

Kontraktbasiertes Black-Box Testen von Webservices

Michael Averstegge
Fakultät für Mathematik und Informatik
FernUniversität Hagen
58084 Hagen
Germany
michael@averstegge.de

1 Einleitung

Mit der Webservice-Technologie erhält der zur Zeit bei weitem bekannteste und verbreitetste Umsetzungsversuch des SOA-Paradigmas verstärkten Einzug in die Praxis ([16], [14], [10], [25]). Durch die plattformunabhängige und einfache Kombinierbarkeit von lose gekoppelten, auf standardisierter Webtechnologie basierenden Komponenten scheint man dem Ziel frei kombinierbarer, wiederverwendbarer und gut wartbarer Software ein Stück näher gekommen zu sein.

Typisch für SOA ist das dynamische Binden neuer Services zur Laufzeit ([3]). Damit entstehen zur Laufzeit Zustände, die zur Designzeit nicht definierbar und zur Entwicklungszeit nicht testbar sind. Auch der klassische Integrationstest kann auf Grund der dynamischen Architektur von komplexen Webservices nicht oder nur sehr unzureichend durchgeführt werden ([26], [13]).

Die WSDL-Spezifikation ([24]) definiert die Webservice Schnittstelle rein syntaktisch, so dass nur die Signatur der Operationen und die Typen der Ein- und Ausgabedaten definiert werden. Die aus syntaktischen Verträgen für den Black-Box Test abgeleiteten Testdaten sind in ihrer Qualität, Fehlerverhalten des zu testenden Webservices (WSUT) zu provozieren, beschränkt ([2], [1], [8], [5]).

Der Mangel an semantischer Information in der Webservice Schnittstellenbeschreibung WSDL ist vielfach in der Literatur adressiert worden (u.a. in [4] und [7]). Im Bereich der Komponentenforschung ist die Forderung nach einer semantischen Spezifikationsebene für funktionale und nicht-funktionale Eigenschaften verbreitet ([22]). Es wird sogar behauptet, dass Webservices nicht nur den *Mangel an Information*, sondern auch den Mangel an angemessenen Methoden, Techniken und Testwerkzeugen aus dem Bereich der Komponententechnologie geerbt hätten ([6]).

Aus den ausgeführten Gründen fordern wir, dass Webservices mit semantischen Informationen Auskunft über ihre funktionalen Eigenschaften geben sollen. Neben der ihren Einsatz bereits hinreichend rechtfertigenden dokumentatorischen Eigenschaft können Verträge zur Laufzeit von signifikant wertsteigernden, qualitätssichernden Maßnahmen genutzt werden, nicht zuletzt weil ihr hoher Automatisierungsgrad von wirtschaftlicher Bedeutung ist. Hierzu zählen die automatisierte Ableitung, Durchführung und Auswertung von Testfällen, QS-Überwachung durch Monitoring und Erhöhung der Robustheit durch Kontraktüberprüfung.

In dem von uns vorgestellten Ansatz wird der Monitor neben der Messung von QoS-Metriken ([21]) zur Prüfinstanz auf vertragskonforme Nutzung und Verhalten der WSUT erweitert. Ferner werden in der Kontraktspezifikation Sprachmittel von Eiffel eingeführt, wie etwa das 'retry' ([17]), welches im Zusammenhang mit einer der geforderten Eigenschaften zur Sicherung der Atomizität von transaktionalen, komplexen Webser-

vices – der *retriability* ([23]) – von unmittelbarer Bedeutung ist.

Der Automatisierungsgrad qualitätssichernder Maßnahmen steigt singnifikant durch den Einsatz von Verträgen zur Erzeugung von Testdaten und Mockups. Letzteres Verfahren nutzt den Vertrag funktional, um die Testbarkeit komplexer Webservices sicherzustellen.

Wir untersuchen die Möglichkeiten des automatisierten, nicht-invasiven Black-Box Tests unter Einbezug des *Design by Contract* Paradigmas ([17], [18]) und lernender Verfahren ([11], [19], [20]). Mit nicht-invasiver Technik soll der Eingriff in den zu testenden Dienst auf Code-Ebene verhindert werden und zugleich die funktionale Flexibilität des Proxi-Musters ([12]) genutzt werden, welches bereits in der Vergangenheit hohe Akzeptanz im professionellen Bereich erhalten hat ([9], [15]).

