RESEARCH REPORT

Assisted SOA Development

Productivity Comparison of SOA Development Tools

http://www.MiddlewareRESEARCH.com

The Middleware Company
Research Team
November 2004

research@middleware-company.com

1 DISCLOSURES

1.1 Code of Conduct

The Middleware Company offers the world's leading knowledge network for middleware professionals. The Middleware Company operates communities, sells consulting and conducts research. As a research organization, The Middleware Company is dedicated to producing independent intelligence about techniques, technologies, products and practices in the middleware industry. Our goal is to provide practical information to aid technical decision making.

- Our research is credible. We publish only what we believe and can stand behind.
- Our research is honest. To the greatest extent allowable by law, we publish the parameters, methodology and artifacts of a research endeavor. Where the research adheres to a specification, we publish that specification. Where the research produces source code, we publish the code for inspection. Where it produces quantitative results, we fully explain how they were produced and calculated.
- Our research is community-based. Where possible, we engage the community and relevant experts for participation, feedback, and validation.

If the research is sponsored, we give the sponsor the opportunity to prevent publication if they deem that publishing the results would harm them. This policy allows us to preserve our research integrity, and simultaneously creates incentives for organizations to sponsor creative experiments as opposed to scenarios they can "win."

This Code of Conduct applies to all research conducted and authored by The Middleware Company, and is reproduced in all our research reports. It does not apply to research products conducted by other organizations that we may publish or mention because we consider them of interest to the community.

1.2 Disclosure

This study was commissioned by Oracle Corporation.

The Middleware Company in the past has done other business with Oracle.

Moreover, The Middleware Company is an independently operating but wholly owned subsidiary of Veritas Software (www.veritas.com, NASDAQ:VRTS). Veritas and Oracle have a number of business relationships in certain technology areas, and compete directly against each other in other technology areas.

Oracle commissioned The Middleware Company to perform this study on the expectation that we would remain vendor-neutral and therefore unbiased in the outcome. The Middleware Company stands behind the results of this study and pledges its impartiality in conducting this study.

1.3 Why are we doing this study? What is our "agenda"?

We are compelled to answer questions such as this one, due to controversy that sponsored studies occasionally create.

First, what our agenda is *not*: It is not to demonstrate that a particular company, product, technology, or approach is "better" than others.

Simple words such as "better" or "faster" are gross and ultimately useless generalizations. Life, especially when it involves critical enterprise applications, is more complicated. We do our best to openly discuss the meaning (or lack of meaning) of our results and go to great lengths to point out the several cases in which the result cannot and should not be generalized.

Our agenda is to provide useful, reliable and profitable research and consulting services to our clients and to the community at large.

To help our clients in the future, we believe we need to be experienced in and be proficient in a number of platforms, tools, and technologies. We conduct serious experiments such as this one because they are great learning experiences, and because we feel that every technology consulting firm should conduct some learning experiments to provide their clients with the best value.

If we go one step further and ask technology vendors to sponsor the studies (with both expertise and expenses), if we involve the community and known experts, and if we document and disclose what we're doing, then we can:

- Lower our cost of doing these studies
- Do bigger studies
- Do more studies
- Make sure we don't do anything silly in these studies and reach the wrong conclusions
- Make the studies learning experiences for the entire community (not just us)

1.4 Does a "sponsored study" always produce results favorable to the sponsor?

No.

Our arrangement with sponsors is that we will write only what we believe, and only what we can stand behind, but we allow them the option to prevent us from publishing the study if they feel it would be harmful publicity. We refuse to be influenced by the sponsor in the writing of this report. Sponsorship fees are not contingent upon the results. We make these constraints clear to sponsors up front and urge them to consider the constraints carefully before they commission us to perform a study.

2 TABLE OF CONTENTS

3 EXECUTIVE SUMMARY

Service Oriented Architecture (SOA) is an architectural paradigm that prescribes the usage of a set of services and a method of communication for those services. A service itself is intentionally defined loosely as some functionality that is independent of other services.

Assisted development automates the many tasks that developers and web designers must accomplish when developing an application. Oracle's Application Server platform, BEA's WebLogic Integration and IBM's WebSphere Studio Application Developer Integration Edition are all toolsets that take the assisted development approach to SOA development.

The Middleware Company conducted an experiment with these tools. Two developers were given the task of building the same application on each of the three platforms. This method conforms with criteria and processes developed to evaluate developer productivity by The Middleware Company in the past.

The developers implemented *The Middleware Company SOA Application Development Specification*, a productivity-oriented specification of the *Expense Reports* section of the *SOA Blueprints Specification*. *The Middleware Company SOA Application Development Specification* was selected to represent a typical SOA development scenario. The *SOA Blueprints Specification* is an expert reviewed multi-vendor initiative to define a set of standard applications.

The tools provided varying approaches and enhancements for the development of SOA applications. The developers were able to construct the BEA version in a total of 184 hours. The IBM application was constructed in a total of 164 hours. The Oracle implementation took 127 hours.

4 INTRODUCTION

This report compares the productivity of two developers building the same Service Oriented Architecture (SOA) application with each of three different development tools. The application followed a functional specification, *The Middleware Company SOA Application Development Specification*, a productivity-oriented specification of the *Expense Reports* section of the *SOA Blueprints Specification*.

The three tools used were:

- Oracle's Application Server 10g (9.0.4), including JDeveloper 10g (version 9.0.5.2) and the Oracle BPEL PM (version 2.0rc10).
- IBM's WebSphere Studio Application Developer Integration Edition (WSADIE) (version 5.1.1)
- BEA's WebLogic Integration (WLI) (version 8.1), which is built upon the BEA WebLogic Workshop platform

All of these tools are intended to assist the developer by automating many of the tedious and error prone tasks of SOA development.

This report is organized as follows:

- We begin in Section 5 with an overview of SOA and the benefits it promises. We also explain the other technologies used in the study, namely MVC and O/R mapping.
- In Section 6 we look at how the development tools used in the study assist development of SOA.

- Next, in Section 7 we describe the study in detail: the specification, the application, the rules, the developers and so forth.
- Finally, in Section 8 we examine the results of the study – the structure and quality of the code produced with each tool, the qualitative experience of the developers, and of course the quantitative results. Finally, we drill deeper and explore which features impacted the developers' productivity.

5 SERVICE ORIENTED ARCHITECTURE

5.1 What is Service Oriented Architecture?

A service is simply some body of functionality that is independent of any other service. For example, a notification service could accept an XML document and send an email or fax based on the contents of that document. Services can utilize each other in what is known as composition but they should be loosely coupled as in the case of components so they can run in separate environments.

Loose coupling is one of the primary goals of SOA. Most corporations now have a large number of applications with numerous interdependencies. If these applications can be expressed as some number of services that are loosely coupled, management of the dependencies is greatly simplified. For example, a financial service and a manufacturing service that depends upon a human resources (HR) service. In a tightly coupled arrangement, the financial and manufacturing services would talk to the HR service through two separate interfaces. If the platform supporting HR is changed then one must rebuild both those interfaces to the other services. However, using SOA, only the one interface to HR would need to be addressed regardless of how many other services depended on it.

While this architecture bears some similarity to the traditional hub-and-spoke scenario, SOA includes the concept of the Enterprise Services Bus (ESB). Differing from the traditional message oriented (MOM) hub offering a single mediator, the ESB is composed of various mechanisms, such as messaging support, along with a robust Web services framework based on numerous Web Services Interoperability (WSI) specifications including WS-Security, WS-Transaction and others (often called the "WS-*" layer).

Services generally fall into two categories: horizontal and vertical. Horizontal services are reusable across any application and often represent the core services layer. Examples of horizontal services are persistence, transactions and notification. Vertical services may be reusable across some applications but in general are usually tied to an application or domain. An example of a business specific service would be the Submit Expense Report service defined in *The Middleware Company SOA Application Development Specification.* Since this study focused on productivity, the services in the specification were all business services, reflecting the fact that end users would probably develop business services and buy generic services.

