Versionierung von Web-Service-Schnittstellen

Christian Wiesing
ORDIX AG
Wiesbaden

Schlüsselworte:

Web-Service, Versionierung, Releasemanagement

Einleitung

Web-Services sind mittlerweile eine der meist genutzten Mechanismen, wenn es um systemübergreifenden Datenaustausch geht. Sie sind in der heutigen Zeit zum Teil schnell implementiert und relativ einfach in die bestehende Architektur einzubinden. Mit aktuellen Frameworks reicht zumeist schon eine einfache Annotation um eine Methode in Form eines Web-Services der Außenwelt zur Verfügung zu stellen. In der folgenden Abbildung ist die mögliche Architektur eines solchen Projektes dargestellt.

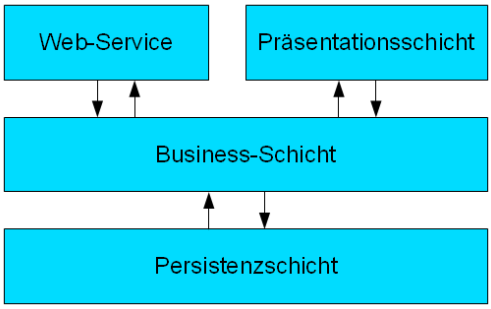

Abb. 1: Einfache Projektarchitektur.

Dennoch kommt es in solchen Projekten häufig zu weiteren Problemen, da es oft nicht ausreicht alleinig einen Web-Service bereit zu stellen. Es muss auch bedacht werden, welche Rolle der Web-Service in einer bestehenden SOA-Architektur spielt und welche Anforderungen an diese Architektur gestellt werden. In einer SOA-Architektur herrscht häufig der Grundsatz, dass eine Web-Service-Schnittstelle, welche einmal produktiv genommen wurde, ohne weiteres nicht mehr geändert oder entfernt werden darf.
Dieser Aspekt wird in vielen Projekten vernachlässigt und erst zu spät in der Entwicklungsphase Beachtung geschenkt. Bei Web-Services reicht es in der Regel nicht aus sich nur um die Implementierung Gedanken zu machen. Auch über eine Versionierung und ein ordentliches Releasemanagement der Schnittstellen sollte sich jeder Verantwortliche des Projekts Gedanken machen, welcher seine Daten über externe Schnittstellen anderen Systemen zur Verfügung stellt.
Im folgenden Vortrag wird anhand von praktischen Beispielen ein mögliches Szenario beschrieben wie Web-Service-Schnittstellen effizient versioniert werden können um dadurch, zum Einen flexibel zu bleiben und zum anderen die Möglichkeit besteht Web-Services nicht anpassen zu müssen.

Änderung der bestehenden Web-Service-Schnittstelle

Früher oder später kommt es in jedem Projekt, mit einer aktiven Web-Service-Schnittstelle, zu neuen Anforderungen bzw. strukturellen Änderungen, welche mittelfristig auch über die Schnittstelle nach außen gegeben werden sollen.
Hierbei ist zunächst abzugrenzen, welche Änderungen auch innerhalb eines bestehenden Web-Services vorgenommen werden können, ohne anhängende Systeme zu beeinflussen. Es ist durchaus möglich dem Web-Service neue Methoden hinzuzufügen. Darüber hinaus können der WSDL auch neue XML-Schema-Typen hinzugefügt werden, solange diese nicht Bestandteil der alten Datentypen sind. Diese Änderungen sind demnach möglich auch ohne eine neue Schnittstelle parallel zur alten bereit zu stellen.
Änderungen wie beispielsweise das Entfernen und Umbenennen von Methoden und Datentypen, sowie die Änderung von Methoden-Signaturen und Rückgabewerten ist jedoch problematisch. Hier ist ein anhängendes System in der Regel dazu gezwungen den zugehörigen Web-Service-Client neu zu generieren und entsprechende Änderungen im Quellcode nachzuziehen. Diese Änderungen sind in der Regel kurzfristig nicht in allen Systemen möglich, was die Notwendigkeit einer zweiten Version des Web-Services verdeutlicht.

