Java in the Oracle Database @ Work – Customer Case Studies

An Oracle Sponsored White Paper
Updated November 2007

ORACLE

Java in the Oracle Database @ Work – Customers Case Studies

FOREWORD AND ACKNOWLEDGMENTS

Almost a year ago, during my presentation at Oracle World San Francisco 2002, I asked the audience to share concrete, real life examples of running Java in the database. I am pleased to introduce this first issue of Customer Case Studies. This White Paper is a must read for all IT managers, DBAs, applications developers, and anyone who has ever questioned the usefulness of Java in the Oracle Database. You will learn how the combination of Java with the SQL engine turns the Oracle Database into an expansible data logic server, beyond a traditional relational database.

The first case study describes Harris Corporation's use of Java stored procedures for RMI call-out in their Weather And Radar Processor system.

The second case study describes Appropriates Solution's use of Java in the Oracle Database for secure credit card processing using JSSE call-outs.

The third case study describes TECSIS's use of Java in the Oracle Database for implementing a custom Enterprise Integration Framework.

I know of more examples that could not be included in this paper because of lengthy authorization processes or corporate policies prohibiting the furnishing of case studies. If you would like to share your experience of using Java in the Oracle Database, please contact me at kuassi.mensah@oracle.com.

I am grateful to Lesley, John, Francisco, Raymond, and Esteban for sharing their experiences with the Oracle community.

I would like to express my gratitude to Sahar Youssef in Oracle Support for testing and expanding Francisco's paper, on her own time. Sahar has updated the JSSE use case as a Metalink Note # 256928.1 which is copy/pasted here.

Finally, special thanks to Ellen Siegal, our document editor, for turning any draft into a well-written, publishable paper.

Kuassi Mensah

Group Product Manager
Java Products Group
Oracle Corporation

Weather And Radar Processor (WARP) Using Java Stored Procedures

A Harris Corporation Case Study
August 2003

Authors: Lesley A. Brown, John C. Greene
Contributors: Ellen Siegal, Kuassi Mensah

"Oracle9i and its internal JVM have been instrumental in providing a framework that supports the requirements of the WARP and WINS systems."

Lesley A. Brown

Harris -Weather And Radar Processor (WARP) Using Java Stored Procedures

About the Company 5
About the Project 5
Architecture Overview 5
Implementation 5
Why OracleJVM? 6
Trigger-Based Notification 6
Outcome 9
About the Authors 10

Harris -Weather And Radar Processor (WARP) Using Java Stored Procedures

ABOUT THE COMPANY

Harris Corporation is an international communications equipment company focused on providing product, system, and service solutions for commercial and government customers. The five operating divisions of the company serve markets for broadcast, microwave, government, tactical radio, and network support systems.

ABOUT THE PROJECT

The Weather And Radar Processor (WARP) system, a product of Harris GCSD, furnishes computer-based weather information processing and display at the Air Route Traffic Control Centers (ARTCCs) and the Air Traffic Control System Command Center (ATCSCC). The primary purpose of WARP is to improve the speed and quality of weather information supplied to the Air Traffic Control (ATC) system, for enhancement of air safety and traffic efficiency.

ARCHITECTURE OVERVIEW

The WARP system takes in data from many sources, including the Harris Weather Data Service (HWDS) and direct connections to WSR88D. The Weather Information Network Server (WINS) is an interface between the WARP system and external systems. WINS is a subserver system that disseminates weather data residing on WARP to other National Airspace Systems (NAS) operating within the ARTCCs. Figure 1 illustrates the WARP/WINS configuration and subset of data sources.

IMPLEMENTATION

An external NAS user may request data from the WARP system through WINS by scheduling delivery at specified times, or upon the data arrival to WARP, or a mixture of both. The purpose of this paper is to explain the use of Java in the database for data delivery to an external NAS user, upon data arrival to WARP.

Weather data is stored on the WARP system by saving the data to disk and storing location pointers to this data in the Oracle9*i* Database. This method offers an easy implementation to sort, locate, and maintain data, as well as providing a clean and controlled way to retain and access this data through the WINS subsystem.

WHY ORACLEJVM?

After the initial WINS system was deployed, we realized that we required WINS to deliver data upon arrival. Because the first WINS system was designed and implemented using Java, we decided to use Java for additional capabilities. We designed the WINS generic function based on the assumption that the overall WARP system would be upgraded to Java. To determine the earliest availability of data on WARP, we could have designed either the WINS system to "poll" the WARP data files, or the WARP system to alert WINS. Because the WARP system already employed an Oracle Database, we chose to notify WINS through a database trigger. In order to employ Java in the Oracle Database, we upgraded from Oracle 8.0.5 to Oracle9*i*, Release 2, which embeds a J2EE 1.3 Java virtual machine (JVM). The Java stored procedure would use Remote Method Invocation (RMI) across the network to indicate to WINS that data is available. The use of RMI protocol integrated well with our architecture because each independent thread on the WINS system was designed to use RMI as the primary means of process communication.

TRIGGER-BASED NOTIFICATION

All requests are inserted into a request table. Upon arrival, data is inserted into a data table. A request trigger associated with the request table is always enabled and is responsible for enabling and disabling the data trigger associated with the data table. Here is a typical scenario for the use of the WINS subsystem: When an external NAS user requests to have data delivered upon arrival, the WINS system registers a "request" for that type of data by inserting a record in the WARP database through JDBC into a dedicated WINS request table for the specified data type. The request table contains the entry identifier and information pertaining to the specific data request. Upon data arrival, the data trigger is fired. If the data that has arrived matches an existing request, then it calls a Java stored procedure, which then uses RMI to send a "message" indicating to the WINS system that the requested data has arrived. The message contains the request identifier (stored in the request table) and the data location (the file path on the WARP server). The remote call to the WINS system stores the "message" in a local queue and then returns. A separate thread on the WINS system then retrieves the product from WARP and disseminates it to the users that requested it. Figure 2 portrays a basic thread of data delivery to an external user upon data arrival to the WARP system.

