

Implementierung von Web Services in Oracle- Datenbankanwendungen

Wie wir legacy Oracle Forms Anwendungen in der
modernen Web-Entwicklung Welt integrieren
können

Technischer Artikel, Februar 2011

1. Einführung.....	3
2. Ein paar Worte über Oracle Forms Anwendungen	3
3. Web Services aufrufen	4
3.1. Aufrufen von Web Services vom Oracle Forms.....	4
3.1.1. Das Synchronous/ Asynchronous Dilemma.....	6
3.1.2. Verwenden des Forms 11g Event Objekts.....	6
3.2. Aufrufen von Web Services aus der Oracle Datenbank	7
4. Bereitstellung von Web Services	9
4.1. Warum Forms Business-Logik in die Datenbank migrieren?..	9
4.2. Datenbank Funktionalität bereitstellen.....	10
4.2.1. PL/SQL Web Services.....	10
4.2.2. BPEL-Prozesse	12
4.2.3. Native DB Web -Services.....	13
5. Fazit.....	15

1. Einführung

Datenbearbeitungsanwendungen existieren seit dem Anfang der IT und unterstützen heutzutage die meisten unserer Geschäftsprozesse. Während dieser Zeit wurden die Anwendungen weiterentwickelt, um sich den ständig ändernden Geschäftsanforderungen anzupassen. In der Bestrebung nach schneller Entwicklung, Anpassung, Bugfixing und Patching, ist die typische Oracle-Datenbankanwendung größer und größer geworden in den Jahren und besteht nun aus einer Mischung von alten wie neuen Komponenten, geschrieben in verschiedenen Programmiersprachen und ausgelegt mit unterschiedlichen Architekturen. Eine unserer größten Herausforderung wird es sein, das Beste aus all diesen Komponenten herauszuziehen, neu zu strukturieren und in modernen, modularen Anwendung interagieren zu lassen, sodass die heutigen und zukünftigen Geschäftsanforderungen flexibel unterstützt werden.



Abbildung 1: Oracle Datenbank Anwendung

Der aktuelle Artikel versucht Möglichkeiten aufzuzeigen, wie eine in die Jahre gekommene Oracle Forms Anwendung zu einem Teil einer modernen Web-Entwicklungswelt wird. Da die Service-orientierten Architekturen zunehmend an Bedeutung gewinnen, möchten wir aufzeigen, mit welchen Mittel externe Anwendungen durch Implementierung von Web Services integriert werden können.

2. Ein paar Worte über Oracle Forms Anwendungen

Oracle Forms ist ein solides und ausgereiftes Framework und wohl die meist verwendete Umgebung für die Entwicklung von Oracle Datenbankanwendungen in den letzten 20 Jahre. Seine Architektur wurde speziell für die schnelle und einfache Entwicklung von komplexen Datenbankanwendungen konzipiert. Doch vor 25 Jahren, als die Forms Architektur entworfen wurde, war die Interoperabilität von Anwendungen und die Kommunikation über Web Services noch kein wirkliches Thema. Deshalb ist es keine leichte Aufgabe, alte Oracle Forms Applikationen so zu modernisieren, dass sie mit externen Anwendungen interagieren:

- **Als Web-Services Verbraucher** besitzen Forms-Anwendungen, im Gegensatz zu modernen ADF oder APEX Anwendungen, keine eigenen Mechanismen, um direkte Verbindungen zu Web-Services herzustellen. Forms kann nur indirekt auf Web Services mit Hilfe von Proxy Diensten oder Datenbank Web Services zugreifen. Beide Lösungen werden wir im Detail betrachten und darstellen, was besser zu unseren Anforderungen passt.
- **Als Web-Services Anbieter:** Wenn wir jedoch Forms-Funktionalitäten (= Business-Logik) als Web-Services öffentlich anbieten wollen, müssen wir den entsprechenden Source-Code in die Oracle Datenbank migrieren, um ihn von dort als Web-Service zu verwenden. Forms kann alleine wegen seiner kompakten Architektur diese Rolle nicht übernehmen. Hier werden wir aufzeigen, wie man in diesem Fall vorgehen kann und warum die Migration von Business-Logik in der Datenbank die Wiederverwendbarkeit des Source-Code erhöht und damit die zukünftigen Migrations- und Modernisierungsschritte unserer Forms-Anwendungen ganz erheblich vereinfacht.

3. Web Services aufrufen

Wie bereits angesprochen, haben wir zwei Möglichkeiten um auf externe Web-Services von Oracle Forms Anwendungen aus zu zugreifen: direkt von Forms mit Hilfe eines Web-Service Proxy-Servers oder von der Oracle-Datenbank aus unter Verwendung seiner Web Service Utilities.

3.1. Aufrufen von Web Services vom Oracle Forms

Wie kann Oracle Forms auf externe Web Services zugreifen? Nun, Forms ist von Natur aus nicht in der Lage direkt mit Web Services zu kommunizieren, jedoch kann es Java-Klassen innerhalb eines Web-Services Proxy kommunizieren. Der Proxy dient hierbei als Vermittler, denn er ist in der Lage, eine Verbindung zum eigentlichen Web-Service zu etablieren, wie in Abbildung 2 dargestellt.



Abbildung 2: Die Kommunikation zwischen Forms und Web Service über Proxy

Um diesen Mechanismus zum Aufruf eines Web-Services in der Forms-Anwendung zu verankern, müssen einige Schritten durchführen:

- **Generierung der Web Service Proxy**
Dieser kann unter Verwendung der Oracle JDeveloper leicht erstellt werden. Man muss lediglich die URL eingeben, wo die Definition des Web Services gefunden werden kann, die sogenannte WSDL (Web Service Definition Language). Dies ist ein XML-Dokument, das alle Informationen über die Operationen, die Adresse des Web Services sowie das Message-Protokoll enthält. Wichtig ist

hier die Wahl der richtigen Java-Version, die zur Kompilierung der Java-Dateien verwendet werden sollten und mit der Forms JRE Java-Version übereinstimmen muss.

