Dieser Artikel zeigt übersichtlich, wie wir Legacy-Oracle-Forms-Anwendungen in der modernen Web-Entwicklungswelt integrieren können.

Implementierung von Web-Services in Oracle-Datenbank-Anwendungen

Florin Serban, Magdalena Serban und Andreas Gaede, PITSS GmbH

In dem Bestreben nach schneller Entwicklung und Anpassung, nach Bugfixing und Patching, ist die typische Oracle-Datenbank-Anwendung größer und größer geworden. Sie besteht nun aus einer Mischung von alten und neuen Komponenten, geschrieben in verschiedenen Programmiersprachen und ausgelegt mit unterschiedlichen Architekturen. Eine unserer größten Herausforderung ist es, das Beste aus all diesen Komponenten herauszuziehen, neu zu strukturieren und in modernen, modularen Anwendungen interagieren zu lassen, sodass die heutigen und zukünftigen Geschäftsanforderungen flexibel unterstützt werden.

Der Artikel zeigt Möglichkeiten, in die Jahre gekommene Forms-Anwendungen zu modernisieren. Da serviceorientierte Architekturen zunehmend an Bedeutung gewinnen, wird aufgezeigt, mit welchen Mittel sich externe Anwendungen durch die Implementierung von Web-Services integrieren lassen.

Traditionelle Forms-Anwendungen

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

- *Als Web-Services-Verbraucher*
  Forms-Anwendungen besitzen 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 mithilfe 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, ist der entsprechende Source-Code in die Oracle-Datenbank zu migrieren, um ihn von dort als Web-Service verwenden zu können. 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-Codes erhöht und damit die zukünftigen Migrations- und Modernisierungsschritte unserer Forms-Anwendungen ganz erheblich vereinfacht.

Abbildung 1: Oracle-Datenbank-Anwendung

Web-Services aufrufen

Wie bereits angesprochen, gibt es zwei Möglichkeiten, um auf externe Web-Services von Forms-Anwendungen aus zuzugreifen: direkt von Forms mithilfe eines Web-Service-Proxy-Servers oder von der Oracle-Datenbank aus unter Verwendung seiner Web-Service-Utilities. Wie kann Forms nun auf externe Web-Services zugreifen? Forms ist von Natur aus nicht in der Lage, direkt mit Web-Services zu kommunizieren, es können jedoch Java-Klassen innerhalb eines Web-Service-Proxy kommunizieren. Der Proxy dient hierbei als Vermitt-

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

ler, denn er ist in der Lage, eine direkte Verbindung zum eigentlichen Web-Service zu etablieren (siehe Abbildung 2).

Um diesen Mechanismus zum Aufruf eines Web-Service in der Forms-Anwendung zu verankern, sind einige Schritte erforderlich:

- *Generieren des Web-Service-Proxy*
  Dieser kann unter Verwendung des Oracle JDeveloper leicht erstellt werden. Man muss lediglich die URL eingeben, in der die Definition des Web-Service steht, die sogenannte „Web-Service Definition Language“ (WSDL). Dies ist ein XML-Dokument, das alle Informationen über die Operationen, die Adresse des Web-Service sowie das Message-Protokoll enthält. Wichtig ist hier die Wahl der richtigen Java-Version, die zur Kompilierung der Java-Dateien verwendet werden sollte und die 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 ist die FORMS_BUILDER_CLASSPATH-Variable im Oracle-Home der aktuellen Forms-Version einzustellen. Der Variablenwert sollte die Adresse beinhalten, unter der die erzeugte Proxy-JAR-Datei zu finden ist. Auf diese Art und Weise werden die Java-Klassen des Web-Service-Proxy für den Forms-Java-Importer sichtbar.
- *Importieren der Java-Klassen in der Forms-Anwendung*
  Hier kommt der Forms-Java-Importer zum Einsatz, um die Proxy-Java-Klassen, die mit dem Webdienst kommunizieren sollen, in unsere Forms-Anwendung zu importieren. Diese Klassen werden in der Regel mit „WebServiceNamePortClient“ benannt. Der Java-Importer kreiert in der Forms-Anwendung für die importierten Java-Klassen eine Package-Spezifikation und deren 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 der Proxy-JAR-Datei hinzufügen. Diese Einstellung ist notwendig, um die Proxy-Java-Klassen während der Laufzeit im Zugriff zu haben.

```
Declare
lJavaObject         Ora_Java.Jobject;
lWeather            Varchar2(4000);
lXml                XmlType;
Begin
  lJavaObject := GlobalWeatherSoapPortClient.New();
  lWeather    := GlobalWeatherSoapPortClient.GetWeather
                    (lJavaObject,:block2.city, :block2.country);
  lWeather    := Dbms_Xmlgen.Convert(lWeather, 1);
  lXml        := XMLType.CreateXml(lWeather);
  :block2.temperature :=
        lXml.Extract(‚/CurrentWeather/Temperature/child::node()‘,
                  ‚xmlns=““‘).GetStringVal();
Exception
  When ORA_JAVA.JAVA_ERROR Then
    Message(‚Unable to call out to Java, , ||ORA_JAVA.LAST_ERROR);
End;
```