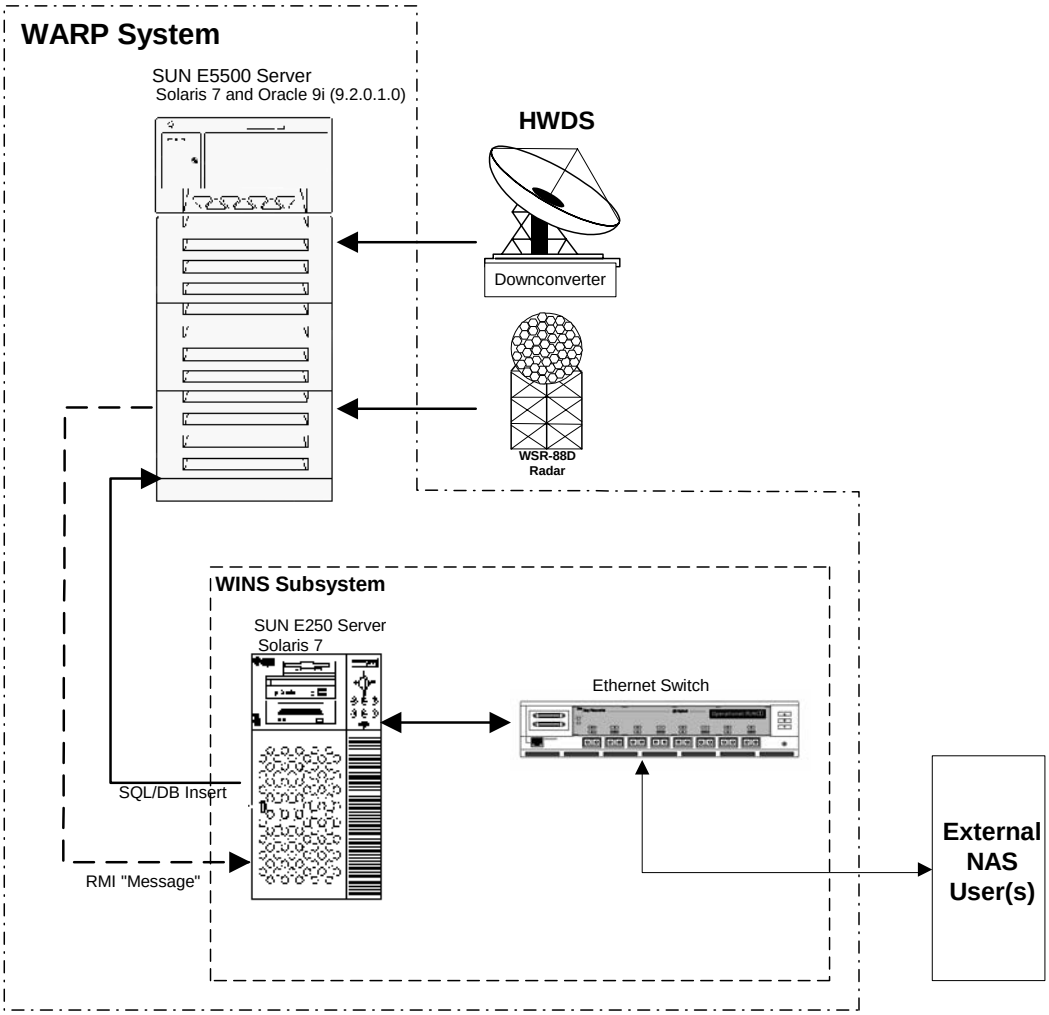


Figure 1: WARP and WINS Systems

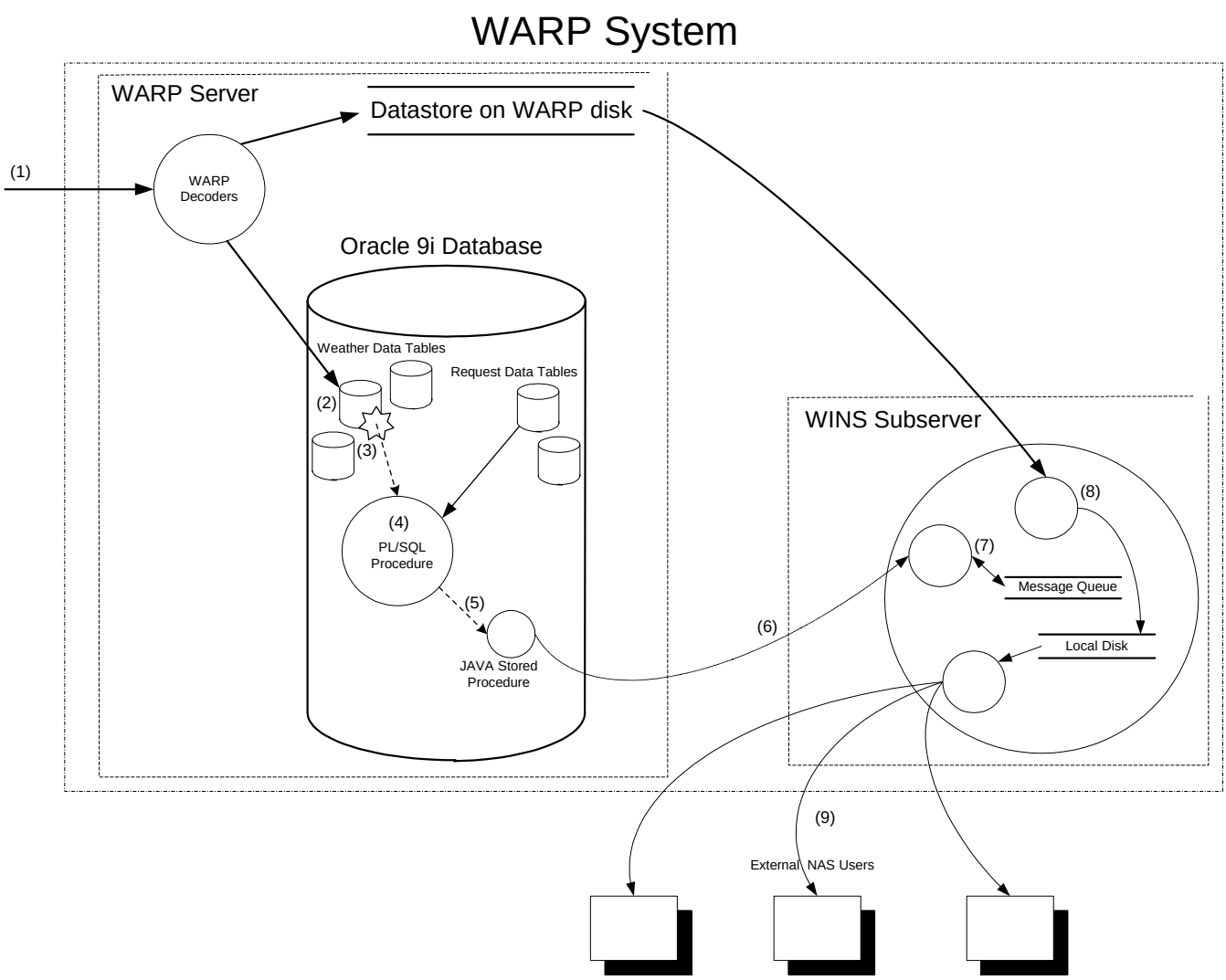


Figure 2: WARP/WINS Communication Layout for a Data Arrival Event

1. WARP receives weather data from external sources and processes it.
2. WARP saves a location pointer for the weather data in the WARP Oracle9*i* Database.
3. If a trigger has been enabled for this type of data, then it fires.
4. A comparison is made to the request tables to check whether this particular piece of data has been requested.
5. If an external NAS user has requested the newly arrived data, then a Java stored procedure is called.
6. The Java stored procedure sends a "message" to the WINS subserver.
7. The thread running the remote method on WINS saves the "message" from the WARP database in its message queue and returns control back to the Java stored procedure in the WARP database.
8. A separate thread of processing copies the weather data from the WARP Server disks to the WINS local disk.
9. WINS then delivers the data to the NAS users who have requested it.

OUTCOME

This Java-based implementation has proven to be reliable. The additional load of the triggers and Java stored procedures has not affected the WARP system performance. The approach of data delivery to NAS users upon arrival, in contrast to scheduling the delivery of data products, has reduced the "spiking" of CPU time to process bulk requests (which account for more than 95 percent of the data products requested). Although the average CPU usage remains consistent between the two delivery implementations, employing the stored procedures with the data triggers results in less dramatic CPU use "spiking." See Figure 3 for a comparison of the CPU usage on WINS. Memory, network, and disk usage have all remained at a constant level when compared to previous results. Using Java stored procedures with RMI meets the requirements of having a separate WINS subsystem. Oracle9*i* and its internal JVM have been instrumental in providing a framework that supports the requirements of the WARP and WINS systems.

