt leer Bindung den *Broker*-Mechanismus aus und entweder der *PowerPoint*- oder der *WinAmp*-Webservice wird durch den *Service Broker* gebunden. Der gebundene wird an den *Service Container* übergeben. Mit Hilfe eines *ant*-Scripts auf Basis von *WSDL2JAVA* (Teil des Axis-Projekts[24]) wird automatisch ein passender Service-Stub generiert, der die verfügbaren Operationen entsprechend der Webservice-Beschreibung, die in *WSDL* vorliegt, konfiguriert und anschließend in den *Service Container* integriert. Im Fall einer Service-Änderung (Wegfall eines gebundenen Services) wird eine Exception geworfen und die Anwendung kann entweder die Verwendung abbrechen oder einen neuen Service verwenden.

Die hier genannten Beispiel zeigen, dass es bereits funktionsfähige Implementierungen einer Architektur gibt, die in der Lage ist, Services zur Laufzeit zu binden.

4 Fazit

[24] http://ws.apache.org/axis/

Literatur

[1] S. Bleul, M. Zapf, and K. Geihs. Flexible automatic service brokering for soas. In *Integrated Network Management, 2007. IM '07. 10th IFIP/IEEE International Symposium on*, pages 410–419, Mai 2007.

[2] Gerardo Canfora, Piero Corte, Antonio De Nigro, Debora Desideri, Massimiliano Di Penta, Raffaele Esposito, Amedeo Falanga, Gloria Renna, Rita Scognamiglio, Francesco Torelli, Maria Luisa Villani, and Paolo Zampognaro. The c-cube framework: developing autonomic applications through web services. *SIGSOFT Softw. Eng. Notes*, 30:1–6, May 2005.

[3] Vladan Devedžić. *Semantic Web and Education*, chapter Introduction to the Semantic Web, pages 29–69. Springer's Integrated Series in Information Systems. Springer, 2006.

[4] W. Dostal and M. Jeckle. Semantik, Odem einer Service-orientierten Architektur. *JavaSPEKTRUM*, 04(1):53–56, 2004.

[5] W. Dostal and M. Jeckle. Semantik und Webservices. *JavaSPEKTRUM*, 04(4):58–62, 2004.

[6] N.E. Elyacoubi, F.-Z. Belouadha, and O. Roudies. A metamodel of wsdl web services using sawsdl semantic annotations. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on*, pages 653–659, Mai 2009.

[7] J.T. Howerton. Service-oriented architecture and web 2.0. *IT Professional*, 9(3):62–64, Mai-Juni 2007.

[8] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell. Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE*, 11(6):60–67, Nov-Dez 2007.

[9] Maria Maleshkova, Jacek Kopecký, and Carlos Pedrinaci. Adapting sawsdl for semantic annotations of restful services. In *Proceedings of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems: ADI, CAMS, EI2N, ISDE, IWSSA, MONET, OnToContent, ODIS, ORM, OTM Academy, SWWS, SEMELS, Beyond SAWSDL, and COMBEK 2009*, OTM '09, pages 917–926, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] Dieter Masak. Grundlagen der Serviceorientierung. In *Digitale Ökosysteme: Serviceorientierung bei dynamisch vernetzten Unternehmen*, Xpert.press, pages 11–112. Springer Berlin Heidelberg, 2009.

[11] A. Patel. Departmental intranets. *Potentials, IEEE*, 18(2):29–32, April, Mai 1999.

[12] Xuan Shi. Sharing service semantics using soap-based and rest web services. *IT Professional*, 8(2):18–24, März-April 2006.

Listings

Listing 1: Einfaches Beispiel einer WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<definitions
        xmlns="http://schemas.xmlsoap.org/wsdl/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:s="http://www.w3.org/2001/XMLSchema"
        xmlns:tns="http://peopleask.ooz.ie /soap"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        targetNamespace="http://peopleask.ooz.ie/soap"
        >
  <types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://peopleask.ooz.ie/soap">
      <s:element name="GetQuestionsAbout">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="query" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetQuestionsAboutResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="GetQuestionsAbout"
                type="tns:ArrayOfstring"
                />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:complexType name="ArrayOfstring">
        <s:sequence>
          <s:element minOccurs="0" maxOccurs="unbounded" name="string" type="s:string"/>
        </s:sequence>
      </s:complexType>
    </s:schema>
  </types>
  <message name="GetQuestionsAboutSoapIn">
    <part name="parameters" element="tns:GetQuestionsAbout"/>
  </message>
  <message name="GetQuestionsAboutSoapOut">
    <part name="parameters" element="tns:GetQuestionsAboutResponse"/>
  </message>
  <portType name="PeopleAskServiceSoap">
```

```
    <operation name="GetQuestionsAbout">
      <input message="tns:GetQuestionsAboutSoapIn"/>
      <output message="tns:GetQuestionsAboutSoapOut"/>
    </operation>
  </portType>
  <binding name="PeopleAskServiceSoap" type="tns:PeopleAskServiceSoap">
    <soap:binding transport ="http://schemas.xmlsoap.org/soap/http" style="document"/>
    <operation name="GetQuestionsAbout">
      <soap:operation soapAction="http://peopleask.ooz.ie /soap/GetQuestionsAbout"
        style ="document"
        />
      <input>
        <soap:body use="literal"/>
      </input>
      <output>
        <soap:body use="literal"/>
      </output>
    </operation>
  </binding>
  <service  name="PeopleAskService">
    <port name="PeopleAskServiceSoap" binding="tns:PeopleAskServiceSoap">
      <soap:address location ="http://peopleask.ooz.ie /soap"/>
    </port>
  </service>
</definitions >
```

Listing 2: WSDL 2.0 Beispiel

```
<?xml version="1.0"?>
<definitions  xmlns:tns="http://example.com/stockquote.wsdl" xmlns:xsd1="http://example.com/stockquote.xsd" x
  <types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.com/stockquote.xs
      <element name="TradePriceRequest">
        <complexType>
          <all>
            <element name="tickerSymbol" type="string"/>
          </all>
        </complexType>
      </element>
      <element name="TradePrice">
        <complexType>
          <all>
            <element name="price" type="float"/>
          </all>
        </complexType>
      </element>
```

```
19      </schema>
20    </types>
21    <Interface  name="StockQuotePortType">
22      <operation name="GetLastTradePrice">
23        <input message="tns:GetLastTradePriceInput"/>
24        <output message="tns:GetLastTradePriceOutput"/>
25      </operation>
26    </Interface>
27  </definitions >
```