Enterprise Application Integration

From Point-to-Point Integration towards Semantically-enriched Approaches

Γκουβάς Παναγιώτης
pgouvas@mail.ntua.gr

Μπούρας Θανάσης
bouras@mail.ntua.gr

ΑΘΗΝΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2006
# Table of Contents

Introduction ............................................................................................................................. 4

1. The roadmap towards EAI .................................................................................................. 5

1.1 Evolution of IT systems ................................................................................................. 5

1.1.1 Monolithic applications ........................................................................................... 5

1.1.2 Client-server architecture ....................................................................................... 6

1.1.3 N-tier architecture ................................................................................................... 7

1.2 The need for EAI ........................................................................................................... 9

1.2.1 Rationale ................................................................................................................ 9

1.2.2 Is EAI the final solution? ....................................................................................... 10

1.2.3 Progression of EAI enabling technologies ........................................................... 10

1.2.4 Why they have failed up to now? ......................................................................... 13

2. Current trends in EAI ........................................................................................................ 14

2.1 Service Oriented Architecture ..................................................................................... 14

2.1.1 Implications of SOA architecture .......................................................................... 16

2.2 Enterprise Service Bus ............................................................................................... 16

2.2.1 The ESB supports multiple integration cases ...................................................... 17

2.2.2 ESB centralisation of control and Distribution of Processes ................................. 18

2.2.3 The role of the ESB and other SOA components ................................................ 20

2.3 Web Services Technology .......................................................................................... 21

2.3.1 SOAP ................................................................................................................... 22

2.3.2 WSDL: Web Services Description Language....................................................... 23
2.3.3 UDDI Universal Description, Discovery, Integration................................. 23

2.3.4 SOAP/HTTP uses existing namespaces and infrastructure...................... 24

2.4 Industrial Standards on Process Management (BPEL, BPEL4WS)................. 24

2.5 EAI deployment lifecycle ............................................................................... 27

3. Industrial EAI Approaches .............................................................................. 30

3.1 SAP’s Enterprise Services Architecture.......................................................... 30

3.2 Oracle Fusion Architecture ............................................................................ 36

3.3 IBM SOA & Business Patterns ...................................................................... 39


References............................................................................................................. 45
Introduction

In this report, a brief outline of the technological part of Enterprise Application Integration (EAI) will be presented. The technological overview comprises of three main parts. The first part is a historical review of the era before EAI that illustrates the necessity of EAI. The second part of the chapter refers to the current trends of EAI regarding the technologies that realise EAI solutions. The third part refers to the current industrial trends of indicative stakeholders in the EAI domain. Finally, the report will point out the pitfalls of the current, industrial trends and technologies in EAI, and present the need of semantics and Semantic Web technologies in EAI, as well as the main research projects and initiatives working in the domains of semantics and EAI.
1. The roadmap towards EAI

In this section a historical review concerning the evolution of IT systems will be presented. The actual goal of this section is to provide an introduction for the strong necessity of Enterprise Application Integration in nowadays.

1.1 Evolution of IT systems

All IT applications can be broken into three layers: a) Presentation Layer (What the user sees), b) Business Logic Layer (The underlying processing engines and their rules), and c) Data Layer (The physical data storage layer).

The necessity for this layered approach is adoptability of systems. In the IT world things change extremely quick and often dramatically. As a result, developers must often modify applications to meet new requirements in short order. How flexible an application will be, will depend on how well the three layers have been separated.

Taking into consideration the three layered approach of every IT application and focusing on information related to the nature of IT systems development, we can begin to see an evolution that stretches from the early days of software engineering into our near future. This history can roughly be broken into three stages: a) monolithic applications, b) Client / server applications and c) n-tier applications.

1.1.1 Monolithic applications

These early applications (Figure 1-1) were more powerful than anything that had come before. However the lack of any collaborative systems meant that most applications were built and designed for single users.

Figure 1-1. Monolithic Applications
From an architectural perspective, monolithic applications were fairly primitive. The reason they were primitive was because they integrated *all three application layers* into one executable application which was maintenance-heavy, hard-to-share, and unscalable.

### 1.1.2 Client-server architecture

As technology advanced, connecting machines and sharing data became an important goal and a pressing reality for application developers. Simple networks have been formed and new applications and application architectures arose. Since networking and resource sharing introduced larger and more complex problems into the development environment, and because the inherent flaws in monolithic applications were becoming clear, a new approach that captured the nature of these new applications was devised. Client-Server applications became all the rage.

In the client-server model, applications *were broken apart, distributing processing between client computers and server computers*. As client-server applications became feasible, so too did the layering of the technology become more important. In the client server model, the three layers of an application could *more easily be isolated*. In fact, such isolation became even more crucial as scalability, distribution and maintenance became even more complex.

Another factor in the separation of the layers came from the data. As sharing data became essential to faster and wider information distribution, the network systems drove applications to evolve into data sharers. Rather than store data locally, in a client-server application, data would be stored in a central repository where it could be accessed by multiple clients who wished to "share" it (Figure 1-2).

![Figure 1-2. Client Server Architecture](image)

The benefit of this architecture was that it gave access to large numbers of users so that they could store and retrieve important data in a consistent and stable manner generally
from a "fully loaded" application on the client machine. Order processing, accounts, internal systems; email and database applications became the norm in the client-server era.

The traditional client server applications enabled, and encouraged, developers to build feature rich solutions that integrated key technologies in a single point of access. Typically a developer would focus on delivery of the graphical user interface and storing data in repositories that enabled users to share data.

The client-server model presented a major paradigm shift when compared to monolithic applications. However, it remained far from ideal. The Client Server approach forced developers to locate the three main development layers in just two locations, in the client or in the server. Typically, presentation and business rules logic would always remain within the purview of the client-side application. This enabled the client application to 'short-cut' the UI to produce clever designs.

Furthermore, data became the responsibility of the database or messaging subsystems. As such they were accessed either via direct API or through thin layers of 'enabling technology' (such as ODBC). In some instances, this provided limited flexibility.

The presentation layer would exist as a single layer within the client side application but the business and database layers could exist in the database or messaging server. This enabled the client side application to respond quickly to cosmetic changes without having to test all business logic cases, and at the same time allowed the business rules to change without requiring mass rollouts at every such occasion.

The Client-Server architectures also suffered under constant maintenance strains due to proprietary standards, technologies and, most importantly, a lack of scalability.

1.1.3 N-tier architecture

Certainly, there are solutions to most of these problems, however, one specific problem remains a constant problem for all multi-client solutions; scalability. As users are added to a solution it becomes more and more difficult to maintain a consistently good level of service without diverting significant resources.

In a collaborative business systems applications developers could much more closely approach the three application layers. Additionally each layer could be supplanted with 'proxy' layers that provide a cushion for processing, migration and scaling issues. Such an
example would be a generic data source access wrapper to enable the database to be substituted without large and costly redevelopment.

