SOA design patterns and principles
Unit objectives

After completing this unit, you should be able to:

● Describe various design principles that can be applied during service design within SOA
● Define SOI and describe patterns and principles that can be applied towards developing SOI architecture
● Define ESB and explain its role in SOI
● Describe IBM’s patterns for e-business and the process for applying these patterns
● Describe SOA runtime patterns such as ESB, ESB Gateway, Business Service Choreography, Business Service Directory
● Explain the SOA Application patterns such as the Direct Connection, Broker, Serial Process, and Parallel Process, plus their variations and their realization using runtime patterns
Role of patterns in SOA Foundation

Governance and Process
- SOA Center of Excellence
- Rational Unified Process (RUP)
- IT Infrastructure Library (ITIL)

Best Practices
- SOA-Related IP
  - Patterns
  - Redbooks
- Engagement Experience

Education
- Introduction to Value and Governance Model of SOA
- Web services for managers
- Technologies and Standards for SOA Project Implementation
- Design SOA Solutions and Apply Governance
Service design principles

- Re-use
- Encapsulation
- Well-defined interfaces
  - High cohesion
  - Meaningful names for operations (semantic operations)
  - Operational granularity
- Statelessness
- Efficient resource utilization
Re-use

● Concept
  – A service interface should be designed with reuse in mind

● Consequences
  – Well factored service interfaces:
    • Anticipate usage scenarios, and consequently facilitate reuse
  – Poorly factored service interfaces:
    • Hinder reuse and encourage functional duplication, which can result in architectural decay (loss of architectural integrity over time)
Encapsulation

● Concept
  - Services should not physically expose any implementation details or deployment details at their interface design.

● Consequences:
  - Well encapsulated services:
    • Benefit adaptability by decoupling the service implementation characteristics and service deployment characteristics from the client implementation.
    • In circumstances where an implementation specific or a deployment specific characteristic needs to be changed, then the client remains unaffected.
  - Poorly encapsulated services:
    • Hinder adaptability as a consequence of coupling the client with the service implementation characteristics or service deployment characteristics.
    • In circumstances where an implementation specific or a deployment specific characteristic needs to be changed, then both the client implementation and the service implementation need to be changed.
Well-defined service interfaces

- **High cohesion - Concept**
  - Services interfaces should be concise, related, and complete sets of operations.
- **Consequences:**
  - **Cohesive services:**
    • Provide every operation required by and appropriate for the consumer.
  - **Un-cohesive services:**
    • Hinder consumability – cause confusion by exposing tenuously related, or unrelated, operations to the consumer.

- **Operational parameter granularity - Concept**
  - Service operation parameters should be coarse grained, and allied to the operation semantics.
    • As coarse-grained as possible whilst maintaining semantic integrity.
- **Consequences:**
  - **Coarse-grained parameter lists:**
    • Benefit consumability and comprehension with concise and semantic signatures.
  - **Fine-grained parameter lists:**
    • Burden consumability and comprehension with clumsy, verbose signatures, resulting in difficult to read code.

- **Semantic operations - Concept**
  - Services, operations, and operation parameters, should use language that conveys the meaning of the operations to the consumer, and should represent the operation's actions.
- **Consequences:**
  - **Well named services:**
    • Benefit understanding by service consumers because they imply the service semantics or behaviour.
  - **Poorly named services:**
    • Burden comprehension and degrade semantic integrity.
Statelessness

● Concept
  – Service implementations should not hold conversational state across multiple requests.
    • Communicate complete information at each request.
  – Each operation should be functionally isolated (separate, independent).

● Consequences:
  – Stateless and connectionless services:
    • Benefit adaptability owing to the independence that exists between a client's successive service requests and the service instance that fulfils each request.
      – This is an enabler for improved runtime qualities (for example, service request throughput or concurrent service requests) via pooling and sharing of service instances (that is, client-service independence).
  – Stateful services:
    • Hinder adaptability as a consequence of tight dependency (coupling) between a client's successive service requests
      – There is then a need for a specific service instance to fulfil a particular request (that is, client-service affinity).
Efficient resource utilization

● Concept
  – Service implementations should be optimistic, brief and courteous with resource usage.

● Consequences:
  – Optimistic (loosely-coupled) resource usage:
    • Facilitate higher system performance qualities; concurrency, scalability and availability, through courteous use of resources.
  – Pessimistic (tightly-coupled) resource usage:
    • Hinder higher system performance qualities through resource monopolisation and obstruction; locking and blocking.
Service-oriented integration defined

Service-oriented integration (SOI) is defined as integrating computing entities using only service interactions in a service-oriented architecture (SOA).

- An architectural style which integrates service providers and service consumers through loosely-coupled services.
  - Changes to clients and service providers can be made independently from each other.
  - New protocols and interfaces can be integrated and supported quickly without impacting the service consumer.
- A programming model complete with standards, tools and technologies such as Web Services.
SOI principles – loose-coupling

- Services are loosely-coupled when changes to several aspects do not affect the consumers of the service.

- Tighter coupling tends to cost more over time:
  - Synchronizing multiple organizations on change.
  - Adapting, changing or redeploying updated components without affecting others.
  - Difficult to move, to scale, to distribute, and to replace.
  - More coupling implies more complex testing.

- Looser coupling requires greater investment up front:
  - More design work.
  - More implementation work.

Several service elements must be considered when thinking about coupling:

- Service
- Message
- Interface
- Contract
- Policy
- Conversation
- State
- Transactions
Loose-coupling aspects of service interactions

- Consumer
- Provider
- Location
- Language
- Data Format
- Delivery Assurance
- Interaction State
- Semantic Interface
- Platform
- Protocol
- Time
- Security
- Service Version
- Service Provider Identity
- Business Data Model

Decoupled to Coupled
**What is an enterprise service bus?**

An ESB powers your SOA by reducing the number, size, and complexity of interfaces.

**So what does this really mean for architecting an ESB?**

An ESB performs the following between requestor and service:

- **ROUTING** messages between services
- **CONVERTING** transport protocols between requestor and service
- **TRANSFORMING** message formats between requestor and service
- **HANDLING** business events from disparate sources

Shape = Protocol
Color = Data type
ESB – enabler for SOI

Result → Greater business responsiveness

- Allows for dynamic selection, substitution, and matching.
- Enables more flexible coupling and decoupling of the applications.
- Enables you to find both the applications and the interfaces for re-use.
- Decouples the point-to-point connections from the interfaces.
ESB role in SOI – service virtualization

- ESB acts as an intermediary (proxy) between requestor and provider.

- ESB provides service virtualization of:
  - Location and identity
  - Interaction protocol
  - Interface

- Interactions are decoupled, supporting separation of concerns.
Enterprise service bus reference architecture

Interaction, Process, Information, Partner, Business App, Access Services

Business Logic

Enterprise Service Bus

Interaction Patterns
Message Flows
Mediation Patterns
Message Models
Transport Protocols

Security
Management
IT Management Services
Registry
## ESB capabilities – functions and features

<table>
<thead>
<tr>
<th>Ubiquitous and heterogeneous</th>
<th>Explicit intermediary</th>
</tr>
</thead>
<tbody>
<tr>
<td>– Universal connectivity</td>
<td>– Proxy for non-enterprise connections</td>
</tr>
<tr>
<td>– For example, J2EE, .NET, Cobol</td>
<td>– Security control-point</td>
</tr>
<tr>
<td>Leverage existing investments</td>
<td>Service Management</td>
</tr>
<tr>
<td>– CICS, IMS, legacy</td>
<td>– Security and access control</td>
</tr>
<tr>
<td>– WBI and other services Adaptors and connectors</td>
<td>– Registry and discovery</td>
</tr>
<tr>
<td>Mediations</td>
<td>– Monitoring and policy-based admin</td>
</tr>
<tr>
<td>– Transformation</td>
<td>Standards-based</td>
</tr>
<tr>
<td>– Validation</td>
<td>– Interoperability with business partners, including security</td>
</tr>
<tr>
<td>– Pattern Recognition</td>
<td>– Vendor-independence</td>
</tr>
<tr>
<td>– Customized Routing</td>
<td>Tools</td>
</tr>
<tr>
<td>– Subsidiary functions (for example, metering and policy integration)</td>
<td>– Ability to deploy, manage, update, access thousands of services</td>
</tr>
<tr>
<td>Virtualization of services</td>
<td>– Ability to intuitively create BPEL linked to Business processes</td>
</tr>
<tr>
<td>– Dynamic selection of service providers</td>
<td>– Export of WSDL interface definitions for partners’ use</td>
</tr>
<tr>
<td>– Flexible management of QoS</td>
<td>– Enterprise apps invoking Web Services</td>
</tr>
</tbody>
</table>

© Copyright IBM Corporation 2006
IBM patterns vision – application integration

- A set of *tooled* pattern language families and “recipes” covering Service Design and Development for end-to-end solutions

Patterns for e-business

- Access Integration
- Self-Service
- Collaboration
- Information Aggregation
- Extended Enterprise
- (Management Integration)

SMB/Enterprise Grain Patterns and Industry Solutions

Micro Grain Patterns – Design Patterns/J2EE

UML2/MOF Language Foundation
Patterns for e-business – overview

● The patterns for e-business extend the domain of software patterns to earlier phases of the application development life cycle.
  – Maximize impact on key design decisions made early in the development cycle.