All three development tools are workflow based. Both the Java Process Definitions (JPD) platform supported by BEA and the Business Process Execution Language (BPEL) specification supported by Oracle and IBM offer a mechanism to assign tasks to users in the midst of a business process. When the human user completes the task, the process is resumed. This is illustrated in the Submit Report and the Authorize Report processes.

5.2 What are the Benefits of SOA?

Service Oriented Architecture offers the following benefits:

Faster development time. Code generation can save you the "grunt work" required to hand-write the same files repeatedly. With the traditional approach, a business process or a Struts front end, for example, could require many Java classes and XML files. A clever SOA tool can automate most of this.

Architectural advantages. As mentioned above, there are significant benefits in terms of loose coupling from using a SOA. From the perspective of the tools, the BPEL or process designers present a graphical "flowchart" style of diagram where the developer can design the process graphically then "fill in the blanks" with code and configuration.

Improved code consistency and maintainability. The maintenance benefit follows from SOA in general. Services can be inserted and removed from the environment due to the loose coupling inherent in the architecture. Code consistency is achieved by the refactored nature of a SOA environment. Since services can be invoked by other services, the core frameworks along with other evolving frameworks are reused and duplication inconsistencies are avoided.

Interoperability across architectures, middleware vendors and platforms. Interoperability comes almost by definition in SOA. Web services are not mandatory for SOA in general but all products involved in this study included very strong support and focus on Web services. Since Web services are standardized by all the major vendors there is a high level of platform interoperability. For example, the Oracle BPEL platform can be used to develop a Java based business process. This process can be exposed as a Web service and invoked by a Microsoft ASP.Net application.

This report focuses on evaluating the benefit of faster development time. It does not address the other potential advantages of SOA.

5.3 What is MVC?

The Expense Reports application was designed to illustrate the use of SOA within the environment of a typical web application. All three products involved used a Model-View-Controller (MVC) approach based on Struts. MVC is a long-established design pattern used to decouple layers in a user based application. The "Model" represents the data layer, such as a database accompanied by data access objects, and should exhibit no behavior. The "View" is the presentation. In the case of the products used in this study, the view is the Java Server Pages (JSPs). The view simply presents the data and the interface to the user. The "Controller" contains the majority of the behavior. The responsibility of the controller is to access and modify data in the model and provide data to the view. The benefits of MVC include minimizing change "ripple" effects, separating design concerns and centralizing control.

Struts is a popular MVC framework distributed as open source by The Apache Software Foundation. Struts provides a controller implemented using servlet technology, along with tag libraries to support the view and form beans to transport data between the view and the model. An example of Struts in action would be a JSP containing fields for customers placing orders to enter their address. When the user submits the address form, the Struts ActionServlet controller receives the information from the fields and places them into the attributes of a form bean. It then passes the form bean to an Action object and invokes the Action, which contains logic to place the data in a database.

Struts Action objects can integrate with a SOA using several different techniques. If the SOA is based upon J2EE components, then the Action object can use JNDI to discover and invoke the J2EE underpinnings of the SOA. BEA offers proprietary controls that expose the WLI processes to web applications. Web services are an alternative available to all platforms, J2EE and otherwise. (This presumes of course that the Web service is compliant with the WS-I interoperability specification.)

5.4 What is O/R?

Object/Relational mapping or O/R is a technology that simplifies an application's access to a relational database by providing a layer of objects that serve as proxies to the database.. BEA supplies "database controls", but these simply map SQL statements to methods in a generated class; these controls don't qualify as O/R since they are not directly mapped to database objects. O/R technologies can expose database elements as "Plain Old Java Objects" (POJO's) or as Enterprise Java Beans (EJB's).

O/R mapping offers many advantages. A significant one is a clear separation of concerns and organization of the data access layer. The "scatter SQL everywhere" antipattern has wreaked havoc on many a poor developer trying to maintain an application. Having an explicit, manageable location for any code accessing the database is a huge plus.

Another advantage is that application developers have a far simpler task when they can access database entities as they would any other object. Code reuse is significantly increased, since the O/R layer handles database connections and performs data conversions.

6 ASSISTED SOA DEVELOPMENT

Some Java IDEs take the approach of *assisting* with common business process and J2EE development tasks. For example, BEA WebLogic Integration (WLI) is a tool that supports Java Process Definitions (JPD) in a J2EE environment. It automates many tasks that developers and web designers must accomplish when developing a business process based application.

6.1 Examples of Assisted Development for SOA

Some examples of assisted development provided by Java IDEs for J2EE and business process development are:

- **New application wizards.** One of the most tedious steps in J2EE process development is just getting started. There are many files to create and they need to be organized in a well-defined directory structure. The application server and database need to be set up and configured. What pieces are needed and how to organize them differ depending on the type of application being built. Getting started can be overwhelming, especially for a novice.

  Java IDEs automate these tasks by providing wizards that step the user through the process and generate initial software artifacts for the application.

- **Graphical process display.** The graphical process display contains "palettes" with various activities that developers can drag and drop onto a flowchart style diagram to create the process. The developer can also access the source code for the process by switching views.

- **Control framework.** Most IDEs contain a control framework for creating various controls. These controls include process controls, Web service controls, Java controls, database controls, JMS controls etc.

- **JSP editor.** Rather than developing a JSP in source form, a JSP page editor allows the user to visually design the page and have the JSP source generated from the design.

- **Page flow designer.** A web application consists of several JSPs linked together. Configuring a web MVC framework using Struts or other similar frameworks is often

tedious and error prone. Java IDEs can automate this by supporting the visual construction of page flows.

- **EJB designer.** J2EE's EJB server-side component model presents developers with many complex tasks. Associated with an EJB is an XML based deployment descriptor. The deployment descriptor can get quite complex. Furthermore, the XML for each EJB must be packaged together into a single XML file. Visual EJB designers generate and maintain the XML descriptors automatically.

  The EJB model also requires a bean class, a remote and/or local component interface and a home interface. There are intricate rules for method signatures defined in each of these classes and interfaces. EJB designers often automate the production and maintenance of these source files. The developer simply views the bean class as the source and the other files are generated and maintained by the EJB design tool.

  Some EJB design tools support mapping objects to database tables for Container Managed Persistence entity beans. A developer visually defines entity beans and connects them together according to the object relationships. Then using a wizard, the developer maps the entity beans and their relationships to database tables and foreign keys.

- **Relational database support.** In order to use a relational database, a J2EE application developer is typically faced with setting up JDBC drivers and writing code to connect to, access and disconnect from a database. This includes code to set up queries and process JDBC result sets. The code maps between SQL types and Java types.

  Java IDEs automate relational database access to differing levels. Some provide full support for object-relational mapping. Others automate pieces of the task. For example, they provide a class that establishes and manages database connections. Similarly, Java IDEs may support the importing database schemas as a way to get started with an object model. They may also support exporting database schemas from a set of Java interfaces or classes.

- **Automatic generation of data transfer objects.** The business tier of a multi-tiered J2EE application receives data from the web tier and produces data for it. So-called *data transfer objects* are serializable Java objects that are passed by value between the tiers. Defining data transfer objects is often quite tedious for a developer. Some Java IDEs can generate data transfer objects based on the application's use cases and/or object model.

- **Automatic generation of business delegates.** Accessing EJBs and external resources from client code is complicated. For example, to access an EJB, the client code needs to look up the home object in the name service, request the EJB from the home object, use the EJB and catch any exceptions. All of this is required just to invoke a method on an EJB. A very popular design pattern is to write so-called *business delegates* to encapsulate these steps on the client side. Some Java IDEs automatically provide business delegates, relieving the developer from writing the business delegate code.

- **Packaging and deployment.** The J2EE model defines several kinds of archives for packaging and deploying related files, including EJB JAR files, WAR files, EAR files and RAR files. Each of these archives has a strictly defined structure so that application servers can find configuration and Java byte code files. Java IDEs assist in the production of these archive files, relieving the developer from writing scripts to produce them. Furthermore, Java IDEs often provide support for deploying these archives to an application server.

