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/

19. Dezember 2011

Abstract

Die Suche nach Konzepten für die dynamische Bindung von Webservices ist die Motivation für diese Fachseminararbeit. Vorraussetzung dafür ist, dass Webservices semantisch beschrieben werden, erst so ist eine automatische Dienstvermittlung zur Laufzeit möglich. Wie Dostal und Jeckle aber in [5, S.55] beschreiben, wurden aktuell verbreitete Standards zur Anbindung von Webservices 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 Webservices 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 am Ende dieses Abschnittes beschrieben wird.

Abschnitt 3 stellt dann schließlich einen möglichen Lösungsansatz für die Implentierung einer *SOA* vor, deren Dienste zur Laufzeit gebunden werden.

Im Abschnitt 4 wird dann die Anwendbarkeit der aufgezeigten Konzepte auf den Bereich der *komplexen Webanwendungen* überprüft, allem auszeichnet, dass sie im Gegensatz zu einfachen Webservices zustandsbehaftet sind und ihre Integration mit den gängigen Mitteln nur individuell und statisch möglich ist.

Inhaltsverzeichnis

1 Einleitung

gehe ich dann auf die Bedeutung der Begriffe die Möglichkeit vorstelle, Webservices semantisch zu beschreiben. Neben einer Einführung in das Thema, in der ich auf die theoretischen Aspekte eingehe, analysiere ich in Umsetzungen dieser Theorien in den Standards *Semantic Annotations for WSDL* (*SAWSDL*) und *Ontology-based Resourceoriented Information Supported Framework* (*ORISF*).

Im 4. Abschnitt beurteile ich dann deren Anwendung bei der Anbindung komplexer Webanwendungen, in dem ich beispielhafte Implementierungen aus der Praxis aufzeige.

Der durch die Konzepte zur *Service Oriented Architecture* (*SOA*) eingeleitete Paradigmenwechsel zu modularen Systemen mit loser Kopplung hat eine ganze Reihe von bewährten Standards hervorgebracht, die die Integration von Diensten über das Internet als sogenannte Webservices (WS) einfach und zuverlässig ermöglichen. Wie Dostal und Jeckle aber in [5, S.55] beschreiben, wurden diese Standards wie z.B. *Web Services Description Language* (*WSDL*) im Hinblick auf die Anbindung von *Services* mit einer konkreten *Application Programming Interface* (*API*) entwickelt. Sie beschreiben den syntaktischen Rahmen einer Schnittstelle — das zu Grunde liegende Wissen über das Domänenkonzept, also die *Semantik*, wird nicht festgehalten. Diese ergibt sich für den Konsumenten einer Schnittstelle aus ausgelagerten Dokumentationen und bildet sich bei die Anbindung seiner Anwendung an eben diese Schnittstelle.

Um ein wirklich modulare Architektur zu erzeugen, muss diese in der Lage sein, die zur Erle-

digung 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 WS maschinenlesbar zur Verfügung gestellt wird, damit eine automatische Vermittlung zwischen einer Aufgabe und zur verfügung stehenden WS erfolgen kann.

Ein Teil der neueren Entwicklung des Internets zum *Web 2.0* basiert auf der Idee, dass Informationen und Funktionen von Software mit Hilfe von *Webservices* verwendet werden können.

Die Kommunikation mit Webservices ist zwar auf Protokollebene standardisiert, muss jedoch vom Konsumenten immer individuell entsprechend dem Domänenmodell des Anbieters implementiert werden, wodurch eine feste Bindung an den Anbieter entsteht [14].

Für sogenannte *Blackbox-Webservices* ist das kein Problem — diese *zustandslosen* Dienste verarbeiten lediglich einfache Daten, d.h. dass der Dienst aufgerufen wird, dieser entsprechend des Aufrufs, z.B. anhand eines übergebenen Datums, reagiert und ein Ergebnis zurückliefert. Jede weitere Anfrage wird unabhängig von einer vorherigen behandelt.

Beispiele hierfür ist z.B. ein Webservice, der Wetterdaten für eine PLZ liefert. Hier gibt der Konsument die PLZ eines Ortes in Deutschland ein und erhält in der Antwort eine Temperatur.

Für die Nutzung des Dienstes reicht die Kenntnis der Schnittstellen aus. Die genauen technischen Abläufe, wie der Webservice aus der PLZ eine Temperatur ermittelt bleiben für den Konsumenten verborgen und sind für diesen auch irrelevant [10].

