# Connecting NonStop Applications to J2EE

**Michael Horst**
**Chief Technology Officer**
**comForte**
**Neuruppin, Germany**

**Dirk Kruetzfeldt**
**Product Manager**
**comForte**
**Neuruppin, Germany**

## Introduction

This article discusses different approaches for NonStop systems to be an integral part in applications implemented using the J2EE development framework. One approach is the "Hybrid" architecture, where the application server runs on commodity front-end systems accessing business-critical services on a NonStop server. We will demonstrate an implementation of the Hybrid architecture, connecting a Pathway system to an application server using J2EE standard interfaces, even without requiring OSS or Java on the NonStop system. Finally we will briefly look at a Web application using this technology in production, handling 3 million transactions per day.

J2EE (Java 2 Enterprise Edition) proves to be one of the standard development paradigms of the Internet age. Defined by Sun Microsystems Inc., J2EE started to get a grip on the market space in the year 2000. Since then, various application servers have been introduced, such as BEA's WebLogic, IBM's WebSphere, SUN's iPlanet, or the open source JBoss server. Thus, it is no surprise that many organizations decided to standardize on J2EE.

Even though an article in *The Connection* mentioned J2EE as early as 2002[1], NonStop system users initially failed to embrace that development[2]. After abandoning its own initiative for a J2EE application server, HP partnered with BEA, making the BEA WebLogic Application server available for running directly on the NonStop platform. With an industry strength J2EE application server on NonStop systems, HP and BEA are hoping to attract customers "who are interested in achieving the highest levels of scalability and reliability for their J2EE applications, while enjoying the lowest total cost of ownership in the industry[3]."

Beyond "bringing J2EE onto NonStop systems," the important issue is how existing applications running on NonStop servers can be integrated with J2EE application server environments, e.g., to access it from an application implementing a business process involving multiple heterogeneous enterprise information systems.

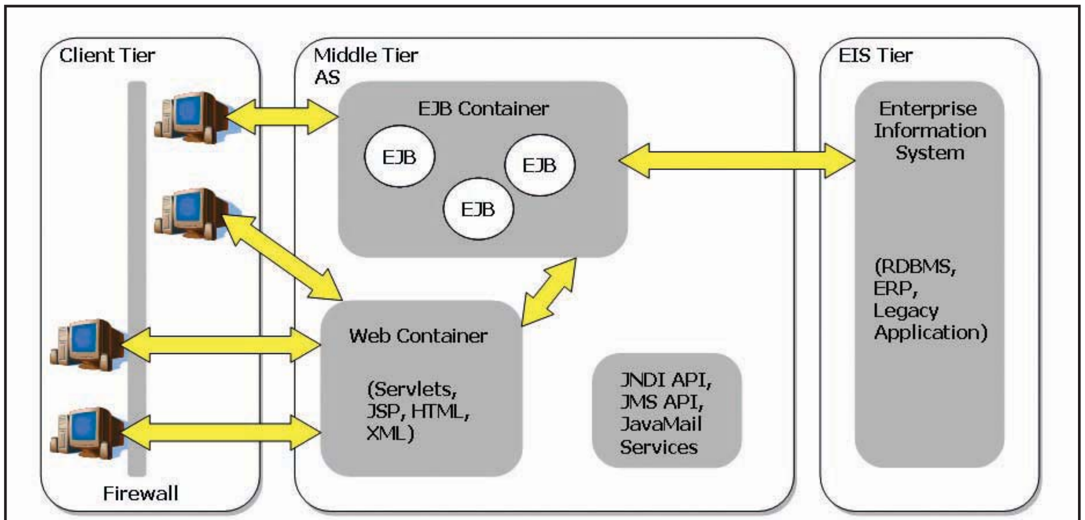
## The J2EE Architecture and Existing NonStop Applications

"The J2EE platform simplifies enterprise applications by basing them on standardized, modular components, by

*Michael Horst is the originator and architect of comForte's MR-Win6530 terminal emulation package. After co-founding comForte in 1998, he helped comForte to extend its product line to become a leading provider of security, Web-enabling and Host integration solutions for HP NonStop Servers.*