- **Einbringen des Web Service Proxy in eine Jar-Datei**

Dies kann ebenfalls mit dem Oracle JDeveloper erfolgen, indem man das Projekt mit dem generierten Web-Service-Proxy als JAR-Datei kennzeichnet.

- **Einstellen der FORMS_BUILDER_CLASSPATH Umgebungsvariable**

Im Registrierungs-Editor, müssen wir die FORMS_BUILDER_CLASSPATH Variable im Oracle-Home der aktuelle Forms-Version einstellen. Der Variablenwert sollte die Adresse beinhalten wo die erzeugte Proxy-JAR-Datei zu finden ist. Auf diese Weise werden die Java Klassen des Web-Service-Proxy für den Form-Java Importer sichtbar.

- **Importieren der Java-Klassen in der Forms-Anwendung**

Hier verwenden wir den Forms Java Importer, um die Proxy Java-Klassen, die mit dem Webdienst kommuniziert sollen, in unsere Forms-Anwendung zu importieren. Diese Klassen werden in der Regel mit WebServiceNamePortClient benannt. Die Java-Importer kreiert in der Forms-Anwendung für die importierten Java-Klasse eine Package-Spezifikation und dessen Body als PL/SQL-Schnittstelle.

- **Einstellen der CLASSPATH Umgebungsvariablen für die Laufzeit**

In der ENV-Datei, die von der Forms-Anwendung zur Laufzeit verwendet wird, müssen wir der CLASSPATH Umgebungsvariablen die Adresse des Proxy-JAR-Datei hinzufügen. Diese Einstellung ist notwendig, um die Proxy-Java-Klassen während der Laufzeit im Zugriff zu haben.

Sobald wir diesen Mechanismus in unserer Forms-Anwendung implementiert haben, sind wir in der Lage, die Funktionalitäten des neu generierten Package zu verwenden und den Web-Service aufzurufen.

```
1 Declare
2   lJsonObject _Java.JObject;
3   lWeather    VARCHAR2(4000);
4   lXml        XMLType;
5 Begin
6   lJsonObject := GlobalWeatherSoapPortClient.New();
7   lWeather    := GlobalWeatherSoapPortClient.GetWeather
8               (lJsonObject,:block2.city, :block2.country);
9   lWeather    := Dbms_Xmlgen.Convert(lWeather, 1);
10  lXml        := XMLType.CreateXml(lWeather);
11  :block2.temperature :=
12      lXml.Extract('/CurrentWeather/Temperature/child::node()',
13                  'xmlns=""').GetStringVal();
14 Exception
15   When Ora_Java.Java_Error Then
16     Message('Unable to call out to Java, ' ||Ora_Java.Last_Error);
17 End;
18 /* Note: currently an Oracle Forms bug is leading to runtime
19 errors when executing the XMLType.CreateXml command. The
20 workaround is to move lines 10-13 to a database function */
```

Abbildung 3: Forms-Code Beispiel für das Einlesen der Temperatur eines bestimmten Ort durch den Aufruf des <http://www.Webservicex.net/globalweather.asmx?wsdl> Web-Service

3.1.1. *Das Synchronous/ Asynchronous Dilemma*

Der Code in Abbildung 3 ist ein Beispiel eines Web Service mit synchronem Aufruf. Hier empfängt der Proxy den Aufruf aus der Forms-Anwendung und sendet diesen weiter an den Webdienst. Kurz darauf erhält er die Antwort vom Web Service und leitet sie zurück zur Forms-Anwendung. Während dieser Abfolge sind die Forms Ressourcen blockiert (locked), die Code-Ausführung wird angehalten, bis die Forms Runtime eine Antwort vom Web Service empfängt. Zur optimalen Auslegung des neuen Web Service, ob synchron oder asynchron, sollten wir uns folgende Fragen stellen:

- Brauchen wir die Antwort des Web Service für die Fortsetzung der Programmlogik? Denn wenn nicht, können wir einen asynchronen Web Service benutzen.
- Wie lange dauert die Ausführung des Web Service? Können wir uns leisten, die Anwendung für diesen Zeitraum zu blockieren?
- Betrachtet man die Ressourcen, die während der Web-Service blockiert werden: werden sie auch für andere Prozesse verwendet?

Die letzten beiden Fragen sind sehr wichtig: denn ein verschachteltes locken vieler Datenbank-Sessions, die versuchen auf die gleichen Ressourcen wie der Web Service zuzugreifen, sollte vermieden werden. Die Lösung wäre: Asynchrone Web Services.

Eine mögliche Architektur für asynchrone Aufrufe würde das Einbinden eines Einweg-BPEL-Prozesses, der Advanced Queuing (AQ) Datenbank-Funktionalität und das neue Forms 11g Event Objekt erlauben.

3.1.2. *Verwenden des Forms 11g Event Objekts*

In unserer neuen Architektur sollte der Proxy nicht die direkte Kommunikation zwischen Forms und einem Web-Service übernehmen, sondern zwischen Forms und einem BPEL-Prozess. Da dieser BPEL-Prozess ein Einweg Prozess ist („shut and forget“), wartet der Web Service Proxy nicht auf die Antwort des BPEL Prozesses. Nach Erhalt der Auftragsbestätigung durch den BPEL-Prozess, lässt der Proxy die Forms Logik weiterlaufen. Der BPEL-Prozess leitet den Anruf an den Webdienst und wartet auf dessen Antwort. Nachdem Empfang übergibt der BPEL-Prozess die Antwort, in diesem Fall als Mitteilung einer Queue, auf die ein Event-Objekt der Forms-Anwendung wartet. Wir haben hier mehrere Möglichkeiten, damit der BPEL Prozess die Antwort sendet:

- direkte Übertragung der Antwort durch einem AQ-Adapter
- indirekt über einen DB-Adapter: bevor wir die Antwort weiter an die Queue senden, können wir zusätzliche Aktionen in der Datenbank mittels eines Datenbank-Triggers, -Funktion oder -Prozedur auszuführen.

