



California  
DEPARTMENT OF TECHNOLOGY

# California Enterprise Architecture Framework

## Enterprise Application Integration (EAI) Reference Architecture (RA)

---

Version 1.0 Final  
January 2, 2014



*This Page is Intentionally Left Blank*



# TABLE OF CONTENTS

1	Introduction.....	1
1.1	Purpose .....	1
1.2	Limitations .....	1
1.3	Intended Users.....	2
1.4	Document Organization.....	2
1.5	Future Directions .....	2
2	Enterprise Application Integration Overview .....	3
2.1	Definitions.....	3
2.1.1	<i>Related Terms</i> .....	4
2.2	Business Benefits .....	4
2.3	EAI Patterns and Usage Scenarios .....	5
2.3.1	<i>Data Consistency Integration</i> .....	5
2.3.2	<i>Business Process-Based Integration</i> .....	5
2.3.3	<i>Composite Application Integration</i> .....	6
2.3.4	<i>Related Patterns</i> .....	7
2.4	Key Capabilities of EAI Solution .....	7
2.5	Key Capabilities of an EDI Solution and Convergence of EAI and EDI.....	8
2.6	Components of EAI Solution .....	9
2.7	Front-End Systems in EAI .....	10
2.8	Back-End Systems in EAI .....	11
2.9	Adaptors and Connectors in EAI .....	11
2.10	Enterprise Business Objects and Data Transformations in EAI.....	12
2.11	Enterprise Service Bus .....	13
3	EAI Reference Architecture Description.....	15
3.1	Conceptual View of EAI RA .....	15
3.2	Logical View of EAI RA.....	17
3.3	Deployment View of EAI RA.....	20
4	EAI Implementation Guidelines.....	21
4.1	Areas of Challenge in EAI .....	21



4.2	Tacit Assumptions as Common Integration Pitfall.....	21
4.3	Architectural Mismatch in EAI .....	22
4.3.1	<i>Detecting Architectural Mismatch</i> .....	22
4.3.2	<i>Repairing Architectural Mismatch</i> .....	22
4.3.3	<i>Preventing Architectural Mismatch</i> .....	23
4.4	EAI-Related Patterns and Strategies.....	23
5	Glossary.....	25
6	References.....	26
7	Document History.....	27



## LIST OF FIGURES

<i>Figure 2-1 EAI Data Consistency Integration Pattern</i> .....	5
<i>Figure 2-2 EAI Multi-Step Process Integration Pattern</i> .....	6
<i>Figure 2-3 EAI Multi-Step Process Integration Pattern</i> .....	7
<i>Figure 2-4 EAI Components Overview</i> .....	10
<i>Figure 2-5 Front-End Systems in EAI</i> .....	11
<i>Figure 2-6 Back-End Systems in EAI</i> .....	11
<i>Figure 2-7 Adaptors and Connectors in EAI</i> .....	12
<i>Figure 2-8 Enterprise BOs and Data Transformation in EAI</i> .....	13
<i>Figure 2-9 ESB in EAI</i> .....	14
<i>Figure 3-1 EAI Reference Architecture – Conceptual View</i> .....	16
<i>Figure 3-2 EAI Reference Architecture – Logical View</i> .....	18
<i>Figure 4-1 Taxonomy of Integration Approaches</i> .....	24



## LIST OF TABLES

<i>Table 3-1 High-level interactions among EAI components</i> .....	19
<i>Table 7-1 Document History</i> .....	27



*This Page is Intentionally Left Blank*

# 1 Introduction

Enterprise and technology in the enterprise have been rapidly changing. The change brings about the challenge of integration and interoperability involving disparate systems and applications. The pieces to integrate are often based on disparate technologies and designs where interoperability was not the primary concern. However, technological factors are not the only source of problems. More often than not, the evolving organizational structure of the enterprise provides a fair share of challenges for integration. With increasing complexity of the enterprise, complexity and the number of interacting systems, the problem of making these systems work together in a situation of changing requirements is clearly not easy – and the main manifestations of the difficulties are time, cost, and the rate of failure in the related integration projects.

However, over time a number of approaches have been developed with the goal of making integration and interoperability more effective than in the past. There have been a number of advances that make the integration task easier. On the technology side, growing adoption of technical standards and standardized APIs is a step forward. On the architectural side, growing adoption of the Service-Oriented architectural style promises simpler integration of application components when they are identified using the business/functional perspective. As far as software construction is concerned, experiences with ever growing complexity of software and the need to develop it flexibly has led to growing adoption of incremental, iterative, or agile software methodologies. Last but not least, the domain of Enterprise Application Integration (EAI) gained visibility when considering integration and interoperability challenges at the enterprise level. This document introduces EAI and presents its Reference Architecture (RA) in the context of the California Enterprise Architecture Framework 2.0 (CEAF 2.0).

## 1.1 Purpose

The EAI Reference Architecture document provides guidelines and options for making architectural decisions when implementing EAI solutions.

The objectives for the document include the following:

- To introduce key terms and distinctions relevant for the topic
- To provide inputs for creating or evaluating architectures for EAI
- To identify building blocks (architectural layers, services, components) for integrating elements of an EAI solution
- To communicate the key architectural decisions relevant for creating or evaluating EAI solutions
- To communicate opportunities for solution and/or platform sharing at agency, cross-agency and/or state levels.

## 1.2 Limitations

The document focuses on EAI and related concepts at the enterprise architectural level in the context of CEAF 2.0 as relevant to the Reference Architecture. It is not intended as an exhaustive introduction to EAI or related products available in the market today. Even though intention for the Reference Architecture is to provide examples of realization using specific products used in production in the State, this document is not intended as a product guide.

## 1.3 Intended Users

The primary intended users of this document are Enterprise Architecture practitioners and other architects that contribute to enterprise architecture. This broad group includes architects from other domains/disciplines such as Security, Application, Information, Business, Technology, Infrastructure, and Solution Architects. It is also beneficial to Managers, at senior or operational levels, who are involved with EAI or related areas, such as Service-Oriented Architecture, Cloud Computing, Identity and Access Management, and similar areas.

## 1.4 Document Organization

The EAI Reference Architecture documentation is organized as follows:

- Section “Enterprise Application Integration Overview” provides background for the EAI RA by introducing descriptions and definitions of EAI, discusses the main usage scenario types found in EAI implementations, and identifies architectural components for respective usage scenarios.
- The section “EAI Reference Architecture Description” elaborates RA for EAI using the following architectural views:
  - The Conceptual View ( in the subsection “Conceptual View of EAI ”) introduces the necessary capabilities for a EAI architecture and how they are supported by Architectural Building Blocks (ABBs)
  - The Logical View (in the subsection “Logical View of EAI RA”) describes key interactions among Layers and/or ABBs to realize functionality specific to EAI solutions
  - The Deployment View (in the subsection “Deployment View of EAI RA”) focuses on system topologies and deployment facets of ABBs.
- The section “Glossary” provides description of the terms and abbreviations used in the document
- The section “References” lists publications used for preparation of the document.

## 1.5 Future Directions

Future evolution of the document includes the following steps:

- Addition of an example or examples of existing realization of the EAI RA
- Identification and elaboration of solution sharing opportunities
- Formulation of implementation guidelines for EAI RA

## 2 Enterprise Application Integration Overview

This section provides a description of EAI, including clarification of key terms and concepts. It identifies EAI's intended business benefits and summarizes its main usage scenarios. A set of key capabilities of EAI solution are identified in this section and key components of the solution are described at a high level.

### 2.1 Definitions

Enterprise Application Integration (abbreviated as "EAI") is typically defined as "giving applications that were designed independently the ability to interoperate".