Contract First vs. Code First

Vor dem Bereitstellen eines Web-Services stellt sich immer die Frage nach welchem Verfahren dieser entwickelt werden soll. Die meisten Entwickler präferieren in den meisten Fällen den Ansatz „Code First“. Hier fällt, anders als beim Ansatz „Contract First, die zum Teil mühselige Entwicklung der WSDL weg, da diese auf Basis des vorhandenen Quellcodes generiert wird.
Grundsätzlich eignen sich beide Mechanismen für eine saubere Web-Service-Versionierung, dennoch besitzt der Ansatz „Contract First ein paar Vorteile.
Bei einem Web-Service beschreibt die WSDL die angebotenen Datentypen und Funktionen, auf dessen Basis sich viele Systeme die zugehörigen Web-Service-Clients generieren. Daher ist es unerlässlich, dass die WSDL nach der Produktivsetzung nicht mehr geändert wird. Beim Verfahren „Code First“ ist man darauf angewiesen, dass mir das jeweilige Framework immer die identische WSDL-Struktur generiert, was gerade bei Framework-Upgrades sorgfältig geprüft werden sollte. Beim Ansatz „Contract-First“ ist alleinig der Entwickler für die Beibehaltung der bestehenden WSDL-Struktur verantwortlich und erfahrungsgemäß werden WSDL-Dateien in der Regeln nicht ausversehen vom Entwickler angepasst, was bei normalen Java-Objekten bzw. Methoden schneller der Fall sein kann.

Konvertierung der WS-Objekte

Es ist bei einer stabilen Web-Service-Schnittstelle wichtig darauf zu achten, dass keine Business-Objekte direkt durch den Web-Service nach außen gegeben werden, da sich sonst interne Änderungen auch direkt auf dem Web-Service auswirken können. Auf dieses Problem ist vor allem beim Ansatz „Code-First“ zu achten.
Abhilfe schaffen hier spezielle Web-Service-Objekte die durch einen Konverter auf Basis der Business-Objekte erstellt werden. Hierbei ist darauf zu achten, dass diese Objekte wirklich nur die Informationen enthalten, die für andere Systeme auch von Relevanz sind. Dadurch können sie relativ „schlank“ gehalten werden und erzeugen somit keinen überflüssigen Datentransfer.

Nur in dem Konverter werden bei produktiven Web-Service-Schnittstellen noch Änderungen vorgenommen, dabei bleiben die Web-Service-Objekte aber unverändert. Dieses Vorgehen ermöglicht, dass auch komplexe Änderungen innerhalb der Business-Logik in die „alte“ Form des jeweiligen Web-Service konvertiert werden können.
In der folgenden Grafik wird das Verfahren noch einmal schematisch skizziert:

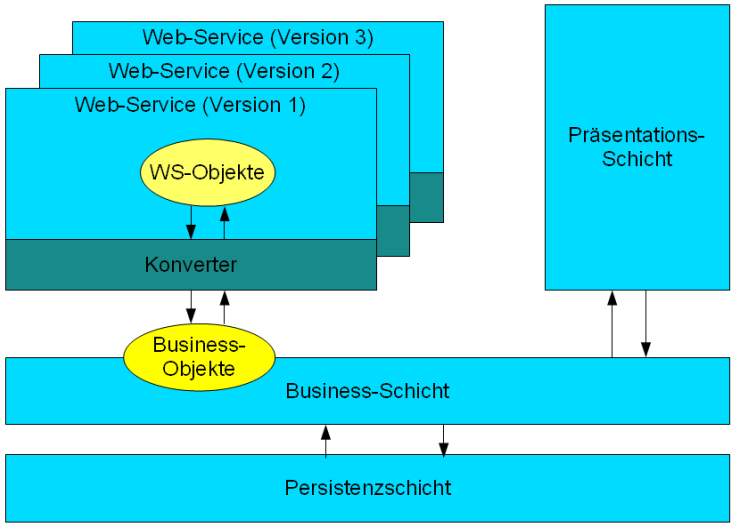

Abb. 2: Konvertierung der Objekte .