- **Automatic generation of Web services.** Producing or using a Web Service by hand is a very tedious and error prone exercise. SOAP messages and WSDL files have to be mapped to Java data types and method calls. Fortunately, Java IDEs are able to automate most of this, allowing the developer to view the problem as a Java one and not have to worry about the XML on-the-wire format. Typically, the Java IDEs can generate a Web Service and WSDL file from a Java interface, class or EJB. To automate Java client code for using a Web Service, Java IDEs can generate Java interfaces from an existing WSDL file.

- **Enhanced Debugger.** Java IDEs assisting J2EE development provide debugging support beyond standard Java debugging. In particular, they provide debuggers that understand the application architecture being constructed using the IDE. For example, they are able to trace the execution across VM boundaries. Without this, developers would only be able to debug clients; server code could not be debugged using the debugger. Furthermore, the debuggers are able to step through non-Java source statements, such as JSP source code, rather than through the generated Java code.

6.2 The Claims as to the Benefits of Assisted Development for SOA

Companies producing assisted development tools make several claims as to the benefits of SOA assisted development:

Developer productivity. By automating the many tedious tasks of business process development, assisted SOA development tools enhance developer productivity. The tools hide many of the details and provide reasonable defaults for the hundreds of configuration parameters involved in a typical application.

Shallower learning curve. Assisted development tools for SOA development lower the bar of entry into SOA programming by hiding complexity and automating many development tasks. This allows inexperienced developers to produce or contribute to a SOA application effort.

Focus on business logic. Since most of the infrastructure code is either abstracted away or generated by the IDE, the software developer can focus on providing correct business logic and not writing the infrastructure code.

6.3 BEA WebLogic Integration and Assisted Development

WLI and WebLogic Workshop (WLWS) provide a visual development environment for process oriented J2EE applications. Many design and implementation tasks are accomplished using visual tools, including a JSP editor, a tool for constructing Struts-based page flows, an EJB designer and a designer for Java and database controls.

6.3.1 JSP Editor and NetUI PageFlow Construction Tool

WebLogic Workshop provides visual utilities to assist in developing JSPs. The tool presents the JSP visually or as JSP source. When the page is presented visually, the developer can drag both HTML and NetUI tags onto the page and configure them in the visual environment. NetUI is a BEA library that defines tags that wrap standard HTML tags and enable JSP pages to submit values to the page flow controller.

WebLogic Workshop also contains a visual environment for creating and designing web page flows. The page flows are based on the Struts Web application programming model. As noted above in Section 5.3, Struts implements the familiar Model-View-Controller pattern for web applications. XML data that associates pages, actions and Java Beans together into page flows drives the Struts controller class. In practice, the XML data can become quite complicated to develop and maintain. Workshop's visual page flow environment automates the generation and maintenance of the XML.

Using Workshop, the developer can define pages, actions and form beans. These items can then be linked together visually according to the desired flow of the web application. From an action, Workshop can generate a JSP containing the fields defined in the associated form bean. The developer can then create another JSP and connect it to the generated page by drawing a line from the action. This page will be invoked by the action. The developer can edit and customize both of these pages in the JSP editor.

6.3.2 Java Controls

Java Controls are components that simplify J2EE development for clients. They serve as business delegates and operate on and/or return data transfer objects. Typically, they are used to encapsulate access to external resources, including EJBs, databases and other data sources. Java controls have well defined policies and architecture, allowing consistent access and management of these resources.

6.3.3 Database Controls

Database controls are Java controls that greatly simplify access to relational databases. Developers define a set of method signatures and associate a single parameterized SQL query with each. WLWS manages the JDBC connection and maps result sets to developer defined Java record classes. The developer does not write any code that uses the JDBC API.

The code using the database control simply calls the method defined on the database control. WLWS automatically provides an implementation of the method that performs the associated query and maps results to the Java record classes.

6.3.4 Packaging and Deployment

WLWS automatically packages the many pieces of a J2EE application into JAR, WAR and EAR files. As part of the build process, these archives are automatically deployed to a running WebLogic Server.

6.3.5 Web Services

WLWS automates the use and production of Web Services, allowing the developer to view the problem as a Java one and not have to worry about the XML on-the-wire format. WLWS can generate a Web Service and a WSDL file from a Java interface, class or EJB.

To enable Java client code to invoke a Web Service using a familiar Java interface, WLWS can generate Java interfaces from an existing WSDL file.

6.4 IBM WebSphere Integration Assisted Development

IBM provides a visual development environment for business processes integrated with J2EE in their WSADIE platform. Development of all items specified within J2EE such as servlets, JSP pages, EJBs, etc. are all supported.

6.4.1 Standard BPEL Architecture

The IBM WSADIE is compliant with BPEL4WS standards; therefore processes developed using this product are portable to other BPEL platforms. Developers having used other BPEL products will rapidly recognize the invoke, switch, and assign activities along with the other standard BPEL definitions.

6.4.2 End-to-End Eclipse Based IDE

An extended version of the open source Eclipse editor is used for all levels of development. Whether developing back-end business processes or front-end JSP pages, this is all accomplished using Eclipse with accompanying plug-ins. Integration of all types of J2EE projects is provided out of the box by the platform.

6.4.3 Web Services

IBM provides generation of two kinds of Web services, of which one is WS-I based. The WS-I based implementations are IBM Web services that are invoked using WSIF (Web Services Invocation Framework). The other possibility is generating Apache Web services. Each of these Web services can be generated in two different ways. Either the user starts off by creating a service interface that is used to generate the service skeleton, or the service definition is produced based on an existing implementation provided as input. The service can be implemented as either a Java service or an EJB service.

In case one does not want to use the SOAP protocol, WSADIE offers other binding possibilities, such as JMS and RMI/IIOP.

6.4.4 Standard JSP/Struts Support

JSP support is provided using the Eclipse editor. There is a graphical layout facility for JSP pages as well as an assisted IDE environment for editing the source code directly. Struts support is included via integration into dynamic web projects and templates available for creating Struts actions, form beans and mappings. There is also a graphical view created with the web project that allows the developer to organize the page flow.

6.4.5 Robust Debugging Environment

The debugging environment has many features, one of which is remote debugging. Developers can create a server and a server configuration, set breakpoints in their project, publish their project to the server and start the server in debug mode. A developer can then step through the code and see the standard display of watches, etc. For BPEL processes, breakpoints can be set on the actual graphical process itself. The debugger can then be used to investigate problems within the process.

6.5 Oracle JDeveloper/BPEL Assisted Development

Oracle's assisted development for SOA centers on JDeveloper and the Oracle BPEL platform. JDeveloper is an IDE for developing end-to-end J2EE applications. It provides all the standard features of a J2EE development environment. It also provides Oracle's Application Development Framework (ADF).

There are two facets to the Oracle BPEL platform, the BPEL Designer and the BPEL Process Manager. The BPEL Designer is available as an Eclipse plugin that provides a visual as well as textual environment for building business processes in BPEL. The BPEL Process Manager provides the infrastructure for deploying and running BPEL processes. In addition to an execution environment it also provides monitoring features.

6.5.1 Persistence Layer

For handling persistence, JDeveloper includes Oracle's O/R mapping tool, Toplink. Toplink is a data persistence architecture that enables developers to map a database to a set of Java objects in the application tier. As noted above in Section 5.4, one fundamental goal of such an architecture is to separate the concerns of the database structure from the application logic. The application developer can work with objects that are transparently persisted and retrieved rather than having to manage both the SQL code and the application code. Because the tool generates this object layer, the development effort is significantly reduced. The Toplink persistence framework provides object-relational mapping for any combination of database, application server and toolset environment.

Toplink has a graphical workbench that allows management of the mappings between the object and relational layers. It also has a query framework that provides various mechanisms for querying persistent data such as SQL, EJB QL and Java expressions.