However, the expression "ability to interoperate" should not be read solely in a narrow technical sense. The factors affecting interoperability (and consequently integration) go beyond the simply technical and include the following groups:

- Technical interoperability challenges:
  - At runtime, including interoperability of different programming languages, libraries, or protocols
  - At construction time, including differences between construction processes, development environment and platform, build process and its outcomes
- Design and architectural interoperability challenges:
  - Challenges resulting from potentially different and not necessarily compatible design or architectural choices, including previous choices of different standards that may affect interaction and interoperability
  - Challenges resulting from assumptions about application domain, about the components at the same level of abstraction
  - Challenges resulting from assumptions about the required infrastructure or compatibility issues between various platforms in operation
- Business process and organizational interoperability challenges, resulting from potentially different assumptions and objectives of business functions or processes as implemented in separate organizations which may have worked correctly for the organizations in question in isolation but not when they get integrated.

Successful implementation of EAI requires consideration of all above types of challenges. For discussion of these topics, please refer to the section "EAI Implementation Guidelines". On the purely technical side of EAI, "giving the ability to interoperate" typically involves adopting an *integration framework* composed of a collection of technologies and (technical) services which form the middleware/ application infrastructure that enables integration of systems and applications across the enterprise.

As far as Enterprise Architecture is concerned, EAI solutions address the following primary patterns of Application Integration:

- Data consistency integration
- Multi-step business process integration (support for both short-lived and long-lived integration operations)
- Composite application integration - development of a composite application or a system of systems

These patterns are further discussed in the section “EAI Patterns and Usage Scenarios” later in the document.

### 2.1.1 Related Terms

The term related to EAI is Electronic Data Interchange (EDI). EDI predates EAI and is still widely known. EDI is of interest for EAI Reference Architecture because a number of features of EDI overlap with those in EAI, hence the need to place EDI with respect to EAI.

Electronic Data Interchange (EDI) is defined as “the transfer of structured data, by agreed *message standards*, from one computer system to another without human intervention”. It can be further characterized as follows:

- EDI implies a sequence of messages between two parties (originator and recipient)
- EDI messages contain electronic documents or business data intended to be processed by a recipient computer system
- EDI standards stipulate the format of EDI messages
- Distinction should be made between EDI message standards and transmission, message flow and software used to interpret and process EDI messages

Given the features of EDI described as above, EDI can be justifiably treated as a domain with its own concerns and techniques within the broader and more general area of Enterprise Application Integration. This is how EDI is treated in this document: as an integral part of EAI rather than a separate or isolated architectural domain that would require its own reference architecture.

## 2.2 Business Benefits

The main business objectives supported by EAI include the following:

- **Productivity and flexibility improvements:** EAI supports these by making available quicker and cheaper more business functions and services than the individual systems taken in isolation, or when they are integrated in a limited way. EAI makes it possible to create new functions or modify old ones quickly when compared to the traditional ways of integrating applications.
- **Data and process quality improvements:** EAI supports these by providing a strong motivation and tools in identifying and removing redundancies in data, functions, and/or business processes. EAI also helps share data and functionality across applications, systems, or functional areas.
- **Facilitating organizational mergers and system replacements:** EAI supports these by reducing complexity when substituting and merging of a number of applications or systems. EAI also simplifies integration with systems or applications that are external to the organization, e.g. suppliers’ systems.

When implemented correctly, EAI can bring in significant business benefits. However, introducing an EAI solution is not risk-free. In case of EAI, *most* risks involved and applicable mitigation strategies are not different from standard known managerial and technical approaches to address them. For discussion of risks and mitigation strategies specific to EAI, please refer to the section “EAI Implementation Guidelines”.

## 2.3 EAI Patterns and Usage Scenarios

EAI solutions address the following primary patterns of Application Integration:

- Data consistency integration
- Multi-step business process integration
- Composite application integration

The EAI scenarios are described in the subsections that follow.

### 2.3.1 Data Consistency Integration

The objective of data consistency integration is to make data across all applications consistent. For example, if a customer changes address in one application, that event is pushed out to other applications so that the applications can update their databases with the current data.

The data consistency integration scenario is applicable in the situations where there is no single system of record functioning as a source of data objects, and the applications in question (possibly more than two) remain mutually independent when executing their business functions. Even though there can be and often there is an actual overlap of data, there is no logical dependency between the executing applications (such as one system providing input to the other). The scenario is schematically depicted in the following diagram:

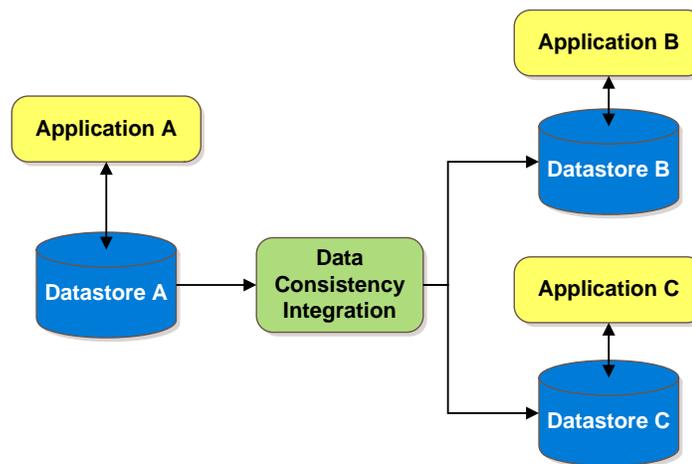


Figure 2-1 EAI Data Consistency Integration Pattern

A typical solution for the scenario is to use dissemination of updates among applications. The dissemination can take the form of batch processing or can be achieved in a more real-time fashion when using event-driven updates and Message-Oriented Middleware. The “Data Synchronization” shown in the above diagram represents the dissemination of updates, either directly or indirectly, using other components for data transformation and to perform the actual updates. Given that this scenario uses a persistent store as its focal component, this approach is sometimes referred to as “database-centric integration”.

### 2.3.2 Business Process-Based Integration

When viewed from integration perspective, execution of a multi-step business process introduces potentially complex logical dependencies between applications that are involved in

the execution of various steps of the business process. During the execution of the process, subsequent steps are executed using functions as provided by existing applications. Depending on the situation, execution of the business process in question can be distributed across many nodes or executed on a single node and the process may be either short-lived or long-lived. Regardless of the properties of the business process however, the applications or systems used in the process become logically dependent because it is unavoidable that outputs from one system will be required to serve as input to another system or systems. The following diagram illustrates this:

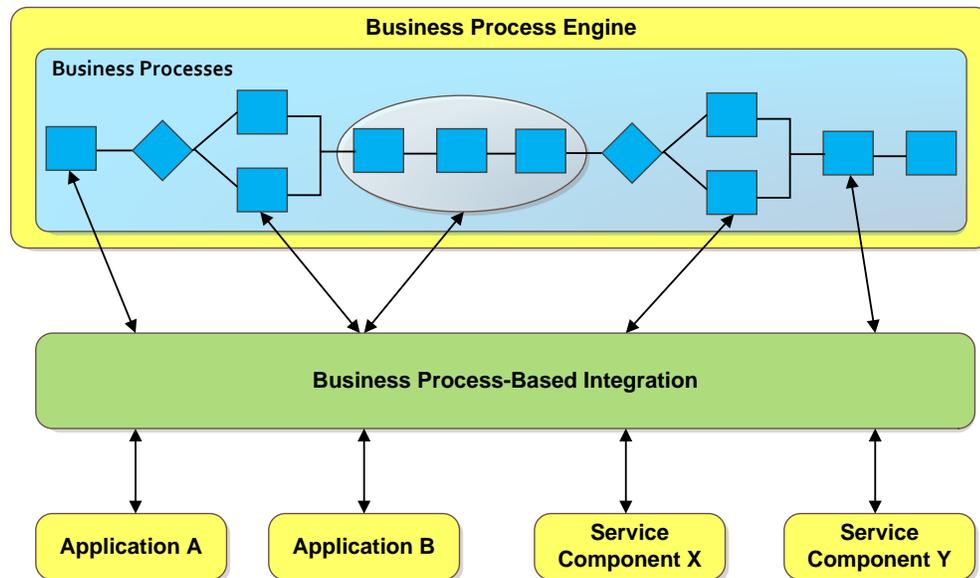


Figure 2-2 EAI Multi-Step Process Integration Pattern

A typical solution is to use either an explicit representation of the business process supported by a business process engine (as shown in the figure above), or – in case of distributed scenarios – broadcasting events representing activity in the steps of the process and handling those events by interested parties (subscribing systems) in the manner specific to those parties.

