A Parachute For Your Java Investment

Migrating a J2EE (Oracle/BEA's WebLogic® Server) based application to a NonStop® Java Server Pages (NSJPSP V6.0) based application

Jürgen Depping

Jürgen Depping is a co-founder of CommitWork GmbH. He is responsible for the Java development at CommitWork. Since 2004 he is the head of the GTUG Java SIG and is Co-Moderator of the Connect Java SIG.

Abstract

The article describes how a J2EE Oracle WebLogic® based application was ported to a NSJSP run time environment. It starts out with a historical overview followed by a short discussion why porting could not be avoided. Next is a comparison between the two run time environments pointing mainly at advantages and disadvantages of NSJSP. The technical requirements, like project architecture, communication, type of J2EE beans and JMS are described and how solutions were found for the NSJSP environment. The article concludes showing the project results, including the investment protection and gives an idea about porting cost.

Project History

After evaluating requirements and objectives for an appropriate run time environment, our customer decided in 2004 to modernize his applications using Java on the NonStop® Server platform.

After delivering a series of individually shaped workshops, the customer finally decided for CommitWork's OmnivoBase development framework on top of a J2EE App Server (BEA's WebLogic® Server (WLS)) run time environment.

In 2004 BEA WLS Release 8.1.2 was available for NonStop® Servers. This App Server was viewed as a strategic product by HP, since the development of a "home grown" App Server had been cancelled. At this time BEA's WLS also was the market leader within the Java App Server market for Open Systems.

The main reasons for using an App Server could be seen in the high level of standardization and the extreme bandwidth of possibilities for applications. The most important promise was and still is to provide the highest level of portability between different server platforms.

The most important criteria supporting the customer's decision could be found in the usage of Web Services for integration with SAP applications and in the parallel usage of a protocol for a performant coupling with Rich Client applications. Other decision criteria were high scalability as well as high availability of the App Server. Traditionally those features were occupied by Pathway, the legacy transaction manager for NonStop® systems, and the customer required them with the same quality for a new run time environment.

Based on the decision for OmnivoBase and BEA's WLS, several development projects were launched and finally set in production successfully.

During these projects some weaknesses of BEA's WLS 8.1.2 became obvious, known bugs which were already fixed by BEA in WLS Release 9.2. Later on, this version has been certified by BEA/Oracle for HP NonStop Integrity Systems."

At the end of 2008 BEA's takeover by Oracle was finalized. After this it is not clear if HP's NonStop® division decided or was forced to by Oracle to let the porting contract expire for future ports of newer WLS releases. Up to this date there is no HP announcement for either to support a newer WLS release on NonStop® Servers, nor to provide any other App Server platform. As WLS is becoming more and more a dead end road on NonStop® Servers, our customer found himself in a situation to react.

At this point the customer had invested a lot of efforts and money in using technologies for to deploy Java applications on NonStop®. Because of the extraordinary architecture and the other advantages of the OmnivoBase framework almost all of these investments could be preserved.

As an alternative run time environment for Oracle's WLS, HP's NonStop® division has developed and integrated a combination of popular Open Source software, mainly a modified version of Apache's Tomcat™, called NonStop® Java Server Pages (NSJSP). Since in this environment Tomcat™ is embedded in the proven Pathway, Tomcat™ inherits Pathway's great advantages and is also integrated with HP's ITP Webserver.

Our customer accepted NSJSP as a valid alternative for Oracle WLS and started porting the existing Java applications. The following report deals with the porting challenges and takes a closer look at the selected porting approaches.

Middleware Stack of the existing projects

In order to understand all requirements for the new middleware, the used architecture has to be presented.

In all projects OmnivoBase represents the highest level of middleware stack components. It embeds the Java programs representing the customer's application. It forces the concept of Client-Server program pairs and provides a lot of additional comfort functions. These are functions like Authorization, Menu-System, Rights for specific Dialogues, Logging, Client control, Master Data Management and a broad collection of supporting object classes.