*Dirk Kruetzfeldt holds a Ph.D. in computer science. He joined comForte six years ago and currently serves as a product manager for comForte's Web-enabling and legacy integration products. He conducted various projects to help customers migrating existing applications to Web technology.*

▶


Client Tier
Firewall
Middle Tier
AS
EJB Container
EJB
EJB
EJB
Web Container
(Servlets, JSP, HTML, XML)
JNDI API, JMS API, JavaMail Services
EIS Tier
Enterprise Information System
(RDBMS, ERP, Legacy Application)

**Key:** Hypertext Markup Language (HTML), Extensible Markup Language (XML), JavaServer Pages (JSP), Java Naming and Directory Interface (JNDI), Java Message Service (JMS), Enterprise Information System (EIS), Application Server (AS), Enterprise Java Bean (EJB)

Figure 1. J2EE Architecture

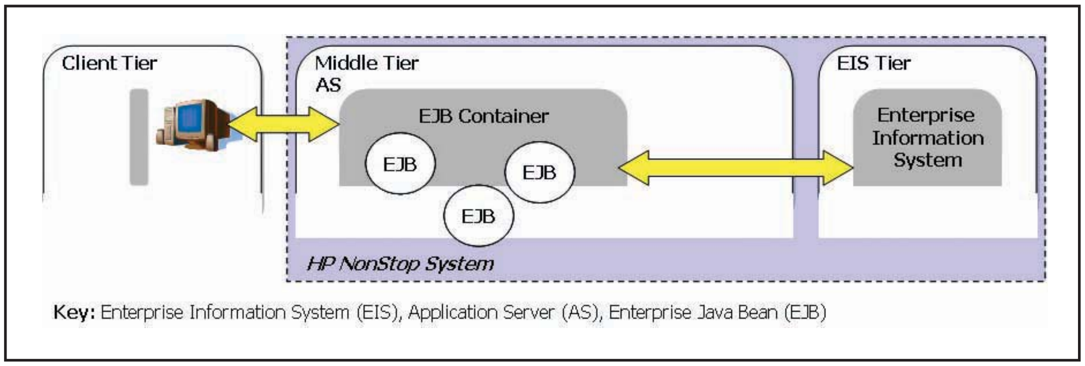

Client Tier
Middle Tier
AS
EJB Container
EJB
EJB
EJB
EIS Tier
Enterprise Information System
HP NonStop System
Key: Enterprise Information System (EIS), Application Server (AS), Enterprise Java Bean (EJB)

Figure 2. The central approach

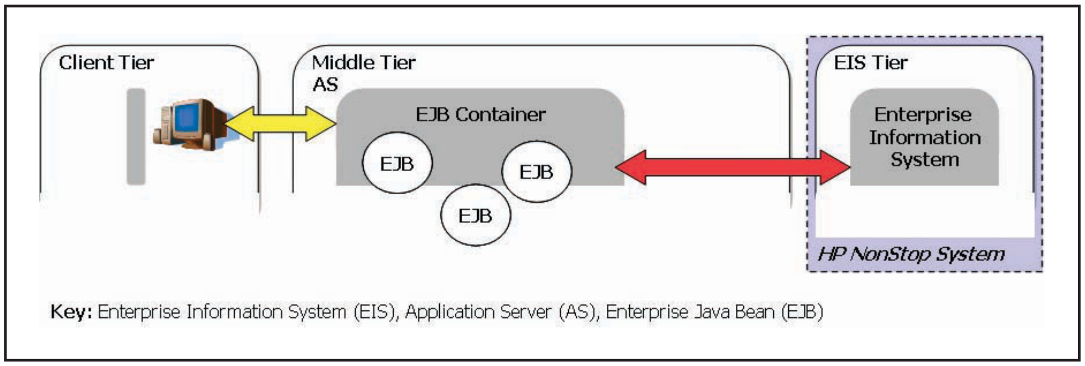

Client Tier
Middle Tier
AS
EJB Container
EJB
EJB
EJB
EIS Tier
Enterprise Information System
HP NonStop System
Key: Enterprise Information System (EIS), Application Server (AS), Enterprise Java Bean (EJB)

Figure 3. The hybrid approach

providing a complete set of services to those components, and by handling many details of application behavior automatically, without complex programming[4].” The J2EE framework slices an application into different layers. Figure 1 shows the architecture of a typical Internet application.