Business Process-based integration is likely to be the main integration pattern in EAI solutions based on service-oriented architectures.

### 2.3.3 Composite Application Integration

In the Composite application integration scenario, new applications are composed or assembled using either pre-existing or newly constructed components. At least some of the components in the mix make use of separate, non-integrated applications or data stores; consequently, the composite application becomes a vehicle of integration of those applications and data stores with other components.

The Composite Application effectively encapsulates its internal components, not just logically, but also at execution time: When executing, the composite application acts as a whole and it does not finish servicing an external request until all of its internal interactions are completed. The following diagram shows a sample composite application that encapsulates integration between two applications and a subsequent flow to yet another application:

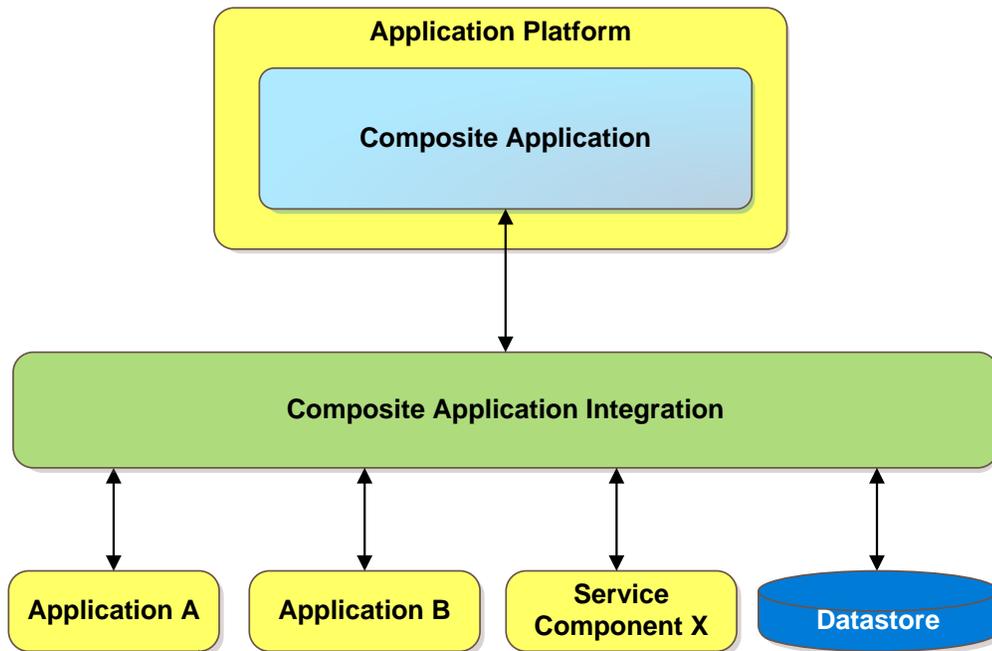


Figure 2-3 EAI Multi-Step Process Integration Pattern

Composite applications can fully encapsulate a business process or become a step in a multi-step business process. In this case, standard connection-oriented technical mechanisms applicable to the applications in questions and their platforms are used to provide connectivity services.

### 2.3.4 Related Patterns

The patterns for EAI listed above are similar to but distinct from the patterns usually brought up when integrating components within single application architecture. The three patterns typically described in that context are data-level integration, business logic-level integration and presentation/UI-level integration. However, these scenarios belong to application architecture rather than enterprise architecture, despite similarity in names and an overlap in the approach to integration and concepts. There are cases when a given integration pattern can be useful both for an enterprise architecture and a corresponding solution architecture – please see the section “EAI-Related Patterns” for a short discussion of these.

## 2.4 Key Capabilities of EAI Solution

The key capabilities of EAI are listed below:

- Communication that reliably moves messages among endpoints
- Support for transformation of canonical documents and messages (XML, EDI, industry standards and WSDL)
- A repository for the storage, browsing and management of message definitions and transformations
- Interoperability with a process execution engine for implementing multistep business process integration

- Support for applying optional intermediary functions and business rules to in-flight messages
- Adapters for applications, databases, A2A and B2B protocols, cloud-to-on-premises application integration APIs, etc.
- Security mediation function for federated identity management
- Interoperability of platform components
- Integration with Enterprise Identity and Access Management
- Integration with Enterprise Information Integration (EII) platform
- Support for fundamental Web and Web services standards
- Support for EDI and domain-specific data/document transfer standards (such as HL7 and NIEM)

## 2.5 Key Capabilities of an EDI Solution and Convergence of EAI and EDI

The key capabilities of EDI include the following:

- Communication that reliably moves EDI messages among endpoints through FTP, VAN (Value Added Networks), VPN, EDIINT AS1, AS2 and AS3, ebXML (including ebMS), RosettaNet RNIF 1.1 and 2.0, cXML, CIDX Chem eStandards 4.0, and SOAP with attachments
- Support for data mapping and transformation between standards-based EDI (ANSI X12, UN/EDIFACT, UCS standards as interpreted by industries such as retail) and internal application data formats
- Business rule execution to validate data and message formats
- A repository for the storage, browsing and management of message definitions and transformations
- Integration with other enterprise applications through SOA, BPM, or through standard message/API based interface
- Support for Transactions
- Interoperability of platform components
- Integration with Enterprise Identity and Access Management for Policy Enforcement and Security Mediation
- Integration with Enterprise Information Integration (EII) platform

Vendors and IT Organizations realized that both EAI and EDI require similar solution capabilities, which include the following:

- Communication
- Data mapping and transformation
- Intermediary functions and business rules
- Repository
- Integration with enterprise applications
- Interoperability
- Overlapping skills

The result is the convergence of functionality in EDI and EAI tools leading to the development of a new class of products capable of handling traditional EDI and EAI. Consequently, this Reference Architecture treats the traditional area of EDI as part of EAI rather than a distinct area of its own.

## 2.6 Components of EAI Solution

In an EAI solution, the following main groups of components can be identified:

- *Connectors* – which are responsible for establishing of a connection between systems and obtaining data through interactions proper to the target system
- *Adaptors* – which are responsible for transformations of data or protocol between systems
- *Enterprise Business Objects (EBOs)* – which are enterprise-level entities representing key data entities in the enterprise
- *Transformation components* – which provide for semantically correct translations between different representations of data
- *Enterprise Service Bus (ESB)* – which provides for a mechanism to deliver and distribute data between data producers on one hand, and data consumers or subscribers on the other hand.

