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 *WSDL2* 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 in Abschnittes 2.2 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 **2**

2 Semantische Webservices **4**

2.1 Semantik 4

2.2 Semantische Beschreibung von Webservices 5

3 Dynamische Verwendung von semantischen Webservices **6**

3.1 Dynamische Verwendung von semantischen Webservices 7

4 Anwendung bei komplexen Webanwendungen **7**

5 Fazit **7**

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 werden und es ist sichergestellt, dass die Datenbestände beider Seiten synchronisiert sind. Entscheidet sich das Unternehmen aber zu einem späteren Zeitpunkt, einen anderen oder

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

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

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

weiteren Anbieter zur Zeiterfassung einzusetzen muss diese Anbindung erneut implementiert werden.

Hier wird das Problem offensichtlich: Die bisherige Vorgehensweise, Webservices 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 Webservices 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 [9].

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 [7] sind etablierte Standards für Webservices der ersten Generation wie *Web Services Description Language 2.0* (*WSDL2*), *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 *WSDL2* 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. [7, 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 Webservices maschinenlesbar zur Verfügung gestellt wird, damit eine automatische Vermittlung zwischen einer Aufgabe und zur Verfügung stehenden Webservices 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 [13, S.29], dass Unternehmensintranets oft zu unspezifisch für die individuellen Bedürfnisse eines einzelnen Mitarbeiters sind. Im unserem Intranet-Beispiel könnte das Unternehmen seinen Mit-

arbeitern 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 [12, 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 Webservices

Wie im vorigen Abschnitt beschrieben wurde, fehlt der bisherigen Beschreibung von Webservices der semantische Aspekt. Unter *semantischen Webservices* versteht man solche Webservices, 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 *WSDL2*, 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 9 findet sich eine *WSDL2* für der Webservice „PeopleAsk“[5], mit dem sich die aktuell in Google gestellten Fragen abrufen lassen. Beschrieben werden 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. 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 [4, 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 Webservices

Mit den *Semantic Annotations for WSDL*[6] (*SAWSDL*) hat das *World Wide Web Consortium*[7] (*W3C*) 2007 einen Entwurf zu einem Standard vorgelegt, der es ermöglicht Informationen zu diesen Ontologien maschinenlesbar, als Teil einer *WSDL2*-Datei, auszuliefern.

SAWSDL ist dabei unabhängig von einem semantischen Konzept und liefert nur den Rahmen um andere semantische Frameworks in *WSDL2* zu integrieren — es wird lediglich vorausgesetzt, dass diese Konzepte anhand von URIs identifierziert werden können [10, S.61]. Eine Mögliche Technik zur semantischen Beschreibung ist, *Web Ontology Language* (*OWL*), das auf *Resource Description Framework* (*RDF*) basiert. In *RDF* werden dabei die Entitäten (in *RDF* „Ressourcen“ genannt) mit ihren Attributen und Beziehungen untereinander syntaktisch beschrieben [11]. 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 *RDF Vocabulary Description Language: RDF Schema* (*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 [8].

In *WSDL2* werden Webservices auf einer syntaktischen Ebene beschrieben. Es wird festgelegt, wie die auszutauschenden Nachrichten *aussehen* und an welchen Endpunkten der *API* diese zum Einsatz kommen, nicht jedoch was sie bedeuten. In *WSDL2* werden die abstrakten Elemente *Element Declaration*, *Type Definition* und *Interface* verwendet, um einen Webservice allgemein zu beschreiben, hier spielen die technischen wie das verwendete Protokoll keine

[6] http://www.w3.org/TR/sawsdl/
[7] http://www.w3.org/

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 10 findet sich ein einfaches Beispiel einer *WSDL2*-Datei.

SAWSDL führt nun für diese drei Elemente zusätzlich Attribute ein, um deren semantische Bedeutung zu beschreiben:

- `modelReference` definiert eine Beziehung zwischen einem der definierten Komponente in der *WSDL2* und einem Objekt im semantischen Modell. Diese Attribut kann auf jedes *WSDL2*- 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 Webservices 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.

[10, S.62ff]

Dieses Verfahren kommt auch bei der Verwendung einer gemeinsamen Ontologie zum Einsatz — ein automatischer Vermittler kann dabei die Daten zwischen zwei Schnittstellen mit den *Lifting*-Informationen des Anfragers und den *Lowering*-Informationen des Empfängers vermitteln.

- http://www.w3.org/TR/sawsdl-guide/
- *SAWSDL* und *Representational state transfer* (*REST*)? [14]
- *SAWSDL* verwenden: Grüner Kasten in [10, S.63], [1] und sehr ausführlich in [15]

3 Dynamische Verwendung von semantischen Webservices

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 *WSDL2*-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 *WSDL2*-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*[8] (*WSIF*) der Apache Group. Entsprechend den Elementen einer WSDL-Beschreibung sind innerhalb des *WSIF* Klassen definiert, die mittels der *WSDL2*-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.

3.1 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 Webanwendungen

5 Fazit

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

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

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

[10] 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.

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

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

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

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

[15] 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">
```

```
43      <operation name="GetQuestionsAbout">
44        <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 >
```

Listing 2: WSDL 2.0 Beispiel

```
1   <?xml version="1.0"?>
2   <definitions  xmlns:tns="http://example.com/stockquote.wsdl" xmlns:xsd1="http://example.com/stockquote.xsd" x
3     <types>
4       <schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="http://example.com/stockquote.xs
5         <element name="TradePriceRequest">
6           <complexType>
7             <all>
8               <element name="tickerSymbol" type="string"/>
9             </all>
10          </complexType>
11        </element>
12        <element name="TradePrice">
13          <complexType>
14            <all>
15              <element name="price" type="float"/>
16            </all>
17          </complexType>
18        </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 >
```