Module 2
SOA Technology Concepts

Extensible Markup Language (XML)

A Brief History

- The Extensible Markup Language (XML) gained popularity during the eBusiness movement of the late 90's, when server-side scripting languages made conducting business via the Internet viable.
- Through the use of XML, developers were able to attach meaning and context to any piece of information transmitted across Internet protocols.
- Not only was XML used to represent data in a standardized manner, the language itself became the basis for a series of additional specifications.

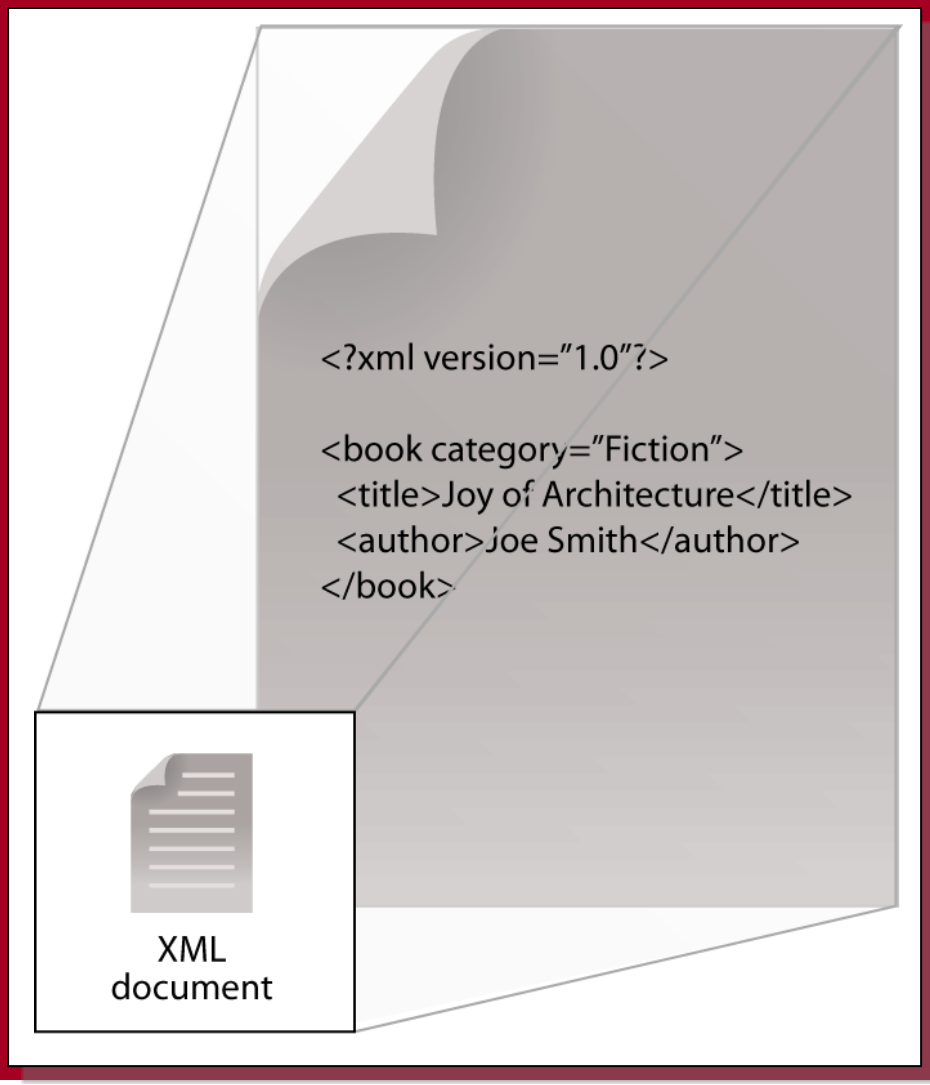
Introduction

- Standardizing the representation of data using XML allows for the creation of an XML data representation architecture.
- This is the most common form of data representation architecture for SOA.
- XML (and XML-based technologies, such as XML Schema) are used for both Web services and REST services.
- It establishes a foundation layer upon which the service layer is eventually built.
- The service layer is commonly comprised of endpoints established by service contracts.

XML as Metadata

XML is metadata.

An XML document contains actual data and the XML metadata that describes the data.



XML vs. HTML

- Like HTML, XML was developed by the World Wide Web Consortium (W3C).
- As with HTML, XML is realized using elements.
- Unlike HTML, XML elements are not predefined.
- As a result, elements can be customized.
- Elements represent metadata values and are used to associate metadata with document data.