The ESB typically contains additional components as follows:

- Service Registry and Repository
- Repository for Message and Transformation Definitions
- Integration Workflow Execution Engine
- Integration Modules

A typical role for the EAI Platform is to be placed in-between the Front-end Systems and the Back-end Systems and their respective service logic and business processes, as shown in the following diagram:

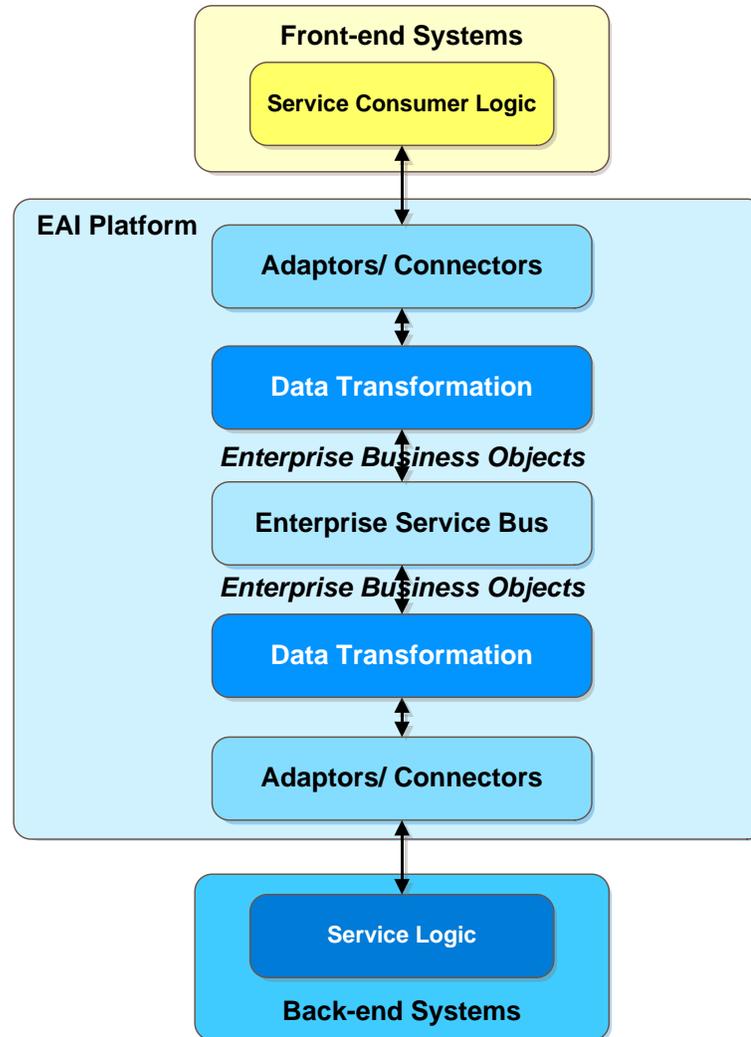


Figure 2-4 EAI Components Overview

The components shown in the above diagram are further described in the subsections that follow.

## 2.7 Front-End Systems in EAI

The Front-end Systems involve, from the EAI's perspective, various kinds of *Consumers* of Service. They contain their Service Consumer Logic which remains de-coupled from service implementation and location. Moreover, the Front-end Systems can discover and consume services through the EAI platform.

Most frequently encountered components of the Front-End Systems include the following:

- Mainframe
- COTS, ERP systems
- Enterprise Content Server (ECS)
- MDM Server
- EDI Gateway
- FTP, SFTP and similar file transfer servers

- Composite applications
- Cloud Service

The above elements are shown in the following diagram:

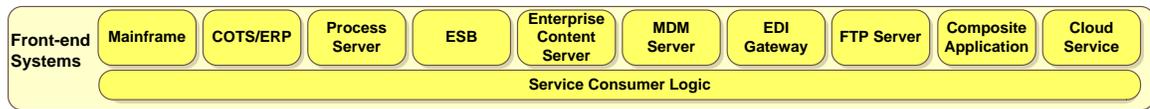


Figure 2-5 Front-End Systems in EAI

## 2.8 Back-End Systems in EAI

The Back-end Systems involve, from the EAI's perspective, various kinds of *Providers of Service*. They contain their Service Logic which is decoupled from all consumers of the Service. Just as the Front-end Systems can discover and consume services through the EAI platform, the Back-end Systems can be discovered and provide services through the EAI platform. In cases when the component providing a service requires authentication, authorization or identity corroboration, it is responsibility of the EAI platform to supply those to the Back-end System.

Typical components of the Back-end systems are shown in the following diagram:

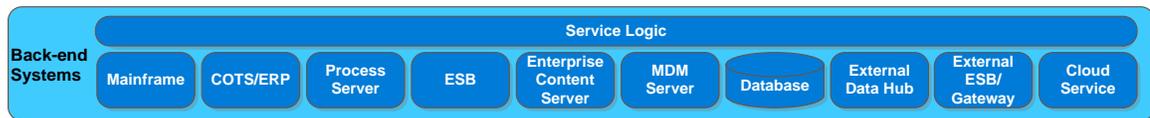


Figure 2-6 Back-End Systems in EAI

Note that a given system may function as a Service Provider or a Service Consumer, depending on the context and the process or transaction in which it participates.

## 2.9 Adaptors and Connectors in EAI

Adaptors and Connectors are components responsible for encapsulating knowledge about the system being integrated using these components. They differ, however. Connectors typically deal with lower level challenges of connectivity, interaction and obtaining data in the format and with the meaning as provided by the Service Producer system. In turn, Adaptors provide for transformations between the APIs of the calling system (Service Consumer) and the Service Provider. Responsibilities of an Adaptor are different and separate from the responsibilities of Connectors or data transformation components, even if the adaptor may use some form of data transformation. Both Connectors and Adaptors can become surprisingly complex, especially when built for complex interactions or APIs using disparate technologies. A number of vendors provide adaptors most often used in a given area, as in the case of e.g., the ESB or in Portals with adaptors to common applications used in the context. Actual implementations of Adaptors can vary from COM and .Net-based languages and libraries, and JVM-based languages and libraries on one hand to Web Services on the other hand.

At the architectural level, Adaptors and Connectors can be characterized as follows:

- They provide a bridge between ESB-aware and ESB-unaware end points
- They allow ESB-unaware clients to use ESB-aware services and ESB-aware clients to use ESB-unaware services

- They are critical for enterprise-wide application interoperability especially when legacy or proprietary COTS applications are involved
- They are required when built-in connectivity from ESB to end point is not available

Note that, in contrast to Connectors, Adaptors either contain or invoke data transformation logic and related business rules.

The typical Adaptors in the enterprise are shown in the following figure:



*Figure 2-7 Adaptors and Connectors in EAI*

The figure shows the following components:

- Message-Oriented Middleware (MOM) Adaptors, such as JMS-based or product-specific adaptor APIs (e.g., for MQ)
- Web Services Adaptors
- EDI Adaptors for specific EDI message formats
- Cloud API Adaptors
- JDBC Connectors for standardized interactions with data stores
- (S)FTP Connectors for interactions with (secure) ftp servers, etc.

## 2.10 Enterprise Business Objects and Data Transformations in EAI

Enterprise Business Objects (EBOs) are application-neutral specifications of entities required for enterprise business processes. It is preferable to have EBOs reflect a single canonical object model rather than to use a number of representations for a single entity. Adopting EBOs has the following advantages for application integration:

- EBOs reduce the number of data transformations required
- EBOs facilitate integration of disparate applications by decoupling service consumer data/message structures from service provider data/message structures

Adoption of EBOs requires transforming some of the data from some sources into the canonical form. This is achieved using Data Transformation Modules, which can be treated as dedicated components to transform data /message structures to/from Enterprise Business Objects.

