Interoperability issues in accessing databases through Web Services

FLORIN STOICA, LAURA FLORENTINA CACOVEAN
Department of Informatics
“Lucian Blaga” University of Sibiu
Str. Dr. Ion Ratiu 5-7, 550012, Sibiu
ROMANIA
florin.stoica@ulbsibiu.ro, laura.cacovean@ulbsibiu.ro

Abstract: - XML and Web Services provide basic data interoperability today. Web Services, as a distributed application technology, simplifies interoperability between heterogeneous distributed systems. Toolkit interoperability means that we can write Web Service code in one toolkit and easily consume it using another toolkit without dropping down to the lower-level XML APIs. Toolkit interoperability is harder to achieve due to the many differences in programming languages, and because there exists no standard XML Schema definition for representing database resultsets. The .NET DataSet is one type that presents interesting challenges to toolkit interoperability. In this paper we present a method for using a .NET DataSet to consume a WebRowSet object provided by a JAX-WS Web Service, without using XML APIs directly.

Key-Words: - Web Service, DataSet, WebRowSet, interoperability.

1 Introduction

Traditional client/server and Web application-environments relied on connection-based synchronous communication with the back-end database. Although this means tight control over locking and transaction behaviour, it is not always possible to connect to a database server over the Internet, since proxies and firewalls do not often allow it. Even if they did, opening up databases over the Internet would pose a security threat.

As a result the Java community, with WebRowSets, are focusing on XML and improved scalability with XML rowset-type architecture that is fundamentally disconnected and asynchronous in nature. The Rowset library allows to serialize and deserialize ResultSets across multi-layer applications, in between different applications, as well as over the web. This paper focuses on the WebRowSetImpl class in particular, a WebRowSet interface implementation that can serialize the data, metadata, and properties of a JDBC ResultSet to XML.

On .NET platform, when dealing with complex XML Schemas, ADO.NET DataSet offers the greatest support and flexibility. The DataSet, which can be used in either relational databases or XML data files, is an in-memory cache of retrieved data. It is the major component for data handling in the ADO.NET architecture [2]. The DataSet provides a consistent relational programming model regardless of the data source; thus, it consists of a collection of DataTable objects that can be related to each other with DataRelation objects. A DataSet reads and writes data and schema as XML documents. The data and schema can be transported across HTTP and used by any application, on any platform that is XML-enabled.

Web Services technology simplifies interoperability between heterogeneous distributed systems. Web Services can be built using XML APIs directly. This approach gives full control over message processing but decrease productivity and introduces bugs that can lead to more interoperability problems.

A Web Service toolkit reduces the XML API code required of the Web Service developer, by defining mappings between application types and XML Schema types, thereby automating the translation between XML documents and object instances at run time. Web Service toolkits make it easy to ignore XML since the Web Service developer deals only with objects.

In the following, we are using Web Service toolkits integrated in Microsoft Visual Studio .NET 2008 and respectively NetBeans IDE 6.8 to consume/deploy Web Services.

Toolkit interoperability means that we can write Web Service code in one toolkit and easily consume it using another toolkit without dropping down to the lower-level XML APIs. Toolkit interoperability is harder to achieve in case of data-oriented Web Services, because there is no defined a standard XML Schema definition for representing database resultsets. Hereby, a .NET DataSet object cannot map directly a Java XML representation of a WebRowset object to a DataTable object.

In the following sections we present a method for using the DataSet class in a .NET client to consume a WebRowSet object provided by a JAX-WS Web Service, without using XML APIs directly.
2 Related work

Web services provide a standard means of inter-operating between different software applications, running on a variety of platforms and/or frameworks. However, achieving toolkit interoperability as was it defined above is not a trivial task. In [7] are presented different techniques to consume a .NET Web Service from .NET, Java or ASP clients. But all clients are parsing the data provided by Web Service in XML form using style sheets which are accessible to them.