● Patterns for e-business help you to do the following:
  – Understand and analyze complex business problems and break them down into smaller, more manageable functions.
  – Capture and publish e-business artifacts that have been used, tested, and proven to be successful.
  – Speed up the solution development and integration process through documented and proven architectural constructs.
  – Assemble an end-to-end solution (and minimize custom one-of-a-kind implementations).
  – Apply expert knowledge to projects with teams that are new to e-business development.
Process for applying patterns for e-business

- Capture customer requirements
  - Standard RUP activity, generates use cases.
- Fit-gap analysis and generic use-cases
  - The Business Analyst or Software Architect decides which generic use-cases apply.
- Select SOA Entry point scenarios (next unit)
  - Using these generic use-cases the Business Analyst or Software Architect matches against the corresponding scenarios.
- Re-use patterns assets to accelerate the solution architecture
  - Exploit the patterns associated with the previously identified scenarios to architect your SOA solution. Discussed next.
- Select the relevant implementation guide
Application and runtime patterns defined

- Application patterns are used to design application architectures.
  - They show the levels of detail of interest at the business or developer level.
- Application patterns are mapped (‘deployed’) onto runtime patterns. Runtime patterns are used to design infrastructure architectures.
SOA runtime pattern family

Infrastructure components for SOA

Enterprise Service Bus
Routing, transformation, mediations, security, and so forth

ESB Gateway

External Service Providers

Internal Service Providers

External Service Consumers

Internal Service Consumers

Business Service Choreography

ESB Namespace Directory

Business Service Directory
SOA runtime pattern family – service consumers and producers

Infrastructure components for SOA
SOA runtime pattern family – ESB and ESB Gateway

Infrastructure components for SOA
SOA runtime pattern family – business service choreography

Infrastructure components for SOA
SOA runtime pattern family – Business Service Directory

Infrastructure components for SOA
SOA Process Integration application patterns

Serial Process
Variation: Serial WorkFlow
Focus: Single series of operations on multiple targets

Direct Connection
Variations: Message/Call Connection
Focus: Sending messages on a single path to a single target

Parallel Process
Variation: Parallel WorkFlow
Focus: Adds joining, splitting, and so forth of multiple series of operations on multiple targets

Serial Interaction
Yes

No

Parallel Interaction
Yes

No

Serial Process Mgr
T1
T2
T3

Serial Interaction

Direct Connector
T

Serial Interaction

Parallel Process Mgr
T1
T2
T3

Parallel Interaction

Broker
Variation: Router
Focus: Adds switching, routing, splitting, joining messages on multiple paths to multiple targets

Broker
T1
T2
T3

Serial Interaction

Parallel Interaction

Serial Interaction

Parallel Interaction

© Copyright IBM Corporation 2006
Direct Connection ESB interaction patterns

Message Connection

Consumer → ESB → Provider

Call Connection (asynchronous or synchronous)

Consumer ↔ ESB ↔ Provider
Broker ESB interaction patterns

**Router variation**

- Consumer
  - Request Service A
  - ESB
    - Router Mediation
    - Provider of Service A
    - Provider of Service B

**Broker variation**

- Consumer
  - Service A Request
  - ESB
    - Broker Mediation
    - Provider 1 of Service A
    - Provider 2 of Service A
Serial and Parallel Process ESB interaction patterns

- The Business Service Choreography (BSC) component works together with the ESB to provide both mediation and process orchestration features (focus of this course).
Having completed this unit, you should be able to:

● Describe various design principles that can be applied during service design within SOA

● Define SOI and describe patterns and principles that can be applied towards developing SOI architecture

● Define ESB and explain its role in SOI

● Describe IBM’s patterns for e-business and the process for applying these patterns

● Describe SOA Runtime patterns such as ESB, ESB Gateway, Business Service Choreography, Business Service Directory

● Explain the SOA Application patterns such as the Direct Connection, Broker, Serial Process, Parallel Process, plus their variations and their realization using runtime patterns