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

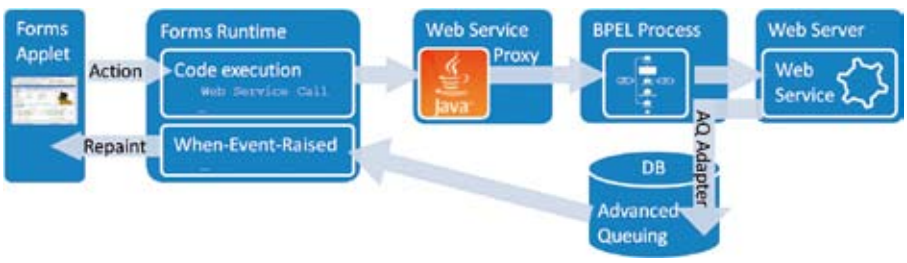

Abbildung 4: Die Kommunikation zwischen Forms und Web-Service über Forms-11g-Event

Sobald dieser Mechanismus in der Forms-Anwendung implementiert ist, lassen sich die Funktionalitäten des neu generierten Package verwenden und der Web-Service aufrufen (siehe Abbildung 3).

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-

```
Function GetWSTemperature( pCountry    Varchar2,
                           pCity       Varchar2) Return Varchar2 Is
lCall                  Utl_Dbws.Call;
lOperationQName        Utl_Dbws.QName;
lPortQName             Utl_Dbws.QName;
lStrResp               Varchar2(32767);
lService               Utl_Dbws.Service;
lServiceQName          Utl_Dbws.QName;
lXmlReq                XMLType;
lXmlResp               XMLType;
Begin
    -- Set proxy if necessary
    -- Utl_Dbws.Set_Http_Proxy(192.168.161.123:3128)

    -- Create a service
    lServiceQName   := Utl_Dbws.To_QName(‚http://www.webserviceX.NET‘,
    ‚GetWeather‘);
    lService        := Utl_Dbws.Create_Service(lServiceQName);
    -- Set port
    lPortQName      := Utl_Dbws.To_QName(‚http://www.webserviceX.NET‘,
    ‚GlobalWeatherSoap‘);
    -- Set operation
    lOperationQName := Utl_Dbws.TO_QName(‚http://www.webserviceX.NET‘,
    ‚GetWeather‘);
    -- Create call
    lCall           := Utl_Dbws.Create_Call(lService, lPortQName, lOpera
    tionQName);
    Utl_Dbws.Set_Property(lCall, ‚SOAPACTION_USE‘, ‚TRUE‘);
    Utl_Dbws.Set_Property(lCall, ‚SOAPACTION_URI‘, ‚http://www.webserviceX.
    NET/GetWeather‘);

    -- Set endpoint address
    Utl_Dbws.Set_Target_Endpoint_Address(lCall, ‚http://www.webserviceX.
    NET/globalweather.asmx‘);
    -- Set xml request
    lXmlReq := XmlType(‚<?xml version=“1.0“ encoding=“utf-8“?>‘
                    ||  ‚<GetWeather xmlns=“http://www.webserviceX.NET“>‘
                    ||    ‚<CityName>‘    || pCity    || ‚</CityName>‘
                    ||    ‚<CountryName>‘ || pCountry || ‚</CountryName>‘
                    ||  ‚</GetWeather>‘);
    -- Get xml response
    lXmlResp := Utl_Dbws.Invoke(lCall, lXmlReq);
    Utl_Dbws.Release_Call(lCall);
    Utl_Dbws.Release_Service(lService);
    lStrResp := lXmlResp.Extract(‚/GetWeatherResponse/GetWeatherResult/
    child::node()‘,
                                 ‚xmlns=“http://www.webserviceX.NET“‘
          ).GetStringVal();
    lStrResp := Dbms_Xmlgen.Convert(lStrResp, Dbms_Xmlgen.ENTITY_DECODE);
    lXmlResp := XMLType.CreateXml(lStrResp);
    lStrResp := lXmlResp.Extract(‚/CurrentWeather/Temperature/
    child::node()‘, ‚xmlns=““‘).GetStringVal();
    Return lStrResp;
End;
```