Arten der Versionierung

Grundsätzlich unterscheidet sich die Versionierung in zwei Arten. Einmal die Package-Versionierung und zum anderen die sogenannte Endpoint-Versionierung. Unter der Package-Versionierung wird das „einfache“ duplizieren des Quellcodes innerhalb des Verzeichnisbaumes eines Projektes bezeichnet. Hierbei existieren die identischen Klassen, mit unterschiedlichen Implementierungen, innerhalb der Package-Struktur:

```
de.ordix.webservice.version1
          - ServiceKlasse1.java
          - ServiceKlasse2.java
          - WebService.wsdl
de.ordix.webservice.version2
          - ServiceKlasse1.java
          - ServiceKlasse2.java
          - WebService.wsdl
```

Innerhalb des neu erzeugten Package wird zusätzlich die Zugriffsinformation des jeweiligen Web-Services angepasst, damit beide Services parallel existieren und aufgerufen werden können. Dieses Prinzip macht aber die Abgrenzung der Schnittstellen voneinander schwierig und den Quellcode über lange Sicht unübersichtlich. Vor allem birgt es das Risiko unbeabsichtigt eine bestehende Schnittstelle zu ändern.
Ein eleganteres Vorgehen ist das Auslagern der Web-Service-Schnittstelle inklusive des Konverters in ein eigenes Projekt. Dieses Vorgehen wird auch Endpoint-Versionierung genannt. Hierbei wird das Anlegen einer neuen Version der Schnittstelle durch ein Versionsverwaltungssystem (wie z. B. CVS oder SubVersion) realisiert. Bei jeder neuen Web-Service-Version wird somit das gesamte Web-

Service-Projekt dupliziert. In diesem Fall ist Beispielsweise jede bestehende produktive Web-Service-Version ein Branch dieses Projektes.

Abb. 3: Endpoint-Versionierung.

Der Hauptzweig bleibt die aktuelle Entwicklungsversion, welche solange geändert werden kann, bis diese produktiv gesetzt wird. Anschließend wird auch auf Basis dieser Version ein Branch erstellt, um sie von der aktuellen Weiterentwicklung abzukapseln. Dadurch ist eine genaue Abgrenzung gewährleistet und ein Quellcode muss nicht unnötig im eigentlichen Hauptprojekt dupliziert werden.
Es ist möglich auf Basis dieses Web-Service-Projektes ein einzelnes EAR bzw. WAR-Archiv zu erzeugen, welches unabhängig von dem bestehenden Projekt auf dem Applikationsserver deployed werden kann.
Der Aufwand bei dieser Vorgehensweise ist Initial relativ hoch, da der Web-Service aus dem bestehenden Projekt herausgelöst werden muss. Im weiteren Entwicklungsverlauf reduziert sich hingegen der Aufwand für die weitere Pflege und Wartung der Schnittstellen.

Isolierung der Services voneinander

In einigen Applikationsservern kann es Konflikte durch das jeweilige verwendete Classloading-Verfahren zwischen den einzelnen Web-Service-Archiven geben. Beim JBoss werden Beispielsweise alle Klassen in einem zentralen Klassen-Pool gehalten. Bei mehreren EARs, die jeweils eine Version eines Web-Services repräsentieren, existieren häufig identische Klassen, da diese zwischen den Versionen häufig nur leicht abgeändert werden.
Damit also keine Konflikte beim Classloading auftreten, ist es hier wichtig die einzelnen EARs voneinander zu isolieren. Das bedeutet, dass die Web-Services sich gegenseitig nicht sehen und ihren eigenen Klassen-Pool besitzen, aber dennoch alle gemeinsam auf die zentrale Businesslogik zugreifen können. Das beschriebene Classloading-Verhalten beim JBoss wird in der folgenden Grafik kurz aufzeigt.

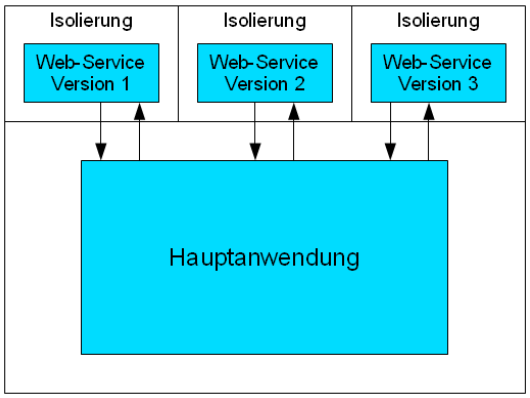

Abb. 4: Isolierung innerhalb des JBoss

Damit das jeweilige Web-Service-Projekt von den anderen isoliert werden kann, muss dem jeweiligen EAR eine sogenannte jboss-app.xml mit folgendem Eintrag mitgeliefert werden:

```
<jboss-app>
  <loader-repository>
  com.example:archive=meinWebService_v1
  <loader-repository-config>
        java2ParentDelegation=false
  </loader-repository-config>
  </loader-repository>
</jboss-app>
```