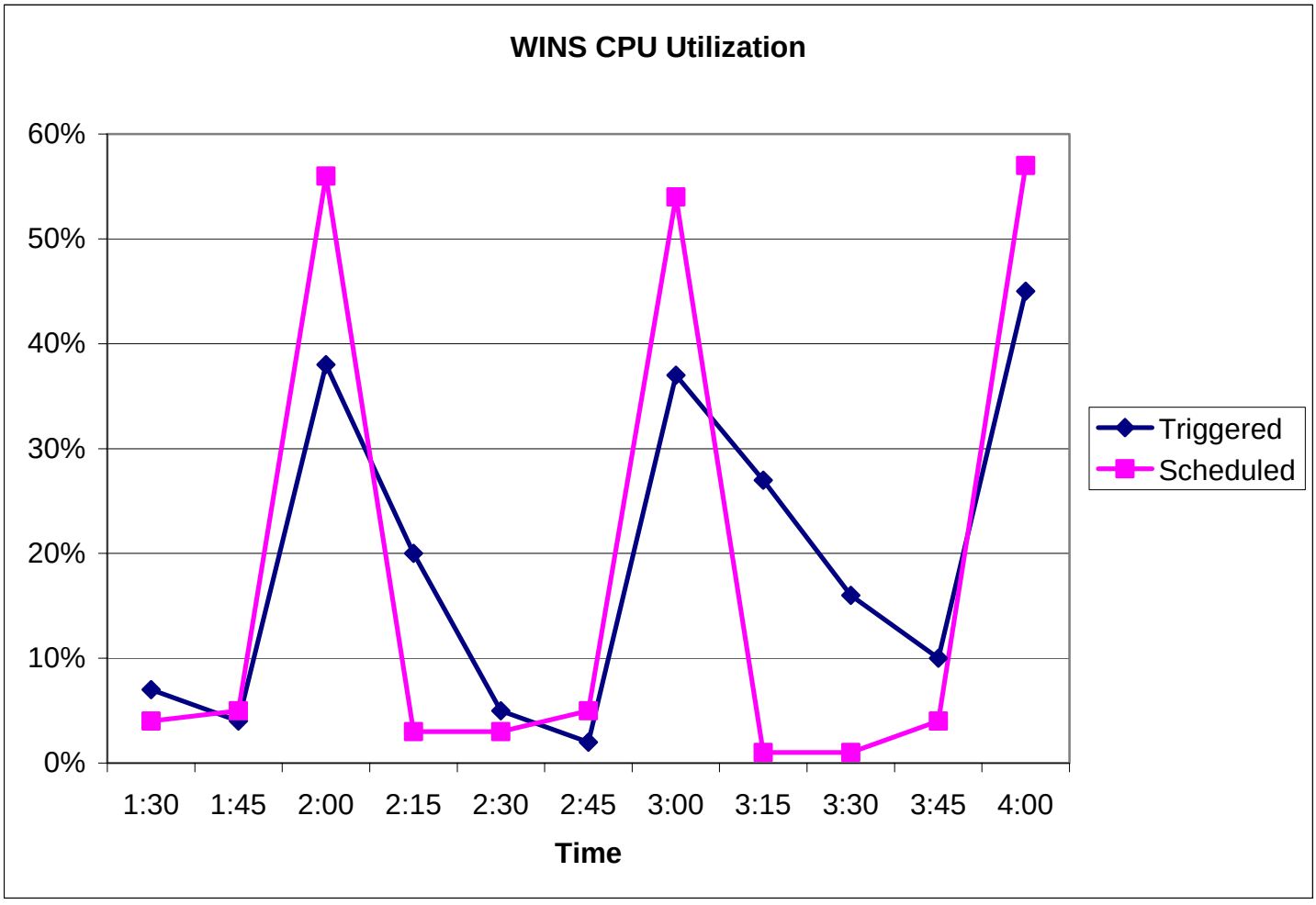


Figure 3: WINS CPU Utilization Comparison

ABOUT THE AUTHORS

.

Lesley A. Brown, Staff Meteorologist / Software Engineer

Lesley Brown is a Staff Engineer for the Harris Corporation Government Communications Systems Division (GCSD) in Melbourne, Florida. In his latest role as a WARP software engineer, Mr. Brown was responsible for the initial research and high-level database design used in the Weather Information Network Server (WINS) Generic modifications to the WARP System.

Mr. Brown holds a B.S. degree in Atmospheric Science from the University of Missouri-Columbia.

John C. Greene, Software Engineer

John Greene is a Staff Engineer for GCSD. Mr. Greene implemented the detailed design used in the WINS Generic Interface.

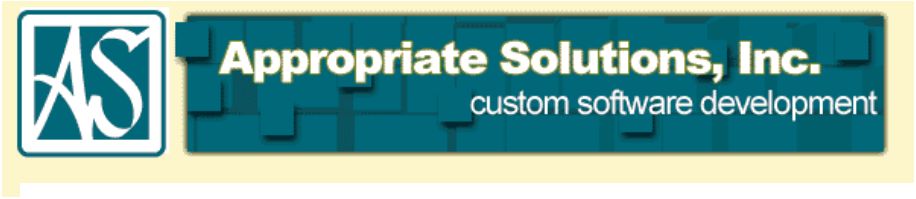
Mr. Greene holds a B.S. degree in Computer Science from the University of West Florida.

CNXO™: Secure Credit Card Processing Using JSSE in the Oracle Database

An Appropriate Solutions, Inc. Case Study
August 2003

Authors: Francisco Juarez and Raymond Côté

Contributors: Sahar Youssef, Ellen Siegal, Kuassi Mensah



85 Grove Street
PO Box 458
Peterborough, NH 03458

603.924.6079
http://www.AppropriateSolutions.com

"Oracle's built-in Java platform allows us to deliver a single, integrated solution based on industry-standard tools. This directly reduces our development, deployment, and support costs."

Fransico Juarez

CNXO™: Secure Credit Card Processing Using JSSE in the Oracle Database

About the Company 13
About the Project 13
Architecture Overview and Design 13
Implementation Challenges 14
CNXO Security Model 16
  High-Level Administration Module (CNXO Registry) 16
  Schema Administration Module (Schema Admin) 16
  Security Module (CNXO Gateway) 16
Instructions for JSSE Implementation Inside Oracle8*i* 16
Instructions for JSSE Implementation Inside Oracle9*i* Rel1 and Rel 2 .. 20
Conclusion 24
About the Authors 24
  Francisco Juarez 24
  Raymond Cote 24
References 25

CNXO™: Secure Credit Card Processing Using JSSE in the Oracle Database

ABOUT THE COMPANY

Appropriate Solutions, Inc. is a custom software development house. The Auric Systems International (ASI) division specializes in both custom and off-the-shelf credit card solutions, with emphasis on the card not present (mail order, phone order, e-commerce) markets. ASI has been designing and building its CreditNow!® and CN!Express® credit card and check processing solutions since 1994.

ABOUT THE PROJECT

The CNXO (CN!*Express* for Oracle) product adds credit card processing abilities to a basic Oracle installation (Workgroup or Enterprise). The processing is performed through an Internet gateway (such as Authorize Net or the Paymentech Orbital Gateway). This capability uses the technology to perform HTTPS (secure HTTP) Web transactions from within Oracle.

CNXO is currently being integrated with the Hospitality Suite and CareTrakker product lines from Computrition, Inc. With CNXO, Computrition can now provide an integrated credit card processing solution for its existing food service management products.