Without going into the details of the various technologies involved, it becomes obvious that integrating a NonStop application boils down to the question of how an Enterprise Bean running in the Application server can be connected to an Enterprise Information System (EIS) on the NonStop server (e.g., NonStop SQL, or a Pathway Legacy Application).

We will now look at two basic approaches for connecting to a NonStop EIS– the central and the hybrid approach.

**The Central Approach.** The first approach is to have the J2EE application server on the NonStop server, where the EIS resides, as illustrated in Figure 2.

Using the Java APIs for legacy application and data access which are available on the NonStop server, the EJBs can be easily connected to an existing application. These Java APIs include:

- JToolkit. Contains a Pathsend API to communicate with Pathway servers, as well as an Enscribe API for direct data file access.
- JDBC drivers. Enable Java applications to use HP NonStop SQL to access NonStop SQL databases.

The advantage of this approach is that full transaction support for these interactions is provided when the BEA WebLogic Server is used.

However, it is not always possible or desired to have the application server running on the NonStop system. For example, the corporate J2EE strategy may not include NonStop as an application server platform. Customer experience also shows that running the complete application server environment along with presentation layer code on the NonStop system may have a significant performance impact on the system. In situations like these the hybrid approach should be considered as a serious alternative.

**The Hybrid Approach.** In the hybrid approach, the middle tier is run on a farm of inexpensive commodity machines which connect to the NonStop system for accessing the EIS. The architecture for accessing a Pathway system on a NonStop server is shown in Figure 3.

The hybrid approach frees costly CPU cycles on the NonStop system while retaining its advantages at the backend of the J2EE infrastructure. However, how can the Application Server communicate with a back-end NonStop server as depicted with a red arrow in Figure 3? While hybrid SQL/MX database access can be implemented with the JDBC Type 4 driver offered by HP, the connection with Pathway applications remains an important issue. This is where the J2EE Connector Architecture (JCA) comes into play.

**The Java Connector Architecture.** The J2EE specification includes the JCA (Java Connector Architecture)[5] which exactly addresses the problem of connecting back-end EIS resources to J2EE application servers. JCA “defines a set of contracts, such as transactions, security, connection management, that a resource adapter has to support to plug in to an application server.”

In terms of the JCA, a NonStop adapter must provide a standard system-level pluggability between application servers and resources on the NonStop server. It also must provide the Common Client Interface (CCI) for accessing the NonStop resources using a standard API.

For a Pathway resource adapter, this would look as depicted in Figure 4.

▶

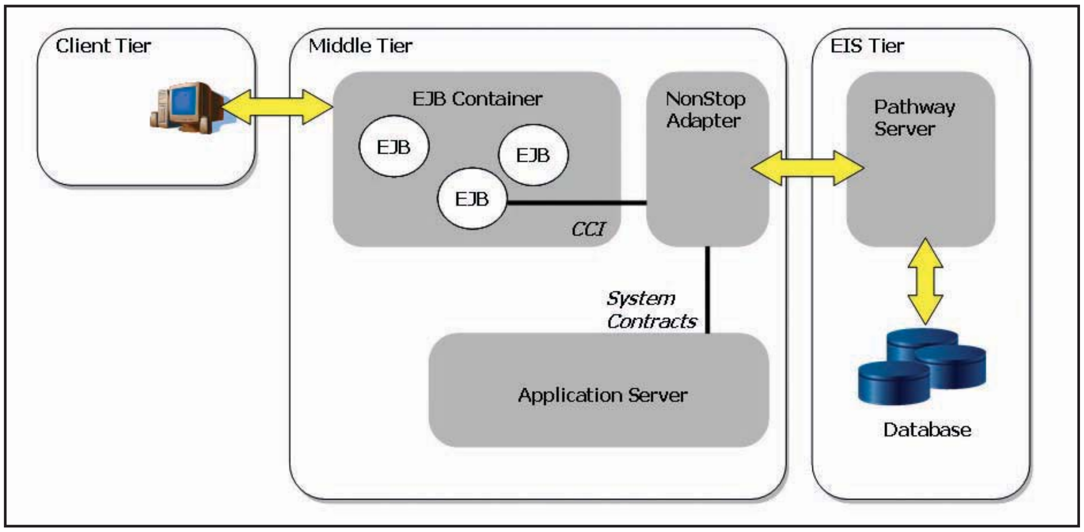


Figure 4. Java Connector Architecture (JCA)