In [2], the author recommends to not use DataSets in Web Service interfaces, or to use typed DataSets, which are bounded to a specific structure, an XML Schema definition, at design time, but this still leaves the client working with the DOM API directly. The ultimate solution, however, to not deal with the raw XML when using DataSets would be to standardize an XML Schema definition for representing database resultsets that could be supported across all toolkits [2]. In the following we present our method for exchanging data between .NET and Java platforms, without using XML APIs directly.

3 WebRowset defined

The javax.sql package includes the ResultSet interface. As its name implies, a rowset is an object that encapsulates a set of rows that have been retrieved from some tabular data source. The javax.sql.ResultSet interface extends the java.sql.ResultSet interface, so in many ways a rowset behaves just like a result set. A RowSet is simply a ResultSet that can function as a JavaBeans component.

The standard implementations of the RowSet interface consist of two parts, the interfaces and the reference implementations. The interfaces are in the javax.sql.rowset package; the implementations are in the com.sun.rowset package. The javax.sql.rowset package includes the following interfaces [1]:

- CachedRowSet - a disconnected rowset that caches its data in memory; not suitable for very large data sets, but an ideal way to provide thin Java clients, such as a Personal Digital Assistant (PDA) or Network Computer (NC), with tabular data;
- A JDBCRowSet class - a connected rowset that serves mainly as a thin wrapper around a ResultSet object to make a JDBC driver look like a JavaBeans component;
- A WebRowSet class – a disconnected rowset used to make it possible for web clients to retrieve and possibly update a set of rows.

![Fig. 1 The rowset interface standard implementation](image)

The WebRowSet interface extends the CachedRowSet interface and therefore has all of the same capabilities. What it adds is the ability to read and write a rowset in XML format. A WebRowSet object uses a WebRowSetXmlReader object to read a rowset in XML format and a WebRowSetXmlWriter object to write a rowset in XML format. The XML version of a WebRowSet object contains its metadata, including its properties, in addition to its data.

A WebRowSet object is designed to work well in a distributed client/server application. A WebRowSet object uses HTTP/XML (Hypertext Transfer Protocol/ eXtensible Markup Language) to communicate with the middle tier, so that, for example, Web clients can talk to Java servlets that provide data access.

XML has become more and more important for Web services because of the portability of data it provides. The JDBC RowSet implementations specification includes a standard WebRowSet XML Schema, available at: http://java.sun.com/xml/ns/jdbc/webrowset.xsd, to which a standard WebRowSet object adheres. This means that if two parties have the XML schema for a WebRowSet object, they can use it to exchange rowsets in a common format even though they may store their data internally in entirely different formats. This makes a WebRowSet object a powerful tool for data exchange using Web services.

The advantages associated with the use of the WebRowSet feature include the following [5]:

- Active connections with the datasource need not be maintained to pass the tabular data between tiers and components.
- The rows (tabular data) referred to as “rowsets” can be read and written in XML format, thus enabling rowsets to be sent over the Internet using the HTTP/XML/SOAP protocol.
4 Loading DataSet from XML
The .NET DataSet represents a subset of the entire database, cached on client machine without a continuous connection to the database. Periodically, a typical application will reconnect the DataSet to its parent database, update the database with changes from the DataSet, and update the DataSet with changes in the database made by other processes. This is highly efficient, but to be effective the DataSet must be a robust subset of the database, capturing not just a few rows from a single table, but also a set of tables with all the metadata necessary to represent the relationships and constraints of the original database. The schema of a DataSet (its tables, columns, relations, and constraints) can be defined programmatically, created by the Fill or FillSchema methods of a DataAdapter, or loaded from an XML document. To load DataSet schema information from an XML document, we can use the ReadXmlSchema method. ReadXmlSchema allows loading the DataSet schema information from the document containing XML Schema definition language (XSD) schema, or an XML document with inline XML Schema. The ReadXmlSchema method takes a single argument of a file name, a stream, or an XmlReader containing the XML document to be loaded. The XML document can contain only schema, or can contain schema inline with XML elements containing data. To fill a DataSet with data from XML, is used the ReadXml method. The ReadXml method will read from a file, a stream, or an XmlReader, and takes as arguments the source of the XML plus an optional XmlReadMode argument.