6.5.2 Standard BPEL Architecture

The Oracle BPEL Designer is compliant with the BPEL4WS standard and provides the standard invoke, switch and assign BPEL definitions. The BPEL Designer has a feature whereby any BPEL processes it generates are completely portable to other BPEL servers. The BPEL source is also completely accessible to the developer, and modification to the BPEL source does not alter the graphical capability.

The features supported by the BPEL Designer include:

- Drag-and-drop process modeler
- UDDI and WSIL service browser
- Visual XPATH editor
- One-click build and deploy

The Oracle BPEL Process Manager is the infrastructure for deploying, executing and managing BPEL processes. It supports the BPEL 1.1 version, context dehydration, both synchronous and asynchronous messaging, advanced exception management, side-by-side versioning and large XML documents.

6.5.3 Oracle Application Developer Framework (ADF)

Oracle ADF is a productivity enhancement solution provided to promote rapid development of J2EE applications. ADF supports using EJBs, Java classes, Web Services and Java objects as business services. In addition, pre built ADF components are available.

Productivity is achieved by the use of metadata-driven ADF components, which are based on J2EE design patterns. ADF components are available for four layers of a typical J2EE application.

Business services: The ADF Business Components technology provides declarative building-blocks for implementing scalable business services, data access objects, and business objects that enforce business rules and handle database persistence

Model: The ADF Model layer provides declarative data-binding against multiple backend technologies, accommodating business services implemented as ADF Application Modules, custom JavaBeans, EJBs, or Web Services. This binding is an implementation of JSR 227 a common API and meta-data format that will offer a standard declarative way to bind data from a business service, such as Web services, EJB, Java, JCA, JDBC, to other entities such as UI components for example Swing component or JSP/JSF Web component

View: ADF Faces is a user interface framework based on Java Server Faces (JSF). It provides a rich set of view components with XML metadata configuration. Many of these components are the building blocks for common Web interfaces such as master-detail forms. In this project we used the Java Server Pages(JSP) technology.

Controller: ADF integrates with Struts, while at the same time providing additional functionality by subclassing the standard Struts Action class.

6.5.3.1 Data Controls

In JDeveloper, data controls can be generated to use Web services, EJB components, Java beans or Toplink POJOs. Along with the ADF Model (binding) layer, these controls provide a uniform and standard data access mechanism as described in JSR 227.

6.5.3.2 Building Page Flows with JDeveloper and ADF

JDeveloper's visual environment for building page flows is Struts compliant. The data controls and the data binding layer constitute the data modeling layer. The controller handles the page flow for the web application. The page flow consists of Data Actions and Data Pages.

Oracle ADF provides Data Action classes to represent page operations within the page flow. Data Action is a subclass of the Struts Action class.

Data Pages are comprised of a DataForwardAction and a JSP. The DataForwardAction implements the "postback" pattern, in which a single action instance includes both page initialization and page event handling logic. In other words, the postback pattern is a design pattern that recommends implementing a single action class for both initializing a form page as well as handling its submission, to eliminate redundant page logic.

All the database operations such as insert, update, commit and rollback are exposed by the data controls for the persistence layer. Within JDeveloper, these operations can be either dropped as buttons on the Data Page or dropped as methods on Data Actions.

6.5.4 Web Services

Using the implementation of an ADF business component as input the Web service generation feature auto generates the WSDL and other artifacts needed to deploy and run a Web service. The result is a document literal Web service. ADF can generate Web Services from Java Classes, EJBs, JMS and even PL/SQL.

6.5.5 Debugging Environment

The debugger permits adding breakpoints to the project code and then running it in the target container in the debug mode. Once this is done, one can step through the code and troubleshoot by looking at the values of variables in the watches display. To monitor network traffic based on the content in the HTTP or TCP packets, JDeveloper provides a TCP packet monitoring facility. For unit testing any of the Toplink POJOs or EJBs a test client skeleton can be generated automatically from the business component. After including the requisite code to call the appropriate methods, the client is now ready for unit testing the components.

7 STUDY DESCRIPTION

This study compared the productivity of three different approaches to developing a SOA application:

- The BPEL compliant based approach embodied by IBM's WebSphere Studio for Application Development Integration Edition (WSADIE)
- The assisted J2EE development approach embodied by BEA's WebLogic Integration product
- The Oracle BPEL, ADF, Toplink development platform

The study centered on two developers building the same Java application with each tool from a common specification.

It is important to note that we did not create separate teams for this study. The same two developers used the three platforms. For comparison, we contrasted their results and experiences with each.

7.1 About the Specification

The functional specification used for this productivity study is *The Middleware Company SOA Application Development Specification*. It is a 48-page functional specification that describes in detail the requirements for an enterprise application, from database schema to web user interface. Using this specification ensured that participants in this study would build comparable applications. You can download the specification from our web site at, http://www.middlewareresearch.com/endeavors/040908SOADEVELOPMENT/endeavor.jsp

The Middleware Company created the specification with the help of a distinguished panel of experts, including book authors, open source contributors, architects and analysts and interested critics of prior studies conducted by The Middleware Company. The expert group members included:

> Abe White (Chief Architect, Solarmetric), Clinton Begin (Author, Open Source JPetStore), Rob Castaneda (Author, CEO, CustomWare Asia Pacific), John Cheesman (Architect, 7Irene), David Herst (The Middleware Company), James Holmes (Struts author), Rod Johnson (Author and Spring pioneer), Howard Lewis Ship (Creator of Tapestry), Bruce Martin (The Middleware Company), Steve Wilkes (The Middleware Company), Mike Sawicki (Architect, Compuware).

The functional specification is based upon the *SOA Blueprints* specification authored by Stephen Wilkes of The Middleware Company. Note that the definition of SOA is still up for debate. However, the *SOA Blueprints* has received significant support from major vendors such as Oracle, BEA and Microsoft and so it represents the state of the art in SOA requirements..

7.2 Choice of Application

The Expense Reports section of the *SOA Blueprints Specification* was chosen for the study since it illustrates business process, data access and user interface. This section defines a web based management application for employee expense reports. The application resembles the corresponding module of popular ERP systems. The functionality includes:

- **User management and security.** Users can sign into the system and manage their account.
- **Business process backend.** Using the UI, users invoke business processes to create and maintain expense reports.
- **Task assignment.** When expense reports are submitted or approved by the manager, they are assigned to the worklist of managers or accountants for approval.
- **Task management.** Users with the role of manager and accountant can approve tasks.
- **Report creation.** Users can create expense reports with the application and either save or submit them.
- **Web services.** Since SOA does not mandate the use of Web services, they are not required. However, there is ample opportunity for a vendor developer to utilize Web services for this implementation.

From a SOA technology perspective, the application includes the following:

- A thin client HTML UI layer
- JSPs to generate HTML on the server
- Front controller middle-tier
- JDBC SQL-based data access with opportunity for O/R mapping enhancement
- Business process workflow
- Worklist implementation
- Data caching
- User/Web session management
- Web services
- Forms-based authentication

7.3 Development Requirements

The specification lays out in detail the rules for building the application. It describes, for example, what data can be cached, database exclusivity rules, requirements for forms-based

authentication and session state rules. It also describes in detail the required experience of using the application.

The requirements for this study are designed to maximize the ability to compare diverse development environments and to factor out extraneous aspects of the comparison. This focuses the on the productivity of tool. In brief:

- The team was required to use the database schema as spelled out in the specification.
- The spec did not mandate any implementation details, including choice of any implementation patterns – those decisions were left up to the developers. What mattered was that the resulting applications behave similarly.
- For the Oracle implementation, the use of Toplink for O/R mapping was mandatory. The other products did not offer any O/R mapping tools.
- Some of the processes could be implemented without the use of the process engine. As long as the functionality remained intact, the best practices in the tool were applied.

7.4 Overview of Testing Process

To ensure that the final applications behaved similarly, continuous testing was performed using the test environments provided by the vendors.

The test scenarios were standard use cases such as logging in, creating an expense report and submitting it.