On the client side a dialogue consists of three different layers. Java Swing elements are used for the GUI presentation. Typically GUI builders like JBuilder®, Netbeans® or Google WindowBuilder are used. The GUI builder allows defining and setting features of text fields, description fields, tables and

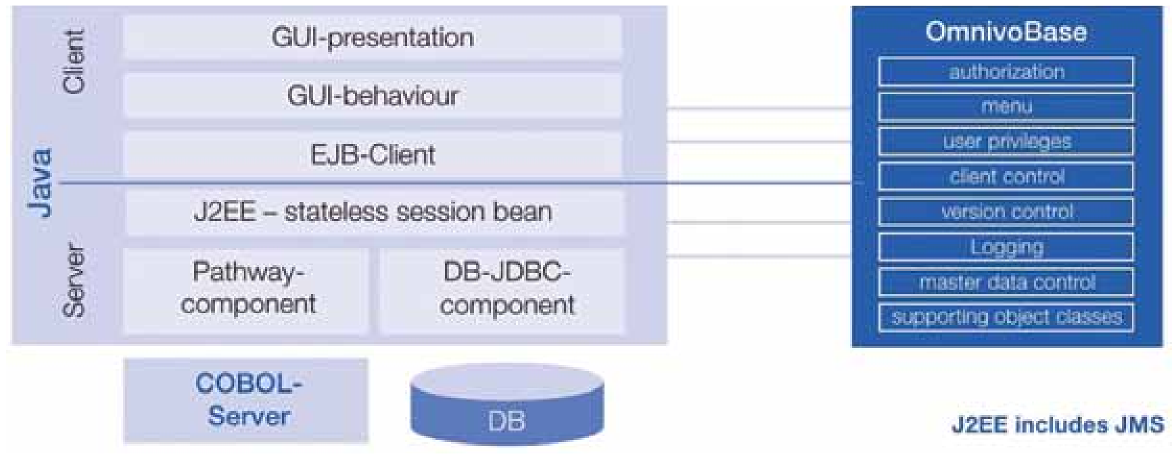

figure 1: OmnivoBase in a J2EE Environment

other elements like for example drop down menus by using a WYSIWYG editor. While working with a chosen editor, source code is generated automatically. This source code is different for every single GUI Builder. In order to minimize dependencies, the so called "Dialogue Behavior" was put into a separate Java class. This layer is called GUI behavior. The decision to separate these specific functions into different layers was necessary to allow customers exchanging the GUI Builder even during the course of a distinct project. So changing the GUI Builder was just an easy task.

Another separate layer within OmnivoBase's Client module is strictly designated for Server Communication. Especially this architectural approach (having every function separated in an extra layer) now allows an easy port to a Tomcat™ based run time environment.

On the server side similar stack components can be found. Client communication connects to a "Stateless Session Bean". Finding the appropriate Bean is provided by the "Naming Service" of the App Server (JNDI). In a fault tolerant J2EE App Server environment a cluster of App Server instances exists. Finding the right instance is accomplished thru the so called "Business Objects Locator". Because of performance reasons previously used "communication stubs" are cached within OmnivoBase by the Business Object Locator.

In all projects only "Stateless Session Beans" have been used. A J2EE App Server also provides several other Beans like "Stateful Bean", "Message Driven Bean" or "Entity Bean". The following text takes a closer look at those Beans.

A Stateful Bean is entitled to manage context data, so Client-Server interactions coming later can have access to this context data. This requires that each and every Client has a Stateful Bean associated. The result is an extreme usage of App Server resources, if most of the application dialogues would use Stateful Beans. In addition, context would have to be replicated in an App Server cluster environment. Therefore the use of Stateful Beans had to be avoided.

Message Driven Beans are typically used in message oriented applications. Messages can be queued by the standardized Java Messaging System (JMS) and are then forwarded to a Message Driven Bean. In the present project cases JMS was only used for Client-Client communication.

Within Java applications Stateless Session Beans are treated like capsules, responsible for the following functions: communication, transaction handling and forwarding of Business Methods. The specific business logic resides in other Java classes. Therefore Stateless Session Beans only provide a very limited collection of Methods.

For communication purposes with exiting Pathway legacy servers (Cobol and C) our specific customer is using his own homemade Java to Pathway object classes.

