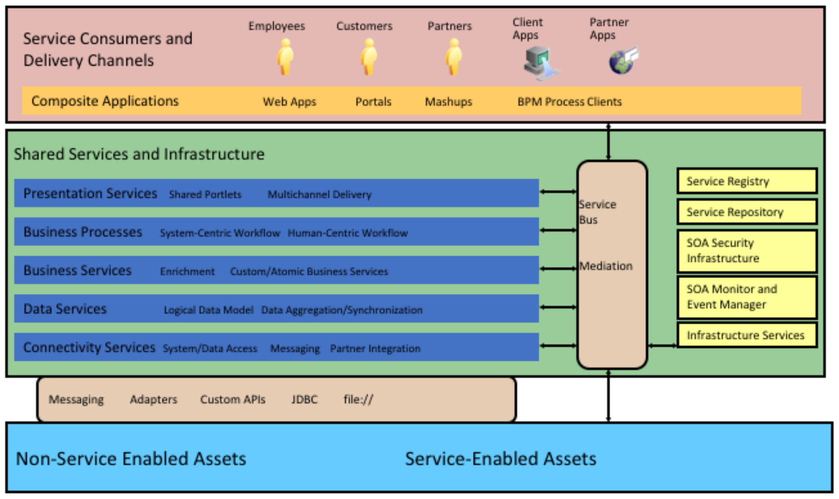
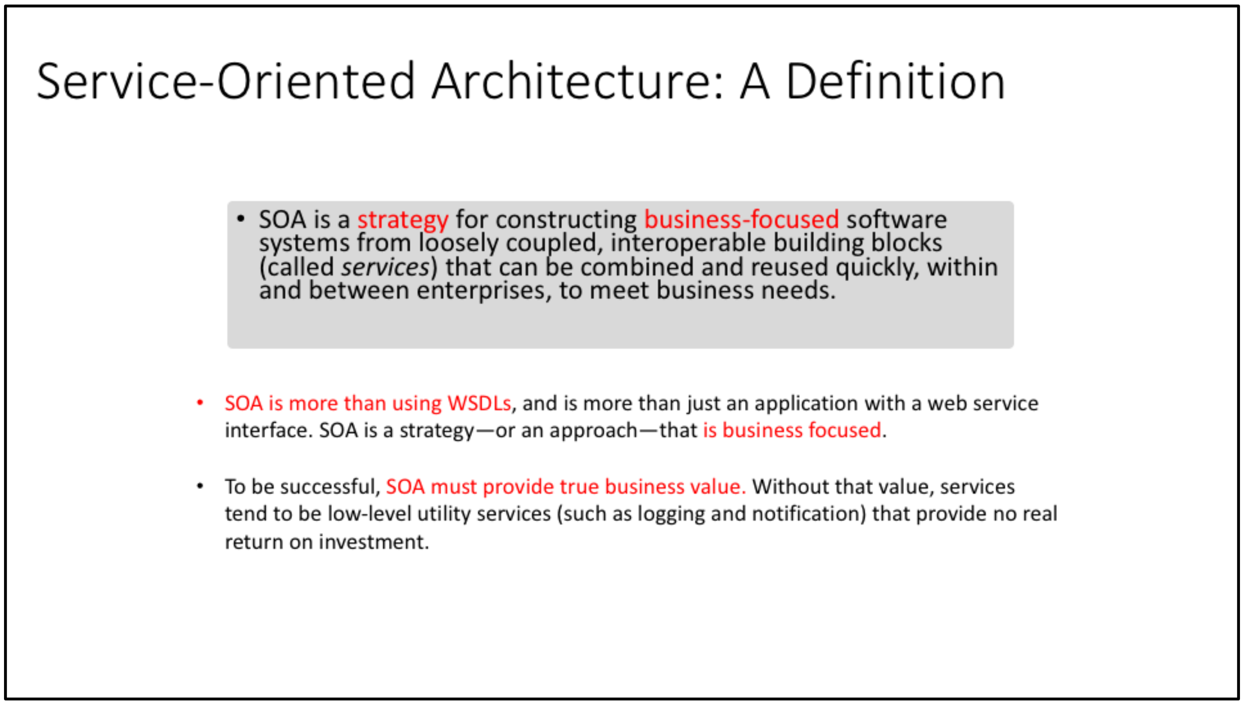
SOA Reference Architecture





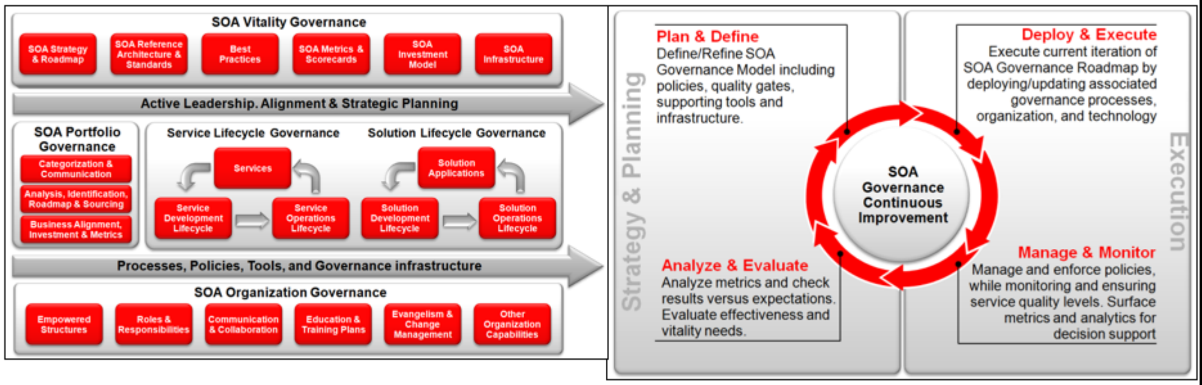
Service-Oriented Architecture: A Definition

- SOA is a strategy for constructing business-focused software systems from loosely coupled, interoperable building blocks (called *services*) that can be combined and reused quickly, within and between enterprises, to meet business needs.

- SOA is more than using WSDLs, and is more than just an application with a web service interface. SOA is a strategy—or an approach—that is business focused.
- To be successful, SOA must provide true business value. Without that value, services tend to be low-level utility services (such as logging and notification) that provide no real return on investment.

SOA is more than using WSDLs, and is more than just an application with a web service interface. SOA is a strategy—or an approach—that is business focused. To be successful, SOA must provide true business value. Without that value, services tend to be low-level utility services (such as logging and notification) that provide no real return on investment. Traditional SOA principles such as loose coupling, interoperability, composability, and reuse are key traits of SOA, but only because they contribute to the business value of an SOA.
Within the scope of the architectural strategy, SOA describes an approach for enterprise systems development and integration that is both technology agnostic (that is, it operates across heterogeneous systems) and aligned with business imperatives. It provides loose coupling of coarse-grained components (services) for rapid and effective reuse of enterprise IT assets. The ultimate goal of SOA is to facilitate the creation of new business solutions through the composition of services—without the need for complex programmatic code development that might otherwise duplicate existing capabilities. This leads to a more agile and efficient enterprise that can respond more rapidly to changing market and regulatory demands.

SOA Organization and Governance Planning: Establish an SOA Governance Framework

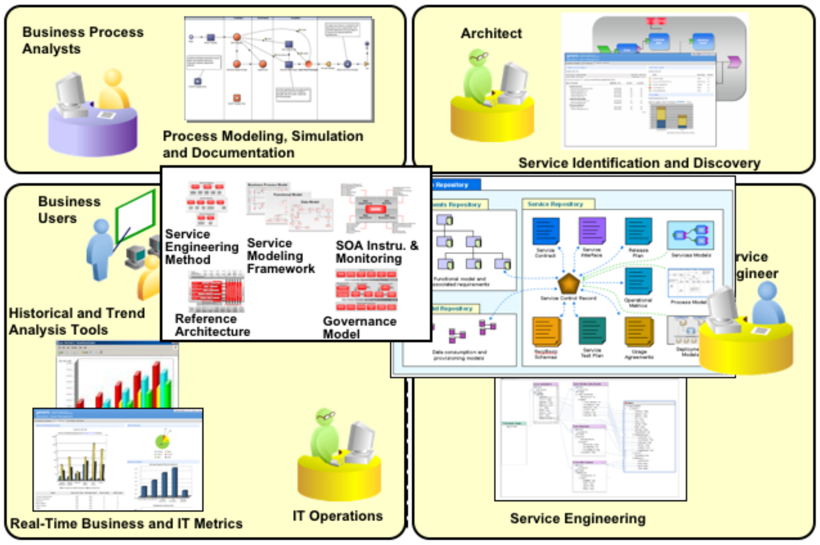

SOA Vitality Governance
SOA Strategy & Roadmap
SOA Reference Architecture & Standards
Best Practices
SOA Metrics & Scorecards
SOA Investment Model
SOA Infrastructure
Active Leadership, Alignment & Strategic Planning
SOA Portfolio Governance
Categorization & Communication
Analysis, Identification, Roadmap & Sourcing
Business Alignment, Investment & Metrics
Service Lifecycle Governance
Services
Service Development Lifecycle
Service Operations Lifecycle
Solution Lifecycle Governance
Solution Applications
Solution Development Lifecycle
Solution Operations Lifecycle
Processes, Policies, Tools, and Governance infrastructure
SOA Organization Governance
Empowered Structures
Roles & Responsibilities
Communication & Collaboration
Education & Training Plans
Evangelism & Change Management
Other Organization Capabilities
Strategy & Planning
Execution
Plan & Define
Define/Refine SOA Governance Model including policies, quality gates, supporting tools and infrastructure.
Deploy & Execute
Execute current iteration of SOA Governance Roadmap by deploying/updating associated governance processes, organization, and technology
SOA Governance Continuous Improvement
Analyze & Evaluate
Analyze metrics and check results versus expectations. Evaluate effectiveness and vitality needs.
Manage & Monitor
Manage and enforce policies, while monitoring and ensuring service quality levels. Surface metrics and analytics for decision support

Because every enterprise is different, there is no single model of good SOA governance. Oracle has a governance framework that has two aspects:

- The actual baseline model
- A continuous feedback loop

It is important to understand that governance is not a one-off project. It is something that you start, and it continues forever. The SOA Governance Continuous Improvement Method is a definition-improvement feedback process to define a focused and customized SOA Governance model.

