Oracle und XML im Einsatz

Auch wenn XML ein Grundsystem für eine Vielzahl von Anwendungen bietet, handelt es sich nur um eine anwendbare Technologie – nicht aber um eine Anwendung. Ohne ein vereinbartes Schema oder eine DTD können Anwendungen mit XML nicht zuverlässig Daten austauschen oder in einer bestimmten Form darstellen. In diesem Kapitel untersuchen wir zwei Anwendungsbereiche, die diese Vereinbarungen nutzen, und diskutieren bestimmte Implementierungen.

XML ist aus einer Reihe von Gründen eng mit dem Internet verbunden. Weil der Inhalt eines XML-Dokuments einfach nur Text ist, können Dokumente über bestehende Internetprotokolle, über Betriebssysteme und über Firewalls ausgetauscht werden. Diese Möglichkeit eröffnet zwei große Anwendungsbereiche – die Übertragung von Inhalten an eine Vielzahl von internetfähigen Geräten und den Austausch von E-Business-Daten.

Bevor wir uns diese Anwendungsbereiche im Detail ansehen, müssen wir zwei Oracle Java-Komponenten genauer betrachten, die in Kapitel 3 vorgestellt wurden und die XML-Infrastruktur-Komponenten nutzen: die Oracle XML SQL Utility und das Oracle XSQL-Servlet.

8.1 Oracle XML SQL Utility

Viele Anwendungen profitieren von der Verfügbarkeit und Abfragemöglichkeit von Informationen, die in Datenbanken zur Verfügung stehen. Eine XML-fähige Datenbank profitiert davon, dass diese Abfragen Daten liefern, die in Übereinstimmung mit dem Datenbankschema bereits als XML-Code gekennzeichnet sind. Die XML SQL Utility ist eine Menge von Java-Klassen, die diese Anwendungsabfragen akzeptieren, und unter Verwendung von JDBC an die Datenbank übergeben und die resultierenden Daten in einem XML-Format, entsprechend dem Datenbankschema der Abfrage, ausgeben. Umgekehrt kann die XML SQL Utility auch ein XML-Dokument akzeptie-

ren, das dem Datenbankschema entspricht, und die Daten ohne Tags über dieses Schema in der Datenbank speichern.

Durch das Lesen und Schreiben der XML-Daten in JDBC-fähige Datenbanken kann die XML SQL Utility eine DTD erzeugen, die das abgefragte Datenbankschema darstellt. Diese DTD kann zur Anwendungsentwicklung mit den Class Generators von Oracle verwendet werden, wie in Kapitel 2 gezeigt wurde. Weil die XML SQL Utility JDBC zur Anmeldung bei der Datenbank benötigt, gibt es verschiedene Versionen für JDBC 1.1 und 1.2.

8.1.1 Abruf von mit XML formatierten Daten

Aufgrund ähnlicher hierarchischer Beziehungen zwischen den Daten kann ein relationales Datenbankschema in XML modelliert werden. Nehmen wir an, dass wir eine Datenbank mit Bücherlisten haben, in denen eine Tabelle **BookList** mit den folgenden Spalten aufgelistet wird: BookID, Title, Author, Publisher, Year, ISBN und Description. Das Folgende ist eine typische Abfrage, die eine Anwendung bei der Datenbank ausführen würde:

```
SELECT Title, Author, Publisher, Year, ISBN
   FROM BookList WHERE BookID = 1234;
```

Wenn die Abfrage über die XML SQL Utility abgesetzt wird, liefert die Datenbank folgendes Ergebnis:

```
<?xml version="1.0"?>
<ROWSET>
  <ROW id="1">
    <TITLE>The Difference Between God and Larry Ellison: Inside Oracle
           Corporation</TITLE>
    <AUTHOR>Mike Wilson</AUTHOR>
    <PUBLISHER>William Morrow and Co.</PUBLISHER>
    <YEAR>1997>/YEAR>
    <ISBN>0688149251</ISBN>
  </ROW>
</ROWSET>
```

Diese Ausgabe kann in Form einer Zeichenkette erfolgen, wenn die Anwendung diese einfach in eine Datei schreiben möchte, oder in Form eines DOM-Objekts, das direkt an den Oracle XML-Parser für die Umwandlung durch den XSLT-Prozessor übergeben werden kann. Durch eine DOM-Ausgabe wird eine Parser-Operation vermieden, die andernfalls ausgeführt werden müsste, bevor XSL-Transformationen zugewiesen werden.

Wie im folgenden Code-Segment gezeigt wird, können Abfragen abgesetzt werden, indem sie an die Klasse **oracle.xml.sql.query.OracleXMLQuery** übergeben werden:

```
/** Simple example on using Oracle XMLSQL API; this class queries the
 database with "select * from Booklist" in scott/tiger schema;
 then from the results of query it generates an XML document */

import java.sql.*;
import java.math.*;
import oracle.xml.sql.query.*;
import oracle.jdbc.*;
import oracle.jdbc.driver.*;

public class read_samp1e
{
//=======================================
//  main()  -  public static void
public static void main(String args[]) throws SQLException
{
  String tabName = "Booklist";
  String user = "scott/tiger";
  DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

//init a JDBC connection by passing in the user
  Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:"+user+"@");
// init the OracleXMLQuery by using the initialized JDB connection
// and passing in "Booklist" as tabName
  OracleXMLQuery qry =  new OracleXMLQuery(conn,"select * from "
     +tabName );

// get the XML document in the string format which allows us
// to print it
  String xmlString = qry.getXMLString();

// print out the result to the screen
  System.out.println(" OUPUT IS:\n"+xmlString);

// Close the JDBC connection
    conn.close();
  }
}
```

Die XML SQL Utility bietet auch eine alternative Befehlszeilenschnittstelle, die bei der Erzeugung einer DTD, die mit einem bestimmten Datenbankschema verknüpft ist, nützlich ist. Angenommen, alles wurde richtig installiert, dann kann das vollständige Listing der Befehlszeilenoptionen einfach mit dem folgenden Befehl abgerufen werden:

```
java oraclexml
```

Die folgende Befehlszeile zeigt, wie die DTD erstellt wird, die mit einem bestimmten abgefragten Schema verknüpft ist:

```
java oraclexml getxml –user "scott/tiger" –withDTD "SELECT *
   FROM BookList"
```

Für die oben dargestellte Booklist-Tabelle gibt die XML SQL Utility die folgende DTD aus, die in das XML-Dokument integriert ist und von der Abfrage erzeugt wird:

```
<!ELEMENT BOOKLIST (Book, Title, Author, Publisher, Year, ISBN,
    Description)>
<!ELEMENT BookID (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
```

8.1.2 Speicherung der mit XML formatierten Daten

Ist ein Schema in einer Datenbank einmal erstellt, kann die XML SQL Utility mit der Speicherung der Daten darin beginnen, solange die XML-formatierten Daten der DTD entsprechen, die aus dem Schema erzeugt wurde. Die XML SQL Utility erlaubt die Zuweisung des XML-Dokuments an Tabellenzeilen. Der Speicher nutzt eine einfache Zuordnungstechnik für die Zuweisung der Element-Tagnamen an Spalten mit XML-Zeichenketten, die über die Standardzuordnungen in die entsprechenden Datentypen konvertiert werden. Falls das XML-Element Kindelemente besitzt, wird es an einen SQL-Objekttyp zugeordnet.

Um die XML-formatierten Daten zu speichern, initiiert die XML SQL Utility eine Einfügeanweisung, die alle Elementwerte in der VALUES-Klausel der Einfügeanweisung bindet. Der Inhalt eines jeden Zeilenelements wird an eine eigene Wertemenge zugewiesen.

Um wieder auf das BookList-Beispiel zurückzukommen, so lautet die erzeugte SQL-Anweisung für die Speicherung eines in XML formatierten Eintrags folgendermaßen:

```
INSERT INTO BOOKLIST (BookID, Title, Author, Publisher, Year, ISBN,
    Description) VALUES (?,?,?,?,?,?,?) and BIND the values,
BookID -> 1234
Title -> The Difference Between God and Larry Ellison: Inside Oracle
         Corporation
Author -> Mike Wilson
Publisher -> William Morrow & Co.
Year -> 1997
ISBN -> 0688149251
Description -> Account of Larry Ellison;
```

Das folgende Beispiel zeigt, wie dies über ein Java-Programm ausgeführt werden kann:

```
/** Simple example of using Oracle XMLSQL API; this class inserts the
 data from an XML document into the database*/

import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;
import java.net.*;

public class save_sample
{
  //=======================================
  //  main()  -  public static void
  public static void main(String args[]) throws SQLException
  {
    String tabName = "BOOKLIST";  // table into which to insert
    String fileName = "sampdoc.xml";  // file containing the xml doc
   DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

    //init a JDBC connection by passing in the user and password
    Connection conn =
      DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    //Init the OracleXMLSave class by passing in the JDBC connection
    //and the "BOOKLIST" table name.
    OracleXMLSave sav = new OracleXMLSave(conn, tabName);

    //Point to the XML file, sampledoc.xml, to save using a URL
    URL url = sav.createURL(fileName);

    //Save the data populating rows in the BOOKLIST table and
    //return the number of rows written
    int rowCount = sav.insertXML(url);

    //Print out the confirmation of the successful insertion
    System.out.println(" successfully inserted "+rowCount+
             " rows into "+ tabName);

    //Close the JDBC connection
    conn.close();

  }
}
```

Wie bei **getXML** gibt es auch eine Befehlszeilenversion für die Speicherungsfunktion namens **putXML**. Dies ist zum Massenladen von XML-Daten nützlich. Die folgende Befehlszeile lädt ein XML-Dokument mit den Buchlisten, die unserer Beispiels-DTD entsprechen:

```
java oraclexml putXML –user "scott/tiger" sampdoc.xml BookList
```

8.1.3 Aktualisierungen mit der XML SQL Utility

Aktualisierungen unterscheiden sich von Einfügungen dadurch, dass sie sich auf mehr als eine Zeile in der Tabelle beziehen können. Das zu aktualisierende XML-Element kann mehr als einer Zeile entsprechen, wenn die passenden Spalten keine Schlüsselspalten in der Tabelle sind. Aus diesem Grund erfordern Aktualisierungen eine Liste von Schlüsselspalten, mit der das Dienstprogramm die zu aktualisierenden Zeilen bestimmt. Dies zeigt das folgende Beispiel einer Buchlisten-Aktualisierung:

```
<ROWSET>
  <ROW num="1">
    <BOOK>1234</BOOK>
    <TITLE> The Difference Between God and Larry Ellison: Inside Oracle
            Corporation </TITLE>
    <AUTHOR>MIKE WILSON</AUTHOR>
    <PUBLISHER>William Morrow and Co.</PUBLISHER>
    <YEAR>1997>/YEAR>
    <ISBN>0688149251</ISBN>
  </ROW>
</ROWSET>
```

Dieses XML-Update führt zu der folgenden SQL-Anweisung, die die BookID und die Schlüsselspalten übergibt:

```
UPDATE BOOKLIST SET TITLE = ?, AUTHOR = ? WHERE BOOKID = ?
     and bind the values,
     BookID -> 1234
     TITLE -> The Difference Between God and Larry Ellison: Inside
              Oracle Corporation
     AUTHOR -> Mike Wilson;
```

Beachten Sie, dass nicht alle Spalten des ursprünglichen XML-Dokuments aktualisiert werden müssen.