However, when integrating legacy applications, it is not realistic to expect them to be EBO-aware. Hence the need for Data Transformation components, which are responsible for translating one data representation (typically specific to a given application) into another data representation, possibly a common one, like EBO. The following figure shows the context for Data Transformation components and EBOs in a data-centric integration scenario:

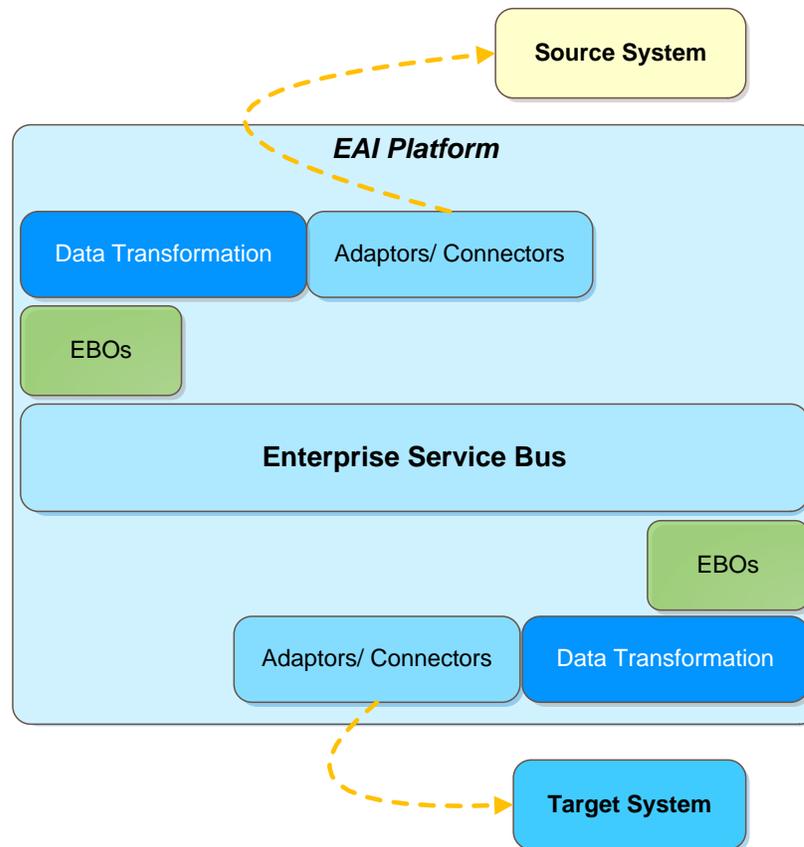


Figure 2-8 Enterprise BOs and Data Transformation in EAI

## 2.11 Enterprise Service Bus

The Enterprise Service Bus in EAI is a focal element of any large-scale integration effort given its responsibility for reliably transporting data and messages from Producers to Consumers. The Bus has a number of responsibilities, including the following:

- Handling messages, which includes validating, enriching, transforming, and similar operations on messages as such (in contrast to their routing, making available, etc.)
- Executing business rules affecting either routing of messages or their transformations
- Enforcing security mediation and supporting policy enforcement
- Supporting execution of integration process flow by using a business process/workflow engine.

The ESB groups a number of components which are crucial for a successful EAI, including the following:

- Service Registry and Repository – where information about available services is stored and available for retrieval and identification
- Message Routing, which can route, in function of the message and external business rules, a given message from the sender to a consumer or to multiple consumers
- Service Invocation, which allows to invoke a particular kind of service (e.g., Web Service) given invocation data and information about the service interface



- Business Process/Workflow engine, which provides for execution of declaratively specified and validated business process definitions
- Aggregations and temporary storage to support technical aspects of ESB operation, such as guaranteed delivery or specific data payload transformations.

The elements pointed out in this subsection are schematically represented in the following figure:

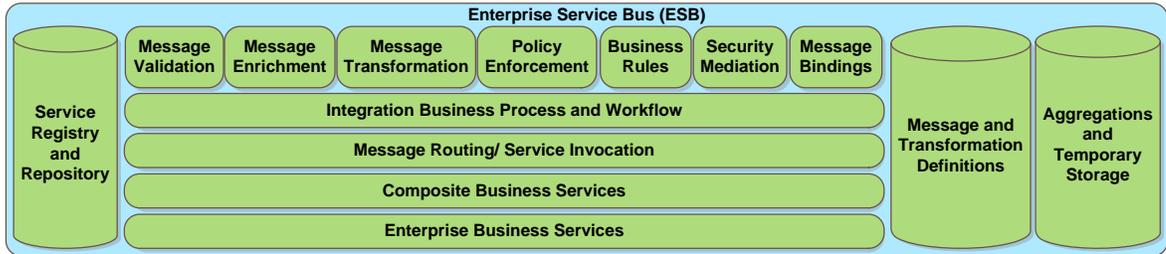


Figure 2-9 ESB in EAI

The parties interacting using the ESB in EAI can be various applications providing dedicated services, or designated Enterprise Business Services, or combinations of the two.

### 3 EAI Reference Architecture Description

This section provides a focused description of EAI Reference Architecture (RA), using three views:

- Conceptual View, which provides a summary of logical-level building blocks for EAI as presented in the Section 2 above
- Logical View, which provides an overview of relationships and interactions between components in an EAI solution for specific usage scenarios
- Deployment View, which illustrates the distribution of processing and components across nodes in the system.

Each of the above views is described in the subsections that follow.

#### 3.1 Conceptual View of EAI RA

The Conceptual View of EAI, shown in the figure below, brings together all major components of an EAI solution that have been described separately in the Section “Components of EAI Solution” earlier in the document. The diagram shows three top-level elements:

- Back-end Systems
- Front-end Systems
- EAI Platform

The EAI Platform element contains the following components portrayed in a layered fashion:

- Adaptors and Connectors
- Data Transformation and Enterprise Business Objects
- ESB and its components

Note the following elements of the Conceptual View:

- ESB is central to the Conceptual View of EAI
- The layering of components above ESB in the view (towards Front-end systems) is symmetrical with the layering of components below ESB in the view (towards Back-end systems): in both cases, the Adaptors and Connectors layer and the Data Transformation and Enterprise Business Objects provide for interactions between the ESB and Back-end Systems on one hand, and between the ESB and Front-end Systems on the other hand
- A number of components present in the Back-end Systems layer are also present in the Front-end Systems layer, given that such components can validly function in either layer. One typical exception is database/persistence components, which usually are not considered as part of Front-end Systems.

Enterprise Application Integration (EAI) Reference Architecture (RA)