Sobald die Antwort in der Queue platziert wird, meldet das Advanced Queuing der Forms Runtime. Der When-Event-Raised Trigger liest beim nächsten Ping des Forms-Applets die Antwort ein¹.

¹ Mehr Details über das Event-Objekt bei:

http://www.oracle.com/technology/sample_code/products/forms/11g/aqinteg/lab1/index.htm#o

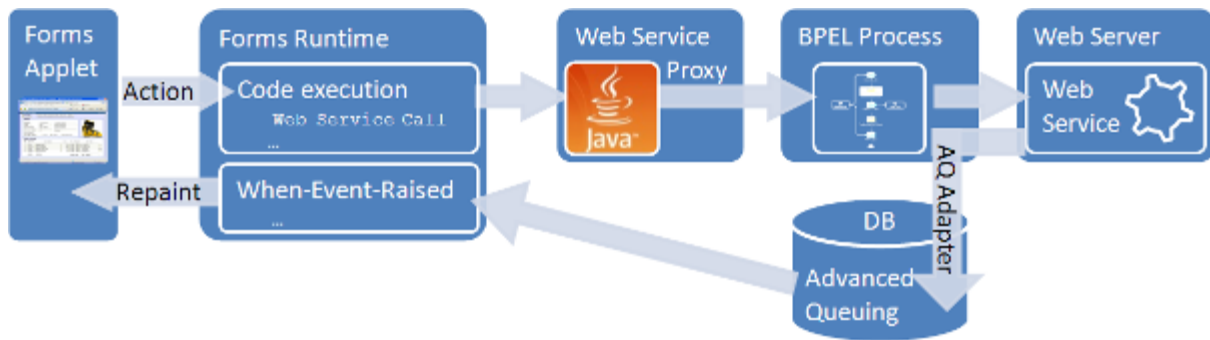


Abbildung 4: Die Kommunikation zwischen Forms und Web-Service über Forms11g Event

Empfehlung zu asynchronen Prozessen: da die Ausführung des Webdienstes einige Zeit dauern kann und die Forms-Anwendung durchaus die Kontrolle über den gestarteten Prozessen verliert, sollte der BPEL-Prozess regelmäßig Nachrichten über seinen Status an die Forms-Anwendung senden. Bei langläufigen Webdiensten bietet sich an, dass der WebService dem BPEL Prozess sofern möglich über seinen Status informiert und der wiederum die Forms-Anwendung benachrichtigt, so behält die führende Forms-Anwendung trotz asynchroner Abarbeitung die Kontrolle.

3.2. Aufrufen von Web Services aus der Oracle Datenbank

Nachdem wir gesehen haben, wie wir Web-Services aus Forms aufrufen, stellt sich die Frage, wie befähigen wir die Datenbank-Funktionalität, wie z.B. die Geschäftslogik, die aus der Forms Anwendung zur Datenbank migriert wurde, um sich vorhandener Web Service zu bedienen? Wie können wir den vorher diskutierten Proxy-Webdienst Aufruf ersetzen? Hier bietet Oracle einige Möglichkeiten an, wie:

- **Java-Lösung:** dieser Lösungsansatz stellt einen Java Web Service-Client in der Datenbank bereit. Allerdings kann der Java Web Service und dessen abhängige Klassen zu Inkompatibilitäten mit vorhandenen Java-Klassen der Datenbank führen, was dann die Datenbankstabilität beeinträchtigen² könnte.
- **UTL_HTTP** – ist verfügbar ab Oracle 7.3.4 und bietet die Möglichkeit HTTP-Aufrufe direkt aus der Datenbank durchzuführen. Es kann durchaus für die Kommunikation mit einem Webdienst verwendet werden, indem man eine HTTP-Anfrage sendet, die eine SOAP-Message für Web-Service enthält und als Antwort wiederum eine HTTP Antwort mit der SOAP-Message erwartet.
- **UTL_DBWS** – ist verfügbar ab Oracle 10g und kann direkt SOAP-Calls verwenden. Im Vergleich zu UTL_HTTP ist es besser auf Web-Services ausgelegt. Die Funktionen und Prozeduren, die in diesem Paket definiert sind, stellen Wrapper für die JAX-RPC dynamische Invocation Interface bereit. Ein Stub (Proxy) wird jedes Mal dynamisch erzeugt, sobald wir auf einen Web-Service über UTL_DBWS zugreifen. Anmerkung: Obwohl UTL_DBWS in einer Oracle 10g DB vorinstalliert ist, ist

² Database Web Service Callout Utility 10.1.3.:

http://www.oracle.com/technology/sample_code/tech/java/jsp/callout_users_guide.htm

die dazugehörige Callout-Utility nicht sofort verfügbar ist und muss nachträglich installiert werden³. In Abbildung 5 sehen wir ein Beispiel zur Anwendung von UTL_DBWS, um einen externen Web Service aufzurufen, der wiederum die Temperatur für einen bestimmten Ort abfragt:

Die letzten beiden dargestellten PL / SQL-Lösungen haben den großen Vorteil, dass sie sich perfekt in eine Oracle-Datenbank-Anwendung integrieren lassen. UTL_DBWS ist neuer und damit auch empfohlen von Oracle. Es enthält viele neue Features, Bugfixes und Patches die nur DBWS Callout Utility verfügbar sind, sofern es um SOAP-Aufrufe geht.

```

1 Function GetWSTemperature ( pCountry   Varchar2,
2                             pCity      Varchar2) Return Varchar2 Is
3 lCall                        Utl_Dbws.Call;
4 lOperationQName             Utl_Dbws.QName;
5 lPortQName                  Utl_Dbws.QName;
6 lStrResp                     Varchar2(32767);
7 lService                     Utl_Dbws.Service;
8 lServiceQName               Utl_Dbws.QName;
9 lXmlReq                      XMLType;
10 lXmlResp                    XMLType;
11 Begin
12     -- Set proxy if necessary
13     -- Utl_Dbws.Set_Http_Proxy('192.168.161.123:3128');
14
15     -- Create a service
16     lServiceQName := Utl_Dbws.To_QName('http://www.webserviceX.NET',
17                                       'GetWeather');
18     lService      := Utl_Dbws.Create_Service(lServiceQName);
19     -- Set port
20     lPortQName    := Utl_Dbws.To_QName('http://www.webserviceX.NET',
21                                       'GlobalWeatherSoap');
22     -- Set operation
23     lOperationQName := Utl_Dbws.To_QName('http://www.webserviceX.NET',
24                                         'GetWeather');
25     -- Create call
26     lCall         := Utl_Dbws.Create_Call(lService      ,
27                                           lPortQName    ,
28                                           lOperationQName);
29     Utl_Dbws.Set_Property(lCall, 'SOAPACTION_USE', 'TRUE');
30     Utl_Dbws.Set_Property(lCall, 'SOAPACTION_URI',
31                             'http://www.webserviceX.NET/GetWeather');
32     -- Set endpoint address
33     Utl_Dbws.Set_Target_Endpoint_Address(lCall,
34                                           'http://www.webserviceX.NET/globalweather.asmx');
35     -- Set xml request
36     lXmlReq := XMLType('<?xml version="1.0" encoding="utf-8"?>'
37                       || '<GetWeather xmlns="http://www.webserviceX.NET">'
38                       || '  <CityName> ' || pCity || '</CityName>'
39                       || '  <CountryName> ' || pCountry || '</CountryName>'
40                       || '</GetWeather>');
41     -- Get xml response
42     lXmlResp := Utl_Dbws.Invoke(lCall, lXmlReq);
43     Utl_Dbws.Release_Call(lCall);
44     Utl_Dbws.Release_Service(lService);
45     lStrResp :=
46     lXmlResp.Extract('/GetWeatherResponse/GetWeatherResult/child::node()',
47                     'xmlns="http://www.webserviceX.NET"').GetStringVal();
48     lStrResp := Dbms_Xmlgen.Convert(lStrResp, Dbms_Xmlgen.ENTITY_DECODE);
49     lXmlResp := XMLType.CreateXml(lStrResp);
50     lStrResp := XmlResp.Extract('/CurrentWeather/Temperature/child::node()',
51                                 'xmlns=""').GetStringVal();
52     Return lStrResp;
53 End;

```

Abbildung 5: Aufruf eines Web Service mit UTL_DBWS

³ Details on installing Oracle Callout Utility: http://www.oracle-base.com/articles/10g/utl_dbws10g.php

Bis zu diesem Zeitpunkt haben wir die Möglichkeiten diskutiert, wie eine Oracle Forms-Anwendung sowohl aus Forms als auch aus der darunterliegenden Datenbank externe Logik in Form von Web-Services konsumieren kann.

Aber wie können wir von außen Forms Funktionalität aufrufen?

4. Bereitstellung von Web Services

Funktionalität für die Außenwelt verfügbar machen

Wir können Funktionalität, die innerhalb der Forms Module definiert ist, nicht direkt als Web Services bereitstellen, aber wir können Oracle Datenbank-Objekte dazu anbieten. Was bedeutet würde, dass die Funktionalität, die wir als Web Service bereitstellen wollen, sich noch in dem Forms-Modul befindet. Damit sollten wir Logik erst einmal in die Datenbank migrieren, um sie anschließend von dort zu verwenden. Lasst uns diese Idee der Migration unserer Geschäftslogik näher betrachten.

4.1. Warum Forms Business-Logik in die Datenbank migrieren?

Die Forms Architektur macht keine klare Trennung zwischen seiner Datenbearbeitung- und den graphischen Komponenten. In ADF, zum Beispiel, kann der Data Modeler (z.B. ADF Business Components (BC)) nicht auf die View Layer Objekte zugreifen oder verweisen. In Forms dagegen, hat jede Programmunit oder Trigger vollen Zugriff auf den graphischen Komponenten. Die in Forms gebotene größere Flexibilität in der Anwendungsentwicklung, wird in der ADF Welt durch die nicht-so-flexible Struktur aber mit einem höheren Grad der Wiederverwendung ausgeglichen: die ADF BC Funktionalität kann als Webdienste freigestellt und aus externen Anwendungen ausgerufen werde. Die einzige Möglichkeit, die wir in Forms-Anwendungen haben um diesen Grad der Wiederverwendbarkeit zu erhöhen ist die Geschäftslogik entweder neu zu schreiben oder sie vom Forms Programm zu trennen. Schließlich sollten die Geschäftsprozesse nicht von der Benutzeroberfläche abhängig sein.

Bei der Migration der Geschäftslogik aus der Forms-Anwendung in die Datenbank ist es ratsam den migrierten Code zu konsolidieren, und ihn beispielsweise mit alldem funktionell-verwandten Code-Segmente in einem DB Pakete⁴ bereitzustellen.

Die Migration des PL/SQL Codes in Datenbank hat eine Reihe von Vorteilen: auf der einen Seite, kann der so migrierte Datenbank-Code von anderen Anwendungen verwendet werden – entweder direkt oder durch Web-Services. Auf der anderen Seite wird die Forms-Anwendung dadurch ganz erheblich vereinfacht und daher leichter migrierbar in anderen Technologien wie ADF⁵ oder APEX.

