CHAPTER 3

Enterprise Application Integration

3.1 Introduction

Many organizations have been forced to reorganize their business processes by using heterogeneous technologies in order to remain competitive in a business world. The integration and collaboration of business processes across multi-organizations are becoming more and more important in enabling business applications such as enterprise application integration (EAI) [17].

3.1.1 Need of EAI

Enterprise computing has progressed enormously in just the last few years. Especially with the advent of the Web, not only is it possible for diverse organizations to automate and integrate their businesses and computer operations, it is imperative that they do so. Suddenly, as more and more corporations become Web enabled and find themselves relying on a myriad of applications, the ability to evolve and integrate existing applications becomes significant.

Enterprises frequently find themselves having to merge with other enterprises, reorganizing their internal structure, and adopting new technologies and platforms as they strive for competitive advantages.

Furthermore, it is not a simple matter for an enterprise to discard its existing applications, or even overhaul its established business processes, to effect a change in its business model. These kinds of changes are financially expensive to undertake and daunting in terms of human resources. Many enterprises cannot afford to make such
changes or discard existing systems. Thus, it is critical for enterprises to be able to leverage their investments in their existing enterprise infrastructure and applications. In these situations, enterprise application integration assumes a great importance. Enterprise application integration (EAI) enables an enterprise to integrate its existing applications and systems and to add new technologies and applications to the mix. EAI also helps an enterprise to model and automate its business processes [18].

3.1.2 Enterprise Application Integration

EAI is the unrestricted sharing of data and business processes throughout the networked applications or data sources in an organization. Early software programs in areas such as inventory control, human resources, sales automation and database management was designed to run independently, with no interaction between the systems. They were custom built in the technology of the day for a specific need being addressed and were often proprietary systems. As enterprises grow and recognize the need for their information and applications to have the ability to be transferred across and shared between systems, companies are investing in EAI in order to streamline processes and keep all the elements of the enterprise interconnected.

There are four major categories of EAI:

- **Database linking**: databases share information and duplicate information as needed.
- **Application linking**: The enterprise shares business processes and data between two or more applications.
- **Data warehousing**: Data is extracted from a variety of data sources and channeled into a specific database for analysis.
- **Common virtual system**: the pinnacle of EAI; all aspects of enterprise computing are tied together so that they appear as a unified application [19].
As the need to meet increasing customer and business partner expectations for real-time information continued to rise, companies were forced to link their disparate systems to improve productivity, efficiency, and, ultimately, customer satisfaction. The need for IT systems to communicate within an organization led to the evolution of enterprise application integration (EAI).

EAI is the process of creating an integrated infrastructure for linking disparate systems, applications, and data sources across the corporate enterprise. The very origin of EAI solutions can be linked to the need for providing a full duplex, bi-directional solution to share seamlessly and exchange data between ERP, CRM, SCM, databases, data warehouses, and other important internal systems within the company [20]. (Fig. 3.1)

3.1.3 Characteristics of EAI

At the core of EAI is the need to share and manage information, but the method of doing this could be complex and varied. The driving force of any EAI approach is to extend useful information to all and to avoid information islands and bottlenecks. Green has identified five such characteristics [21]:

1. EAI systems are directed towards integration at the business level - they can include all business and data processes.
2. EAI methods re-use and fully distribute all the applicable business processes and data.

3. EAI methods involve no real understanding of specific system functions to integrate the application. This ability to integrate with minimal understanding of specific functions is due to the technology's focus on the user side of application, which require no system knowledge.

4. EAI technology does not require source code or code administration rights to any of the target application.

5. EAI technology generally requires no changes to the existing hardware infrastructure.

3.1.4 The Challenges of Integration

While every organization is unique in its use of products, data formats, naming standards and hardware, the challenges of integration are same [22]:

- Data formats
- Changing infrastructure
- Unique business processes
- Heterogeneous systems
- Data integrity requirements
- The Future unknowns

Enterprise application integration is nothing new to financial institutions. Banks, broker/dealers, and other financial institutions for years have been moving data between numerous complex, disparate legacy systems though middleware. Historically, EAI solutions act as Hubs to transport data between applications, simplifying the technical environment by reducing the number of bilateral interfaces between applications. EAI is typically viewed as a technology infrastructure solution, and for IT to recognize the most value out of it, large, costly enterprise implementations have been the norm. Typical EAI implementations within financial institutions have had a price tag of more than $1 million. As with most other markets,
the economic bust of the last few years triggered the end of an era of plentiful IT spending. As the economy slowed, companies dramatically cut back on IT spending, tightening their purse strings on discretionary spending or shutting it off all together. EAI projects began to feel the pressure. This economic environment has and will continue to change the way financial services organizations make EAI decisions. The new economy is bringing about a new rationale and approach that the FI community will use in defining, purchasing, and implementing EAI. This economic environment has brought to light a number of challenges EAI solutions will have to address immediately to succeed [23].

3.1.5 EAI Requirements

Enterprise EAI requirements consist of:

**Business-User requirements:** Integrated enterprise applications are tools as used by business users to create, deploy and monitor their workflows and business processes. A decision-maker at a major manufacturing supplier, for instance, needs to set up a detailed workflow that connects customers, suppliers, sales force, inventory control systems and other services. Business-users need the drag-and-drop simplicity of connecting pre-built Enterprise Services within a service-oriented framework that can provide real-time information reliably.

**Developer requirements:** Whether application integration is outsourced to system-integrators or delivered by in-house resources, developers need standards-based solutions that can be easily deployed – with an increasing preference for remote administration. Security, both inside and outside corporate firewalls, continues to remain a key concern. Ideally, developers need to be able to modify or add new services to dynamically modify and extend solutions for their business-users without any application downtime. Addition of new users or partners with appropriate access controls needs to be trivially simple [24].

23
3.2 Logical EAI Architectures

EAI combines separate applications into a co-operating federation of applications. Two logical integration architectures for integrating applications exists [25]:

- Point-to-point integration
- Middleware-based integration

3.2.1 Point-to-point integration

EAI developers pursue point-to-point integration because they find it easy to understand and quick to implement when they have just a few systems to integrate. A point-to-point integration example: One application makes direct JDBC (Java Database Connection) calls to another application's database tables. Initially, when you integrate two applications, the point-to-point integration solution seems like the right choice; however, as you integrate additional applications, you get a situation shown in Figure 3.2.

Considering all that, point-to-point integration's infrastructure proves brittle. Each application is tightly coupled with the other applications through their point-to-point links. Changes in one application may break the applications integrated with it. As another disadvantage, the number of integration points require support. If you have five applications integrated with each other, you will need 10 different integration points, as Figure 3.3 illustrates. As such, each additional application becomes harder to integrate and maintain.

To avoid that problem, we need an intermediate layer to isolate changes in one application from the others.
3.2.2 Middleware-based integration

Middleware has stepped up to the task of providing a mediation point between applications. Middleware provides generic interfaces with which all integrated applications pass messages to each other. Each interface defines a business process...
A service-oriented architecture lets you add and replace applications without affecting the other applications. If you have five applications to integrate, you’ll have just five integration points. Compared to the point-to-point solution, middleware-based solutions easily support numerous integrated applications and require less maintenance. In addition, middleware can perform complex operations—transforming, aggregating, routing, separating, and converting messages—on the data passed from application to application(s). The only downside: the added initial complexity of setting up the middleware and converting existing applications to use the middleware APIs.

### 3.3 Levels of Integration

Once you have selected the logical EAI architecture, you next must choose the integration Level [25].

- Data-level integration
- User interface (UI)-level integration
- Application-level integration
- Method-level integration
3.3.1 Data-level Integration

With data-level integration, you integrate the backend datastores that the integrated applications use. Data-level integration can be push- or pull-based. With push-based, one application makes SQL calls (through database links or stored procedures) on another application's database tables. Push-based data-level integration pushes data into another application's database. In contrast, pull-based data-level integration utilizes triggers and polling. Triggers capture changes to data and write the identifying information to interface tables. Adapters can then poll the integrated application's interface tables and retrieve the pertinent data. You'd use pull-based data-level integration when an application requires passive notification of changes within another application's data.

Use data-level integration when the application up for integration does not provide any APIs or client interfaces, and you intimately understand how business operations affect your application's data model. Data-level integration typically represents the only option with custom applications lacking application APIs.

In data-level integration, changes propagated from dependent systems bypass the integrated application, so all inserts, updates, and deletes are done to data that the integrated application accesses. Developers typically implement data-level integration with database gateways or triggers and stored procedures. The major downside: keeping the integrated application's data intact. For example, some ERP (enterprise resource planning) systems include thousands of tables. One table might have dependencies to others, and the integrated application may be the sole enforcer of those dependencies.

3.3.2 UI-level Integration (Refacing)

UI-level integration ties integration logic to user interface code. UI-level integration is scripting- or proxy-based. Scripting-based UI-level integration embeds integration
code into the UI component events, common with client/server applications such as PowerBuilder or Vantive. For example, when you click the Submit button of an Add Customer screen, data must be sent to the application's database and a JMS (Java Message Service) topic. Proxy-based UI-level integration uses the integrated application's interface (through screen scraping) to pass data to and from the legacy system.

Use UI-level integration when you cannot easily or directly access the database, or when your business logic is embedded in the user interface. Mainframe and client/server applications represent typical candidates for UI-level integration. Mainframes generally cannot access friendly data stores and usually do not provide public APIs. For their part, many client/server applications embed the business logic in the client. In these cases, UI-level integration represents the only way to access and maintain data integrity.

In most cases, UI-level integration is your last resort. Adding scripting logic to catch events within client/server applications quickly becomes difficult to maintain as the integration level increases and changes occur. In either case, UI changes can break the integration triggers and logic. Moreover, tight coupling forever links UI maintenance with the integration code's maintenance.

3.3.3 Application-level Integration

Application-level integration, probably the best way to integrate applications, uses the integrated application's integration frameworks and APIs. Application interfaces let you invoke business logic to preserve data integrity. Integration API examples include Siebel's Java DataBeans and SAP's JCA (J2EE Connector Architecture). Prefer application-level integration because it is transparent to the integrated application and preserves the application's data integrity.
3.3.4 Method-level Integration

Method-level integration, a less frequently used superset of application-level integration, aggregates common operations on multiple applications into a single application that fronts the integrated applications.

Use method-level integration when each integrated application provides a similar set of API or functional methods. Typically, you’d create an aggregating (front) application, which fronts the aggregated applications using distributed components (CORBA, Enterprise JavaBeans (EJB), DCOM (Distributed Component Object Model), and so on).

Method-level integration requires the integrated applications to support a RPC (remote procedure call) or distributed component technology. All applications that interact with the integrated applications do so through the front application. So, if a client application wants to add a customer, it would call the front component’s method. The component would then add the customer to the ERP and CRM (customer relationship management) systems.

Method-level integration’s major disadvantage stems from the tight application coupling in front components. Changes in the integrated application API break the front application components and the applications that rely on them. Pursue method-level integration when you have CORBA-based integration technology or a distributed component technology. Because method-level integration is a more complicated form of application-level integration, it usually makes more sense to pursue application-level integration using middleware.

3.4 Enterprise Application Integration Approaches

There are a number of different approaches to achieving enterprise application integration. Following are five approaches that are typically used to integrate existing enterprise information systems with enterprise applications. These are [18]:

29
• Using a two-tier client-server approach
• Using synchronous adapters
• Using asynchronous adapters
• Using a message broker approach
• Using an application server-based approach

3.4.1 Two-tier Client-Server Approach

This approach is based on the two-tier client-server model, and it is typical of applications that are not based on the Web. It was a widely used approach prior to the advent of Web-based applications, but is less used now. With this approach, an EIS provides an adapter that defines an API for accessing the data and functions of the EIS. A typical client application accesses data and functions exposed by an EIS through this adapter interface. The client uses the programmatic API exposed by the adapter to connect to and access the EIS. The adapter implements the support for communication with the EIS and provides access to EIS data and functions. Communication between an adapter and the EIS typically uses a protocol specific to the EIS. This protocol may provide support for security and transactions. It also typically supports content propagation from an application to the EIS. Most adapters expose an API to the client that abstracts out the details of the underlying protocol and the distribution mechanism between the EIS and the adapter. Refer Figure 3.5.

3.4.2 Using Synchronous Adapters

An adapter can expose either a synchronous or an asynchronous mode of communication between the client applications and the EIS. Figure 3.6 illustrates using adapters designed for synchronous communication. Adapters designed for this approach provide a synchronous request-reply communication model for use between an application and an EIS.
How might a synchronous adapter work? As an example, let’s consider an adapter that defines an API that includes a remote function callable by an application. This remote function creates an accounts receivable item in the EIS. When an application wants to interact with the EIS to create an accounts receivable item, it invokes this remote function on the EIS. The application that initiated the call then waits until the function completes and returns its reply to the caller. The reply contains the results of the function’s execution on the EIS. An interaction such as this is considered synchronous because the execution of the calling application waits synchronously during the time the function executes on the EIS. One form of synchronous adapter allows bidirectional synchronous communication between an application and an EIS. This type of adapter enables an EIS to synchronously call an application.

### 3.4.3 Using Asynchronous Adapters

Asynchronous adapters provide another approach to application integration. Let’s use the same example of an adapter that exposes an API with a remote function that permits an application to interact with the EIS and create an accounts receivable item. This function is callable by an application. With asynchronous communication, an application calls the remote function to create a new accounts receivable item in the EIS. The application makes the remote call, then immediately returns and continues its own processing. The remote function is sent to the EIS. The EIS handles the function and returns some reply information to the application as a separate
asynchronous invocation. The resource adapter dispatches the asynchronous call from the EIS to the application. The important point to remember is that the application does not suspend its own processing while the remote function executes on the EIS. Rather, the application continues its own work and receives notification at some later point of the results of its earlier remote function invocation In addition, an EIS is able to asynchronously invoke or call an application.

![Asynchronous Outbound Communication Diagram](image1)

![Asynchronous Inbound Communication Diagram](image2)

Figure 3.6 Using Synchronous Adapter

**Queue-Based Approach**

Asynchronous message-based communication may also be used to integrate enterprise applications and EISs. There are two forms of asynchronous messaging viz. queue-based messaging and publish-subscribe messaging. A message broker may provide either one of these forms of messaging.

![Queue-Based Messaging System Diagram](image3)

Figure 3.7 Using a Message Queue for EIS Integration
Figure 3.7 illustrates queue-based communication. Queue-based communication, which is also called point-to-point messaging, involves one application sending a message to a message queue. With queue-based communication, a queue that is independent from both the sender and receiver applications acts as a message buffer between the communicating applications. The sender application sends a message to this queue, while the receiver application receives its messages from the same queue.

**Publish-Subscribe Approach**

The publish-subscribe approach works differently from the queue-based approach.

![Publish-Subscribe system for EIS Integration](image)

Figure 3.8 Using a Publish-Subscribe system for EIS Integration

Figure 3.8 illustrates publish-subscribe messaging. This might be a stock quote service that publishes messages—updated stock prices—to subscribed portfolio applications. With publish-subscribe messaging, there are message publishers, who produce messages, and message subscribers, who register their interest in particular messages. There is also a separate publish-subscribe facility that acts as the integration point—publishers publish messages to this facility and the facility delivers messages to subscribers.

Here’s how publish-subscribe messaging works. A publisher application publishes messages on a specific topic, such as up-to-the-minute quotes on a specific stock symbol. Multiple applications can subscribe to this topic and receive the messages published by the publisher. The publish-subscribe facility takes the responsibility of
delivering the published messages to the subscribing applications based on the subscribed topic.

When an application needs to use either queue-based or publish-subscribe messaging, it must also hook into a messaging system that provides these mechanisms. The application typically uses an API exposed by the messaging system to access the messaging services. Typically, the messaging system uses a messaging adapter, also called a provider, to implement the messaging API. Java Message Service (JMS) provides an API for enterprise messaging systems. Applications, called JMS clients, use the JMS API to access the messaging service and either a queue-based or publish-subscribe messaging system. (Refer Figure 3.9.)

![Figure 3.9 Using JMS Provider](image)

3.4.4 Using a JMS Provider

Figure 3.10 illustrates using a message broker for EIS integration. Notice that an adapter enables an application to access the message broker. In this scenario, an adapter maps the application-level interface for the message broker to the underlying asynchronous messaging mechanisms supported by the message broker, plus the adapter maps the message formats supported by the message broker. (The underlying messaging mechanisms supported by the message broker may be a queue-based or a publish-subscribe mechanism, for example.) Some adapters layer additional
functionality between the application and the message broker. For example, they may add the capability to do message transformations—an adapter may transform application-specific messages to a format expected by the message broker. The message broker then converts the message to a format expected by the message receiver or subscriber.

Figure 3.10 Using a Message Broker for EIS Integration

When applications and EISs use a message broker for integration and message delivery, the applications and the EISs can act as both message producers and consumers. For example, a financial accounting application can subscribe to messages that carry information on financial transactions. An order management application may send a message through the message broker that updates an account payable in the accounting application. Most message broker vendors provide vendor-specific adapters for popular EISs.

When an application and an EIS communicate using asynchronous messaging, they are considered as loosely coupled. There are both advantages and disadvantages to a loosely coupled integration. With a loosely coupled integration between a target EIS and an application, the application can continue processing client requests without blocking on EIS performance or communication glitches. This improves scalability. However, application developers may find it difficult to program against an
asynchronous messaging model. Also, these asynchronous messaging systems do not always support the propagation of security and transactional contexts.

A message broker may provide additional services for enterprise application integration. These additional services are: message routing, transaction management, reliable message delivery, messages priority and ordering, and message transformation.

3.4.5 Application Server-Based Integration

Figure 3.11 shows how an application server can be used for integration with existing enterprise applications and EISs.
An application server is a natural point for application integration, because it provides a platform for development, deployment, and management of web-based enterprise applications. Application servers are the platform of choice for applications that are developed using a multi-tier architecture.

A typical multi-tier application consists of three tiers: a client tier, a middle tier, and an EIS tier. The middle tier typically implements the business logic for an application. As part of its implementation of business logic functionality, the middle tier might access data and functions associated with applications running on the EIS tier. The middle tier also serves up both static and dynamic presentation content to the client tier.

The EIS tier contains the systems that run existing enterprise applications and databases. As described earlier, these EISs can be custom or off-the-shelf applications. The client tier is composed of different types of client applications. A client can be a Web browser-based HTML client or a peer application. A typical application server supports a component-based model for developing applications. With this model, an application may be composed of different types of components, such as web components or business components. The application server provides deployment and runtime support for these application components. In effect, an application server provides an extremely useful platform for the development of Web-based, transactional, secure, distributed, and scalable applications. This increases the usefulness of an application server environment for enterprise application integration.

Typically, an application server provides a set of runtime services to its deployed components. These runtime services are hidden from the application components through a simplified application-programming model. The services provided include:

- Support for transactions
- Security
- Load balancing and failover
• Database access
• Asynchronous messaging
• Distributed communications
• Web protocols
• XML support

It is possible to develop and deploy applications on an application server such that the applications can connect and aggregate access to multiple heterogeneous EISs and existing enterprise applications. When applications are developed with this ability to access multiple heterogeneous EISs, Web and business components that are deployed on the middle tier (or application server) use adapters to access the data and functions associated with the applications on these EISs.

Application components deployed on the application servers use synchronous resource adapters to connect and access EISs. As explained earlier, this is tightly coupled integration between applications and EISs. Application components can also use an adapter (or JMS provider) to a Message broker to integrate with EISs based on asynchronous messaging.

3.5 An XML based framework for EAI

Adaptability and flexibility have been the key considerations in evolving the framework. The framework is therefore general enough to suite a wide variety of scenarios. It suggests an open architecture and spans the four crucial areas of EAI: communication, data, process and monitoring these three through human intervention.

EAI can be achieved through XML and related technologies. The components of the framework are:

1. XML interface, which is part of enterprise application.
2. Enterprise Integration Engine, which handles the core functionality
3. EAI portal, used to configure and defines events and processes.
The framework scores in several respects over the existing EAI techniques. The following are the important advantages:

1. Using XML, data is more transparent during transfers and facilitates easy adoption of changes the schemas and business processes to suite growing needs.
2. Single point control, co-ordination and customization through the “EAI portal”, providing for human intervention, when necessary.
3. Easy to query the business processes and modify them to suite changes in the business environment.
4. The interactions are document-based for easy presentation of information.
5. Does not involve any programming on the part of the end user.
6. Provide meaningful, interpretable message format for easy documentation [23].

3.6 EAI is the best approach for accessing Legacy data

For most organizations, the best approach to accessing legacy data is usually via multi-tier application modeling. It provides flexible, manageable and scalable user access in real time and rapid time-to-market, within a reasonable budget. Some of the key features of a complete multi-tier approach will include [26]:

1. Support for a wide range of client front-ends, including Java, Windows APIs (such as ActiveX), and C-language libraries.
2. Support for XML.
3. Support for Application servers, including WebSphere and WebLogic
4. Ability to act as a legacy data source to standard end-user and development environments.
5. Management features and scalability enabling thousands of concurrent sessions
6. Automated tools for capturing screens and navigation, minimizing the amount of code to be written.
7. Intelligent design so that the client user interface, the database of screens and navigation and the back-end data access are independent, optimizing code reuse