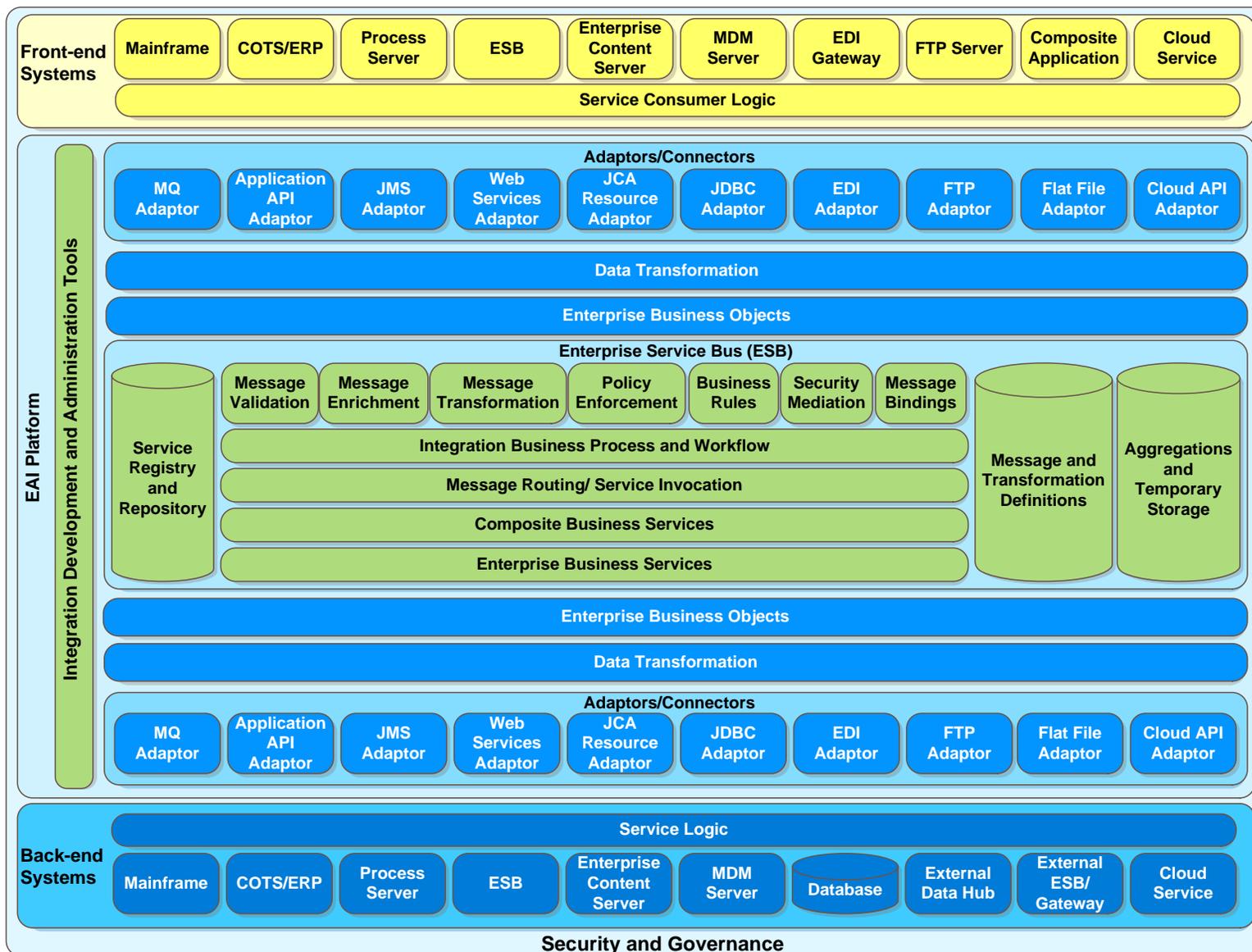


Figure 3-1 EAI Reference Architecture – Conceptual View



## 3.2 Logical View of EAI RA

The following diagram shows high-level typical interactions among components of an EAI solution for a generic *service invocation* scenario. It should be noted that the interactions shown are logical abstractions representing typical interactions. In practice, however, due to the specific *integration pattern* employed and due to the physical packaging of EAI logical components and services in the *specific system software products* selected, the component interactions (and also the names of components and services) may change. Therefore, this section is intended to enable the reader understand how EAI components logically realize a given scenario or use case, and use that knowledge to evaluate specific technology choices to provide desired capabilities.

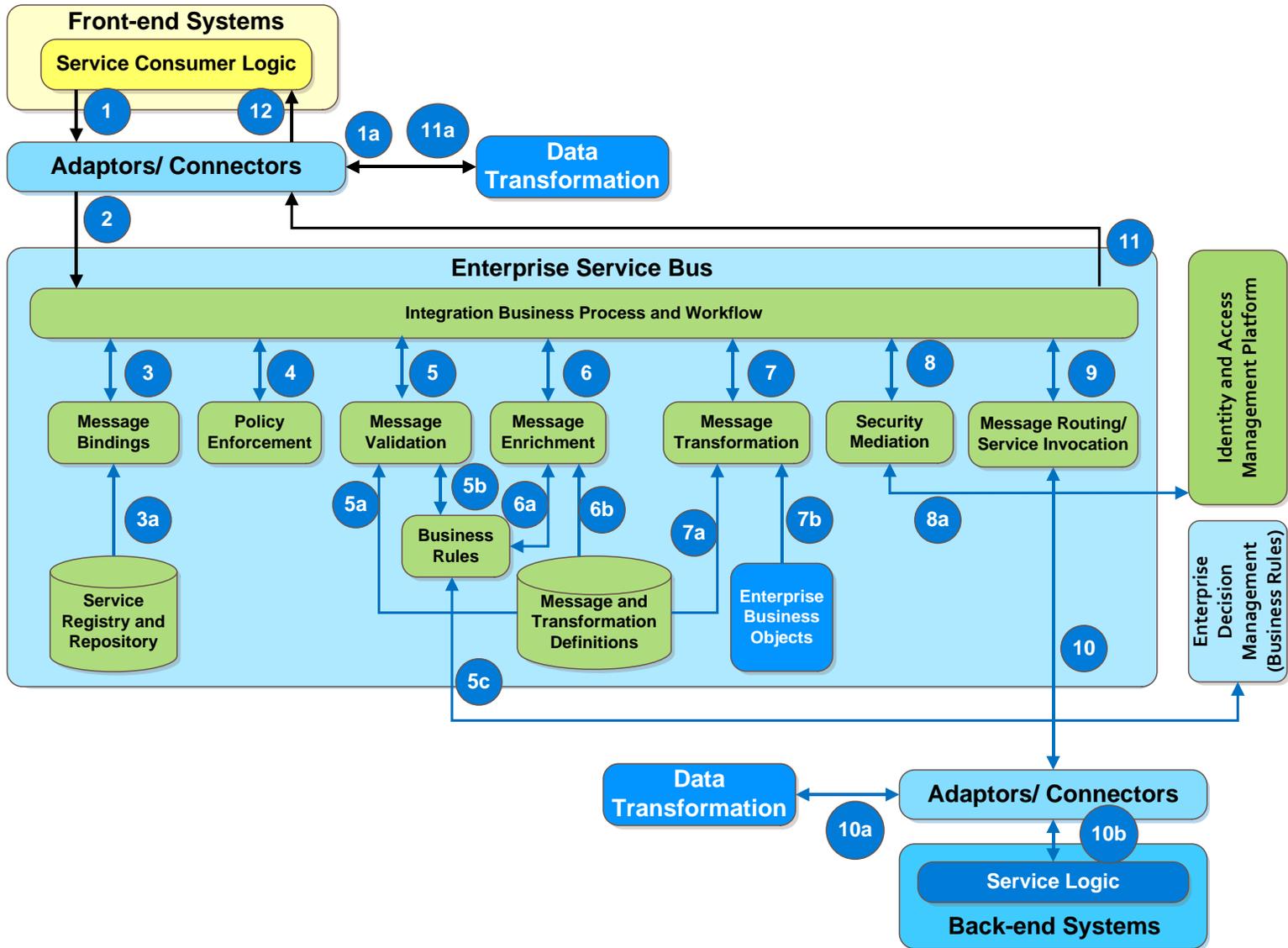


Figure 3-2 EAI Reference Architecture – Logical View

The following table summarizes interactions portrayed in the above diagram.

Table 3-1 High-level interactions among EAI components

Label	Description
(1)	A <i>Front-end System</i> sends a request for invoking a service through its <i>Service Consumer Logic</i> to the corresponding <i>Adaptor/ Connector</i> . If the front-end system is an ESB-aware system, an <i>Adaptor</i> is not necessary.
(1a)	The <i>Adaptor/ Connector</i> performs <i>Data Transformation</i> according to pre-defined rules typically using a <i>data transformation component</i> . This data transformation is done to convert the internal representation of data (in the front-end system) to the <i>externally published request or message format</i> of the service.
(2)	The <i>Adaptor/ Connector</i> sends the formatted service request to the <i>Enterprise Service Bus (ESB)</i> using a protocol/format through which the service is made available (published). It should be noted that this service request may be intercepted by a <i>Service Gateway</i> in a distributed environment. In this case, the gateway performs basic security checks before routing the request to the ESB. Once the request reaches the ESB, the <i>Integration Business Process and Workflow</i> is initiated.
(3), (3a)	During this step, the message bindings and information in the <i>Service Registry/ Repository</i> will be used to identify the target service including its location, invocation protocol etc.
(4)	The ESB enforces any defined security policies by invoking the <i>Policy Enforcement</i> component before further validating, enriching, transforming and routing the request to the service. The <i>Policy Enforcement</i> component may interact with the <i>Enterprise Identity and Access Management platform</i> for policy decisions (not shown in figure).
(5), (5a), (5b), (5c)	ESB validates the request message as a whole. <i>Message Validation</i> is performed to confirm the message payload to the applicable pre-defined message format and to validate the data items in the payload confirm to the data items specified in the message format. In addition to validating the message using definitions such as an XSD schema definition, the ESB may execute pre-defined <i>Business Rules</i> to further validate the data items in the message. Execution of <i>Business Rules</i> may be performed by interacting with the <i>Enterprise Decision Management platform</i> .
(6), (6a), (6b)	During this step, the request message may be enriched, for example, to add additional attributes (typically derived attributes) required by the target service and/or the attributes associated with quality of service.
(7), (7a), (7b)	During this step, the ESB transforms the incoming message to a generic structure which includes transforming the data items from the request