XML vs. HTML

Elements are referred to as "tags" when implemented.

HTML

```
<b>John</b>
```

We know that "John" is bolded due to the <b> tag, but we don't know what this data means.

XML

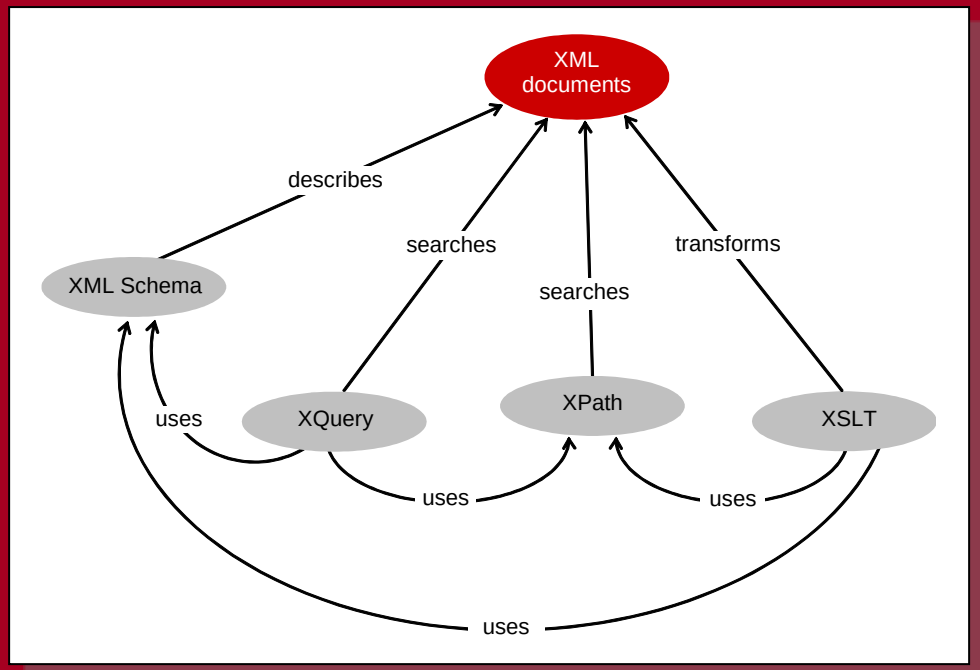
```
<name>John</name>
```

We know that "John" represents a name because of the <name> tag.

XML Technologies

The core family of XML technologies consists of:

- The XML Schema Definition Language (XML Schema)
- XML Style Sheet Language Transformations (XSLT)
- XML Query Language (XQuery)
- XML Path Language (XPath)



XML Schema Definition Language (XML Schema)

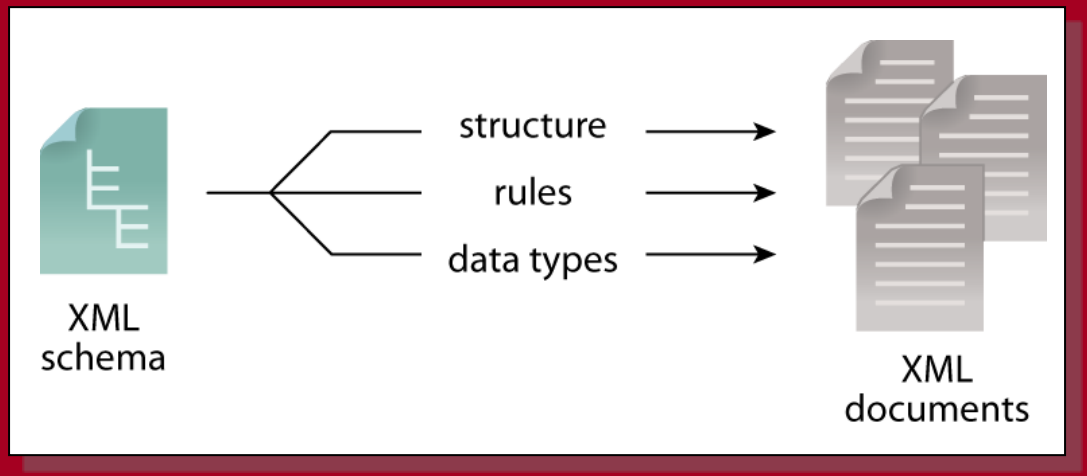
Introduction

- A set of predefined and related XML elements represents a vocabulary.
- Vocabularies can be created to describe specific types of business documents, such as invoices or purchase orders.
- Vocabularies can be formally defined using a schema definition language.
- A schema language essentially provides a means of expressing the definition of a data model for a vocabulary.