2 Ansatz

Wir wollen unseren Ansatz anhand eines kleinen, einfachen Beispiels skizzieren: der Suche eines Reiseagenten nach einem Flug. Die WSUT `flightagent` stellt u.a. die Funktion `flightsearch` mit den Parametern:

1. `ftmin`: frühester Zeitpunkt des Abflugs,
2. `ftmax`: spätester Zeitpunkt des Abflugs,
3. `fbmin`: frühester Zeitpunkt des Rückflugs,
4. `fbmax`: spätester Zeitpunkt des Rückflugs,
5. `expticket`: Anzahl der Personen über 2 (≥ 2) Jahren,
6. `chpticket`: Anzahl der Personen unter 2 (< 2) Jahren und
7. `prmax`: Obergrenze für die Summe der Ticketpreise

zur Verfügung.

2.1 Nicht-Invasivität

Um das Kriterium der Nicht-Invasivität zu erfüllen, erzeugen wir in einem ersten Schritt eine intermediäre Schicht zwischen Aufrufer (client) und WSUT. Diese Schicht ist wiederum ein Webservice, der die Funktionalitäten der WSUT in validierender Absicht wrapped: den VWS. Die VWS-Architektur wird in Abbildung 1 dargestellt.

In den PRE- resp. POST-Bereich werden vom VWS dynamisch zur Laufzeit Javamodule geladen, die dem Visitor-Pattern ([12]) entsprechende und für unsere Zwecke definierte Schnittstellen enthalten.

2.2 Vollständiger Vertrag vs. interessierender Vertrag

Wir legen axiomatisch fest, dass ein Vertrag genau dann vollständig ist, wenn der menschliche Benutzer ihn hierzu deklariert hat. Der Vertrag wird durch den *interessierenden Vertrag* approximiert. Das Approximieren durch Bildung des *interessierenden Vertrages* bezeichnen wir als *Lernen von Verträgen*.

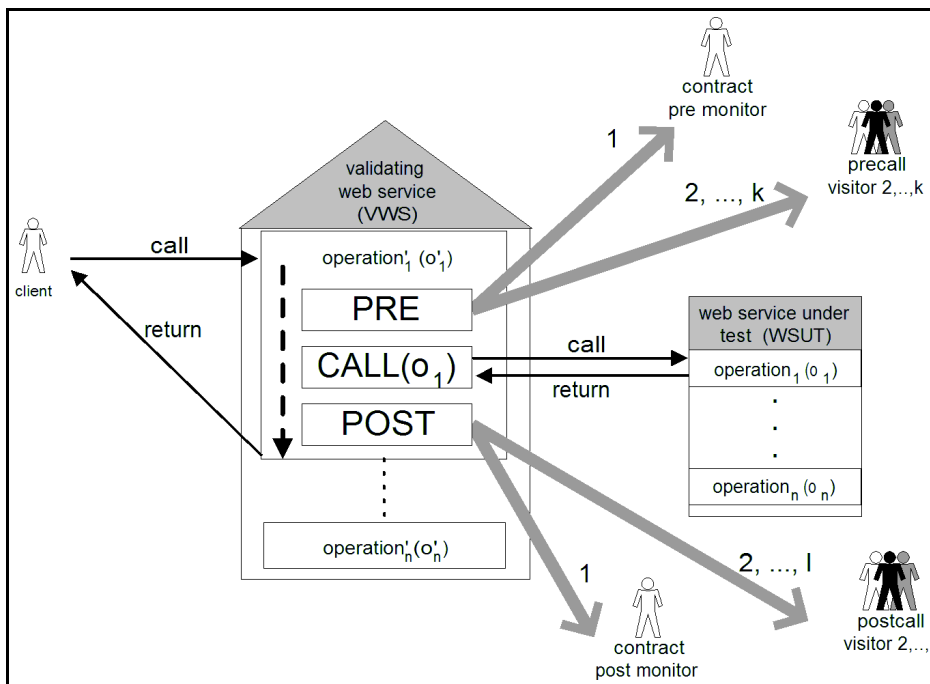


Abbildung 1: VWS Architektur

Als Vorbedingung $prefs := pre(gc(flightsearch))$ formulieren wir:

$$\begin{aligned}
 & (ftmin \geq self.vws.today() \text{ and } ftmin \leq ftmax \text{ and} \\
 & \quad ftmin \leq fbmin \text{ and } fbmin \leq fbmax) \\
 & \quad \text{and} \\
 & (expticket > 0 \text{ and } chpticket \geq 0) \\
 & \quad \text{and} \\
 & (prmax > 0).
 \end{aligned} \tag{1}$$

Das VWS-Framework generiert aus der Vorgabe $prefs$ u.a. den in Abbildung 2 aufgeführten Testdatenvektor 1 (zur besseren Nachvollziehbarkeit sind einige konkrete Werte durch in \langle und \rangle eingeklammerte, abstrakte Ausdrücke beschrieben worden).