All three applications passed the tests.

7.5 Overview of Result Collection Process

During the implementation phase, both the developers followed certain rules while accounting for timing results. They did not include:

- Ramp up time spent in getting familiarized with the product
- Time spent on figuring out the best combination of tools within a product to implement the expense reporting application
- Time spent implementing certain pieces of the application which were later on replaced by using more productive techniques or tools

7.6 Overview of the Developers

Two J2EE architects from The Middleware Company (TMC) participated in the study. Both had significant experience with J2EE development on a variety of application servers. As far as their prior experience with the tools in question:

- One developer had prior experience and training with WLI; the other had none.
- Both had experience with prior IBM integration technologies, but neither had experience with the WSADIE tool that was used.
- Neither developer had any experience with the Oracle BPEL Product, JDeveloper or ADF.

Both developers spent several days learning the relevant features of these products. None of this time spent on training counted toward the total.

7.7 Overview of Project Schedule and Project Management Approach

The project commenced with two people on the team. Both these developers worked on the BEA implementation. They synchronized their work using a source code control system and conducted regular meetings to update their notes about the implementation. This team then

broke up to work individually on the IBM and Oracle pieces. Each of these developers maintained a detailed account of every step taken during development. The remaining sections of the report are based on their individual as well as collective experiences with the products.

7.8 Study Generalizability

Studies of this type involve some degree of variability. Ideally, the study would involve many replications over a larger sample size of developers. Because of resource constraints, this proves to be prohibitive. The study is intended to represent our experience using the toolsets mentioned, and is represented here as the best available information about the productivity of these tools from the perspective of The Middleware Company. In accordance with our stated policy, we only publish what we believe. The intention is to provide a structured basis for comparison, and empirical data to support conclusions about the tools. We encourage readers of this report to replicate the study in order to draw their own conclusions.

Also, results of this particular study cannot be generalized for every SOA scenario. The conclusions drawn are based on the results and the experiences of the developers with each of the products in implementing the expense reporting application. Developers building applications similar to the expense reporting application in terms of functionality, design and size, will find these results applicable.

8 STUDY RESULTS

In this section, we discuss the results of the study. The results are organized into the following upcoming sections:

- The **Architectural Analysis** section describes the architectures used.
- The **Qualitative Results** section summarizes each developer's qualitative thoughts on the development approach they chose – the issues encountered and how they resolved those issues.
- The **Quantitative Results** section presents the final productivity result numbers for each implementation.
- The section **Factors that Affected Productivity** gives a detailed breakdown of the reasons for the difference in productivity.
- The section **Qualitative Overview** reflects the thoughts of the developers about their overall qualitative experience with the products which might not have been evident in the previous sections.

8.1 Architectural Analysis

8.1.1 Design of the Application Back End

Since the application was well defined by the specification, no additional modeling activities were performed. The developers identified these core tasks:

- Login Process
- User Profile Information
- Get Reports
- Get Report
- Submit Report
- Authorize Report
- Save Report
- Get Item Types
- Miscellaneous other processes

Since all tools provided a graphical process design facility, the processes were all designed in this interface and the screens were designed in the JSP editors.

8.1.2 Design of the Application User Interface

Again, the specification provided the bulk of the design for this portion. One available design decision is which functionality could be implemented using a front controller communicating with an O/R layer rather than a process. The following pages comprised the core of the user interface:

- Login
- Expense Reports (Main)
- My Reports
- View Report
- Create Report
- Staff Reports
- Authorize Report
- Choose Transactions
- Update Report
- Add/Edit Line Item
- Authorize Line Item

The page flow designed for these pages is shown in Figure 17 of the specification.

8.1.3 Overall Shape of the WLI Code

The code produced by the team for WebLogic Integration was very much dictated by the WebLogic Workshop framework. It includes a series of JSPs. The JSP pages were created using the JSP editor. WebLogic Workshop provides a tag library called “NetUI” and graphical environment that generates skeletons of Struts based actions and form beans in the page flow controller (JPF file).

These actions requested business operations from process controls. The process controls executed the processes as defined in the specification. Database controls were used to provide the database interface to the processes. Email controls were used to perform the email notification required by the Submit Report and Authorize Report processes.

In general, the development process using WLI was to first develop the JPD process along with any necessary database control code. Then we would develop the JSP page. Finally, we would add the binding action method(s) and form bean(s) into the page flow controller.

For the special Authorize Report and Submit Report processes that involved task assignment, WebLogic offered task controls and a section of the WLI administration console for task administration. A task control could be dragged onto the graphical process diagram and this would specify a step in the flow to create or assign a task. Assigned tasks could be viewed and their state changed in the WLI administration console. Most users in practice build some custom interface to these tasks and even use some LDAP interface to expose all the users in the company but the console was adequate for this study.

For WLI the developers decided not to define any EJB components. Many of the services associated with EJB containers, such as automatic transactions, were already provided by the WLI runtime for Java and database controls.

8.1.4 Overall Shape of the WSADIE Code

The IBM WSADIE platform mandated some differences in approach. The IDE is Eclipse combined with some custom plug-ins such as the graphical BPEL Designer. The open source version of Struts is used with the Eclipse interface and Tomcat is used for the servlet engine. There is a JSP editor that is an open source Eclipse plug-in.

Since WSADIE has no data controls analogous to the ones in BEA and Oracle, the services, which needed access to the database, used standard JDBC. A Java helper package was written to eliminate repeating JDBC code for every database operation performed. With the use of these helper classes one could provide the SQL statement as an input and receive the result set, which in turn could be used to populate an XML instance. Due to the absence of Java controls, IBM Web services were used. The input and output message schemas were defined, based on the schemas provided in the specification, using WSADIE's graphical schema editor. These services were invoked from Struts actions. Besides the asynchronous processes, submission and authorization of expense reports, all the other processes were implemented as Web services. The asynchronous processes were constructed using the graphical BPEL designer.

The steps required in order to build a BPEL process were as follows:

1. Created a service project with the IDE. This created an empty WSDL file.
2. Edited the WSDL file specifying port types and messages. For receive and return schemas specified in Expense Reports import these types and use them for the types of the message parts.
3. Used the IDE to create a BPEL process.
4. Created partner links and variables for the process, this used the information specified in the WSDL file.
5. Modified receive and reply activities created with the BPEL process. Used receive and reply variables created in the last step for entry and exit values.
6. Filled out the remainder of the BPEL process using invokes, Java Snippets, assigns, switches, etc.
7. Configured an IBM test server then deployed and tested the process.

The BPEL processes were defined by Web services WSDL files and thus could be invoked via Web services. The simplest approach seemed to be developing the JSP and Struts pageflow using the Eclipse editor, then, within the action, calling a Web service to invoke the backend BPEL process.

8.1.5 Overall Shape of the Oracle Code

The implementation on the Oracle platform was significantly different. The developer approached the problem from the bottom up, starting with the generation of Toplink Plain Old Java Objects (POJOs). To extract relevant data based on different queries, he added named queries to the POJOs. Dragging and dropping these POJOs on top of the data control palette automatically generated the data controls. These data controls were used as data providers for the presentation layer.

The presentation layer was built using the Struts framework and ADF bindings. The Data Pages for the report creation and submission processes did not invoke any BPEL processes or Web services since the JDeveloper data controls offer Create, Retrieve, Update, Delete (CRUD) operations. The data and the methods from the data controls are available to the presentation layer only through the binding layer. The creation and modification of the report

was implemented using the Struts framework, ADF and Toplink technologies in keeping with Oracle's best practices. The Authorize Report process performs additional business logic other than database operations, hence it was the only process that was implemented using BPEL. This process is an asynchronous process since it involves a user task. A user task is an asynchronous service, which waits for input from a human. The web pages and control logic for invoking and completing this process were developed using ADF components and the Java API offered by the BPEL server.