ARCHITECTURE OVERVIEW AND DESIGN

As Figure 1 illustrates, the architecture comprises the end user, the PL/SQL interface, the Java interface, and the processor. All end-user interfaces are written in PL/SQL to allow backward compatibility with existing applications, interfaces, and schemas; all interaction with the processors is written in JAVA, using the Java Secure Socket Extension (JSSE) libraries by Sun Microsystems (the JSSE JAR files are loaded to the Oracle JVM).

First, the end user calls a PL/SQL procedure with XML or Delimited Text input, describing the transaction. Next, the PL/SQL interface parses the incoming data and makes any necessary transformation. Then it checks permissions, invokes the appropriate packages, and sends validated data to a Java stored procedure. Finally, it gets results back from the Java stored procedure, formats the data according to end-user specifications, performs logging, and delivers the report.

The Java stored procedure retrieves the processor, its URL, and the type of the transaction from the incoming data. It then opens a secure HTTP connection (HTTPS) to the processor, which handles the transaction. Finally, it passes the results back to the PL/SQL interface.

Here are the steps a basic credit card transaction call follows:

1. The registered user calls a PL/SQL procedure with XML or Delimited Text data input.
2. The PL/SQL package validates the data, stores some information on the system, and calls a JAVA stored procedure.
3. The JAVA stored procedure checks with the JAVA_POLICY_TABLE to ensure that the target URL is registered.
4. The JAVA stored procedure opens a secure HTTPS connection using JSSE. This connection can also be made through a proxy if desired.
5. The JAVA stored procedure negotiates the transaction with the processor and sends the results back to the calling PL/SQL package.
6. The PL/SQL package logs the results of the transaction and formats the data to the same format that the procedure received (XML or Delimited Text).
7. The procedure sends the results back to the schema, application, or Web site that made the call.

The user remains isolated from all the HTTPS, JAVA, and SQL code and does not need to perform any type of programming to set up or run this system.

IMPLEMENTATION CHALLENGES

Implementing CNXO in Java was easy because CNXO utilizes the Java virtual machine (JVM) built into the Oracle8*i* Database and Oracle9*i* Database. The technique of calling Java stored procedures from a PL/SQL wrapper provided the flexibility to integrate Java stored procedures with PL/SQL stored procedures. In addition, it saved us from having to use an external interface to make secure connections to a Web server.

Because security is a key requirement for CNXO, all Web interaction must be performed through HTTPS.

The biggest implementation challenge was providing HTTPS communications from Java within the Oracle Database. Because HTTPS protocol is not supported from the Oracle HTTPClient library, we decided to use the Sun Microsystems JSSE libraries—it is the standard technique to implement HTTPS calls from Java. JSSE is an optional package for J2SE versions 1.2.x and 1.3.x. From J2SE version 1.4.x and forward, it is a mandatory component. Because JSSE is pure Java, it can be loaded to the Oracle JVM using the LOADJAVA utility. After several unsuccessful attempts, we achieved a consistent and reliable installation in Oracle Database versions 8.1.6, 8.1.7, 9.0.1, and 9.2.0.

Because all HTTPS interactions must be performed with registered URLs in the Oracle JAVA_POLICY_TABLE, before any attempt is made to use a URL it must be added to that table. CNXO performs checks before accessing the URL to ensure that it is already registered.

CNXO does not store sensitive data. Instead, when a client provides a credit card number, it is stored in one-way hash MD5 encryption format by an activity log. To find transaction information, the client must supply a credit card number, which is encrypted by CNXO into MD5 hash format. This format is used to look up the transaction information in the activity log.

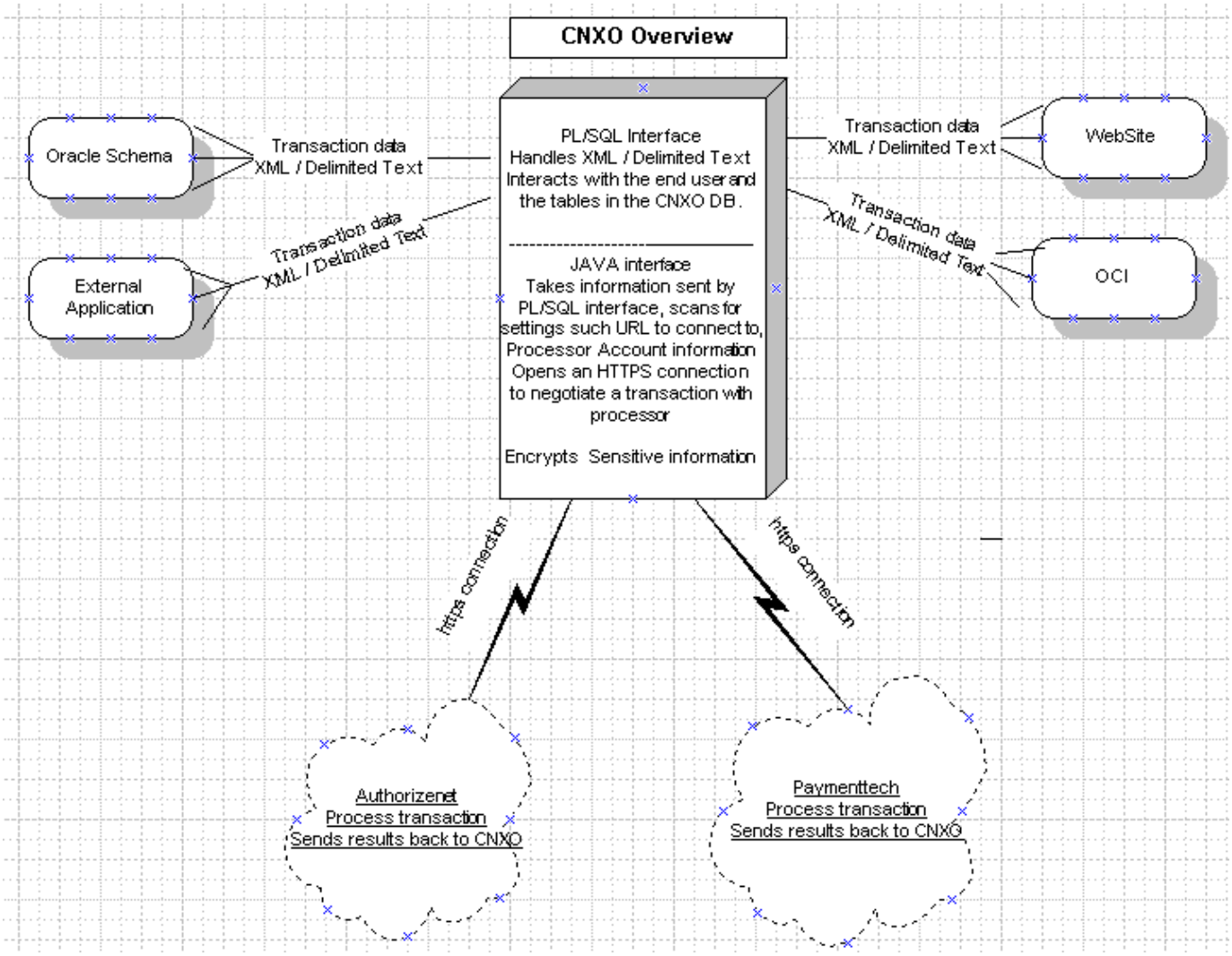


Figure 1: CNXO Overview

CNXO SECURITY MODEL

These were our design goals for the CNXO security model:

1. Use HTTPS protocol to communicate with the card processor's Gateway.
2. Use a reliable technique such as the Sun Microsystems JSSE.
3. Offer our users the ability to consume our framework from PL/SQL modules, Java modules, Web Services, and external applications.

