



## J2EE and EAI - A Primer for the practice



WHITE PAPER

Kevin Rodrigues  
April 28, 2009

### Executive Summary

The use of technologies such as Web Services, Enterprise Service Buses (ESBs) and Business Process Management tools (BPMs) are often the impulsive answer from management's perspective to the problem of Enterprise Application Integration (EAI). Tibco Business Works and iProcess, Websphere ESB and Process Server or other technologies such as Jcups, Webmethods, etc. are viewed as the silver bullets to the problem of EAI. The use of these products might indeed be appropriate on the one hand but may be an overkill on the other.

There have been instances where existing enterprise application integration has been achieved partly by integrating data at the database level through dblinks and materialized views - only to find the system so brittle that it couldn't handle synchronization of data when records are deleted. Consequently, a separate application needed to be written to handle synchronization of data on deletion, adding to the conglomerate of applications to be integrated. In addition, there are converse instances where EAI has been used where simple ETL tools or data replication might have sufficed.

The real cause of failure of EAI projects lies in the lack of understanding what 'Enterprise Application Integration' and more importantly, what 'integration' fundamentally entails.

This paper intends to explore these very fundamentals of EAI in the realm of the good old J2EE, explore the very nature of integration and dissect the concept of EAI.

## Table of Contents

1. DOCUMENT CONTACT INFORMATION .....	2
2. EXECUTIVE SUMMARY .....	2
3. INTRODUCTION .....	2
4. THE EVOLUTION OF INTEGRATION .....	2
5. LEVELS OF APPLICATION INTEGRATION .....	4
6. MIDDLEWARE TECHNOLOGIES FOR APPLICATION INTEGRATION .....	4
7. J2EE - AN INTEGRATION PLATFORM .....	5
8. INTEGRATING THE ENTERPRISE - THE BASIC STRATEGY .....	5
9. CONCLUSION .....	8

### 3. Introduction

Data or Information is the bloodline of any organization and accessibility to information is what applications provide. Stand-alone applications provided the much-needed data and therefore mushroomed within an organization's IT space. Data, however, often needs to flow across applications to enable more precise and real-time enterprise wide decisions. Data flow across applications is governed by workflows that are in turn governed by the so-called business processes and EAI is what is required to achieve this flow of data.

Data flow across applications is cast in myriads of problems. Over a period of time each stand-alone application becomes legacy. Compounding this is the fact that applications - modern as well as legacy, have their functional requirements changing periodically and this has ramifications on the flow of data across applications. Data belongs to different domains - HR data and Payroll data of an organization often participate in a workflow. Data could be sourced from and targeted to different architectures. Extraction, Transformation and Loading often lead to redundancy of data.

The fast pace of commercialization of the world has placed an additional demand on the flow of data. Data not only now needs to flow across applications within an organization - it also needs to flow outside, across organizations, allowing for Business to Business Integration; perhaps coercing the need for standardizations as part of the EAI scenario.

Integration, and the problems it presents, has not been exclusive to the domain of enterprise level applications. It has been all pervasive at various levels right from monolithic stand-alone applications, to two-tier applications to multi-tier applications to distributed applications to enterprise applications to eCommerce.

It therefore follows that one needs to investigate the evolution of integration at each of these chronological levels.



### 4. The evolution of integration

#### Monolithic Applications

Almost all legacy applications are monolithic. Nevertheless, at some early point of time (perhaps in the 1980s) these were no legacy systems. They were state of the art systems. Growth of the enterprise soon began to depend upon integration of these applications. The appropriate way then was to do direct data level integration where all applications had their own local data and the data that they needed to share was stored in a centralized repository for all to access. This approach though widely used had its shortcomings - firstly, all applications were cast to the same database and secondly, any change in the centralized data had a ripple effect on the rest of the applications.

Every application was architected in isolation, not as part of an enterprise wide information system. The choice of technologies for the stand-alone application typically was left to the individual developer. In the context of an enterprise information system, an organization wide dictat on the choice of technologies did not prevail.

## Two Tier Applications

Almost all monolithic legacy applications were mainframe applications. The arrival of the PC paved the way for a variety of platforms, technologies, languages and architectures. The PC substituted the dumb terminals of the mainframes (or of even other systems such as PDP-11/RSX-11M+, VAX/VMS, XENIX, UNIX, etc) as new clients, now capable of 'client side processing'.

All presentation logic and business logic was coupled in the client side (thick clients), and the server simply constituted of the database. Thereafter, the business logic was then separated from the presentation logic and pushed from the client side, onto the server side, in the form of stored procedures - thus transitioning from a thick client to a thin client.

These co-existing two-tier and monolithic applications, however, soon needed to be integrated. Each of these applications inadvertently had some functionality that was common across them and hence there was some duplication of functionality. This duplicated functionality operated upon similar data (though not identical) and instead of sharing data of one application simply by extending its attributes for the other application, separate data sets of similar data with different formats took shape. This introduced data maintenance nightmares, as there was not a single version of the truth.

Decision support systems needed to analyze large amounts of this structured data residing across applications and the quick way out was to extract, transform and load (ETL) most of the data into data warehouses and do the analysis off that data warehouse.

To resolve the issue of duplication of functionality, Enterprise Resource Planning (ERP) systems made their way and they intended to replace existing business applications - lock, stock and barrel. A single vendor would provide a complete suite of applications that took care of every aspect of the desired business functionalities, using a shared database as the enterprise integration style. The ERP solution, however, had to be generic to cater to a variety of business needs and businesses itself. They could not address the requirements of every organization in totality - every business had its own facet with its own idiosyncrasy. This only reinforced the need to integrate existing systems with these ERP systems, defeating the very purpose of the ERP systems itself. Ironically, ERP systems still exist today because even the new entrants such as SAP, PeopleSoft, etc. solve only a fraction of the business functions required by a typical enterprise.

## Multi-Tier Applications

Monolithic and Two Tier Applications could be dissociated into separate well-encapsulated, functionally re-usable sub applications called 'components' or 'services', each component offering fine-grained services through a well-defined interface (Think of EJBs). Each component or service could then run on a distinct platform and therefore needed to integrate with one-another to represent the main application that they dissociated from. Because each service could run on a separate platform, the services needed a container to run in (Think of EJBs requiring an Application Server).

J2EE was the framework that addressed the complete gamut of integration issues. However, J2EE restricted itself to integration of services based on the Java platform only. It had to adapt itself to integration with other platforms and technologies. RMI had to move on to RMI/IIOP. EJB and JavaIDL had to take shape. JMS and JCA had to have a role to play. Along with EJB, CORBA and .NET were the only dominant technologies offering multi-tier distributed architectures.

The 'service' based approach of multi-tier distributed sub applications naturally lent itself to Service Oriented Architectures (SOA) whereby the main applications (Application Servers, for example) themselves could be modeled as services that need to be integrated - their fine-grained services being combined into coarse-grain services. While JMS architecture was suitable for EAI, it allowed for integration via the back-end. In some instances, integration to the Application Server via the front-end was required and Web Services is the technology that came into play. Riding on these basic application integration technologies, came higher-level tools in the form of Enterprise Service Buses such as OpenESB, Tibco, Jcaps, AquaLogic, etc.

Distributed architectures provide many benefits such as re-usability, encapsulation, scalability, loose coupling, better reliability, quicker development, etc., but they do come with their disadvantages of poorer performance, security risks and the nightmare of application integration deteriorating into a mesh.

## 5. Levels of Application Integration

From the above discussion, it can be viewed that integration can occur at various levels .

- Hardware & Operating System Level
- Data Level
- Application Interface Level
- Business Process Level
- Presentation Level
- B2B Level

**Hardware and Operating System Level Integration** is at best left to the specific vendors, but data level integration is best solved when one starts by viewing the complete organization wide data through one single unified data model. This is how the key ERP vendors address EAI.

**Application Interface Level** is typically achieved by using middleware such as message-oriented middlewares, object request brokers or remote procedure calls. The key to achieving this lies in the fact that services or components are modeled such that they are accessible only through the interfaces that they expose. Such integration solves only the technical aspects of integration to achieve interoperability, not business level problems.

**Business Process Level Integration** becomes possible only if one views it not as a technical problem but rather a business problem involving workflows or processes. The problem is defined at a very high level of abstraction and is solved by using different layers of technologies superimposed on various enterprise applications.

**Presentation Level Integration** examples would be Web Browsers, Web Servers, JSPs and Servlets, where users get a unified view to information contained in disparate backend systems. Portals and mashups too are a good example of presentation integration at an even higher level of abstraction.

**B2B integration** is built upon EAI integration and is very closely related to it. Web Services are the gateways for B2B integration.

## 6. Middleware Technologies for Application Integration

While there are various levels at which integration can occur, there is a variety of corresponding middleware available to achieve integration.

### Persistence Tier

At the persistence tier, there are database integration technologies such as Java Data Objects and JDBC for java platform or Active Data Objects and ODBC for Microsoft platform. They provide a layer of abstraction over the actual database - be it Oracle, DB2, Flat Files or an XML Database.

Transaction Processing Monitors too are a technology used by legacy systems for integration of applications at the persistence tier.

### Business Tier

At the business tier, there is Message Oriented Middleware (MOM) and Remote Procedure Calls (RPC) to manage interoperability and communication between distributed heterogeneous platforms. The basic difference between MOM and RPC is that in MOM, the applications access the Message Oriented Middleware through APIs. There are APIs on the client side as well as on the server side. The communication is asynchronous.

In RPC, the client application uses a proxy of the server application and invokes calls as though the server application is local to its address space. It is synchronous in nature.

The asynchronous nature of MOM, however, makes the server application susceptible to overloading by flooding it with messages.

Object Request Brokers (ORBs) are also a business tier level integration middleware and are architecturally different from MOM and RPC in that stubs on the client side act as the glue between the client application and the orb and skeletons on the server side act as the glue between the orb and the server application.

These stubs and skeletons are pieces of code that get compiled and linked to the respective client and server applications. They are generated by compilers run over platform neutral code, which define the interfaces that the server application provides. For e.g. idl2cpp and idl2java are compilers that compile an IDL file (written in IDL - interface definition language) and the output is the relevant stubs and skeletons in C++ or Java code respectively.

The ORB itself runs over GIOP/ IIOp protocol. CORBA, Java IDL, RMI/IIOp and Microsoft COM/DCOM and COM+ are some predominant ORB technologies.

Another integration technology at the business tier is the Application Server, which mainly provides a container for execution of business components, and the container

manages all the communication aspects of the business components. EJB is a prime example. Application Servers are more of an assembly of basic middleware technologies along with its own extensions.

#### Presentation Tier

At the presentation tier, there are the Web Servers and Load Balancers. Some transaction processing monitors do provide load balancing and security features and thus may be considered a presentation tier integration technology.

## 7. J2EE - An Integration Platform

J2EE lends itself as an excellent integration platform. Applications that need to be integrated typically fall under the following categories:

- Modern Java Based Applications
- Modern Non Java Based Applications and
- Legacy systems

#### Integration with Modern Java Based Applications

J2EE provides support at various tiers:

- At the Presentation Tier, J2EE provides support for simple java applications, applets, thin clients and browsers.
- At the Web Tier, it provides Servlets and JSPs.
- At the Business Tier, it provides EJBs.
- At the Persistence Tier, it provides JDBC.

It provides additional technologies such as JAXP, JAXB, JCA, JTA, JAAS, JNDI Java Mail, JMS, etc. that can be used at both - the Web Tier and the Business Tier as well.

#### Integration with Legacy Systems

The typical technologies that J2EE provides and that can be used for integration with legacy systems are:

- JDBC, JDO and XML (for data access and exchange)
- XSL (for data transformation)
- JMS (for message broking)
- Java Connector Architecture - JCA (for integration with Enterprise Information Systems (EIS) )
- RMI, RMI/IIOP (for distributed object support)
- EJB (for component support and integration)
- SOAP (for integration via web services)

#### Integration with Modern Non-Java Based Applications

For integration with modern, non-java based applications, J2EE supports many CORBA based technologies such as:

- IIOP
- JTA and JTS based on CORBA Object Transaction Service
- EJB based on CORBA Component Model and
- Direct access to a CORBA Naming Service via the orb

It also supports Microsoft technology integration via:

- SOAP
- COM Automation bridges (e.g. JacoZoom)

## 8. Integrating The Enterprise - The Basic Strategy

Having understood the evolution of integration, the basic levels of integration, the various middleware technologies for integration and how J2EE provides a platform for integration, we can now move closer to the subject of enterprise level application integration.

Being aware of the basic levels of integration and the various middleware technologies available for the same, and as described above, is only part of the integration story. Any integration process has many hidden complexities and therefore, has to be a well-defined integration strategy in place.

#### Extension and Replacement

The simplest and most naïve strategy would be to replace existing systems with new systems and extend the functionality of some existing systems. Replacement is an extremely costly solution and extension is very limited in nature. Neither provides a global level integration architecture.

#### Global Level Integration Architecture

Since extension and replacement are not viable solutions, it follows that existing applications need to be integrated as is - with all their quirks, idiosyncrasies, irregularities, inconsistencies, singularities, etc. A global level integration architecture is required, which means that the integration problem has to be resolved using a top-down approach.

A sound integration infrastructure comprising a set of appropriate middleware technologies, collectively called the '*Integration Broker*', also known as an '*Enterprise Service Bus*', is the basis of such an architecture.

To facilitate integration, however, the broker requires that all applications adhere to contracts. Every

application is required to define and expose its services as an 'interface'.

Every application that requires to be integrated can then be coupled to a 'low level proxy counterpart'. These low-level proxy components, on one side, will provide the contract/interface that the integration broker expects and on the other side, it will have code to 'connect/adapt' itself to the actual existing component. In short, we are introducing a proxy component as a layer between the actual application to be integrated and the integration broker to make the existing application amenable to the integration broker. See Figure 1.

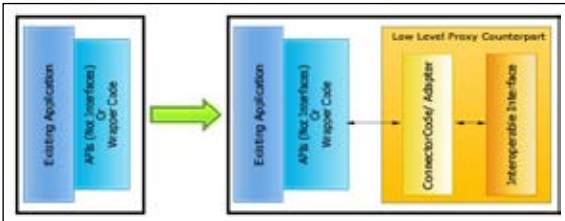


Figure 1.

This pattern can then be applied at the persistence tier as well as at the business tier. But Data Level Integration is best avoided because the various validations and other associated rules that govern the reliability and integrity of data, which typically reside in applications that access the database, get bypassed.

EAI is best achieved at the Application Interface Level and this typically is in the business tier.

This also implies that all existing applications that are connected to by its representative proxy need to expose APIs so that these proxies can link up with them via their connector/adaptor code. Some applications may, however, not be API based and in such cases, their source code will have to be changed to ensure that they do expose APIs. We employ wrappers to expose these APIs for integration as shown schematically above.

In cases where the source code may not be modifiable, as in the case where integration may happen via screen scraping, then the user interaction must be simulated and an application bridge may be used, as shown below in Figure 2.

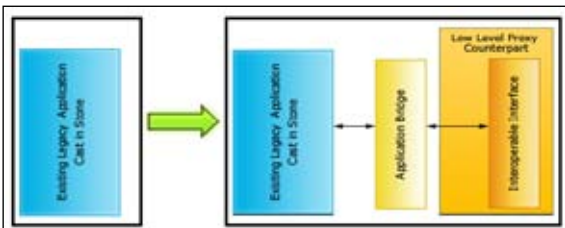


Figure2.

**B2B By Proxy Extension**

Typically, these existing applications that need to be integrated expose low-level fine-grained APIs that are

used for Enterprise Application Integration at the business tier. These low-level proxies are typically implemented using JMS, EJB, CORBA or RMI/IIOP.

The low-level proxy components can themselves be aggregated into high-level proxy components and the pattern thus extended to achieve coarse-grained business level methods thus paving the way for B2B integration. These high-level proxy components can be considered to exist at a new tier, which we can call the Web Services Tier, because Web Services is the standard technology, used to achieve B2B integration.

**Shared Functionality**

Quite often different applications implement the same functionality. For e.g. a billing application might validate a user's address and the shipping application may also do the same. As part of the integration strategy, such common or shared functionality should be isolated from within these applications and be placed in a separate service itself which provides an interface through which its functionality can be accessed indirectly through the integration broker. See Figure 3 below.

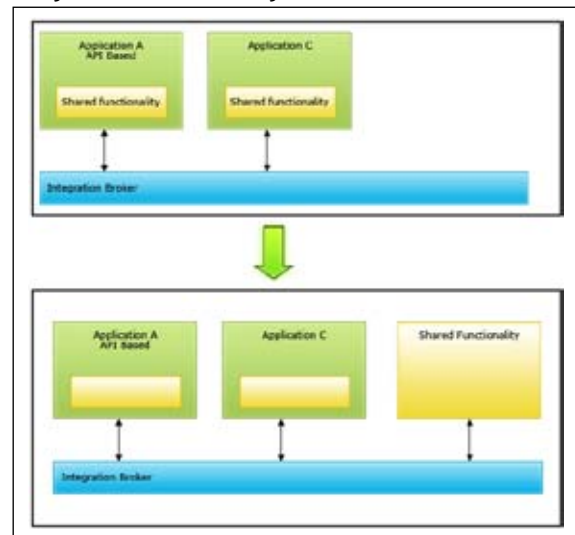


Figure 3.

**Java Connector Architecture**

An application bridge and proxy combination makes an existing application amenable to become integrated via an integration broker. It is worth mentioning that, however, if a legacy application (EIS in particular) needs to be integrated directly to a J2EE application, without the use of an integration broker, then Java Connector Architecture (JCA) is the technology of choice.

Legacy systems need to integrate with the J2EE container

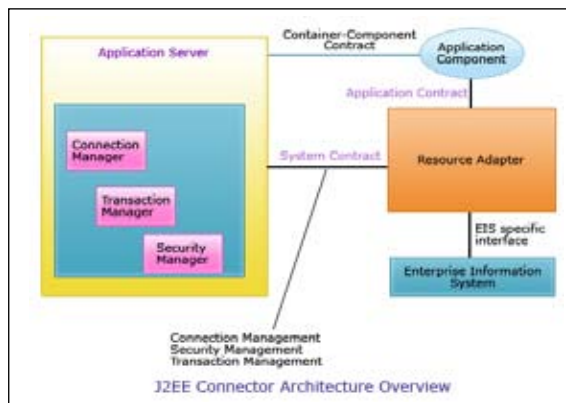
and therefore JCA specifies that the legacy system provide a resource adapter to be deployed within the J2EE application server and this resource adapter necessarily implement the specified system level contracts between the J2EE application server and the adapter. As per the contract, the adapter must provide:

- Connection management to allow the application server to pool connections to the underlying EIS systems
- Transaction management to allow the application server to commit or rollback transactions across multiple resource managers
- Security management to reduce security threats to the EIS system

Over and above these system level contracts, the adapter must also provide component level contracts . The adapter provides this contract in the form of a Common Client Interface (CCI), which enables application

components on the J2EE side to drive interactions with the EIS systems on the adapter side.