The steps involved in building the process were as follows:

1. Created a project for building an Asynchronous/Synchronous BPEL process
2. Introduced complex types defining the input and output message schemas for the process
3. Created the required partner links and global variables for the process.
4. Used the IDE to build the process with assigns, Java embedding, user tasks, conditional switches etc
5. Built the JSPs for the user task using the Worklist API. If these JSPs were deployed in another container outside the BPEL environment the relevant JNDI properties would have to be to be set for RMI. Created an individual page flow to test the invocation and execution of the business process
6. Started the BPEL server. Unit tested the BPEL process by invoking it from the BPEL console. Checked the messages on the wire using the TCP tunneling feature provided with Apache Axis.

A page flow already existed for the creation and modification of reports.This page flow along with the new page flow for the authorization functionality were then integrated.

8.2 Qualitative results

In this section, we'll review the developers' qualitative thoughts, summarized directly from their periodic status updates. Qualitative results are based on a small sample size and a low number of observations. They are intended to represent an audited version of the developer's notes and experiences. These are included here expressly to help readers of this research understand the nuances of the quantitative result, and the variances in the approaches and capabilities of the compared tools.

8.2.1 WLI Development

This section contains a description of the WLI development of the Expense Reports application.

8.2.1.1 WLI Development – Early Stages

The developers decided to first do a *vertical slice* of the application. That is, they implemented a single use case completely, from the business process tier to the database. Getting the list of reports was chosen as this initial slice. Producing the vertical slice went fairly smoothly and this effort got the team started setting up source code control system, the database, connection pools and data sources. Since one developer had no experience with the WLI product, there was some ramp up required on that end. Once the vertical slice was finished, the two developers then divided the use cases between them, with each working on his independently.

As soon as one of the developers had completed building his set of use cases, the two decided to split the implementation effort horizontally. The reason why both developers did not work simultaneously on the presentation layer was that the single controller configuration file would become a bottleneck. Both developers would have been constantly modifying this XML file and the merge process would have been complicated. One developer had training and experience

with the process side of WebLogic Integration, which made him best suited to continue work on the processes.

While one developer worked on the presentation layer the other developer continued to work with the asynchronous business processes that involved human task activities.

8.2.1.2 WLI Development – Later Stages

Defining the interface between the web tier and the presentation tier enabled the developers to work independently and concurrently. This worked fairly well. The primary activity that required interaction was bug fixes in the JPDs that impeded the front end development.

The developer implementing the JSPs started hooking them up to the stub implementations of the business tier. Concurrently, the developer working on the business tier began implementing them completely. As the stubs were replaced with completed implementations, the application began functioning.

The developer implementing the business tier was able to test his code without all of the JSPs because WLI has a test environment for JPD processes. There is a form generated where the developer can input a SOAP request and invoke the JPD. A stack trace style monitoring output is produced. This proved to be an effective approach to testing of business functionality.

One area of difficulty at the JPD level was the implementation of the task assignment. The specification indicated that the check cut process should remove a task from the list. The programmatic interface to task manipulation seemed present but appeared complex and, due to time constraints, this was left to the WLI admin console, which has this capability.

The developer implementing the page flow on the front end found the process difficult in that the page flow itself was very large and the reusability of actions and form beans was limited. Also, without a WYSIWYG JSP editor, laying out all of the JSP pages was cumbersome.

8.2.1.3 WLI Development – Finishing Up

The team was able to produce good portions of the functionality in fairly short order. However, the implementation was incomplete and had various bugs.The team anticipated that the debugging period would not be that long, but in the end it took several days to fix them. Much of the time taken on the extended effort was due to the slow code-compile-deploy-test cycle.

During the last phase, the team realized it had overlooked a significant piece of functionality required by the specification – the Web service to query an order. WLWS excels in the production of Web services. A new Java interface was defined for returning an order and a simple query was defined on the database control to query the database. Then, by right-clicking on the Java interface, WLWS generated a complete Web services, WSDL file and test form. The entire effort to add and test the Web service took about an hour.

8.2.2 IBM WSADIE Development

This section contains a description of the WSADIE development of the Expense Reports application.

8.2.2.1 WSADIE Development – Early Stages

The early stages of the WSADIE development consisted of installing the tool, configuring the environment and starting with an end to end use case. Configuration of the data source took a while and was complicated compared to the experiences with JDeveloper and WLI. The first task chosen was to implement GetItemsList, a service that simply retrieves the list of item types that are predefined as seed data. The service was implemented as a Java service and deployed as a stateless session bean. The service proxy generated for the service was used in the Struts action to invoke the necessary service operation. A simple page flow was built and tested to invoke this service and display the results in a JSP. Once the developer had an end to

end experience of working with one slice of the application he moved on to working with the other vertical slices.

The front end JSP/Struts layer was similar to BEA's but closer to the actual open source versions of these tools. The development and testing at this level was difficult. WSADIE has a slower build and deploy process compared to WLI and JDeveloper. There is a particularly slow publishing phase that took as long as 2 minutes. Restarting the server was required frequently as well.

During this phase the developer faced a small hiccup in the development process. One of the message schemas had a complex element type whose name matched that of another complex element type in the same namespace. The generated bean overwrote the existing bean which was already in use by a service implementation. As a result the developer had to regenerate the services after altering the schemas.

8.2.2.2 WSADIE Development – Later Stages

The development of the BPEL processes was difficult at first but then became easier. Deletion of business processes was problematic. On deleting a BPEL process in the package view, WSADIE failed to remove all of the references in the corresponding EJB project. The developer had to manually edit the EJB deployment descriptor and look out for other artifacts that may not have been deleted.

The development of the front end JSP/Struts code was akin to using the corresponding open source products with Eclipse plug-ins. The developer created a dynamic web project with Struts enabled. There is a file generated where the page flow can be specified and this acts as a springboard for creating the JSP pages. Then the action skeletons can be generated by the IDE and the developer codes the behavior. For the asynchronous processes, the approach used was to generate a Web service using the JMS binding WSDL file generated with the BPEL process and invoke it in the Struts action as with the BEA case. It could also have been possible to use a J2EE project based integration of the two but the team decided the Web service would be faster.

8.2.2.3 WSADIE Development – Finishing Up

Tasks that took longer were those that involved debugging the business processes and those that occurred once the projects had grown in size. The front end was faster due to the fact that server restarts were not as frequent. A considerable amount of time was spent in writing data transformation code, which meant copying values of member variables from an action form object to an XML bean and vice versa.

8.2.3 Oracle Development

This section contains a description of the Oracle development phase.

8.2.3.1 Oracle Development – Early Stages

A single developer was involved in this phase. The work began with setting up the database by using the predefined schema and adding some test data to the tables. The idea was to deal with each layer of the application one at a time rather than taking the vertical slice approach. After setting up the environment by configuring the database and application server connections, the developer commenced working on the persistence layer. The mappings from the database tables to the Toplink POJOs were automatically generated. After unit testing the POJOs, he moved on to develop the business services. After getting back on track, the developer incorporated Named Queries in the Toplink POJOs to return the data which was expected from the BPEL processes. This was done with ease and the developer moved on to creating the Web pages and the web application's control logic. By dragging and dropping the Toplink POJOs onto the data control palette, he automatically generated the data controls. All the participating components were unit tested. Unit test clients are generated by a single click,

which greatly simplifies the testing process. The odd issue with the Oracle single click and deploy process is that, despite declaring dependencies for the project, one needs to manually add the classes to the deployment profile.

It took a while for the developer to understand the new constructs of Data Action and Data Page in relation to Struts as well as updating his knowledge on JSTL.The time spent on learning these new technologies were not counted.