⁴ PITSS.CON Application Engineering AE, Konzeption und Umsetzung, A. Gaede, 2009, http://pitss.com/fileadmin/pitss/images/de/White_Papers/2009-07_PITSS_White_Paper_AE_-_DE.pdf

⁵ Von Oracle Forms zu Oracle ADF und JEE, M. Serban, B. Us, 2010 http://www.pitss.de/fileadmin/pitss/images/de/White_Papers/2010-11_PITSS_White_Paper_ADF_DE.pdf

4.2. Datenbank Funktionalität bereitstellen

Die Oracle-Datenbank als Web-Service Provider

In einer Service Orientierten Architektur kann eine Oracle-Datenbank nicht nur die Rolle des Web-Service-Clients übernehmen, wie wir bereits gesehen haben, sondern auch die Rolle als Web-Service Provider. Mit der neuesten Datenbank Version hat sich die Interoperabilität mit anderen Anwendungen drastisch erhöht, denn jetzt haben wir Zugriff auf Oracle Daten und Funktionalität via Web Services.

Welche Objekte können wir bereitstellen? Aus der Liste der Oracle-Komponenten, die als Web Services bereitgestellt werden können, wie vordefinierten SQL-Befehle, XQuery oder DML-Anweisungen, PL/SQL und Java gespeicherte Programmblöcke oder Advanced Queues, richten wir unseren Blick auf PL/SQL Programmblöcke. Die PL/SQL Programmblöcke wie Funktionen, Prozeduren oder Packages sind diejenigen, die den Kern unserer Geschäftslogik ausmachen, und damit auch die Business-Logik, die wir aus der Forms-Anwendungen in die Datenbank migriert können.

Abbildung 6 zeigt drei Möglichkeiten auf um die DB-seitige als Webservice zu konsumieren:

- **PL/SQL Web Services** - sind die erste Wahl und vielleicht die am meisten verwendete Lösung, um DB Prozeduren und Funktionen als Web Services zu verwenden. Sie müssen auf einem Webserver wie dem WebLogic deployed werden.
- **BPEL-Prozesse** - werden in der Regel verwendet, um komplexe Prozesse darzustellen und zu orchestrieren. Sie können auch verwendet werden, um Prozeduren und Funktionen der Datenbank einzubinden. Auch sie müssen auf einem Webserver wie dem WebLogic deployed werden, da sie eine BPEL Laufzeitumgebung benötigen.
- **Native XML DB Web Services** – werden erst mit der Oracle 11g-Datenbank angeboten; es stellt die neueste Lösung dar, um Datenbank-Logik als Web Services zur Verfügung zu stellen. Vorteil hier: Diese Web-Services benötigen keine Code oder Web-Server für ihre Bereitstellung.

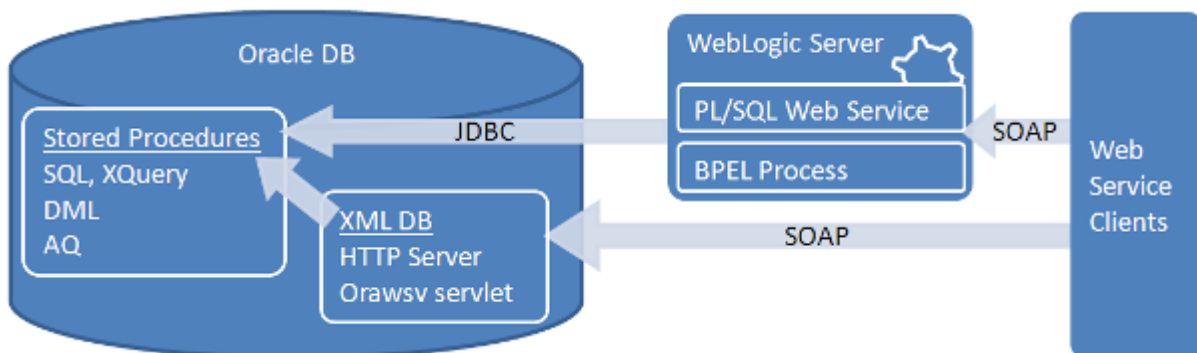


Abbildung 3: Oracle Datenbank as Web Service Anbieter

4.2.1. PL/SQL Web Services

Der PL/SQL Web Service besteht in der Regel aus einer oder mehreren Java-Klassen die über JDBC die Oracle Datenbank-Verbindung herstellen und so die Programmblöcke anrufen.

Auch hier stellt sich die Frage: welche Funktionalität kann als Webservice angeboten werden? Dies sind allein stehende Prozeduren und Funktionen sowie Prozeduren und Funktionen, die in Datenbank-Package definiert sind, einbinden. Hier gibt es jedoch einige Beschränkungen, die durch den Oracle-JDBC-Treiber gegeben sind: nicht alle SQL- und PL/SQL-Datentypen, die die Oracle-Datenbank kennt, werden unterstützt. Wir können, zum Beispiel, keine Funktionen und Prozeduren deren Argumenten SQL-Datentypen sind, wie `Interval day to second`, `Interval year to month`, `Timestamp with local time zone`, `Timestamp with time zone` oder PL/SQL Datentypen wie `PL/SQL boolean`, `PL/SQL record` oder `PL/SQL index by table`.

Jedoch lassen sich diese Einschränkungen durch alternative Lösungen umgehen: wie eine PL/SQL-Wrapper für solche Programmblöcke. Diese Wrapper werden so definieren, dass sie nur SQL-Datentypen benutzen, die von den JDBC-Treiber unterstützt werden bevor sie als Web-Services bereitgestellt werden. Abbildung 7 bietet ein Wrapper-Beispiel, das einen gleichwertigen Datentyp für `Interval year to month` verwendet.

```

1 Function GetDate ( pInterval Interval Year To Month) Return Date
2
3 Function GetDateWrap ( pInterval Varchar2) Return Date Is
4 lInterval Interval Year To Month;
5 Begin
6   lInterval := Sys.Sqljut1.Char2iym(pInterval);
7   Return GetDate (lInterval);
8 End;
```

Abbildung 4: PL/SQL Wrapper für die GetDate Function

Die untenstehende Tabelle veranschaulicht die eingeschränkten Datentypen sowie die dazu passende Äquivalente. Hier können wir sehen, dass für den `PL/SQL record` oder `PL/SQL index by table` müssen wir zusätzlich entsprechende Objekte oder Collection Datentypen erstellen.