Introduction

Schemas protect the integrity of XML document data by defining:

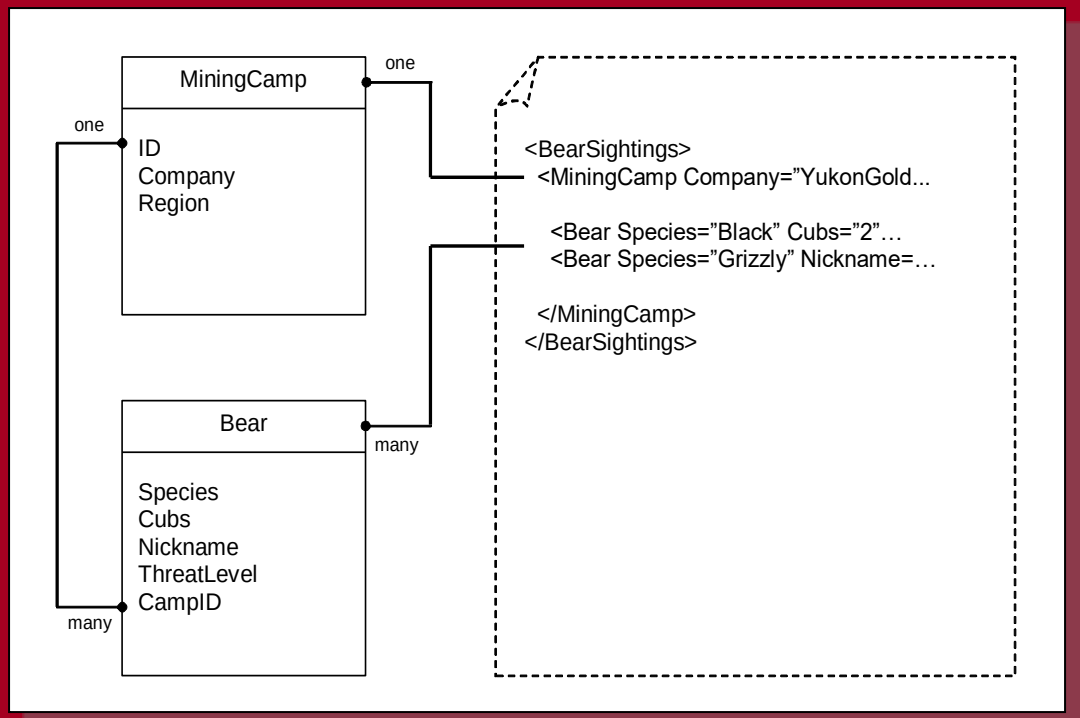
- structure
- validation rules
- type constraints
- inter-element relationships



XML Schema vs. Relational Database

- XML schemas are comparable to database data models, which makes XML documents comparable to database records.

- Unlike data representation in relational databases, XML documents need to be based on a tree or directory-type structure.