5 XML gateway architecture for JDBC-ADO.NET interoperability
Database Web services sustain the corporate existing server-side infrastructure, allowing using a database as data service provider, thus enabling to share data and metadata across network. One of the principal reasons for implementing Web services is to achieve interoperability. Clients can access Web services regardless of the platform or operating system upon which the service or the client is implemented. In addition to interoperability, Web service clients can use standardized approaches to access services through firewalls. Such access extends the capabilities of clients. The transport protocol used by Web Services enables clients to operate with systems through firewalls. In figure 2 is showed the XML gateway architecture for JDBC-ADO.NET interoperability that allows extracting data from a JDBC data source, serializing the JDBC resultset to XML and sending the XML version of extracted resultset across network, using SOAP messages, to .NET client applications. A .NET client application can deserialize the data, metadata, and properties of a JDBC resultset from XML, using a DataSet object. Having created and populated the DataSet, the next step is to bind the data to the Visual Studio .NET controls for user interaction.

6 Web Service implementation
Firewalls do not often allow connecting to a database server over the Internet. Opening up databases over the Internet would pose a security threat. To avoid this lapse in security, we implement a Web service designed to act as a proxy for the database, using Java API for XML Web Services (JAX-WS). A follow-up to the release of Java API for XML-based RPC (JAX-RPC), JAX-WS simplifies the task of developing web services using Java technology. It addresses some of the issues in JAX-RPC 1.1 by providing support for multiple protocols such as SOAP 1.1, SOAP 1.2, XML, and by providing a facility for supporting additional protocols along with HTTP. JAX-WS uses JAXB 2.0 for data binding and supports customizations to control generated service endpoint interfaces.

```java
@WebMethod(operationName = "getRowSet")
public String getRowSet(@WebParam(name = "selectCommand")
        String selectCommand) {
    WebRowSetImpl rowset = null;
    ResultSet resultSet = null;
    Connection con = null;
    ...
In our example we are using the Chart controls for graphical reporting based on source data:

```java
class WebServiceClient {
    private DataSet dataSet = new DataSet();
    DataTable dataTable = new DataTable();
    private string selectCommand;

    public WebServiceClient (string selectCommand) {
        this.selectCommand = selectCommand;
    }

    public DataTable GetData() {
        string xmlRowSet;
        server1.JWebRowSetService javaService = new server1.JWebRowSetService();
        xmlRowSet = javaService.getRowSet(selectCommand);
        StringReader readerSchema = new StringReader(xmlRowSet);
        dataSet = new DataSet();
        dataSet.ReadXmlSchema(readerSchema);
        StringReader readerData = new StringReader(xmlRowSet);
        dataSet.ReadXml(readerData);
        CreateDataTable();
        FillDataTable();
        return dataTable;
    }

    private void CreateDataTable() {
        DataColumn dataColumn;
        DataRowCollection columnDefinitions = dataSet.Tables["column-definition"].Rows;
        foreach (DataRow col in columnDefinitions) {
            dataColumn = new DataColumn();
            dataColumn.DataType = getSystemType(col[16].ToString());
            dataColumn.ColumnName = col[9].ToString();
            dataTable.Columns.Add(dataColumn);
        }
    }

    private Type getSystemType(String type) {
        switch (type) {
            case "VARCHAR":
                return Type.GetType("System.String");
            case "INTEGER":
                return Type.GetType("System.Int32");
            case "DOUBLE":
                return Type.GetType("System.Double");
            case "DATETIME":
                return Type.GetType("System.DateTime");
        }
        return Type.GetType("System.String");
    }