To implement the security model, the CNXO functionality has been divided into three different modules, each allowing different functionality and requiring different security permissions. Following is an explanation of each module.

High-Level Administration Module (CNXO Registry)

This high-level administration security module, called CNXO_SCHEMA_REGISTRY, is responsible for registering the existing schemas, and permits them to use all the CNXO functionality. This interface, protected by an Administrator password, makes this functionality easy to use.

Schema Administration Module (Schema Admin)

This schema administration security module, called CNXO_ADMIN, provides an interface for all registered schemas to set up their processor information, register users, remove users, change users' settings, and get activity reports. We currently support two processors: AuthorizeNet and Paymentech. Any additional processors are easy to implement.

Security Module (CNXO Gateway)

Our core functionality, the CNXO_GATEWAY, offers an easy interface for registered users to send credit card transactions to the processor registered with their schema.

INSTRUCTIONS FOR JSSE IMPLEMENTATION INSIDE ORACLE8*i*

1. Download JSSE version 1.0.3 from the Sun Microsystems Web site `http://java.sun.com/products/jsse/index-103.html` .
Save and unzip the file on your local disk.

Locate the three JSSE .jar files: jnet.jar, jcert.jar, and jsse.jar

for example under the directory `D:\jsse1.0.3_01\lib.`
Load these files to the JVM.

2. You must load the JSSE .jar files to the SYS schema. Do not load them to any other schema; it will not work. If the JSSE .jar files are already loaded to another schema, drop them first using the DROPJAVA utility, as shown below

```
dropjava –v –u myschema/mypassword@myserver jnet.jar
dropjava –v –u myschema/mypassword@myserver jcert.jar
dropjava –v –u myschema/mypassword@myserver jsse.jar
```

3. Load the JSSE .jar files to the SYS schema using the LOADJAVA utility. You must declare them as PUBLIC synonyms when loading. Otherwise, other schemas will not see those libraries.

```
loadjava –v –r –u myschema/mypassword@myserver -s –g
                  PUBLIC jnet.jar
loadjava –v –r –u myschema/mypassword@myserver -s –g
                  PUBLIC jcert.jar
loadjava –v –r –u myschema/mypassword@myserver -s –g
                  PUBLIC jsse.jar
```

4. Place the Certificate Authority file where the JSSE classes can retrieve it. Locate the CACERTS file, which contains the authoritative root certificates needed to validate a certificate obtained from the Gateway site.

In an Oracle8i installation, it is located at $OracleHome\Apache\jdk\jre\lib\security\.

Then copy the CACERTS file to the following location: `$OracleHome\javavm\lib\security.`

5. Set the proper permissions to be able to communicate with external sites. If you are behind a firewall, you will need extra permissions for the proxy machine. You must grant those permissions to every schema that is going to use the JSSE.

Log in to Oracle as SYS/<password> and issue the following commands (the order/sequence is important).

```
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission',
'java.protocol.handler.pkgs', 'write');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission', 'http.proxyHost',
'write');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission', 'http.proxyPort',
'write');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission', 'http.proxySet',
'write');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.net.SocketPermission',  'secure.authorize.net',
'connect, resolve');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.security.SecurityPermission',
'insertProvider.SunJSSE', 'insert');
```

```
SQL> CALL dbms_java.grant_permission(
'MYSCHEMA','SYS:java.util.PropertyPermission',
'file.encoding', 'write' );
SQL> CALL dbms_java.grant_permission(
'MYSCHEMA','SYS:java.util.PropertyPermission',
'https.proxySet','write' );
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission', 'https.proxyHost',
'write');
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.util.PropertyPermission', 'https.proxyPort',
'write');
```

6. Grant access to the resources by adding an entry in the JAVA_POLICY_TABLE, which is an Oracle table owned by SYS.

Two views, DBA_JAVA_POLICY and USER_JAVA_POLICY, provide read-only access to this table. To grant access to a resource, you must create an entry in this table. In the current Java implementation, a remote URL is considered a resource with access rights. The SYS.DBMS_JAVA procedures provide a mechanism for maintaining this table. The table contains many different entries. For example:

- Entries of the type java.io.FilePermission control access to the files. The NAME column specifies the specific file.
- Entries of the type java.net.SocketPermission control access to remote sites. The NAME column specifies the specific server or socket.
- Entries of the type oracle.aurora.rdbms.security.PolicyTable control whether users are allowed to create other sorts of entries in the table.

  The NAME column specifies which entry types the grantee is allowed to create.

See more details Java 2 Security implementation in OracleJVM in chapter 2 of Kuassi's book[1].

In this example, execute the following statement to provide access to the Verisign Web site:

```
SQL> CALL dbms_java.grant_permission('MYSCHEMA',
'java.net.SocketPermission', 'verisign.com', 'connect,
resolve');
```

```
Sample Code
-----------
```

The following code snippet uses HTTPS to connect to Verisign's secure Web site.

It includes several commented lines that are necessary only if you are connecting through a proxy server.

[1] http://www.amazon.com/gp/product/1555583296

1. PL/SQL Wrapper

```
// PL/SQL Wrapper for this Sample
create or replace function SecureURL return varchar2 is
 language java name 'SecureURL.SecureURL() return
java.lang.String';
```

2. Java stored procedure

```
// Java method for accessing a secure http web server.
import java.net.*;
import java.io.*;
import javax.net.ssl.*;
import java.security.*;

public class SecureURL {
      public static void main (String args []) throws
Exception {
      SecureURL r = new SecureURL();
      r.SecureURL();
}

public static String SecureURL () throws IOException {
      System.setProperty("java.protocol.handler.pkgs",
      "com.sun.net.ssl.internal.www.protocol");
Security.addProvider(new
com.sun.net.ssl.internal.ssl.Provider());

//These Properties are used if you are connecting
outside a firewall
//System.setProperty("https.proxyHost", "www-
myproxy.mydomain.com");
//System.setProperty("https.proxyPort", "80");
// System.setProperty("http.proxyUser", "proxyuser");
// System.setProperty("http.proxyPassword", "proxy
password");

URL mysecureURL = new URL("https://www.verisign.com/");
BufferedReader in = new BufferedReader(
      new InputStreamReader(mysecureURL.openStream()));
String inputLine;

//Read some data
inputLine = in.readLine();
in.close();

// inputLine contains the content of the Verisign page.
```

```
return inputLine;
}
}
```

3. PL/SQL procedure to call the wrapper

```
  create or replace procedure runssl IS
   urloutput varchar2(2000);
  begin
   urloutput := SecureURL;
   DBMS_output.put_line ('the output is   ' ||
urloutput);
  end;
```

INSTRUCTIONS FOR JSSE IMPLEMENTATION INSIDE ORACLE9*I RELEASE 1 AND ORACLE9I RELEASE 2*

1. Download JSSE version 1.0.3 from the Sun Microsystems Web site http://java.sun.com/products/jsse/index-103.html.

Save and unzip the file on your local disk. Locate the three JSSE .jar files: jnet.jar, jcert.jar, and jsse.jar for example under the directory jsse1.0.3_04/lib.