To learn more about how a resource adapter can be used to connect an EJB to a NonStop application, we will now look at a hands-on example. We will use comForte's JSL product to develop a simple EJB accessing a Pathway server.

## Using JSL to Connect Pathway Servers to Enterprise Java Beans

J2EE Server Link (JSL) is a fully functional JCA compliant resource adapter to provide Pathway server connectivity to a standard J2EE application server. Using JSL, you can integrate a J2EE application with an existing Pathway application, without even requiring Java or OSS on the NonStop server. JSL supports any standard J2EE container (e.g., BEA Weblogic Server, IBM Websphere, JBoss, etc.).

**A Simple Connecting Scenario.** To illustrate the use of JSL we will look at an Enterprise Java Bean connecting to the Enable General Server and accessing a little Enscribe Database called VIP. The database is stored in a single file with the following content[6]:

```
$DATA1 DKVIP 3> fup copy vip,,share
00001Modde                   Rene
00002Kruetzfeldt             Dirk
00003Luedtke                 Lothar
00004Lutz                    Andreas
00005Horst                   Michael
00006Mueller                 Ramona
00007Burg                    Thomas
00008Anja                    Lutz
8 RECORDS TRANSFERRED
$DATA1 DKVIP 4>
```

The Enable General Server is a standard Pathway Server for generalized access to Encribe files. It is available on almost any NonStop system as a part of HP's ENABLE product. Using this server for a sample not only illustrates the connection to a pathway server but also provides a general method to access any Enscribe file on the NonStop Server[7].

Figure 5 shows the entire connecting scenario. To link the Enable General Server with the Enscribe Reader Bean, the JSL Server Component on the NonStop system converts the incoming requests into Pathway Server Class Sends. The counterpart on the middle tier is the JSL resource adapter which is a pure Java implementation.

As shown in Figure 5, we will now take a closer look at the three important parts of the solution:

(1) Converting the DDL of message and file definitions into appropriate structures for the application server,
(2) Formatting a request and retrieving response data, and
(3) Sending requests to the Pathway server using the Common Client Interface (CCI).

**Conversion of DDL.** To communicate with the Enable General Server (EGS) we need to convert server request and response data to and from a data format that can be handled by the EnscribeReaderBean. The definitions of the request/reply structures of the EGS and the record format of the VIP data file are maintained in a data dictionary. The DDL source of the VIP Enscribe file is shown below:

```
RECORD VIP.
FILE is VIP KEY-SEQUENCED.

 05  VIP-number     PIC 9(5)   HEADING "VIP-number".
 05  Name.
   10  Surname      PIC X(30)  HEADING "Surname".
   10  Given-names  PIC X(30)  HEADING "Given …".
 05  Sex            PIC X(1)   HEADING "Sex".
 05  Date-of-Birth  PIC 9(8)   HEADING "Date …".
 05  Place-of-Birth PIC X(30)  HEADING "Place …".
 05  Nationality    PIC X(30)  HEADING "Nation…".
 05  Family-Status  PIC X(1)   HEADING "Family …".
 05  Children       PIC X(1)   HEADING "Children".
 05  Phone          PIC X(20)  HEADING "Phone".
 05  Mobile         PIC X(30)  HEADING "Mobile".
 05  Address
   10    Street     PIC X(30).
   10    ZIP        PIC 9(5).
   10    City       PIC X(30).

KEY IS VIP-number.
KEY "NA" IS Name.
END
```