Der folgende Beispielcode zeigt, wie dies in einem Java-Programm ausgeführt werden kann:

```
/**simple example using Oracle XMLSQL API; this class updates the
database data from an XML document submitted into the database*/

import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;

public class ListUpdate
{
  //===================================
  // main()  - public static void
```

```
  public static void main(String argv[]) throws SQLException
  {
    String tabName = "Booklist";      //table into which to insert
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDri-
    ver());

    // init a JDBC connection by passing in the user and password
    Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    // Init the OracleXMLSave by class passing in the JDBC connection
    // and the "Booklist" table name.
    OracleXMLSave sav = new OracleXMLSave(conn, tabName);

    // init the key column used for the update
    String [] keyColNames = new String[1];
    keyColNames[0] = "BookID";
    sav.setKeyColNames(keyColNames);

     // Assume that the user passes in this document as the first
     //argument!
    sav.updateXML(argv[0]);
    sav.close();
  }
}
```

8.1.4 Löschvorgänge mit der XML SQL Utility

Die XML SQL Utility unterstützt auch das Löschen von XML-Dokumenten. Die zu löschenden Zeilen werden wie bei den Aktualisierungen über Schlüsselspalten bestimmt. Wenn eine oder mehrere Schlüsselspalten nicht angegeben werden, wird immer noch versucht, die Spalten im Dokument zuzuweisen. Im Folgenden wird das XML-Dokument und die entsprechende SQL-Anweisung zum Löschen erzeugt:

```
<ROWSET>
  <ROW num="1">
    <BOOK>1234</BOOK>
    <TITLE> The Difference Between God and Larry Ellison: Inside Oracle
            Corporation </TITLE>
    <Author>MIKE WILSON</AUTHOR>
    <PUBLISHER>William Morrow and Co.</PUBLISHER>
    <YEAR>1997>/YEAR>
    <ISBN>0688149251</ISBN>
  </ROW>
</ROWSET>

DELETE FROM BOOKLIST WHERE TITLE = ?  AND Author = ? AND PUBLISHER = ?
AND YEAR = ? AND BOOKID = ?
     binding,
```

```
BOOKID <- 1234
TITLE <- The Difference Between God and Larrry Ellison: Inside
         Oracle Corporation </TITLE>
AUTHOR <- Mike Wilson;
PUBLISHER <- William Morrow & Co.
YEAR <- 1997
ISBN <- 0688149251;
```

Das folgende Beispiel zeigt, wie diese Löschaktion in Java mit BookID als Schlüsselspalte ausgeführt werden kann:

```
/**simple example using Oracle XMLSQL API; this class deletes the
database data from an XML document submitted into the database*/

import oracle.xml.sql.dml.*;
import java.sql.*;
import oracle.jdbc.driver.*;
import oracle.jdbc.*;

public class ListDelete
{
  //====================================
  // main()  - public static void
  public static void main(String argv[]) throws SQLException
  {
    String tabName = "Booklist";      //table into which to delete data
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDri-
    ver());

    // init a JDBC connection by passing in the user and password
    Connection conn =
    DriverManager.getConnection("jdbc:oracle:oci8:scott/tiger@");

    // Init the OracleXMLSave by class passing in the JDBC connection
    // and the "Booklist" table name.
    OracleXMLSave sav = new OracleXMLSave(conn, tabName);

    // init the key column used for the update
    String [] keyColNames = new String[1];
    keyColNames[0] = "BookID";
    sav.setKeyColNames(keyColNames);

     // Assume that the user passes in this document as the first
     //argument!
    sav.deleteXML(argv[0]);
    sav.close();
   }
}
```

8.1.5 Installation der XML SQL Utility

Die Installation der XML SQL Utility auf einem Client oder Server unterscheidet sich nicht von anderen Java-Bibliotheken. Da es in einem **.jar**-Archiv gepackt ist, müssen Sie bei der Verwendung von **xsu111.jar** oder **xsu12.jar** für Ihre Sitzung den Dateinamen und den Pfad explizit im CLASSPATH deklarieren. Die XML SQL Utility basiert auf dem Oracle XML-Parser for Java V2. Die jar-Datei ist im Archiv enthalten und muss ebenso im CLASSPATH deklariert werden. Die Datei **env.bat** für Windows und eine **env.csh**-Datei für Unix stehen zur Verfügung, um die Sitzungsumgebung einzurichten. Schließlich muss die entsprechende Menge an JDBC-Klassen, 1.1 oder 1.2, für die Anbindung an die Datenbank verfügbar sein.

Bei Anwendungen, in denen die XML SQL Utility mit einer Oracle8*i*-Datenbank kommuniziert, ist eine weitere Installationsoption verfügbar. Aufgrund der Integration der JServer Java Virtual Machine in 8*i* können Sie die jar-Datei der XML SQL Utility in die Datenbank laden, wo sie von internen oder externen Prozeduren aufgerufen werden kann. Da Version 8.1.6 mit einem Java 1.1-kompatiblen Server ausgeliefert wird, laden Sie hier **xsu111.jar**; für 8.1.7., das zu Java 1.2 kompatibel ist, verwenden Sie **xsu12.jar**. In dieser Konfiguration profitiert das Dienstprogramm von einer JDBC-Verbindung im Hauptspeicher, die die Abrufzeiten großer XML-Dokumente drastisch reduziert.

Um die Installation in 8*i* zu erleichtern, stehen für Windows die Datei **oraclexmlsqlload.bat** und für Unix die Datei **oraclexmlsqlload.csh** zur Verfügung. Bei der Ausführung laden diese Skripts alle Klassen aus **oraclexmlsql.jar**, außer den OracleXML-Befehlszeilenklassen. Sie laden auch den XML-Parser für Java V2 und führen das Skript **xmlgensql.sql** aus, das das PL/SQL-Paket **xmlgen** erzeugt und das Testskript **oraclexmltest.sql** ausführt. Dieses Paket ermöglicht die Nutzung der XML SQL Utility in SQL oder in einer PL/SQL-Prozedur.

8.1.6 Erweiterung der XML SQL Utility

Die XML SQL Utility unterstützt derzeit DOM- und String-Ausgaben, kann aber auch auf andere Arten einschließlich SAX erweitert werden. Abbildung 8-1 zeigt die Architektur aus der Perspektive eines Funktionsblocks.

Die zentralen Funktionen werden mit dem abstrakten **OracleXMLDocGen** Layer gekapselt, um ein XML-Dokument zu erzeugen. In der aktuellen Implementierung wird diese abstrakte Klasse durch **OracleXMLDocGenDOM** erweitert, um eine DOM-Ausgabe zu erzeugen, und durch **OracleXMLDocGenString**, um eine Zeichenkettenausgabe des XML-Dokuments zu erzeugen. Weitere Klassen können die Klasse OracleXML DocGen erweitern, um alle anderen Darstellungen zu unterstützen.

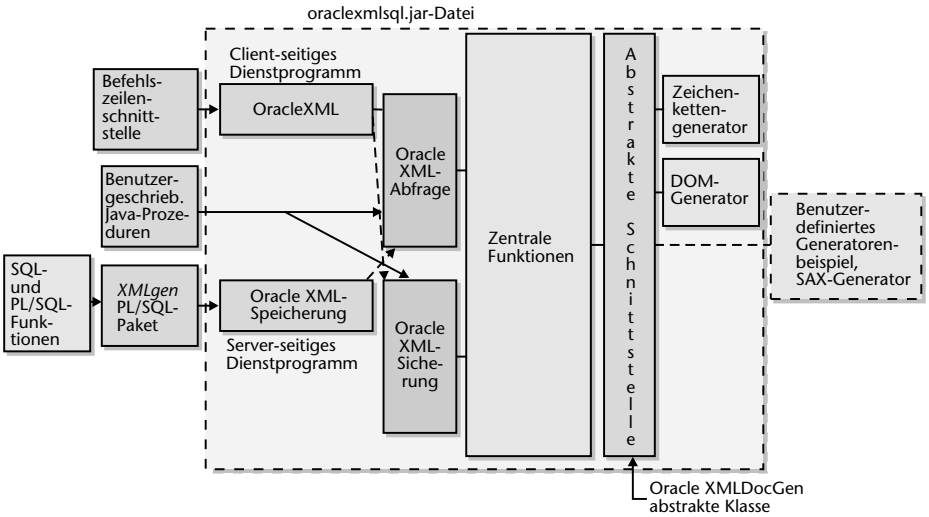

Abbildung 8-1: Die Architektur der XML SQL Utility

8.2 Oracle XSQL Servlet

Das XSQL-Servlet, das in Kapitel 3 vorgestellt wurde, baut auf der Funktionalität der XML SQL Utility und des XML-Parsers auf. Das Servlet ist in Java geschrieben und bietet eine hochrangige deklarative Schnittstelle für Entwickler und Webmaster, um die Daten über das Internet dynamisch und in eigenen Formaten darzustellen. Es ist mit den Servlet-Engines der meisten Web-Server lauffähig und erlaubt die Umwandlung einer einzigen Datenquelle automatisch auf der Basis des Client-Browsers und des Formats, das am besten zu den Funktionalitäten und denen der jeweiligen Plattform passt.

8.2.1 Die XSQL-Seiten

Das Herzstück des XSQL-Servlets ist die XSQL-Seite. Diese Seite ist einfach eine XML-Datei, die spezielle Elemente enthält, um die Aktion des Servlets festzulegen. Die folgende Datei booklist.xsql ist ein einfaches Beispiel hierfür:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="booklist.xsl"?>
<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  select * from Booklist
</xsql:query>
```

Abbildung 8-2 zeigt den Prozessfluss, wenn ein Internet-Browser diese Seite anfordert und der Web-Server die Anforderung an das XSQL-Servlet übergibt, nachdem die **xsql**-Erweiterung auf dem Server registriert wurde. Das Servlet übergibt die Seite dann an den XML-Parser, um die Anweisungen zu erhalten. In diesem Fall wird es aufgefordert, eine JDBC-Verbindung mit dem Alias von **Demo** zu eröffnen und die Abfrage „Select * from Booklist" abzusetzen. Dies geschieht durch Übergabe dieser Daten an die XML SQL Utility, die die Abfrage wie bereits beschrieben durchführt und die Ergebnisse als ein XML DOM-Objekt liefert. Schließlich übergibt das Servlet die Stylesheet-Referenz mit dem DOM-Objekt an den XSLT-Prozessor des XML-Parsers, um die Umwandlung in HTML für die Darstellung im Client-Browser vorzunehmen.

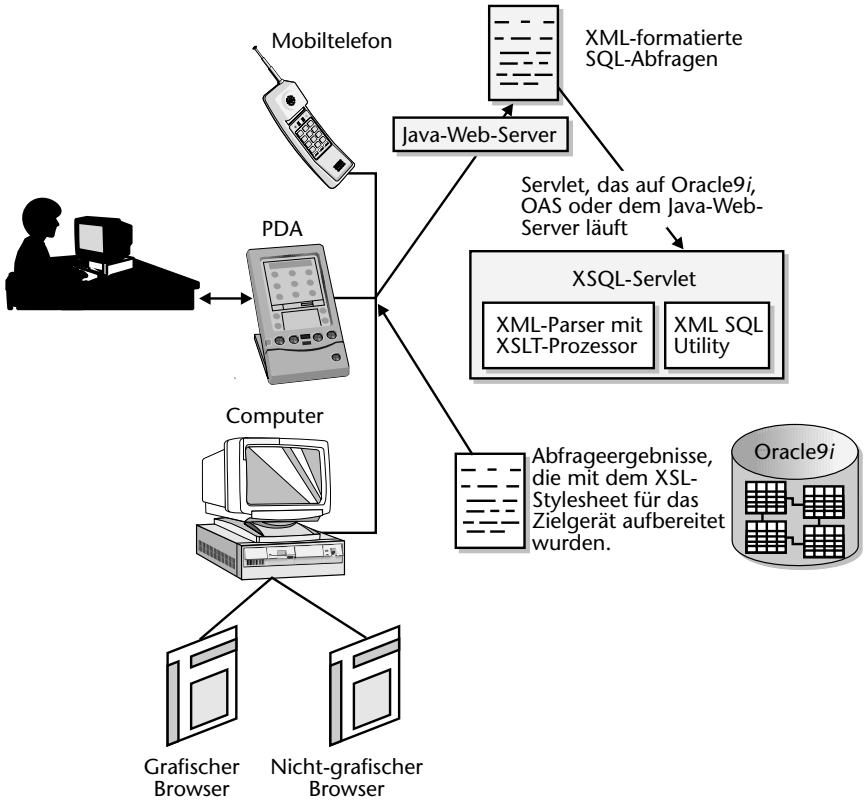

Abbildung 8-2: Verarbeitungsprozess einer XSQL-Seite

Die wesentlichen Elemente der Datei sind das Element <**xsql:query**>, das die Datenbankverbindungsinformation im Attribut **connection** enthält, und die SQL-Abfrage im Hauptteil. Der Verbindungswert ist ein Alias, der im Bereich <**connectiondefs**> der Datei XMLConfig.xml steht:

