An Overview and Comparison of Three Major Enterprise Application Development Platforms

Dr Ashley M. Aitken, Member, IEEE

[1]Ashley M. Aitken, School of Information Systems, Curtin University of Technology, Perth, Australia
e-mail : ashley.aitken@cbs.curtin.edu.au

Abstract—**This paper presents an overview and comparison of three major industrial-strength enterprise application development (and deployment) platforms. The platforms are Microsoft.Net, Java 2 Enterprise Edition, and WebObjects. The comparison includes discussion of the presentation tier, application tier, persistence tier, deployment, and tools. Although not a complete survey and somewhat subjective in its final recommendations, the paper should provide a general understanding of these three major platforms and how they compare. The choice of development platform is a significant one for developers and organisations (because they generally have a significant learning curve) and should not be made without a good understanding of the alternatives.**

Index Terms — **enterprise, application, development platform.**

I. Introduction

There are currently many enterprise-level application development (and deployment) platforms on which to build industrial and enterprise applications. In this paper, an overview and comparison of three major platforms is given. It focuses on the Java 2 Enterprise Edition, Microsoft.Net, and WebObjects (although a brief mention of a few other platforms is made). Knowledge of some of the available platforms and pertinent issues related to their comparison is important because of the investment of time (mostly in the significant learning curve) needed to become competent in any of these platforms.

Language versus Platform

It is also important to understand how the nature of development has changed. In the past, the primary consideration for developers was the programming language to be used. These days, however, in industry most new development is done using an object-oriented programming language. Although the syntax may differ from one OO programming language to another, the general concepts are quite similar. So, to a certain degree the choice of which OO programming language is no longer a primary consideration. There is some distinction, however, between the more dynamic and the more static OO programming languages (with the current trend towards more dynamic OOPLs).

Today software development is defined more by which software *platform* one is developing on (than the specific OO programming language). A platform consists of many things, including the programming language, the run-time environment, and most importantly the (usually very large) library of reusable classes. While learning a new OO programming language should take only a few days (for someone who is already knowledgeable and skilled in OO programming), it can take many months to learn the details of a new (and large) development platform, and even longer to become competent in its use. As a result, the choice of the application development platform is a significant decision and usually not a choice that can easily be changed (for an application or a developer).

Enables versus Supports

It is important to remember that most computing platforms are Turing Complete, in the sense that any software that can be developed could (in theory) be developed on any of the platforms. The obvious question then is, why choose one platform over another. The reason is clear if we contrast a platform that *enables* a particular system to be developed with one that *supports* it [1]. A platform enables a system to be developed if it is possible to develop that system using the platform. Whereas, a platform supports the development of a particular system if it assists the development. A simple example is that assembly language (only) enables the development of complex systems, whereas an object-oriented programming language supports the development of complex systems.

Contents

The first section provides an overview of the three platforms, the second section introductions the area of comparison and provides the comparison itself, and finally the third section provides a summary and subjective recommendations as to when it would be most appropriate to use each of these Enterprise application development platforms.

II. Overview of Platforms

Microsoft .Net

.NET [2] represents Microsoft's (relatively) new application development platform for the desktop and the World Wide Web (to eventually replace the current Windows 32 and other APIs for software development). It has been developed relatively recently, as Microsoft's response to the Java platform (which they were unable to "embrace and extend"), and so Microsoft was in the fortunate position to be able to learn from issues with the Java platforms and to incorporate the latest focus of the Web, i.e. Web services and XML throughout .NET. .NET is more than a platform for Microsoft it is their focus, their strategy, and their plan to

