Design by Contract zur semantischen Beschreibung von Web Services

Gregor Engels[1], Marc Lohmann[1], Stefan Sauer[2]

[1]Institut für Informatik, [2]Software Quality Lab (s-lab)
Universität Paderborn, 33095 Paderborn
{engels, mlohmann, sauer}@upb.de

Abstract: Die Vision von Web Services ist, dass ein Service Requestor einen Service Provider dynamisch finden und binden kann. Für das Finden eines Web Services müssen die Anforderungen eines Requestors und die Beschreibung eines Services miteinander verglichen werden. Syntaktische Beschreibungen reichen hierfür nicht aus. Eine Möglichkeit zur semantischen Beschreibung von Web Services basiert auf der Technik *Design by Contract*. In diesem Papier führen wir eine UML-basierte Notation für Kontrakte sowie ein Matching-Konzept ein. Damit wird eine automatisierte, semantische Suche nach Web Services möglich.

1. Einleitung

Mit Hilfe von Suchmaschinen (z.B. Google) können Anwender innerhalb von Sekunden Webseiten mit den unterschiedlichsten Diensten finden. Doch nicht nur Benutzer, sondern auch Entwickler von Anwendungen möchten über das Internet zugängliche Dienste und Komponenten nutzbringend verwenden. So können in der Anwendungsentwicklung geeignete Komponenten identifiziert und statisch in die Anwendungssoftware eingebunden werden. Darüber hinaus sollen Anwendungsprogramme in die Lage versetzt werden, geeignete Dienste dynamisch über das Internet zu finden und zu integrieren.

Bisher stehen weder für den statischen noch für den dynamischen Fall geeignete Prozesse zum Finden von Komponenten zur Verfügung, da geeignete Möglichkeiten zur Beschreibung der Dienste und Suchanfragen fehlen. Im statischen Fall wird dieser Nachteil heutzutage durch die Intelligenz der Entwickler ausgeglichen. Um diese Situation zu verbessern, ist eine möglichst präzise, semantische und mechanisch auswertbare Beschreibung des Verhaltens der Dienste und Suchanfragen notwendig. Eine syntaktische Beschreibung von Services, wie sie bisher gewöhnlich verfolgt wird, allein genügt nicht.

Web Services basieren auf Service-orientierten Architekturen (SOA), um die Vision zu realisieren, unterschiedliche Services automatisiert zu finden und zu binden. SOA definiert jedoch nur die Rollen Service Provider, Service Requestor und Discovery Service. Entscheidend sind jedoch die verwendeten Technologien zum Austausch von Informationen zwischen diesen Rollen. Die für Web Services üblicherweise verwendeten Technologien wie WSDL (Web Service Description Language) und UDDI unterstützen allerdings keine Überprüfung, ob die Semantik (das Verhalten) eines angebotenen Service mit dem vom Service Requestor gesuchten Verhalten kompatibel ist.

Ein verbreiteter Ansatz zur semantischen Beschreibung von Operationen und Schnittstellen ist *Design by Contract*. Design-by-Contract-Methoden ermöglichen es, das Verhalten von Schnittstellen bzw. Operationen durch Vor- und Nachbedingungen (Kontrakte) zu beschreiben. Diese Kontrakte werden klassischerweise in Form von logischen Ausdrücken im Quellcode angegeben. Im Rahmen einer modellbasierten Softwareentwicklung schlagen wir allerdings eine graphische Repräsentation der Kontrakte in einer Modellierungssprache vor. Die Vor- und Nachbedingungen werden dazu als UML-Objektdiagramme spezifiziert, welche über einem gegebenen Klassendiagramm getypt sind. Die Modelle können auf unterschiedliche Technologien abgebildet werden. Für Web Services bedeutet das, dass ein Service Provider bzw. Service Requestor seine angebotenen bzw. gesuchten Services mit Hilfe von Kontrakten beschreiben kann. Auf Basis dieser Beschreibungen kann ein Kompatibilitätsbegriff zwischen angebotenen und gesuchten Services entwickelt werden.

2. Beschreibung von Web Services und Anfragen

Grundlage für das automatisierte Finden von Web Services ist eine detaillierte Beschreibung der angebotenen Schnittstellen sowie der Suchanfrage eines Requestors. Neben syntaktischen Beschreibungen von Schnittstellen ermöglicht Design by Contract [Me97] semantische Beschreibungen mittels Vor- und Nachbedingungen. Vorbedingungen beschreiben Eigenschaften, die beim Aufrufen der Operation erfüllt sein müssen. Nachbedingungen beschreiben, welche Ergebnisse nach Ausführung der Operation garantiert werden. Wir verwenden Objektdiagramme zur Darstellung der Vor- und Nachbedingungen auf Modellebene. Im Folgenden zeigen wir an Beispielen, wie Kontrakte zur Beschreibung von Web Services und Suchanfragen verwendet werden können und erläutern, wann eine Servicebeschreibung und Suchanfrage miteinander kompatibel sind.