Formale Parameter	Testdatenvektor 1 ($prefs$)	Testdatenvektor 2 ($prefs_{ic}$)
ftmin	2006-12-27	2006-12-27
ftmax	2006-12-27	2006-12-27
fbmin	2006-12-27	2007-01-01
fbmax	2006-12-27	2007-01-01
expticket	\langle UntereGrenze \rangle	1
chpticket	\langle ObereGrenze \rangle	1
prmax	1	$\langle 42 * expticket \rangle$

Abbildung 2: Testdaten flightsearch

Durch Auswertung von Laufzeitinformationen entdeckt das VWS-System, dass eine lineare Relation zwischen `prmax` und `expticket` mit einem Faktor besteht, der größer als das neutrale Element der Multiplikation ist und schlägt als Kandidaten für die untere Domaingrenze von `prefs` folgenden Konjunktionsterm vor:

```
prmax >= const42 * expticket.
```

`const42` erhält einen Wert größer 1.

Dies ist der Zeitpunkt, an dem das VWS-System zur Stärkung der Qualitätssicherung assistiert und den Benutzer durch die Aufgabe der Entscheidung zur Übernahme des Terms in die Vorbedingung zur Reflexion veranlasst. In unserem vereinfachten Beispiel wird die Preisuntergrenze von 0-Euro Flügen durch das Produkt aus Gebühren/Flug (`fee`) und `expticket` gebildet. Da `fee` eine variable Größe ist, muss eine Selektionsmethode `getFee()`, die den Zustand der WSUT nicht verändert („purely applicative“, [17, S. 400–403]), zur Verfügung stehen. Mittels `getFee()` kann der Term wie folgt exakter formuliert werden:

```
prmax >= getFee() * expticket.
```

Die Qualität dieses Testdatums ist signifikant größer als im Testfall 1, denn es liefert einen aus Laufzeitinformationen gewonnenen Wert, der als kleinster Wert zu einem positivem Ergebnis geführt hat. Die Betonung liegt darauf, dass dieser Wert erst nach Vertragsformulierung gefunden wurde und somit zu einer nachträglichen Korrektur der Vertragsformulierung, d.i. Lernen des *interessierenden Vertrages*, veranlasste.

Es gibt in unserem Beispiel noch mehrere Abhängigkeiten, die wir noch nicht erfasst haben und die wir hier auf Grund der Kürze nicht angeben können. Wir verweisen auf ein unveröffentlichtes Paper, das zum Workshop ausgegeben wird. Es wird schließlich folgender *interessierender Vertrag* `prefs_ic` gelernt:

```
(ftmin >= self.vws.today() and ftmin <= ftmax and
 ftmin <= fbmin + 5 and fbmin <= fbmax)
 and
(expticket > 0 and chpticket >= 0 and
 expticket >= chpticket)
 and
(prmax >= self.getFee() * expticket). (2)
```

Wir wollen einen hohen Grad der Automatisierung erzielen, damit auch dort getestet werden kann, wo aus Kostengründen nicht oder nur wenig getestet wird. Wir wollen betonen, dass selbst Testdaten, die in einem hochautomatisierten (Lern-)Prozess gewonnen werden, von Nutzen sind, da sie den prinzipiellen a posteriorischen Nachteil des „späten Testens“ durch induktive Auswertung „später Information“, über welche der Tester nicht zur Design- oder Entwicklungszeit verfügt, in einen Vorteil wandeln. Wenn man diese Qualität durch Automatisierung „geschenkt“ bekommt, gibt es unserer Einschätzung nach wenig ersichtliche Gründe, dieses Verfahren nicht einzusetzen.