Abbildung 5: Aufruf eines Web-Service mit UTL_DBWS

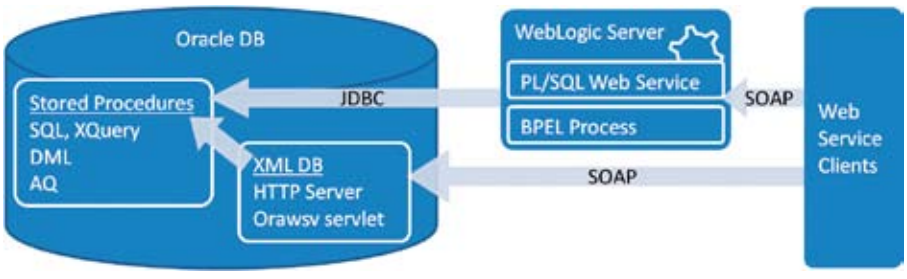

Abbildung 6: Oracle-Datenbank als Web-Service-Anbieter

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? Falls nicht, können wir einen asynchronen Web-Service benutzen.
- Wie lange dauert die Ausführung des Web-Service? Können wir es uns leisten, die Anwendung für diesen Zeitraum zu blockieren?
- Betrachtet man die Ressourcen, die während des Web-Service blockiert werden: Werden sie auch für andere Prozesse verwendet?

Die letzten beiden Fragen sind sehr wichtig: Denn ein verschachteltes Blocken 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 wäre das Einbinden eines Einweg-BPEL-Prozesses, der die Datenbank-Funktionalität Advanced Queuing (AQ) und das neue Forms-11*g*-Event-Objekt erlaubt.

Forms-11*g*-Event-Objekt

In der 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 ein Einweg-Prozess ist („shut and forget“), wartet der Web-Service-Proxy nicht auf dessen Antwort. 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. Nach dem Empfang übergibt der BPEL-Prozess die Antwort, in diesem Fall als Mitteilung einer Queue, auf die ein Event-Objekt der Forms-Anwendung wartet

(siehe Abbildung 4). Wir haben hier mehrere Möglichkeiten, damit der BPEL-Prozess die Antwort sendet:

- Direkte Übertragung der Antwort durch einen AQ-Adapter
- Indirekt über einen DB-Adapter: Bevor wir die Antwort weiter an die Queue senden, können wir zusätzliche Aktionen in der Datenbank mittels Datenbank-Trigger, -Funktion oder -Prozedur auszuführen.