Diese Eigenschaft steht in direktem Zusammenhang mit einem der wichtigsten Konzepte in der Softwareentwicklung der vergangenen Jahre: In einer *SOA* werden Anwendungen nicht mehr monolithisch aufgebaut, sondern in kleinere, in sich geschlossene Komponenten unterteilt. Diese kommunizieren miteinander in einem Intra- oder ein Extranet über ihre öffentliche Schnittstellen, der sogenannten *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 [12].

Webbasierter Anwendungen zeichnet jedoch aus, dass sie komplexe Arbeitsabläufe abbilden — die Anwendung wird dadurch *zustandsbehaftet*. Als Beispiel für diese Art von webbasierten Diensten werde ich in dieser Seminararbeit die Online-Zeiterfassung *mite*[1] betrachten, mit dem man Arbeitszeit Erfassen und Auswertung kann.

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.

Das Problem bei Schnittstellen zu zustandsbehaften Webservices resultiert daraus, dass der Verwender des Services eine Vermittlungsschicht zwischen seiner Domänen-Logik und der

[1] http://mite.yo.lk/

[2] beschreibt eine Tätigkeit, z.B. Programmierung, mit einem Stundensatz

des Webservices implementieren muss, die zwischen beiden Parteien das Verständnis über die verarbeiteten Entitäten vermittelt und er sich so fest an den jeweiligen Dienst bindet.

Angenommen, ein IT-Unternehmen setzt in seinem Intranet eine Software zur Projektverwaltungssoftware ein, in dem 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 fehlerhaft und wird von den Mitarbeitern deswegen nicht verwendet.

Das Unternehmen entscheidet sich nun, *mite* zur Zeiterfassung einzusetzen. Mit dessen öffentlicher API[3] ist es möglich, alle Businessdaten 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*. Mitarbeiter entsprechen dabei Benutzern, die rechnerischen Stundensätzen von Kostenstellen entsprechen Leistungen. Projekte, Kunden und Zeiten existieren zwar in beiden Domänen, aber mit gänzlich unterschiedlichen Attributen — auch hierfür ist ein Mapping notwendig. Mit Hilfe das Mappings können beide System parallel verwendet werden und es ist sichergestellt, dass die Datenbestände beider Seiten synchronisiert sind.

Was ist aber in dem Fall, dass der gewählte Service nicht mehr eingesetzt werden soll oder kann?

Nach Elyacoubi, Belouadha und Roudies in [9] sind etablierte Standards für Webservices der ersten Generation wie *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 Webservices zu etablieren — ihnen fehlt also eine Standardisierung für das Auffinden, Zusammenstellen und Auswählen von Diensten um eine *lose Kopplung* zu ermöglichen.

> However, these standards are not sufficient to allow an automation of the various tasks of the Web service's life cycle, namely the discovery, the invocation, the publication and the composition. Recently, the W3C consortium produced the SAWSDL language. [9, S.653]

Für ein lebendiges Web-Öko-System ist die lose Kopplung jedoch von entscheidender Bedeutung — im Idealfall lassen sich Dienste so anbinden, dass sie jederzeit und mit geringem Aufwand ausgetauscht werden können.

Das Mapping zwischen den Diensten müsste also so allgemein definiert sein, dass lediglich die *Definition* angepasst werden müsste, aber nicht die *Implementierung* — ein Wechseln des Services hätte lediglich das Ändern einer Schnittstellenbeschreibung zur Folge, ohne dass man konkreten Quellcode anpassen muss.

Führt man diesen Gedanken noch einen Schritt weiter, wäre es optimal, wenn selbst der Schritt des Erstellen des Mappings automatisiert erfolgen kann, da sich die zum Einsatz kommenden Systeme über die Begrifflichkeiten, die der jeweils andere verwendet im Klaren sind [17].

Im Hinblick auf ökonomische Aspekte kann es sogar von Vorteil sein, die parallele Verwen-

[3] http://mite.yo.lk/api/index.html

dung mehrere Dienste der gleichen Art zu ermöglichen. Patel beschreibt z.B. in [18], dass Unternehmensintranets oft zu unspezifisch für die individuellen Bedürfnisse eines einzelnen 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 [16] 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.

2 Semantische Webservices

Semantische Webservices sind ein Konzept, mit dem es möglich wird, die Anbindung von Webservices abstrakt zu beschreiben und so eine lose Kopplung zu erreichen. In diesem Abschnitt erläutere ich deren Grundlagen.