Tabelle 1: Datentypzuordnung

Datentypen nicht von JDBC unterstützt	Äquivalenten Strukturen
<code>Interval day to second</code>	Vachar2
<code>Interval year to month</code>	Vachar2
<code>Timestamp with local time zone</code>	Vachar2
<code>Timestamp with time zone</code>	Vachar2
<code>PL/SQL boolean</code>	Integer
<code>PL/SQL record</code>	Object Type (mit entsprechende Struktur)
<code>PL/SQL index by table</code>	Collection Type (mit entsprechende Struktur)

Die manuelle Erstellung eines PL/SQL Web Service ist ein komplizierter Prozess, der eine breite Palette von Kenntnissen erfordert. Hier bietet es sich an auf leistungsfähige Wizard-Programme zurückzugreifen, die unsere Arbeit erleichtern können, wie die von JDeveloper 11g oder PITSS.CON. Sicherlich kann ein Wizard nicht alle Fälle abdecken, aber mit seiner Hilfe und einer nachgelagerten

Anpassung ist man schneller und flexibler als die fehlerbehaftete, manuelle Entwicklung. Typische Einschränkungen sind:

- **Überladene Programmblöcke** - Diese Einschränkung ist bereits durch das WSDL-Dokument gegeben, das verhindert möchte, dass verschiedenen Operationen unter dem gleichen Namen veröffentlicht werden. Die Lösung ist die Erstellung eines weiteren Web-Service als weitere Schnittstelle für den überladenen Programmblock.
- **Standalone Programmblöcke** - Der Assistent des JDeveloper erlaubt derzeit nicht die Verwendung einfacher Funktionen oder Prozeduren. PITSS.CON 8 bietet hier einen leistungsfähigen Assistenten, der solche Programmblöcke als Webservice generiert.
- **Nicht unterstützte Datentypen** wie in Tabelle 1 und `binary_float`, `binary_double`, `nclob` - Die mögliche Lösung wären Wrapper.

Der letzte Schritt zum lauffähigen PL/SQL Webservice ist das Deployment auf einem Webserver. Hier können wir zwischen Oracle WebLogic, JBoss, Tomcat oder WebSphere-Server wählen. Der Deployment-Prozess kann in mehrere Arten durchgeführt werden: vom JDeveloper, mit einem Ant-Task oder, wenn wir uns für WebLogic Server entscheiden, von der WebLogic-Konsole.

4.2.2. BPEL-Prozesse

BPEL ist eine auf XML-basierte Sprache für das Erzeugen von komplexen Prozessen, die auch mehrere Webservices einbinden, man spricht hier auch vom Orchestrieren des Prozesse. Eingesetzt auf einem Webserver wie Oracle WebLogic oder jedem anderen Server, der eine BPEL Laufzeitumgebung besitzt, kann der generierte BPEL-Prozesse selbst auch als Web Services angeboten werden.

Um einen BPEL Prozess zu konstruieren, können wir den im JDeveloper eingebundenen BPEL-Editor verwenden. Das Arbeiten mit BPEL-Editor ist ein einfaches Drag-and-Drop und benötigt keine Java-oder PL/SQL Kenntnisse.

Um uns nicht im Detail zu verlieren, konzentrieren wir uns hier auf BPEL Fähigkeit die uns erlaubt die Oracle-Datenbank-Funktionalität als Web-Services bereitzustellen. Die Schlüsselkomponente, die dies möglich macht, ist der BPEL-DB-Adapter. Dieser Adapter ist von Oracle vordefiniert um auf Standalone-Prozeduren oder -Funktionen zu zugreifen, um DML oder Selects- auszuführen, oder nach neuen bzw. geänderten Datensätze in einer Tabelle zu suchen.

Zur besseren Einschätzung sind hier einige Fähigkeiten oder Einschränkungen des DB-Adapters aufgeführt:

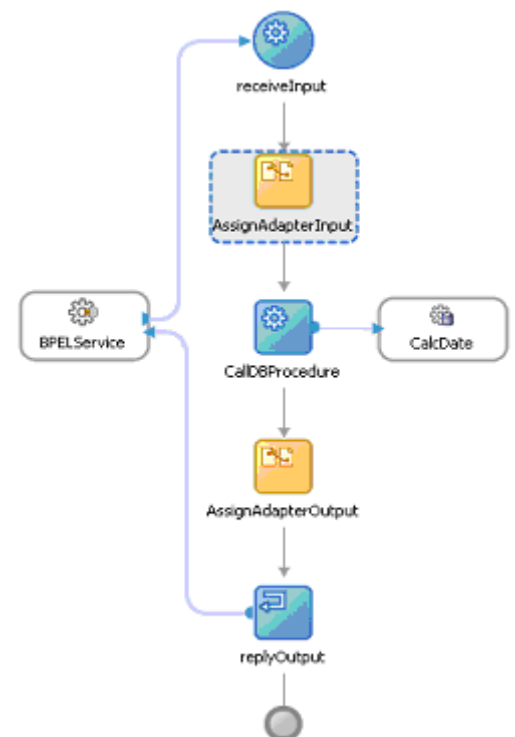


Abbildung 8: Beispiel einer synchronen BPEL-Prozess in der Lage eine Funktion freizustellen