2. Copy the certificates to the database folder security folder

```
Copy $ORACLE_HOME/jre/1.4.2/lib/security/cacerts to
$ORACLE_HOME/javavm/lib/security/cacerts
```

Then bounce the database.

3. Copy the JSSE jar files; jnet.jar, jcert.jar and jsse.jar to $ORACLE_HOME/javavm/lib

4. Create the schema owning the jsse objects, as jsse classes must be loaded to a schema different than the SYS Schema. Loading those files to SYS Schema will cause the load process to fail with *ORA-1031 Insufficient privileges*.

Connect as SYS, create the schema and grant the appropriate roles, as follows.

```
SQL> GRANT connect, resource, unlimited tablespace,
create public synonym, drop public synonym, javasyspriv
TO JSSE IDENTIFIED BY JSSE;
```

5.From SQL*PLUS, load the JSSE .jar files to the JSSE schema using DBMS_JAVA.LOADJAVA assuming that your jar file are on $ORACLE_HOME/javavm/lib directory

```
SQL> CONNECT jsse/jsse
SQL> SET serveroutput on
SQL> CALL dbms_java.set_output(100000);
SQL> CALL dbms_java.loadjava(' -r -v -definer -g public
$ORACLE_HOME/javavm/lib/jcert.jar
$ORACLE_HOME/javavm/lib/jnet.jar
$ORACLE_HOME/javavm/lib/jsse.jar');
```

6. Connected as jsse user, create PUBLIC synonym for the HTTPS handler and the SSL Provider.

```
SQL> CREATE PUBLIC SYNONYM
"com/sun/net/ssl/internal/www/protocol/https/Handler"
for
"com/sun/net/ssl/internal/www/protocol/https/Handler";
```

```
SQL> CREATE PUBLIC SYNONYM
"com/sun/net/ssl/internal/ssl/Provider" for
"com/sun/net/ssl/internal/ssl/Provider";
```

Note: If you received the following error

```
ORA-00955: name is already used by an existing object
```

Then you will need to drop the synonyms and recreate them using the drop public synonym statement

```
SQL> DROP PUBLIC SYNONYM
"com/sun/net/ssl/internal/www/protocol/https/Handler";
SQL> DROP PUBLIC SYNONYM
"com/sun/net/ssl/internal/ssl/Provider";
```

Then recreate them as in above in step 6.

7. Set the proper permissions to be able to communicate with external sites. If you are behind a firewall, you will need extra permissions for the proxy machine. You must grant those permissions to every schema that is going to use the JSSE. To do that, you must issue the following commands, in order.

```
SQL> CONNECT sys/<password>

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'insertProvider.SunJSSE', '' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'putProviderProperty.SunJSSE', '' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'getProperty.ssl.ServerSocketFactory.provider', '' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'getProperty.cert.provider.x509v1', '' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.util.PropertyPermission',
'java.protocol.handler.pkgs', 'write' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.util.PropertyPermission', 'https.proxyHost',
'write' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.util.PropertyPermission', 'https.proxyPort',
'write' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'getProperty.ssl.SocketFactory.provider', '');
```

```
SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'getProperty.sun.ssl.keymanager.type', '' );

SQL> CALL dbms_java.grant_permission( 'JSSE',
'SYS:java.security.SecurityPermission',
'getProperty.sun.ssl.trustmanager.type', '' );
```

8. By granting the permission to the schema that is going to use the HTTPS protocol and SSL calls, your SSL environment is set.

```
SQL> CONNECT sys/<password>

SQL> CALL dbms_java.grant_permission( 'SCOTT',
'SYS:java.security.SecurityPermission',
'insertProvider.SunJSSE', '' );

SQL> CALL dbms_java.grant_permission( 'SCOTT',
'SYS:java.util.PropertyPermission',
'java.protocol.handler.pkgs', 'write' );

SQL> CALL dbms_java.grant_permission( 'SCOTT',
'SYS:java.util.PropertyPermission', 'https.proxyHost',
'write' );

SQL> CALL dbms_java.grant_permission( 'SCOTT',
'SYS:java.util.PropertyPermission', 'https.proxyPort',
'write' );

SQL> CALL dbms_java.grant_permission( 'SCOTT',
'SYS:java.security.SecurityPermission',
'setProperty.cert.provider.x509v1', '' );
```

In this example, execute the following statement to provide access to the *login.oracle.com* Web site:

```
SQL> CALL dbms_java.grant_permission('SCOTT',
'java.net.SocketPermission', 'login.oracle.com',
'connect, resolve');
```

9. create the following in the schema of the user who is going to do HTTPS calls

```
SQL> CONNECT scott/tiger

SQL> create or replace java source named "SecureURL" as
import java.net.*;
import java.io.*;
import java.security.*;
```

```
public class SecureURL {
    public static void main(String[] args) throws
      Exception {
      SecureURL r = new SecureURL();
      r.SecureURL();
    }

  public static void SecureURL() throws IOException
  {
      Security.addProvider(new
        com.sun.net.ssl.internal.ssl.Provider());
        System.setProperty("java.protocol.handler.pkgs",

"oracle.aurora.rdbms.url|com.sun.net.ssl.internal.www.pr
otocol");
       System.setProperty("https.proxyHost","www-
myproxy.mydomain.com");
       System.setProperty("https.proxyPort","proxy
port");
        System.setProperty("http.proxyUser",
"proxyuser");
        System.setProperty("http.proxyPassword", "proxy
password");
      URL verisign = new
URL("https://login.oracle.com/");
        BufferedReader in = new BufferedReader( new
InputStreamReader(verisign.openStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null)
          System.out.println(inputLine);
            in.close();
      }
};
```

10. Create the PL/SQL Wrapper

```
// PL/SQL Wrapper Code for this Sample

SQL> CONNECT scott/tiger

SQL> create or replace procedure testssl as language
java name 'SecureURL.SecureURL()';
/
```

11. Call the PL/SQL wrapper

```
SQL> Set Serveroutput ON
SQL> CALL dbms_java.set_output(1000000);
SQL> CALL testssl();
```

CONCLUSION

Because Java itself is standardized on different platforms, it becomes the best choice when dealing with different systems. The implementation uses the Sun JSSE package to provide HTTPS interaction with the credit card processing system. Using Java stored procedures from within the Oracle JVM offers a reliable, robust, scalable and secure credit card processing mechanism. In addition, Oracle's built-in JVM furnishes the flexibility of loading and using well-scoped JAR files. The ability to wrap those Java objects with PL/SQL enables the integration of Java, SQL, XML, and PL/SQL worlds, resulting in greater database flexibility and expandability. For all these reasons, Oracle became the database of choice for implementing our solution. A future step will be to expose the transaction-processing engine through a Web service—using SOAP over HTTPS.

Send questions and comments to:
`fjuarez@AppropriateSolutions.com`

ABOUT THE AUTHORS

Francisco Juarez

Francisco Juarez is a Software Developer for Appropriate Solutions, Inc., based in New Hampshire, USA.

He has worked in a range of languages, including MFC, C++, Java, Delphi, XML, VFP, PL/SQL, and SQL.

In recent years, Francisco has been developing interfaces, front ends, and fat clients to Oracle Databases.

You can contact him at: `fjuarez@AppropriateSolutions.com`

Raymond Cote

Raymond Cote is a founder and president of Appropriate Solutions, Inc., a custom software development house specializing in Oracle and high-volume credit card applications.