![3-tier architecture](image)

**Figure 1-3.** 3-tier architecture

With the popularization of the internet / intranet / extranet solution a range of problems could be addressed in a consistent manner, while at the same time enabling the holy grail of "layer separation" to move closer. In breaking the approach into layers (e.g. n-tier architecture), see Figure 1-3, it becomes possible to focus on solving domain specific issues. At the same time, design flexibility could be introduced:

- Using a browser to access services assures a familiar lightweight interface.
- The browser and HTML provide a standard implementation and deployment approach. Applications aren't really deployed; more borrowed.
- No client-side issues (ignore cross browser issues for now).
- Highly tailored solutions. Users are not required to "USE" everything.
- Applications can be seamlessly divided across logical and physical locations.
- Disconnected or Just-In-Time (JIT) operation.
- Protection for both resources and clients. Fewer concurrent accesses, less resource usage.
- Platform independent...nearly.
1.2 The need for EAI

1.2.1 Rationale

Most enterprises contain a systemic infrastructure of several heterogeneous systems, creating a complex, fuzzy network of interconnected applications, services and data sources, which is not well documented and expensive to maintain (Samtani & Sadhwani, 2002). Moreover, the introduction of multi-oriented, separate legacy systems concerning enterprise resource planning (ERP), customer relationship management (CRM), supply chain management (SCM), e-business portals and B2B transactions, increases the complexity of systems integration, making the support of the interoperability among these systems a challenging task.

Figure 1-4. The enterprise system environment: with and without an EAI system

In this emerging business context, a clear need appears to link these former incompatible systems to improve productivity and efficiency. The solution to this need is what is called Enterprise Application Integration (EAI), which can be defined as the use of software and architectural principles to bring together (integrate) a set of enterprise computer applications (see Figure 1-4).

The goal of EAI is to integrate and streamline heterogeneous business processes across different applications and business units. We distinguish between intra- and inter-organizational enterprise application integration. Intra-organizational EAI, commonly referred to as "Application to Application"-Integration (A2A) (Bussler, 2003), specifies the automated and event-driven exchange of information between heterogeneous enterprise applications and systems operating within an organization or enterprise. On the other hand, inter-organizational EAI, or else B2B integration (Bussler, 2003), specifies the automated and event-driven information exchange between various systems of several collaborating organizations and enterprises. Moreover, (Apshankar et al., 2002) identify different types of EAI levels / layers, explaining the various dimensions of the integration task, namely:
- data-oriented integration, occurring at the database and data source level, either real time or non-real time, constituting the most widespread form of EAI today,

- function or method integration, involving the direct and rigid application-to-application integration of cross-platform applications over a network – it can be achieved using custom code, APIs, Remote Procedure Calls (RPCs) or distributed middleware and distributed objects (CORBA, RMI, DCOM),

- user interface integration, consisting on using a standardized user interface for accessing a group of legacy systems and applications. The new presentation layer is integrated with the existing business logic of the legacy systems or packaged applications, and

- business process integration, occurring at the business process level.

1.2.2 Is EAI the final solution?

In recent years, most enterprises and organizations have made extensive investments in several EAI systems and solutions that promise to solve the major integration problem among their existing systems and resources. The business driver behind all these traditional EAI projects is to integrate processes across third-party applications as well as legacy systems to decrease the number of adapters one has to develop if connecting two systems.

Therefore, the traditional EAI focuses on the message-based communication of software applications interfaces, by pipelining different middleware technologies and developing various adapters, connectors and plug-ins to provide efficient messaging support among heterogeneous systems, allowing their effective interconnection.

However, traditional EAI efforts lack of an upper abstraction layer, as well as standardized architectures and implementations, making customers and end-users captive of EAI vendor-specific solutions, and arising a new, high level integration problem of interconnecting various EAI systems with one another. The growth of EAI market and the involvement of new EAI vendors have intensified the integration problems identified, considering the standardization of integration frameworks and architectures a necessity.

1.2.3 Progression of EAI enabling technologies

The motivation of EAI was the heterogeneity that exists in the systemic infrastructure of
Enterprises. Furthermore as IT infrastructures evolved the necessity for interconnecting these infrastructures together led to the concept of inter and intra Enterprise Application Integration; the first targeting the integration of different applications that exist in an enterprise and the latter targeting in the effective collaboration among enterprises. In fact the actual need for integration was present from the very early era of IT systems (Monolithic Applications).

However EAI is supported by a key enabling Technology which varies from time to time. The term key ‘enabling technology’ refers to a set of protocols that vary from data transmission to data formalisation (XML, SOAP etc) and supports the whole lifecycle of Application integration. Additionally when the discrimination for presentation, business logic and data took place the need for the EAI was amplified and the actual task of achieving EAI became more difficult.

Since EAI is not something new, conceptually at least, and since it aims at integration of all IT layers (data, functional) a short quotation concerning the, at first, suggested solution for EAI, should be given.

![Diagram of Point-to-Point vs. Ideal EAI](image)

**Figure 1-5.** Point-to-Point vs. Ideal EAI

The first EAI approach was **Point-to-Point integration** (Figure 1-5). Point-to-Point integration leads to complex and fuzzy networks of interconnected applications which are not well documented and extremely hard to maintain. The main drawbacks are the existence of custom interfaces, high complexity and low tolerance in a possible change of a designed process. Additionally integration costs are high since such heterogeneous systems demand long integration projects. Consequently IT environments become increasingly rigid.
Each EAI architecture is based on a key-enabling technology as mentioned above. As key enabling technologies became more sophisticated, Point-to-Point approach was replaced by message-based communication of software applications interfaces. This technique relies in the pipelining of different Middleware technologies and demands the development of various adapters, connectors and plug-ins to provide efficient messaging support among heterogeneous systems.

![Diagram showing Service Layer, Component Layer, and Object/Class Layer](image)

Figure 1-6. Services in SOA

**Finally the latest stage of EAI evolution is Service-Oriented Architecture (SOA).** The key concept in SOA is the “Service” (Figure 1-6). A service encapsulates a well-defined invokable unit of business function, and exists either to provide information or to facilitate the change of business data from one valid and consistent state to another.

Services are defined using explicit interfaces that are independent of service implementations, and both service requestors and service providers agree to. Services should be invokable through defined communication protocols that stress interoperability and location transparency.

![Diagram showing the evolution of key-enabling technologies](image)

Figure 1-7. Evolution of Key-enabling Technologies
The key enabling technology in the SOA approach is the **Web Service**. A web service is an open standard for system interaction **independent** of technical architecture. The development and introduction of **Web Service enabled Service-Oriented Architecture** solutions; completely based on widely known and accepted standards would visionary overcome most of the EAI obstacles. Figure 1-7 presents the evolution of the key enabling technologies concerning EAI.

### 1.2.4 Why they have failed up to now?

As mentioned above traditional EAI efforts lack of 1) **an upper abstraction layer** and 2) **standardized architectures and implementations**, making customers and end-users captive of EAI vendor-specific solutions, and arising a new, high level integration problem of interconnecting various EAI systems with one another.
2. Current trends in EAI

The second part of this report refers to the current trends of EAI regarding the technologies that realise EAI solutions. At first an introduction to Service Oriented Architecture (SOA) is provided. Moreover a realisation technique of SOA called Enterprise Service Bus (ESB) is briefly discussed. Additionally, main key-enabling technologies of SOA (e.g. Web Services, SOAP etc.) are analysed. Finally the last part of this section undertakes the task of introducing the SOA deployment lifecycle.