```
<connectiondefs>
   <connection name="demo">
      <username>scott</username>
      <password>tiger</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
   </connection>

   <connection name="xmlbook">
      <username>xmlbook</username>
      <password>xmlbook</password>
      <dburl>jdbc:oracle:thin:@localhost:1521:ORCL</dburl>
      <driver>oracle.jdbc.driver.OracleDriver</driver>
   </connection>

   <connection name="lite">
      <username>system</username>
      <password>manager</password>
      <dburl>jdbc:Polite:POlite</dburl>
      <driver>oracle.lite.poljdbc.POLJDBCDriver</driver>
   </connection>
</connectiondefs>
```

Dieser Abschnitt aus der XMLConfig.xml-Standarddatei zeigt die Deklaration des Datenbankverbindungsstrings und des JDBC-Treibers, der von der XML SQL Utility verwendet wird. Da sich diese Datei auf dem Server in einem Verzeichnis befindet, auf das der Client nicht zugreifen kann, bleibt diese Information sicher.

8.2.2 Installation des XSQL-Servlets

Das XSQL-Servlet ist über das OTN unter technet.oracle.com/tech/xml verfügbar und mit einem flexiblen Design für seine Installation und sein Setup ausgestattet. Es kann auf jeder JVM 1.1 oder höher und mit jeder JDBC-fähigen Datenbank eingesetzt werden. Spezielle Tests wurden mit JDK 1.1.8 und 1.2.2 auf Windows und einigen UNIX-Plattformen, wie Solaris, Linux und HP-UX, ausgeführt. Das Folgende ist eine Liste der unterstützten und getesteten Servlet-Engines:

- Allaire JRun 2.3.3
- Apache 1.3.9 mit JServ 1.0 und 1.1
- Apache 1.3.9 mit Tomcat 3.1 Beta1 Servlet Engine
- Apache Tomcat 3.1 Beta1 Web Server + Servlet Engine
- Caucho Resin 1.1
- NewAtlanta ServletExec 2.2 for IIS/PWS 4.0
- Oracle8*i* Lite Web-to-Go Server
- Oracle Application Server 4.0.8.1 (mit „JSP Patch“)

- Oracle9*i* Application Server 1.x
- Oracle8*i* 8.1.7 Beta „Aurora“ Servlet Engine
- Sun JavaServer Web Development Kit (JSWDK) 1.0.1 Web Server

In Kapitel 9 finden Sie detaillierte Anweisungen zur Installation des XSQL-Servlets und der dazugehörenden Demos.

8.2.3 Absetzen von Abfragen an das XSQL-Servlet

Mit Hilfe des XSQL-Servlets können aus Datenbankabfragen dynamische Webseiten erstellt werden. Die XSQL-Seiten können mit jeder Website verbunden werden und enthalten eine oder mehrere Abfragen, deren Ergebnisse in den entsprechenden **<xsql:query>**-Bereich in der Seite eingefügt werden. Diese Ergebnisse können über den Einsatz von Attributen im Tag **<xsql:query>** weiter angepasst werden. Tabelle 8-1 zeigt die verschiedenen Optionen.

Tabelle 8-1: Attributoptionen für das Element <xsql:query>

Attributname	Standard	Beschreibung
rowset-element	<ROWSET>	Elementname für die Abfrageergebnisse. Wenn auf eine leere Zeichenkette gesetzt, wird die Ausgabe eines Dokumentenelements unterbunden.
row-element	<ROW>	Elementname für jede Zeile in den Abfrageergebnissen. Wenn auf eine leere Zeichenkette gesetzt, wird die Ausgabe eines Dokumentenelements unterbunden.
max-rows	Alle Zeilen abrufen.	Maximalzahl der von einer Abfrage abzurufenden Zeilen. Nützlich für den Abruf der **top-N** Zeilen oder, in Kombination mit dem Überspringen von Zeilen, der **next-N** Zeilen aus einem Abfrageergebnis.
skip-rows	Keine Zeilen überspringen	Zeilenzahl, die übersprungen werden soll, bevor die Abfrageergebnisse zurückgegeben werden sollen.
id-attribute	id	Attributname für das id-Attribut für jede Zeile im Abfrageergebnis.
id-attribute-column	Zeilenzahlwert	Spaltenname zur Bereitstellung des Werts des id-Attributs für jede Zeile in den Abfrageergebnissen.

Tabelle 8-1: Attributoptionen für das Element <xsql:query> (Fortsetzung)

Attributname	Standard	Beschreibung
null-indicator	Elemente mit einem NULL-Wert auslassen.	Wenn auf **y** oder **yes** gesetzt, wird ein **null-indicator**-Attribut vom Element für alle Spalten verwendet, deren Wert NULL ist.
tag-case	Verwenden der Groß-/Kleinschreibung des Spaltennamens oder Alias aus der Abfrage.	Wenn auf **upper** gesetzt, werden die Elementnamen für Spalten im Abfrageergebnis groß geschrieben. Wenn auf **lower** gesetzt, werden die Elementnamen in den Abfrageergebnissen klein geschrieben.

Parameter können auch aus der HTTP-Anforderungszeile an die Abfrage übergeben werden. Wenn ein @ vor den Parameternamen gesetzt wird, durchsucht das XSQL-Servlet die HTTP-Anforderungsparameter und dann die <**xsql:query**>-Attribute, um ein passendes Element zu finden. Ist dieses gefunden, wird eine direkte lexikalische Ersetzung durchgeführt. Das Folgende ist ein Beispiel einer XSQL-Seite, die diese Funktion einsetzt, wenn die HTTP-Anforderungszeile
http://localhost/xsql/demo/booksearch.xsql?year=2001 lautet:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
    SELECT Title, Author, Description FROM Booklist
      WHERE Year = @year
</xsql:query>
```

Abfragen, die keine Zeilen liefern, können auch dadurch gehandhabt werden, dass ein optionales <**xsql:no-rows-query**>-Element in den <**xsql:query**>-Tags eingefügt wird. Hierdurch erhält der Benutzer eine formatierte Seite anstatt des Raw-Fehlers angezeigt. Das Folgende ist ein Beispiel, das anfänglich versucht, die Listings auf der Basis des Autorennamens abzurufen; misslingt dies, wird eine Fuzzy-Zuordnung für den weggelassenen Namen versucht:

```
<?xml version="1.0"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
    SELECT Title, Author, Description FROM Booklist
      WHERE Author = UPPER ('@author')

  <xsql:no-rows-query>
    SELECT Title, Author, Description FROM Booklist
      WHERE Author LIKE UPPER ('%@author%')
      ORDER BY Author
  </xsql:no-rows-query>

</xsql:query>
```

8.2.4 Umwandlung der XSQL-Ausgaben mit Stylesheets

Die echte Leistungsstärke des XSQL-Servlets liegt in der Fähigkeit, die Abfrageergebnisse durch die Zuweisung von XSL-Stylesheets dynamisch umzuwandeln. Die Stylesheet-Deklaration ist in der XSQL-Datei enthalten und wird einmal zugeordnet, wenn die XML-Ausgabe der Abfrage ausgegeben wird. In der Regel werden die Abfrageergebnisse in HTML umgewandelt, wie im folgenden Beispiel zu sehen. Die folgende XSQL-Seite und das verknüpfte Stylesheet geben die Ergebnisse an den anfragenden Browser als HTML-Tabelle zurück.

Searchauthor.xsql

```
<?xml version="1.0"?>
<xsql-stylesheet type="text/xsl" href="totable.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
    SELECT Title, Author, Description FROM Booklist
      WHERE Author = UPPER ('@author')

  <xsql:no-rows-query>
    SELECT Title, Author, Description FROM Booklist
      WHERE Author LIKE UPPER ('%@author%')
      ORDER BY Author
  </xsql:no-rows-query>

</xsql:query>
```

Totable.xsl

```
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <head>
    <title>Book Listing</title>
  </head>
  <body>
    <table border="1" cellspacing="0">
        <tr>
         <th><b>Author</b></th>
         <th><b>Title</b></th>
         <th><b>Description</b></th>
        </tr>
      <xsl:for-each select="ROWSET/ROW">
        <tr>
         <td><xsl:value-of select="TITLE"/></td>
         <td><xsl:value-of select="AUTHOR"/></td>
         <td><xsl:value-of select="DESCRIPTION"/></td>
        </tr>
      </xsl:for-each>
    </table>
  </body>
</html>
```

Abbildung 8-3 zeigt die Abfrageergebnisse und die folgende Umwandlung.