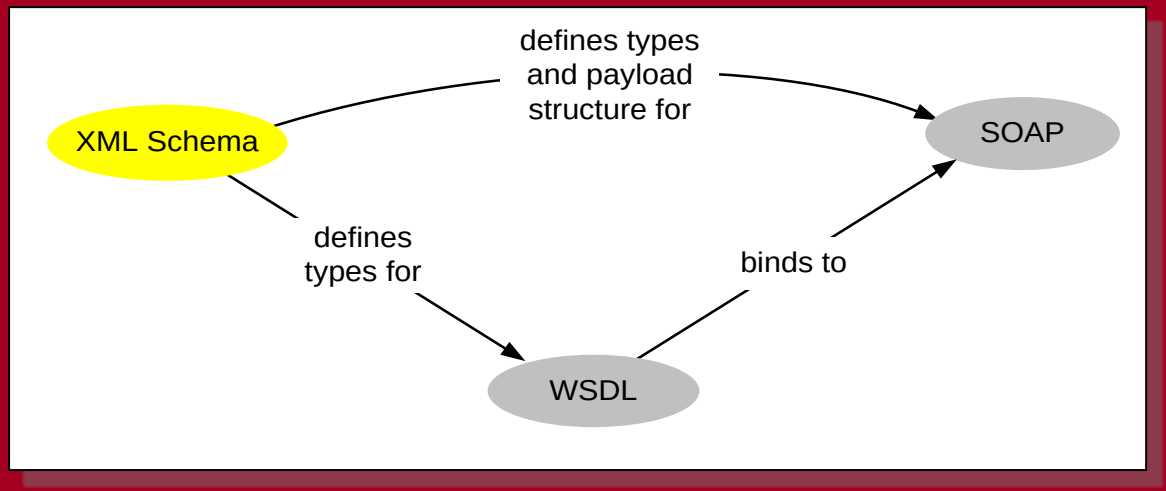
XML Schema Languages

- Numerous XML schema languages exist. The two most common are:
  - Document Type Definitions (DTD)
  - XML Schema Definition Language (XML schema or XSD)
- The XML schema language was also produced by the W3C and is by far the most common language used in service-oriented solutions.

XML Schema, WSDL and SOAP

XML schema is used intrinsically by many Web services specifications to define the data models for the corresponding specification languages.

XML schema is also used together with WSDL to establish the data model for SOAP-based Web service contracts. (This relationship is explored later in this course.)



XML Schema Definitions

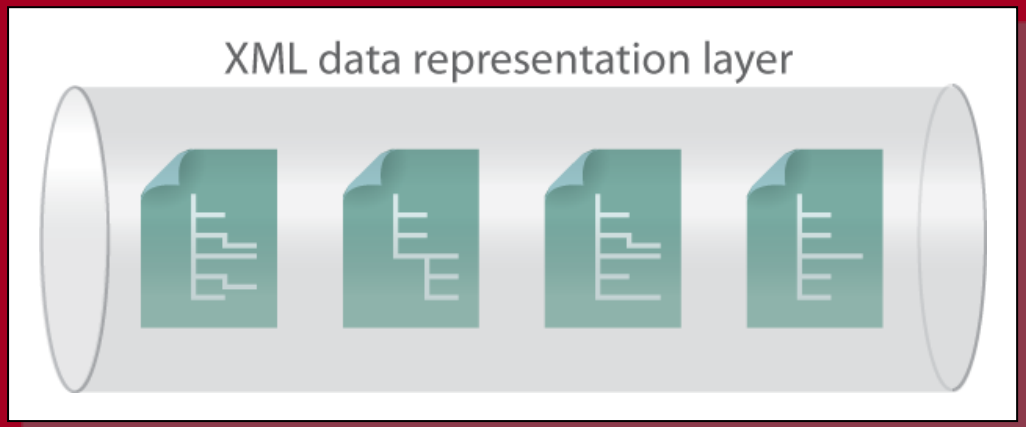
An XML schema usually exists as a separate file in which the data model for a specific type (vocabulary) of XML document is defined.

XML Schema Definitions

- Multiple XML documents can link to the same XML schema definition.
- One XML document can link to multiple XML schema definitions.
- XML schemas can also be embedded within XML documents.
- Most commonly, XML schemas exist as separate files.

XML Schema Types

- One of the most important features of the XML schema specification is its wide range of support for vendor-neutral data types.
- When creating a data representation architecture for SOA, it is these data types that can blanket disparate data sources with a federated and standardized data representation layer.



XML Schema Types

- Elements defined in XML schemas that contain child elements or attributes have complex types.
- Elements that only contain values have simple types.
- All attributes have simple types because they only contain values.
- Simple types are used to refer to built-in data types.

XML Schema Types

- Complex types are typically constructs or mini element hierarchies that can represent a large body of related data.
- When creating service contracts, it is common for complex types to represent the entire structure of input and output messages.

Address
- Street
  - Name
  - Number
- City
- State
- Zipcode

A sample Address complex type construct.

Namespaces

- Namespaces are an important part of XML-based languages that essentially establish a means of creating unique identifiers.
- Every XML language (including XML Schema, WSDL, SOAP, WS-Addressing, WS-BPEL, etc.) has a pre-defined namespace associated with it.
- This way, when these elements get mixed together in the same XML document, the runtime environment can figure out which language each element belongs to.
- As a result, multiple elements within the same XML document may be parsed and validated using different processors.

Namespaces

- A namespace declaration establishes a prefix:

  `abc="123"`

  In this case, `abc` is an alias used to represent the namespace `123`.