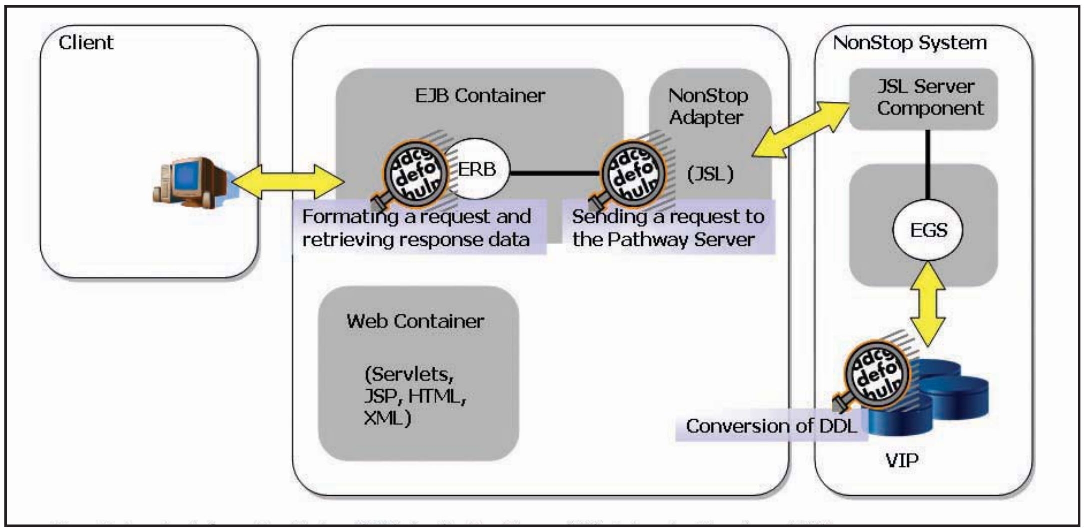

Client
EJB Container
NonStop Adapter
ERB
(JSL)
Formating a request and retrieving response data
Sending a request to the Pathway Server
Web Container
(Servlets, JSP, HTML, XML)
NonStop System
JSL Server Component
EGS
Conversion of DDL
VIP

Figure 5. Connecting Scenario

In order to use these structures in our EnscribeReaderBean, we need to convert the DDL structures using the conversion tools which are included with JSL.

The JSL adapter handles two complex format types: XML and Java objects. In our scenario we will use Java objects. To generate the Java classes we first generate the XML format from the DDL with the following command on the NonStop System:

```
$DATA1 DKVIP 4> run cslbcxml -i $DATA1.dkdict -d VIP -o
$data1.dkdict.vip
```

The second step uses the XML on the J2EE development machine to create a Java class for each structure:

```
C:\JSL > XML2Java vip.xml ddl
```

The XML2Java will create three Java source files ("vip.java," "name.java," and "address.java") within the package "ddl." The main Java class for VIP is shown below.

```
package ddl;

import java.util.Vector;

public class Vip implements
       com.comforte.csl.utils.CSLSerializable {

  protected String vipNumber;
  protected Name name;
  protected String sex;
  protected String dateOfBirth;
  protected String placeOfBirth;
  protected String nationality;
  protected String familyStatus;
  protected String children;
  protected String phone;
  protected String mobile;
  protected Address address;

  public Vip() {
  }

  public String getVipNumber() {
     return vipNumber;
  }

  public void setVipNumber(String theVipNumber) {
     vipNumber = theVipNumber;
  }

  // ... all other getter and setters

  public Integer size(String propName) {

     if(propName.equals("vipNumber"))
        return new Integer(5);
     else ... // and so on

     else
        return new Integer(0);

  }

  public String[] getPropertyOrder() {
     String [] positions = new String[14] ;

     positions[0] = "vipNumber";
     positions[1] = "name";
     ... // and so on
     return positions;
  }
}
```

As the VIP code shows, the Java classes generated for the DDL structures contain member variables for each field. Additionally, the “getPropertyOrder” and “size” methods provide information about field order and sizes which is used by JSL to create a correctly formatted server request, or to extract data from the response.