Abbildung 1 zeigt ein einfaches Klassendiagramm für das Bestellen von Büchern. Ein Service für Bestellungen von Büchern kann durch den in Abbildung 2 (oben) dargestellten Provider-Kontrakt beschrieben werden. Der Kontrakt besagt, dass der Service als Eingabe Daten über das zu bestellende Buch, die Lieferadresse sowie Kreditkarteninformationen benötigt. Das Ergebnis der Ausführung ist, dass eine Bestellung und eine Rechnung für das bestellte Buch erzeugt wurden. Die Rechnung wird mit der Kreditkarte bezahlt.

In unserem Ansatz besteht ein Kontrakt aus einem Paar von Objektdiagrammen, die über einem Klassendiagramm getypt sind. Die einfache, intuitive Interpretation dieser Kontrakte ist, dass Objekte, welche nur auf der rechten Seite des Kontrakts vorhanden sind, neu erzeugt werden. Objekte, welche nur auf der linken Seite des Kontrakts vorhanden sind, werden gelöscht. Objekte, welche sowohl auf der linken als auch der rechten Seite des Kontrakts auftauchen, werden bei der Ausführung des Services nicht verändert. Mit Hilfe dieser graphisch spezifizierten Kontrakte wird es möglich, die Semantik verschiedenster Services über einem gegebenen Klassendiagramm zu beschreiben, wie z.B. Services, die Bücher zu einer Bestellung hinzufügen oder aus einer Bestellung löschen.

Im Gegenzug muss ein Requestor die Semantik eines gesuchten Services beschreiben. Auch hierzu können Kontrakte verwendet werden, welche über einem Klassendiagramm

getypt sind. Die linke Seite eines Requestor-Kontrakts beschreibt, welche Informationen der Requestor an einen Web Service zu schicken bereit ist. Die rechte Seite beschreibt die Situation, welche ein Requestor durch den Aufruf eines Services erreichen möchte. Abbildung 2 (unten) zeigt einen Requestor-Kontrakt für die Suche nach einem Buchhändler. Dieser Kontrakt drückt aus, dass ein Requestor Informationen über ein Buch, seine Lieferadresse, seine Kreditkarte sowie seine Kontoinformationen bereitstellen kann. Der Requestor sucht nach einem Service zur Bestellung von Büchern, wobei es dem Requestor egal ist, ob er mit Kreditkarte oder per Lastschrift bezahlt. Offensichtlich erfüllt der obige Provider die Anforderungen des Requestors. Der Requestor hat lediglich der Vorbedingung zusätzliche Informationen hinzugefügt (*BankAccount*), ohne eine Bedingung an die Bearbeitung dieser zusätzlichen Informationen zu stellen.

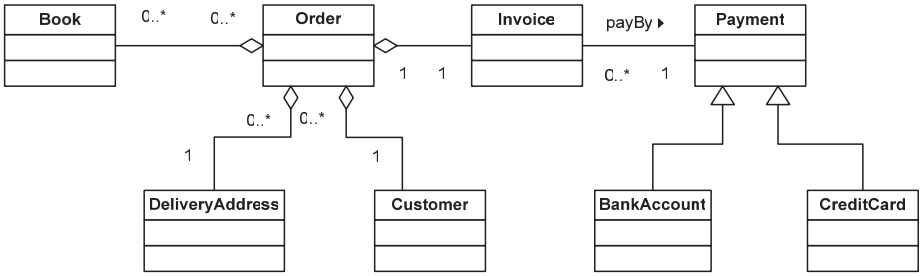

Abbildung 1: Klassendiagramm für das Bestellen von Büchern

Um bestimmen zu können, wann ein Provider die Anforderungen eines Requestors erfüllt, müssen deren Kontrakte miteinander verglichen werden. (Zur Vereinfachung nehmen wir an, dass beide Kontrakte über demselben Klassendiagramm getypt sind – andernfalls sind entsprechende Transformationen erforderlich). Der Versuch, diese Abhängigkeiten auf einfache Subgraph-Beziehungen zurückzuführen scheitert jedoch. Zwar ist die Vorbedingung des Provider-Kontrakts ein Subgraph der Vorbedingung des Requestor-Kontrakts, d.h., der Requestor stellt alle zur Ausführung des Services benötigten Informationen (und eventuell zusätzliche) zur Verfügung. Jedoch ist die Nachbedingung des Requestor-Kontrakts kein Subgraph der Nachbedingung des Provider-Kontrakts, obwohl der Provider durchaus die Anforderungen des Requestors erfüllt, da der Requestor keine Anforderungen an die zusätzlich bereitgestellten Objekte liefert.