2.1 Semantik

Die *Semantik* (griechisch, „bezeichnen“) 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 verdreht sich in das Gegenteil.

Wie schon auf Seite 4 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 11 findet sich eine WSDL für der Webservice „PeopleAsk“[5], mit dem sich die aktuell in Google gestellten Fragen abrufen lassen. Sie beschreibt die Entitäten

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

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.

2.2 Ontologien

In der Informatik sind *Ontologien* 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 [4] 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* existieren als Konzepte, Beziehungen und Eigenschaften von *L*.

2.3 Semantische Beschreibung

Mit Hilfe des *Resource Description Framework* (*RDF*), der *RDF Vocabulary Description Language: RDF Schema* (*RDFS*) und deren Erweiterung *Web Ontology Language* (*OWL*) ist es möglich, Webservices semantisch zu beschreiben.

In *RDF* werden dabei die Entitäten (in *RDF* „Ressourcen“ genannt) mit ihren Attributen und Beziehungen untereinander syntaktisch beschrieben [15].

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 *RDFS* wurde deswegen die Möglichkeit geschaffen, Gruppen zusammengehöriger Ressourcen und ihrer Beziehung untereinander zu beschreiben [2].

Mit *OWL* ist es schließlich möglich, Ontologien in Form von Klassen, Eigenschaften, Instanzen und Operationen zu beschreiben [11].

Die sich damit bietende Möglichkeit, auch Webservices semantisch zu beschreiben ist also die Voraussetzung dafür, dass Technologien entstehen, mit denen die Vermittlung zwischen zwei Domänen automatisiert statt finden können.

Weiterführende Publikationen In einer Artikelserie aus dem Jahr 2004 ([5], [7], [8] und [6]) beschreiben Wolfgang Dostal, Mario Jeckle und Werner Kriechbaum ausführlich das Konzept der semantischen Webservices.

3 Lösungsansätze

Ziel einer Lösung muss es also sein, dass man Web Services, die zur Entwicklungszeit noch unbekannt sind, dynamisch binden kann. Wie Dostal und Jeckle in [6, 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 Implementierungszenario 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 Angreifenr 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*[6] (*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.

In diesem Abschnitt stelle ich zwei konkrete Ansätze zur automatischen, domänenübergreifenden Vermittlung zwischen Webservices vor.

3.1 SAWSDL

SAWSDL is the World Wide Web Consortium's (W3C; www.w3.org) first step toward standardizing technologies for Semantic Web services (SWSs). As a standard, it provides a common ground for the various ongoing efforts toward SWS frameworks, such as the Web Service Modeling Ontology (WSMO; www.wsmo.org)2 and the OWL-based Web Service Ontology (OWL-S; www.daml.org/services/owl-s).[13, S.60]

On the semantic side, SAWSDL is independent of any ontology technology and assumes that semantic concepts can be identified via URIs. For instance, SWS frameworks can use the Resource Description Framework (RDF) and Web Ontology Language (OWL) with SAWSDL to annotate Web services..[13, S.61]

SAWSDL is a set of extensions for WSDL, which provides a standard description format for Web services. WSDL uses XML as a common flexible data-exchange format and applies XML Schema for data typing. It describes a Web service on three levels:

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

- Reusable abstract interface defines a set of operations, each representing a simple exchange of messages described with XML Schema element declarations.
- Binding describes on-the-wire mes- sage serialization; it follows the structure of an interface and fills in the necessary networking details (for instance, for SOAP or HTTP).
- Service represents a single physical Web service that implements a single interface; the Web service can be accessed at multiple network endpoints.

WSDL aims to describe the Web service on a syntactic level: it specifies what messages look like rather than what they mean. SAWSDL is a simple extension layer on top of WSDL that lets WSDL components specify their semantics. SAWSDL defines extension attributes that we can apply to elements both in WSDL and in XML Schema to annotate WSDL interfaces, operations, and their input and output messages. The SAWSDL extensions take two forms: model references that point to semantic concepts and schema mappings that specify data transformations between messages' XML data structure and the associated semantic model. In Table 1, we summarize the complete syntax introduced by SAWSDL.

... bis Seite 63 in [13]

In *SAWSDL* werden mit einem *liftingSchemaMapping* und einem *loweringSchemaMapping* die Abbildung der Funktionen eines Webservices auf eine Semantik abgebildet.