**Formatting a Request and Retrieving Response Data.** With the Java classes created by the conversion tools we can easily access the server message data. For example, to retrieve data from a VIP record contained in the response from the EGS, we simply use the “get” methods of the VIP object we read from the message input stream. The following code shows how to access the VIP’s name:

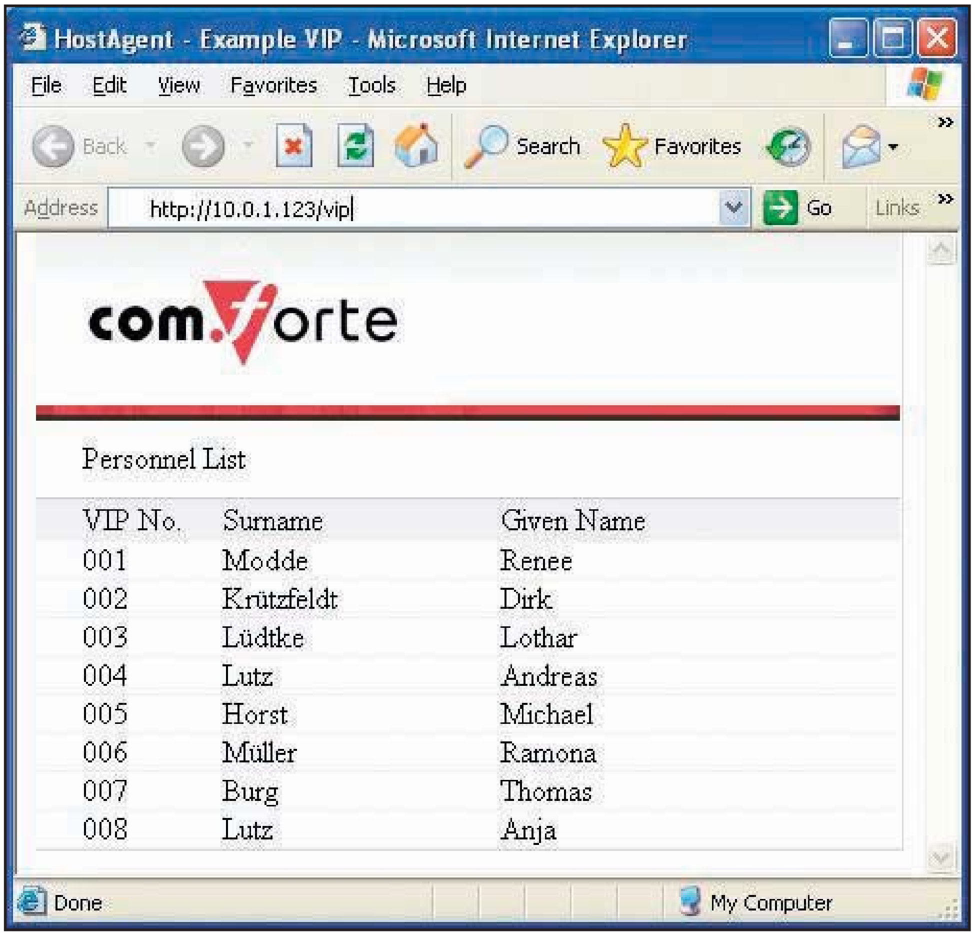
```
/*
  * get the response structure from the CCI
  */
 JSLInputStream in = getJSLInputStream(/*CCI*/);

 vip = (Vip) in.readObject(vip.getClass());

 …

 Name name = vip.getName();
 String surname    = name.getSurename();
 String givennames = name.getGivenNames();
```

**Sending a Request to the Pathway Server.** Much of the Pathway server connection handling is hidden by the standard services that the JSL resource adapter provides together with the J2EE application server. This extremely simplifies the code required in the EJB. The following EnscribeReaderBean’s pseudo code looks up the NonStop Adapter “JSL” to get a physical connection to the NonStop system from the pool handled by the application server. With this connection, several server class sends to the EGS Pathway Server are performed, each retrieving a single record from the VIP data file until the end of the file is reached.