He has been programming and writing professionally since 1976 in many different languages, from assembler to PL/SQL.

You can contact him at: `rgacote@AppropriateSolutions.com`

REFERENCES

AuthorizeNet: Gateway to various credit card processors.
`www.AuthorizeNet.com`

Computrition, Inc.: Publishers of fully integrated food services and nutrition management software systems; including Hospitality Suite and Caretakker.
`www.Computrition.com`

Paymentech: Credit card processor accessible over the Internet through Orbital Gateway. `www.Paymentech.net`

UPS implementation of SSL white paper on Oracle 8.1.7 by M. B. Piermarini

`http://osi.oracle.com/~mbpierma/SSL_Java_DB.html`

Appropriate Solutions, Inc.: Custom software developers since 1989.

`www.AppropriateSolutions.com`

Auric Systems, International: Publishers of credit card processing solutions.
`www.AuricSystems.com`

A Custom Enterprise Integration Framework Using Java in the Oracle Database

A TECSIS Case Study
August 2003

Author: Esteban Capoccetti
Contributors: Ellen Siegal, Kuassi Mensah

"I want to share with Oracle users that employing Java in the Oracle Database to its full extent, in our solution, we achieved simplicity, reusability, and cost savings."
Esteban Capoccetti

TECSIS - A Custom Enterprise Integration Framework Using Java in the Oracle Database

About the Company 28
About the Application 28
Our Business and Technical Requirements 28
  Business Requirements 28
  Technical Requirements 29
  Design and Programming Choices 29
Integration Framework Architecture 30
  Typical Use Case Scenarios 30
  Java Stored Procedures Calling External Systems 30
  External Systems Calling Java Stored Procedures 31
Putting This All Together 32
Conclusion 32

TECSIS - A Custom Enterprise Integration Framework Using Java in the Oracle Database

ABOUT THE COMPANY

Tenaris, the world leader in tubular technologies, represents eight established manufacturers of steel tubes: AlgomaTubes, Confab, Dalmine, NKKTubes, Siat, Siderca, Tamsa, and Tavsa.

Tenaris is a leading supplier of tubular goods and services to the global energy and mechanical industries, with a combined production capacity of 3,000,000 tons of seamless and 850,000 tons of welded steel tubes, annual sales of $3 billion, and 13,000 employees in five continents. Our market share is about 30 percent of world trade in OCTG seamless products and 13 percent of total world seamless tube production.

The main goal of Tecsis, the System Technology division of Tenaris, is to validate and disseminate technology throughout the companies within the Tenaris group.

This paper describes our experience of using Java in the Oracle Database and how it solved our integration requirements.

ABOUT THE APPLICATION

For the last three years, our company has been using Oracle Database not just as a database, but also as an integration infrastructure. We started by implementing the business rules, using PL/SQL Stored Procedures, which gave us many advantages. With the Java virtual machine embedded in the database (OracleJVM), we extended the capabilities of our database and turned it into a data integration hub.

OUR BUSINESS AND TECHNICAL REQUIREMENTS

Business Requirements

Our business requirements made it necessary to integrate online information from different platforms, including: SAP, AS400, ADABAS/NATURAL, and COBOL Tandem. Our PL/SQL-based business rules needed to send and get data from these platforms. Existing legacy systems, as well as new intranet/Internet-based development, required cross-platform integration. Our main goal was cost savings through reuse of software, systems, and skills.

Technical Requirements

We needed to integrate the following platforms: Oracle PL/SQL Stored Procedures, SAP R3, Natural/Adabas, RPG/DB400, COBOL Tandem, COM Components, and non-Oracle databases (Adabas-D, MSSQL Server). We tried different RPC technologies to integrate legacy systems, but we were unhappy with their degree of integration. By that time, it became crucial for us to reach information available online on other platforms, from existing PL/SQL packages.

In summary, our most important requirements were to:

- Simplify cross-platform integration.
- Save costs: Instead of adding a new integration layer, we decided to leverage existing components and use each of these components to the best of its capacity.
- Avoid the explosion of communication that would be generated by point-2-point integration.

Design and Programming Choices

We chose to leverage OracleJVM and its ability to run Java libraries in the database, because all existing ERP systems, as well as non-Oracle databases, furnish either a Java-based Remote Procedure Call (RPC) software or a pure Java JDBC driver that can be loaded into the OracleJVM. PL/SQL wrappers make these mechanisms available to the SQL world as Java stored procedures. Existing or new PL/SQL-based business rules can easily interact with other systems. By centralizing our business rules in the Oracle Database, along with transformation rules and making the whole thing accessible by both Web clients and batch jobs, the Oracle Database became our integration engine.

All our new systems are based on Web pages that call stored procedures, which access the business rules. Existing batch jobs, as well as client applications, share the same business rules. We have also been able to standardize the way in which we call the procedures, using XML-based IN and OUT parameters. These XML parameters are parsed or generated using Oracle XDK.

That the system became operational in a few days, without costly retraining of our PL/SQL programmers, is the best illustration of the simplicity of the solution. In addition, we accomplished the integration of batch jobs through three-line SQL* Plus scripts.

Using a traditional Enterprise Application Integration (EAI) product would have been more complex and expensive. Instead, employing Java in the database not only simplified our integration process, it saved us money.

INTEGRATION FRAMEWORK ARCHITECTURE

Typical Use Case Scenarios

We had three use cases. In the first, Code Validation, system A (COBOL Tandem) needs to check whether a specific code value exists in system B (Oracle Database). In the second, Pop-Up Lists, system A (AS400 screen) needs to display a list of values using content from system B (Natural Adabas). In the third case, Cross-Platform Modifications, a new product is added to system A (Oracle Database), and the same product also must be added to system B (Natural Adabas).

Java Stored Procedures Calling External Systems

We selected the Software AG EntireX Communicator (formerly EntireX broker) to make remote calls to Natural/Adabas programs (OS/390), RPG programs (AS400), and Tandem COBOL programs.

Although SAP Java connector (SAP JCO) is distributed as a JAR file, it is not 100% Java based because it uses several `.sl` (libraries). For security reasons, OracleJVM does not allow Java classes to use external `.sl` libraries (that is, JNI calls). We worked around this restriction by running the SAP JCO as an external RMI server. Doing this allows us to issue RMI calls to the SAP JCO from within Java stored procedures.

We loaded third-party pure Java JDBC drivers into the database for interaction between Java stored procedures and non-Oracle databases. If we need to interact with a remote Oracle Database, we can load the Oracle pure Java JDBC driver, also known as thin JDBC, into the database.

Then we created standard PL/SQL wrappers, called EAI_PKG, for each loaded module to allow uniform invocation from the PL/SQL-based business rules.

Finally, we distributed an application integration guide internally to all PL/SQL programmers. In a few days, they were able to build procedures that interact with other platforms.

By centralizing the business rules, and adding integration and transformation rules, we created a complete data integration framework. These business, integration, and transformation rules all interact with external systems through Java stored procedures, using the EAI_PKG. Our new system comprises a Web-based presentation layer, as well as batch jobs (SQL*PLUS scripts). We use Oracle Object for OLE (OO4O) to execute Oracle procedures from our presentation layer (`.asp`). Both the presentation layer and the batch jobs use this same integration framework.

Figure 1 on the following page illustrates this architecture.