Mit Hilfe dieses Eintrages wird dem JBoss mitgeteilt, dass er das zugehörige EAR in einem eigenen Kontext ablegt. Somit sind die zugehörigen Klassen für andere Projekte nicht sichtbar. Dennoch kann der isolierte Web-Service auf den allgemeinen Klassen-Pool zugreifen.

Releasemanagement

Der Aufwand für die parallele Pflege von mehreren Versionen einer Web-Service-Schnittstelle ist nicht zu unterschätzen. Daher gehört zu jedem Lebenszyklus einer Schnittstelle auch die Abkündigung dieser. Mit der Zeit wird es immer schwieriger die alte Web-Service-Version konstant zu halten, da sie in mehreren Projekten häufig vielen strukturellen Anforderungen unterliegen und dadurch die Konvertierungslogik von Zeit zu Zeit komplexer und schwieriger zu pflegen sein wird.
Es sollte zunächst eine maximale Anzahl parallel existierender und zu wartender Web-Service-Schnittstellen definiert werden. Der Wert sollte in der Praxis nur selten über drei liegen, damit der Verwaltungs- und Pflegeaufwand nicht zu groß wird.
Bei Erreichen der maximalen Anzahl an Schnittstellen, sollte die älteste Schnittstellen-Version abgekündigt werden. Das bedeutet, anhängende Systeme über deren Auslaufen zu informieren, mit der Bitte, die aktuellste Version des Web-Services zu verwenden und die jeweiligen Änderungen im eigenen Quellcode vorzunehmen. Hierbei wird in der Regel ein Zeitraum angegeben bis die Schnittstelle endgültig aus der Wartung und somit vom System genommen wird.
Nur so lässt sich gewährleisten, flexibel auf Änderungen reagieren zu können und dennoch anhängende Systeme nicht mit zu vielen Umstiegen auf neue Versionen zu belasten.

Logging der Schnittstelle

Ein wichtiger Aspekt in einer funktionierenden SOA-Architektur ist genau zu wissen, welches System auf die jeweiligen Web-Service-Schnittstellen zugreift. Daher ist in größeren SOA-Systemen ein gezieltes Logging aller Zugriffe notwendig.
Für einen einfachen Überblick reichen hier bereits vorhandene Logging-Verfahren, wie beispielsweise die Access-Logs beim Apache bzw. Tomcat aus. Um erweiterte Informationen zu erhalten, können dem Web-Service gezielt Benutzerinformationen übergeben werden. Diese Informationen dienen häufig weniger der Authentifizierung, sondern vielmehr der Informationssammlung welcher Benutzer auf den jeweiligen Web-Service zugreift, um diesen beispielsweise bei einer Abkündigung der Schnittstelle frühzeitig zu informieren. Nur dadurch kann eine stabile SOA-Architektur gewährleistet werden.

Schlussbetrachtung

Der Vortrag zeigt, dass die Bereitstellung von Web-Service-Schnittstellen mehr Aufwand verursacht als auf den ersten Blick ersichtlich ist. Es reicht häufig nicht aus sich um die reine Implementierung Gedanken zu machen, auch bei strukturellen Änderungen muss dafür gesorgt werden, dass anhängende Systeme problemlos weiter auf die bestehenden Schnittstellen zugreifen können, ohne selbst Änderungen am eigenen Quellcode vornehmen zu müssen.
Anhand eines in der Praxis erprobten Vorgehens wird gezeigt, wie sich eine Versionierung von Web-Service-Schnittstellen realisieren lässt, um dauerhaft flexibel mit ihnen agieren zu können und welche Gedanken sich ein Architekt zum Releasemanagement der Schnittstellen machen sollte.

Kontaktadresse:

Christian Wiesing
ORDIX AG
Kreuzberger Ring 13
D- 65205 Wiesbaden

Telefon: +49 (0) 611 77840 00
Fax: +49 (0) 180 1 67349 0
E-Mail info@ordix.de
Internet: www.ordix.de