Sobald die Antwort in der Queue platziert wird, meldet das Advanced Queuing dies der Forms-Runtime. Der „When Event Raised"-Trigger liest beim nächsten Ping des Forms-Applets die Antwort ein (siehe http://www.oracle.com/technology/sample_code/products/forms/11g/aqinteg/lab1/index.htm#o).

Dazu eine Empfehlung zu asynchronen Prozessen: Da die Ausführung des Webdienstes einige Zeit dauern kann und die Forms-Anwendung durchaus die Kontrolle über den gestarteten Prozess verliert, sollte der BPEL-Prozess regelmäßig Nachrichten über seinen Status an die Forms-Anwendung senden. Bei langläufigen Webdiensten bietet sich an, dass der Web-Service den 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.

Aufrufen von Web-Services aus der Oracle-Datenbank

Jetzt stellt sich die Frage, wie man die Datenbank-Funktionalität beziehungsweise die Geschäftslogik, die aus der Forms-Anwendung zur Datenbank migriert wurde, befähigt, sich vorhandener Web-Services 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 können der Java-Web-Service und dessen abhängige Klassen zu Inkompatibilitäten mit vorhandenen Java-Klassen der Datenbank führen, was dann die Datenbankstabilität beeinträchtigen könnte (siehe http://www.oracle.com/technology/sample_code/tech/java/jsp/callout_users_guide.htm).
- *UTL_HTTP*
  Die Funktion ist verfügbar ab Oracle 7.3.4 und bietet die Möglichkeit, HTTP-Aufrufe direkt aus der Datenbank durchzuführen. Sie 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*
  Die Funktion ist ab Oracle 10*g* verfügbar und kann direkt SOAP-Calls verwenden. Im Vergleich zu UTL_HTTP ist sie besser auf Web-Services ausgelegt. Die Funktionen und Prozeduren, die in diesem Paket definiert sind, stellen Wrapper für die JAX-RPC Dynamic 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 10*g*-Datenbank vorinstalliert ist, ist die dazugehörige Callout-Utility nicht sofort verfügbar und muss nachträglich installiert werden (siehe http://www.oracle-base.com/articles/10g/utl_dbws10g.php). Abbildung 5 zeigt ein Beispiel zur Anwendung von UTL_DBWS, um einen externen Web-Service aufzurufen, der wiederum die Temperatur für einen bestimmten Ort abfragt.

Die beiden zuletzt 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 in der DBWS-Callout-Utility verfügbar sind, sofern es um SOAP-Aufrufe geht.

Bereitstellung von Web-Services

Wir können Funktionalität, die innerhalb der Forms-Module definiert ist, nicht direkt als Web-Services bereitstellen, aber wir können Oracle-Datenbank-Objekte dazu anbieten. Das bedeutet, 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.

Die Forms-Architektur macht keine klare Trennung zwischen seiner Datenbearbeitung und den grafischen Komponenten. In ADF kann zum Beispiel der Data Modeler (etwa ADF Business Components (BC)) nicht auf die View-Layer-Objekte zugreifen oder verweisen. In Forms dagegen hat jede Programm-Unit oder jeder Trigger vollen Zugriff auf grafische Komponenten. Die in Forms gebotene größere Flexibilität in der Anwendungsentwicklung wird in der ADF-Welt durch einen höheren Grad der Wiederverwendung ausgeglichen: die ADF-BC-Funktionalität kann als Webdienst freigestellt und aus externen Anwendungen aufgerufen werden. 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 all den funktionell verwandten Code-Segmenten in einem DB-Paket (siehe http://pitss.com/fileadmin/pitss/images/de/White_Papers/2009-07_PITSS_White_Paper_AE_-_DE.pdf) bereitzustellen. Die Migration des PL/SQL-Codes in die Datenbank bietet 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

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

Tabelle 1: Datentyp-Zuordnung

```
Function GetDate ( pInterval  Interval Year To Month) Return Date

Function GetDateWrap ( pInterval  Varchar2) Return Date Is
lInterval  Interval Year To Month;
Begin
  lInterval := Sys.Sqljutl.Char2iym(pInterval);
  Return GetDate(lInterval);
End;
```

Abbildung 7: PL/SQL-Wrapper für die GetDate-Function

Forms-Anwendung dadurch ganz erheblich vereinfacht und daher leichter migrierbar in anderen Technologien wie ADF oder APEX (siehe http://www.pitss.de/fileadmin/pitss/images/de/White_Papers/2010-11_PITSS_White_Paper_ADF_DE.pdf).