As we noted in section 1, the technology of the Web services was accompanied by many standards. However, these standards do not remedy to the problem of adaptability to Web services' changes, and do not cover all aspects related to different tasks of the Web service's life cycle, namely, the discovery, the invocation, the publication and the composition. Indeed, many enterprises applications, in particular, can constantly have need to discover and use existing Web services, or to compose them to meet new requirements or a complex request (eg. a travel agency can for example use both fly and hotel booking services to respond a customer's query). The number of Web services which increase on Internet makes their discovery and/or composition a complex task. Automating these tasks is a solution which would reduce the cost of the Web services' implementation. Nevertheless, with this intention, we think that it is necessary to enrich the description of the Web services by explicit and comprehensible semantics, which can be used by the machine. [9]

The more valuable gain of SAWSDL lies in opportunities to annotate existing WSDL descriptions in a bottom-up fashion while at the same time only use descriptions of services which are relevant to specific domain requirements. [20]

http://www.docstoc.com/docs/82666005/SAWSDL-tutorial-at-Semantic-Technology-Conference-2007

- http://www.w3.org/TR/sawsdl-guide/
- *SAWSDL* und *Representational state transfer* (*REST*)? [19]
- *SAWSDL* verwenden: Grüner Kasten in [13, S.63], [1] und sehr ausführlich in [21]

3.2 Dynamische Verwendung von semantischen Webservices

The proposed architecture contains a Service Broker to register semantic service descriptions and perform automatic service discovery, and a Service Container for monitoring services and service integration. The process of automatic service replacement is achieved by an interaction of service brokering request, service discovery, and service integration between Service Broker and Service Container. [1, S.410]

The specification and publishing of service descriptions, either semantic or syntactic, is done manually. ... The specification of the semantic service description of a service offer and its publication is manually done by the service vendor. ... Furthermore, service descriptions must be deleted at the Service Broker if they become unavailable. [1, S.416]

Beispiel: Media-Player [1, S.418]

Weiterer Artikel: [3]

4 Anwendung bei komplexen webbasierten Anwendungen

5 Fazit

6 Ausblick

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] Dan Brickley and R.V. Guha. Rdf vocabulary description language 1.0: Rdf schema. Technical report, W3C, Februar 2004.

[3] 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.

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

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

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

[7] W. Dostal, M. Jeckle, and W. Kriechbaum. Semantik und Webservices: Beschreibung von Semantik. *JavaSPEKTRUM*, 04(2):45–49, 2004.

[8] W. Dostal, M. Jeckle, and W. Kriechbaum. Semantik und Webservices: Vokabulare und Ontologien. *JavaSPEKTRUM*, 04(3):51–54, 2004.

[9] 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.

[10] Mathias Habich. Anwendungsbeispiele einer XML Web Service basierten Service-orientierten Architektur. Bachelorthesis, Fachhochschule Furtwangen, 2005.

[11] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Patel-Schneider, and Sebastian Rudolph. Owl 2 web ontology language primer. Technical report, W3C, Oktober 2009.

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

[13] 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.-dec. 2007.

[14] G. Kotonya and J. Hutchinson. A service-oriented approach for specifying component-based systems. In *Commercial-off-the-Shelf (COTS)-Based Software Systems, 2007. ICCBSS '07. Sixth International IEEE Conference on*, pages 150–162, März 2007.

[15] Frank Manola and Eric Miller. Rdf primer. Technical report, W3C, Februar 2004.

[16] Dieter Masak. Ultra large scale systems. In *SOA?*, Xpert.press, pages 297–304. Springer Berlin Heidelberg, 2007.

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

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

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

[20] Tomas Vitvar, Jacek Kopecky, Maciej Zaremba, and Dieter Fensel. Wsmo-lite: lightweight semantic descriptions for services on the web. In *Web Services, 2007. ECOWS '07. Fifth European Conference on*, pages 77 –86, nov. 2007.

[21] Tomas Vitvar, Adrian Mocan, Mick Kerrigan, Michal Zaremba, Maciej Zaremba, Matthew Moran, Emilia Cimpian, Thomas Haselwanter, and Dieter Fensel. Semantically-enabled service oriented architecture : concepts, technology and application. *Service Oriented Computing and Applications*, 1:129–154, 2007. 10.1007/s11761-007-0009-9.

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>
```

```
66      </service>
67  </definitions >
```