- **Überladene Programmblöcke:** Er ermöglicht die Bereitstellung von überladenen Funktionen oder Prozeduren.
- **Datentypen:** Unterstützt keine Datentypen wie `rowid`, `urowid`, `interval`, `timestamp with (local) time zone`, `bfile`. Wie bei den PL/SQL Web Services, kann der DB-Adapter für Funktionen oder Prozeduren Wrapper generieren und bereitstellen, indem die nicht unterstützten Datentypen mit `varchar2` ersetzt werden. Für die PL/SQL `record`, PL/SQL `index by table` und PL/SQL `Boolean` generiert und erstellt er Datenbank-Wrapper und zusätzliche Objekte oder Tabellentypen.
- Der DB-Adapter kann nur für eine alleinstehende Funktionen oder Prozeduren konfiguriert werden. Dies ist eine klare Einschränkung, wenn wir auf ganze Datenbank-Pakete zugreifen wollen. In einem solchen Fall müssen wir DB-Adapter für jede Funktion oder Prozedur im Paket erstellen. Wenn wir im Paket mehrere Programmblöcke haben, die nicht unterstützte Datentypen verwenden, dann wird jeder der entsprechenden DB Adapter separate Wrapper oder zusätzliche Objekte / Table-Typen generieren. Dies würde zu Namens-Konflikten führen und wir müssten die erstellten Namen so ändern, um die Konflikte zu beseitigen.

4.2.3. Native DB Web -Services

Das Feature der "Native Datenbank Web Services" ist eine Erweiterung der Oracle XML DB, verfügbar ab Oracle11g. Damit eliminiert Oracle den Aufwand an einen Webdienst und mehr als das, es eliminiert die Notwendigkeit eines Webserver überhaupt Bereitstellung. Die Oracle-Datenbank übernimmt hier die Web-Server-Rolle, mit Hilfe ihrem integrierten HTTP-Server, Teil der Oracle XML DB Funktionalität.

Durch Aktivierung der nativen Datenbank Web Services sind die Datenbank-Prozeduren und Funktionen, die entweder Standalone oder in einem Paket definiert wurden, automatisch als Webdienste bereitgestellt. Darüber hinaus enthalten die native Datenbank Web-Services einen Webdienst, der von der Web-Client-Anwendungen verwendet werden kann, um SQL-Anweisungen und XQuery-Ausdrücke dynamisch auszuführen.

Die Aktivierung der nativen DB Web Services Feature

Aus Sicherheitsgründen ist das native DB Web Service Feature in der Oracle-Datenbank nicht automatisch aktiviert. Um die Oracle XML DB zu konfigurieren und dieses Feature zu aktivieren, müssen wir die Oracle XML DB HTTP-Server aktivieren und die 'orawsv' Servlet im `xdbconfig.xml` Datei editieren.

Die Aktivierung des XML DB HTTP-Servers wird abgeschlossen durch das Setzen der Port-Nummer auf einen Wert größer als 0. Eine Überprüfung ist einfach, indem wir die Port-Nummer des XML-HTTP-Server DB mit Hilfe der `DBMS_XDB.GetHttpPort` oder `.SetHttpPort` Funktionalität einstellen, wie in der folgenden Befehlszeile:

```
SQL> EXEC dbms_xdb.sethttpport(8080);
```

Das gleiche DBMS_XDB Paket kann benutzt werden, um die 'orawsv' Servlet-Konfiguration in der xdbconfig.xml Oracle-Konfigurationsdatei hinzuzufügen, siehe Skriptbeispiel in Abbildung 9.

```
1 Declare
2 lServletName      Varchar2(30) := 'orawsv';
3 Begin
4 DBMS_XDB.DeleteServletMapping(lServletName);
5 DBMS_XDB.DeleteServlet(lServletName);
6 DBMS_XDB.AddServlet(NAME      => lServletName,
7                     LANGUAGE => 'C',
8                     DISPNAME => 'Oracle Query Web Service',
9                     DESCRIPT => 'Servlet for issuing queries as a Web Service',
10                    SCHEMA    => 'XDB');
11 DBMS_XDB.AddServletSecRole(SERVNAME => lServletName,
12                           ROLENAME  => 'XDB_WEBSERVICES',
13                           ROLELINK  => 'XDB_WEBSERVICES');
14 DBMS_XDB.AddServletMapping(PATTERN => '/orawsv/*' ,
15                            NAME     => lServletName);
16 End;
17 /
```

Abbildung 9: SQL Skript zum Hinzufügen des 'orawsv' Servlet-Konfiguration zur XML DB-Konfigurationsdatei (laufen als SYS Datenbankbenutzer)

Aktivieren der nativen DB Web Services für einen DB-Benutzer

Obwohl das native Datenbank Web Services Feature jetzt aktiviert ist, brauchen die Datenbank-Benutzer zusätzliche Rechte, um in die Lage zu kommen eigene Programmblöcke als native Web-Services bereitzustellen. Es gibt 3 Rollen, die den Zugriff auf die native Datenbank Web Services steuern:

- **XDB_WEBSERVICES** - erforderliche Rolle, die die Verwendung von nativen Web Services über HTTPS ermöglicht
- **XDB_WEBSERVICES_OVER_HTTP** - erlaubt die Verwendung von nativen Web Services über HTTP
- **XDB_WEBSERVICES_WITH_PUBLIC** - ermöglicht den Zugriff auf PUBLIC Datenbank-Objekte.

Finden und ausführen von Native DB Web Services

Nach der Aktivierung des nativen Datenbank Web Services Feature und der Erteilung der entsprechenden Rollen für die Datenbankbenutzer, folgt der nächste Schritt, das Identifizieren der Web-Services Adresse, d.h. die URLs des nativen Web Services WSDL-Dokumentes. Das WSDL-Dokument gibt uns nicht nur die Adresse des Web Service, sondern auch die enthaltenden Operationen samt Parameter. Es bleibt anzumerken, dass die Oracle XML DB SOAP1.1 unterstützt und der Web-Service-Client diese Version benötigt um im HTTP POST Prozess die SOAP-Anfrage an die native Datenbank-Webdienste weiterzureichen.