Datenbank-Funktionalität bereitstellen

In einer serviceorientierten 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 stark erhöht, denn jetzt haben wir Zugriff auf Oracle-Daten und -Funktionalität via Web-Services.

Aus der Liste der Oracle-Komponenten, die als Web-Services bereitgestellt werden können, wie vordefinierten SQL-Befehle, XQuery oder DML-Anweisungen, in PL/SQL und Java gespeicherte Programmblöcke oder Advanced Queues, richten wir unseren Blick auf PL/SQL-Programmblöcke. Diese Funktionen, Prozeduren oder Packages sind diejenigen, die den Kern unserer Geschäftslogik ausmachen und damit auch die Business-Logik, die wir aus den Forms-Anwendungen in die Datenbank migrieren können. Abbildung 6 zeigt drei Möglichkeiten auf, um Datenbank-seitige Web-Services zu konsumieren:

- *PL/SQL-Web-Services*
  Sie sind die erste Wahl und vielleicht die am meisten verwendete Lösung, um Datenbank-Prozeduren und -Funktionen als Web-Services zu verwenden. Sie müssen auf einem Webserver wie dem WebLogic deployed sein.
- *BPEL-Prozesse*
  Sie 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 sind auf einem Webserver wie dem WebLogic zu deployen, da sie eine BPEL-Laufzeitumgebung benötigen.
- *Native XML-DB-Web-Services*
  Sie sind erst mit der 11*g*-Datenbank verfügbar und repräsentieren die neueste Lösung, um Datenbank-Logik als Web-Services zur Verfügung zu stellen. Der Vorteil hier: Diese Web-Services benötigen keine Code oder Web-Server für ihre Bereitstellung.

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 Web-Service angeboten werden?

Dies sind alleinstehende Prozeduren und Funktionen sowie solche, die in Datenbank-Packages definiert sind. 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 verwenden, deren Argumente 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“.

Diese Einschränkungen lassen sich jedoch durch alternative Lösungen umgehen: mit einem PL/SQL-Wrapper für solche Programmblöcke. Diese Wrapper werden so definiert, dass sie nur SQL-Datentypen benutzen, die von den JDBC-Treibern unterstützt werden, bevor man sie als Web-Services bereitstellt. Abbildung 7 zeigt ein Wrapper-Beispiel, das einen gleichwertigen Datentyp für „Interval year to month“ verwendet.

Tabelle 1 veranschaulicht die eingeschränkten Datentypen sowie die dazu passenden Äquivalente. Hier können wir sehen, dass für den „PL/SQL record“ oder „PL/SQL index by table“ zusätzlich entsprechende Objekte

oder Collection-Datentypen zu erstellen sind.

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 die Arbeit erleichtern können, wie 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 verschiedene Operationen unter dem gleichen Namen veröffentlicht werden. Die Lösung ist die Erstellung eines weiteren Web-Service als zusätzliche 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 Web-Service generiert.
- *Nicht unterstützte Datentypen*
  Beispielsweise wie in Tabelle 1 oder „binary_float“, „binary_double“, „nclob“ – die mögliche Lösung wären Wrapper.