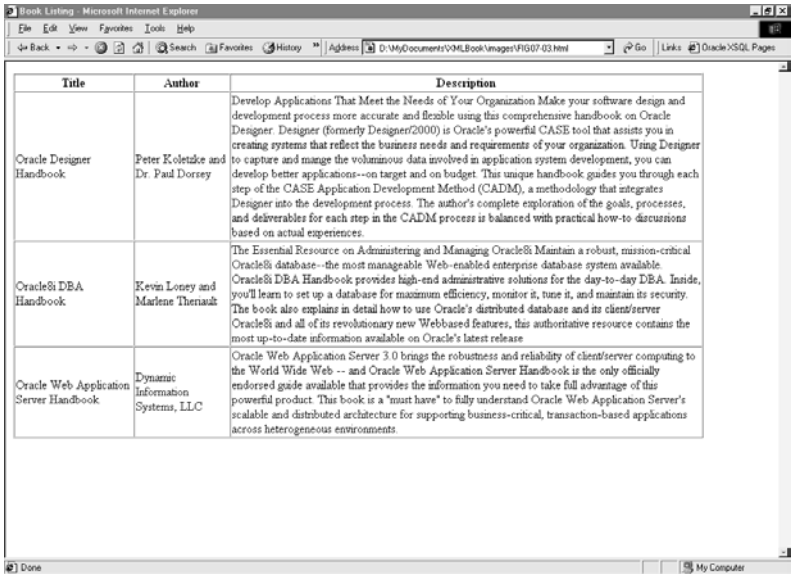

Abbildung 8-3: Formatierte Ausgabe der Seite SearchAuthor.xsql

Es werden auch mehrere Stylesheet-Deklarationen unterstützt. In solchen Instanzen wählt das XSQL-Servlet die Umwandlungsmethode, indem die Zeichenkette **user-agent** im HTTP-Header an ein optionales **media**-Attribut im Element <**xml-stylesheet**> zugeordnet wird. Bei der Zuweisung wird die Groß-/Kleinschreibung nicht berücksichtigt und das erste passende Element in der Dateireihenfolge wird zugewiesen. Das folgende Beispiel zeigt, wie mehrere Browser unterstützt werden.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" media="lynx" href="booklist-lynx.xsl"
?>
<?xml-stylesheet type="text/xsl" media="msie" href="booklist-ie.xsl" ?>
<?xml-stylesheet type="text/xsl" href="booklist.xsl"?>

<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  select * from Booklist
</xsql:query>
```

Beachten Sie, dass die letzte Stylesheet-Deklaration kein **media**-Attribut besitzt. Es wird an jede HTTP-Anforderung zugewiesen, die zu keinem anderen Element passt, und fungiert somit als das Standard-Stylesheet. Tabelle 8-2 enthält die zulässigen Attribute, die zum Element <**xml-stylesheet**> und dessen Funktionen hinzugefügt werden können.

Tabelle 8-2: Attributoptionen für das Element <xml-stylesheet>

Attribut	Erforderlich	Beschreibung
type	Ja	Muss auf den Wert **text/xsl** gesetzt sein, andernfalls wird die Anweisung <**?xml-stylesheet?**> vom XSQL Page Processor ignoriert.
href	Ja	Gibt die zu verwendende relative oder absolute URL des Stylesheets an.
media	Nein	Wenn gesetzt, wird dieser Attributwert verwendet, um die Zuordnung bei der Zeichenkette **User-Agent** (unter Beachtung von Groß-/Kleinschreibung) des anfragenden Browsers durchzuführen, so dass je nach verwendeter Software/Gerät das geeignete Stylesheet verwendet wird.
client	Nein	Wenn auf **yes** gesetzt, wird der unbearbeitete XML-Code auf den Client heruntergeladen und die Verarbeitungsoption <**?xml-stylesheet?**> für die Verarbeitung im Browser geladen. Wenn nicht anders angegeben, wird per Voreinstellung die Umwandlung auf dem Server durchgeführt.

Die letzte Methode für die Zuweisung eines Stylesheets ist die Übergabe seiner URL als ein HTTP-Parameter, wie folgt:

```
http://localhost/yourdatapage.xsql?param1=value&xml-stylesheet=
                                        yourstyle.xsl
```

Diese Technik ist vor allem für Prototypen und die Entwicklung hilfreich. Durch das Ersetzen der Stylesheet URL durch **none** wird das unbearbeitete XML-Dokument ohne jegliche Stylesheet-Verarbeitung gesendet.

8.2.5 Einfügen von XML-Dokumenten mit dem XSQL-Servlet

Durch die vollständige Nutzung der XML SQL Utility kann eine XSQL-Seite auch so eingerichtet werden, dass damit XML-Dokumente in eine Datenbank eingefügt werden können. Ein XML-Dokument kann an die Klasse **OracleXMLSave** der XML SQL Utility übergeben werden, indem das Aktionselement <**xsql:insert-request**> verwendet wird. Wie bereits erklärt, muss das Schema in der Datenbank vorhanden sein, um ein Dokument zu speichern. Wenngleich dies auf den ersten Blick als Einschränkung betrachtet werden kann, stellt die Fähigkeit des XSQL-Servlets, ein Stylesheet an ein XML-Dokument zuzuweisen, doch die erfordliche Funktionalität zur Verfügung, um die Dokumente nach Bedarf zu filtern oder umzuwandeln.

Bei dem bereits erwähnten Buchlisten-Beispiel kann eine XSQL-Seite so eingerichtet werden, dass sie Buchlisten nicht nur in dem in der Datenbank vorliegenden Format

akzeptiert, sondern praktisch in jedem Textformat. Nehmen wir an, dass ein lokaler Buchhändler seine Bücher auf diesen Seiten auflisten möchte, aber seine Buchliste andere Tags als das Datenbankschema verwendet. Die folgende XSQL-Seite akzeptiert das Buch, dass aus „Joe's Books“ via HTTP eingelesen wird, kann es über die Zuweisung des Stylesheets joesbooks.xsl in das Booklist-Datenbankschema umwandeln und dann das XML-Ergebnis an die Klasse **OracleXMLSave** zum Einfügen übergeben:

```
<?xml version="1.0">
<xsql:insert-request xmlns:xsql="urn:oracle-xsql"
             connection = "demo"
             table = "Booklist"
             transform = "joesbooks.xsl"/>
```

Es muss aber noch ein Element eingerichtet werden, damit dieses Beispiel korrekt funktioniert. Die Datenbank erzeugt die Spalte BookID, daher muss dieser Spalteneintrag für jede neue Einfügung erstellt werden. Dies kann über das Einrichten eines Triggers in der Booklist-Tabelle geschehen, der diese ID immer erzeugt, sobald ein neuer Eintrag hinzugefügt wird. Das folgende SQL-Skript erzeugt eine neue BookID, wenn ein neues Listing hinzugefügt wird, und setzt voraus, dass Sie bereits eine Sequenz namens **bookid_seq** erstellt haben:

```
CREATE TRIGGER booklist_autoid
BEFORE INSERT ON BOOKLIST FOR EACH ROW
BEGIN
   SELECT bookid_seq.nextval
   INTO :new.BookID
   FROM dual;
END;
```

Weitere vom XSQL-Servlet unterstützte Aktionselemente und ihre Funktionen beschreibt Tabelle 8-3.

Tabelle 8-3: Aktionselemente und ihre Funktionen in XSQL-Seiten

Aktionselement	Beschreibung
<xsql:query>	Führt eine beliebige SQL-Anweisung aus und nimmt die Ergebnismenge im XML-Standardformat auf.
<xsql:dml>	Führt eine SQL DML-Anweisung oder einen anonymen PL/SQL-Block aus.
<xsql:set-stylesheet-param>	Setzt den Wert des obersten XSQL-Stylesheet-Parameters. Der Parameterwert kann gesetzt werden, indem das optionale **value**-Attribut gesetzt wird, oder indem eine SQL-Anweisung als Elementinhalt angegeben wird.

Tabelle 8-3: Aktionselemente und ihre Funktionen in XSQL-Seiten (Fortsetzung)

Aktionselement	Beschreibung
<xsql:insert-request>	Fügt das (optional umgewandelte) XML-Dokument, das in der Anforderung enthalten war, in eine Datenbanktabelle oder View ein. Wenn es sich um ein HTML-Formular handelte, wird das angefügte XML-Dokument aus HTTP-Anforderungsparametern, Cookies und Sitzungsvariablen zusammengestellt.
<xsql:include-xml>	Nimmt beliebige XML-Ressourcen an jedem beliebigen Punkt Ihrer Seite über die relative oder absolute URL auf.
<xsql:include-request-params>	Nimmt Schlüsselinformationen wie HTTP-Parameter, Sitzungsvariablenwerte und Cookies in Ihre XSQL-Seite auf, um sie in Ihrem Stylesheet adressieren zu können.
<xsql:include-xsql>	Nimmt die Ergebnisse einer XSQL-Seite an einem beliebigen Punkt in einer anderen auf.
<xsql:include-owa>	Nimmt die Ergebnisse der Ausführung einer Stored Procedure, die die *Oracle Web Agent* (*OWA*)-Pakete nutzt, in die Datenbank auf, um XML zu erzeugen.
<xsql:action>	Ruft einen benutzerdefinierten Action Handler auf, der in Java implementiert wird, um benutzerdefinierte Logik auszuführen und angepasste XML-Informationen in eine XSQL-Seite aufzunehmen.
<xsql:ref-cursor-function>	Nimmt die XML-Standarddarstellung der Ergebnismenge eines Cursor auf, der von einer gespeicherten PL/SQL-Funktion zurückgegeben wird.
<xsql:set-page-param>	Setzt einen (lokalen) Parameter auf Seitenebene, auf den in nachfolgenden SQL-Anweisungen auf der Seite verwiesen werden kann. Der Wert kann mit einem statischen Wert, dem Wert eines anderen Parameters oder den Ergebnissen einer SQL-Anweisung gesetzt werden.
<xsql:include-param>	Nimmt einen Parameter und seinen Wert als Element in die XSQL-Seite auf.
<xsql:set-session-param>	Setzt einen HTTP-Parameter auf Sitzungsebene. Der Wert kann mit einem statischen Wert, dem Wert eines anderen Parameters oder den Ergebnissen einer SQL-Anweisung gesetzt werden.

Tabelle 8-3: Aktionselemente und ihre Funktionen in XSQL-Seiten (Fortsetzung)

Aktionselement	Beschreibung
<xsql:set-cookie>	Setzt ein HTTP-Cookie. Der Wert kann mit einem statischen Wert, dem Wert eines anderen Parameters oder den Ergebnissen einer SQL-Anweisung gesetzt werden.
<xsql:insert-param>	Fügt den Wert eines einzelnen Parameters, der XML-Code enthält, ein. Kann optional eine Umwandlung durchführen, um ihn in das Standardformat zu überführen.

8.2.6 Aktualisieren von Daten mit dem XSQL-Servlet

Viele Anwendungen erfordern, dass die Daten oder Dokumente aktualisiert statt komplett ersetzt werden. In ähnlicher Weise wie die BookID automatisch erstellt wurde, können Sie eine Triggerart verwenden, um diese Funktionalität bereitzustellen.

Oracle8*i* stellt einen INSTEAD OF-Trigger zur Verfügung, mit dem eine Stored Procedure in PL/SQL oder Java aufgerufen werden kann, wenn ein beliebiges INSERT versucht wird. Diese Trigger nutzen die Object Views von Oracle8*i*, die mit INSERT assoziiert sind.

Wenn beispielsweise eine Booklist-Tabelle aktualisierbar sein soll, können Sie anfänglich nach einer eindeutigen Kombination aus Titel und Autor suchen und, wenn diese gefunden wird, ein UPDATE anstatt eines INSERTs durchführen. Um dies einzurichten, müssen Sie eine Object View erstellen, die der Booklist-Tabelle entspricht. Dies kann mit der folgenden SQL-Anweisung durchgeführt werden:

```
CREATE VIEW Booklistview AS
SELECT * FROM Booklist;
```

Als Nächstes muss der Trigger erstellt werden und mit dieser View verknüpft werden. In diesem Beispiel wird PL/SQL verwendet, aber dies könnte auch mit einer Java Stored Procedure durchgeführt werden.

```
CREATE OR REPLACE TRIGGER insteadOfIns_booklistview
INSTEAD OF INSERT ON booklistview FOR EACH ROW
DECLARE
  notThere BOOLEAN := TRUE;
  tmp      VARCHAR2(1);
  CURSOR chk IS SELECT 'x'
                FROM BOOKLIST
                WHERE TITLE = :new.title
                AND AUTHOR  = :new.author;
BEGIN
  OPEN chk;
  FETCH chk INTO tmp;
```

```
  notThere := chk%NOTFOUND;
  CLOSE chk;

  IF notThere THEN
    UPDATE INTO Booklist(TITLE,
                         AUTHOR,
                         PUBLISHER,
                         YEAR,
                         ISBN,
                         DESCRIPTION)
          VALUES (:new.title,
                  :new.author,
                  :new.Publisher,
                  :new.Year,
                  :new.ISBN,
                  :new.Description);
  END IF;
END;
```

Schließlich muss die XSQL-Datei wie folgt geändert werden, um die Booklist-View anstelle der Booklist-Tabelle zu aktualisieren.