2.1 Service Oriented Architecture

Service-oriented architecture is an approach to defining integration architectures based on the concept of a service. It applies successful concepts proved by Object Oriented development, Component Based Design, and Enterprise Application Integration technology. The goal of SOA can be described as bringing the benefits of loose coupling and encapsulation to integration at an enterprise level.

In order to describe SOA, it is first necessary to define what we understand by a “service” in this context. This is key as, unless we are confident that the services that we define really are well designed, we cannot be sure to achieve the promoted benefits of SOA. The most commonly agreed-on aspects of the definition of a service in SOA are:

- Services are defined by explicit, implementation-independent interfaces
- Services are loosely bound and invoked through communication protocols that stress location transparency and interoperability
- Services encapsulate reusable business functions

The use of explicit interfaces to define and encapsulate services function is of particular importance and is illustrated in Figure 2-1. Note how the interface encapsulates those aspects of process and behaviour that are common to an interaction between two systems, while hiding the specifics of each implementation. By explicitly defining the interaction in this way, those aspects of either system (for example the platform they are based on) that are not part of the interaction are free to change without affecting the other system.

After the function has been encapsulated and defined as a service in an SOA, it can be used and reused by one or more systems that participate in the architecture. For example, when the reuse of a Java™ logging API could be described as “design time” (when a
The intention of SOA is to achieve the reuse of services at:

- **Runtime**: Each service is deployed in one place and one place only, and is remotely invoked by anything that must use it. The advantage of this approach is that changes to the service (for example, to the calculation algorithm or the reference data it depends on) need only be applied in a single place.

- **Deployment time**: Each service is built once but redeployed locally to each system or set of systems that must use it. The advantage of this approach is increased flexibility to achieve performance targets or to customize the service (perhaps according to geography).

Note that in contrast to reusing service implementations at runtime, the encapsulation of functions as services and their definition using interfaces also enables the substitution of one
service implementation for another. For example, the same service might be provided by multiple providers (such as a car insurance quote service, which might be provided by multiple insurance companies), and individual service requesters might be routed to individual service providers through some intermediary agent. The encapsulation of services by interfaces and their invocation through location-transparent, interoperable protocols are the basic means by which SOA enables increased flexibility and reusability.

2.1.1 Implications of SOA architecture

The encapsulation of reusable business function, the achievement of loose coupling, the definition of appropriate levels of granularity, and so forth are analysis issues as much as a technology issues. They are difficult issues to grasp, so SOA cannot be successful without skilled architects and designers who understand and are able to articulate them. It is easy to see these concerns becoming hostage to time, skill, and cost issues, leading to another generation of isolated systems that will require integration.

Widespread implementation of an SOA and infrastructure is a long-term endeavour that involves all of the usual hard business decisions, questions of data, and process ownership. It requires serious, long-term commitment by business and by the IT organization that supports it. It may involve upfront costs, centralized costs, and many other challenges:

- No specific technologies are ruled in or ruled out. Legacy implementations are possible
- EAI implementations are commonplace (for example, XML over MQ / JMS)
- Web services are potentially a very good fit, but are still maturing.

2.2 Enterprise Service Bus

In order to implement an SOA, both applications and infrastructure must support the SOA principles. Enabling applications involves the creation of service interfaces to existing or new functions, either directly or through the use of adapters. An Enterprise Service Bus (ESB) generally provides an abstraction layer on top of an Enterprise Messaging System which allows integration architects to exploit the value of messaging without writing code (Figure 2-2). Enabling the infrastructure at the most basic level involves the provision of capability to route and transport service requests to the correct service provider. The role of the Enterprise Service Bus is, in part, simply to enable the infrastructure in this way.
The true value of the Enterprise Service Bus concept, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability, and to operate and integrate in a heterogeneous environment. The implications of these requirements go beyond basic routing and transport capability. The ESB should enable the substitution of one service implementation by another with no effect to the clients of that service. This requires both the service interfaces that are specified by SOA and that the ESB allows client code to invoke services in a manner that is independent of the service location and communication protocol that is involved.

**Figure 2-2, Enterprise Service Bus**

### 2.2.1 The ESB supports multiple integration cases

In order to fully support the variety of interaction patterns that are required in a comprehensive SOA (for example, request / response, publish / subscribe, events), the Enterprise Service Bus must support in one infrastructure the three major styles of Enterprise Integration:

- **Service-oriented architectures** in which applications communicate through reusable services with well-defined, explicit interfaces. Service-oriented interactions leverage underlying messaging and event communication models.

- **Message-driven architectures** in which applications send messages through the ESB to receiving applications.
Event-driven architectures in which applications generate and consume messages independently of one another.

The ESB does this while providing additional capabilities to mediate or transform service messages and interactions, enabling a wide variety of behaviours and supporting the various models of coupling interaction.

2.2.2 ESB centralisation of control and Distribution of Processes

The ESB is sometimes described as a distributed infrastructure and is contrasted with solutions (such as broker technologies) that are commonly described as hub-and-spoke. Figure 2-3 illustrates this common depiction of the ESB.

However, this view of the ESB is not very helpful in describing how the ESB is physically implemented. For example, what infrastructure components implement the ESB, which is depicted as a line in this diagram?

![Figure 2-3. The Enterprise Service Bus as a physical infrastructure](image)

In contrast, hub-and-spoke integration solutions (Figure 2-4) seek to centralize control of configuration: routing information, service naming, and so forth.

![Figure 3-4. Hub-and-spoke integration](image)

In the Patterns for e-business Process Integration patterns an ESB is classified as a type of bus, which in turn is classified as a type of hub, as shown in Figures 2-5 and 2-6.
The distinction between distributed bus and centralized hub-and-spoke solutions is really a false one. Two different issues are being addressed here: the centralization of control and the distribution of infrastructure. In initial or small-scale implementations of integration solutions, the physical infrastructure is likely to be centralized: concentrated on a single cluster, or hub, of servers. However, as the implementation evolves, the infrastructure may become more physically distributed, as a bus, while retaining at least logically the central control over configuration. Figure 2-7 shows the resulting implementation of an ESB. (The Configuration and Control Services node is shown dotted to illustrate that it is a logical construct.)
Of course, this wide distribution of broker technology in a bus pattern is dependent on the capabilities of specific technologies to support such distribution patterns. Equally important from the perspective of incremental implementation and deployment of ESB technology is the ability to extend existing deployments by adding further distributed processing capacity without affecting the existing infrastructure.

### 2.2.3 The role of the ESB and other SOA components

The ESB is not the only infrastructure component in a SOA. Although individual scenarios vary, there are other commonly occurring components whose role we should position relative to the ESB:

- The Business Service Directory, which provides a taxonomy and details of available services to systems that participate in an SOA.
- The Business Service Choreography, which is used to orchestrate sequences of service interactions into short or long-lived business processes.
- The ESB Gateway, which is used to provide a controlled point of external access to services where the ESB does not provide this natively. Larger organizations are likely to keep the ESB Gateway as a separate component. An ESB Gateway can also be used to federate ESBs within an enterprise.
2.3 Web Services Technology

Web services\(^1\) are a recent set of technology specifications that leverage existing proven open standards such as XML, URL, and HTTP to provide a new system-to-system communication standard. Based on this communication model, additional higher-level Web services standards have also been defined to address transactions, security, business processes, and so forth: the higher-order functions that are required to get systems, applications, and processes (rather than objects and components) talking to each other. Web services learn from the way the Web revolutionized how people talk to systems: new customers, new business models, extensions of opportunity, new transparency and improved collaboration between employees and employers, and in some cases reductions in infrastructure costs and complexity.

The key to these successes was a universal server-to-client model that is consistent with a highly distributed environment, based on simple open standards and industry support. Figure 2-8 shows the basic interaction model supported by Web services.

![Diagram of Web Services Interaction Model](http://www.w3.org/TR/2004/WD-wsdl20-20040803/)

\(^1\) Web Services Specifications [http://www.w3.org/TR/2004/WD-wsdl20-20040803/]
Basic Web services define interactions among Service Requesters, Service Providers, and Service Directories as follows: Service Requesters find Web services in a UDDI\(^2\) Service Directory. They retrieve WSDL descriptions of Web services offered by Service Providers, who previously published those descriptions to the Service Directory. After the WSDL has been retrieved, the Service Requester binds to the Service Provider by invoking the service through SOAP.

The basic Web services are often described in terms of SOAP, WSDL, and UDDI, each of which will be analysed below. However, it should be noted that each of these standards can be used in isolation, and there are many successful implementations of SOAP alone, or SOAP and WSDL, in particular.

### 2.3.1 SOAP

SOAP\(^3\) is an XML messaging protocol that is independent of any specific transport protocol. SOAP defines a framework within which messages contain headers, which are used to control the behaviour of SOAP-enabled middleware, and a message body. As SOAP is an XML format, and as XML is text-based, SOAP is supportable in the vast majority of existing and new technical environments and can be transported over a vast variety of protocols. In practice, SOAP is most often communicated over HTTP, although this is likely to evolve rapidly because HTTP is an unreliable protocol. (For instance, it is already possible to send SOAP messages through JMS implementations such as WebSphere MQ.) Basic SOAP also makes no reference to characteristics of interactions such as security and transactionality. However, as SOAP headers provide an extensible model, these aspects are being added gradually to the Web services specifications as extensibility elements, as we describe further in the next section.

The use of SOAP over specific protocols, such as HTTP, is usually written as SOAP/HTTP, SOAP/JMS, and so forth. The SOAP V1.2 specification is available from the World Wide Web Consortium, and deliberately does not define a meaning for SOAP as an acronym. (SOAP is sometimes referred to as Service Oriented Architecture Protocol, or by its definition in the more widely supported SOAP V1.1 specification, Simple Object Access Protocol).

---


\(^3\) SOAP [http://www.w3.org/TR/soap/](http://www.w3.org/TR/soap/)
2.3.2 WSDL: Web Services Description Language

WSDL is an XML-based interface definition language that separates function from implementation, and enables design by contract as recommended by SOA. WSDL descriptions contain a PortType (the functional and data description of the operations that are available in a Web service), a Binding (providing instructions for interacting with the Web service through specific protocols, such as SOAP/HTTP), and a Port (providing a specific address through which a Web service can be invoked using a specific protocol binding).

The value of WSDL is that it enables development tooling and middleware for any platform and language to understand service operations and invocation mechanisms. For example, given the WSDL interface to a service that is implemented in Java, running in a WebSphere environment, and offering invocation through HTTP, a developer working in the Microsoft .Net platform can import the WSDL and easily generate application code to invoke the service. As with SOAP headers, the WSDL specification is extensible and provides for additional aspects of service interactions to be specified, such as security and transactionality.

2.3.3 UDDI Universal Description, Discovery, Integration

UDDI servers act as a directory of available services and service providers. SOAP can be used to query UDDI to find the locations of WSDL definitions of services, or the search can be performed through a user interface at design or development time. The original UDDI classification was based on a U.S. government taxonomy of businesses, and recent versions of the UDDI specification have added support for custom taxonomies.

A public UDDI directory is provided by IBM, Microsoft, and SAP, each of whom runs a mirror of the same directory of public services. However, there are many patterns of use that involve private registries; see Steve Graham’s articles: The role of private UDDI nodes in Web services, Part 1: Six species of UDDI\(^4\). The role of private UDDI nodes, Part 2: Private

nodes and operator nodes.$^5$

### 2.3.4 SOAP/HTTP uses existing namespaces and infrastructure

One of the important potential benefits of Web services is to reduce the reliance of integration on specific integration technologies that require heavyweight deployment where such deployment would be problematic or impossible —typically, in business-to-business interactions, particularly as they become more widespread and dynamic.

The specific use of Web services with HTTP as a communication protocol has some extraordinary benefits in this area. Because SOAP/HTTP uses HTTP as a communication protocol and URL as the addressing format, the entire global network of distributed, resilient routing and communications infrastructure that the Internet provides can be used. Allowances must be made for the unreliable nature of HTTP, but the advantages of a service communication protocol that is already deployed and globally pervasive should not be underestimated.

### 2.4 Industrial Standards on Process Management (BPEL, BPEL4WS)

SOA advocates that developers create distributed software systems whose functionality is provided entirely by services. SOA services:

- can be *invoked remotely*
- have *well-defined interfaces* described in an implementation-independent manner, and
- are *self-contained* (each service’s task is specific and reusable in isolation from other services). Business Process Management introduces a *fourth layer* to the ESB architecture.

Using an SOA, all of an organization’s IT systems can be viewed as services providing particular *business functions*. Because the ESB resolves integration issues, Business Process Execution Language$^6$ (BPEL) can orchestrate these individual tasks into *business*...

---


$^6$ [BPEL4WS](http://www-128.ibm.com/developerworks/library/specification/ws-bpel/)
**processes.** BPEL expresses a business process's *event sequence* and *collaboration logic*, whereas the underlying Web services provide the process functionality. To gain the most from BPEL, developers must understand the dividing line between the logic implemented in the BPEL processes and the functionality that Web services provide. BPEL has several core features. Actions are performed through activities, such as invoking a Web service or assigning a new value in an XML document. Activities such as while or switch offer the developer control over activity execution. Because it was designed to implement only the collaboration logic, BPEL offers only *basic activities*. BPEL describes communication with partners using partner links, and messages exchanged by partners are defined using WSDL. Web services operate using client-server or peer-to-peer communications. In client-server communication, the client must initiate all invocations on the server, whereas in peer-to-peer communication, partners can make invocations on each other. BPEL extends WSDL with *partner link definitions* to indicate whether client-server or peer-to-peer communication will be used. In peer-to-peer communication, each partner uses WSDL to define its Web service interfaces; partner links define each partner's role and the interfaces they must implement (WSDL 1.1 alone can't do this satisfactorily).

BPEL supports *asynchronous message exchanges* and gives the developer great flexibility regarding when messages are sent or received. It also gives the developer full control over when incoming messages are processed. Using event handlers, BPEL processes can handle *multiple incoming messages* as they occur. Alternatively, they can use the *receive activity* to ensure that particular messages are processed only once the business process reaches a given state. These process instances can persist over extended periods of inactivity. A BPEL engine stores such instances in a database, *freeing up resources and ensuring scalability*. BPEL provides *fault handlers* to deal with faults that occur either within processes or in external Web services. Developers can also use *compensation handlers* to undo any previous actions, which gives them an alternative approach to providing a two-phase commit based on distributed transaction support. When a business process instance extends over a long period or crosses organizational boundaries, it's impractical to have transactions waiting to commit. The compensation handler approach is more appropriate in this scenario.

BPEL grew out of WSFL and XLANG and is serialized in XML. There were ten original design goals associated with BPEL:

**Goal 1:** Define business processes that interact with external entities through Web Service operations defined using WSDL 1.1, and that manifest themselves as Web services defined using WSDL 1.1. The interactions are "abstract" in the
sense that the dependence is on portType definitions, not on port definitions.

**Goal 2:** Define business processes using an XML based language. Do not define a graphical representation of processes or provide any particular design methodology for processes.

**Goal 3:** Define a set of Web service orchestration concepts that are meant to be used by both the external (abstract) and internal (executable) views of a business process. Such a business process defines the behavior of a single autonomous entity, typically operating in interaction with other similar peer entities. It is recognized that each usage pattern (i.e. abstract view and executable view) will require a few specialized extensions, but these extensions are to be kept to a minimum and tested against requirements such as import/export and conformance checking that link the two usage patterns.

**Goal 4:** Provide both hierarchical and graph-like control regimes, and allow their use to be blended as seamlessly as possible. This should reduce the fragmentation of the process modeling space.

**Goal 5:** Provide data manipulation functions for the simple manipulation of data needed to define process data and control flow.

**Goal 6:** Support an identification mechanism for process instances that allows the definition of instance identifiers at the application message level. Instance identifiers should be defined by partners and may change.

**Goal 7:** Support the implicit creation and termination of process instances as the basic lifecycle mechanism. Advanced lifecycle operations such as "suspend" and "resume" may be added in future releases for enhanced lifecycle management.

**Goal 8:** Define a long-running transaction model that is based on proven techniques like compensation actions and scoping to support failure recovery for parts of long-running business processes.

**Goal 9:** Use Web Services as the model for process decomposition and assembly.

**Goal 10:** Build on Web services standards (approved and proposed) as much as possible in a composable and modular manner.

BPEL language is an **orchestration** language, not a **choreography** language. The primary
difference between orchestration and choreography is scope. A choreography model provides a larger scope, encompassing all parties and their associated interactions (e.g. a peer to peer model). An orchestration model is between two participants (specifically focusing on the view of one participant).

BPEL’s focus on modern business processes, plus the histories of WSFL and XLANG, led BPEL to adopt web services as its external communication mechanism. Thus BPEL’s messaging facilities depend on the use of the Web Services Description Language (WSDL 1.1) to describe outgoing and incoming messages.

In addition to providing facilities to enable sending and receiving messages, the BPEL programming language also supports:

- A property-based message correlation mechanism
- XML and WSDL typed variables
- An extensible language plug-in model to allow writing expressions and queries in multiple languages: BPEL supports XPath 1.0 by default
- Structured-programming constructs including if-then-elseif-else, while, sequence (to enable executing commands in order) and flow (to enable executing commands in parallel)
- A scoping system to allow the encapsulation of logic with local variables, fault-handlers, compensation-handlers and event-handlers
- Serialized scopes to control concurrent access to variables

2.5 EAI deployment lifecycle

1. Develop: During the development phase, companies design and build services that correspond to specific steps within a business process. Combining these services produces a composite service or application for carrying out a specific business function. For example, a composite service created for updating inventory might involve business services for reserving inventory and updating addresses. A service can be represented in a number of ways to the organization, such as through Web services interfaces. Plan for further development after the initial service is deployed because
requirements will change over time, and managing those changes can increase cost. Proper service metadata management and service versioning allows organizations to productively and costeffectively enhance services and manage the deployment of multiple service versions.

2. Integrate: Once the service is designed and the interface is developed, it typically needs to be integrated with other services or IT systems such as databases, transactional management systems and applications. These integrations often require *transformation of data to map between different data schemas*, as well as *dynamic routing* for connecting the appropriate services at run-time.

3. Orchestrate: Once a few services have been developed, companies can combine them in orchestrated steps to create seamless, reliable process flows. The process of “gluing” services together with flow logic is called orchestration.

4. Secure: Before services are deployed, access to them must be secured. Processes for authorizing and authenticating users, as well as provisioning them and managing their identities, must be mapped out before potentially sensitive information is exposed as a Web service.

5. Manage: Management entails defining and enforcing *service-level* agreements for services, and operational policies for auditing and billing (if appropriate) for service usage. A well-managed SOA also provides the organization with the confidence that their services are reliable, available and constantly monitored for exceptions or failures.

6. Access: Services are typically exposed to users through a portal or a composite Web application, as well as through wireless devices such as cell phones and handheld devices. An SOA environment supports *multi-channel access* to services and enables organizations to adapt user interfaces without modifying the underlying services, making the user experience much more flexible.

7. Analyze: Analysis of services, events and business processes involved in business operations often needs to occur in real time so operational managers and workers can effectively monitor, analyze and respond to time-sensitive issues. This also allows organizations to identify bottlenecks in their processes and alert the appropriate personnel when a particular event
warrants attention. By adhering to this life cycle, organizations may rest assured that their services are being deployed using a common approach that ensures proper security, reliability and availability.
3. Industrial EAI Approaches

The third part of this report refers to the current industrial trends of indicative stakeholders in the EAI domain. It is extremely crucial to be aware of the positioning of the main stakeholders concerning their approach in EAI, since the actual goal of this section is to pinpoint that the key enabling technologies described earlier (e.g. BPEL etc.) are widely accepted. The purpose of this chapter is to outline the industrial approach regarding major stakeholders in the area of Enterprise Application Integration. This executive summary is a proof of concept concerning the utilisation of the new, up-coming key enabling technologies such as Web Services, BPEL etc.

3.1 SAP's Enterprise Services Architecture

**Enterprise Services Architecture (ESA)** is SAP's blueprint for service-based, enterprise-scale business solutions that offer the increased levels of adaptability, flexibility, and openness required to reduce total cost of ownership. It combines SAP's experience in enterprise applications with the flexibility of web services and other open standards. The **SAP NetWeaver platform** is the technical foundation for Enterprise Services Architecture.

ESA elevates the design, composition, and deployment of Web services to an enterprise level to address business requirements. An **enterprise service** is typically a series of Web services combined with simple business logic that can be accessed and used repeatedly to support a particular business process. Aggregating Web services into business-level enterprise services provides more meaningful building blocks for the task of automating enterprise-scale business scenarios.

Enterprise services allow effective **development of composite applications**, which are applications that compose functionality and information from existing systems to support new business processes or scenarios. All enterprise services communicate using Web services standards, can be described in a central repository, and are created and managed by tools provided by SAP NetWeaver.

ESA moves IT architectures step-by-step to dramatically higher levels of adaptability and

---

7 Enterprise Services Architecture: Blueprint for Services-Based Business Solutions
help companies move closer to the vision of the real-time enterprise. The promise of ESA is twofold: *facilitating business innovation* while *leveraging existing* resources. Enterprise Services Architecture takes Web services standards and services-oriented architecture principles and extends them to meet the needs of enterprise business solutions.

The fundamental premise of Enterprise Services Architecture is the **abstraction of business activities** or **events, modelled as enterprise services**, from the actual functionality of enterprise applications. Aggregating Web services into business-level enterprise services provides more meaningful **building blocks** for the task of automating enterprise-scale business scenarios. Enterprise services allow IT organizations to efficiently develop **composite applications**, defined as applications that compose functionality and information from existing systems to support new business processes or scenarios. The order-to-cash business scenario illustrates the benefits of Enterprise Services Architecture.

![Figure 3-1. Order-to-Cash Scenario Without Enterprise Services Architecture](image)

### An Order to Cash Scenario

As Figure 3-1 shows, order-to-cash involves **multiple applications**: customer-facing applications, such as Customer Relationship Management (CRM), supplier-facing applications, such as supply chain management (SCM), and enterprise resource planning (ERP) systems where the order resides, and where all transactions and fulfilment entities are stored. In the typical order-to-cash scenario, employees act as human integrators, sitting in front of many different applications, **transferring information** from one to the next by copying and pasting and retying information, making process-flow decisions as needed.
The applications, when they are communicating, are hardwired through brittle connections that are expensive to maintain.

With Enterprise Services Architecture, a composite application can use enterprise services to automate the flow of information from application to application. Each user in the business scenario has a role-based interface that provides exactly the information and functionality required to meet their goals. The process is defined, controlled, implemented, and managed at a business level, with SAP NetWeaver providing the environment to construct enterprise services to control the flow of information from one enterprise application to the next.

![Enterprise Services Architecture](image)

Figure 3-2, Order-to-Cash with Enterprise Services Architecture

The key characteristics of Enterprise Services Architecture (ESA) listed below are crucial to implementing business scenarios like the order to cash scenario:

- ESA extends the benefits of Web services to enterprise business scenarios by aggregating existing systems functionality into business-level enterprise services.

- Enterprise Services are modelled from an “outside-in” perspective. While the core set of enterprise services identified by a company may be substantially enabled by legacy or enterprise applications (including those from SAP), they are not defined or constrained by SAP or any other vendor’s applications. In other words, Enterprise Services Architecture
defines or models services “outside-in” for any application that is based on business events relevant to enterprise business processes, not necessarily on an existing application or implementation. SAP will evolve its applications to support enterprise services designed for each business domain or industry that it currently addresses.

✅ Enterprise Services Architecture offers a **blueprint for enterprise-wide business process evolution** with complete investment protection.

✅ Enabled by SAP NetWeaver, Enterprise Services Architecture offers a gradual path to flexible, **service-centric system landscapes**. Enterprise Services Architecture allows for a gradual and nondisruptive transition of existing applications and architecture to higher levels of flexibility and value.

✅ Enterprise Services Architecture allows new business processes to be **developed, deployed, and changed independently of existing applications**. “Consumers” of enterprise services are isolated from changes in applications that “provide” the service. Enterprise Services Architecture leverages an abstraction layer between the way an enterprise service is used, and the way the corresponding functionality is implemented within an enterprise application. This abstraction allows composite applications or custom user interfaces (UIs) using the service, or its so-called “consumers,” to be decoupled from the applications “providing” the service.

With Enterprise Services Architecture, enabled by **SAP Net-Weaver**, the language of business becomes the language of IT. Enterprise services are defined at a granularity where they can be understood by **business analysts**, rather than requiring a developer’s intervention. With the model-driven approach of Enterprise Services Architecture, and SAP NetWeaver tools that support the entire life cycle from business process modelling to code generation, a business analyst can “assemble” enterprise services into composite applications that enable new business scenarios. This approach is clearly more efficient and helps customers build, deploy, and maintain solutions with significantly greater agility, cost-efficiency, and speed.

This allows composite applications supporting business scenarios to be built **rapidly**, with a minimum amount of coding. However, this approach is distinguished from older, rapid development paradigms in that it is not merely about creating new modules of code quickly. It is about modelling business processes that can then be assembled by wiring together existing data and business logic across a **heterogeneous landscape**. As SAP’s own development processes evolve from a traditional development paradigm towards solution
assembly, customers can also expect to see a shift in the speed in which SAP delivers new solutions.

Figure 3-3. Key Drivers for the Enterprise Services Architecture Road Map

Enterprise Services Architecture (ESA) is the blueprint for an architecture that enables innovation and standardization in a single environment, allowing IT management to deliver at the speed and efficiency the business requires.

To achieve these goals, SAP has developed ESA Adoption Program, a formalized -- yet flexible -- methodology designed to streamline and simplify the move to ESA. The adoption program is a series of logical steps: **discovery, evaluation, implementation, and operations.** Each step is supported by a compact portfolio of field-tested "enablers," which include a variety of tools, templates, samples, and workshops. The program's flexibility allows the selection of one or two enablers on an as-needed basis.

- **Discovery** -- Tools used during this phase include "opportunity workshops" -- interactive sessions designed to explore the value and cost of various implementation alternatives for ESA. Typically, this is where organization's IT and business leaders work together to brainstorm on the ways and areas in which a service-oriented architecture could foster innovation and lower costs.

- **Evaluation** -- During this phase, SAP works with the end customer to design a company-specific road map. Workshop activities focus on identifying the scope of ESA, the business processes that it might enhance, and the corresponding IT projects that are likely to make a company more competitive and more responsive.

- **Implementation** -- This phase of the adoption program is designed to help the end customer put ESA to work. A key objective during this phase is the selection of the
specific support services that will yield the best results. By focusing on matching needs with services, this phase ensures a speedy, secure implementation while minimizing risks and costs.

✔ **Operations** -- During this phase, SAP provides a number of enablers to help the customer govern the applications and maintain a consistent strategy over time. There's a close focus on ROI during this stage, with SAP providing tools that help measure results and quantify benefits derived from ESA and from SAP NetWeaver as the underlying technology platform.