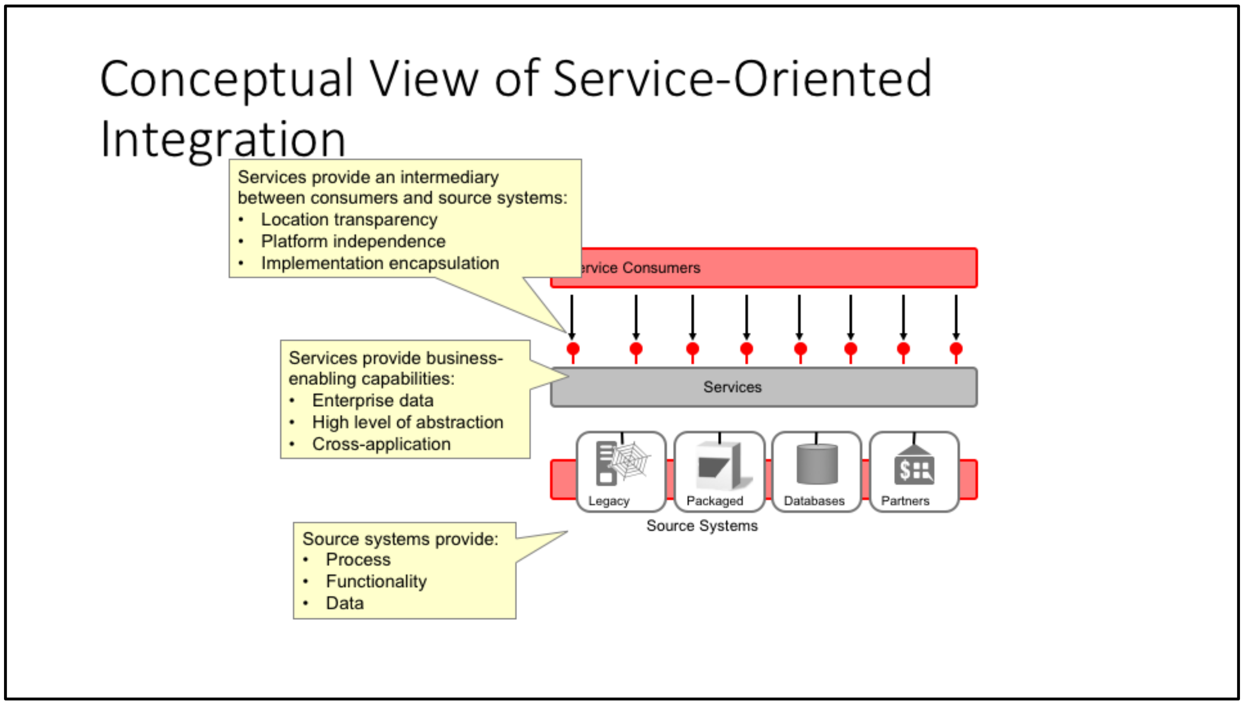
SOA Execution: A 360-Degree View


Business Process Analysts
Process Modeling, Simulation and Documentation
Architect
Service Identification and Discovery
Business Users
Historical and Trend Analysis Tools
Service Engineering Method
Service Modeling Framework
SOA Instru. & Monitoring
Reference Architecture
Governance Model
rvice gineer
Real-Time Business and IT Metrics
IT Operations
Service Engineering

Good communication and access to information are often among the missing pieces of an SOA initiative. When you execute SOA, it is important to have a 360-degree view of the whole software development life cycle (SDLC) of the program-level activities. Rather than just building one big application, an SOA initiative involves building multiple applications and potentially hundreds of services. Each service is made up of a contract, an interface, and a payload. As work is handed from one department to another and from one person to another, a missing part can become a key issue.

The execution phase covers the different life cycles of delivering solutions and services and the associated service infrastructure. An enterprise repository provides a single source of truth for the SOA portfolio and manages SOA assets, projects, and associated metadata. With a repository, you can make rapid informed decisions about dependency tracking, impact analysis, usage tracking, and compliance.

Within the life cycle, analysts capture requirements and perform process, functional, and data modeling. Architects then identify and discovery the appropriate services and define the associated service contracts. Service engineers can design, test, and deliver the services. IT Operations then provides facilities to provision, monitor, and manage services. The life cycle is then complete. Business and IT users can utilize tools to visualize historical and trend analysis. That information gets fed into planning for version 2.0 of a service. Version 2.0 of an application if usually released after 12–18 months. But a service might be versioned quite frequently.



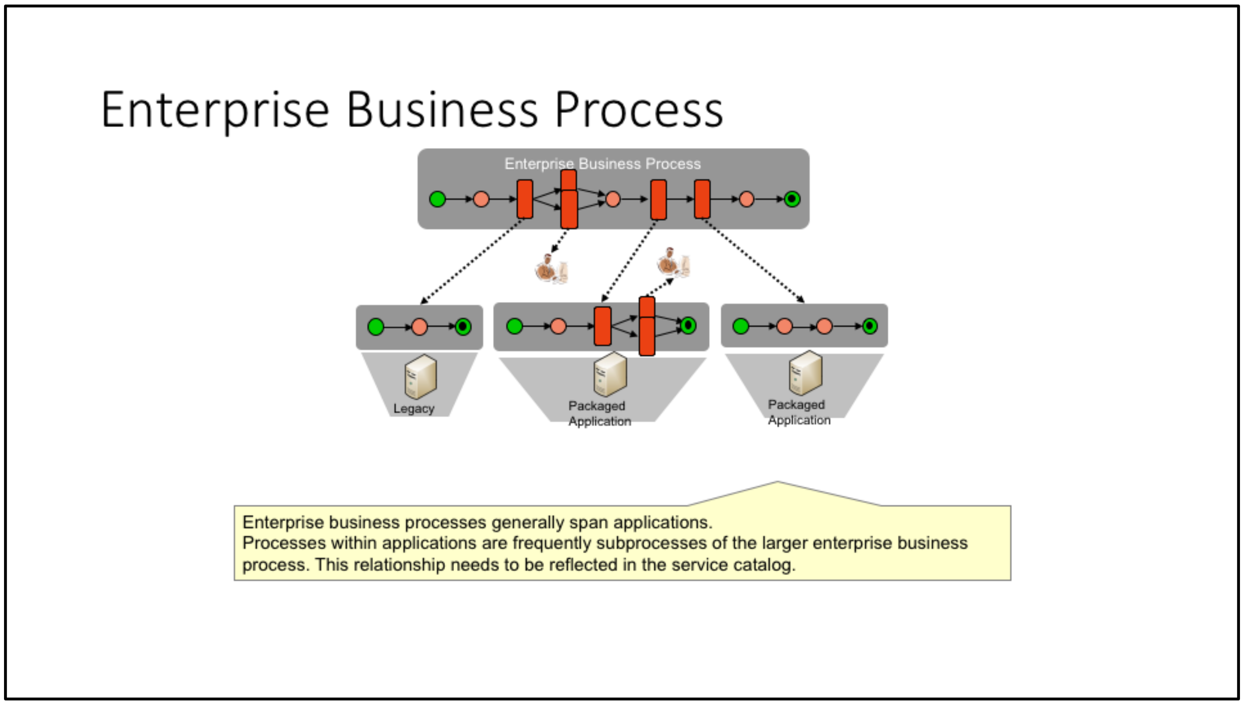
Conceptual View of Service-Oriented Integration

Services provide an intermediary between consumers and source systems:
- Location transparency
- Platform independence
- Implementation encapsulation

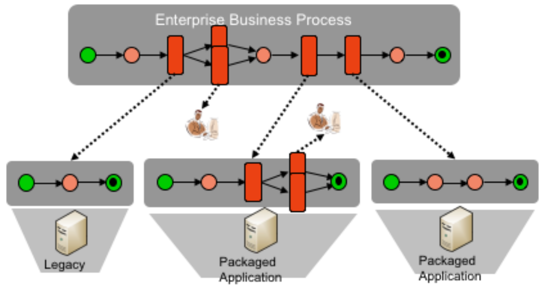
Services provide business-enabling capabilities:
- Enterprise data
- High level of abstraction
- Cross-application

Source systems provide:
- Process
- Functionality
- Data

Creating a SOA service from existing assets generally requires much more than just adding a standards-based interface. Simply service-enabling existing assets is insufficient. The SOA service needs to expose processes, functionality, and data that are usable in a broader context than the source of the capability was designed to meet. Therefore, creating a SOA service usually entails some amount of aggregation, transformation, or expansion of existing capabilities provided by the source systems. This requires a SOA services layer between the existing assets and the consumers (as illustrated in the slide). The essence of service-oriented integration is the building of a catalog of SOA services that expose—in a business-enabling way—the data, functionality, and processes contained in existing systems. To achieve the benefits of service-oriented integration, the SOA services must decouple the consumers from the source systems without creating tight coupling between the consumer and the SOA service itself. Additionally, the SOA services must expose functionality that is aligned with the business needs of the service consumers, rather than merely providing an API or data-level pass through to the source systems.

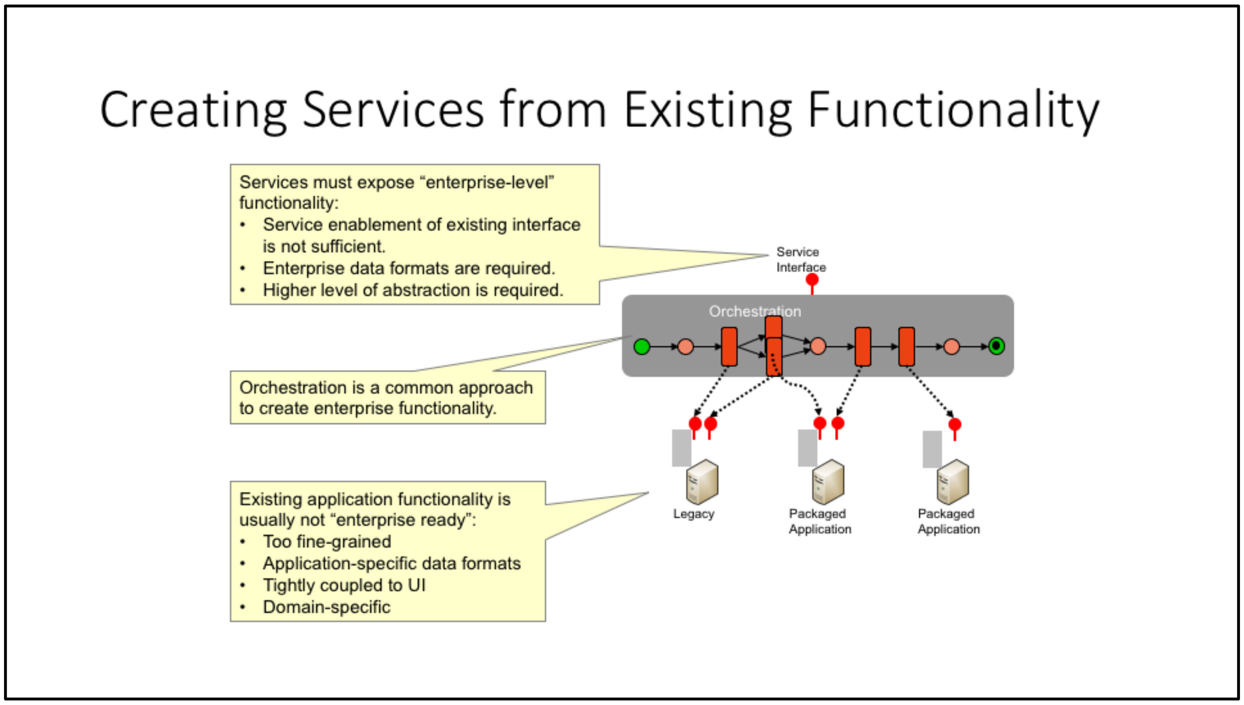


Enterprise Business Process



Enterprise business processes generally span applications.
Processes within applications are frequently subprocesses of the larger enterprise business process. This relationship needs to be reflected in the service catalog.

The existing source systems frequently include a business process or workflow built into the system. From the perspective of the source system, this is a complete business process. However, from the broader context of the organization, the built-in business process is actually only a portion of a larger business process that spans multiple back-end systems. Therefore, when you create the catalog of SOA services, these built-in business processes should be viewed as subprocesses that are part of the larger enterprise business process. This relationship is illustrated in the slide.