```
<?xml version="1.0">
<xsql:insert-request xmlns:xsql="urn:oracle-xsql"
             connection = "demo"
             table = "Booklistview"
             transform = "joesbooks.xsl"/>
```

Als letzter Hinweis sei erwähnt, dass die Eindeutigkeit über die Kombination von Titel und Autor geprüft wird und so ein eindeutiger Index erzeugt werden kann, um die Prüfung zu beschleunigen und die Leistung zu verbessern. Die folgende SQL-Anweisung erzeugt den Index:

```
CREATE UNIQUE INDEX booklist_index ON booklist(Title, Author);
```

Während das obige Beispiel erläuterte, wie eine Aktualisierung mit der Triggerfunktionalität einer Object View durchgeführt wird, können Sie auch die Aktualisierungseinrichtung der XML SQL Utility aus dem XSQL-Servlet unter Verwendung der folgenden einfachen **.xsql**-Datei verwenden:

```
<?xml version="1.0"?>
<xsql:dml connection="demo" xmlns:xsql="urn:oracle-xsql">
  update Booklist set status='S' where BookID = '1';
</xsql:dml>
```

Dieses Beispiel illustriert die Einfachheit und Leistungsstärke der XML-basierten Schnittstelle von XSQL.

8.3 Eine Website auf Basis von XML

Bei herkömmlichen Websites muss der Inhalt für eine bestimmte Bildschirmgröße vorformatiert sein, wobei wenig oder gar keine Möglichkeit für eine automatische Entdeckung und Anpassung eine Clients gegeben ist. Die meisten Sites, die eine Anpassung an verschiedene Browser versuchen, sind auf die Auswahl durch den Benutzer zwischen Versionen mit „Frame“ oder „Nicht-Frame“ und „Text“ oder „Text+Grafik“ beschränkt. Diese Seiten werden in der Regel statisch gespeichert und müssen manuell aktualisiert werden, so dass sich hier Fehler einschleichen oder Daten verloren gehen können. Alle Formatänderungen müssen auf jeder Seite wiederholt werden.

Die Einführung von Cascading Stylesheets verbesserte diese Situation, indem ein Großteil der HTML-Formatierung in eine eigene Datei verschoben wurde. Wenngleich dies viele Formatierungsaufgaben aus der HTML-Quelldatei entfernt, bleibt es abhängig von einer statischen Quelldatei mit HTML-Formatierung. Die Bereitstellung echter Datenquellen, die über Stylesheets dynamisch im Browser dargestellt werden, ist das Ziel vieler E-Commerce-Unternehmen. Das beschleunigt die Bereitstellung der aktuellen Daten und hat zu unternehmensweit verfügbaren Datenbanken wie Oracle geführt, die die Website-Backends erweitert haben.

8.3.1 Die XML-fähige Lösung

In Kapitel 3 stellten wir den Einsatz der XML-Funktionalität zur Beschreibung von Daten in strukturierter Form vor, wobei in Kombination mit XSL praktisch jede textbasierte Umwandlung durchgeführt und hiermit ein echtes, eindeutiges und einheitliches Daten-Repository angelegt werden kann. Dieses Daten-Repository kann als Backend für ein Anzeigesystem dienen, ganz gleich, ob dies ein Browser, ein PDA, ein Mobiltelefon oder ein Pager ist. Statt statische Webseiten aufzurufen, können integrierte Links Datenbankabfragen erzeugen, die stets die aktuellsten Inhalte zur Verfügung stellen.

Auf dem Backend können Datenbanken eine Vielzahl von Dateneingaben akzeptieren, einschließlich derer, die in XML formatiert sind und die Daten zur sofortigen Verwendung speichern. Auch Content Repositorys können erstellt werden, die wiederverwendbare Dokumentenfragmente oder Module enthalten, die bei Bedarf von XSL-Stylesheets zusammengefügt werden können, um das Veröffentlichen von Online-Dokumenten erheblich zu vereinfachen. Diese Trennung von Inhalt und Darstellung vereinfacht auch den Produktionsprozess, da die Designer an Stylesheets arbeiten können, ohne dass die Gefahr besteht, dass inhaltliche Fehler auftauchen.

Ein Praxisbeispiel für eine solche Website ist eine Zusammenstellung von Grundstücken, die Listings aus vielen verschiedenen *Multiple Listing Services* (*MLS*) an einer Stelle zusammenfassen können. Das Szenario ist, dass Sie eine Website haben, die diese

MLS-Listings aus einer Vielzahl von Quellen mit jeweils eigenen XML-Formaten akzeptiert. Ein XSL-Stylesheet, das für jede Quelle eindeutig ist, wird der XML-Dateneingabe zugewiesen, und die Daten werden in ein normalisiertes Format umgewandelt, um die Daten in die Datenbank einzufügen.

8.3.2 Die Design-Anforderungen

Um eine gute Website zu entwerfen, müssen zwei allgemeine Design-Anforderungen erfüllt werden: Die Datenbank muss XML akzeptieren und die entsprechende XSL-Umwandlung für den Quellcode durchführen können. Ebenso muss sie die Listings dynamisch auf mehrere Arten skalierbar und schnell darstellen können.

Das System muss auch in der Lage sein, mit vielen verschiedenen Formaten umzugehen, die entsprechenden Typen für die Anzeige auf dem System des Anforderers auszuwählen oder im Falle eines Mobiltelefons oder Pagers keine Grafiken anzuzeigen. Das System sollte idealerweise nicht nur auf direkte Anforderungen reagieren, sondern auch Meldungen erzeugen, wenn Listings empfangen werden, die den vordefinierten Kriterien entsprechen. Dies kann besonders für Immobilienmakler hilfreich sein, die nach neuen Listen suchen, um sie Ihren Kunden zu zeigen.

8.3.3 Die Architektur

Dieses System ist ideal als Repository für eine Datenbank geeignet. In diesem Fall können Sie Oracle8*i* verwenden, um bei Bedarf sowohl die Daten wie auch die Grafiken zu speichern. Object Views werden eingerichtet, um die Daten mit den Grafiken zu verbinden und Internet-Browsern die Anzeige des vollständigen Listings nach der XSL-Umwandlung mit dem entsprechenden Stylesheet zu ermöglichen.

Abbildung 8-4 zeigt ein Diagramm von der Gesamtarchitektur der Immobiliensite. XML-Dateneingaben werden vom XSQL-Servlet akzeptiert, in ein normalisiertes Format übertragen und in der Datenbank gespeichert.

Clients können mit einem Browser die Listings durchsuchen oder die verfügbaren Listings auf der Basis verschiedener vordefinierter Kriterien durchsuchen. Das Format wird automatisch entsprechend der Verarbeitungsmöglichkeiten des anfordernden Browsers verschlüsselt. Darüber hinaus können Benachrichtigungen über den Statuswandel eines Listings oder dessen Aktualisierung versendet werden.

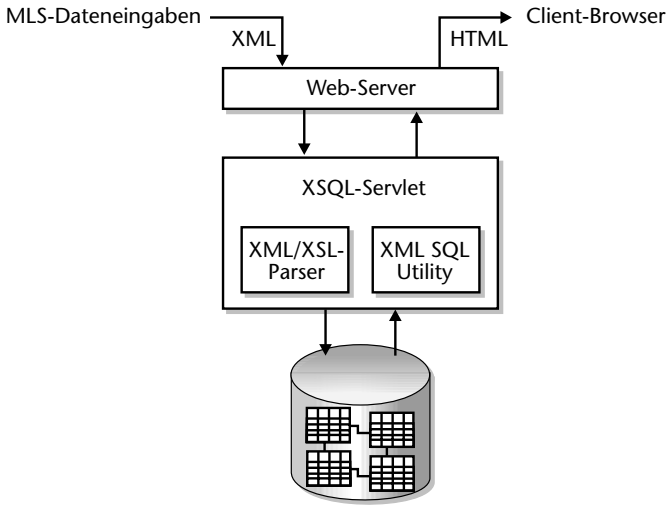

Abbildung 8-4: Funktionsdiagramm der Immobiliensite

8.3.4 Eine Beispielimplementierung

Um dieses System zu implementieren, müssen Sie zuerst ein normalisiertes MLS-Schema entwerfen. Zu diesem Zweck stellt die folgende DTD das Datenmodell dar:

```
<!ELEMENT LISTING (Listing, MLS_Number, Address, Area, City, State,
                   Zipcode, Description, AskPrice, Agent, ImageURI,
                   Category, Status)>
<!ELEMENT Listing (#PCDATA)>
<!ELEMENT MLS_Number (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Area (#PCDATA)>
<!ELEMENT City (#PCDATA)>
<!ELEMENT State (#PCDATA)>
<!ELEMENT Zipcode (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT AskPrice (#PCDATA)>
<!ELEMENT Agent (#PCDATA)>
<!ELEMENT maNAgeR (#PCDATA)>
<!ELEMENT CATEGORY (SFH | CONDO | APTBLDG | DUP | THM | COM | LOT)>
<!ELEMENT Status ( A| P | S | C)>
```

Diese DTD kann als Schablone für die Erstellung von Stylesheets dienen, die für spezielle MLS-Dienste kodiert sind. Wenn beispielsweise der NorCal-Service **Neighborhood** benutzt statt **Area**, und **Zip** anstatt **Zipcode**, kann das folgende Stylesheet die Umwandlung durchführen:

```
<?xml version="1.0"?>
<ROWSET xmlns:xsl=http://www.w3.org/1999//Transform
        xsl:version="1.0">
  <xsl:for-each select="Norcal/LISTING">
  <ROW>
  ...
  <Area><xsl:value-of select"Neighborhood"/></Area>
  ...
  <Zipcode><<xsl:value-of select="Zip"/></Zipcode>
  ...
  </ROW>
 </xsl:for-each>
</ROWSET>
```

Wurden alle Stylesheets angelegt, kann eine XSQL-Seite eingerichtet werden, um jeden Service zu akzeptieren und die entsprechende Umwandlung zuzuordnen, wie bei der folgenden XSQL-Seite mit dem bereits beschriebenen Aktionselement **<xsql:insert-request>**.