8.2.3.2 Oracle Development – Late Stages

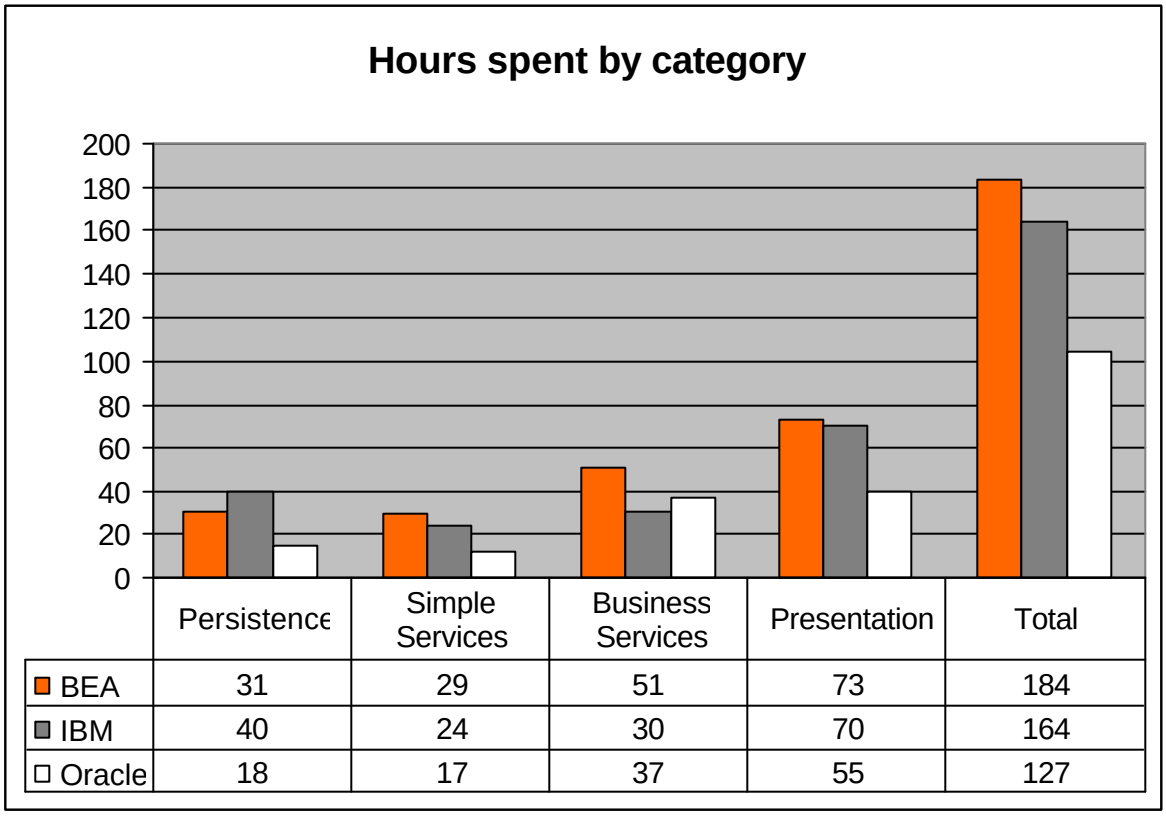
During the last week of the implementation phase, the Authorize Report Process was created using the BPEL Designer. At this point the developer had to deal with issues of serialization related to RMI. The BPEL process was built and tested individually by invoking it from the BPEL console. The related Web pages and control logic were created in the JDeveloper environment using ADF components. This ADF page flow was kept separate from the main page flow of the application for testing purposes.

8.2.3.3 Oracle Development Finishing Up

The last few days were spent integrating the Authorize Report page flow with the application page flow. To integrate these two flows, an additional action was needed to update the status column of the expensereport table after committing the expense report to the database. This was addressed by a session façade bean with a method to provide this functionality. A data control for this session bean was autogenerated and a Data Action to execute this method was added to the page flow. The findForward() method for the same Data Action was overridden to invoke the Authorize Report business process. The remaining Data Pages from the individual page flow were then directly integrated. Other necessary JSPs were added as defined by the specification.

8.3 Quantitative Results

Here is the final tally of development hours spent to develop ExpenseReports to spec with each product:



	Persistence	Simple Services	Business Services	Presentation	Total
BEA	31	29	51	73	184
IBM	40	24	30	70	164
Oracle	18	17	37	55	127

The results in this table comply with The Middleware Company process for analyzing developer productivity and thus does not include a number of special circumstances including time spent learning the product, and any cases of design error which can be attributed to the practitioner, not the tool itself. All of the hours spent are thoroughly documented and audited by The Middleware Company based on these processes.. Factors that Affected Productivity with Oracle

8.4 Factors that Affected Productivity with WLI

The developers identified some factors that impacted their productivity in either a positive or negative fashion from the WLI experience. In addition, they decided to discount some of the logged time spent due to some lack of experience with the platform.

8.4.1 Java Process Definitions (JPD)

The environment for developing JPDs had both assets and liabilities. The team found the control features to be very productive since the controls assisted and organized any behavior they wished to include in the process. One could create any type of control, include it in the project tree then drag it onto the process flow.

At times, the graphical interface would lock up and the user would not be allowed to modify names and other items. This was usually resolved by closing and reopening the application but was time consuming.

The code behind the process flow was accessible but the user had to be very careful in modifying this code. If the code was modified in the wrong area the portions of the graphical display would become disabled. Also, the team noticed that deleting graphical objects on the

flow did not necessarily delete all of the source code associated with the object. They did not observe any functionality impact but “dead code” was created.

8.4.2 XML Beans

XML Beans are an integral part of the WLI platform. When an XML schema is imported into the application, the schema is validated and a bean class definition is generated. Instances of the bean hold the XML documents.

To facilitate the processes, the developers created XML schemas representing the process input and output then XML beans were generated for each schema. Within the graphical flow environment, the XML bean could be specified as process input and used via the XML bean API.

There are obvious productivity enhancements from XML beans. They certainly enable the developer to avoid some troublesome XML effort. However, the support for XML beans from the WLI platform could be improved.

8.4.3 Database Access

The database control in WLWS was felt to enhance productively greatly. The simple facility allows developers to associate SQL queries with method signatures and automatically map the result sets to Java record classes. WLWS automatically manages the JDBC connection and implements all of the mappings between SQL and Java types. The developer does write any code using the JDBC API.

However, the database control really just represented a mapping from some methods in a generated class to a SQL statement. There was no logical grouping of attributes and behavior as in the case of a true O/R mapping tool. So as the project grew, the management of these controls became more difficult.

8.4.4 Debugging

The developers felt that the server side debugging environment was well designed from an interface standpoint but very slow. The debugger is able to attach to JSP, servlets, Java controls and JPD code running in WLI. There were the standard breakpoint and stepping operations available to the developer. However, the time required for the developer to modify, build, deploy and debug code could be lengthy especially as the project grew in size. The developers measured the deployment into the debugger as taking three minutes or more. There is a negative effect on the concentration of the developer as well as the delay itself.

8.4.5 Web Services

The support for automatic generation of Web Services is excellent in WLWS. It completely hid all of the complexity of implementing a Web Service. By simply right-clicking on the Java interface, a Web Service and the WSDL file were generated to expose a JPD process. There was also a straightforward clicking approach to generate a Web service to test a database control.

8.4.6 Code-compile-package-deploy-test cycle

The code-compile-package-deploy-test cycle is also quite heavyweight in WLWS. As the application implementation became more complete, compiling, packaging and deploying took several minutes each time changes were made. As mentioned above, the lack of rapid feedback especially slowed down the debugging phase of the project.

8.4.7 WLI Stability issues

WLI crashed on a number of occasions. This usually occurred after all the process controls were created and the size of the project was at a maximum. Restarting WLI always solved the problem and the developers never lost any data due to WLI stability issues.

The developers identified this as a significant factor that was not counted in the total development time.

8.4.8 Learning Curve

One developer had no prior experience with WebLogic Workshop or Integration. Prior to beginning the Expense Reports implementation, this member spent a few days learning to use WLI using the samples provided in the distribution. But when the implementation effort got started, there were still features of WLI that were not fully appreciated, especially to support the JPD process tier. During the implementation effort, some time was taken to learn these features in more detail. This time spent on learning the features was not included in the total.

8.5 Factors that Affected Productivity with WSADIE

The developers identified some factors that impacted their productivity in either a positive or negative fashion from the IBM WSADIE experience..