	message to the application-neutral data model represented by the enterprise business objects, if this transformation is not done by the adaptor/converter.
(8), (8a)	During this step, a security token will be created in a format required by the target service by interacting with the <i>Enterprise Identity and Access Management platform</i> .
(9)	During this step, ESB constructs the request message in a format in which the target service accepts the requests. If the target service is a composite service, the original request will be broken down into multiple requests. ESB then routes the message to the target service. It should be noted that the original request for service may be for a composite service, in which case, ESB invokes the elementary services that constitute the composite service using the pre-defined service invocation flow.
(10), (10a), (10b)	The <i>Adapter/Connector</i> corresponding to the target service receives the request, performs data transformation necessary, invokes the target service and sends the response back to the ESB.
(11)	ESB sends the response back to the Adaptor/Connector of the Front-end System.
(11a)	Adaptor/Connector performs the data transformation necessary to convert the response into the format the required by the front-end system.
(12)	The front-end system receives the response.

### 3.3 Deployment View of EAI RA

This subsection is to be completed in a future release. It is intended to show best-practice-based system topologies and implementation patterns of EAI, based on existing realizations of the EAI RA in the state.



## 4 EAI Implementation Guidelines

This section is intended to provide descriptions of challenges and pitfalls in the EAI domain and basic guidelines that can help implement an EAI solution successfully.

The section is expected to grow over time in subsequent releases of the document.

### 4.1 Areas of Challenge in EAI

As already indicated in the section 2 earlier in the document, the goal of integrating applications involves a number of interoperability challenges, related to technology, existing design and architectures, and *also* business process and organizational changes.

EAI experience shows that it is technology-related challenges that are most readily identified when considering integration and interoperability issues in the systems to be integrated. This is obviously a crucial area in EAI, but not the *only* one. There are additional areas of investigation when introducing EAI: design and architectural interoperability challenges, and Business Process and organizational interoperability challenges. Ignoring or downplaying these *other* challenges can have a significant negative effect on an EAI implementation.

### 4.2 Tacit Assumptions as Common Integration Pitfall

There is a common facet in most types of challenges encountered in EAI that affects the rates of success/failure in interoperability and integration solutions: it is *tacit assumptions* that were made (possibly validly) in the context of an application or a system to be integrated considered on its own as contrasted with the new context imposed by the integration. Those tacit assumptions introduce significant risk when integrating, especially as long as they *remain* tacit.

There are a number of general categories of assumptions affecting interoperability and integration, as follows:

- Assumptions about the basic characteristics of the components being integrated, including their control and transactional models
- Assumptions about the nature of available connectors and adaptors and their ramifications
- Assumptions about the overall architectural structure within which the integration is to take place
- Assumptions about available software construction processes, including development environment and platform, testing approach, build environment, etc.

It is a good practice to include examination of potential tacit assumptions of the kinds listed above in any EAI effort. Without proper planning, such assumptions may possibly remain hidden and not identified until late in the implementation of EAI, for example, until the stage of system test, or even until production if the testing of the EAI implementation is insufficient. Discovering these so late in the process is not desirable from the cost and time frame point of view.

Taking components as an example, focusing early on the following facets of the integration mitigates risks resulting from unidentified tacit assumptions:

- What are the respective assumptions and dependencies of the components in question with respect to the infrastructure on which they depend?
- What are the assumptions about the components in question in the applications and systems that use these components?

- What are the assumptions about the components in question that can be extracted from the way peer components interact one with another?

An analogous approach applies to identifying tacit assumptions in the area of architecture of the systems being integrated, and when considering integration of business process.

### 4.3 Architectural Mismatch in EAI

In the context of integration efforts at enterprise level, it is useful to introduce the notion of “architectural mismatch”, which gained some currency in literature of the topic and in practice. The creators of the term defined “architectural mismatch” as the phenomenon of having “incompatible assumptions that each party had made about its operating environment” [14]. It is important to note that even though the assumptions are likely to be valid within the original constraints and scope of the relevant system, they may become invalid in a broader scope or be simply incompatible with a similar set of (architectural) assumptions that were made in other systems. Moreover, the assumptions in question may be known only partially, or are misidentified.

The guidelines to deal with architectural mismatch that can be offered can be grouped as follows:

- Detecting the mismatch
- Repairing the mismatch
- Preventing the mismatch

The above areas are discussed in the subsections that follow.

#### 4.3.1 Detecting Architectural Mismatch

Detecting architectural mismatches remains a domain of tentative heuristics rather than well-established procedures. The areas for examination when trying to detect mismatches include the following:

- The (original) business context of the system in question, which may help identify crucial capabilities and assumptions for the system
- Existing architectural and design documentation – ranging from informal textual descriptions to more rigorous notations (e.g., using UML, ArchiMate, WDSL, programming language APIs, etc.). Depending on the age and rigor of the description, the expected usefulness of the documentation may vary.
- Existing interfaces with other systems or application, which may help describe the actual working interaction mechanisms
- Passive collecting of data or payloads that get processed by the system being examined as a result of specific actions of interest
- Active testing of the interactions, e.g., by using targeted interactions tests, to corroborate existing understanding of assumptions in a given system

#### 4.3.2 Repairing Architectural Mismatch

There are a number of architectural and software design patterns that are useful when repairing architectural mismatch. The most frequently used patterns include the following:

- Adapters or Wrappers, which provide a compatible interface to a component with incompatible interface/implementation

- Mediators, which encapsulate how a set of components interact
- Bridges, which decouple the interface of a component from its implementation.

The above patterns, used in isolation or used in combination, typically provide an adequate way of repairing a mismatch. Note also that these patterns, even though they originate in Object-Oriented languages, are also applicable as architectural patterns applicable to architectural building blocks, rather than only programming language-level artifacts.

In context of SOA, however, wrapping a system or an application as a Service is of special interest, because it fits the overall CEAF architectural style and at the same time allows for architectural decoupling with important practical ramifications: it allows for future replacement or modifications of the service-encapsulated system without affecting the service-imposed contract and the systems that rely on that contract.

### 4.3.3 Preventing Architectural Mismatch

The most effective ways of preventing the architectural mismatch typically involve the following approaches:

- Constraining of the architectural and design choice space, by adopting, correspondingly, applicable reference architectures and applicable design patterns
- Explicitly documenting architectural and design decisions that were made in the construction of the system or that were discovered in the examination of the system
- Identify solution-level integration patterns used in a solution (see e.g. [15] for an extensive collection of such patterns)
- Adopting and following well-known standards or quality, well-described APIs
- Documenting and keeping up-to-date specifications of the system using a standard applicable notation, such as UML or ArchiMate, accompanied by structured natural language descriptions.

In case of enterprise architectures, the preferred specification language is ArchiMate, and the preferred reference architectures are those specified by CEAF 2.0.

## 4.4 EAI-Related Patterns and Strategies

There are a number of publications related to EAI *patterns*. In many, if not most cases, the patterns described in those sources are arguably solution architecture patterns rather than enterprise architecture strategies for EAI. Compared to the EAI patterns, the concept of “integration strategies” in EAI is much broader, because it encompasses not only patterns, but also processes, techniques and solutions applicable to EAI. A tentative taxonomy of such architecture integration strategies presented in [8]. This is notable because the classification presents over 25 specific EAI approaches that are derived from composing (with additional constraints) just three core integration strategies:

- **Controller** “coordinates and mediates the movement of information between components using some predefined decision-making process”. The controller coordinates interactions explicitly and it needs to be aware of identities of the components involved in the interaction.
- **Translator** “converts data and functions between component formats, without changing the content of the information”, and without being aware of the source or destination of the data [8].