```
<?xml version="1.0"?>
<xsql:insert-request xmlns:xsql="urn:oracle-xsql"
             connection = "demo"
             listingview = "LISTING"
             transform = "norcal.xsl"/>
```

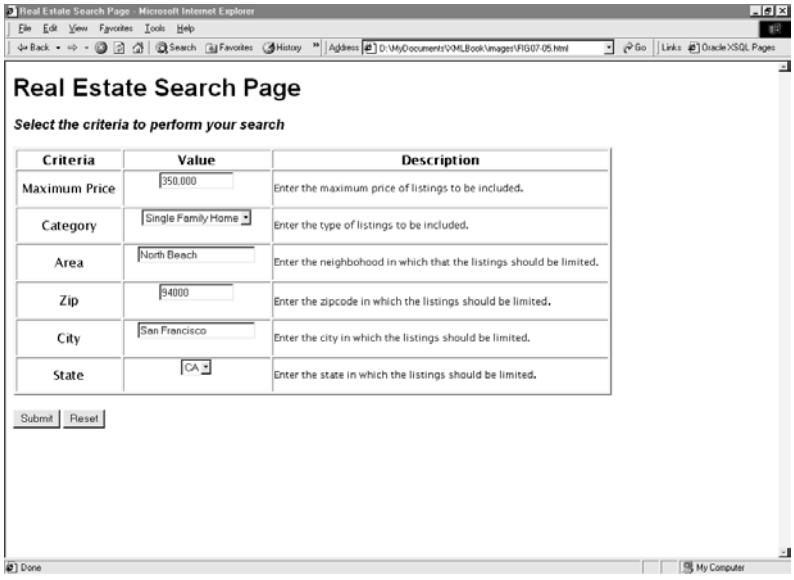

Abbildung 8-5: Beispiel für MLS-Listing der Suchseite

XSQL-Seiten können auch auf der Client-Seite dazu verwendet werden, Listings abzufragen und verschiedene Seiten zu erstellen, wie Übersichtsseiten auf der Basis von **Area** oder **Zipcode,** oder zur Verfeinerung von Suchen mit Preisspannen oder nach Immobilientyp, wie in Abbildung 8-5 gezeigt.

Sie können diese Suchaktionen durchführen, indem Sie Links auf XSQL-Seiten erstellen, die bestimmte Abfragen ausführen, oder indem Sie die **@parameter**-Technik zur Übergabe der Kriterien an eine allgemeine XSQL-Seite verwenden.

Mit dem Aufkommen von webfähigen Handys, PDAs und Pagern kann diese Site auch deren eingeschränkte Anzeigeanforderungen erfüllen, indem die **user-agent**-Zeichenkette im HTTP-Header und das Attribut **media** in der Stylesheet-Deklaration genutzt werden. Das folgende XSQL-Beispiel unterstützt den Palm Pilot, ein Unwired Planet-Handy und die Standardbrowser:

```
<?xml version="1.0"?>
<?xml-stylesheet type"text/xsl" media="Palm" href="ListingPP.xsl"?>
<?xml-stylesheet type"text/xsl" media="Mozilla" href"istingMZ.xsl"?>
<?xml-stylesheet type"text/xsl" media="UP" href="istingWML.xsl"?>
<?xml-stylesheet type"text/xsl" href="Listing.xsl"?>

<xsql:query connection="demo" xmlns:xsql="urn:oracle-xsql">
  select * from Listingview WHERE
           Zipcode=94065
</xsql:query>
```

8.3.5 Erweiterung des Beispiels

Mit dem bereits beschriebenen Trigger INSTEAD OF INSERT können Sie diese Website proaktiv machen, indem registrierte Benutzer bei neuen Listings, anstehenden Sonderverkäufen und Schließungen benachrichtigt werden, wenn diese eingefügt oder aktualisiert werden. Die Stored Procedure, die vom Trigger aufgerufen wird, könnte eine Registrierungstabelle der Interessenten mit Email-Adressen abfragen und eine Email an diejenigen senden, die für einen bestimmten Bereich, Postleitzahl etc. registriert sind.

Eine vollständige Service-Site könnte die finanziellen Informationen verbinden, um Kunden oder Makler über Darlehensfreigaben, geänderte Grundbucheinträge und andere Mitteilungen bei einer Immobilientransaktion zu informieren. Schließlich könnte die Site mit der zunehmenden Verbreitung von Internetauktionen auch um eine Online-Auktion erweitert werden, die Gebote akzeptiert und die Bieter über eine breite Palette an Kommunikationsgeräten benachrichtigt.

8.3.6 Oracle9*i* Application Server Wireless Edidtion

Ein Produkt, das bereits über einen Großteils der Funktionalitäten verfügt, die in diesem Beispiel beschrieben werden, ist Oracle9*i* Application Server Wireless Edition. Es ist vor allem für Anbieter von funk- und kabelbasierten Diensten gedacht und ermöglicht, dass ein einzelnes Inhalts-Repository eine breite Palette von Diensten über Gateways anbietet, um Formatierungen speziell für das anfordernde Gerät zur Verfügung zu stellen.

Abbildung 8-6 zeigt die Architektur der Oracle9*i* Application Server Wireless Edition. XML-formatierte Informationseingaben werden an eine Oracle8*i*-Datenbank angebunden, die den Inhalt in einer normalisierten Art speichert. Oracle9*i* Dynamic Services, die die Websites durchsuchen und die Daten in XML zurückgeben, können weitere Informationen liefern.

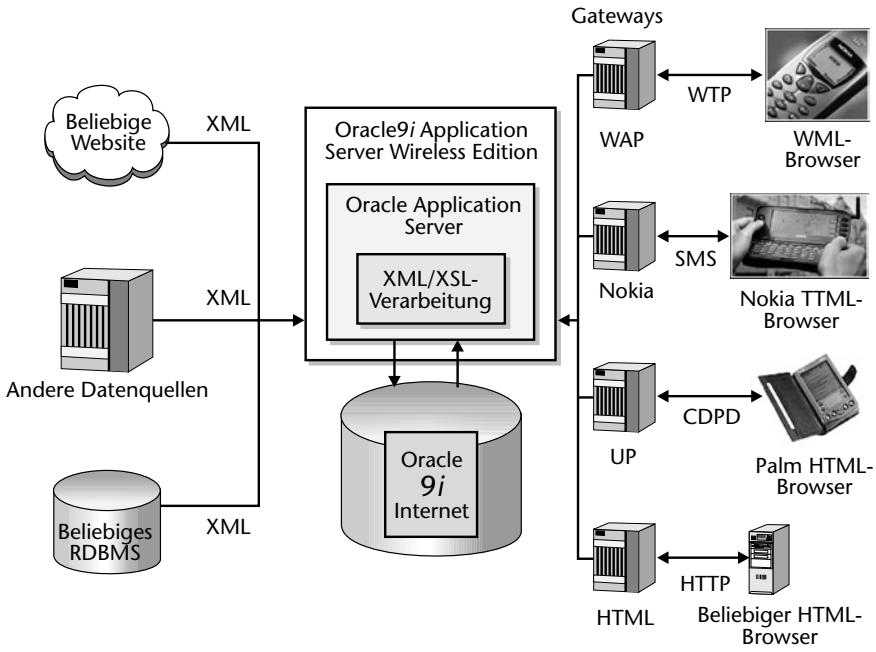

Abbildung 8-6: Die Architektur von Oracle9*i* Application Server Wireless Edition

Auf der Ausgabeseite werden gerätespezifische Gateways eingerichtet, um eine intelligente Interaktion mit den verschiedensten zweiwegigen funk- und kabelbasierten Geräten zu ermöglichen. Die Benutzer können eigene Views und Inhaltsauswahlen anpassen. Über Portlets, dies sind vollständig definierte HTML/XML-Bereiche in einer Webseite, die die Basiszugriffsfunktionalität auf Informationen bereitstellen, können Benutzer sogar im nicht-verbundenen Modus arbeiten, was für viele Benutzer mit funkbasierten Geräten ideal ist.

8.4 XML-gestützte Nachrichtenübertragung für das E-Business

Die Anforderung, Daten zwischen Anwendungen und Unternehmen auszutauschen, wächst von Tag zu Tag. Historisch gesehen hat der Datenaustausch einen hohen Grad an technischen Übereinkünften über Datentypen, Strukturen, Übertragungsprotokolle und so weiter erfordert. Dienste wie CORBA stellten diese Fähigkeit bereit, waren aber für den Einsatz im Internet, bei Übertragung über Firewalls und beim Einrichten von Ad-hoc-Sitzungen nur begrenzt verwendbar. Benötigt wurde eine Methode, damit Anwendungen Daten in Textform über unterschiedliche Übertragungswege und Protokolle in selbstbeschreibender Weise senden und empfangen konnten. Unter diesen Voraussetzungen benötigten Anwendungen keine strenge Zugehörigkeit zu einer Technologie, um zu kommunizieren. Dies erlaubte Unternehmen dann die Integration von Anwendungen über das Internet, so dass sie hiermit die Forderungen der neuesten wirtschaftlichen Entwicklungen erfüllen und einen ganz neuen Anwendungsbereich für Application Hosting eröffnen können.

8.4.1 Die XML-fähige Lösung

Durch das Aufkommen von XML und seiner verwandten Technologien wie XSL, XML-Schema, XML Query und XPath wird die Möglichkeit zur echten unternehmensweiten Anwendungsintegration Wirklichkeit. Ob für Bestellungen oder Aktualisierungsnachrichten – Anwendungen müssen strukturierte Daten austauschen. XML-formatierte Daten machen Metadaten und deren Struktur, die über festkodierte Anwendungslogik übertragen werden muss, überflüssig. XML-Schemadefinitionen oder DTDs können aufgenommen werden, um die Nachrichten zu validieren. Mit XSL-Stylesheets können diese Nachrichten in spezielle Formate für bestimmte Anwendungen umgewandelt werden. Schließlich kann die gesamte Verarbeitung über generische Komponenten vorgenommen werden, die auf offenen Standards basieren, deren Entwicklung wiederum den Unternehmen garantiert, dass sie weithin unterstützt und eingesetzt werden.

Da dies ein sehr weitgefasster Anwendungsbereich ist, kehren wir zum Buchlisten-Beispiel zurück und erstellen einen Online-Bookshop, der diese verschiedenen XML-Technologien nutzt. Das Szenario ist das Folgende: Wir haben einen Online-Bookshop und möchten gleichzeitig die Listings der angebotenen Bücher abrufen können und den Kunden erlauben, Bücher über das Internet zu kaufen. Wir würden dieses Bücherlisten auch gerne aus anderen im Einsatz befindlichen Bookshops zusammenstellen, um einen landesweiten Kundenstamm zu unterstützen.

8.4.2 Die Design-Anforderungen

Um diese Website und eine gute Anwendung zu erstellen, müssen wir Datenmodelle für die verschiedenen Objekte entwerfen, die benutzt werden. Da die Datenbank vor allem als Primärspeicher dient und bei der Modellierung die Komponente mit den meisten Constraints ist, sollten wir zuerst das Datenbankschema entwerfen. Dann wird die XML SQL Utility eingesetzt, um die DTDs für die richtige Abbildung zu erzeugen. Der Class Generator konvertiert diese DTDs in die DOM-Klassen, die im Anwendungs-Frontend zur Erstellung von XML-Dokumenten benutzt werden, die dann von der XML SQL Utility in die Datenbank eingefügt werden.

Das System muss in der Lage sein, Buchlisten verschiedener Einzelanbieter wie auch Massenlistings von anderen Buchhändlern zu verarbeiten. Außerdem müssen Buchlisten interessierten Käufern angezeigt, Verkaufstransaktionen verarbeitet und die Verkäufer benachrichtigt werden. Schließlich müssen die Zahlungen der Kunden an die Verkäufer übermittelt werden.

Ebenso muss die Gesamtleistung bei den Designanforderungen berücksichtigt werden. Während der Abruf der Daten im XML-Format gut skalierbar ist, können komplexe Umwandlungen und die Anforderung von einem Thread pro Instanz zu einem erheblichen Flaschenhals bei schweren Belastungen führen. Die Minimierung der Stylesheet-Verarbeitung und die Nutzung von Caches auf der mittleren Ebene, wie iCache von Oracle, kann wesentlich zur Vermeidung dieses Flaschenhalses beitragen.

8.4.3 Die Architektur

Das System kann implementiert werden, indem Oracle8*i* als Daten-Repository und der Oracle Application Server gleichzeitig als Web-Server und mittlere Schicht genutzt wird. Clients können mit der Site über die Standardbrowser interagieren. Die Buchhändler können XML-formatierte Listings absenden und Nachrichten über Verkäufe empfangen. Abbildung 8-7 zeigt ein Funktionsdiagramm des Systems.

Um eine Buchliste für den Laden einzufügen, gibt der Verkäufer die Daten in eine *Java Server Page* (*JSP*) ein, die die DOM-Klassen aufruft, die das einzufügende XML-Dokument erstellen. Das Dokument wird dann geparst und die Daten werden in das Booklist-Schema eingetragen. Darüber hinaus wird die Kontoinformation des Verkäufers in ein Account-Schema eingefügt. Clients erhalten die Buchlisten nach Kategorie oder als Ergebnis von Suchen angezeigt, indem sie mit den HTML-Seiten interagieren, die vom XSQL-Servlet erstellt wurden. Buchkäufe werden als Transaktion in der Datenbank festgehalten und abgespeichert, und der Verkäufer wird per Email benachrichtigt.

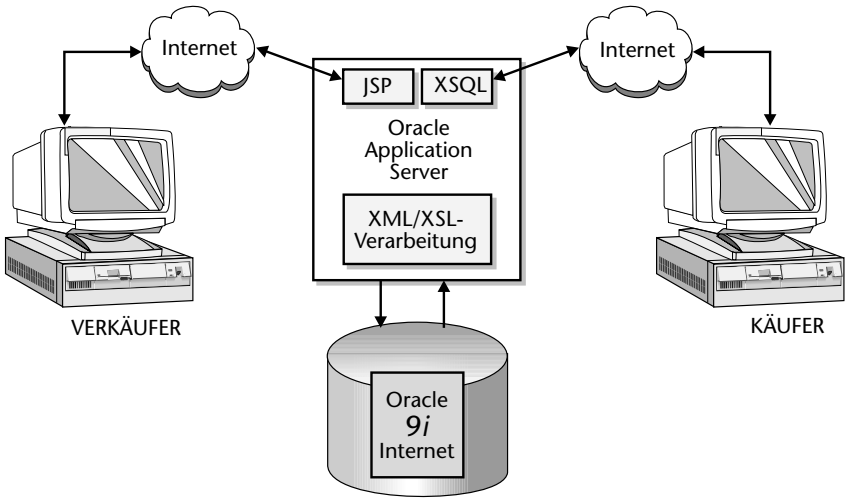

Abbildung 8-7: Funktionsdiagramm des Online-Buchladens

Obwohl dies natürlich noch keine vollständige Anwendung und Architektur darstellt, ist die Erörterung und Beispielimplementierung der verschiedenen Teile ein gutes Beispiel für die Verwendung von XML und kann als Grundlage für ähnliche Anwendungen dienen.

8.4.4 Eine Beispielimplementierung

Durch die Erweiterung des bereits vorgestellten Booklist-Schemas und das Hinzufügen eines Client-Schemas können wir die Besonderheiten dieser Implementierung aufzeigen. Das Folgende ist eine XML-DTD für die erweiterte Buchliste:

```
<!ELEMENT BOOKLIST (Book, Title, Author, Publisher, Year, ISBN,
                    Description, Category, Cost, Status)>
<!ELEMENT BookID (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Year (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Category (FICTION | NONFICTION | REFERENCE | TECHNCAL)>
<!ELEMENT Cost (#PCDATA)>
<!ELEMENT Status ( A | S)>
```

Diese DTD kann als Eingabe für den Class Generator dienen, um ein Set von DOM-Klassen zu erstellen, die die entsprechenden XML-Dokumente erzeugen können. Das Folgende ist ein Befehlszeilenbeispiel:

```
Java samplemain booklist.dtd
```

Es werden Java-Quelldateien für alle Elemente erstellt, die dann mit dem Javac-Compiler in Java-Klassen kompiliert werden können. Als Nächstes wird ein JSP mit diesen Klassen erstellt, wie das folgende Listing zeigt:

```
<HTML>
  <HEAD>
    <TITLE>Book Submission</TITLE>
  </HEAD>
<BODY bgcolor = "#ffffff">
<%@ page import="java.io.*" %>
<%
String Title = request.getParameter("TITLE");
String Author = request.getParameter("AUTHOR");
String Publisher = request.getParameter("PUBLISHER");
String Year = request.getParameter("YEAR")
String ISBN = request.getParameter("ISBN");
String Description = request.getParameter("DESCRIPTION")
String Category = request.getParameter("CATEGORY");
String Cost = request.getParameter("COST")
String status = new String("A");  // A for available
try {
    out.println(AUTHOR + " " + TITLE + " " + CATEGORY + " " + ISBN
                + " " + PUBLISHER + " " + YEAR + " " + DESCRIPTION
                + " " + COST +" "+ STATUS);

    Booklist entry = new Booklist();
    oracle.xml.parser.v2.DTD dtd = entry.getDTDNode();
    TITLE t = new TITLE(TITLE);
    AUTHOR a = new AUTHOR(AUTHOR);
    PUBLISHER p = new PUBLISHER(PUBLISHER);
    YEAR y = new YEAR(YEAR);
    ISBN i = new ISBN(ISBN);
    CATEGORY c = new CATEGORY(CATEGORY);
    COST ct = new COST(COST);
    STATUS st = new STATUS(STATUS);

    entry.addNode(t);
    entry.addNode(a);
    entry.addNode(p);
    entry.addNode(y);
    entry.addNode(i);
    entry.addNode(c);
    entry.addNode(ct);
    entry.addNode(st);
    entry.print(new FileOutputStream("result.xml"));
   }
catch (Exception e) {
    out.println("...." + e + "..e.");
    e.printStackTrace(new PrintWriter(out));
}
```

```
%>
</BODY>
</HTML>
```

Durch den Einsatz von JavaScript und das Bereitstellen einer Formularschnittstelle kann dieses JSP aus einer HTML-Seite aufgerufen werden. Die folgende HTML-Seite ist ein Beispiel hierfür, sie erstellt das Bucheintragsformular, das in Abbildung 8-8 dargestellt wird.

```
<html>
<head>
  <title>Book Submission Form</title>
</head>
<body bgcolor="#ffffff">
<script language="JavaScript">
  function checkRequiredFields() {
     var frmR = window.document.frmReport;
     if (frmR.title.value  == "") {
       alert("Please enter a value for the title");
       return false;
     }
     if (frmR.author.value  == "") {
       alert("Please enter the author");
       return false;
     }
     if (frmR.publisher.value  == "") {
       alert("Please enter the publisher");
       return false;
     }
     if (frmR.year.value  == "") {
       alert("Please enter the Year");
       return false;
     }
     if (frmR.isbn.value  == "") {
       alert("Please enter the ISBN number");
       return false;
     }
     if (frmR.description.value  == "") {
       alert("Please enter the description");
       return false;
     }
     if (frmR.category.value  == "") {
       alert("Please enter the category - Fiction, Nonfiction,
             Reference, or   Technical");
       return false;
     }
     if (frmR.isbn.cost  == "") {
       alert("Please enter the cost");
       return false;
     }
```

```
  frmR.action = "procSell.jsp";
  frmR.submit();
  return true;
}
/script>
</p>
<h3><img src="openbook.gif"> Oracle Used Books Store  <a
href=""><font size="-1"><i>Main</i></font></a></h3>
<br>
<br>
<br>
<h4>Seller Form</h4>
<form name="frmReport">
  <table width="90%" align="center">
    <tr>
      <td><div align="left"><p><font color="red">*</font>Title</td>
      <td><input type="text" name="TITLE" value size="60"></td>
    </tr>
    <tr>
      <td><div align="left"><p><font color="red">*</font>Author</td>
      <td><input type="text" name="AUTHOR" value size="60"></td>
    </tr>
    <tr>
     <td><div align="left"><p><font color="red">*</font>Publisher</td>
      <td><input type="text" name="PUBLISHER" value size="60"></td>
    </tr>
    <tr>
      <td><div align="left"><p><font color="red">*</font>Year</td>
      <td><input type="text" name="YEAR" value size="4"></td>
    </tr>
    <tr>
      <td><div align="left"><p><font color="red">*</font>ISBN</td>
      <td><input type="text" name="ISBN" value size="20"></td>
    </tr>
    <tr>
      <td<div align="left"><p><font color="red">*</font>
          Description</td>
      <td><input type="text" name="DESCRIPTION" value size="200"></td>
    </tr>
    <tr>
      <td><div align="left"><p><font color="red">*</font>Category</td>
      <td><input type="text" name="CATEGORY" value size="12"></td>
    </tr>
    <tr>
      <td><div align="left"><p><font color="red">*</font>Cost $</td>
      <td><input type="text" name="COST" value size="6"></td>
    </tr>
  </table>
  <p><br>
```

```
    <input type="button" value="Submit" onClick=
         "checkRequiredFields();">
    <input type="reset" value="Reset">
  </p>
 </form>
</body>
</html>
```

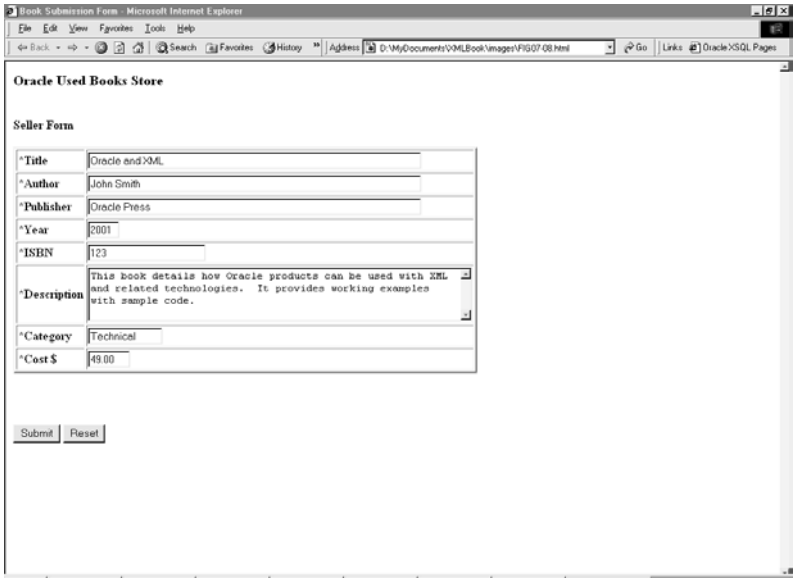

Abbildung 8-8: Beispiel des Bucheintragsformulars

Das Folgende ist ein Beispiel für ein XML-Dokument, das vom JSP erstellt und an die XML SQL Utility zum Einfügen übergeben wird:

```
<?xml version="1.0"?>
<Booklist>
  <BookID>001234</BookID>
  <Title>The Difference Between God and Larry Ellison: Inside Oracle
         Corporation</Title>
  <Author>Mike Wilson</Author>
  <Publisher>William Morrow & Co.</Publisher>
  <Year>1997</Year>
  <ISBN>0688144251</ISBN>
  <Description>Account of Larry Ellison</Description>
  <Category>Computer</Category>
  <Cost>30.00</Cost>
  <Status>A</Status>
</Booklist>
```

Um die Käufer-Website zu entwerfen, können wir eine XSQL-Seite verwenden, die anfangs eine Abfrage auf der Basis der verfügbaren Kategorien darstellt, wie in Abbildung 8-9 gezeigt wird. Jeder Link könnte wie folgt eine bestimmte Seite aufrufen, oder die Parameter-Funktion des Servlets kann die Kategorie an eine allgemeine Seite übergeben.