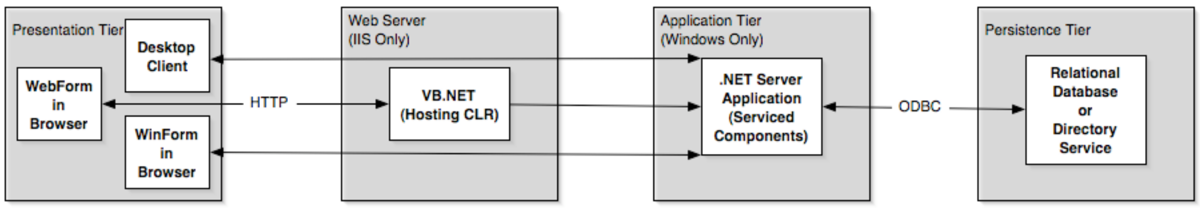

Figure 1 - Overview of Microsoft.Net Architecture

move from "software as product" to "software as service." It has not been, and will not be plain sailing for Microsoft though, e.g. their plans to be the only gatekeeper to personal data and services with "MyServices" has received a very poor reception from business and so been reorganised (but not removed from) within .NET.

.NET, the platform, includes a number of parts. Firstly, there is the Common Language Runtime (CLR), which is similar to the Java Virtual Machine (JVM) in that it runs all .NET managed code. Unlike the JVM, however, the CLR works with many languages, ranging from VB.NET, C# and C++, to Eiffel and J# (the Java language for .NET). It should be made clear that what primarily separates these languages is their syntax as they all compile to the CLR Intermediate Language (IL). They all share the same object model. As a result, VB.NET is only similar to traditional VB in its syntax. The underlying object model is completely different (so at last VB is to become a real object-oriented language). Code that runs on the CLR is called managed code and, for example, this places restrictions on what can be done with C++ in .NET. All code is compiled (automatically) before it runs so that no code is interpreted (as is default for the JVM).

Included in .NET with the CLR are a large number of class libraries, ranging from foundation class libraries to high-level GUI and Web libraries. All of these libraries will work with any of the .NET languages (again because they are all compiled into the IL). .NET also includes ASP.NET and ADO.NET, the Web front-end and database back-end integration libraries respectively. ASP.NET is actually another application that hosts the CLR and provides Web integration. These will be discussed in more detail below. Central also to .NET is the use of XML throughout and the focus on Web services, both the provision and consumption of Web services, as a means of application provision and integration. Further, improvements have been made with .NET application packaging and deployment, and there are provisions in .NET (that will not be covered here) to enable development for personal digital assistants and similar devices. The general architecture and composition of enterprise applications in .NET can be seen in Figure 1.

Java 2 Enterprise Edition (J2EE)

Java 2 Enterprise Edition (J2EE) [3] represents Sun Microsystems' attempt to define a standard platform for application development that is independent of any particular operating system (and thus to wrestle developer's focus from Microsoft's operating system Windows by enabling applications to run on all operating systems that support the Java platform). This is what motivated the expression "write once, run anywhere" which has never been completely true, but surely is a different approach than Microsoft's for .NET (which could be captured in a similar fashion as "write in any language, run on Windows only"). It is important to point out also that Java is not a product, but a proprietary (but published and freely licensable) standard originally defined by Sun but now directed through a community consultation process called the Java Community Process (JCP). As a result there are many implementations of the Java platform and a healthy market place of vendors competing to produce the most innovative products at various price-points.

Java, like .NET, also contains a number of components (it is much more than just a programming language). Firstly, there is the Java Virtual Machine (JVM), which interprets the Java bytecode produced by Java compilers. Unlike, .NET's common language runtime, Java primarily supports one language, the Java programming language. There are, however, a number of third parties that offer compilers (to Java bytecode) for other programming languages. Most Java developers, however, find Java to be adequate for their development needs (as it is no longer the programming language that is the primary focus of developer's attention, it is the class libraries and platform technologies). As well, and secondly, Java includes a large number (thousands) of classes within the Java standard. Third parties have also been quick to provide a large number of complimentary and supplementary class libraries that work with Java (for just about every area and any develop-

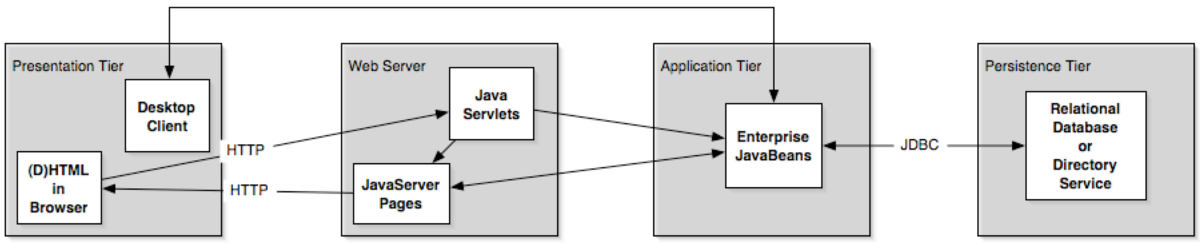

Figure 2 - Overview of J2EE Architecture

ment need).

Java consists of three different versions (primarily defined by the different class libraries available by default within each version). These are the Java 2 Micro Edition (J2ME) for embedded or personal devices, the Java 2 Standard Edition (J2SE) for desktop applications, and the Java 2 Enterprise Edition (J2EE) for enterprise applications. The micro edition has the requisite small footprint and a subset of the standard class libraries. The standard edition is focussed on desktop applications (and was in previous versions the primary focus of Java). The enterprise edition has additional class libraries to support enterprise application infrastructure, in particular middleware services. Numerous vendors (including IBM, BEA and Macromedia) have offered competing application server products built to the J2EE standard. The J2EE standard represents (like .NET) a declarative application server environment where developers configure application services by way of configuration files (as opposed to coding calls to services themselves).

Java also specifies a standard for Web front-ends, the Java Servlet engine and the Java Servlet and Java Server Pages libraries, and a library for connecting to relational databases, JDBC, and directory services, JNDI, amongst many others. Java applications and libraries can be deployed as Java Archives (JARs), Web Archives (WARs) or Enterprise Archives (EARs), although the enterprise deployment situation varies somewhat from product to product. Further, although by default Java is interpreted, there are, again, many third party product JVMs that incorporate Just-In-Time (JIT) compilers (e.g. the HotSpot JVM) that compile code at runtime that is determined to be important to the performance of an application (e.g. code in tight loops). And, although Java has suffered from poor performance in the past, mostly on the desktop where application launch time is critical and resources are often limited and to a lesser degree on the server where resources are often more abundant, the latest versions of Java that include various JIT technologies sees most Java applications run at a similar speed to native applications. The general architecture and composition of enterprise applications in J2EE can be seen in Figure 2.

Finally, with respect to the overview the of J2EE platform, it is important to point out that although J2EE is a standard with many implementations, there are also many extensions to the standard in many areas. For example, implementations of J2EE by IBM and BEA add considerably to the standard in terms of additional libraries and frameworks, as well as vendor-specific technologies and development tools. There are also a large number of other third-party (with a lot of these being free and/or open source) extensions to Java, technologies that work with Java and products that supplement J2EE development.

WebObjects

WebObjects [4] is an award-winning Java-based Web application server development and deployment environment. It is one of the most mature and technologically advanced Web application environments available (and yet also one of the least advertised and, as a result, least well known). It includes presentation, application, and persistence technologies, as well as advanced rule-based development facilities, that provide a complete solution for rapid and high quality Web application and Web service development. WebObjects is a commercial, industrial strength, and fully supported application server without the high price tag of comparable J2EE or .NET commercial products.

As WebObjects is Java-based, application developed with WebObjects can be deployed on any server supporting Java 1.4 or higher (including servers running Windows, Solaris, Linux and Mac OS X). As well, the development tools are available for Windows and Mac OS X. WebObjects was initially developed in 1996 by NeXT and was used by many Fortune 500 companies, particularly trading companies. In 1998, Apple acquired the technology from NeXT and committed to keeping the technology cross-platform. WebObjects is a leading-edge application server but is under-marketed by Apple, although they and many large corporations and government bodies around the world use it for many mission-critical Intranet and Internet Enterprise Web applications. The general architecture and composition of enterprise applications in WebObjects can be seen in Figure 3.

Others

1) ColdFusion

Macromedia's ColdFusion [5] has been a popular Web application server for a number of years. It has, however, undergone a significant change recently, making it more suited for enterprise Web applications and services (as opposed to content management, which some would say was its strength in the past). In particular, the ColdFusion of the past used a tag-based scripting language (ColdFusion Mark-up Language) to construct applications scripts. After the change, ColdFusion still supports the ColdFusion Mark-up Language (CFML) but it can now integrate with a J2EE application server or .NET on the back-end. Macromedia will provide their own J2EE application server (called JRun), or any of the J2EE compatible application servers can be used in its place (e.g. those from IBM or BEA systems). In this,

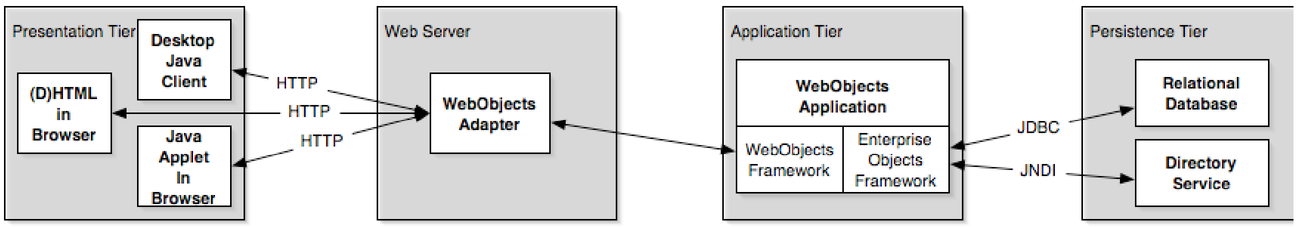

Figure 3 - Overview of WebObjects Architecture

regard ColdFusion is just another J2EE application server with a different front-end scripting language.

However, this is not the entire story with regards to ColdFusion. Macromedia is also the developer of Flash, the leading vector-based engine for Web pages. Flash pages allow the developer to incorporate vector graphics, scripted dynamics, and multimedia in a Web page but requires a Web browser plug-in to enable this functionality. To date this has been used primarily for "flashy" home pages or product overviews and demonstrations. With the new ColdFusion Macromedia have extended this to enable developer to make "flashy" user-interfaces, i.e. user-interfaces using Flash, and a Flash user-interface widget set, that connect to ColdFusion and J2EE and .Net application servers. This technology is called Flash Remoting and enables developers to make highly dynamic and interactive interfaces for Java or .Net enterprise Web applications.

2) PHP

PHP [6] has stood for many things over its lifetime, from Personal Home Pages to the current recursive acronym PHP Hypertext Preprocessor. PHP is another tag-based Web application scripting environment. What sets it apart from the rest is that it is open-source and, as a result, free, and supported by a community of developers. It is not really an application server like J2EE, .NET or WebObjects but it does provide high-level libraries and integrated database connectivity, that make it appropriate for small to medium-sized Web applications. The PHP scripting language is also evolving to be a lot like Java. PHP also has a good community and library of reusable code (PEAR).

III. Comparison

Unfortunately, there is not the space to include the full details of a comparison of these platforms. As a result, only certain key aspects of the application platforms will be considered. These include: 1) Presentation Tier - the presentation tier usually relates to that part of the enterprise application that is most involved with the user-interface (if there is one). Predominantly this is a Web browser, but it may also be a thick desktop client; 2) Application Tier - the application tier contains the application and core business logic and data of the application. The application tier is also where the application server (middleware software that provides application support infrastructure and services) resides; 3) Persistence Tier - the persistence tier usually relates to the retrieving and saving of business data to a (primarily relational) database. It, however, may alternatively involve other forms of repositories (e.g. an object-oriented database or a light-weight directory service); 4) Deployment - deployment relates to the method, tools, and complexity involved with deploying enterprise applications built within the platform. This may include deployment on the presentation, Web, application and persistence tiers; 5) Tools - tools relates mostly to development tools (i.e. usually integrated development environments) but also to testing (e.g. unit or load testing) and deployment and monitoring tools; and 6) Miscellaneous - this aspect of the comparison represents a "catch-all" category for other general considerations.

Microsoft.Net

1) Presentation Tier

The presentation layer development facilities offered by .NET continue in the "visual" paradigm that made Microsoft's suite of Visual tools (e.g. Visual Basic) so successful and popular. The presentation layer in .NET can, as is currently the case for Windows, be developed using drag-and-drop of GUI elements, and event-based programming with WinForms. Interestingly, Microsoft has extended this paradigm to Web interface development with WebForms, which employs a very similar drag-and-drop of Web elements, and event-based programming. Obviously, this is an attempt to build on expertise and familiarity with such development for the desktop. Clearly, to move event-based programming to the Web, an inherently request-response architecture, has required some extra work. To Microsoft's credit they have done this extra work (e.g. maintaining input element state across trips to the server, and enabling DHTML with browsers that support it) and provided a consistent approach for developing both Web and GUI interfaces. As well, it is possible to download WinForm interfaces and run them from within the Internet Explorer Web browser (much like Java applets but only on the Windows platform).

2) Application Tier

Microsoft also includes in .NET a declarative application server environment that provides many application services and infrastructure (such as object pooling, caching and load balancing), on an appropriately configured and resourced Windows server(s). Use of the application server services requires primarily that the developer subclass the "ServiceComponent" class and declaratively states (with annotations in the source code and configuration files) what services are required for particular components. This is Microsoft's attempt to get into the enterprise server room, which is currently dominated by more traditional enterprise application environment (including for example, CORBA and, more recently, J2EE). As mentioned above, Microsoft when developing .NET was in a position to reflect on the successes and failures of previous technology and had a chance to start fresh, leaving behind most (although still allowing integration with) legacy Microsoft technologies (like ActiveX and the old Visual Basic).

3) Persistence Tier

To facilitate persistence, Microsoft provides a much revamped (but similarly named) ADO.NET. Active Data Objects link certain classes within .NET applications to relational databases or other data services. Although an improvement over the traditional ADO, it still (at it's lowest level) requires the developer to send SQL to the database, and to process the data that is returned. Admittedly links can be created with GUI or Web tables to automatically display the data retrieved from the data source. However, all this is a long way from creating business objects based on the data (which should usually be the goal in an object-oriented environment). As a result, this is probably the weakest leg of the .NET strategy that otherwise is fully object-oriented and presents a high-level of abstract. That said, there are some object-relational mapping systems that are targeting .NET and so, perhaps, they can provide the additional support needed in this area.

4) Deployment

.NET makes a significant (and desperately needed) change to the packaging and deployment of Microsoft desktop and Web applications. .NET applications are now packaged as a single unit, not spread throughout the operating system (as has been the case to date). The first result of this is that there will be no more "DLL Hell." The second result is that multiple versions of the same library or application can co-exist on the same machine. Thirdly, it now enables applications to be run from a CD-ROM or from a server. Microsoft has surely learnt from past mistakes in this area, and it was about time.

5) Tools

The premier tool Microsoft provides for .NET development is Visual Studio .NET. It represents a single development environment for creating all .NET applications (including desktop, Web, and enterprise applications). It represents the culmination and coordination of all Microsoft's previous effort in "visual' development tools and, as such, is a most sophisticated, advanced, and powerful integrated development environment. On the other hand, considering it is relatively expensive, it would be hoped that significant productivity gains would arise from the IDE, and this seems to be the case. It should also be said that Visual Studio.NET is not needed, in theory, for development with .NET. However, practically speaking, to develop without it (or an equivalent tool) on an enterprise application would not be a sensible option.

6) Miscellaneous

One of the most successful application development environments for the Windows platform has been the Delphi (Object Pascal-based) integrated development environment. .NET now includes most of the powerful abstractions and rapid development facilities that made Delphi such a powerful application development environment (at least compared to traditional Windows development) and adds some more of its own, as discussed above. As a result, some reshuffling is was in the Delphi camp. Borland has announced Delphi.NET as their move to be a part of the .NET future. Delphi.NET attempts to maintain the traditional Delphi environment and components whilst compiling to the .NET CLR and libraries. Although this is technically successful, it is difficult to see what this provide developers (above the continuation of legacy APIs and the Delphi development tools) above and beyond Microsoft's .NET offering.

Java 2 Enterprise Edition

1) Presentation Tier

The primary mechanism within J2EE to facilitate the presentation layer is the Java Servlet engine specification. Servlets are Java classes that produce HTML (or Dynamic HTML or XML) and run within a Java Virtual Machine incorporated within a Web browser. The fact that the JVM is incorporated into the Web server allows the Web server to respond to Web requests using Java Servlet classes without having to start JVM for each (CGI) request. J2EE also incorporate a specification for Java Server Pages (JSP). Java Server Pages are a template-based mechanism for producing Web content (HTML, DHTML or XML) that facilitates better separation of page design and code development than Java Servlets (that produce the HTML directly from Java output statements). Java Server Pages are converted into Java Servlet pages but the engine handles this automatically. The standard design pattern for J2EE front-ends is to use Java Servlets as the control classes (e.g. to respond to submitted Web forms and to instigate business functions) and to use Java Server Pages to produce response pages.

Java offers great flexibility in the presentation layer as well. Java Applets (small Java programs that are downloaded and run in the Web browser), as discussed earlier, allow for a much more dynamic interface within the Web browser but have limitations (e.g. download speed and limitations of the Web browser JVM). Of course, Java can also be used for the development of desktop applications that can work as thin or thick clients for the enterprise Web application. Although Java on the desktop has had problems in the past due to slow start-up time and tardy performance (as mentioned above), the latest JVMs and other technology has come a long way to improving the performance and reliability of Java on the desktop. The deployment of Java desktop applications has also been substantially improved with the Java WebStart technology. It enables Java applications (thin, thick, or full desktop applications) to be downloaded and run from a Web page URL (and/or a desktop icon), as well as being automatically and incrementally updated if a new version becomes available on the server. This removes most of the difficulties associated with using desktop client applications for an enterprise application (but, of course, not any bandwidth problems).

2) Application Tier

J2EE incorporates the Enterprise JavaBean (EJB) model for service components. It is a declarative application server environment wherein developers produce EJBs and deploy them within an EJB container. A configuration file specifies the middleware services that the EJB requires, leaving the EJB code to focus primarily on the business logic. There are three types of Enterprise JavaBeans: 1) Entity Beans that are primarily used to model business objects, 2) Session Beans that are primarily used to model control objects, and 3) Messaged-based Beans that are used to process asynchronous requests. Session beans may also be stateful or stateless. While the EJB model is complex it is also powerful and provides a high level of abstraction for the developer. A lot of the complexity can be handled by the developer tools (as discussed below) leaving a full-featured serviced environment in which enterprise applications can run. The performance and quality of this environment, however, depends upon which implementation of the J2EE specification is being used (as is discussed below in the Tools section).

3) Persistence Tier

To facilitate persistence, the J2EE recommends one of two (or more) EJB-based approaches. These are bean-based persistence and container-based persistence. In the former, it is the bean's responsibility to manage it's own persistence and in the latter it is the container's responsibility to manage the persistence of beans within it. J2EE, however, provides little in the way of a specification of how this persistence should be implemented. J2EE does provide the JDBC (apparently not an acronym for Java database connectivity, but that is what it provides) and JNDI (an acronym for Java Native Directory Interface) APIs to facilitate persistence in a relational database or directory service. However, all this is a long way from creating enterprise objects based on the data. As a result, this is probably the weakest leg of the J2EE specification. That said, there are object-relational mapping systems that are Java-based (e.g. Hibernate and Cayenne) and so, perhaps, they can provide the additional support needed in this area and may even be incorporated into a future version of the specification. The EJB standard went through a major upgrade with EJB2 and is now in the process of being completely overhauled in EJB3.

4) Deployment

J2EE enterprise applications are deployed as Web Archives (WARs) or Enterprise Archives (EARs). These are similar to Java Archives (JARs) but contain configuration files for the enterprise Web application. The details on deployment, configuration, and application management depends on the particular J2EE product that is used. As for most Web applications, deployment is usually split between the Web server and the application server. Java Server pages and Java Servlets are deployed on the Servlet engine (Web server with container) and the enterprise application is deployed on the application server.

5) Tools

It is important to remember that Java is a specification not a product itself. As a result, there is not one specific solution for Java development; there are a number of competing and mostly compatible implementations of the specification. Each of these products provides its own suite of tools to assist with development, and these may encompass minimal to full support for development. The industrial strength tools like WebSphere from IBM and WebLogic from BEA Systems provide impressive integrated development environments that assist in all stages of the Web application development. These products, however, are generally quite expensive. On the other hand, there are open source J2EE implementations (like JBoss and OpenEJB) that provide minimal tools support for development (but work well with open source IDEs like Eclipse) but are free. The benefit of this variety is that the developer has a choice and that code developed in one produce should be (relatively) easy to move to any of the other products.

6) Miscellaneous

Although Java was developed long before Web Service standards (such as SOAP, UDDI, and WSDL) had become popular, Sun have taken aggressive steps to incorporate Web Services into the J2EE standard. The current version of J2EE now has a similar level of integration as Microsoft's .NET. Of course, the different J2EE product vendors have taken their own steps to facilitate the development of Web Services into their products and applications developed with their products.

WebObjects

1) Presentation Tier

WebObjects provides great flexibility in the support of the presentation tier. WebObjects applications can produce HTML, Dynamic HTML or XML as their output. WebObjects can be used to rapidly produce template-based Web pages, components, and site-wide page layouts. WebObjects also includes (what it calls) JavaClient technology, facilitating a client-side Java application to provide a more dynamic desktop-app-based user-interface for a Web application. The JavaClient technology also enables thick client applications by enabling downloading of developer-partitioned business object to the client application via standard HTTP (or HTTPS) protocols. This enables the client to perform some business processing locally without needing to contact the server and with a dynamic user-interface. The partitioning allows only parts of the business data or logic to be downloaded to the client, and to require that certain business operations only be performed on the application server itself. This is useful for protecting important data or logic from being exposed on the client. The JavaClient technology also enables the user-interface itself to be downloaded to the client as XML and to be constructed dynamically.

2) Application Tier

WebObjects was one of the first application servers on the market. It employs two object-oriented frameworks to provide Web integration and persistence. The first will be discussed here, and the second in the next section. The first framework, call the WebObjects Frameworks (WOF) enables business objects to be linked to parts of a Web page, e.g. so that a customer's name will be displayed on a Web page, as well as enabling form fields to be linked to business objects, e.g. so that the address a customer enters can be put into the customer business object. The application server itself also provides load balancing (across multiple instances of the application on one or more hosts) and sophisticated caching of objects fetched from relational databases. Although not, perhaps, as full-featured or declarative as the .NET or J2EE application servers, the WebObjects application server is considerably less complex and provides most of the features needed in all but the biggest enterprise applications.

3) Persistence Tier

WebObjects includes another object-oriented framework, called the Enterprise Objects Framework, that provides persistence for business and other objects within a WebObjects enterprise application. This is an object-relational mapping system that handles fetching and saving objects to relational databases. It automatically generates the SQL to fetch and update data in the database as required by the object-oriented applications. This enables the developer to focus on the object-oriented application as opposed to writing SQL database code. As well, the EOF automatically handles fetching of related object and updating only the data that has changed within the objects. The EOF allows applications to connect to (and have relationships across) multiple database that have JDBC drivers (which includes all of the major relational databases), as well as directory services, that have JNDI drivers (e.g. LDAP directories). It is also relatively easy to swap between different databases protecting an organisation from vendor-lockin.

4) Tools

WebObjects comes with a suite of tools for Web application and service development and deployment. These include (but are not limited to) Project Builder, EO Modeller, WO Builder, and JavaMonitor. Project Builder is an integrated development environment for constructing, editing, running, and debugging projects in the direct-connect developer mode (which does not require the application to be installed on a Web server). Enterprise Objects (EO) Modeller is a powerful tool for constructing object-relational mappings that also can, amongst other things, automatically reverse engineer an object-relational mapping from a relational database, automatically generate the SQL to construct tables and relationships within a relational database from an object-relational mapping, and also automatically generate skeleton Enterprise Objects Java classes from an object-relational mapping. WebObjects (WO) Builder is a tool for building Web pages that are linked to business (or other) objects (as well as components to use within other pages). Finally, JavaMonitor, is a WebObjects application that allows an administrator to setup, track, and manage the deployment and operation of WebObjects applications across multiple machines from a Web browser.

5) Deployment

WebObjects applications can be deployed using JavaMonitor (as mentioned above) as standalone applications or they can be deployed within a Java Servlet container. A WebObjects application can scale easily by the addition of additional instances of the application on one machine, or multiple applications across multiple machines. The WebObjects adapter (which can run as a CGI application or be linked into major Web servers) can handle load balancing (including random, round-robin, and load-based). The JavaMonitor tool enables the administrator to gracefully shutdown an application, track its usage, or periodically restart an application. WebObjects applications can also be set up to email an administrator when an exceptional event occurs within the application, and to automatically restart if the application or operating system crashes. Although not a J2EE application server itself, WebObjects is J2EE compliant, which means that WebObjects applications can be installed in a J2EE server, and integrate with a J2EE application, for example with its Enterprise JavaBeans.

6) Miscellaneous

Finally, WebObjects also leads the Web application server market with three additional (but similar) technologies called DirectToWeb, DirectToJavaClient and DirectToWebServices. These technologies are built around a rule-engine that allows rule-based development. Rule-based development is an even a higher level of abstraction than object-oriented development (in which it is implemented). It enables developers to rapidly build and modify the presentation and operation of Web applications, both with browser-based interfaces and JavaClient desktop-application-based interfaces. And it does this without code generation! For this (automatic user-interface generation) to be possible with Java desktop application, WebObjects incorporates technology that allows partial or complete user-interfaces to be specified in XML and generated dynamically by a small (base) client-side application. WebObjects can also apply this technology to rule-based Web Service development. The power and productivity of these tools is unprecedented in the Web application and service market.

IV. Summary and Recommendations

In summary, we have considered three major enterprise application development platforms. The first, Microsoft.Net is a proprietary product based on an unpublished standard that works only on the Windows platform. The second, J2EE, is a published (and freely licensable) standard with many competing product implementations (ranging from free and open source products to relatively high-priced commercial products) and an innovative community producing a number of competing and complimentary technologies and products. The third, WebObjects, is a mature proprietary product developed on the J2SE platform that provides a solid foundation and a number of innovative features that the other two platforms don't as yet include.

Of course, which is the best platform for an organisation's next Enterprise application development depends on many different issues and criteria. Any final recommendations made here are mostly subjective and based on generalised assumptions. However, that said, it is possible to say that if you have a Windows-only environment then Microsoft.Net can provide a good solution for Enterprise applications (with limited choice and limited influence on future directions but the benefits of one source and a known supplier). If, however, you are in a heterogeneous environment then one of the Java platforms will be most appropriate. The Java 2 Enterprise Edition platform is very large and has something for just about any type of application from the a simple Web tier application up to the largest of Enterprise applications requiring a large degree of scalability, security, reliability and robustness. Further, there are many add-ons and third party solutions (open-source and proprietary) that work within the J2EE space and compliment the offerings within the official standard (e.g. Tapestry, Hibernate, and Cayenne). Finally, if you want a well-balanced solution (from the presentation layer to the persistence layer and beyond), and don't mind (or even prefer) a relatively cheap but proprietary/supported solution for medium to large Enterprise application that works on the Java platform (and so can benefit from all that is Java) and has some unsurpassed rapid development technologies, then WebObjects is an excellent choice.

V. References

[1] Stroustrup, B. *The C++ Programming Language*, New York: Addison-Wesley, 1997, 18-26.
[2] Microsoft.Net at http://www.microsoft.com/net/
[3] Java 2 Enterprise Edition at http://java.sun.com/j2ee/
[4] WebObjects at http://www.webobjects.com/
[5] ColdFusion at http://www.macromedia.com/software/coldfusion/
[6] PHP at http://www.php.net/