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

BPEL ist eine XML-basierte Sprache für das Erzeugen von komplexen Prozessen, die auch mehrere Web-Services einbinden, man spricht hier auch vom „Orchestrieren“ der Prozesse. Eingesetzt auf einem Webserver wie Oracle WebLogic oder jedem anderen Server, der eine BPEL-Laufzeitumgebung besitzt, kann der generierte BPEL-Prozess selbst auch als Web-Service angeboten werden (siehe Abbildung 8). 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 die BPEL-Fähigkeit, die es uns erlaubt, die Oracle-Datenbank-Funktionalität als Web-Services bereitzustellen. Die Schlüsselkomponente, die dies möglich macht, ist der BPEL-DB-Adapter. Dieser ist von Oracle vordefiniert, um auf Standalone-Prozeduren oder -Funktionen zuzugreifen, um DML oder Selects auszuführen oder um nach neuen beziehungsweise geänderten Datensätze in einer Tabelle zu suchen. Zur besseren Einschätzung sind hier einige Fähigkeiten oder Einschränkungen des DB-Adapters aufgeführt:

- *Überladene Programmblöcke*
  Ermöglicht die Bereitstellung von überladenen Funktionen oder Prozeduren
- *Datentypen*
  Unterstützt keine Datentypen wie zum Beispiel „rowid”, „urowid”, „interval“, „timestamp with (local) time zone“ oder „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 „PL/SQL record“, „PL/SQL index by table“ und „PL/SQL Boolean“ generiert und erstellt der Datenbank-Wrapper zusätzliche Objekte oder Tabellentypen.
- *DB-Adapter*
  Der DB-Adapter kann nur für 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 Datenbank-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 Adapter separate Wrapper oder zusätzliche Objekte/Table-Typen generieren. Dies würde zu Namens-Konflikten führen und wir müssten die erstellten Namen ändern, um die Konflikte zu beseitigen.

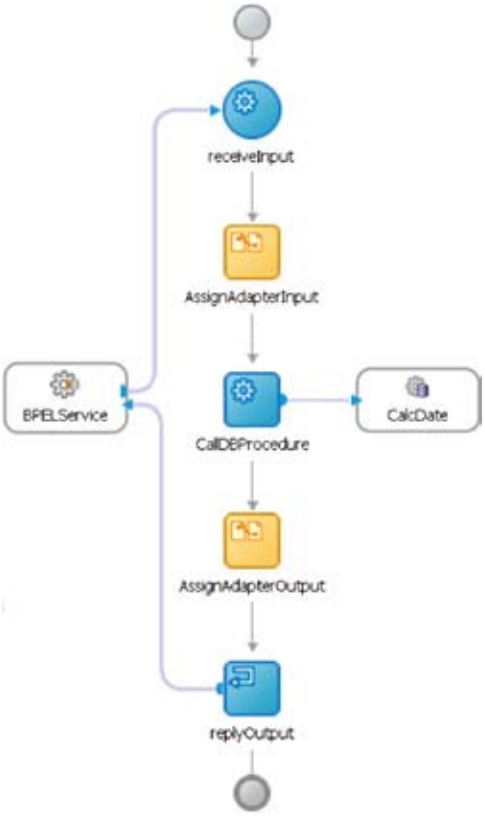

Abbildung 8: Beispiel eines synchronen BPEL-Prozesses, um eine Funktion freizustellen

Das Feature der „Nativen Datenbank-Web-Services“ ist eine Erweiterung der Oracle-XML-DB, verfügbar ab 11*g*. Damit eliminiert Oracle den Aufwand eines Webdienstes und mehr als das, es eliminiert die Notwendigkeit eines Webservers überhaupt. Die Oracle-Datenbank übernimmt hier die Web-Server-Rolle, mithilfe ihres integrierten HTTP-Servers, eines Teils der Oracle-XML-Datenbank-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 nativen Daten-

```
Declare
 lServletName        Varchar2(30) := ‚orawsv‘;
Begin
 DBMS_XDB.DeleteServletMapping(lServletName);
 DBMS_XDB.DeleteServlet(lServletName);
 DBMS_XDB.AddServlet(NAME     => lServletName,
                  LANGUAGE => ‚C‘,
                DISPNAME => ‚Oracle Query Web Service‘,
                DESCRIPT => ‚Servlet for issuing queries as a Web Service‘,
                  SCHEMA   => ‚XDB‘);
 DBMS_XDB.AddServletSecRole(SERVNAME => lServletName,
                            ROLENAME => ‚XDB_WEBSERVICES‘,
                           ROLELINK => ‚XDB_WEBSERVICES‘);
 DBMS_XDB.AddServletMapping(PATTERN => ‚/orawsv/*‘,
                            NAME    => lServletName);
End;
/
```