```
<?xml version="1.0"?>
<xsql-stylesheet type="text/xsl" href="categorydetail.xsl"?>
<xsql:query xmlns:xsql="urn:oracle-xsql" connection="demo"
    SELECT Title, Author, Description, Cost FROM Booklist
      WHERE Category = FICTION
</xsql:query>
```

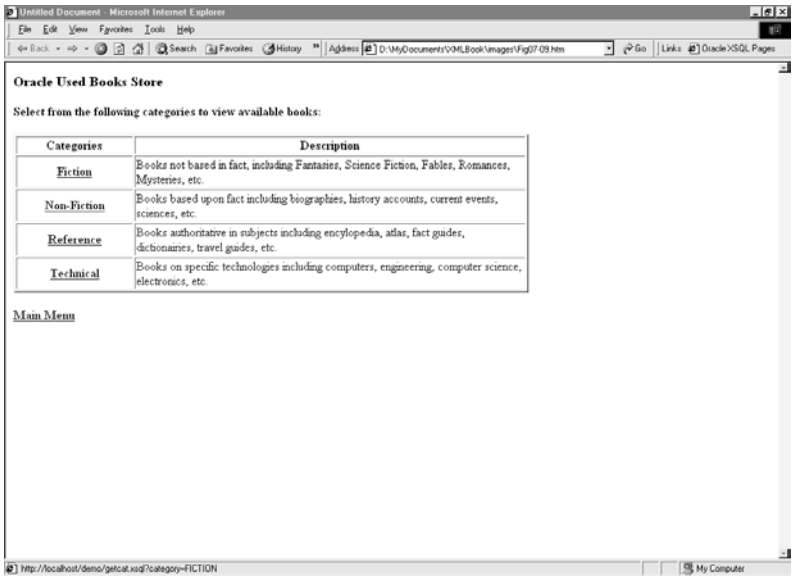

Abbildung 8-9: Auswahlseite des Buchladens

Das Stylesheet categorydetail.xsl kann dann die Anzeige der Listings formatieren und die Formularaktion hinzufügen, um den Verkauf zuzulassen. Durch die Nutzung der Aktualisierungseinrichtung kann die Status-Spalte von *A* (verfügbar) auf *S* (verkauft) geändert werden, und die Trigger-Prozedur kann unter Verwendung des **utl smtp**-Pakets erweitert werden, damit eine Email-Nachricht über den Verkauf versendet wird.

Die restliche Aufgabe besteht darin, ein Formular aufzusetzen, um Konten im System einzurichten und die Neueinträge und Verkäufe zu verfolgen. Sie können dies mit der XML SQL Utility, dem Class Generator und der JSP-Architektur zum Eintragen von Büchern tun. Die Implementierung bleibt dem Leser überlassen, aber die folgende XML-Beispieldatei soll den Einstieg erleichtern:

```
<Client>
  <ClientID>123456</ClientID>
  <Name>Mike Wilson</Name>
  <Address>123 Main St.</Address>
  <City>San Francisco</City>
  <State>CA</State>
  <Country>USA</Country>
  <Zipcode>94000</Zipcode>
  <Email>mwilson@anyhere.com</Email>
  <Date>03-MAY-2002</Date>
  <Time>22:00</Time>
  <Status>A</Status>
  <Account>
     <Buy>
        <BookID>103454</BookID>
        <Date>04-JUN-2003</Date>
        <Time>11:05</Time>
     </Buy>
     <Sell>
        <BookID>001234</BookID>
        <Date>03-MAY-2002</Date>
        <Time>22:00</Time>
     </Sell>
  </Account>
</Client>
```

8.4.5 Erweiterung des Beispiels

Das Buchladen-Beispiel ist einfach und kann leicht in die verschiedensten Richtungen erweitert werden. Auf der Client-Seite können Sie eine Suchfunktion hinzufügen, die die bereits erwähnte „Fuzzy Match"-Funktionalität verwendet. Java oder PL/SQL Stored Procedures können einen Authentifizierungsdienst aufrufen, um Kreditinformationen zu prüfen und eine Kartenfreigabenummer zu bestätigen. Werden XML-Buchlisten von anderen Anbietern akzeptiert, kann die Site auch einfach als Übersicht benutzt werden, wobei den Verkäufern die eigentliche Auslieferung überlassen bleibt.

Während dieses Beispiel eine Consumer-to-Business-Beziehung aufzeigt, kann die gleiche Technologie und Architektur auch für eine Business-to-Business-Site genutzt werden. Großhändler und Hersteller können ihre Waren zum Verkauf anbieten, oder umgekehrt können Unternehmen eine Bestellung aufgeben, die Meldungen auslöst, die an die Anbieter zur Ausführung weitergesendet werden.

Im Falle eine Nachrichtenübertragung von Anwendung-zu-Anwendung können die Klassen der XML SQL Utility direkt aufgerufen werden, um Daten zu lesen und zu schreiben. Durch die Nutzung von Oracle8*i*'s *Advanced Queuing* (*AQ*) und Oracle Workflow, kann die Weiterleitungslogik angewendet werden, um die Nachrichten ge-

mäß ihrer Priorität in der richtigen Reihenfolge zu versenden. Der folgende Abschnitt über den Oracle Integration Server enthält hierzu weitere Details.

8.4.6 Oracle Integration Server

Der *Oracle Integration Server* (*OIS*) ist eine Produkt-Suite, die dafür konzipiert wurde, eine generische, XML-basierte Messaging-Infrastruktur bereitzustellen, die eine einfache Implementierung der Anwendungsintegration über Intranets oder das Internet erlaubt. Abbildung 8-10 zeigt die Architektur des OIS. Beachten Sie, dass diese auf einer Hub-and-Spoke-Architektur basiert, die die Administration und Verwaltung durch die geringere Anzahl der Weiterleitungsprozesse vereinfacht. Adapter werden erstellt oder bereitgestellt, um die XML-Nachrichten-Payload zu erstellen, die vom *Oracle Message Broker* (*OMB*) akzeptiert wird. OMB packt dann die Nachricht in einen Java Messaging Service-Envelope und leitet sie an den Hub weiter. Diese Übertragungsart hat den Vorteil, dass sie für Firewalls transparent ist und auch die Kontrolle der Service-Qualität und Ressourcen-Nutzung erlaubt.

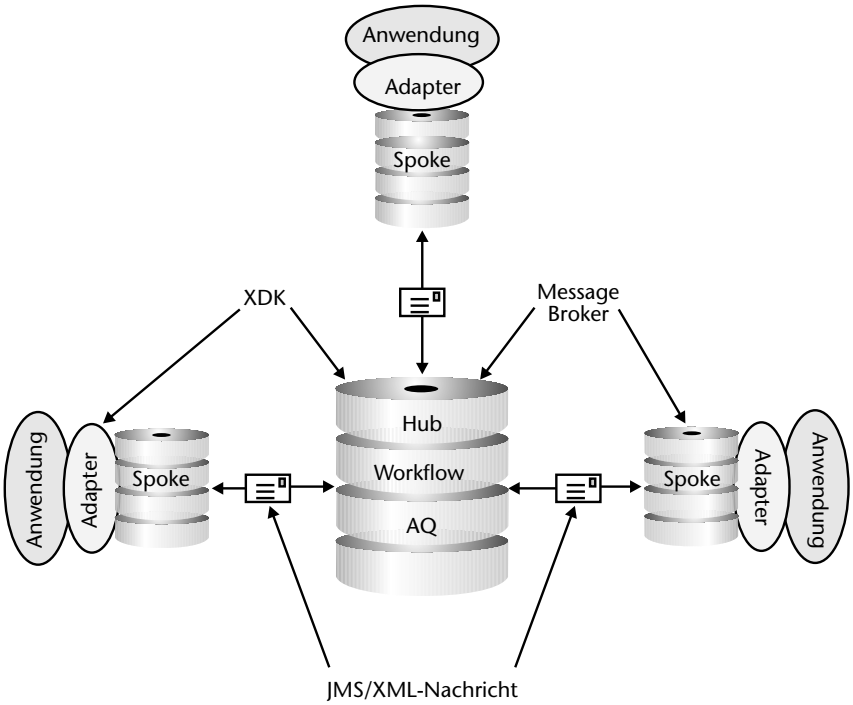

Abbildung 8-10: Architektur des Oracle Integration Servers

Erreicht die Nachricht den Hub, wie in Abbildung 8-11 dargestellt, wird sie in eine Warteschlange gestellt und an die Regel-Engine im Oracle Workflow zur korrekten Weiterleitung gesendet. XSL-Stylesheets dienen der Umwandlung der Nachricht in das gewünschte Zielformat, und sie wird erneut in die Schlange gestellt und über OMB an eine oder mehrere empfangende Anwendungen geschickt.

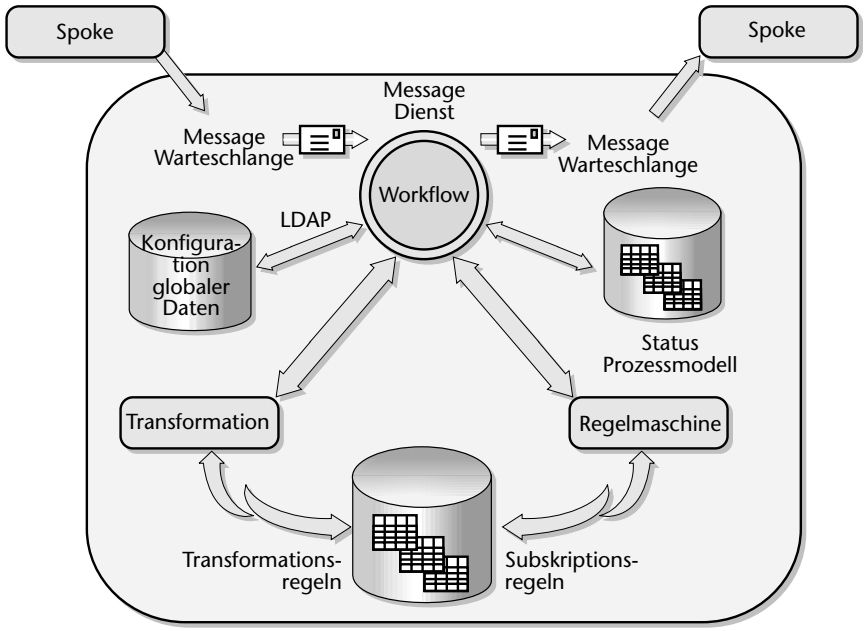

Abbildung 8-11: Die Hub-Architektur des Oracle Intergration Servers

Diese Art der Nachrichtenübertragung, die durch die offenen XML-Standards möglich wird, erlaubt die lose Koppelung und asynchrone Kommunikation von Anwendungen, weil die Integrationslogik sich im Hub befindet, anstatt fest in den Anwendungen kodiert zu sein.

Wie aus diesen Beispielen ersichtlich ist, sind XML und Datenbanken auf vielen Ebenen der Anwendungsintegration eng verbunden. Das fehlende Stück, durch das dies alles nahtlos verwirklicht werden kann, ist der XML Schema-Standard. So haben bereits herstellerunabhängige Gremien wie XML.org und die Open Applications Group begonnen, Repositorys für spezielle Industrie-DTDs und Schemas zu erstellen, mit denen Unternehmen innerhalb derselben Branche leicht Daten austauschen können. Das Hinzufügen von einfachen und komplexen Datentypen in die XML-Technologie wird genaue Datenabbildungen in einem offenen Standard ermöglichen. Dies wird die Übernahme von XML als dem Datenaustauschformat nur noch beschleunigen.