SAP NetWeaver supports the most important business processes and provides tools to help the understanding on how these processes work. One of these tools is the **solution map** shown in Figure 3-4. Built using input from customers and industry analysts, plus the technical expertise SAP has acquired through extensive business experience and research, **SAP solution maps** are multilevel blueprints of processes. They help the **visualisation, planning and implementation** of a coherent, integrated, and comprehensive information technology solution.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Unification</td>
<td>Master-Data Harmonization</td>
<td>Master-Data Consolidation</td>
<td>Central Master-Data Management</td>
<td>Enterprise Data Warehousing</td>
<td>Enterprise Search</td>
<td></td>
</tr>
<tr>
<td>Business Information Management</td>
<td>Enterprise Reporting, Query, and Analysis</td>
<td>Business Planning and Analytical Services</td>
<td>Enterprise Data Warehousing</td>
<td>Enterprise Knowledge Management</td>
<td>Enterprise Search</td>
<td></td>
</tr>
<tr>
<td>Business Event Management</td>
<td>Business Activity Monitoring</td>
<td>Business Task Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custom Development</td>
<td>Developing, Configuring, and Adapting Applications</td>
<td>Enabling Platform Interoperability</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unified Life-Cycle Management</td>
<td>Software Life-Cycle Management</td>
<td>SAP NetWeaver Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consolidation</td>
<td>Enabling Platform Interoperability</td>
<td>SAP NetWeaver Operations</td>
<td>Master-Data Consolidation</td>
<td>Enterprise Knowledge Mgmt.</td>
<td>Enterprise Data Warehousing</td>
<td></td>
</tr>
<tr>
<td>ESA Design and Deployment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Enabling Enterprise Services</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3-4. ESA Solution Map**
3.2 Oracle Fusion Architecture

Besides SAP’s ESA architecture Oracle also paved the way towards SOA adoption throughout the Oracle Fusion Architecture⁸. The Oracle Fusion Architecture is grounded throughout the Fusion Middleware which will be explained below. Companies worldwide spent billions of dollars in the past decade on monolithic applications like ERP, CRM and supply-chain solutions. More money was spent on the middleware needed to connect these behemoths to legacy systems in order to leverage existing data. While they certainly served the purpose at the time bringing old systems and data into new, Internet-enabled environments these monolithic suites have become burdensome as the pace of business has continued to quicken, for two key reasons:

- **Lack of leverage across existing systems**: Monolithic systems typically orchestrate their own business processes, use their own business rules and logic, and rely upon their own security schemes. These systems are inflexible and inefficient, and often the knowledge housed in one may not be accessible across the enterprise.

- **Costly to maintain**: Many of these monolithic applications must be maintained by trained professionals with highly specialized skill sets. Organizations often can’t leverage this ongoing expense enterprise-wide, particularly in the case of custom-built applications. SOA meets these challenges head-on. Because it is based upon loosely coupled services rather than tightly coupled integrations, service-oriented infrastructures and applications

---

can change as quickly as business needs change. Services can be constructed, deployed and reused virtually on demand, and easily integrated enterprise-wide, across heterogeneous platforms. The implementation of SOA can yield a cost-effective, efficient integration of systems and processes because it lets organizations rationalize and reuse services and easily automate processes based on those services. This directly addresses the two key burdens of existing IT environments lack of leverage across existing systems and high maintenance costs.

According to Oracle philosophy, in order to be successful, an SOA strategy must address every phase of the services life cycle mentioned above, from initial development of services through changes to services in order to meet new business requirements (Figure 3-5).

This can be achieved using the following software components that comprise the Oracle Fusion Middleware Solution. Oracle’s portfolio of solutions for business integration lets companies attack their integration challenges holistically, as part of the SOA life cycle, as well as at the micro-level for those working their way toward an SOA deployment. Key solutions in the family include:

- **Oracle BPEL Process Manager**: The Oracle BPEL Process Manager solution provides seamless process integration that connects applications, IT systems, and business partner interactions into orchestrated, reliable process flows. BPEL, or Business Process Execution Language, is the industry standard for orchestration. BPEL coordinates the execution of multiple discrete services, exposed from any IT system, as an end-to-end process flow. The end result is a composite application that leverages existing systems, yet is orchestrated to enable a new business process. Oracle BPEL Process Manager enables companies to **model, deploy** and **manage processes**. Several characteristics of this solution make it stand out from the competition:

  - **Native BPEL support**: Oracle BPEL Process Manager supports the BPEL standard natively—eclipsing solutions that import and export BPEL. Native BPEL support provides completely portable processes and reduces complexity of managing multiple process definition types.
  - **Extensible binding framework**: In addition to orchestrating existing Web services, including Web services built on JCA (J2EE Connector Architecture) adapters and JMS (Java Message Service), Oracle BPEL Process Manager supports an extensible binding framework that enables connections to IT systems that are not natively Web service- enabled. This allows Oracle BPEL Process Manager to orchestrate functionality of any system as part of a BPEL process.
• **Cross-platform support:** Oracle BPEL Process Manager supports Oracle Application Server, BEA WebLogic, IBM WebSphere, and JBoss, providing maximum flexibility for IT deployments.

• Oracle BPEL Business Process Manager uses a graphical, **model-driven** approach that gives business analysts a visual tool to define, then automate, connect and synchronize business processes across many disparate systems. The technology lets organizations manage key steps in the SOA life cycle, including developing, orchestrating, monitoring and changing business processes. The ability to orchestrate Web services, in particular, is a key linchpin of Oracle’s SOA technology because it is an essential step to changing the application development environment from traditional code development to service-oriented assembly—building applications through the assembly of services is a much faster, more robust method than writing code.

✔ **Oracle Enterprise Service Bus:** Oracle Fusion Middleware includes an Enterprise Service Bus (ESB) infrastructure that provides **messaging**, **routing** and **data-transformation** services geared for service-oriented and event-driven architectures. Oracle Enterprise Service Bus simplifies environments by connecting disparate systems across a common foundation, hereby enabling a reusable framework that helps businesses increase agility.

✔ **Oracle Business Activity Monitoring.** Organizations deploy service-oriented architectures to increase their ability to respond faster and more effectively. Oracle Business Activity Monitoring (Oracle BAM) gives companies the tools they need to build interactive, real-time dashboards and alerts for monitoring business services and processes. Using these solutions, executives and operations managers can get the information that they need to make better business decisions as change occurs.

✔ **Oracle Web Services Manager** administers service-oriented architectures by letting IT organizations centrally define and enforce policies that govern Web services operations such as access policy, logging policy, and content validation. Oracle Web Services Manager supports Web services deployed on any platform, including Java and .NET-based systems, and its policies can be integrated with existing services without modification. Oracle Web Services Manager also collects and displays monitoring statistics in a Web dashboard, improving control and visibility over Web services. Most importantly, Oracle Web Services Manager works hand-in-hand with Oracle BPEL Process Manager. These applications can call each other and share metadata, greatly
simplifying the process of building processes that are secure.

☑ **Oracle Business Rules.** Organizations use business rules when developing new applications because they help meet requirements for agility and transparency. Implementing a business rules solution creates a new level of agility by enabling rules to be modified or customized very quickly without reprogramming. Business rules solutions also enable transparency by allowing auditors or business analysts to view business rules and determine whether an application correctly implements business policies especially critical to insurance, health care and financial services companies that have strict requirements around transparency. Companies can also leverage new opportunities or answer competitive threats while trimming the costs of modifying applications by using business rules.