Although the concept of the Oracle Advanced Queuing system does not appear in Figure 1, the EAI_PKG furnishes a queue corresponding to every remote system. If a remote system is down or unreachable, the EAI_PKG package automatically enqueues the message in the corresponding queue. An internal DBMS_JOB job is

associated with every queue and is scheduled to run at a determined frequency. This job dequeues the pending messages and attempts to process them until the target system becomes available.

Compared to Java stored procedures, traditional EAI products do not offer the same level of ease-of-use.

External Systems Calling Java Stored Procedures

After we were able to call all our external systems from Java stored procedures, the next step was enabling external systems to call Java stored procedures. To accomplish this second step, we reused both the EntireX Communication and the SAP Java connector. Natural Adabas, AS400, and COBOL Tandem place a call against EntireX, which, in turn, invokes a Java stored procedure. The response from the Java stored procedure is sent back to the legacy system.

Similarly, SAP/ABAP applications call the SAP Java connector, which, in turn, calls the Oracle stored procedure. The response from the stored procedure is returned to the ABAP application.

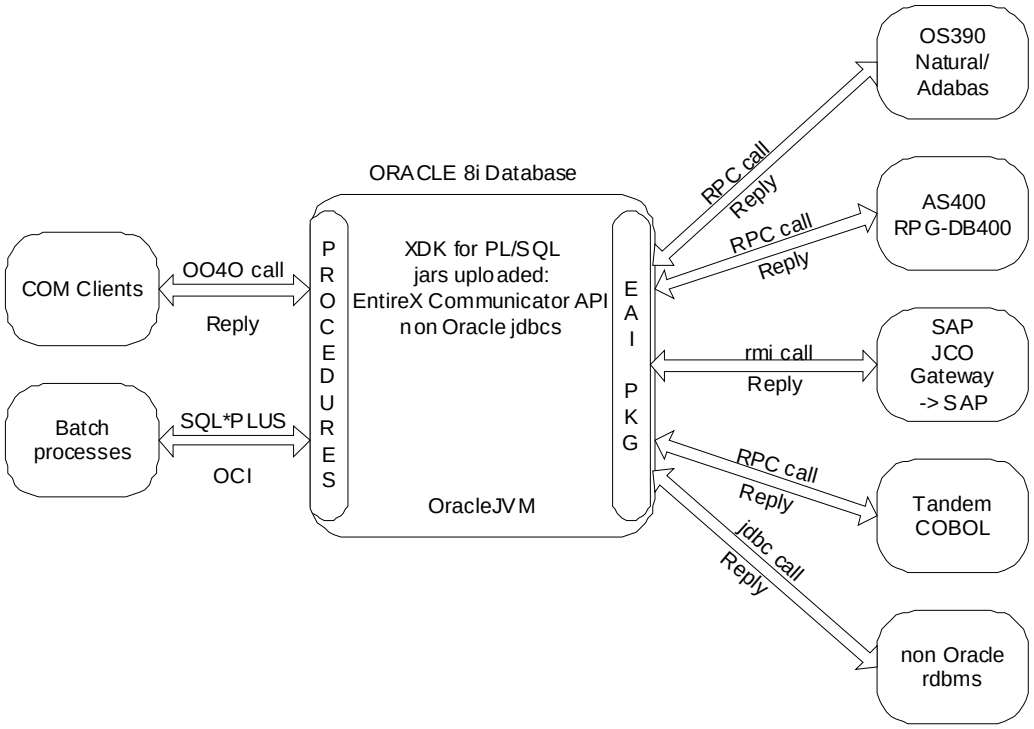


Figure 1: Calling External Systems

PUTTING THIS ALL TOGETHER

As Figure 2 illustrates, by using the Oracle Database as a central point of communication, we enabled any system to talk to any other system, while avoiding point-to-point communication.

The entire framework is monitored by an Oracle stored procedure that, at a determined frequency, performs a call to each target system as well as the EntireX and SAP connector.

If one of the monitored systems returns an error, the procedure sends a notification e-mail. Our next step will be to send SNMP traps to a central console. We are currently working on this.

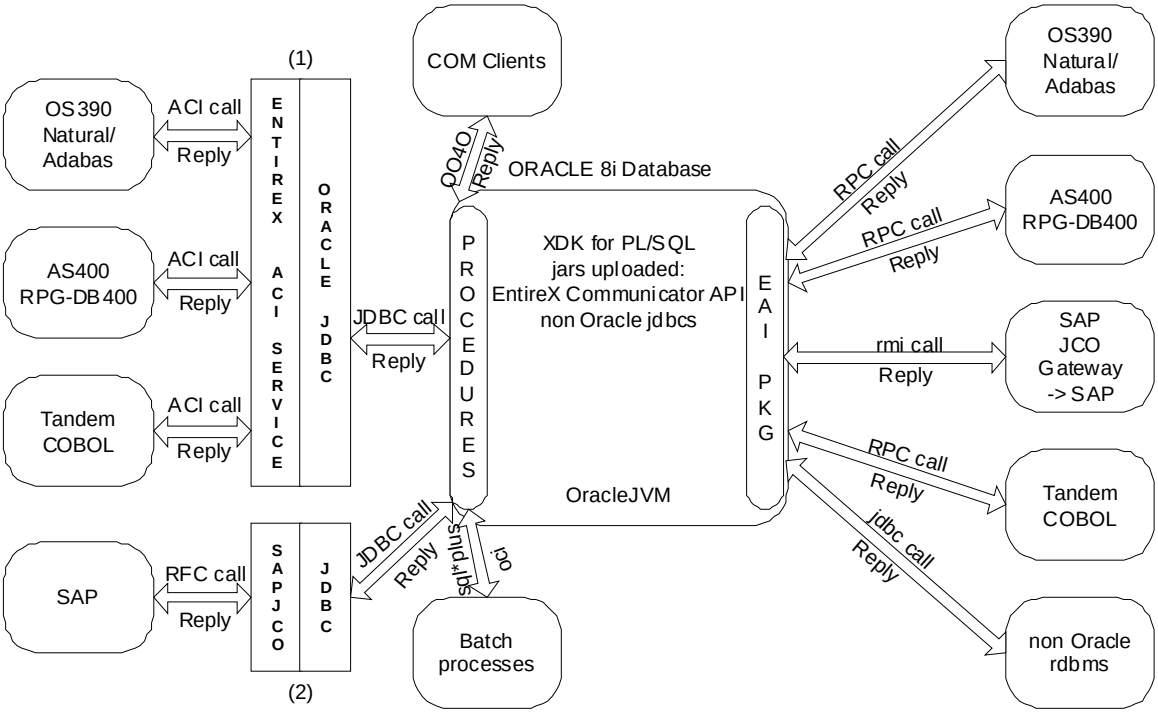


Figure 2: The Entire Framework

CONCLUSION

We were able to implement a complete, easy-to-use integration framework employing Java and PL/SQL procedures. PL/SQL procedures were easy to create and maintain, and we were able to use the existing skills of our programmers. We do not see Java as a replacement for PL/SQL—but, rather, as an enabling technology to extend and improve PL/SQL usage.

By using Java in the Oracle Database to its full capacity, we were able to turn the database into an online integration broker. In addition, we were able to shield our developers from the underlying complexity of our platform, thereby simplifying the integration process.

White Paper Title: Java in the Oracle Database @ Work: Customers Case Studies
August 2003
Author: Lesley A. Brown, John C. Greene, Francisco Juarez, Raymond Cote, Esteban Capoccetti
Contributing Authors:, Sahar Youssef, Ellen Sieg, Kuassi Mensah

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
www.oracle.com