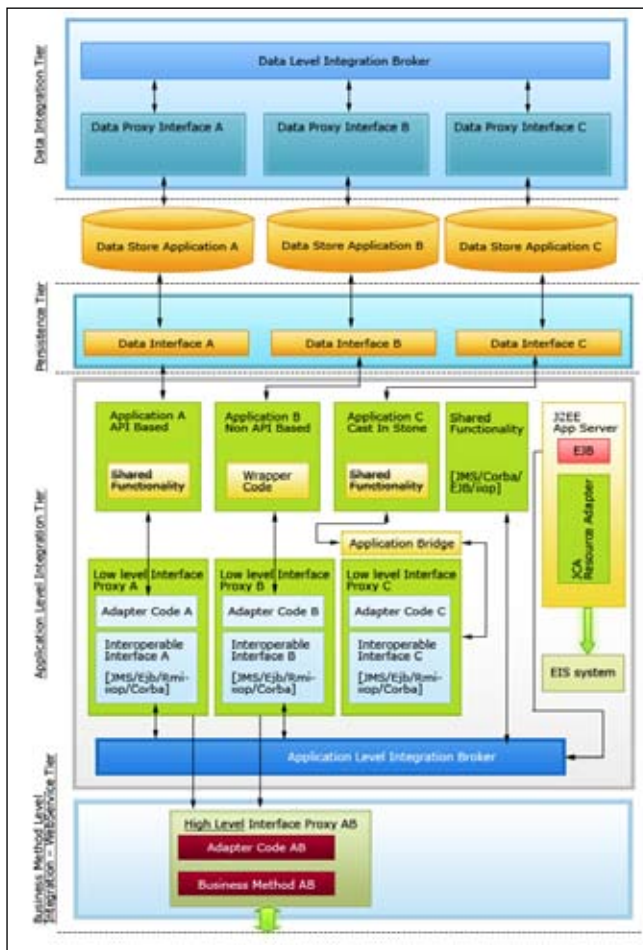
In essence, since the J2EE application server does not have control over the EIS application, it enforces the EIS



Source : <http://java.sun.com/j2ee/connector/overview.html>

### Putting It Together

This basic pattern for data level integration, application level integration and Business Method level Integration is diagrammatically expressed as below.



## 9. Conclusion

While we have discussed the core nature of integration above, we need to move one-step closer to the real world implementation of an enterprise integration solution. We need to analyze the 'flow' of messages in an integrated system and bridge the gap between high-level integration vision and the actual implementation of the integration solution. EAI Patterns will help us bridge this gap.

While one may be tempted to address the variety of EAI patterns in this whitepaper itself, to preserve continuity of the topic, that has been best avoided so to maintain brevity of this already not too brief paper. One could refer to this topic in another, as of now deferred, whitepaper titled 'EAI - Patterns For The Practice'.

As a senior architect belonging to the Integration practice at Mphasis, there is a need to provide a comprehensive and coherent technical perspective of what Enterprise Application Integration entails. Though this paper is not exhaustive, it has attempted to do so. If it can be used a jumpstart or even as a basis for reference models that can be used in RFPs and other integration projects underway within the organization, then its purpose would have been solved.

## Contact us

### USA

460 Park Avenue South  
Suite #1101, New York  
NY 10016, USA  
Tel.: +1 212 686 6655  
Fax: +1 212 686 2422

### UK

88 Wood Street  
London EC2V 7RS,, UK  
Tel.: +44 20 85281000  
Fax: +44 20 85281001

### Australia

410 Concord Road  
Rhodes, NSW 2138, AUS  
Tel.: +61 290 221 146,  
Fax: +61 290 221 134

### INDIA

Bagmane Technology Park  
Byrasandra Village,  
C.V. Raman Nagar  
Bangalore 560 093, India  
Tel.: +91 80 4004 0404,  
Fax: +91 80 4004 9999

## About Mphasis

Mphasis is a leading Applications, Infrastructure Technology, and BPO services provider.

The company delivers real improvements in business performance for clients through a combination of technology know-how, domain and process expertise. With currently over 36,000 people, Mphasis services clients in Financial Services, Healthcare, Communications, Media & Entertainment, Transportation & Logistics, Energy & Utilities, Consumer & Retail, and Governments around the world. To know more, visit [www.mphasis.com](http://www.mphasis.com).

Mphasis and the Mphasis logo are registered trademarks of Mphasis Corporation. All other brand or product names are trademarks or registered marks of their respective owners. Mphasis is an equal opportunity employer and values the diversity of its people. Copyright © Mphasis Corporation. All rights reserved.