Creating Services from Existing Functionality

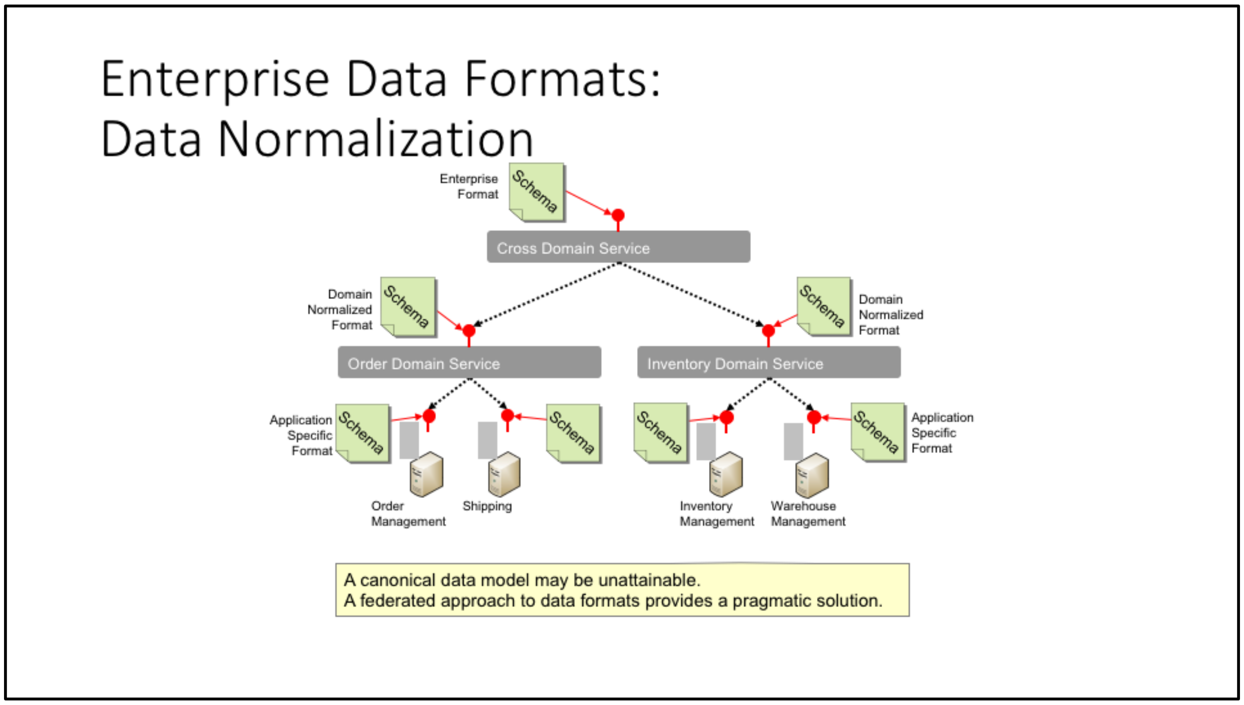
Services must expose “enterprise-level” functionality:
- Service enablement of existing interface is not sufficient.
- Enterprise data formats are required.
- Higher level of abstraction is required.

Orchestration is a common approach to create enterprise functionality.

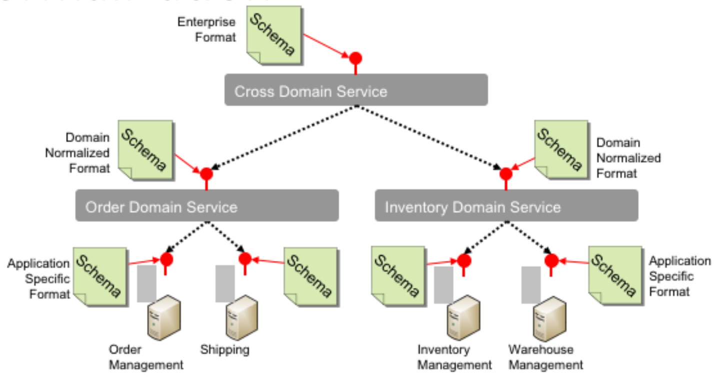
Existing application functionality is usually not “enterprise ready”:
- Too fine-grained
- Application-specific data formats
- Tightly coupled to UI
- Domain-specific

The functional capabilities contained in existing source systems are usually too fine grained and domain specific to be useful, as is, for creating composite applications. The application hosting the functionality was usually constructed to provide users fine-grained control. Additionally, the application invariably contains far more functionality than is necessary in a broader context. Most of the functionality is specific to the domain (for example, HR, Finance, Shipping) and has little relevance in an "enterprise" context.
Therefore, exposing existing functionality as SOA services usually requires abstracting the functionality to a higher level so that it has meaning (and is more useful) in a broader context. Frequently this means combining several fine-grained operations into a more coarse-grained operation.
Sometimes the fine-grained operations are associated with a built-in workflow exposed through the user interface (a set of prescribed operations that the end user performs in sequence). In this instance, when you create a SOA service, it is usually desirable to have a single operation on the SOA service that encapsulates and hides the built-in workflow by performing the individual steps automatically. The orchestration may even span two or more existing applications. This orchestration of fine-grained operations to create a coarser-grained operation is a common technique to abstract functionality. This type

of orchestration is illustrated in the slide.



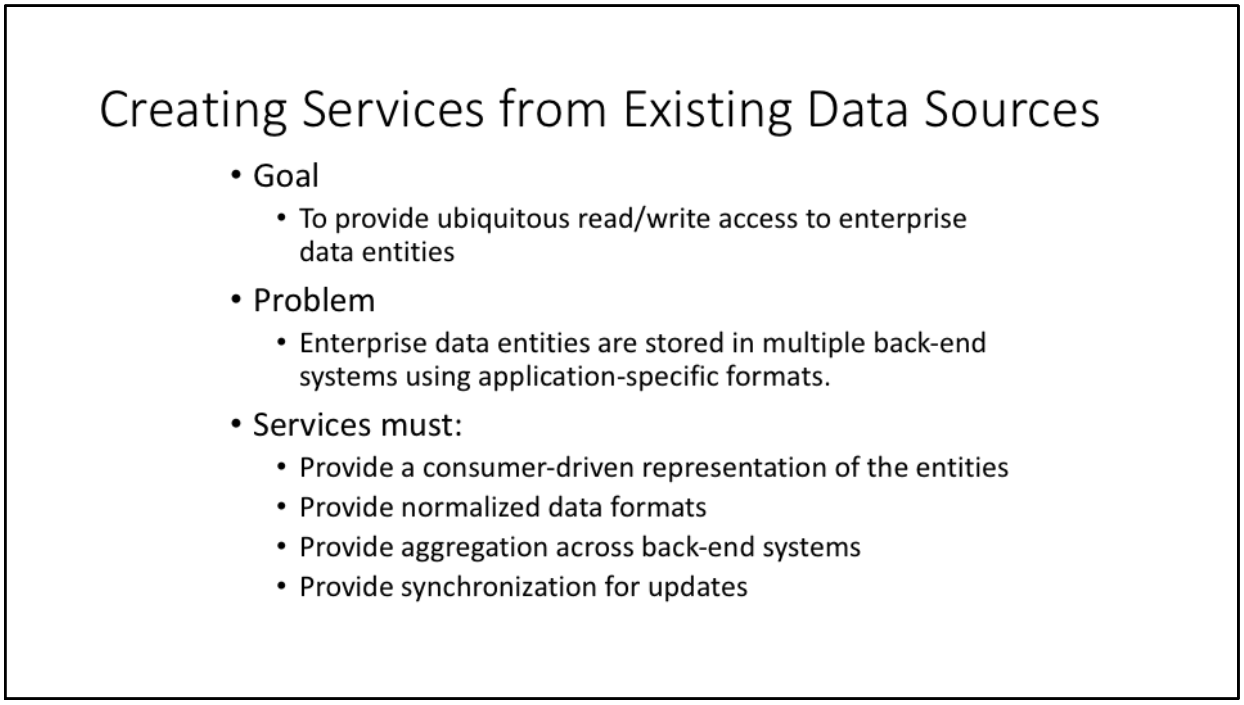
Enterprise Data Formats: Data Normalization



A canonical data model may be unattainable.
A federated approach to data formats provides a pragmatic solution.

Each existing application contains its own data model and data formats. This proliferation of data models and data formats is made worse by the fact that a single enterprise entity (for example, a customer, product, or order) frequently has data elements stored in multiple existing applications. To be successful at exposing existing data via SOA services, the integration approach must manage this complexity.
Providing consumer representations and reading from and writing to multiple source systems lead to the issue of data format transformations. For a very small number of source systems, point-to-point transformations can be used by the SOA services.
However, this approach becomes unworkable as the number of source systems increases.
A better approach is to create a normalized format for the data entities and then provide transformations to and from the normalized format for each source system. A single canonical data model for the entire enterprise is not required to successfully employ normalized data formats. Rather, a federated approach to normalization can be used. For example, in a large enterprise, each functional domain could create a normalized format. Transformations between the domain formats would then be created for SOA services that

span domains. This approach is illustrated in the slide.


Creating Services from Existing Data Sources

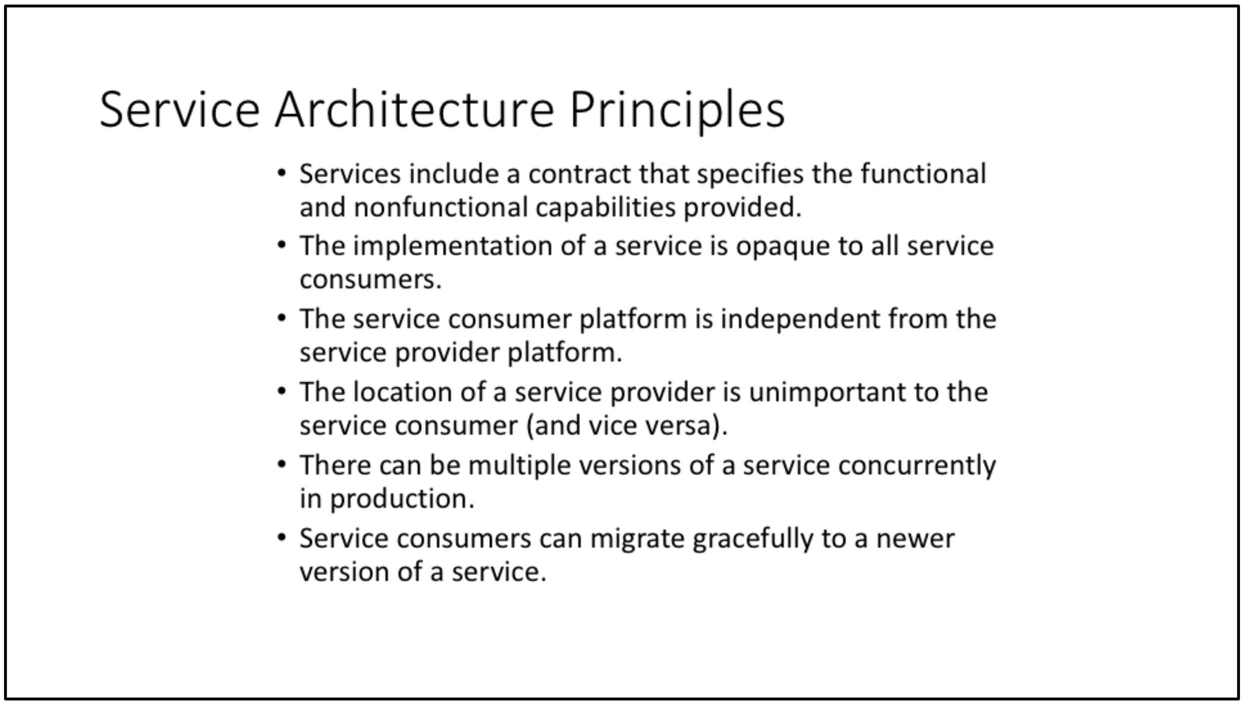
- Goal
  - To provide ubiquitous read/write access to enterprise data entities
- Problem
  - Enterprise data entities are stored in multiple back-end systems using application-specific formats.
- Services must:
  - Provide a consumer-driven representation of the entities
  - Provide normalized data formats
  - Provide aggregation across back-end systems
  - Provide synchronization for updates

The primary goal of a SOA service that exposes an enterprise data entity is to make it easier to work with the entity and hide the complexity of how that entity is stored in existing source systems. To achieve this goal, the SOA service needs to incorporate the following capabilities:

Consumer representation: The SOA service should present an interface to the entity that is appropriate for the consumer of the data. A single SOA service might provide multiple representations of a singe entity, with each representation appropriate for a different type of consumer.

Aggregation: The SOA service may need to pull data elements from multiple source systems to construct a representation for a particular consumer.

Synchronization: An update to an entity may require updating data elements in more than one source system. The SOA service needs to ensure that all updates are done properly so that the entity maintains consistency.