- The prefix is associated with elements as follows:

  `abc:Invoice`

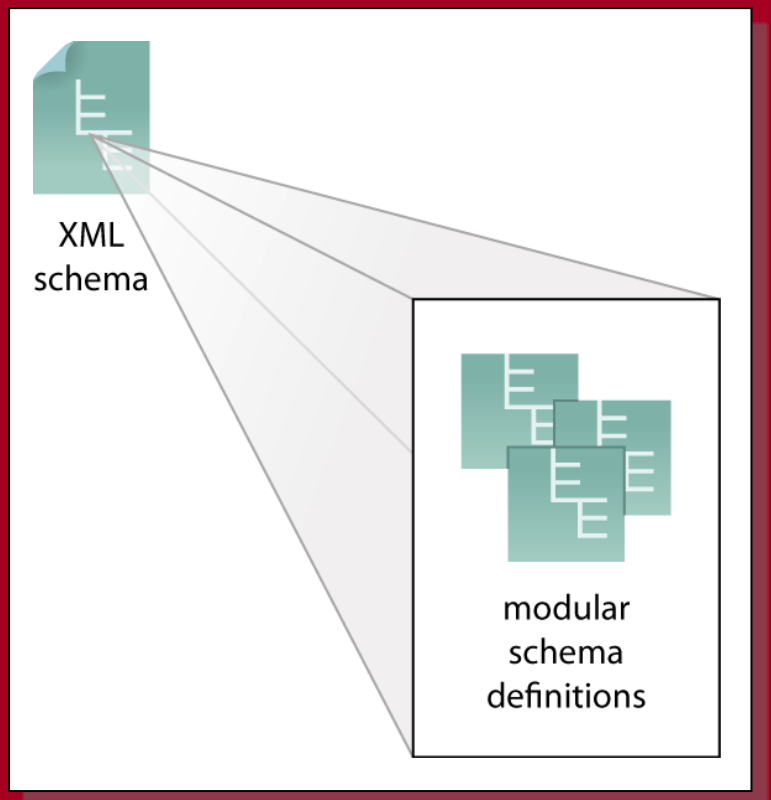
  The `Invoice` element is associated with the `123` namespace because it is prefixed with `abc`.

Namespaces

- XML schemas also provide support for namespaces that allow schema authors to differentiate between similar XML vocabularies.
- Namespace values are traditionally URLs, but they don't have to be.
- Namespaces are especially important when working with WSDL definitions because these documents usually require that elements from WSDL, SOAP, and XML Schema documents co-exist with additional elements from custom XML vocabularies.