Literatur

- [1] Karine Arnout and Bertrand Meyer. Uncovering hidden contracts: The .net example. *IEEE Computer*, 36(11):48–55, 2003.
- [2] Xiaoying Bai and Wenli Dong. WsdL-based automatic test case generation for web services testing. In *Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05)*, 2005.
- [3] Luciano Baresi, Reiko Heckel, Sebastian Thöne, and Dániel Varró. Style-based modeling and refinement of service-oriented architectures. *Journal of Software and Systems Modelling*, 2005. In press.
- [4] M. Barnett and W. Schulte. Spying on components: A runtime verification technique. In *Proceedings of the OOPSLA 2001*, 2001.
- [5] Antoine Beugnard, Jean-Marc Jézéquel, Noel Plouzeau, and Damien Watkins. Making components contract aware. *Computer*, IEEE Computer Society Press, 32(7):38–45, 1999.
- [6] S. Beydeda and V. Gruhn. State of the art in testing components. In *International Conference on Quality Software (QSIC)*. IEEE Computer Society Press, 2003.
- [7] Philippe Collet. On contract monitoring for the verification of component-based systems. In *OOPSLA Workshop on Specification and Verification of Component-Based Systems (OOPSLA'2001)*, 2001.
- [8] Antonio Ruiz Cortés, Miguel Toro, Rafael Corchuelo, and Amador Durán. Automated support for quality requirements in web-service-based systems. In *FTDCS*, pages 48–55. IEEE Computer Society, 2001.
- [9] Sérgio Manuel Serra da Cruz, Maria Luiza Machado Campos, Paulo F. Pires, and Linair Maria Campos. Monitoring e-business web services usage through a log based architecture. In *ICWS*, pages 61–69, 2004.
- [10] Schahram Dustdar and Stephan Haslinger. Testing of service-oriented architectures - a practical approach. In Mathias Weske and Peter Liggesmeyer, editors, *Net.ObjectDays*, volume 3263 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2004.
- [11] Michael D. Ernst. Research summary for dynamic detection of program invariants. In *ICSE*, pages 718–719, 1999.
- [12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software*. 1. Aufl., 4. korrigierter Nachdruck, Addison Wesley, 1996.
- [13] R. Heckel and M. Lohmann. Towards contract-based testing of web services, 2004.
- [14] Hai Huang, Wei-Tek Tsai, Raymond Paul, and Yinong Chen. Automated model checking and testing for composite web services. In *ISORC*, pages 300–307, 2005.

- [15] Wei Ma, Vladimir Tasic, Babak Esfandiari, and Bernard Pagurek. Extending apache axis for monitoring of web service offerings. In *BSN '05: Proceedings of the IEEE EEE05 international workshop on Business services networks*, pages 7–7, Piscataway, NJ, USA, 2005. IEEE Press.
- [16] Alberto Martínez, Marta Pati no Martínez, Ricardo Jiménez-Peris, and Francisco Pérez-Sorrosal. Zenflow: A visualweb service composition tool for bpel4ws. In *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, 2005.
- [17] Bertrand Meyer. *Object-Oriented Software Construction*. Santa Barbara Prentice Hall Professional Technical Reference, 2. edition, 1997.
- [18] Bertrand Meyer. Design by contract and the component revolution. In *Technology of Object-Oriented Languages and Systems (TOOLS 34'00)*., pages 515–517, 2000.
- [19] Jeff H. Perkins and Michael D. Ernst. Efficient incremental algorithms for dynamic detection of likely invariants. In *SIGSOFT '04/FSE-12: Proceedings of the 12th ACM SIGSOFT twelfth international symposium on Foundations of software engineering*, pages 23–32, New York, NY, USA, 2004. ACM Press.
- [20] Harald Raffelt, Bernhard Steffen, and Therese Berg. Learnlib: a library for automata learning and experimentation. In *FMICS '05: Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, pages 62–71, New York, NY, USA, 2005. ACM Press.
- [21] Niko Thio and Shanika Karunasekera. Automatic measurement of a qos metric for web service recommendation. In *Australian Software Engineering Conference*, pages 202–211. IEEE Computer Society, 2005.
- [22] Egon Valentini, Gerhard Fliess, and Edmund Haselwanter. A framework for efficient contract-based testing of software components. In *29th Annual International Computer Software and Applications Conference (COMPSAC'05), Volume 2.*, 2005.
- [23] K. Vidyasankar and Gottfried Vossen. A multi-level model for web service composition. In *ICWS*, pages 462–469, 2004.
- [24] W3C. *Web Services Description Language (WSDL) 1.1*, 2001, <http://www.w3.org/TR/wsdl> (last visited (2006.05.01)).
- [25] Jian Yang. Web service componentization. *Commun. ACM*, 46(10):35–40, 2003.
- [26] Jia Zhang and Liang-Jie Zhang. Criteria analysis and validation of the reliability of web services- oriented systems. In *EEE International Conference on Web Services (ICWS' 05)*, pages 621–628, 2005.