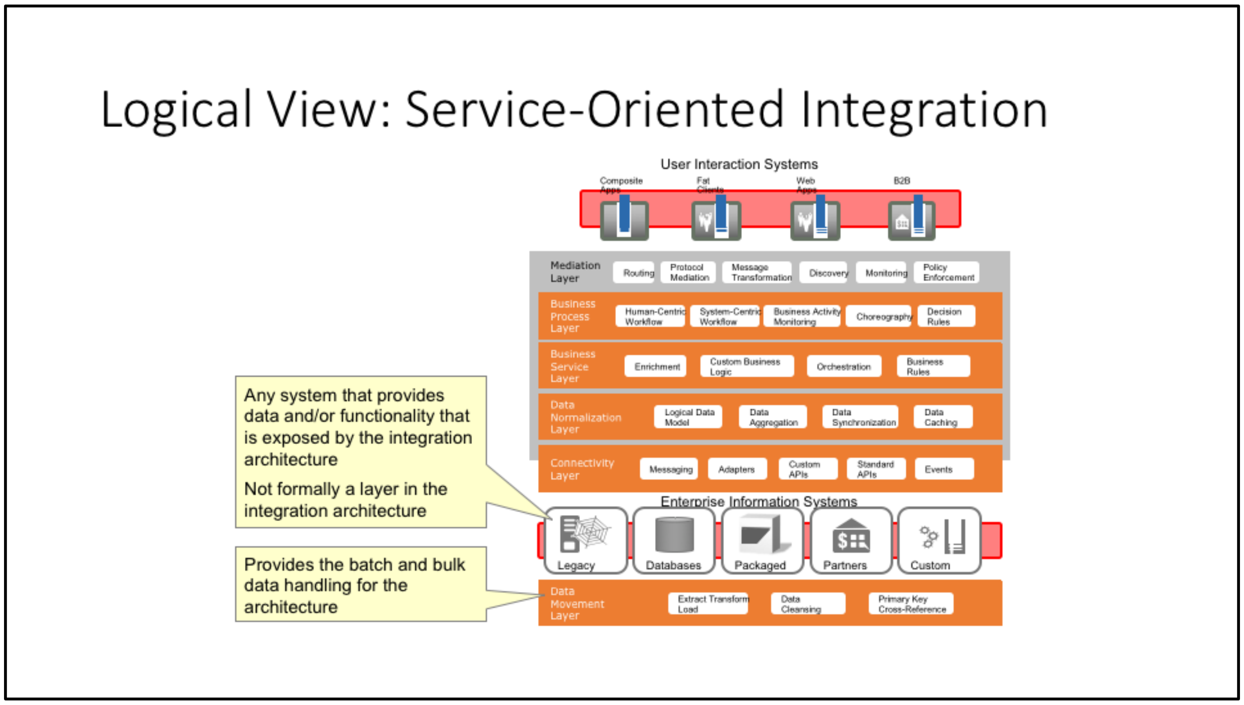
Service Architecture Principles

- Services include a contract that specifies the functional and nonfunctional capabilities provided.
- The implementation of a service is opaque to all service consumers.
- The service consumer platform is independent from the service provider platform.
- The location of a service provider is unimportant to the service consumer (and vice versa).
- There can be multiple versions of a service concurrently in production.
- Service consumers can migrate gracefully to a newer version of a service.

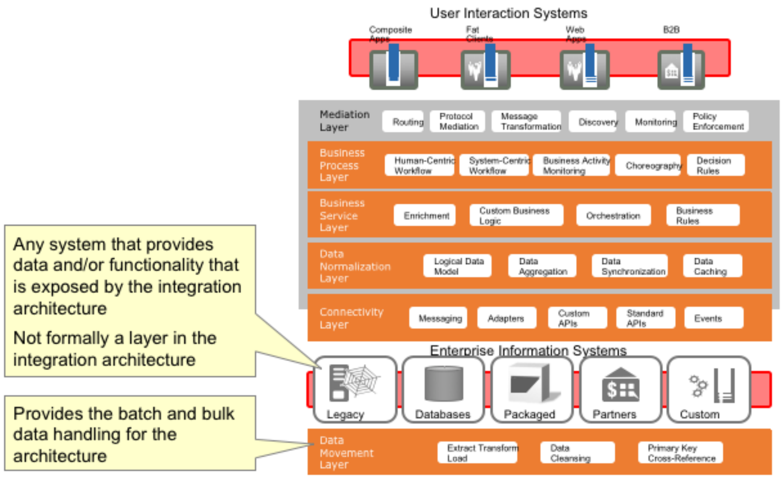
Service Contract
SOA services include a contract that specifies the functional and nonfunctional capabilities provided. To support business-level composition, the SOA service must have a contract that is understandable to a business person. A service engineering process that enforces creation of the service contract must be established. The architecture must support discovery of existing SOA services based on the service contracts. Some of the capabilities provided by the SOA service and documented in the service contract may be fulfilled by configuration of infrastructure.

Opaque Service Implementations
The implementation of a SOA service is opaque to all service consumers. Service consumers must be able to successfully call a SOA service without needing to understand the internal workings of SOA services. The service interface must be constructed so that implementation details do not propagate through the interface. The architecture must support opaque service implementations.

Platform Independence
The service consumer platform is independent of the service provider platform. SOA services need to support any kind of service consumer regardless of the particular platform on which the consumer is built. SOA services must be constructed so that platform dependencies are not introduced. The architecture must support service consumers hosted on platforms that are different from the service providers.
Location Transparency
The location of a service provider is unimportant to the service consumer (and vice versa). Location transparency helps provide the decoupling of service consumers and providers that is necessary for extensive SOA service reuse. Service providers should make no assumption about the location of the service consumer. The architecture must support the ability for a service consumer to call a service provider, regardless of the location of either.
Concurrent Service Versions
There may be multiple versions of a SOA service concurrently in production. A SOA service will almost always require modifications to support new consumers or to expand functionality. Supporting concurrent versions of a SOA service is essential for a sound service-versioning approach. A service-versioning strategy needs to be established. The architecture must support multiple concurrent versions of a SOA service.
Graceful Service Migration
Service consumers should be able to migrate to a newer version of a SOA service gracefully. Service consumers should migrate to a new version of a SOA service as part of a normal maintenance process. The coordinated deployment of service consumers and service providers should not be necessary. A service migration strategy needs to be established. The architecture must support graceful service migration.
Summary
The preceding principles provide sound guidance for creating a service-oriented architecture for integration. Each organization embracing service-oriented integration should evaluate the principles listed in the slide and derive their own set of principles to match the specific environment and goals of their organization.



Logical View: Service-Oriented Integration



Any system that provides data and/or functionality that is exposed by the integration architecture

Not formally a layer in the integration architecture

Provides the batch and bulk data handling for the architecture

The data movement layer provides batch and bulk data handling for the architecture. This layer exists primarily to offload bulk data movement from the upper layers in the architecture. Bulk data movement is inevitable in many enterprises; therefore, the architecture must provide a mechanism to provide this capability in an efficient, controlled manner. Without this layer, the other layers in the architecture might be misused to move large blocks of data—a task for which the other layers are not well suited.

The key capabilities encapsulated by this layer include:

Extract, transform, load: The ETL/ELT capability provides bulk data movement from one persistent store to another.

Data cleansing: Data cleansing ensures the quality of the data that is being moved in bulk. Data cleansing can also be applied to ensure quality and consistency on individual data updates.

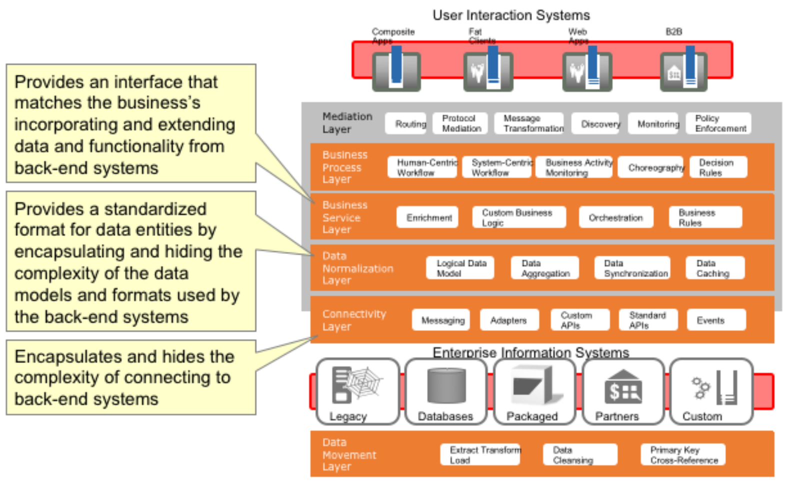
Primary-key cross-reference: Each persistent data store is likely to have its own unique primary key scheme. This layer provides the cross-reference between these primary key schemes so that an identity can be maintained across disparate data stores.

The data movement layer is shown below the enterprise information systems because bulk data movement is done "behind the scenes" and this processing is primarily invisible to the upper layers of the architecture. However, the results achieved by the data movement layer are clearly valuable to the upper layers, and both the quality data and the primary-key cross-reference are leveraged by the upper layers in the architecture wherever appropriate.

In some IT departments, bulk data movement is used to create online aggregate data views by pulling data from multiple systems together into a single data store. Although this architecture does not prohibit such an approach, the preferred approach is to create enterprise data entities in the data normalization layer that aggregate data across source systems without the need to persistently store copies of data.

An enterprise information system (EIS) is any system that provides data or functionality that is exposed by the integration architecture. Examples of EISs include packaged applications, legacy systems, databases, partner systems, and so on. An EIS may also be a service consumer. As SOA becomes more prevalent, it will become more common for EISs to participate natively in service-oriented integration. This is not formally a layer in the integration architecture. Rather, the entire purpose of the architecture is to expose the data and functionality that is contained in the various enterprise information systems to provide business value beyond what is already provided by the systems themselves.

Logical View: Service-Oriented Integration


Provides an interface that matches the business's incorporating and extending data and functionality from back-end systems

Provides a standardized format for data entities by encapsulating and hiding the complexity of the data models and formats used by the back-end systems

Encapsulates and hides the complexity of connecting to back-end systems

The connectivity layer provides access to the enterprise information systems. There are a variety of technologies and products that can be leveraged to connect to existing EISs. Generally, both synchronous and asynchronous techniques are employed by this layer to provide connectivity. Both proprietary and standards-based technologies and products are used.
The key capabilities provided by this layer include:

Messaging: Messaging provides asynchronous connectivity to EISs. Many traditional integration products (for example, MQSeries and TIBCO) are based on messaging systems. Enterprises have successfully used these products to connect to EISs, and this existing connectivity can be leveraged by this architecture.

Adapters: Adapters (for example, JCA) provide a standardized approach for connecting to EISs.

Standard APIs: The EIS may provide access via one or more standard application programming interface (for example, JDBC, RMI, and WS*). The connectivity layer includes these standard interfaces natively, thereby allowing easy access to any EIS that exposes a standard API.

Custom APIs: Some EISs (especially legacy) provide access only via a custom application programming interface. The connectivity layer provides the capability to call these custom APIs and wrap them with a standardized interface.

Events: Events allow the EISs to initiate actions. The connectivity layer provides the mechanism so that an EIS can raise an event that is handled by the mediation layer (just like any other message). The primary purpose of this layer in the architecture is to encapsulate and hide the complexity of connecting to back-end systems. This allows the upper layers in the architecture to treat the EISs in a more uniform and generic fashion, thus providing greater reusability and portability across back-end systems.

The data normalization layer provides a standardized format for data entities. Each EIS stores data in its own (usually proprietary) format. This layer transforms the data into a form that is readily consumable by the upper layers in the architecture. The key capabilities provided by this layer include:

Logical data model: The logical data model provides an "enterprise" view of data entities. The logical data model for an enterprise frequently incorporates industry-standard data formats (for example, HR-XML, HL7, and IATA).