Entity Beans for accessing the databases were explicitly abandoned, because their features were not covering real life project needs. Only the very latest App Server versions are providing the new Java Persistence APIs (JPA).

The customer is using NonStop® SQL/MX as the primary database. SQL/MX tables are accessed directly thru the genuine database engine while SQL/MP tables are accessed via database aliases using the SQL/MX engine. A homemade DB generator supports database programming by generating DB object classes including standard access methods and transport classes for the Client-Server exchange of data. These generated classes could also be used for implementing extended queries. The OR mapper Hibernate was evaluated as an alternative, but was finally rejected because of its different database programming approach.

As a result from looking at the given situation, the porting project had to face 2 major challenges:

1. Exchanging the „Business Object Locator" and the

continued on page 42

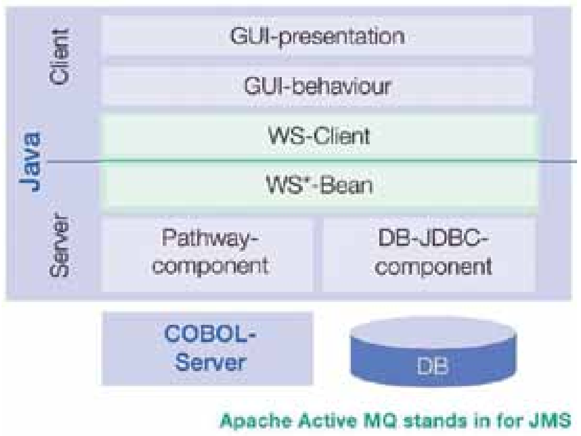

figure 2: OmnivoBase in a NSJSP Environment

A Parachute For Your Java Investment

continued from pg. 19

communication layer

2. A different JMS (Java Messaging System) for the Client-Client communication

Exchange of Communication layer

Web containers like Tomcat™ provides fewer communication features compared to a fully fledged App Server. Additional Open Source software is available to compensate. Non standardized Binary Protocols could be used as well as the so called Spring™ Framework.

Because of the desire for independence of Web services, Apache's Axis2™ was evaluated.

Axis2™ created problems with the so called "Exception Handling". It is used in Java programs for returning error situations. Historically server functions produced a return value, capable of indicating errors. For Java and also for C++, Exception Handling is the option of choice for reporting error conditions. In an error situation an exception is returned without limiting other returns from methods. Therefore a method can return several exceptions.

This desired Exception Handling concept could not be implemented by using Axis2™. The result would have been a much higher porting effort.

Looking for alternatives finally led to Apache's CXF Web services. CXF was selected as the best choice and it showed later much better performance values than Axis™.

Exchange of the Client Business Object Locator

App Servers are using the JNDI Naming services, Web services are addressed thru URLs.

Within OmnivoBase the JNDI Naming service is not invoked directly, the address detection is encapsulated within the Business Object Locators instead. This concept enables supporting cluster environments and switching between several servers in case of a failure.

In the process of exchanging the Business Object Locators, the interface was kept the same. The different behavior of Web Services was moved into object classes of the Business Object Locators. For the application project porting, no additional effort was necessary.

The Application Server Remote Interfaces – created by "xdoclet" – could be used with CXF without any changes. Only for the "cxf-servlet.xml" file – every single Bean had to be filled in separately – data had to be entered manually. Nowadays with new dialogues this process is covered automatically by the OmnivoBase wizard.

Exchange of Server Beans

Server Beans in combination with the App Server were created with "xdoclet. Having this creation process in place, the features of Beans were defined via Meta Comments.

Those Meta Comments could be preserved in the object classes or they have been removed during a rework process.

Because of the abstract declaration of the "Application Server Beans", some rework had to be performed on them. In addition the "Implement Command" had to point to the "CFX Remote Interface". The methods required by the Interface Session Bean could be removed. For "Session Context" control, a complementary object class from OmnivoBase could be used.

Only "Bean Managed Transactions" (BMT) and no "Container Managed Transactions" were used. Therefore transaction clauses were set traditionally by the programmer and were not left up to be set by the container. This approach allowed a simple porting of transaction handling to the Tomcat™ (NSJSP) environment.