HostAgent - Example VIP - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Address
http://10.0.1.123/vip
Go
Links
comforte
Done
My Computer

Personnel List

| VIP No. | Surname | Given Name |
|---|---|---|
| 001 | Modde | Renee |
| 002 | Krützfeldt | Dirk |
| 003 | Lüdtke | Lothar |
| 004 | Lutz | Andreas |
| 005 | Horst | Michael |
| 006 | Müller | Ramona |
| 007 | Burg | Thomas |
| 008 | Lutz | Anja |

Figure 6. VIP data in a Web browser

```
  /*
   *  Look up NonStop Adapter
   */

  connectionFactory =
            applicationServer.lookup(/*JSL*/);

  /*
   * get connection from
   * connection pool
   */

connection =
            connectionFactory.getConnection(
            /*User, Password*/);

  /*
   * create a interaction
   * to NonStop server
   */

  interaction = connection.getInteraction()

  while (/*not EOF*/) {

   interaction.execute(
/* System, EnableGS, Data */);
  }
```

Having analyzed the important parts of the EnscribeReaderBean, it is obviously very easy to integrate remote Pathway servers using the standard interfaces of the J2EE Connector Architecture. Of course, a typical J2EE Web application would also include the presentation layer using the EJB functionality. Our sample uses a Java Servlet displaying the data provided by the EnscribeReaderBean in a Web browser. Figure 6 shows the Web browser display.

**Evolving the Scenario.** Of course, real life applications consist of more than one Enterprise Java Bean reading data from a single Pathway server. Multiple EJBs implementing the business logic will be involved, some of them performing add and update transactions with the application. This is when the real advantages of J2EE start to pay off. Much of the complexity is managed by the application server without the developer having to bother.

For instance, how many code changes do you think are required to add transaction management (via TMF) to an EJB performing several interactions with Pathway servers? None! The transaction management is transparently handled by the Transaction Manager of the J2EE Application server, after we configured it to enable transactions for our EJB.

## The Hybrid Approach in Real Life

The hybrid approach for J2EE NonStop integration has already been proven to provide a solid architecture for real life applications with very demanding performance requirements. For example, a NonStop customer converted a large Pathway application into a true Internet application handling high volumes of online bookings and reservations. The J2EE application runs with IBM's WebSphere server on several AIX machines. The connection to the Pathway system running on NonStop platform is provided by JSL.

The architecture runs in production on an S86.000 system with 6 CPUs. On average, three million transactions per day are processed peaking at about 300 TPS. The architecture allows for 30 million transactions per day.

This implementation highlights the benefits of combining the best of both worlds. Running the J2EE application server on Unix systems keeps the presentation layer off the NonStop server, freeing it to only run the business-critical part of the Web application.

## Conclusion

Undoubtedly, J2EE will be a technology of choice to implement complex Web and enterprise integration applications within the years to come. In order to increase its market share, the NonStop has to move beyond Pathway as application infrastructure while at the same time retaining access to the high number of business-critical legacy applications running on the platform. The future will reveal if NonStop servers can establish themselves as a viable platform to host J2EE application servers. In any case, however, NonStop servers will continue to provide business-critical back-end application services. The fundamental advantages of the NonStop platform can be unleashed for J2EE with the hybrid approach presented in this article. Standard J2EE connectors for NonStop resource access, like comForte's J2EE Server Link for Pathway integration, provide the glue between the best of both worlds.

1. "J2EE on the NonStop™ Platform" by Rich Raposa, The Connection, January/February 2002, Volume 23, No. 1.
2. "Web Technologies on the NonStop Platform: Why Bother? Which Ones?" by Thomas Burg, comForte GmbH, The Connection, July/August 2002 Volume 23, No. 4.
3. "BEA WebLogic Server software on HP NonStop servers" (http://h71028.www7.hp.com/ERC/downloads/5982-3503EN.pdf)
4. "Java 2 Platform, Enterprise Edition (J2EE) Overview" on http://java.sun.com/j2ee/overview.html
5. J2EE Connector Architecture Specification", Version 1.0, Final Release, August 22, 2001
6. Content shown only partially for brevity
7. The complete source code of the sample can be downloaded from www.comforte.com/j2ee/sample along with an out-of-the-box application ready to be installed with a BEA Weblogic or JBOSS application server.