Data aggregation: To construct a complete view of an enterprise data entity, it is frequently necessary to aggregate data from several sources. This layer includes this ability to aggregate data from multiple sources to provide a complete enterprise data entity.

Data synchronization: Data synchronization ensures that all data entities remain consistent regardless of where an update or change is made. Changes are propagated to all copies and views in an orderly manner.

Data caching: Data caching allows the computational expense of aggregation and transformation to be shared across more than one service request. The primary purpose of this layer in the architecture is to encapsulate and hide the complexity of the data models and formats used by the back-end systems. This allows the upper layers in the architecture to operate on data entities that match the needs of the business rather than operating on data that match the storage approach of the back-end systems. The data normalization provided by this layer should not be confused with database normalization.

The business service layer exposes data and functionality that is understandable and has meaning in a business context. This layer uses the lower layers to create SOA services that a business person would understand. The key capabilities provided by this layer include:

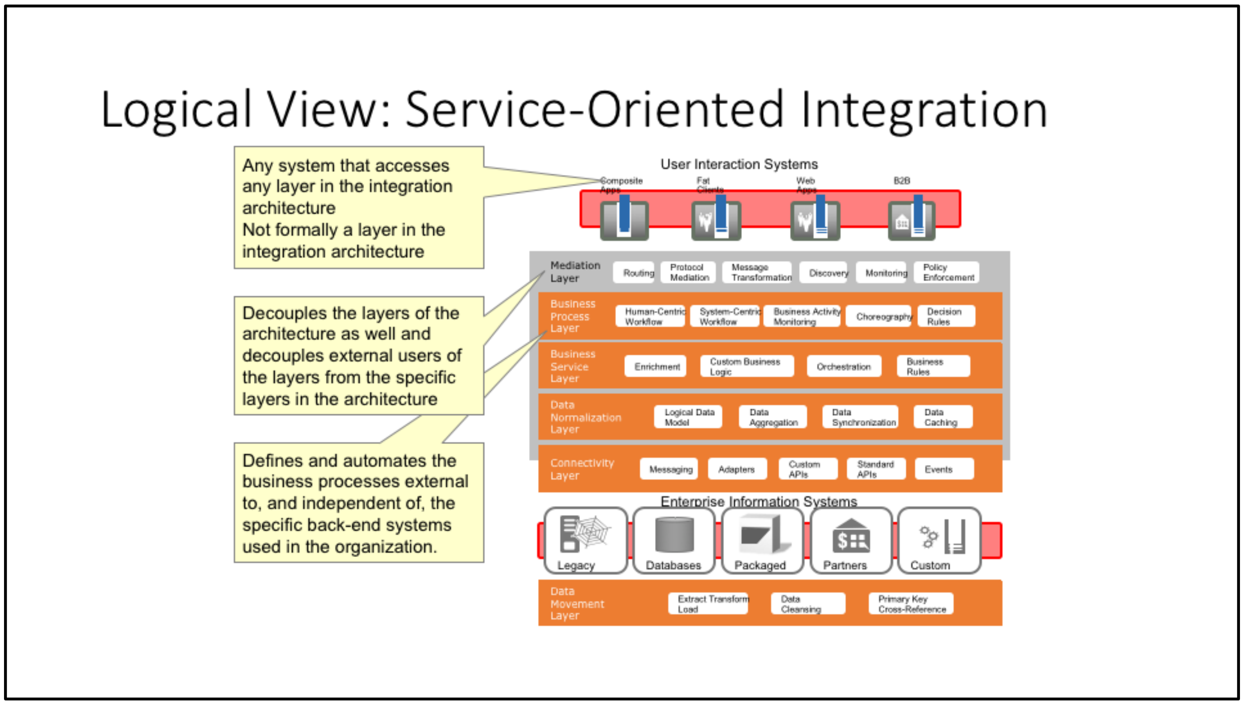
Enrichment: Enrichment is the process of adding data elements to a data entity to give the entity increased information (and hence value) in a business context.

Orchestration: Orchestration is used to combine multiple lower-level operations into a business service that hides the complexity of the lower-level operations.

Custom business logic: Custom business logic is hosted in this layer of the architecture. Custom business logic frequently leverages lower-level layers to provide some of the exposed data and functionality.

Business rules: Business rules describe the operations, definitions, and constraints that apply to an organization. Essentially they are if-then statements that define or constrain some aspect of the business. Although business rules are frequently captured in custom code, it is far better to extract these business rules into explicit rule sets so that they can be more easily understood, modified, and maintained.

The primary purpose of this layer in the architecture is to provide an interface that matches the business. This layer also provides the layer of innovation over the existing back-end systems. Custom business logic and rules can be implemented that use existing systems yet provide capabilities that extend beyond the capabilities provided by the back-end systems.



Logical View: Service-Oriented Integration

Any system that accesses any layer in the integration architecture
Not formally a layer in the integration architecture

Decouples the layers of the architecture as well and decouples external users of the layers from the specific layers in the architecture

Defines and automates the business processes external to, and independent of, the specific back-end systems used in the organization.

The business process layer automates business processes. This layer uses the lower layers, especially the business service layer, to create and automate business processes. The business processes generally span multiple enterprise information systems.
The key capabilities in this layer include:

Human-centric workflow: Human-centric workflow is a business process that is primarily or entirely composed of human tasks. There may be some relatively small amount of system interaction to support the human tasks.

System-centric workflow: System-centric workflow is a business process that is primarily composed of system-to-system activities. There may be some human tasks in the business process, or humans may be required to handle the exception cases in an otherwise entirely system-to-system process.

Choreography: Choreography defines the messages that flow back and forth between systems that are participating in business processes. An example of choreography would be a Rosetta Net Partner Interface Process (PIP).

Business activity monitoring: Business activity monitoring (BAM) provides visibility into business processes. The BAM information is exposed in dashboards that are used to measure and improve business processes..

Decision rules: Most business processes include decision points where branching is done based on certain criteria. Decision rules are the encapsulation of these decision criteria into a rule that is then more easily modified and maintained.

The primary purpose of this layer in the architecture is to define and automate the business processes in a way that is external to—and independent of—the specific back-end systems used in the organization. This isolates the business process from back-end system changes and, conversely, isolates the back-end systems from business process changes. Decoupling the business processes from the back-end systems simplifies changes and maintenance for business processes and back-end systems. This layer generally provides the greatest and most measurable business value.

The mediation layer provides loose coupling for the entire architecture. It decouples the layers of the architecture and also decouple external users of the layers from the specific layers in the architecture. The key capabilities in this layer include:

Routing: Routing provides the ability to send the client request to the appropriate provider based on certain criteria. The routing may even include sending the client request to multiple providers. This capability facilitates location transparency, versioning, scalability, partitioning, request pipelining, SLA management, and so on.

Protocol mediation: Protocol mediation is the ability to handle a client request that uses one protocol (for example, WS*, JMS, or REST) with a provider that uses a different protocol. This provides protocol decoupling between the provider and the consumer.

Message transformation: Message transformation allows a client request that uses one message format to be handled by a provider that expects a different message format. This provides message format decoupling between the provider and the consumer.

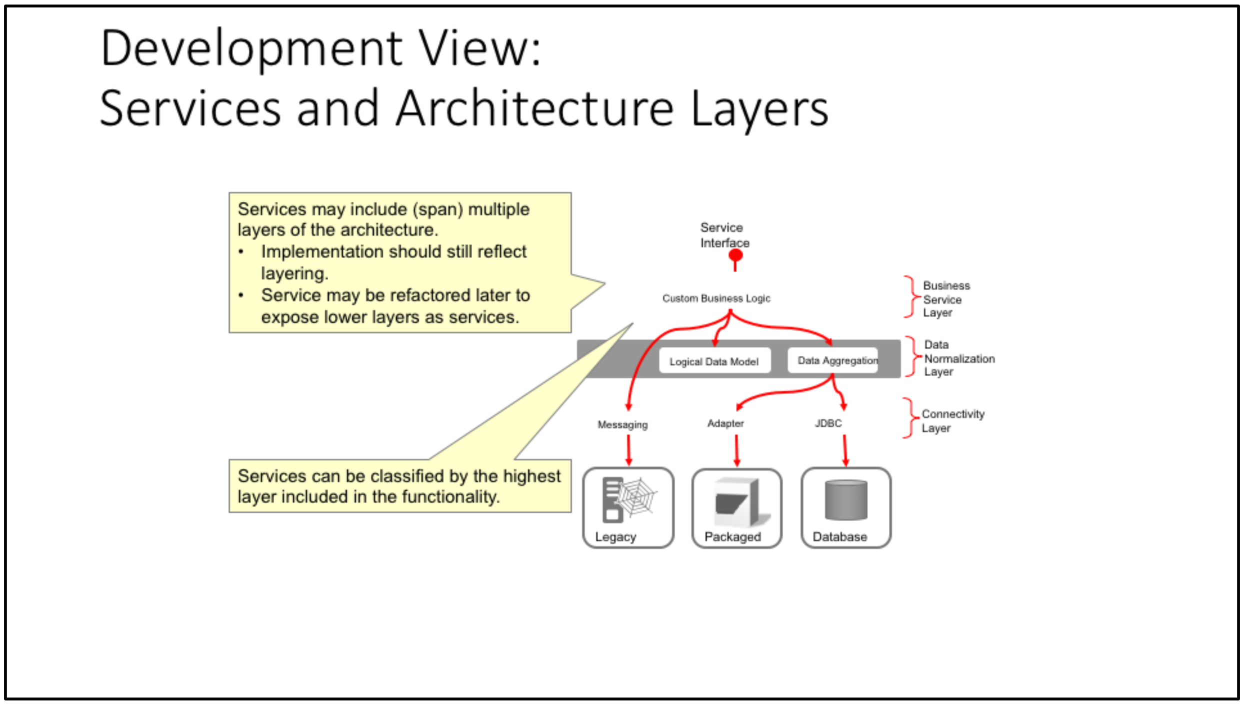
Discovery: Discovery is the mechanism by which a client finds a provider of a particular SOA service. Discovery can occur at design time or runtime.

Monitoring: Monitoring captures runtime information about the messages flowing through the mediation layer. Because the mediation layer is an intermediary for message traffic, it provides a centralized monitoring capability.

Policy enforcement: Policy enforcement provides consistent application of policies (for example, WS-SecurityPolicy) across all messages flowing through the mediation layer. Because the mediation layer is an intermediary for message traffic, it provides a centralized policy enforcement capability.

The primary purpose of this layer in the architecture is to facilitate communication between layers in the architecture and between this architecture and the systems that connect to it. This layer is “infrastructure” in the truest sense and therefore rarely maps directly to business requirements. However, this layer provides key capabilities that make the architecture service oriented and is the primary focus for

meeting nonfunctional requirements such as scalability, reliability, availability, and maintainability.
A User Interaction System is any system that accesses any layer in the integration architecture. There are a wide variety of systems that can access the integration layers. Examples include web-based applications, fat-client applications, and partner applications. This is not formally a layer in the integration architecture. Rather, these are the systems that gain value from the data and functionality exposed by the layers of the architecture

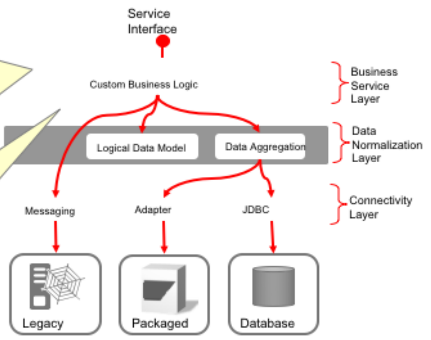


Development View: Services and Architecture Layers

Services may include (span) multiple layers of the architecture.

- Implementation should still reflect layering.
- Service may be refactored later to expose lower layers as services.

Services can be classified by the highest layer included in the functionality.



A SOA service can be exposed at any layer in the logical architecture. A SOA service implementation can also span layers in the integration.

When constructing a SOA service that spans layers, developers should understand and apply the layers so that the implementation uses the same separation of concerns. This not only makes the SOA service easier to maintain, it also makes it possible to easily refactor the SOA service to expose lower-level functionality as a separate and distinct service.

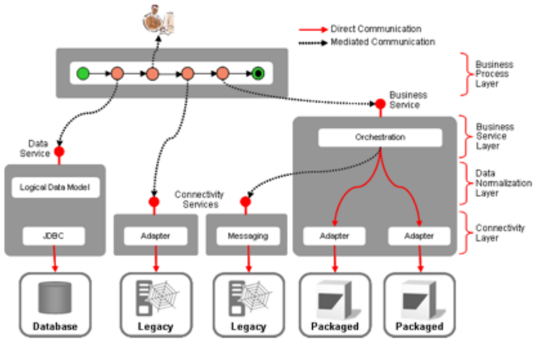
SOA services can be classified by the uppermost layer that is included in the service functionality. The diagram in the slide illustrates several types of SOA services and also illustrates how upper layers call services from lower layers.

Dotted lines indicate communication paths through the mediation layer, whereas solid lines indicate direct communication that is not mediated. Communication that is internal to a SOA service implementation does not use the mediation layer. All communication that uses the interface to a SOA service should pass through the mediation layer.

The diagram shows the business process directly accessing a connectivity service. While the architecture allows this direct interaction, in general this should be avoided since it tends to make the business process directly dependent on the actual back-end system being called. A better approach is to keep the business process isolated from the back-end systems by having the business process call only business services or data services.

Service Composition

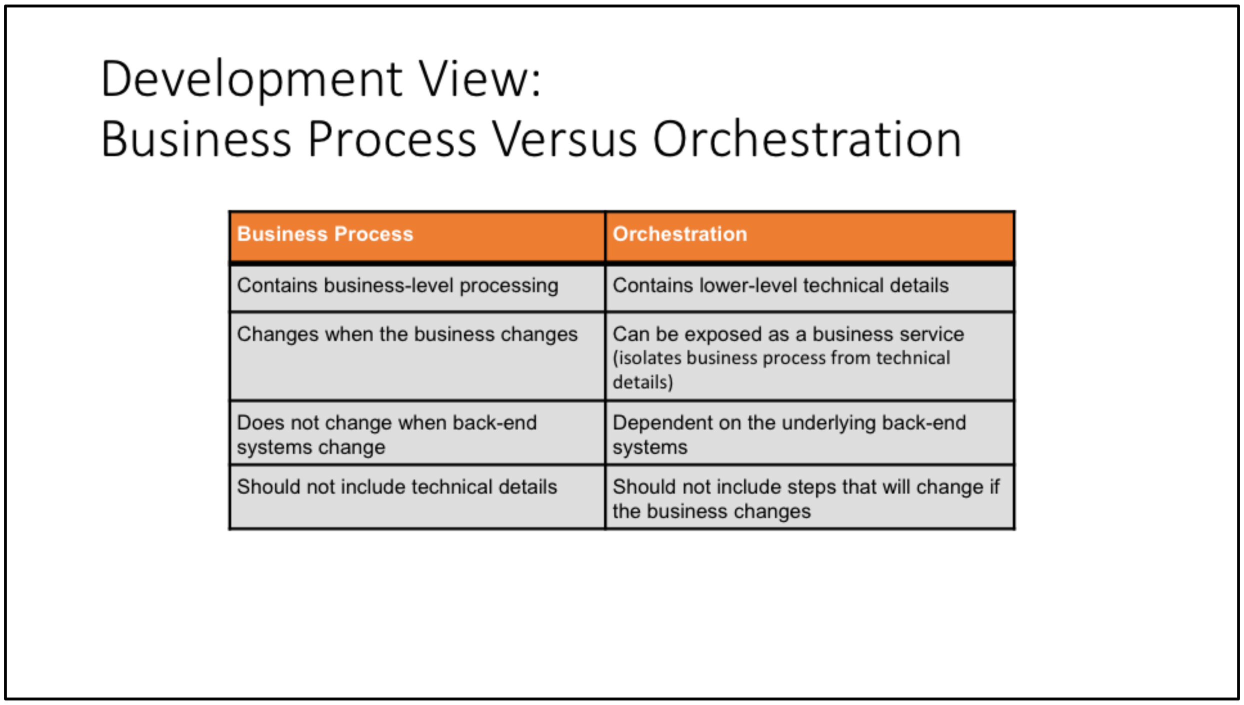
Service composition is supported and encouraged.
Upper-layer services call lower-layer services through the mediation layer.



Business processes using connectivity services should be avoided:
- This exposes application details.
- Exceptions include standard interface (for example, messaging and OSS/J).

Service composition is the ability to leverage lower-level services to create a higher-level service. The orchestration shown in the slide is an example of service composition because the business service leverages the connectivity service to supply some of the necessary capability. This architecture supports and encourages service composition as a primary approach for developers to apply.
When doing composition, developers should respect the layering of the architecture. A business service could leverage existing connectivity services or data services, and a data service could leverage existing connectivity services. But a data service should not call a business service, and a connectivity service should not call a data service or a business service. Composition by calling services within the same layer of the architecture (for example, a business service calling another business service) is supported, but care must be taken when using composition within a layer because this can lead to complex dependencies that present problems for maintenance and versioning.
Depending on the products and technology used to implement the mediation layer, it may also be possible to construct service compositions in the mediation layer. This is commonly referred to as a *service pipeline*. In a service pipeline, the mediation layer handles a service request by routing the request to one or more services potentially doing message transformations and protocol translations as well. Although it is powerful, this capability should be used judiciously since the mediation layer is not a general-purpose programming environment and is usually a shared resource. Excessive computing or configuration errors within the mediation layer could

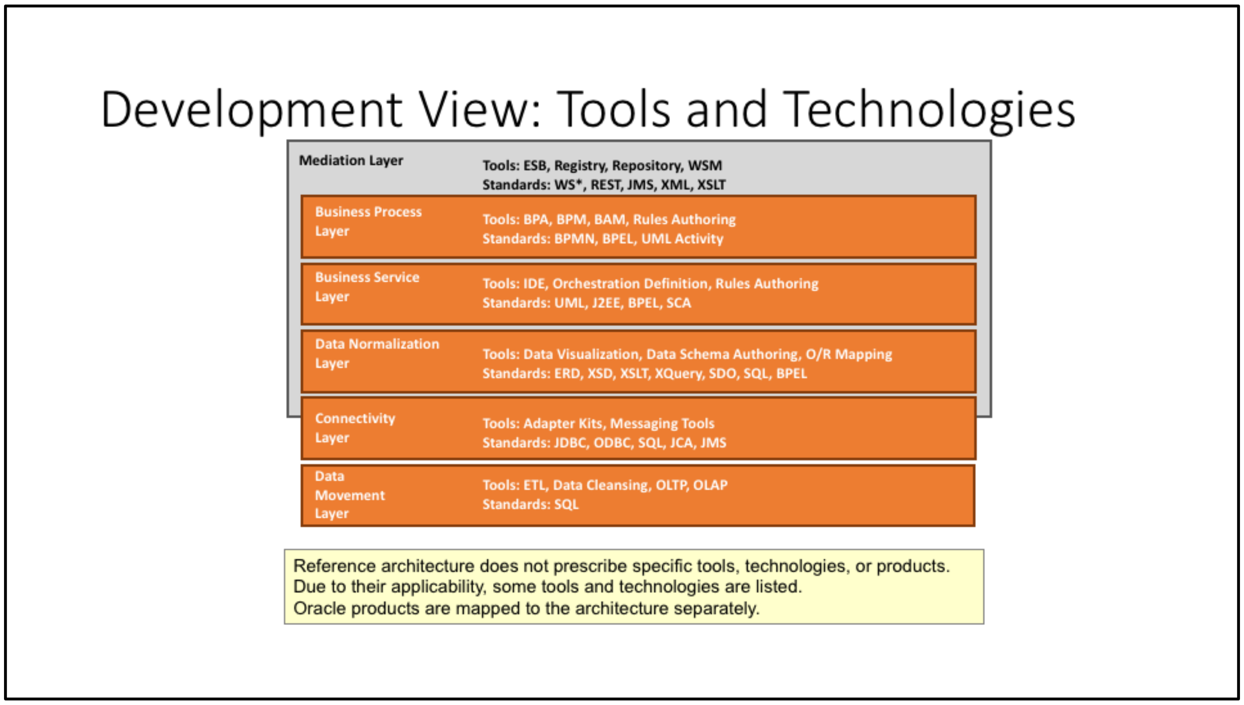
negatively affect all service traffic passing through the mediation layer.



Development View: Business Process Versus Orchestration

Business Process	Orchestration
Contains business-level processing	Contains lower-level technical details
Changes when the business changes	Can be exposed as a business service (isolates business process from technical details)
Does not change when back-end systems change	Dependent on the underlying back-end systems
Should not include technical details	Should not include steps that will change if the business changes

The table in the slide illustrates the differences between a business process and an orchestration. Business processes should not contain lower-level technical details. The technical details should be encapsulated by business services that expose an interface that is meaningful to a business user.
For example, the business process may have a step that requires updating a customer address. However, the customer address may be stored in several back-end systems. This complexity is not of interest to a business user, nor is it a core part of the business process. Therefore, this technical complexity should be encapsulated by a shared business service that exposes an "update customer address" operation. The operation then updates all necessary systems. Notice that this encapsulation also makes it possible to change the back-end systems or even remove back-end systems without affecting business processes that update customer addresses. Only the shared business service needs to be modified. Conversely, the orchestration that is included in the business services should not include steps that are likely to change due to changes in business strategy or execution. Those steps belong in the business process layer so that they can be easily changed whenever the business changes.



Development View: Tools and Technologies

Mediation Layer	Tools: ESB, Registry, Repository, WSM Standards: WS*, REST, JMS, XML, XSLT
Business Process Layer	Tools: BPA, BPM, BAM, Rules Authoring Standards: BPMN, BPEL, UML Activity
Business Service Layer	Tools: IDE, Orchestration Definition, Rules Authoring Standards: UML, J2EE, BPEL, SCA
Data Normalization Layer	Tools: Data Visualization, Data Schema Authoring, O/R Mapping Standards: ERD, XSD, XSLT, XQuery, SDO, SQL, BPEL
Connectivity Layer	Tools: Adapter Kits, Messaging Tools Standards: JDBC, ODBC, SQL, JCA, JMS
Data Movement Layer	Tools: ETL, Data Cleansing, OLTP, OLAP Standards: SQL

Reference architecture does not prescribe specific tools, technologies, or products.
Due to their applicability, some tools and technologies are listed.
Oracle products are mapped to the architecture separately.

This service-oriented integration architecture does not require the use of specific tools or technologies. It is based on principles that were presented earlier, and it can be realized using a wide variety of tools and technologies. Nonetheless, there are types of tools and technologies that are well suited to service-oriented integration, fit nicely with the architecture, and therefore are likely to be used by developers applying this architecture. These tools and technologies are shown in the slide and described for each layer in the architecture.

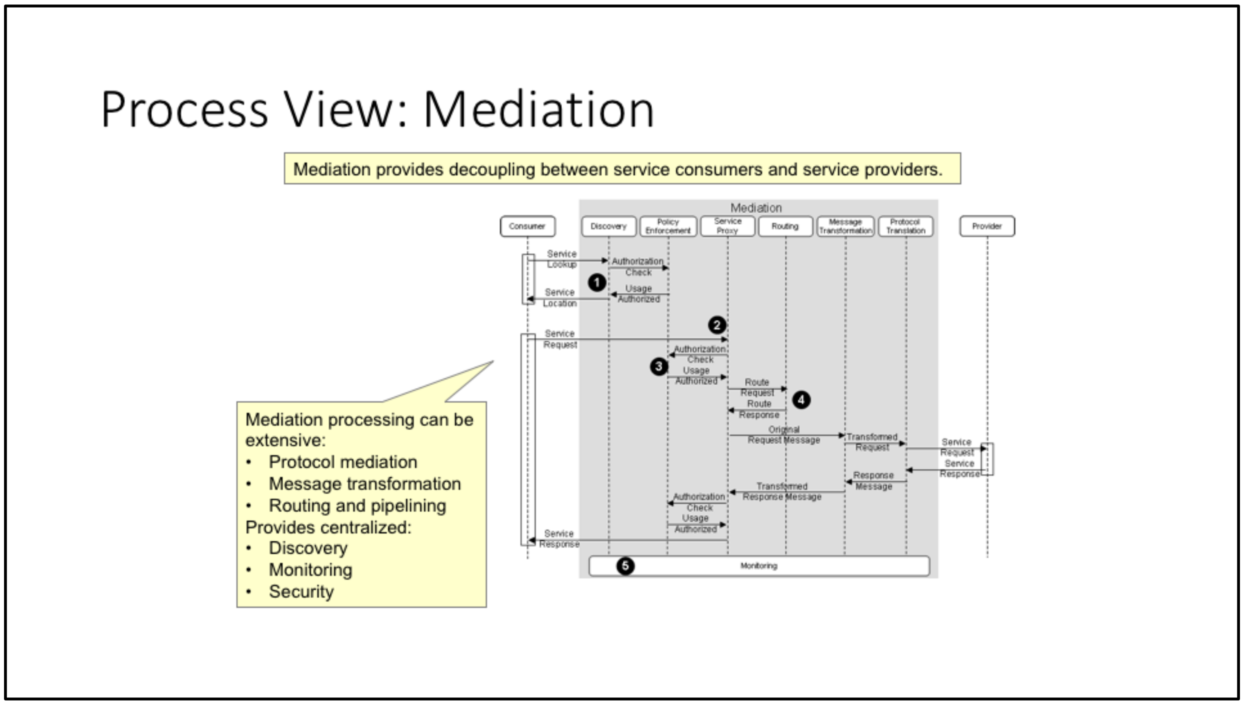
Mediation Layer

The primary tools and technologies for this layer are Enterprise Service Bus (ESB), Registry, Repository, and Web Service Management (WSM). There are also many existing and emerging protocol standards, including WS*, REST,JMS, XML, and XSLT.

Business Process Layer

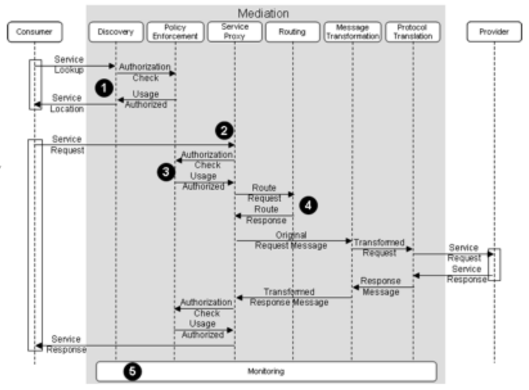
The primary tools and technologies for this layer are for capturing, analyzing, executing, and monitoring business processes. These include business process analysis (BPA), business process management (BPM), and business activity monitoring (BAM). Rules engines can also be used to capture and execute decision rules for business processes. There are several associated standards, including BPMN, BPEL, and UML Activity Diagrams.

Business Service Layer
The primary tools and technologies for this layer include traditional software development tools (for example, IDE). Additionally, the orchestration capability requires a developer-focused process definition tool. Rules engines can also be used to capture and execute business rules. Associated standards include UML, J2EE, BPEL, SCA, WSCDL, and so on.
Data Normalization Layer
The primary tools and technologies for this layer are for modeling data and for defining and manipulating data formats. Thus, the primary tools are visualization and authoring tools for ERDs, O/R mapping, XML, XSD, XQuery, XSLT, and so on. Industry-standard data formats are also important for this layer. Because this layer also performs data aggregation and data synchronization, orchestration tools (for example, BPEL) are also essential for this layer.
Connectivity Layer
The primary tools and technologies for this layer are adapter development kits and messaging technologies. Also, there are many possible technologies that may be needed to connect to the wide variety of systems that may be included as source systems. Associated standards include JDBC, ODBC, and JCA.
Data Movement Layer
The primary tools and technologies for this layer are ETL tools and data cleansing tools. Persistent storage tools and technologies for both OLTP and OLAP are vitally important for this layer.



Process View: Mediation

Mediation provides decoupling between service consumers and service providers.



Mediation processing can be extensive:
- Protocol mediation
- Message transformation
- Routing and pipelining

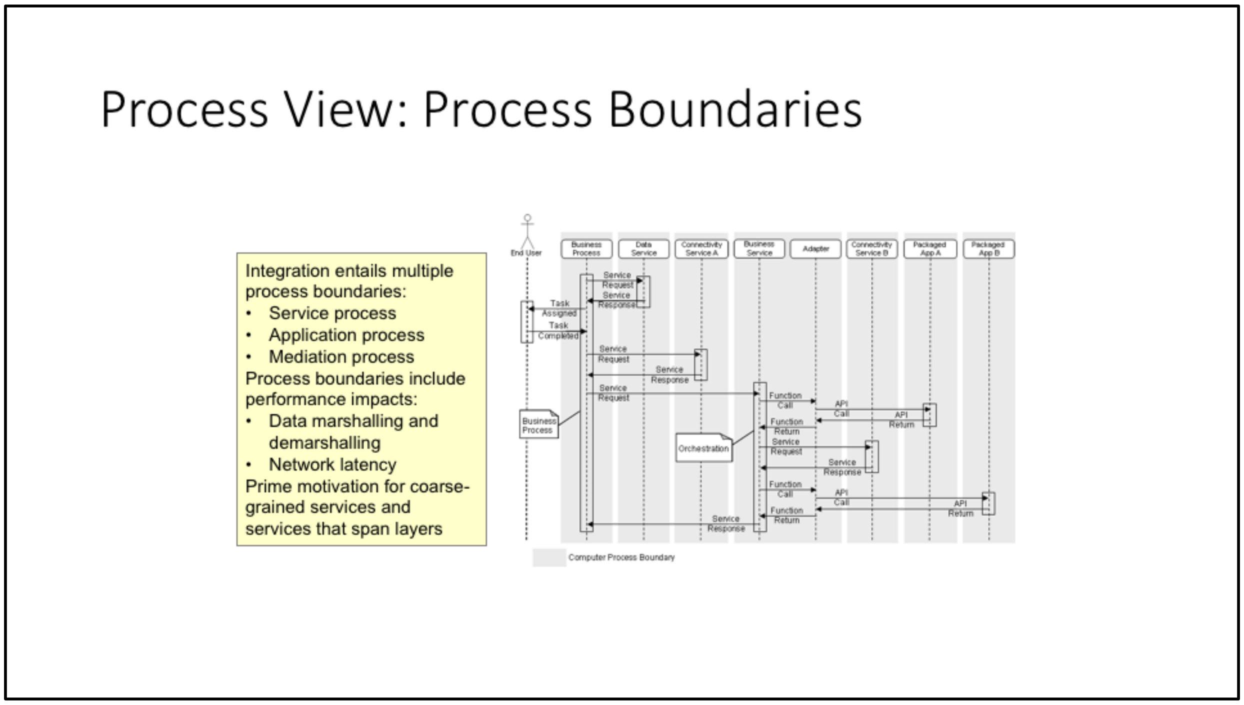
Provides centralized:
- Discovery
- Monitoring
- Security

Mediation is a key component in the overall architecture, providing the decoupling between consumers and providers. The diagram in the slide illustrates the typical flow of a message through the mediation layer.

1. Before calling a SOA service, the service consumer must first find the appropriate SOA services. This is called *service discovery* and may be performed at development time or at runtime. The architecture supports both. There is also an authorization check done to see which SOA services the consumer is allowed to see.
2. The service request is made to a service proxy rather than directly to the service provider. The service proxy allows the mediation layer to apply all the capabilities that have been configured for that service request.
3. An authorization check is performed on the service request. Although it is not always required, leveraging the authorization capability within the mediation layer provides a centralized approach to securing SOA services. An authorization check may also be performed before forwarding the response message to ensure that the consumer is allowed to see the contents within the response message.
4. After the service request has been authorized, the

request is routed to the appropriate service provider. The request message may be transformed before being sent to the provider; the protocol may be translated as well.

5. All of the activities within the mediation layer are monitored. This provides a centralized monitoring capability across all SOA services.

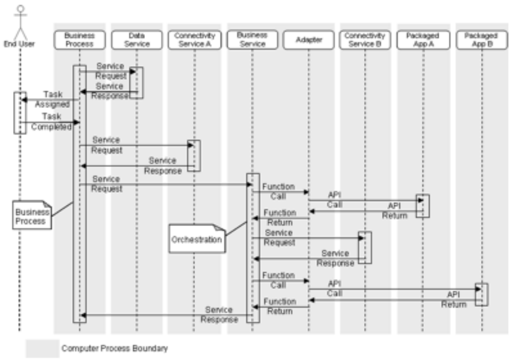


Process View: Process Boundaries

Integration entails multiple process boundaries:
- Service process
- Application process
- Mediation process

Process boundaries include performance impacts:
- Data marshalling and demarshalling
- Network latency

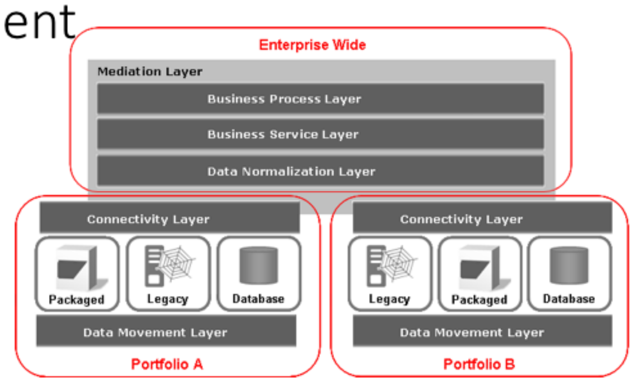
Prime motivation for coarse-grained services and services that span layers



The service-oriented integration architecture spans multiple systems and the computer processes within those systems. The diagram in the slide illustrates the message flow and the process boundaries that are crossed by the messages.
A single business process may result in messages that span many computer processes. In general, every service invocation results in a process boundary being crossed. In fact, since each service invocation also passes through the mediation layer, each service invocation crosses two process boundaries: mediation and service provider. Additionally, the service implementation may also cross service boundaries. For example, the business service shown in the slide makes API calls to packaged applications (via adapters), and each packaged application runs in its own process.
Although seven different processes are shown, the diagram actually simplifies the number of processes included in the integration. Both of the connectivity services (A and B) would likely include out-of-process calls to the source system for which they are providing connectivity.
Each time a process boundary is crossed, there are performance impacts from the network and message marshalling and de-marshalling. This is a primary reason that SOA services should expose relatively coarse-grained interfaces.

This is also a reason that a service implementation might span multiple layers in the architecture. This also explains why the architecture includes a separate data movement layer, because moving large quantities of data across service boundaries is computationally very expensive.

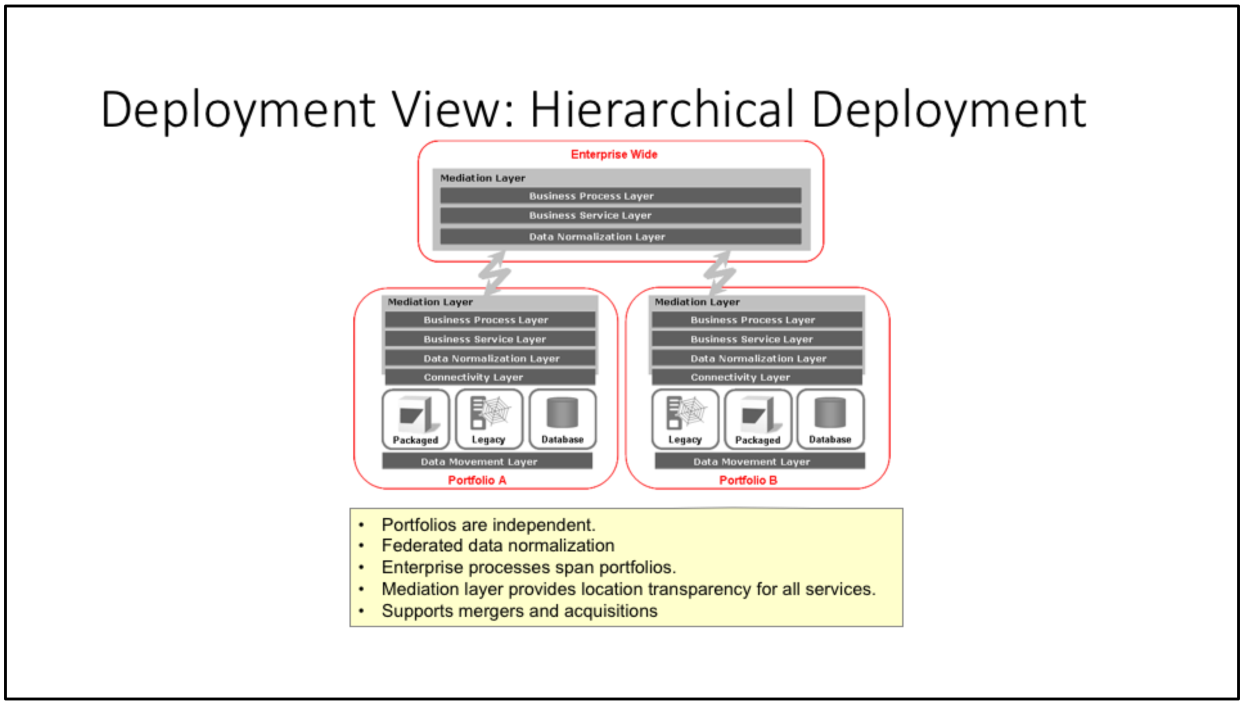
Deployment View: Mostly Shared Deployment



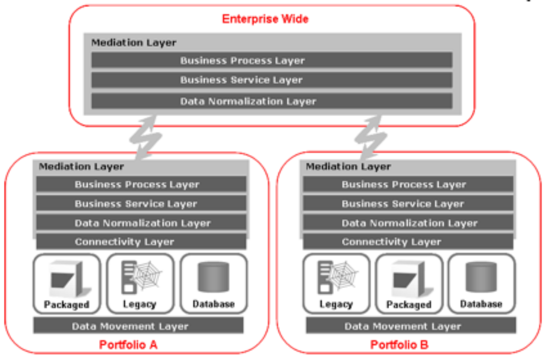
- Upper layers are shared across the enterprise.
- Lower layers are deployed independently to portfolios.
- Small IT environments may share all layers.

There are many ways that the architecture can be deployed in an enterprise. Organizational size, structure, and span of management control play a significant role in determining many aspects of deployment, especially how much is shared across the enterprise and how much is specific to a portfolio or division.

In a mostly shared deployment, only the lower levels of the architecture are deployed independently across portfolios or divisions. This type of deployment is illustrated in the slide. This type of deployment has most of the layers deployed so that they are shared across the enterprise. Only the lowest levels of the architecture are specific to the portfolios. It should also be clear that it is possible for a smaller enterprise to have all of the integration layers shared enterprise-wide.



Deployment View: Hierarchical Deployment



- Portfolios are independent.
- Federated data normalization
- Enterprise processes span portfolios.
- Mediation layer provides location transparency for all services.
- Supports mergers and acquisitions

As the enterprise size increases, it becomes more difficult to deploy a single shared environment for integration. It becomes more likely that a distributed or hierarchical deployment will be a better fit (as illustrated in the slide). This type of hierarchical deployment has several advantages, including the following:

Each portfolio can define its own business processes, SOA services, and data formats that are independent from other portfolios.

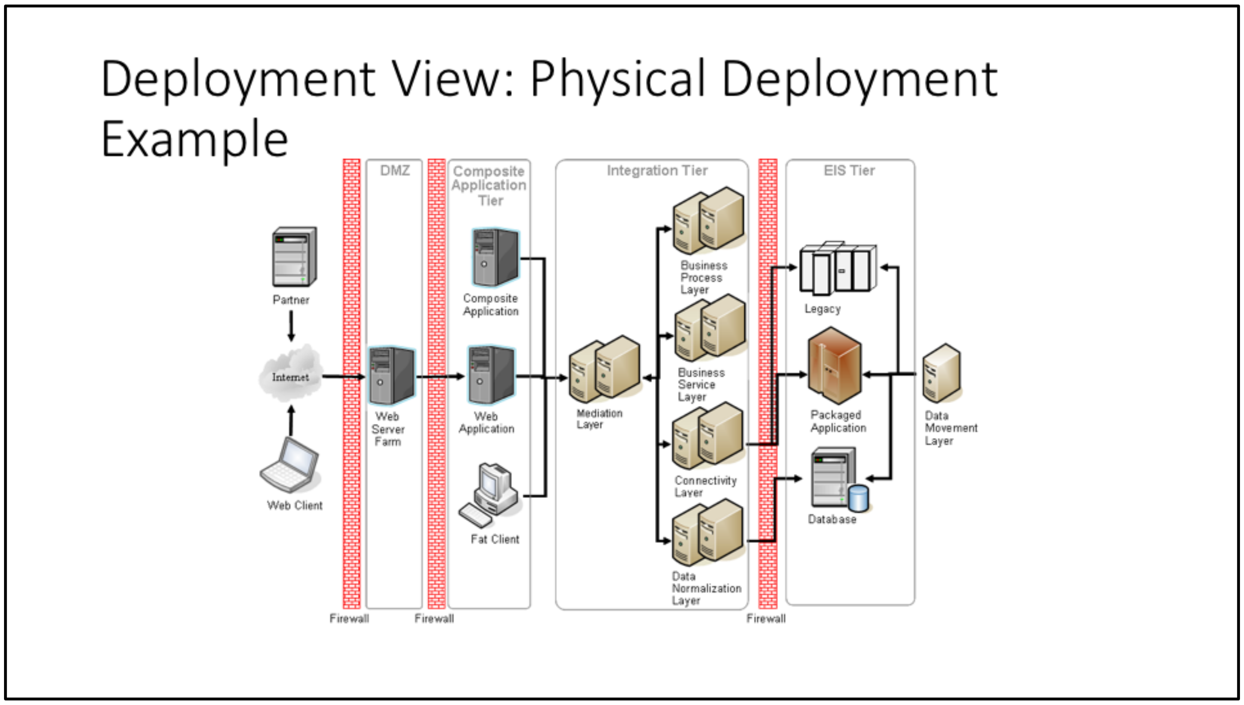
This deployment supports the federated data normalization approach (described earlier).

Business processes or SOA services that span portfolios exist at the enterprise level and can leverage the portfolio business processes and SOA services.

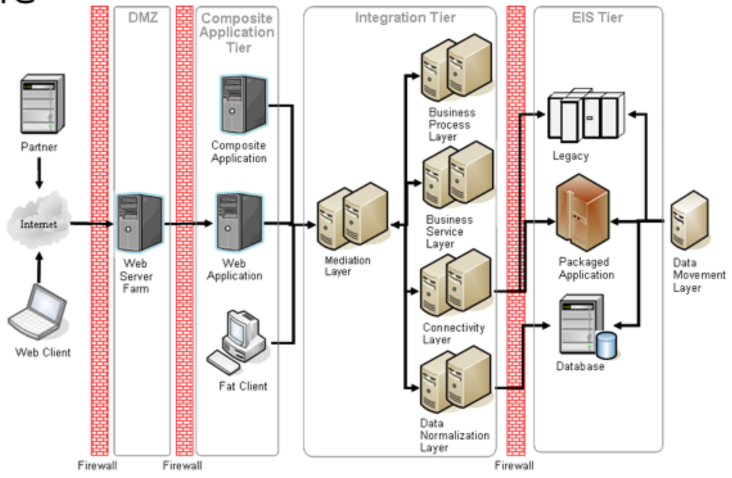
The mediation layer provides location transparency between the portfolio and enterprise domains.

This type of deployment easily supports acquisitions because a new acquisition can be treated as simply another portfolio.

The primary disadvantage of a hierarchical deployment is the increased cost and complexity of supporting more hardware and software. Adherence to standards to support interoperability is also more important in a hierarchical deployment because the various portfolios may select different products.



Deployment View: Physical Deployment Example



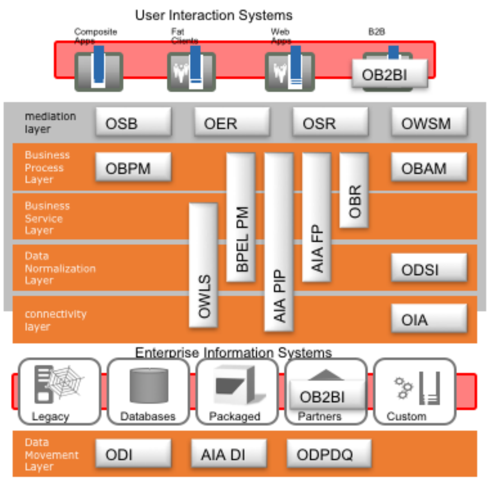
The slide shows a high-level view of how the architecture might be deployed to hardware.
The diagram shows each layer deployed to separate hardware. Although this deployment is possible, it may also be desirable to deploy layers of the architecture to the same hardware. Showing the layers deployed to separate hardware makes the communication paths more explicit. This also shows the layers deployed to multiple hardware boxes to provide scalability and reliability. In a modern data center where server virtualization is widespread, the layers would be deployed to virtual servers.
The number of virtual servers hosting each layer is determined by the load on the layer and can be dynamically provisioned as necessary. Many of the detailed hardware deployment decisions are driven by the specific products and technologies used to realize the architecture.

Agenda

- Service-Oriented Integration
- Reference Architecture
- Product Mapping

Product Mapping

AIA: Application Integration Architecture
AIA DI: Direct Integration
AIA FP: Foundation Pack
AIA PIP: Process Integration Pack
BPELPM: Oracle BPEL Process Manager
OB2BI: Oracle B2B Integration
OBAM: Oracle Business Activity Monitoring
OBPM: Oracle Business Process Manager
OBR: Oracle Business Rules
ODI: Oracle Data Integrator
ODPDQ: Oracle Data Profiling and Data Quality
ODSI: Oracle Data Service Integrator
OER: Oracle Enterprise Repository
OIA: Oracle Integration Adapters
OSB: Oracle Service Bus
OSR: Oracle Service Registry
OWLS: Oracle WebLogic Suite
OWSM: Oracle Web Service Manager



The following Oracle Fusion Middleware products are included in the product mapping:

Oracle BPEL Process Manager (BPEL-PM): Provides the ability to quickly build and deploy orchestrations and business processes in a standards-based manner

Oracle Business Activity Monitoring (OBAM): A complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes and services

Oracle Business Process Management (OBPM): A complete set of tools enabling collaboration between business and IT to create, automate, execute, and optimize business processes

Oracle Business Rules (OBR): Evaluate rules rapidly using a lightweight, high-performance rules engine

Oracle Business-to-Business Integration (OB2BI): Provides a single integrated solution for rapidly establishing online collaborations and automated processes with business partners

Oracle Data Integrator (ODI): A next-generation extract, load, and transform (ELT) technology that offers the productivity of a declarative design approach, as well as the benefits of a platform for seamless batch and real-time integration