In addition every Bean had to have an entry in the "cxf-servlet.xml" file.

By now it should be obvious, that only very few changes were necessary on the Server side.

Exchanging Interfaces made adaption necessary

During the exchange process for existing interfaces a specific challenge popped up: CFX returned a "null" value within the array instead of an empty array. But the programs expected an array with zero entries. This different behavior had to be adapted within the programs.

The objects which had to be ported had to be restricted to typical Web services features. The transport classes must contain "Setter"- and "Getter" methods for each attribute. They also require primitive data types, arrays or objects, built within a complementary approach. Only in a few exceptions, transport classes had to be modified accordingly.

Additional activities

Each of the 20 application projects had to have an adapted project structure and the „Build Scripts" had to be customized.

The previously mentioned changes were also necessary for OmnivoBase internally and a few new utility object classes had to be introduced for the projects.

The Oracle WLS specific Time Services were replaced by proprietary implementations within OmnivoBase.

An alternative for J2EE JMS

While App Servers provide a built in JMS (Java Messaging System), Tomcat™ is missing this kind of functionality. In the customer projects JMS was only used for Client-Client communication. "Message Driven Beans" were not used at all.

Exchanging messages between Control Stations to notify about changes in the customer's production process is an example for Client-Client communication. Also administrators can use this feature for controlling Clients. For example messages can be sent to Clients in order to check the version of the Client application or to shut down the Client application remotely. In both cases Client-

Client JMS communication is involved.

Within the Tomcat™ environment a JMS replacement was built by using Apache's " Active-MQ". NonStop® features were used to create an environment for ActiveMQ™ by configuring it as a generic process. When the JMS replacement system fails, it will be restarted automatically. From the application project point of view no changes were necessary, because corresponding object classes were customized within OmnivoBase.

Advantages and Disadvantages of using NSJSP

Performance measurements showed that time used up for communication has tripled. The reason can be found in the increased amount of data when communicating. Web services communication involves user data as well as a big portion of metadata for description purposes to be transmitted. The follow up process interprets XML messages and creates Java objects.

Looking at the whole work within a given service including database accesses, the communication portion is fairly small compared to the time elapsed for whole service. So eating up more time for communication is less important and is also acceptable for the users. Response time is typically below 5 sec in 98% of all cases. Exceptions to this limit are reasonable and are due to complexity and very high data volumes.

A drawback was the necessary replacement of JMS (Java Messaging System) by using an additional technology. But in production this was not visible as a disadvantage at all.

The App Server provides a sophisticated GUI for administration in a very high granularity for monitoring the application and the App Server environment. Here is NSJSP much simpler and offers fewer options. In the future optional Open Source software in addition could eliminate this disadvantage.

One of the major advantages using NSJSP is the ongoing development process which will continuously provide new versions of the Open Source software involved within the Tomcat™ environment. In contrast, the usage of an App Server on NonStop® was critical in this respect: only major new releases were certified for NonStop® and therefore provided for this platform.

Less system resources were required for NSJSP compared to the resource requirements of an App Server on the NonStop® platform. This turns out to be complementary for NonStop®, since JVM (Java Virtual Machine) on NonStop® can only handle less than one Gigabyte of memory.

HP's dedication for supporting the whole NSJSP environment turns out to be a big advantage over the support for WLS provided by Oracle: no finger pointing and the customer always knows whom to contact.

Conclusion

The porting case from WLS to NSJSP did not introduce any new and unknown obstacles. It finally took 10 days to migrate those 20 application projects onto the new middleware. Operation of the new environment was showing higher stability compared to before. Also the new JMS replacement system did not bring up any problems.

Exchanging the middleware helped preserving the high investment in new applications.

Therefore the migration from WLS onto NSJSP was a big success for this specific customer.

For sure, the shown migration process cannot be equally adopted for all J2EE App Server projects. But there are always ways to find appropriate solutions for missing functionalities. For example, a good candidate could be the Spring™ Framework, which is already supported by HP within the NonStop® SASH stack.