    private void FillDataTable() {
        int numCols = dataTable.Columns.Count;
        // Further data binding to Windows controls...
    }
}
```

Supposing that the name of Web service is JWebRowSet, the WSDL file is provided at address: http://localhost:8080/JWebRowSet/JWebRowSetService?WSDL. After deployment of the Web service, the client application can invoke the web service to retrieve tabular data.

### 7 .NET client implementation

The client application was implemented in C#. The core of our solution is represented by WebServiceClient class, which invokes the Java Web Service and handles received XML data using a DataSet object. Originally Java WebRowSet object is mapped to a .NET DataTable object. The DataTable is the primary building block of ADO.NET. The DataTable is made up of DataColumn and DataRow collections. The columns define the schema of the DataTable and the rows make up the actual data in the DataTable. With CreateDataReader() method of the DataTable class, is obtained a DataTableReader object, which can be used further in data binding to Windows controls. Thus, the data provided by Web service may be used as data source for reports in both graphical and tabular forms. The Crystal Reports Engine is delivered with the Visual Studio .NET, and has powerful dynamic reporting capabilities. The Chart controls enable .NET developers to create ASP.NET pages or Windows Forms applications with simple, intuitive, and visually compelling charts for complex statistical or financial analysis [6].
int k = 0;
DataRow newRow = dataTable.NewRow();
DataRowCollection rowData =
dataset.Tables["columnValue"].Rows;
foreach (DataRow row in rowData) {
if (dataTable.Columns[k].DataType.Name.
Equals("DateTime"))
newRow[k++] =
JMiliSecondToDateTime((row[0].ToString()));
else
newRow[k++] = row[0];
if (k % numCols == 0) {
dataTable.Rows.Add(newRow);
k = 0;
}
}

The method getRowSet() of the Java Web service
returns data serialized into XML which is stored in a
StreamReader object. The schema information from the
XML document is loaded using ReadXmlSchema
method of the DataSet object. Then, the DataSet is filled
with data from XML, using ReadXml method. In the
first step is created the structure of a DataTable object
with data definitions of originally Java WebRowSet
object, stored now in the .NET DataSet object. The
getSystemType() method is used to map JDBC types to
ADO.NET types for fields definitions. The DataTable
object is filled with data by FillDataTable() method.
Again, a conversion is necessary between java.util.Date
(Java) and System.DateTime (.NET). This conversion is
accomplished by JMiliSecondToDateTime() method.
In the C# GraphicForm, the DataTable received through
method GetData() of WebServiceClient class is used as
data source for a Chart control (graphical reporting):

public partial class GraphicForm : Form {
private DataTable dataTable;
private WebServiceClient webServiceClient;

private string selectCommand =
"select article, date, sales as quantity," +
"sales=price as [value], stock from sales";

public void BuildGraphic(string what) {
DataTableReader dataReader =
dataTable.CreateDataReader();
chart.Series.Clear();

if (what.Equals("Quantities")) {
chart.DataBindCrossTable(dataReader,
"article","date","quantity",
"Label=quantity");
} else{
chart.DataBindCrossTable(dataReader,
"article","date","value",
"Label=value(C)");
} // chart configuration [...]
chart.Invalidate();
dataReader.Close();
}

public DateTime JMiliSecondToDateTime(
string javaMS) {
DateTime dateUTC = new DateTime(
1970, 1, 1, 0, 0, 0, DateTimeKind.Utc);
DateTime date = dateUTC.Add(
new TimeSpan(long.Parse(javaMS) *
TimeSpan.TicksPerMillisecond)).
ToLocalTime();
return date;
}

Fig. 3 A Chart control with retrieved WebRowSet as
data source
8 Conclusion
The ability to process data regardless of where it comes from is the most primitive level of interoperability. Web Service toolkits make it easy to ignore XML since the Web Service developer deals only with objects. Problems occur when is used a different toolkit on the other end of the wire.
Without an industry accepted standard XML Schema definition for representing database resultsets that could be supported across all toolkits, can be harder for developers who use different toolkits to consume certain database – oriented Web Services
In this paper was presented a method which enables a .NET DataSet to consume the XML data provided by a JAX-WS Web Service. The WebRowSet object exposed by the Java Web service is mapped to a DataTable object on .NET client, without using XML APIs directly. The DataTable object can be used further in data binding to Windows controls.

References: