Alternative Technologies

Application Platform Suites
for
Departmental Integration

David McGoveran
Alternative Technologies
6221A Graham Hill Rd., Ste #8001
Felton, California 95018
Website: www.AlternativeTech.com
Email: mcgoveran@AlternativeTech.com
Vmail: 831/338-4621 Facsimile: 831/338-3113

Report Number 040420

Disclaimer and Notice

This report is produced and published by Alternative Technologies, an independent consulting and analyst firm in Boulder Creek and Felton, CA. The information and opinions presented in this report are exclusively those of Alternative Technologies, except where explicitly quoted and referenced. Although reasonable attempts have been made to insure the accuracy of the report, no guarantees or warrantees of correctness are made, either express or implied. Readers are encouraged to verify the opinions stated herein through their own efforts.

This independent report was supported in part by BEA Systems, Inc. without influence on the opinions herein. Client (BEA) is granted a non-exclusive license for unlimited distribution of this report in its unabridged and unaltered form for internal use. Neither this report, nor any abridgement or derivation of this report, may be otherwise reproduced in any form or media without the explicit written permission of AT (Alternative Technologies). This restriction in rights is not intended in any way to impede the mutually intended use of this report by Client and AT; rather, it is intended to preserve both the integrity of the report while maintaining both AT's rights of reuse of AT's intellectual property and any confidential information of either Client or AT contained herein. Portions of this report which directly discuss facts pertaining to Client, Client's products and services, or other aspects of Client's business are assumed to be Client Confidential (if this report is designated "Client and AT Confidential") and otherwise are published by permission. This report is, in general, the sole property of AT and protected under International Copyright laws.

For information about this or other reports, or other products and services (including consulting and educational seminars), contact Alternative Technologies directly by telephone, mail, or via our Web site:

Alternative Technologies
6221-A Graham Hill Rd., Ste #8001
Felton, California 95018
Telephone: 831/338-4621 FAX: 831/338-3113
Email: mcgoveran@AlternativeTech.com
Website: www.AlternativeTech.com

TABLE OF CONTENTS

1. Introduction

Most software projects, and especially those involving integration, are managed at the departmental level to meet tactical objectives. Important productivity tools such as application servers and integration brokers have barely penetrated this market. In fact, according to a leading independent analyst firm, only about 6% of all integration projects (both corporate and departmental) use a commercial integration broker.[1] Designed for strategic software projects more likely to be managed by corporate IT and focused on long-term and corporate-level objectives, products such as application servers and integration brokers have not addressed the problems that departmental project managers face. Until recently, departmental project managers have not had the option of tools and facilities that address their particular development and integration problems. Instead, low cost, flexible integrated development environments have been used that do not facilitate runtime management, integration, or reuse.

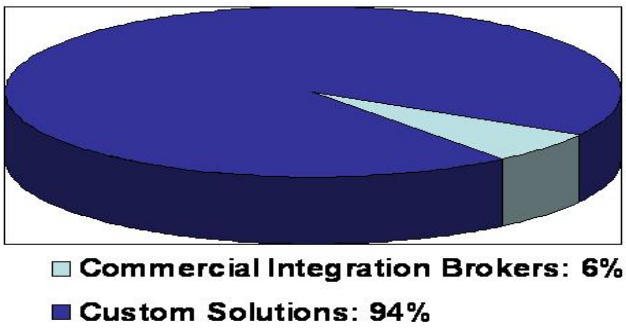

Figure 1: Commercial Integration Broker Market Penetration

The advent of the APS (Application Platform Suite) promises to change the situation. An APS comprises the design time and runtime facilities of an application server, and integration broker, and a portal in a single environment. The synergy obtained by seamlessly combining these facilities in a unified architecture has many benefits, not the least of which is to bring enterprise class infrastructure within the reach of departmental software projects. Development and integration no longer need be treated as separate disciplines with separate tooling. Significant strategic IT value can be obtained without sacrificing tactical IT objectives, violating departmental operational constraints, or

[1] The analyst firm also points out that most corporations that have adopted an integration broker approach have purchased multiple, different products. This "broker proliferation" suggests that corporate standards and commitments to a single vendor have yet to be adopted. It also suggests that, though managed by corporate IT, projects involving integration brokers are significantly less than enterprise-wide in scope.

prematurely committing to a proposed standard infrastructure. As a result, we predict that the percentage of departmental development and integration projects using an APS will increase dramatically over the next five years.

This report will help departmental technical managers understand the potential of a J2EE Application Platform Suite such as BEA WebLogic. We begin by motivating for the convergence of development and integration tooling and explaining why that separation leads to inefficiencies (Section 2). As will become obvious, these inefficiencies are never more dramatically realized than during departmental software projects. After discussing some of the difficulties faced by departmental project managers (Section 3), we describe the facilities that comprise an Application Platform Suite, key desirable features, and how those facilities and features address departmental concerns (Section 4). Finally, we examine the crucial importance of a unified architecture (including a common development model) for departmental adoption of an APS.

Although much of the discussion in this report is product independent, we use the specifics of BEA's APS (WebLogic Server) to explain certain key features. We note in passing that BEA's implementation of an APS addresses departmental requirements in ways that alternatives do not.

2. Integration Requires Development

Integration and development are so intertwined conceptually and in practice that their distinction is mostly a matter of historical accident. Departmental development projects often require integration with pre-existing applications and other resources. It is no accident that application development has always included system *integration* as a key phase. Even if departmental development efforts do not involve the kind of real-time integration of distributed systems anticipated by enterprise application integration vendors, the problems of establishing compatible interfaces among disparate software applications and modules are just as complex.

Similarly, integration projects usually require development. Without sophisticated application integration tools, integration usually degenerates into a complex custom development effort. Even when sophisticated integration brokers and messaging infrastructures are in place, they must be supplemented with software development tools, a fact that most businesses and integration broker vendors have already discovered. With the rise of component based and service oriented architectures, integration of components and services is conceptually indistinguishable from application testing and deployment, a fact which is being leveraged by some APS vendors.

Integration requirements arise from two distinct sources:

- Pre-existing or planned departmental applications and resources need to be used in some combination, including being integrated into a departmental workflow; and,

- Corporate IT (including other departments) assets may need access to departmental resources, or departmental applications may need access to corporate IT assets.

Application integration is accomplished via numerous methods, including custom or point-to-point integration, message-based integration, data integration, and presentation centric integration. Each of these have come to be perceived as best managed through corporate IT mandates for infrastructure deployment to the detriment of departmental tactical goals and integration efforts. As we shall argue in this section, they have erroneously become distinct from their development roots.

Custom or Point-to-point Integration

With all distributed systems, departmental developers have to worry about communication protocols and interfaces. In the absence of an application server or integration broker, even pure point-to-point integration requires setting up and tearing down connections. This process represents considerable overhead and development that often cannot be implemented as a service even within a single application (due to

limitations on the use of global variables or a lack of multi-threading). To make matters worse, not all platforms and operating systems support the same protocols and communication methods in the same way, and the developer may have to write to platform-specific APIs. Older distributed systems were usually developed with synchronous invocations because of a lack of the programming skills required to do otherwise. Unfortunately, this has the further effect of more tightly coupling system components and demanding even more extensive development in response to changing requirements than would otherwise be the case.

Message-based Integration

Asynchronous invocation enables loosely coupled distributed systems, and thereby potentially reduces some of the burden of development during integration. However, that development burden is shifted to the distributed communication layer where MOM (message-oriented middleware) provides standard facilities for queuing, guaranteed once delivery, and recovery. It has not been until recently that such facilities have become standardized (JMS), enabling adopters to reduce some of the burden of customized development.

Even so, message-based application integration has hardly been plug-and-play, particularly without extensive and traditionally complex infrastructure investments. Every application potentially presents a unique set of challenges to any developer wanting to access its data or to drive its functionality with data. While acquisition of packaged enterprise application software certainly means that some business functions are semi-standardized (simply because the number of vendors was relatively small), it also means that IT has little or no knowledge of how to develop a custom interface to the package and little assurance that it would not need to redevelop the interface with the next release of the product. Some vendors publish APIs that provided direct access to business functions, others publish APIs that provided access to particular subsets of the data (sometimes called business objects), and yet others published no APIs at all.

Developing an interface or adapter that permits an application to pass messages between one or more other applications is a non-trivial task. Vendors of integration brokers have developed a long list of “out-of-the-box” adapters for the most popular packaged applications, DBMSs, and the like to help reduce the amount of development required. However, this does not mitigate the need to develop other adapters to, for example, the custom applications which are so prevalent in departmental IT environments. Development support of the JCA (Java Connector Architecture) standard can greatly facilitate this requirement.

Data Integration

Data integration is an alternative approach to direct system level integration. Data integration is effectively a hub-based approach to integration. The techniques for this method of integration were originally developed for creating consolidated databases including, for example, ERP databases and data warehouses, and recently extended with

EII (Enterprise Information Integration). If all applications (or modules) use a common data store or at least a common data schema[2], then the complexity of developing an interface between every application to every other application is greatly reduced. Unfortunately, data integration as a principle strategy brings its own set of problems. If database access standards (e.g., ODBC or JDBC) are not adopted, every application typically involves custom development of the data access layer. Consequently, redeploying the data to an enterprise data store means redevelopment of that layer.

Although many companies have pursued the concept of an enterprise data model, the reality remains a complex array of data stores with poorly articulated, let alone consistent, data models. The result is that data integration requires that the movement of data between data stores be carefully coordinated for operational reasons (the extract and load phases of ETL), and mediated by transformations. Data integration tools are therefore restricted to those data stores for which the vendor can supply specialized interfaces and those transformations that could be easily scripted using a standard library of functions. Support for other data stores and more complex transformations require custom development.

Presentation-centric Integration

Yet another approach to integration is to provide a presentation layer within which multiple applications or data sources are combined on the screen. This architecture is especially common when legacy applications or databases are involved, and when a department's access to an application, application code, or database is restricted. Traditionally, presentation-centric integration required the development of fat client applications as well as interfaces to backend systems. In some cases, screen scraping was used. With the introduction of web-based solutions of this type, called portals, a relatively non-intrusive, yet effective approach has developed. Used in conjunction with information integration tools, portal technology provides integration while minimally interfering with other systems. Nonetheless, most portal tools do not completely eliminate the need for development except for the simplest of portal applications.

Development or Integration?

All of the problems discussed above are common to both development and integration projects. The customization requirements of departmental development often arise from the need to integrate existing applications. Despite predispositions, it should be clear that development and integration are more similar than distinct and are clearly entangled in practice. In particular, development practice has moved increasingly toward a combination of component development and component assembly. The rapid adoption of

[2] The data transformation capabilities of message-based integration brokers and of ETL tools are converging as messages grow in complexity and frequency, and as ETL tools support for continuous load is improved.

EJBs (Enterprise Java Beans), service oriented architectures, and, currently, of Web Services is promoting the view that all components should be understood as providers of services, whether those components are EJBs or enterprise applications. With an APS, this viewpoint has a remarkable unifying effect on development and integration. In the next section, we examine how this unified viewpoint can be given its most beneficial expression for departmental development and integration.

3. Departmental IT Challenges

Departmental project managers work under severe constraints and pressures, frequently forcing them to be at odds with corporate IT agendas and procedures in an effort to achieve tactical advantage or operational efficiency. In order to control budgets, functional scope, and deliver in a timely fashion, departments discovered long ago that they must maintain a degree of autonomous control over departmental software projects.

More often than not, applications are developed with a minimal development environment such as a traditional IDE (Integrated Development Environment) and generally without the sophistication of either a complete application server or an integration broker for integration needs. The key benefits of this departmental approach have been rapid time to deployment, finely tuned performance, and functional customization. These benefits often outweigh such deficiencies as a lack of flexibility, reusability, corporate standards adherence, scalability, system administration, and cost effectiveness, especially when the constraints described below are taken into account. Indeed, we believe that the relative importance of the former benefits in conjunction with departmental challenges (such as limited budgets) accounts for the low market penetration of both application servers and integration brokers.

The key challenges faced by departmental IT managers, and the painful effects of those challenges, include the following:

- **Limited budgets** – Departmental IT budgets exist solely for the purpose of ensuring that departmental objectives can be met rapidly without the entanglement of enterprise IT budgeting and lengthy approval processes or restrictions. With limited budgets, managers are seldom able to fund the high cost and lengthy returns on IT infrastructure and reusable assets. *Departments pay repeatedly for custom, non-standard, infrastructure and integration costs that (according to analyst estimates) exceed 40% of each project budget.*

- **Short term, focused goals** – Departmental software projects often need to be delivered within a very short time frame, typically ranging from weeks to months. Approval for such projects is usually within the department, using departmental resources. Project objectives are focused on delivery of specific functional capabilities that will contribute to the departmental operations in the near-term. *Departments walk a tightrope between compliance with corporate IT (and the threat of being usurped) and meeting operational, tactical objectives.*

- **Focused integration** – Functional gaps in existing software are filled in either by extending that software or supplementing it. Inefficiencies may be addressed either by integrating existing applications, by integrating diverse user interfaces, or by introducing workflow automation. Most departmental integration is point-to-point,

custom development. *Departments repeatedly pay the costs and time delays for inflexible, non-scalable integration solutions.*

- **Highly specialized developers** – Departmental development teams are often more knowledgeable about departmental business requirements and use-cases than their enterprise IT counterparts, but are not as familiar with enterprise infrastructure. *Departmental software seldom complies with corporate IT standards and must be redeveloped when regulatory, corporate IT, or another department requires access, management, auditing, and security controls.*

- **Process Integration** – Departmental integration involves few applications and more human interactions than enterprise integration. Custom application development may result in stovepipes spanned by departmental processes and workflows and requiring process integration. *Departmental use of traditional workflow tools or custom process integration promotes inflexibility and higher development costs.*

- **Small Teams** – Departmental development teams are often quite small, consisting of four or five seasoned developers. *Departments cannot afford the training costs associated with multiple development and integration models, application servers, integration brokers, portals, and so on.*

- **Customization** – Raw performance, functionality, and resource consumption requirements for departmental solutions are frequently the most important design goals. *Departmental applications and integration suffer from a lack of standardization, reusability, flexibility, and scalability.*

The choice of application development and integration environments for departmental solutions must be sensitive to the constraints listed here. Additionally, even if it's a departmental solution, the load on an application will grow rapidly when it becomes accessible to other departments, the sales force, customers, or the general public. By contrast with enterprise strategies, traditional departmental application development and deployment strategies cannot handle this increased load in an equally graceful manner. Project managers need a way to meet both the strategic goals of corporate IT and the tactical goals of departmental IT.

As we shall see, an Application Platform Suite, based on a unified architecture and development model affords businesses an opportunity to absorb the impact of change while leveraging existing resources. Departments can simultaneously respond to short-term operational constraints (as described above) while continuing to build a consistent set of technology assets compatible with corporate mandates. The result is better budgetary responsibility and tactical efficiency.

4. Application Platform Suites

APSs (Application Platform Suites), such as BEA's WebLogic 8.1 (see Figure 2), combine the capabilities of IDEs, application servers, integration brokers, and portals into a single integrated set of facilities. As such, they inherit most, if not all, of the capabilities of those product categories. Much like the transition from separate facilities for design, development, test, and source code control to IDEs, APSs promise to integrate the entire IT environment from design through runtime management. They bring a degree of uniformity and therefore productivity to the development projects, whether focused on new applications, integration, or some combination.

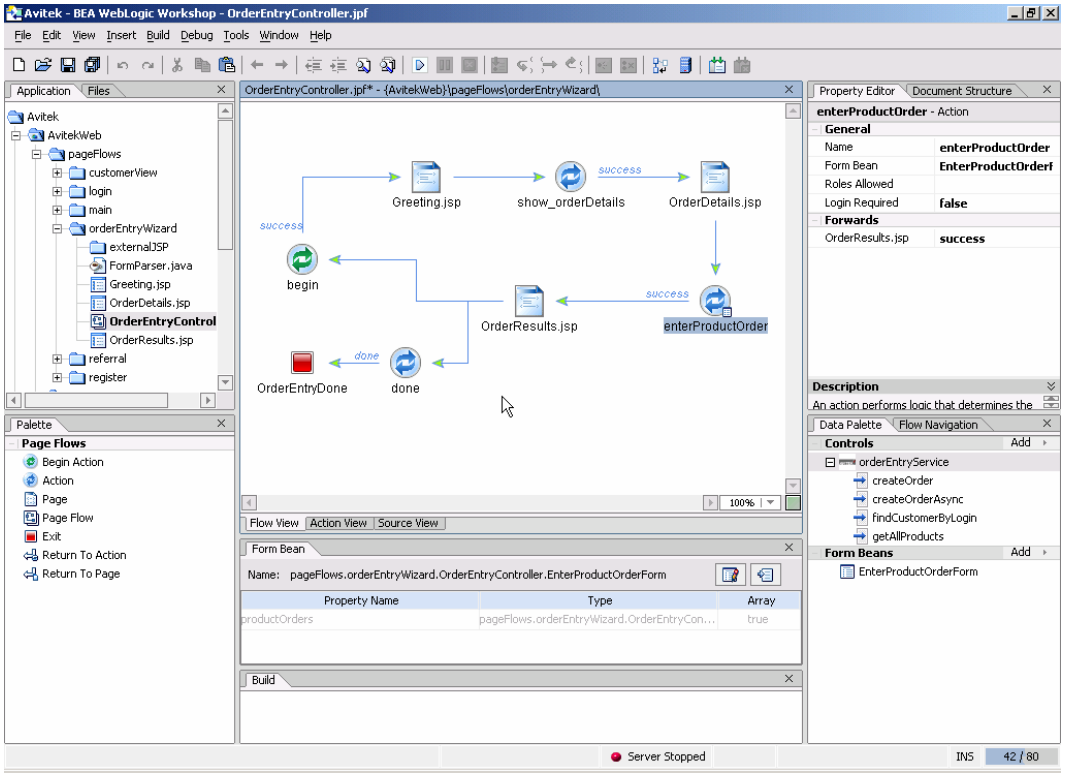

Figure 2: BEA WebLogic Workshop Visual Development Environment

APS products can be classified into two categories: proprietary and J2EE. Proprietary APSs, such as Microsoft's .NET, are likely to take advantage of particular proprietary facilities, languages, and platforms, but in doing so limit deployment to specific platforms and operating systems. By contrast, J2EE APSs build on the popularity of the Java standard and offer relatively unrestricted deployment to any J2EE compliant platform. For the purposes of this report, we restrict the discussion to J2EE APSs.

Application Platform Suites bring numerous capabilities for departmental integration, but certain key features have an important impact and should be kept in the forefront. The ideal, fully integrated APS includes support for:

- **Common User Interface**

  The APS should comply with a uniform presentation model. The developer need learn a single user interface and look-and-feel regardless of whether developing components and Web services, integrating applications, developing portals, or designing processes and flows. *This improves productivity while lowering the learning curve and training costs, thereby addressing key concerns for departmental IT.* Few APS products have the degree of uniformity in the user interface that is found in WebLogic Workshop.

- **Portals**

  Portal style user interaction facilitates not only rapid deployment and a uniform, easily personalized user interface, it also provides integration within the user interface. Portlets, each providing access to a distinct application service, can be combined in a single portal and yet inherit a common look and feel as well as interact with each other. This type of front-end integration is extremely valuable when departments need to provide extra-departmental access to departmentally

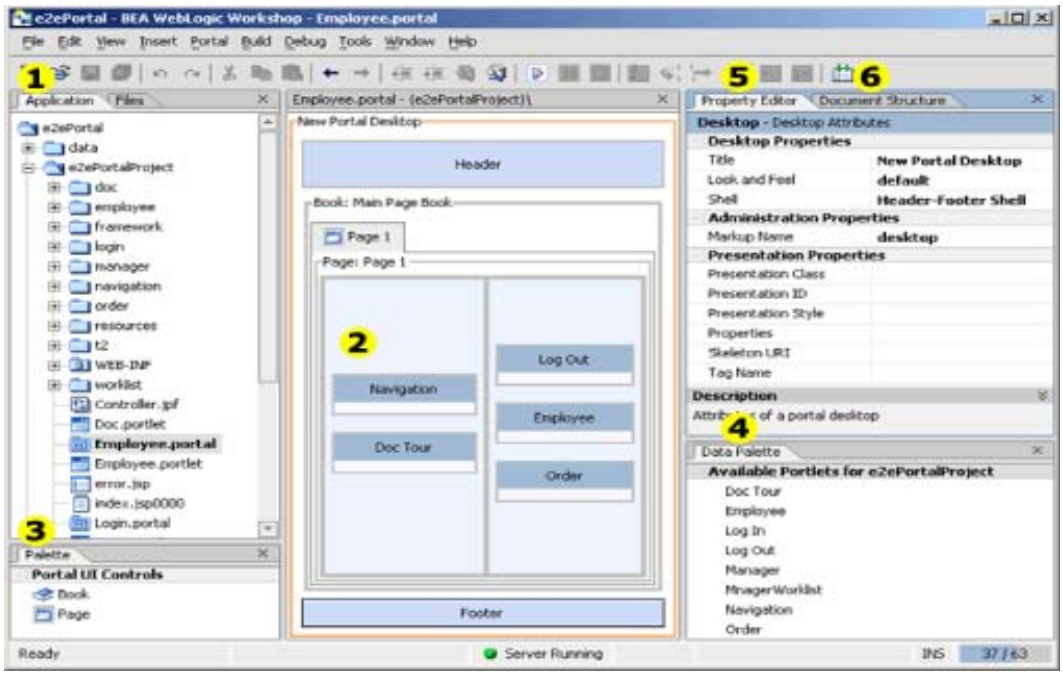

Figure 3: BEA WebLogic Portal Designer

owned and managed, highly tactical applications. Portal services should run on the application server. See Figure 3 above showing WebLogic Portal Designer components including the Navigation Panel (1), Portal Layout (2), Portal UI Controls Palette (3), Data Palette (4), Property Editor (5), and Document Structure (6).

- **Integration Facilities**

  The APS should provide uniform access to integration broker facilities including adapters, message routing, and data transformation. An APS must be able to integrate with a wide variety of packaged application software, component based applications, legacy applications, data sources, and middleware. Integration is usually based on a relatively small amount of custom software. Integration facilities should run on the application server.

- **Process Driven Development**

  The APS should support a uniform model of process driven development. Thus, data and control flow between application components are treated conceptually the same as the business processes that connect business functions (activities). These processes should run on the application server. Few APSs provide this support. WebLogic's Java Control architecture treats

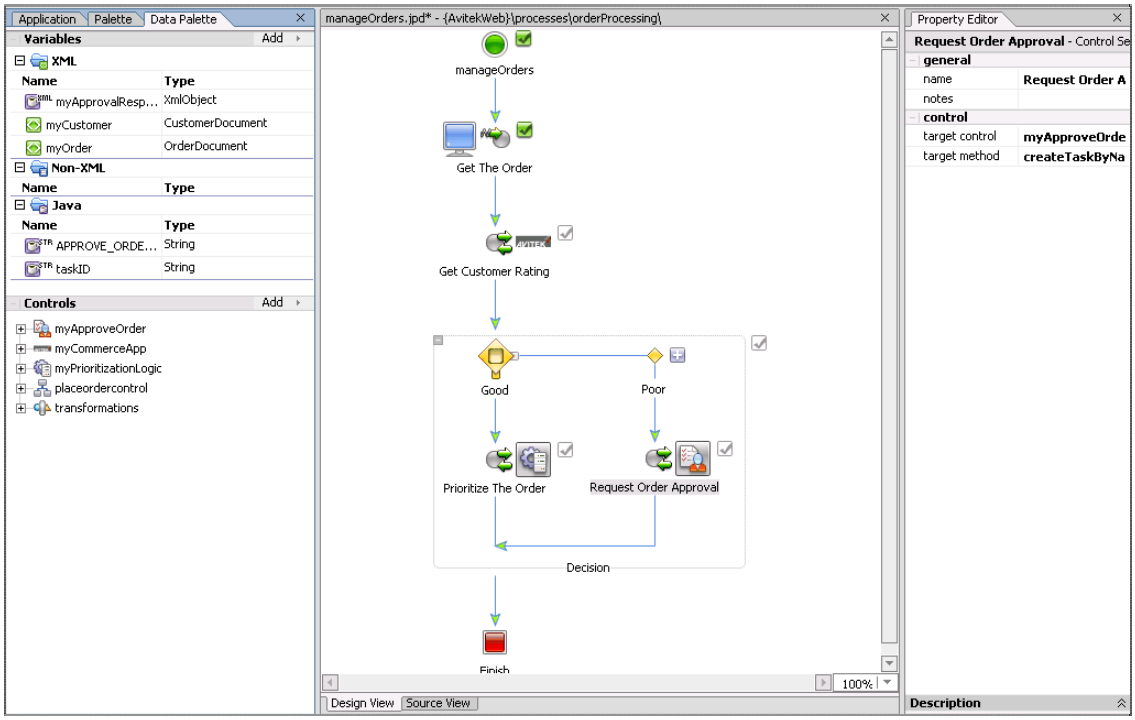

Figure 4: BEA WebLogic Process-driven Development

processes as resources and as a natural extension to the development environment, ultimately enabling the workflow and process needs of departments to be met without additional acquisitions or retraining. Eventually, process driven development will converge with top-down business process management. This goal requires bi-directional editing, which maintains synchronization between code and process diagrams automatically: Changes, whether made in the process diagram or in code, are always consistently represented. See Figure 4.

- **Application Server**

  The APS should include an application server with an IDE and runtime environment. The runtime environment should include distribution services, workload management, transaction management, security, system management, and so on. APSs eliminate the need for developers to handle connection management and pooling, load balancing, transaction management and recovery, data access, etc., in low-level code. This significantly reduces project development and deployment costs, addressing a key concern of departmental IT. BEA WebLogic Server is a well-known, robust application server.

- **Service Oriented Architecture**

  The ability to develop and deploy components as services greatly improves reuse, scalability, reliability, and flexibility. Native APS services should run on the application server, and a means for interacting with external services should be provided. In particular, support for Web Services is becoming increasingly important.

- **Standards**

  Support for standards is crucial to the life expectancy of applications and to preserving investments. While it is not necessary (and may even be undesirable) for an APS to support immature, emerging standards, it is important that the APS provide facilities for migration to those standards when they do mature.

- **Interoperability**

  A variety of component application frameworks now exist, the most popular of which are BEA's WebLogic, IBM's WebSphere, and Microsoft's .NET. Ideally, the Application Platform Suite should permit a degree of interoperability among these. For example, WebLogic provides out-of-the-box support for .NET Web Services. Shallow interoperability will permit components developed under a foreign framework to be invoked from the native suite. Deep interoperability will permit those components to be run and managed on the suites application

server as though they were native components. *This addresses concerns regarding compliance with corporate standards whenever they are adopted.*

- **Incremental Adoption**

  The adoption of an APS for departmental integration and development is greatly facilitated by support for incremental components purchase and installation. For example, with a common, component-based development model, the IDE, integration broker/server, and runtime application server need not all be purchased at once. With WebLogic, the IDE can be adopted initially for development, and integration and portal components added later. *This addresses several key departmental concerns: allowing departmental IT to invest in an enterprise class deployment platform and infrastructure over time, and improving compliance with corporate IT.*

- **Deployment Flexibility**

  The APS should be platform and database independent. Thus standards must be supported, and Java deployment capability would also be advantageous.

- **J2EE Extensions**

  The Java model, despite the many system services offered, does not currently provide direct support for integration. Depending on the approach to a uniform development model, the APS will need to provide J2EE extensions that support that model. How this is done is a key differentiator among APSs, although code annotations are often used. WebLogic provides a unique Java Controls architecture (see Figure 5 and Section 5 below).

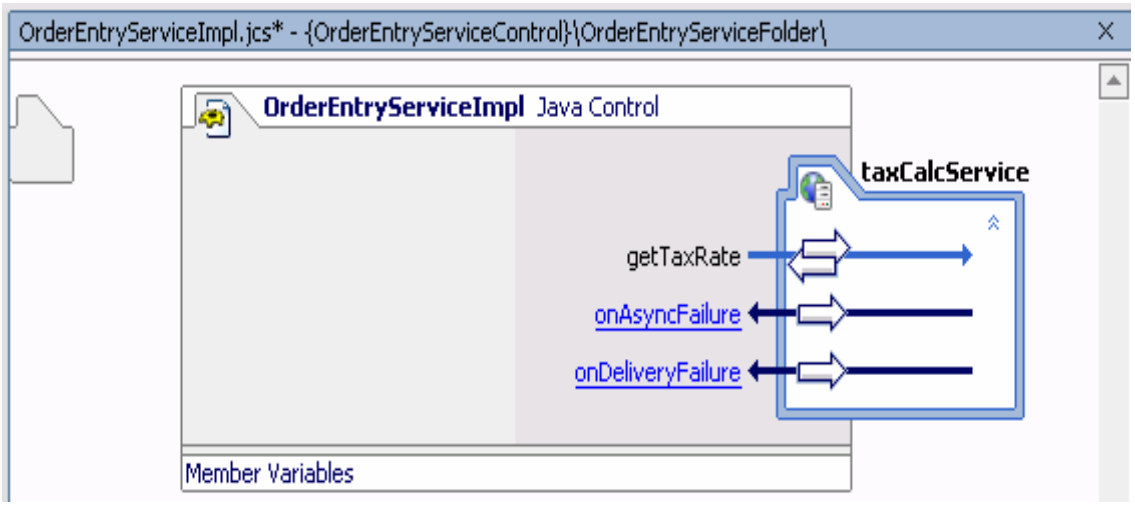

Figure 5: BEA WebLogic Workshop Java Control Example

Key Issues for Departmental APS Adoption

Over time, the reduced costs associated with the scalability, reliability, availability, runtime manageability, development environment consistency, and tight component integration afforded by enterprise APS use will obviously provide a positive return. However, one can be mislead into believing that an APS is overkill for departmental development and integration. This is indeed the case if the APS is not well-integrated, and does not support incremental adoption and a unified architecture. We discuss WebLogic's unified architecture in more detail in the next section.

A recent independent study surveyed actual users of APS solutions and compared the characteristics of a non-integrated APS to those of a fully integrated APS with respect to project cost and productivity.[3] The fully integrated APS was found to deliver total savings in *both* time-to-production and total project costs of 21-23% (see Figure 6). These savings were independent of the size of the projects studied, which the report classified into small, moderate, and complex. By task, the most significant were savings in maintenance, management, and enhancements, which reached 60% for some tasks of this type. Needless to say, maintenance, management, and enhancements are significant cost and time contributors in departmental IT projects, and controlling such costs is important to meeting departmental budgetary limitations. Another important factor for departmental IT projects was the reduction in senior IT staff requirements: 34% savings in Sr. J2EE developer time and 47% for architect time.

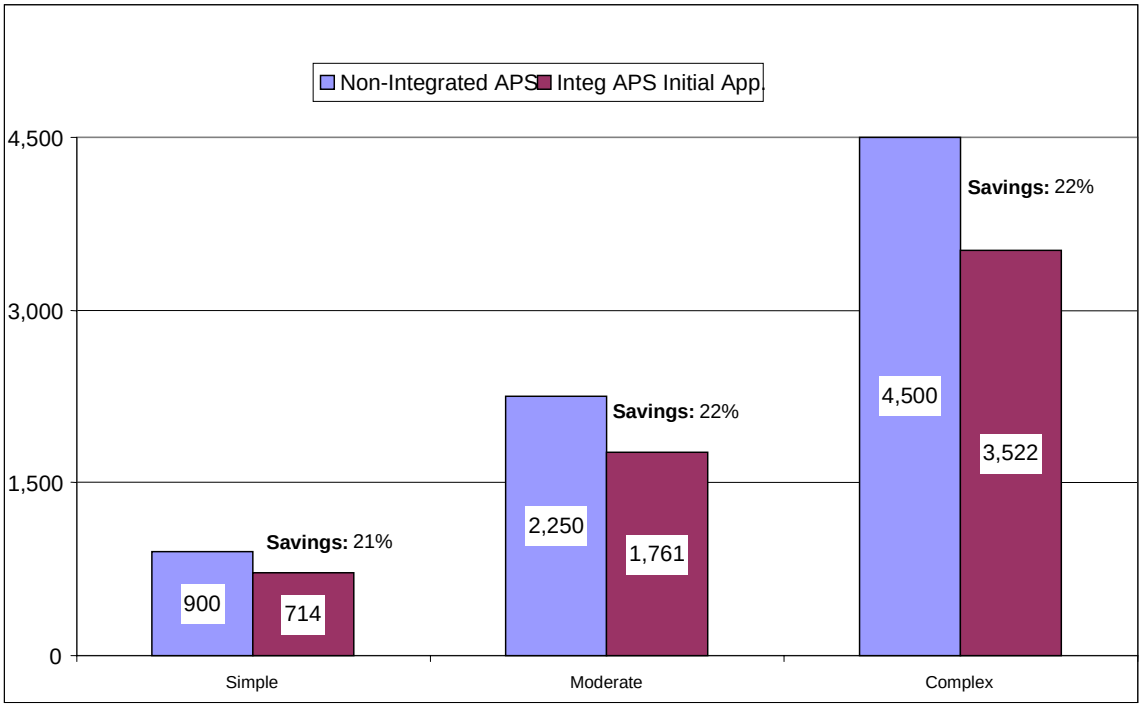

Figure 6: Full Application Lifecycle Savings in Days of Effort[3]

[3] *Application Platform Suites: An Architectural Cost Analysis*, Nov. 2003

By adopting a standards-based Application Platform Suite incrementally, departments can avoid both high upfront investment and the disruption that accompanies the inevitable replacement of proprietary approaches. Component, service oriented frameworks permit a level of reuse and integration that is not otherwise possible. The costs of deployment and management are reduced and the ability to integrate with corporate IT initiatives is greatly enhanced. Furthermore, by using a portal based presentation layer, both departmental and external presentation requirements can more easily be met while preserving departmental autonomy. Integration as well as development costs are reduced, and time to deployment is improved. The result is a much higher rate of return on tactical projects than is possible with customized code approaches.

The high cost of starting over with a new development environment need not be a barrier to adopting a J2EE Application Platform Suite. Vendors like BEA have recognized that not all components of the APS may be needed immediately and so package the components for incremental adoption. The adoption rate can be as incremental or as rapid as is desired. Because of wide JVM, Web services, and portlet support, one can easily begin with the use of the design and development components without committing to an application server until demanded by system management, scalability, or availability requirements. Similarly, a unified development model means that access to resources is uniform, whether those resources are local components or remote applications. It also means that the flows of data and control are treated in a uniform manner, whether it is between program components or activities in a process.

Many IT departments have acquired IDEs, application servers, integration brokers, and portals in an effort to develop their own APS. Combining their capabilities, while not impossible, is a difficult integration task in itself. Furthermore, the lack of a common development model represents considerable development costs and a barrier to use. These issues define the APS integration problem. Even if an IT department were able to solve the APS integration problem, the high cost of acquiring the various components separately, let alone of combining them seamlessly, can prevent departmental IT from doing so. More important, there is little hope that all the components could be made compatible with a single application server or that the requirements of a uniform architecture could be met.

5. A Unified Architecture

As noted above, the degree to which an APS vendor has managed to create a unified architecture and a common development model is a key differentiator among APS products. If this is not done well, the developer will have a sense of moving between distinct facilities with resulting loss of productivity. Furthermore, business logic specific to new applications versus that necessary for integration (so-called “integration logic”) are likely to be created and maintained in different ways. Yet all business logic is about enabling the business, should be consistent, and so should not be separated. Indeed, business logic used for integration today may well be used for new application development tomorrow and so should be maintained in the same manner.

Application development environments have long focused on enabling developers to meet the business requirements for new applications, and therefore to capture business logic. Various methods have been introduced to make the business logic reusable and more easily maintainable. Component-based and object-oriented approaches address the problem by modularizing programs into reusable objects. Some objects are clearly meant to model the business rather than the technical environment, and so their behaviors represent business logic. In a modern J2EE environment, business logic is most often captured in EJBs with an application server as the runtime environment.

When it comes to integration, most traditional application development environments have had little to offer. The issues of providing interfaces between applications, reformatting, transforming, and routing are rarely addressed, especially if these involve EAI or data integration (e.g., ETL) facilities. The result is that, although a developer may use a traditional application development environment to develop an interface to an application (i.e., an adapter or connector), they must step outside the development environment, its concepts, and development model in order to engage in application integration. There is thus a barrier between application and integration that results in development and deployment delays, the need for additional expertise, maintenance costs, and so on.

By contrast with the development focus, application integration environments have long focused on enabling developers to meet business requirements for interconnecting business applications, and therefore to capture what might be called integration logic. Integration logic has little to do with the business rules pertaining to specific business functions and everything to do with those for business processes. Integration logic has traditionally been embedded in various middleware facilities, with the runtime environment most notably comprising a combination of integration brokers, adapters, process or workflow engines, and custom integration code, and quite independent of any application server.

With the advent of Application Platform Suites, the barrier between application and integration environments is finally dissolving. By definition, an Application Platform

Suite makes integration facilities available to developers and development facilities available to integrators.

A specific example of how the APS integration problem has been solved is worth considering. BEA's solution to the APS integration problem is particularly elegant, creating a seamless environment with a *common development model*. The key concept is that of a Java control, an object that presents an arbitrary resource as an ordinary component. In effect, Java controls are wrappers around resources such as data sources, remote applications or adapters, integration facilities, processes, and so on, with property sheets that can be set by the developer. Once a control has been created and registered, it is accessible to any developer and behaves like any other component.

BEA's common development model provides uniformity with respect to:

- Resource Incorporation And Invocation

  New Java controls can be created easily and efficiently. All resources are invoked as Java controls. Java controls can be easily defined to encapsulate any Web Service without significant programming. See Figure 5 above.

- Flow Control

  Flow control, whether between components, applications (encapsulated as Java controls), or workflow activities, can be specified via visual process definition. The approach to process-driven development, in conjunction with Java controls and integration facilities, makes process integration easy. It is compatible with Business Process Management, although the current facilities only address the developer and IT perspectives. Additionally, bi-directional edits keep process diagrams and code synchronized.

- Presentation

  Portals and portlets provide a uniform approach to presentation and content management. In effect, a portal is merely another resource with its own encapsulated behavior and properties.

- Runtime Uniformity

  All elements of a WebLogic application, including integration, process, presentation, and services, enjoy the benefits of an application server as the runtime environment.

A unified architecture and common development model thus converts an integration project into a development project. The result is that integration becomes as natural for departmental development as any other component-based development (see Figure 7).

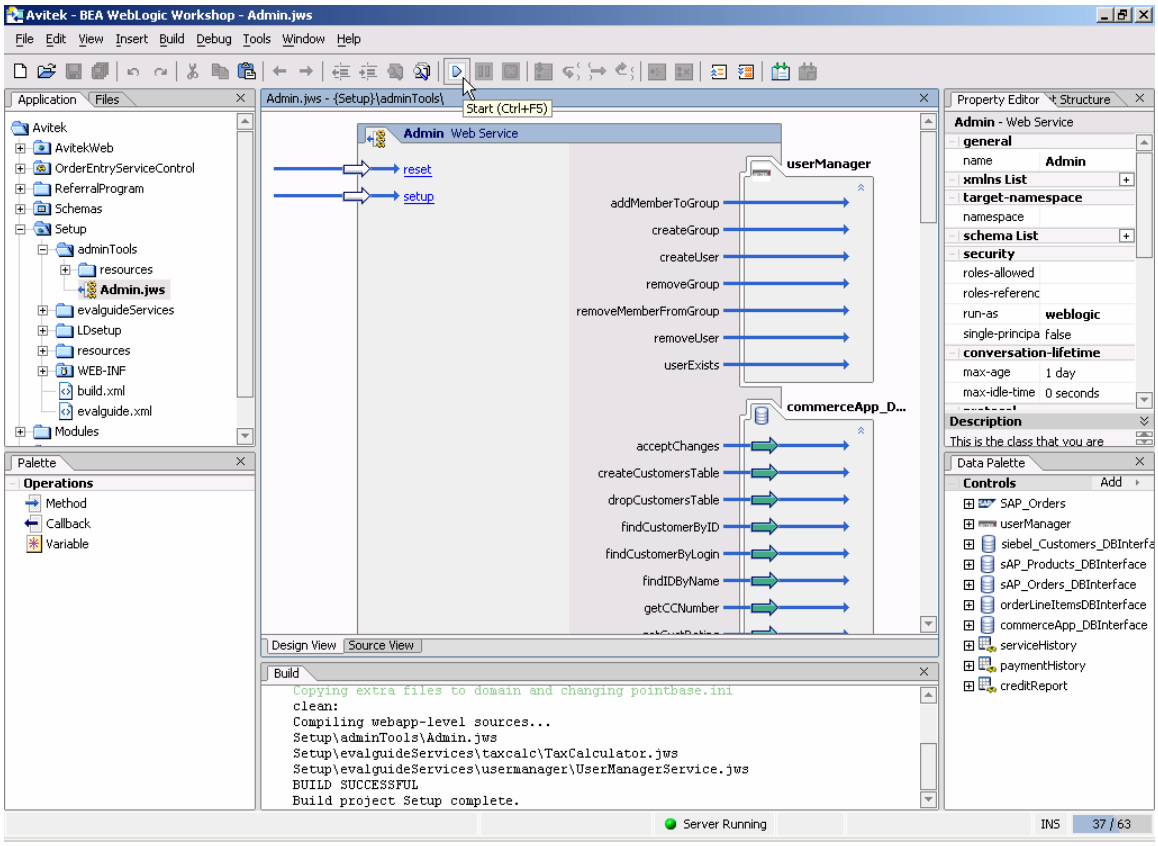

Figure 7: BEA WebLogic Workshop JWS

6. Summary

Departmental applications are no longer as segregated and standalone as they once were. Few tactical operations can be performed in isolation. There is tremendous pressure for departments to provide visibility to internal peers, external partners, and sometimes to customers. In this sense, they are providers of business services through an integrated presentation layer. Just as business logic and presentation logic must be separate, so business logic and integration logic[4] should be unified. In addition, they must satisfy corporate and legal reporting requirements, and be seamless participants in a variety of business processes. The result is a need to support business process integration in addition to the more familiar and local data, application, and workflow integration. Unfortunately, the customized code which so effectively met short term tactical goals a few years ago now creates a barrier to interdepartmental integration and so is no longer an answer. If corporate IT is unable to effectively integrate with departmental solutions, they will eventually restrict the autonomy of departmental IT efforts with the familiar negative effects.

Even without these external pressures, using customized code for integration is merely a quick and temporary fix, being difficult to leverage, intrinsically rigid, and prone to functional failures which can disrupt the departmental operations. When departmental business requirements had several years of stability, this was a tolerable condition in exchange for rapid delivery of functionality. This is no longer the case: Competitive, budgetary, and regulatory pressures are felt by every department, leading to frequently changing business requirements.

Adoption of a J2EE Application Platform Suite for departmental integration and application development is clearly a good choice. As we have seen, the resulting business benefits of adopting a J2EE Application Platform Suite thus include:

- lower integration costs
- accelerated time-to-deployment
- rapid responsiveness to change
- reduced training costs
- incremental commitment and adoption
- compatibility with corporate IT architectures

[4] We recognize that some use integration logic to include integration code that is technology specific, and believe that such code is properly the province of integration services to be supplied by the APS.

- real-time access by extra-departmental users
- scalability
- reliability

Given these benefits, it cannot be argued successfully that a J2EE Application Platform Suite such as BEA WebLogic is only an enterprise, strategic approach to integration and development. Rather, adopted incrementally, such APS products support tactical, functional, departmental needs just as well.

About the Author and Alternative Technologies

David McGoveran is the President and Founder of Alternative Technologies, an independent analyst and consulting firm founded in 1976. Mr. McGoveran has been a pioneer in the definition, technical architecture, and uses of relational databases, distributed applications, integration, and Business Process Management Systems. He has helped define marketing strategies, products, and technical direction for companies such as Hewlett-Packard, IBM, Candle Corporation, Microsoft, and many others. He has been lecturing and writing publicly on the topics of EAI and BPMS since 1997. He is Senior Technical Editor and co-founder of the Business Integration Journal – formerly the eAI Journal, in which appears his monthly column *Enterprise Integrity*. Mr. McGoveran provides consulting and teaches seminars on Business Process Management, as well as other topics.

For pricing and availability of reports or other products and services (including consulting and educational seminars), please contact Alternative Technologies directly at (831) 338-4621 or via email addressed to mcgoveran@AlternativeTech.com.

Alternative Technologies
6221-A Graham Hill Rd., Ste #8001
Felton, California 95018
Telephone: 831/338-4621 FAX: 831/338-3113
Email: mcgoveran@AlternativeTech.com
Website: www.AlternativeTech.com