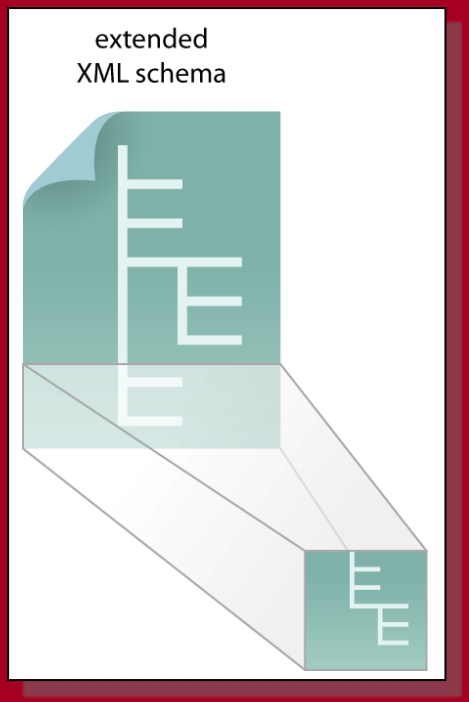
Modular XML Schemas

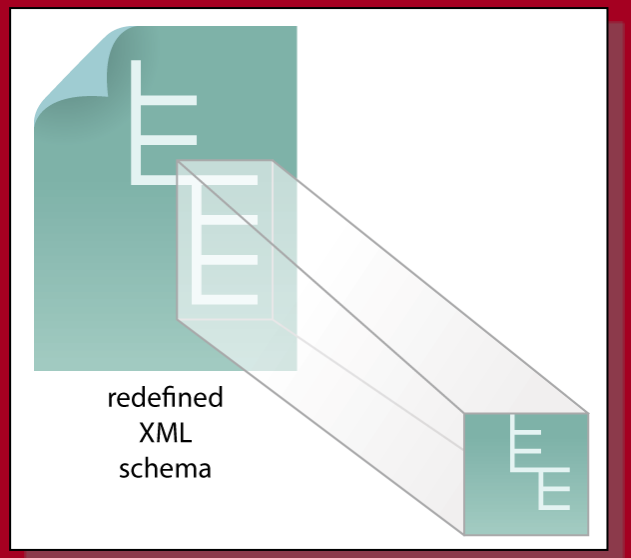
- An XML schema definition is capable of containing multiple schema documents.
- This allows for the creation of modular schemas that can be combined.
- The use of schema modules promotes reuse and the usage of standardized schemas.



Extending and Redefining XML Schemas

Each schema can be dynamically extended with supplementary constructs.





Parts of a schema definition can be redefined (overridden) by other schema definitions.

XML Schema Challenges

Common challenges with XML schemas and SOA:

- XML schemas are often task, process, or solution-specific.
- Some advanced XML schema features are not supported with WSDL.
- XML schemas can be complex and verbose.
- DTDs are sometimes still in use.

XSLT, XQuery, and XPath

XSLT

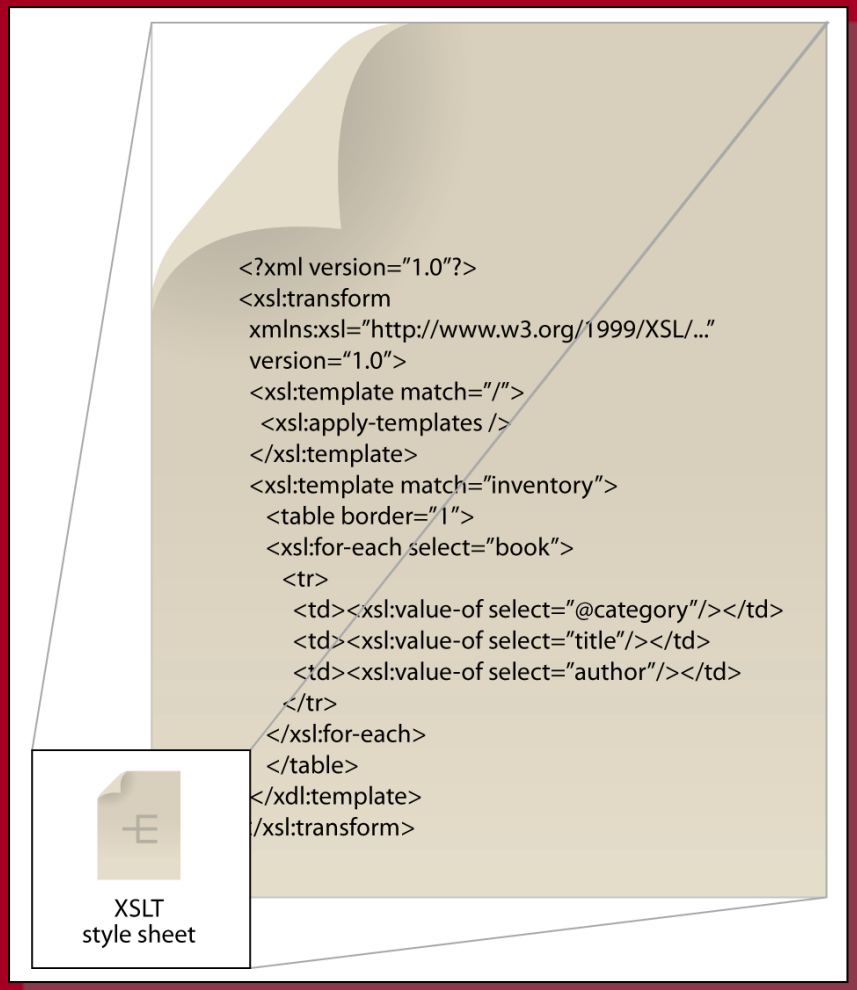
- There is often a need for the structure (data model) of an XML document to be dynamically changed.
- The most common requirement for this is when two services or applications need to exchange the same type of data, but each uses a different XML schema (data model).
- To overcome this incompatibility, the data must be "transformed".

XSLT

- Because XML provides a separation of content and structure, the output structure of an XML document can be separately controlled.
- An XML-based technology called XSL Transformations (XSLT) is the foremost means of dynamically converting the structure of XML documents.
- The use of XSLT facilitates data transformation requirements.