Oracle Business Rules consists of a rule **authoring tool for defining rules**; an SDK that provides **rules access** and update for embedded programs; and a **rules engine**, where rules are executed. Oracle Business Rules can be **seamlessly integrated with BPEL** flows to create very flexible business processes.

Overall, it provides the infrastructure for the development and deployment of business rules for both the business analyst developing the rules and the programmer integrating rules into business applications. Using these tools within an SOA environment, organizations can:

☑ Increase efficiency

☑ Drive better business decisions

☑ Process more accurate information

☑ Adapt to changing business needs

### 3.3 IBM SOA & Business Patterns

To improve the process over time, during an Information System development, **capturing** and **reusing** the experience of the IT architects is necessary in such a way that future engagements can be made simpler and faster. This can be accomplished by capturing knowledge gained from each engagement and using it to build a **repository of assets**. IT
architects can then build future solutions based on these proven assets. This reuse saves time, money, and effort and helps ensure delivery of a solid, properly architected solution. The IBM Patterns\(^9\) for e-business help facilitate this reuse of assets. Their purpose is to capture and publish **e-business artifacts** that have been used, tested, and proven to be successful.

![Diagram of assets and their relationships](image)

**Figure 3-6. Assets and their relationships**

The Patterns approach is based on a set of layered assets that can be exploited by any existing development methodology. These layered assets are structured in a way that each level of detail builds on the last and include:

- **Business patterns** that identify the interaction between users, businesses, and data.
- **Integration patterns** that tie multiple Business patterns together when a solution cannot be provided based on a single Business pattern.
- **Composite patterns** that represent commonly occurring combinations of Business patterns and Integration patterns.
- **Application patterns** that provide a conceptual layout that describe how the application

\(^9\) IBM Rebooks-Patterns: SOA with an Enterprise Service Bus
components and data within a Business pattern or Integration pattern interact.

- **Runtime patterns** that define the logical middleware structure that supports an Application pattern. Runtime patterns depict the major middleware nodes, their roles, and the interfaces between these nodes.

- **Product mappings** that identify proven and tested software implementations for each Runtime pattern.

- **Best-practice guidelines** for design, development, deployment, and management of e-business applications.

Figure 3-6 shows these assets and their relationships to each other.

A Business pattern describes the relationship between the users, the business organizations or applications, and the data to be accessed. There are four primary Business patterns that are outlined in Figure 3-7.

<table>
<thead>
<tr>
<th>Business Patterns</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self-Service (user-to-business)</td>
<td>Applications where users interact with a business via the Internet or Intranet.</td>
<td>Simple Web applications</td>
</tr>
<tr>
<td>Information Aggregation (user-to-data)</td>
<td>Applications where users can extract useful information from large volumes of data, text, images, and so forth.</td>
<td>Business intelligence, knowledge management, and Web crawlers</td>
</tr>
<tr>
<td>Collaboration (user-to-user)</td>
<td>Applications where the Internet supports collaborative work between users.</td>
<td>Community, chat, videoconferencing, e-mail, and so forth</td>
</tr>
<tr>
<td>Extended Enterprise (business-to-business)</td>
<td>Applications that link two or more business processes across separate enterprises.</td>
<td>EDI, supply chain management, and so forth</td>
</tr>
</tbody>
</table>

**Figure 3-7.** Primary business partners

It would be very convenient if all problems fit nicely into these four slots, but reality says that things can often be more complicated. The patterns assume that most problems, when broken down into their basic components, will fit more than one of these patterns. When a problem requires multiple Business patterns, integration patterns can be utilised. Integration patterns allow the combination of multiple Business patterns to solve a business problem (Figure 3-8).
The Access Integration pattern maps to User Integration. The Application Integration pattern is divided into two essentially different approaches:

- **Process Integration**, which is the integration of the functional flow of processing between the applications.

- **Data Integration**, which is the integration of the information that is used by applications.

Business and Integration patterns can be combined in order to implement installation-specific business solutions called a Custom design. Custom design is achieved after the Service-Oriented Modelling and Architecture (SOMA) design phase.

Service-Oriented Modelling and Architecture facilitates integration with techniques for analyzing legacy applications, custom and packaged, to identify, specify and realize services for use in a service-oriented architecture (Figure 3-9). It breaks out the business functions of each existing application, identifying candidate services that can be used to realize business goals under the new architecture. It also identifies potentially problematic areas and highlights areas where new services need to be developed or sourced from an external provider.
4. Current Trends Pitfalls and the Need for Semantics in EAI

Nowadays, users and professionals have high expectations towards software applications and enterprise application integration. They want to access the content they need, while this content must be accurate and free of redundancy. So, the enterprise applications must be intuitive, easy to use, reusable and extendable, implemented in a short and inexpensive way and within the current IT legacy environment. Enterprise applications and information systems also need to support a more general notion that involves relating the content and representation of information resources to entities and concepts in the real world.

Although, web services technology applied to specific EAI scenarios provide an abstraction and flexibility layer supporting SOA and simplifying the application integration, they are based on exclusively syntactical-oriented technologies, not defining formally the semantics of services' interfaces and of the data structures of the messages web services exchange. The main reason resulting in the failure of the majority of EAI implementations (some articles even account for 70% of EAI projects as failure) is that the semantics of different systems have to be formally defined and integrated at one point.

The lack of formal semantics regarding the applications and services to be integrated makes it difficult for software engineers and developers to manually interconnect heterogeneous applications, impeding the automation regarding application integration, data exchange and complex services composition. Engineers integrating the enterprise application systems have to know the meaning of the low-level data structures in order to implement a semantically correct integration. No formal definition of the interface data exists (Bussler, 2003), which implies that the knowledge of every developer of applications involved in the integration project is assumed to be consistent.

The above-mentioned needs impose the use and interpretation of semantics in enterprise application integration. Semantic interoperability will support high-level, context-sensitive information requests over heterogeneous information resources, heterogeneous enterprise applications, hiding systems, syntax, and structural heterogeneity. This semantically enriched approach eliminates the problem of knowing the contents and structure of information resources and the structure and architecture of heterogeneous enterprise applications.

---

Semantics and ontologies are important to application integration solutions because they provide a *shared and common understanding of data, services and processes* that exist within an application integration problem domain, and facilitate *communication between people and information systems*. By leveraging this concept we can organize and share enterprise information, as well as manage content and knowledge, which allows better interoperability and integration of inter- and intra-enterprise information systems.

In conclusion, the problem that still exists, which the traditional web services technologies are weak to solve, refers to the formalization and the documentation of the semantics related to the interfaces and the data structures of the deployed web services. By applying semantic web technologies to Service-Oriented Architectures, and deploying Semantic Web Services so as to integrate various systems, the notion of *Semantic Web Services enables SOA* is emerging, paving the way to the *semi-automated semantic-based enterprise application integration*. 
References