Um dennoch ein automatisiertes Matching basierend auf Subgraph-Relationen zu ermöglichen, berechnen wir in einem Zwischenschritt ein Objektdiagramm, welches nur die *Effekte* eines Kontrakts beschreibt. Zur Berechnung der Effekte werden die Vor- und die Nachbedingung miteinander verglichen und alle Objekte aus der Nachbedingung entfernt, die nicht verändert werden. Die grau schattierten Objekte in Abbildung 2 zeigen die berechneten Effekte der Kontrakte. Das Matching ist erfolgreich, wenn die dargestellten Subgraph-Relationen gültig sind. Damit kann ein Requestor einen Provider aufrufen, wenn die Vorbedingung des Requestors stärker oder identisch zur Vorbedingung des Providers ist und wenn die Effekte des Requestors schwächer oder identisch mit den Effekten des Providers sind. Das bedeutet, die Vorbedingung des Providers ist erfüllt, wenn die Vorbedingung des Requestors erfüllt ist, und die Effekte des Requestors sind erfüllt, wenn die Effekte des Providers erfüllt sind. Das vorgestellte Matching-Konzept

haben wir auf unterschiedliche Arten formalisiert. In [HHL04] haben wir eine mengentheoretische Formalisierung angegeben; in [He03] haben wir eine operationale Interpretation der Kontrakte mit Graphtransformationsregeln gewählt.

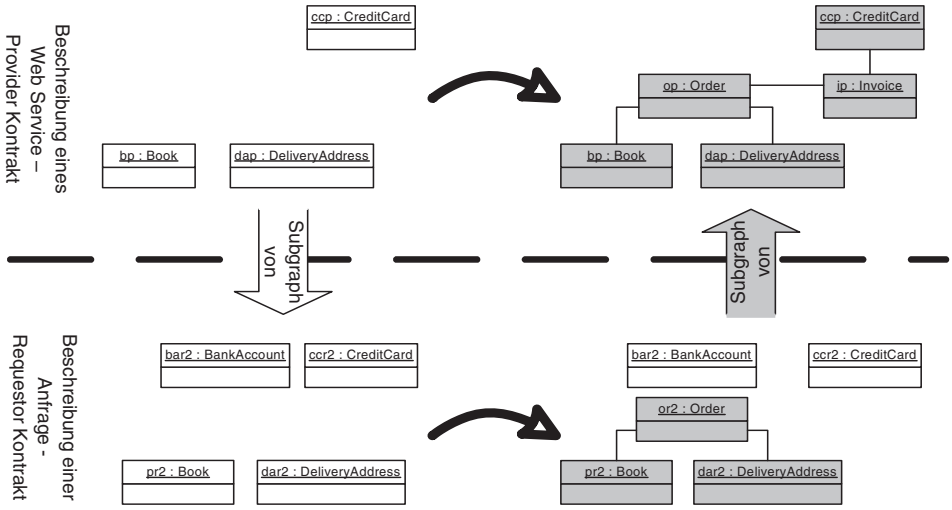

Abbildung 2: Kontrakte zur Beschreibung von Web Services

3. Implementierung

Zur praktischen Umsetzung der zuvor beschriebenen Konzepte haben wir eine Werkzeugkette [Se03] entwickelt, mit der Kontrakte zur Beschreibung von Web Services und Service Requests visuell erstellt werden können. Das Matching-Konzept wurde mit Hilfe existierender Semantic-Web-Sprachen umgesetzt und implementiert.

Zur Modellierung der Kontrakte verwenden wir das Werkzeug AGG (The Attributed Graph Grammar System) [Ta04]. Mit AGG können über einem Typgraphen getypte Graphtransformationsregeln modelliert werden. In unserem Fall entspricht der Typgraph einer abstrakten Repräsentation eines UML-Klassendiagramms bzw. einer Ontologie. Zur Darstellung der Ontologie verwenden wir die Semantic-Web-Sprache DAML+OIL als RDF-basiertes (Resource Description Framework) Dateiformat. Die mit AGG modellierten Graphtransformationsregeln entsprechen Kontrakten, die über einer Ontologie getypt sind. Zur Übersetzung zwischen dem Dateiformat von AGG und der Repräsentation der Kontrakte in DAML+OIL haben wir entsprechende Java-Anwendungen entwickelt.