8.5.1 Database Support

Database support with the product was virtually nonexistent. One could use CMP entity beans to form a persistence layer but that would overcomplicate matters. There was also some suggestion in the IBM documentation that Hibernate could be used but this seemed to be introducing yet another open source component when there were already many embedded in the product itself.

8.5.2 BPEL Designer

An issue with the BPEL environment in general was that the business processes were implemented via a J2EE mapping. This is similar to the approach used by BEA. An issue this can cause is that error and warning messages are given to the developer based upon this J2EE code and not the actual BPEL code the developer produced. This layer of abstraction can make the resolution of programming errors very difficult. One particular instance occurred when a developer was trying to resolve some issues in a process and there were some warning messages about some of the deployment descriptors for this generated J2EE EAR file. The messages turned out to be harmless but served as a misdirection and cost of time in trying to resolve the overall issue.

At the very beginning of the BPEL implementation process the embedded messaging system failed to function in the desired way. The JMS broker would fail to start up and would complain about a missing embedded messaging installation. This was confusing to the developer and he spent some time searching through the WSADIE forums for an answer to this problem. After performing an extensive search, the developer discovered some IBM documentation that pointed out the error. The error was due to a missing variable definition in the server configuration that was supposed to point to the messaging libraries.

The controls provided by BEA were significantly more productive. The alternatives available in the IBM environment were Java snippets and external services. The external services were more complex to develop and integrate than the BEA controls. The snippet itself was similar to a Java method.

8.5.3 WebSphere Integration Server

The deployment to the server was cumbersome. The process was called “publishing” and could be slow depending upon the size of the process. Also, almost any change to the BPEL process

required both republishing and restarting the server. There also seemed to be no way to disable certain processes from being built and deployed. All of them would be contained in the EAR file that was published every time.

8.5.4 JSP

The JSP development environment was similar to that of BEA. There was no WSIWYG freeform editing so the developer had to work with a grid that controls the component placement. Since this application doesn't have complex page graphic requirements this didn't cause too much delay.

8.5.5 Struts

The Struts support was in line with the Eclipse plug in. The issue of slow build, deploy and run of the test server decreased productivity significantly. One area of improvement versus the BPEL process side was that server restarts weren't required. The facility for creating action classes was reasonable, although there were minor areas where WLI was more convenient. For example, when creating the action, if there was no form bean created the IDE did not give the developer the option to create one as in the case with WLI. As compared to the Struts page flow implementation in JDeveloper, the Struts page flow in WSADIE took much longer to implement and involved more manual coding.

8.6 Factors that affected Productivity with Oracle JDeveloper/BPEL

A single developer was involved in the Oracle implementation. The factors that impacted productivity as a result of inexperience with the Oracle platform have been excluded. There were a number of factors that boosted productivity and a few that hampered it.

8.6.1 Persistence Layer

The highlight of this implementation was the persistence layer. The O/R mapping along with the inclusion of the named queries was completed in a couple of hours. As the implementation progressed the developer realized the need for additional queries and modifications to the POJOs. JDeveloper's visual expression builder greatly simplified the generation of named queries.

The sequencing feature in TopLink automatically generates a sequential number for columns associated with this feature using a custom sequence table. Use of the sequencing feature eliminated the need to write stored procedures or JDBC code to provide the same functionality. Sequencing was utilized for generating values for primary keys of tables whose values were not dependent on user input.

UnitOfWork objects hide the database transactional details from the developer. The developer just focused on utilizing the POJO getter setter methods to construct and modify the associated persistent objects. The ability to encapsulate these Toplink POJOs within Java controls had a direct affect on the overall shape of the code. As we discussed in section 8.1.5, these data controls performed the same functionality as the business processes for report creation, retrieval and submission. This as a result reduced the complexity of the business services layer and simplified the development process.

8.6.2 BPEL Designer

The Oracle BPEL Designer, unlike the designers in WLI and WSADIE, is not part of the integrated environment. The BPEL processes are deployed in a different instance of the oc4j container from the one that contains the JSPs. This implies that these BPEL processes need to be invoked using RMI by components deployed in any J2EE container. In this case, XML messages needed to be converted to string objects before they were transmitted over the wire. We avoided using serialization for efficiency reasons.

During the implementation of the JSPs that invoked the Authorize Report asynchronous process, the developer ran into a deserialization error for the objects returned by the Java API for retrieving the list of user task objects. After an entire afternoon spent debugging this problem, it turned out that the two container instances (one hosting the JSPs, the other running the BPEL process) need to use identical JVMs.

Some of the other errors the developer ran into include:

- Validation error that pointed to the zeroeth line of the BPEL source. In this case the problem was the absence of any activity on one of the case condition branches of a switch statement. It needs at least an empty activity to validate the process.
- A random ClassCastException that is thrown every few times the process is executed. The reason for this behavior is still unknown.
- To invoke a synchronous document/literal Web service it is necessary to set the sendXSIType property to true in the BPEL.xml file or else it fails. In the case of a one way invocation it does not throw any runtime errors.
- On any change in the WSDL types, the BPEL server needs to be bounced to reflect the changes.

The definite advantage is the standards based BPEL Designer which makes designing processes simple and yet portable to any BPEL compliant server. The monitoring and auditing capabilities are exceptional. The orchestration was done fairly easily. The problems encountered had more to do with the use of RMI to talk to these business processes.

8.6.3 ADF Components

ADF includes a business services layer where business logic can reside for use by ADF components. This implementation, however, did not put any functionality in that layer, except for a session bean to update the status column of the expense report table. Instead, all ADF components talked directly to the persistence layer.

JDeveloper's visual features for generating Web pages and the basic control logic to tie all of the components together were a big advantage. They completely eliminated the need to add any JSP code for accessing data. Using just the ADF's Data Pages, Data Actions and data controls, the developer created the complete page flow.

The learning curve is not steep if the application is not too complex. If one needs to add any code to the Data Actions, then one must understand the page lifecycle and implement the right methods for the Data Action. If the need includes access to data present in the binding context, then understanding the ADF data binding layer as well as JSTL is a must. Understanding the binding layer, including the various bindings within a common binding context, can be challenging. Additionally, in the rare situation where data from one data control must be assigned to data in another data control within a page's context, the UnitOfWork for both controls need to be merged.

During the implementation, there were a couple of issues with ADF that hindered development.

- To create a form using the data bindings from a data control, one should always drag the entire set of data returned from the data control onto the Web page as an input form, rather than dragging individual elements as input fields. When doing the latter, the developer experienced binding and form submission related problems that returned unexpected results.
- On a couple of occasions, after changing the classpath the developer had to restart JDeveloper to see the changes.

8.7 Conclusion

We found factors that contributed to this improved productivity in the implementation of each layer of the expense reporting application:

- **Persistence Layer.** Only Oracle offered an O/R mapping tool (Toplink), significantly easing the representation of data as Java objects.

- **Simple Services.** These services were mainly database related and involved CRUD operations. They are used as component services to build more complex services or are used standalone. Within JDeveloper, database controls were generated directly from the TopLink POJOs and queries were built using the graphical expression builder. In WLI, by contrast, the developer had to write some code and include the SQL annotation for the database operation. And in WSADIE, the developer had to build Web Services for the same functionality.

- **Business Services.** In the expense reporting application, workflows were composed of simple services and human input tasks to automate human workflows. WLI provides a proprietary solution for building workflows. Both WSADIE and Oracle's BPEL solution contained BPEL designer tools and an execution environment. Oracle BPEL Process Manager provided out of the box monitoring capabilities while WSADIE provided a process debugger.

- **Presentation Layer.** Oracle ADF simplified generation of JSPs that present data from a database control. Neither of the other tools provided such features, which meant the developer had to implement such functionality.

We recommend that individuals who are considering SOA development tools consider these factors in evaluating the productivity gains offered by these products.

Please write to us at research@middleware-company.com to share your impressions and experiences with us.