Abbildung 9: SQL-Skript zum Hinzufügen der „orawsv“-Servlet-Konfiguration zur XML-Konfigurationsdatei (als „SYS“-Datenbankbenutzer)

bank-Web-Services einen Webdienst, der von den Web-Client-Anwendungen verwendet werden kann, um SQL-Anweisungen und XQuery-Ausdrücke dynamisch auszuführen.

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

Die Aktivierung des XML-HTTP-Servers wird durch das Setzen der Port-Nummer auf einen Wert größer als 0 abgeschlossen. Eine Überprüfung ist einfach, indem wir die Port-Nummer des XML-HTTP-Servers mithilfe 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 -Konfigurationsdatei hinzuzufügen (siehe Abbildung 9).

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 drei Rollen, die den Zugriff auf die nativen 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.

Nach der Aktivierung des nativen Datenbank-Web-Services-Feature und der Erteilung der entsprechenden Rollen für die Datenbank-Benutzer folgt der nächste Schritt, das Identifizieren der Web-Services-Adresse, also der URLs des nativen Web-Services-WSDL-Dokuments. Diese enthält nicht nur die Adresse des Web-Service, sondern auch die enthaltenen Operationen samt Parametern. 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 nativen 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, wobei „Host“ der Datenbank-Host und „Port“ die Port-Nummer ist, die vom XML-DB-HTTP-Server verwendet wird.

Bei den PL/SQL-Programmblöcken haben wir eine andere Situation. Jede Prozedur oder Funktion, Standalone oder im Paket, hat ihren eigenen dynamischen Web-Service. Die WSDL-Dokumente dieser Web-Services stehen für Standalone-Funktionen und -Prozeduren unter http://host:port/orawsv/DBSCHEMA/FUNCTION_or_PROCEDURE?wsdl, und bei Funktionen und Prozeduren aus Paketen unter http://host:port/orawsv/DBSCHEMA/PACKAGE/FUNCTION_or_PROCEDURE?wsdl.

Zusätzlich gibt es native Web-Services, die für Datenbank-Pakete definiert sind. 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 für Schema-, Paket-, Funktions- oder Prozedurnamen 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*
  Diese können nur über die nativen 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 bereitstellten.

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 Technologien für die Entwicklung von Datenbank-Anwendungen konfrontiert werden, ist allein schon die Auswahl der besten schwieriger als je zuvor. Auf Oracle-Konferenzen sind die Agenden voll mit neuen Themen wie ADF, Apex, BI-Publisher etc. Sie zeigen, dass eine neue Ära der Anwendungsentwicklung längst begonnen hat: die 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 mithilfe der aktuell verfügbaren Ressourcen unterstützt?

Viele Unternehmen, die Legacy-Forms-Anwendungen besitzen, verharren noch in einem „Wait-and-See"-Ansatz und machen erst einmal das Upgrade ihrer Forms-Anwendungen auf die zurzeit neue Version 11*g*. Auf diese Weise erhalten sie die Vorteile, die diese Version mit sich bringt, bleiben unter Support und gewinnen Zeit, um die neuen Technologien zu bewerten, zu verstehen, die modernen Entwicklungs-Sprachen und -Architekturen zu erlernen und die neuen Technologien reifen zu lassen.

Ob wir nun beschließen, auf die neuen Technologien zu migrieren oder erst einmal bei Forms bleiben, es 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, überarbeitet und als Web-Service anderen Anwendern und Kunden zugänglich gemacht. Der Hauptvorteil jedoch liegt darin, dass man viel einfacher auf die neuen Technologien migrieren kann.

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

Kontakt:
Magdalena Serban
mserban@pitss.de