Die DAML+OIL-Repräsentation der Ontologie sowie der Vor- und Nachbedingung basiert auf RDF-Graphen. Damit kann unser oben beschriebenes Matching-Konzept für Kontrakte auf das Matching von RDF-Graphen zurückgeführt werden. Zur Berechnung der oben beschriebenen Subgraphen-Relationen verwenden wir das Semantic-Web-Framework Jena von HP, insbesondere die RDQL-Implementierung von Jena. RDQL

(RDF Data Query Language) ist eine Anfragesprache für RDF zur Spezifikation von Graph-Pattern, welche in einem Host-Graphen gesucht werden sollen. Zur Berechnung der Graph-Pattern und Host-Graphen geht der Algorithmus wie folgt vor: Als erstes wird aus der Vorbedingung des Provider-Kontrakts ein Graph-Pattern und aus der Vorbedingung des Requestor-Kontrakts ein Host-Graph erstellt. Wenn das Graph-Pattern in dem Host-Graphen gefunden wird, stellt der Requestor alle für den Aufruf des Providers benötigten Informationen zur Verfügung. Dann berechnet der Algorithmus die Effekte der Kontrakte und erstellt aus den Effekten des Provider-Kontrakts ein Graph-Pattern und aus den Effekten des Requestor-Kontrakts einen Host-Graphen. Diese werden verwendet um zu testen, ob der Provider alle vom Requestor benötigten Informationen erstellt. Sind beide RDQL-Anfragen erfolgreich, ist ein Matching gefunden.

4. Zusammenfassung und Ausblick

Wir haben einen UML-basierten Ansatz zur Beschreibung der Semantik von Web Services und Service Requests mit Kontrakten (Design by Contract) entwickelt. Die Vor- und Nachbedingungen der Kontrakte sind mit Objektdiagrammen beschrieben, die über einem Klassendiagramm getypt sind. Das Klassendiagramm gibt einen Sprachgebrauch vor. Auf Basis dieser Verhaltensbeschreibung haben wir einen Kompatibilitätsbegriff eingeführt, der sich auf die Überprüfung von Sub-Graphen zurückführen lässt.

Der Kompatibilitätsbegriff berücksichtigt strukturelle Veränderungen, beispielsweise das Löschen oder Erzeugen eines Objekts. Die Fokussierung auf die strukturellen Veränderungen ermöglicht eine effektive Berechnung des Matching zwischen Beschreibungen von Services und Service Requests. Es besteht darüber hinaus die Möglichkeit zur Erweiterung des Ansatzes z.B. um logische Ausdrücke zur Beschränkung der Wertemengen von Objektattributen, wodurch die Berechnung des Matching jedoch komplizierter wird. Für die Kompatibilitätsprüfung beim Discovery Service ist das nach unseren Erkenntnissen aber nicht erforderlich, da als Ergebnis nur potentielle Service Provider geliefert werden. Ein Service Requestor muss zusätzliche Schritte unternehmen, um unter den potentiellen Service Providern einen geeigneten Service auszuwählen. Da bei dieser Auswahl nur eine kleine Anzahl von Kandidaten überprüft werden muss, können in dieser Situation weitere Kriterien zur Berechnung des Matching angewendet werden.

Literaturverzeichnis

[Me97] Meyer, B: Object-Oriented Software Construction - Second Edition. Prentice-Hall,1997.

[HHL04] Hausmann, J.H.; Heckel, R.; Lohmann, M.: Model-based discovery of Web Services. In Proceedings of the IEEE International Conference on Web Services, 2004; S. 324-331.

[He03] Heckel, R.; A. Cherchago; Lohmann M.: A Formal Approach to Service Specification and Matching based on Graph Transformation. In Workshop on Web Services and Formal Methods. 2003. Pisa, Italy.

[Se03] Sevilmis, N.: Grafische Reaktionsregeln zur Beschreibung der Semantik von Web Services, Diplomarbeit, Universität Paderborn, 2003.

[Ta04] Taentzer, G. AGG: A Graph Transformation Environment for Modeling and Validation of Software. In Applications of Graph Transformations with Industrial Relevance: Second International Workshop. 2004. Springer-Verlag Heidelberg.