- **Extender** adds new features and functionality, and enhances component capabilities. However, extending components capabilities typically means involving a variation of Controller or Translator or both.

The following figure shows a partial taxonomy of integration approaches, adapted from [8].

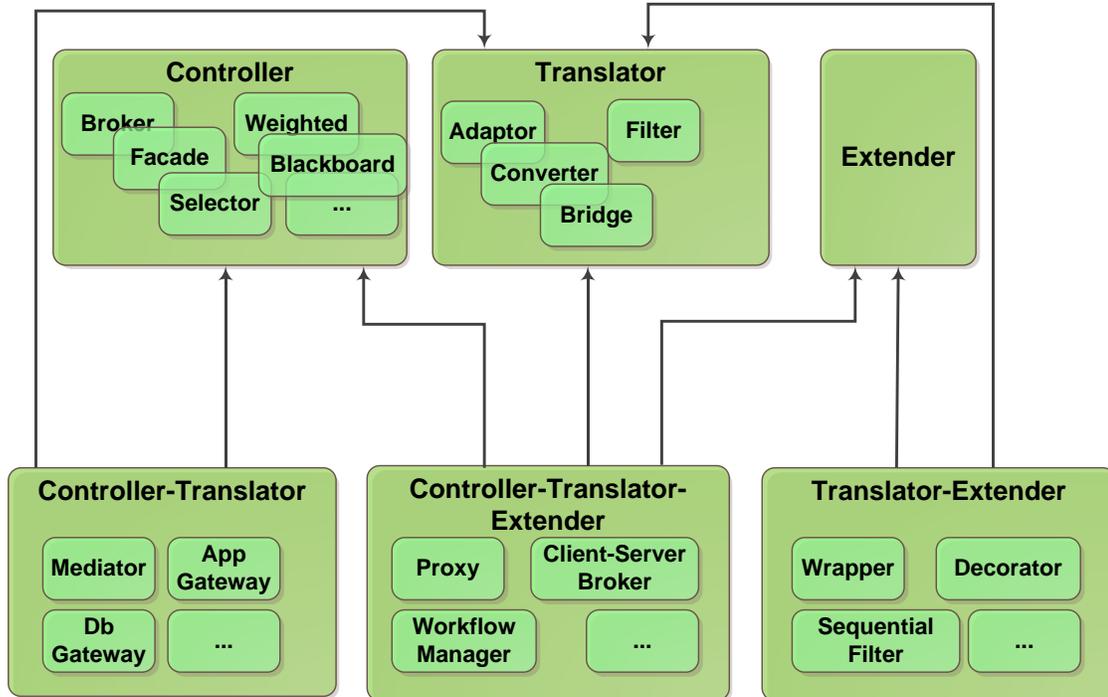


Figure 4-1 Taxonomy of Integration Approaches

The above figure shows core integration strategies (Controller, Translator, Extender) and mixed strategies (Controller+Translator, Translator+Extender, Controller+Translator+Translator). Also, some of the more specific variations of a strategy, targeted at addressing given constraints in an integration problem, are shown. For example, Mediator is a variation of Controller+Translator strategy where loose coupling is required in order to coordinate interactions between groups of objects; Wrapper strategy is a variation of Translator+Extender that can be used to encapsulate a legacy system and to provide new functions not present in the encapsulated system, in a way that is transparent to the user [8]. However, adopting such form of conceptualizing integration strategies is subject to future discussions and feedback from Enterprise Architects.

## 5 Glossary

**Adaptors** are small, focused programs that expose functionality and/or data in a legacy application.

**Application Architecture** defines the major applications or service components needed to manage data and support business functions.

**Architecture** is a set of design artifacts, or descriptive representations, which is relevant for describing an object such that it can be produced to requirements (quality) as well as maintained over the period of its useful life (change). [John Zachman & adopted by the Federal Chief Information Officer Council]

**Connectors** are small, focused programs that encapsulate logical and physical connections between separate components that need to interoperate.

**Data Transformations** are focused programs that encapsulate changing the schema or properties in the input data to produce the desired output; they may involve EBOs as source or the target of the performed transformations.

**EAI**, or Enterprise Application Integration, is giving applications that were designed independently one from another the ability to interoperate in order to carry out a desired business function or process.

**EBOs**, or Enterprise Business Objects, are technology-neutral specifications of entities required for enterprise business processes.

**EDI**, or Electronic Data Interchange, is the transfer of structured data, by agreed message standards, from one computer system to another without human intervention.

**Reference Architecture** models the abstract architectural elements in the domain independent of the technologies, protocols, and products that are used to implement the domain.

**Service Component** is an actual application, program or subsystem providing implementation of a Service treated as a contract

## 6 References

### State and Federal Documents

- A. State of California, California State Information Technology Strategic Plan, November 2004
- B. State of California, California Performance Review Report
- C. Chief Information Officers Council, Federal Enterprise Architecture Framework, Version 1.1, September 1999
- D. Chief Information Officers Council, A Practical Guide to Federal Enterprise Architecture, Version 1.0, February 2001
- E. Federal Enterprise Architecture Program Management Office, The Business Reference Model, Version 2.0, June 2003
- F. Federal Enterprise Architecture Program Management Office, The Service Component Reference Model, Version 1.0,
- G. Federal Enterprise Architecture Program Management Office, The Technical Reference Model, Version 1.1, August 2003
- H. Federal Enterprise Architecture Program Management Office, The Data Reference Model, Version 1.0, September 2004

### Books and Papers

1. B. Gold-Bernstein, W. Ruh, Enterprise Integration: *The Essential Guide to Integration Solutions*, Addison-Wesley Professional 2004
2. Themistocleous, M., and Irani, Z., *Benchmarking the Benefits and Barriers of Application Integration*, Benchmarking: An International Journal, 2001, Vol.8, No. 4, pp.317-31.
3. Xu He; Hongqi Li; Qiaoyan Ding; Zhuang Wu, *The SOA-Based Solution for Distributed Enterprise Application Integration*, Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on , vol.3, no., pp.330,336, 25-27 Dec. 2009  
doi: 10.1109/IFCSTA.2009.321
4. Gleghorn, R., *Enterprise application integration: a manager's perspective*, *IT Professional* , vol.7, no.6, pp.17,23, Nov.-Dec. 2005, doi: 10.1109/MITP.2005.143
5. Garlan, D.; Allen, R.; Ockerbloom, J., *Architectural Mismatch or why it's hard to build systems out of existing parts*, *Software Engineering, 1995. ICSE 1995. 17th International Conference on* , vol., no., pp.179,179, 23-30 April 1995
6. Garlan, D.; Allen, R.; Ockerbloom, J., Architectural Mismatch: Why Reuse Is Still So Hard, *Software, IEEE* , vol.26, no.4, pp.66,69, July-Aug. 2009, doi: 10.1109/MS.2009.86
7. G. Hohpe, B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison Wesley 2003
8. R. Keshav, R. Gamble, Towards a taxonomy of architecture integration strategies, in *Proceedings of the third international workshop on Software architecture*, 1998, pp. 89–92.
9. G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, 2004

### Web Sites

- a. California State Chief Information Officer, California Information Technology Council, Committees <http://www.cio.ca.gov/ITCouncil/Committees/Committees.html>
- b. Gartner IT Glossary, <http://www.gartner.com/it-glossary>
- c. Information service patterns series by D. G. Sauter, <http://www.ibm.com/developerworks/webservices/library/ws-soa-infoserv1/index.html>



## 7 Document History

*Table 7-1 Document History*

Release	Description	Date
Version 1.0 Draft	Initial creation	06/03/2013
Version 1.0 Second Draft	Revised based on internal review comments	06/21/2013
Version 1.0 Final Draft	Addressed EAC review comments	10/21/2013
Version 1.0 Final	Final version	01/02/2014