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/

Abstract

Die Integration von Dienstangeboten über das Internet als sogenannte *Webservices* ist heutzutage kein Problem mehr. Existierende Standards wie z.B. *Simple Object Access Protocol* (SOAP) und *Universal Description, Discovery and Integration* (UDDI) liefern hierfür bewährte Werkzeuge.

Diese Standards der ersten Generation wurden jedoch im Hinblick auf die Anbindung zustandsloser Webservices entwickelt [7, S. 653] und bilden die Verbindung zwischen zwei Diensten immer individuell ab.

Hieraus ergibt sich jedoch ein Problem bei der Anbindung komplexer Webanwendungen: werden diese mit Hilfe der genannten Techniken angebunden, wird das zur Vermittlung zwischen den jeweiligen Domänenkonzepten nötige Wissen in der Implementierung der Integration *hart kodiert* — andere oder zusätzliche Dienste gleicher Art können deswegen nicht ohne erneuten Aufwand angebunden werden.

Die Suche nach Konzepten für die dynamische Bindung von komplexen Webanwendungen ist die Motivation für diese Fachseminararbeit, in der ich in Abschnitt 2 die Möglichkeit vorstelle, Webservices semantisch zu beschreiben. Neben einer Einführung in das Thema, in der ich auf die theoretischen Aspekte eingehe, analysiere ich in Abschnitt 3 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.

Inhaltsverzeichnis

1 Einleitung

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. [11]

Für sogenannte *Blackbox-Webservices* ist das kein Problem — diese *zustandslosen* Dienste verarbeiten lediglich einfache Daten, d.h. dass der Dienst durch Übergabe eines Datums aufgerufen wird, dieser entsprechend des Aufrufs reagiert und ein Ergebnis zurück liefert. Jeder 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.[8]

Diese Eigenschaft steht in direktem Zusammenhang mit einem wichtigen Trend in der Softwareentwicklung: *Service Oriented Architecture* (SOA) bei der Anwendungen nicht mehr monolithisch aufgebaut werden, sondern in kleinere, in sich geschlossene Komponenten unterteilt werden, die miteinander in einem ein Intra- oder ein Extranet über ihre öffentliche Schnittstellen, die sogenannte *Application Programming Interface* (API), kommunizieren.

Diese Kapselung von Diensten hat auch zum Ziel, eine möglichst hohe Kohäsion innerhalb eines Systems zu ermöglichen — Quellcode soll wenn möglichst 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. [10]

Anbieter *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] (*mite*) 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 notigen 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 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 nicht verwendet.

Das Unternehmen entscheidet sich nun, *mite* zur Zeiterfassung ein zu setzen. Mit dessen öffentlich 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 [7, Seite 653] sind etablierte Standards für Webservices der ersten Generation wie SOAP und 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.

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 Schnitt-

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

stellenbeschreibung zur Folge, ohne dass man konkreten Quellcode anpassen muss.

Im Hinblick auf ökonomische Aspekte kann es sogar von Vorteil sein, die parallele Verwendung mehrere Dienste der gleichen Art zu ermöglichen. Im 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. Patel beschreibt z.B. in [14], dass Unternehmensintranets oft zu unspezifisch für die individuellen Bedürfnisse eines einzelnen Mitarbeiters sind.

Neben der Möglichkeit zur Wahl, erhält man so auch automatisch Redundanz.

Masak hat in [13] 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. Intuitiv verbinden wir damit: „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 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 *Web Services Description Language* (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 8 findet sich eine WSDL für einen Webservice, mit dem sich die aktuell in Google gestellten Fragen abrufen lassen[5]. Sie beschreibt die Entitäten *GetQuestionsAbout*

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

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 [2] 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 [12].

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 Mlöglichkeit geschaffen, Gruppen zusammengehöriger Ressourcen und ihrer Beziehung untereinander zu beschreiben [1].

Mit *OWL* ist es schließlich möglich, Ontologien in Form von Klassen, Eigenschaften, Instanzen und Operationen zu beschreiben [9].

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 ([3], [5], [6] und [4]) beschreiben Wolfgang Dostal, Mario Jeckle und Werner Kriechbaum ausführlich das Konzept der semantischen Webservices.

3 Lösungsansätze

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

4 Anwendung bei komplexen webbasierten Anwendungen

5 Fazit

6 Ausblick

Literatur

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

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

[3] W. Dostal and M. Jeckle. Semantik, odem einer service-orientierten architektur. *JavaSPEKTRUM*, 04(1):53–56, 2004.

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

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

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

[7] 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.

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

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

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

[11] 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.

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

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

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

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"/>
```

```
45          <output message="tns:GetQuestionsAboutSoapOut"/>
46        </operation>
47      </portType>
48      <binding name="PeopleAskServiceSoap" type="tns:PeopleAskServiceSoap">
49        <soap:binding transport ="http://schemas.xmlsoap.org/soap/http" style="document"/>
50        <operation name="GetQuestionsAbout">
51          <soap:operation soapAction="http://peopleask.ooz.ie /soap/GetQuestionsAbout"
52            style ="document"
53            />
54          <input>
55            <soap:body use="literal"/>
56          </input>
57          <output>
58            <soap:body use="literal"/>
59          </output>
60        </operation>
61      </binding>
62      <service  name="PeopleAskService">
63        <port name="PeopleAskServiceSoap" binding="tns:PeopleAskServiceSoap">
64          <soap:address location ="http://peopleask.ooz.ie /soap"/>
65        </port>
66      </service>
67    </definitions >
```