Das native Datenbank Web-Services-Feature bietet nur einen Webdienst, der verwendet werden kann, um SQL-Anweisungen oder XQuery-Ausdrücke in der Datenbank durchzuführen. Seine WSDL-Dokument URL verweist auf eine Adresse wie <http://host:port/orawsv?wsdl>, wo der Host der Datenbank-Host und der Port die Port-Nummer ist, die von der XML DB HTTP-Server verwendet wird.

Bei den PL/SQL-Programmblöcke haben wir eine andere Situation. Jede Prozedur oder Funktion, Standalone oder im Paket, hat seinen eigenen dynamischen Web Service. Die WSDL-Dokumente diese Web-Services befinden sich für Standalone-Funktionen und Prozeduren:

http://host:port/orawsv/DBSCHEMA/FUNCTION_or_PROCEDURE?wsdl,

und bei Funktionen und Prozeduren aus Paketen:

http://host:port/orawsv/DBSCHEMA/PACKAGE/FUNCTION_or_PROCEDURE?wsdl.

Zusätzlich gibt es native Web-Services definiert für Datenbank-Pakete. Solch ein Webdienst stellt alle Prozeduren und Funktionen zur Verfügung, die im zugehörigen Paket definiert sind, sowie die URL des WSDL-Dokuments <http://host:port/orawsv/DBSCHEMA/PACKAGE?wsdl>.

Hier müssen wir darauf achten, dass die URLs Schema-, Paket-, Funktion- oder Prozedurname in Großbuchstaben geschrieben werden, sonst bringt uns der Browser einen Fehler zurück.

Nachdem wir bisher die Vorteile der nativen DB Web Services betrachtet haben, bleibt die Frage nach ihren Beschränkungen? Einige typische Probleme, auf die wir stoßen können, sind:

- **Überladene** Funktionen oder Prozeduren können nur über die native Web-Services erreicht werden
- **Unterstützte Datentypen:** Die Prozeduren und Funktionen mit Parametern wie `PL/SQL record`, `PL/SQL indexed tables` oder `database collection`, lassen sich nicht als native Web-Services bereitstellen.

5. Fazit

Wir haben hier die Möglichkeiten einer besseren Integration von Oracle Forms-Anwendungen in die moderne Web-Welt betrachtet. Da wir und unsere Anwendungen ständig mit neuen, modernen Technologie für die Entwicklung von Datenbankanwendungen konfrontiert werden, ist alleine schon die Auswahl der Besten schwieriger als je zuvor. Werfen Sie einen Blick auf Oracle Konferenzen: die Agenda's sind mit neuen Themen, wie ADF, APEX, BI Publisher und so weiter ausgefüllt und es ist richtig so, da eine neue Ära der Anwendungsentwicklung längst begonnen hat: die Ära der Web-Entwicklung. Doch während wir die Strategien bewerten, um den besten Weg zu wählen, sind wir mit einer sehr bodenständigen, wichtigen und dringenden Frage konfrontiert: Was passiert mit dem gegenwärtigen System, sodass es weiterhin die Geschäftsprozesse unterstützen mit Hilfe der aktuell verfügbaren Ressourcen.

Viele Unternehmen, die Legacy Forms-Anwendungen besitzen, verharren noch in einem "Wait-and-See"-Ansatz und machen erst einmal den Upgrade ihrer Forms-Anwendungen auf die zurzeit neue 11g Version. Auf diese Weise erhalten sie die Vorteile, die die 11g-Version mit sich bringt, bleiben unter Support, während man Zeit gewinnt, um die neuen Technologien zu bewerten, zu verstehen, die moderne Entwicklungssprachen und -Architekturen zu erlernen, und die neuen Technologien reifer werden.

Aber, ob wir nun beschließen in die neuen Technologien zu migrieren, oder erst einmal bei Forms zu bleiben, bietet sich eine gute Strategie an, so viel wie möglich von der Business-Logik einer Forms-Anwendung in die Datenbank zu verschieben. Der Applikations-Code wird entrümpelt, der Code überarbeitet und als Webservice anderen Anwendern und Kunden zugänglich gemacht. Der Hauptvorteil jedoch liegt darin, dass Sie, wann immer Sie wollen, viel einfacher in die neuen Technologien migrieren.

Die Implementierung von Web Services und das Orchestrieren in einer modernen Architektur stellt alleine schon eine exzellente Verbesserung dar und kann mit minimalen Investitionen erreicht werden. Ebenfalls lässt sich der Kerngedanke der Service-orientierten (SOA) Prinzipien erreichen – Wiederverwendung, Interoperabilität und setzen von Standards – während unsere Datenbank-Anwendungen flexibler werden und schneller auf die sich ändernden Geschäftsanforderungen reagieren.

Über PITSS

Die PITSS GmbH ist der führende Anbieter von integrierten Komplettlösungen für das effektive Management von Oracle Forms Applikationen. Die innovative Software PITSS.CON unterstützt Unternehmen ganzheitlich bei der erfolgreichen Analyse, Migration, Weiterentwicklung und Wartung von Oracle Forms Applikationen. Damit bietet PITSS Oracle Forms Kunden einen fließenden Übergang in die SOA Welt. PITSS.CON überzeugt durch seine hochgradige Automatisierung und Leistungsfähigkeit. Migrations- und Entwicklungsprojekte werden so äußerst schnell, wirtschaftlich und zuverlässig umgesetzt. PITSS.CON erzielt für Unternehmen über alle Entwicklungsprozesse eine Kostenersparnis von durchschnittlich 30% in Migrationsprojekten sogar von 90%. PITSS ist Oracle Certified Advantage Partner und hat seine Kunden in Europa, USA und Asien.



Implementierung von Web Services in Oracle-Datenbankanwendungen

Februar 2011

Autor: Florin Serban
Co-Autoren: Magdalena Serban, Andreas Gaede

PITSS in Europe

Deutschland
+49-711-728.752.00
info@pitss.com
www.pitss.de

PITSS in Amerika

USA
248.740.0935
info@pitssamerica.com
www.pitssamerica.com

Copyright 2011, PITSS GmbH