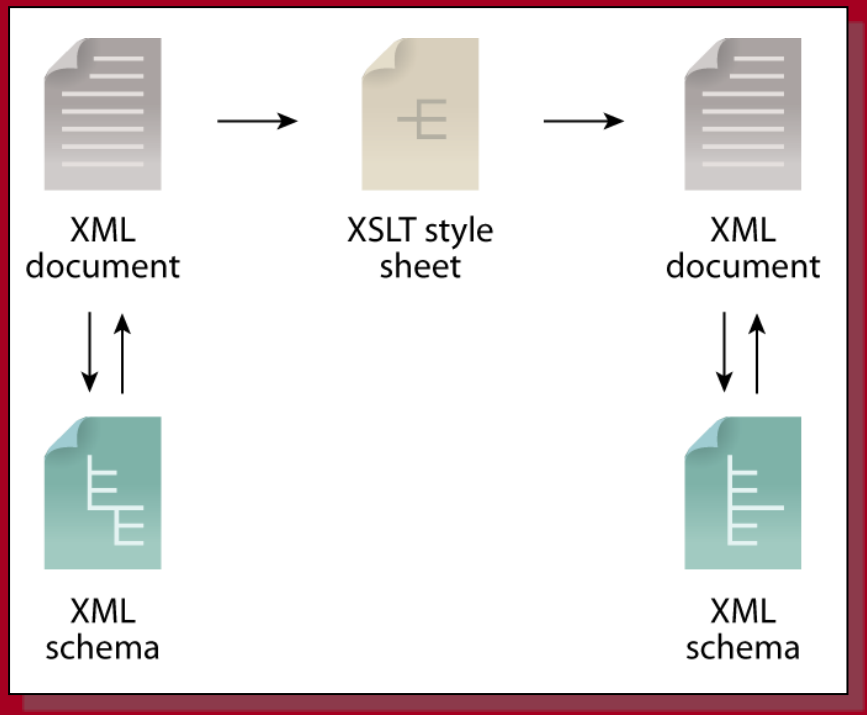
XSLT

- XSLT logic exists in XML documents called XSLT Stylesheets.
- The same XSLT Stylesheet can be used by multiple XML documents.
- XSLT is another language developed by the W3C.



XSLT

- XSLT Stylesheets map one schema structure to another.
- XSLT Stylesheets carry out this mapping logic at runtime to perform structural data transformation.
- This allows services to overcome data representation disparity.



XSLT

The XSLT feature set facilitates the:

- manipulation
- ordering
- filtering

... of XML document data to provide alternative views and renditions of information for any number of document transformation scenarios.

XSLT

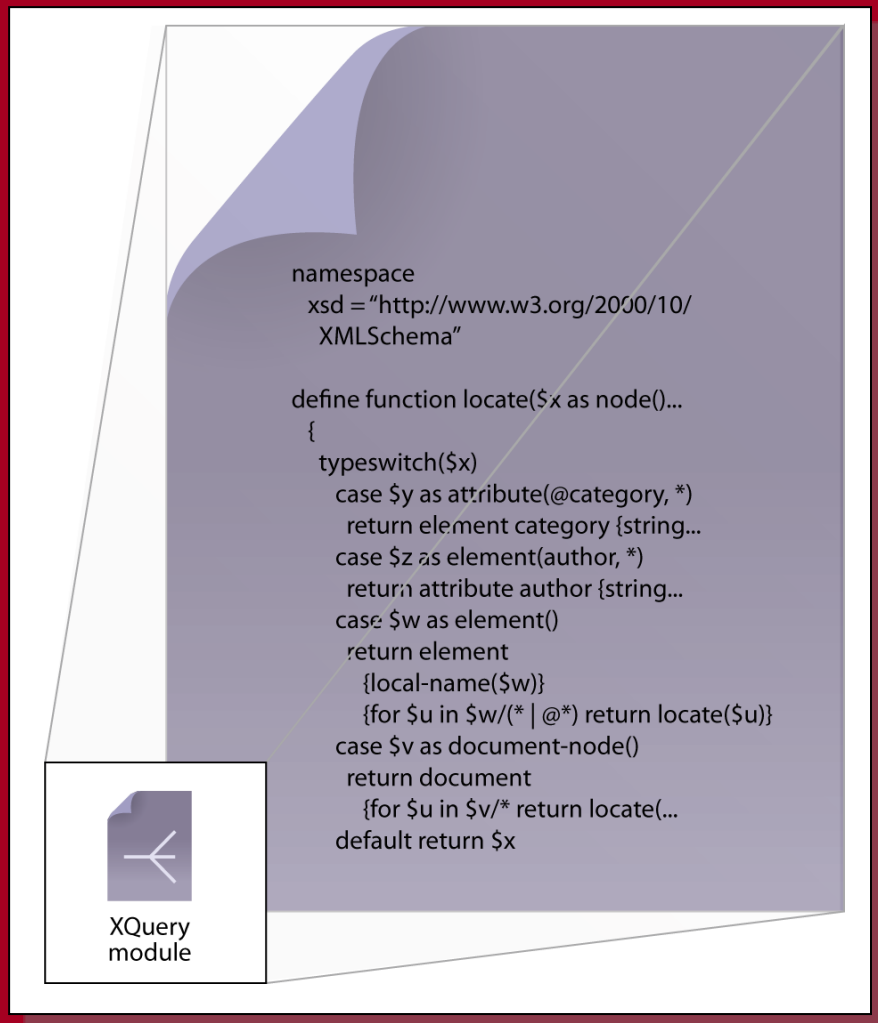
Note:

Although XSLT is a proven and established technology used to enable interoperability, it is a technology we try to avoid having to use when designing services for SOA.

The transformation layers introduced by XSLT Stylesheets add development effort, design complexity, and runtime performance overhead – all of which can be avoided by standardizing XML schema definitions.

XQuery

- The XQuery specification establishes a comprehensive data query language, designed specifically for XML documents.
- XQuery is to XML documents what SQL is to relational databases.
- Some XQuery syntax is actually derived from SQL.

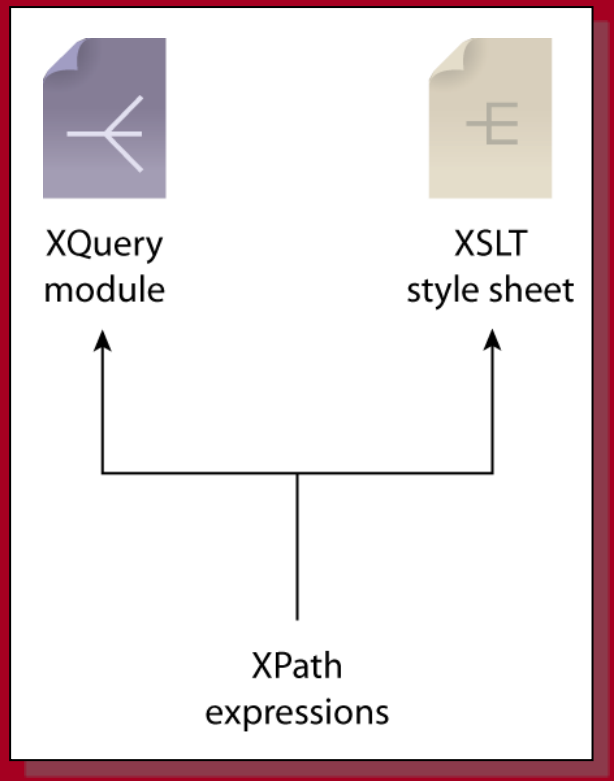


XPath

- Elements or element values within XML documents can be searched and filtered using XPath functions.
- XPath approaches the addressing of an XML document tree similarly to how traditional file paths refer to directory structures, or how virtual paths refer to internal Web site structures.
- As a result, XPath addresses can be relative or absolute.

XPath

- XPath statements are called expressions.
- XPath expressions are often embedded within XSLT Stylesheets and XQuery Modules.
- XPath is one of the few remaining non-XML specifications.



Exercises

Exercises 2.1, 2.2, 2.3

Create an XML Vocabulary and Solve a Transformation Problem

Web Services, REST Services, and Traditional Distributed Architecture

Web-based Service Nomenclature

- Currently, there are two distinct types of services that leverage the open infrastructure of the Web:
  - Web services
  - REST services
- Although they are complementary and share some of the same basic infrastructure, they are distinctly different. Their differences will be described later in this module.
- This initial section focuses on a fundamental comparison of Web-based service architecture and traditional, component-based distributed architecture.

Web-based Service Nomenclature

- Note that both REST services and SOAP-based Web services are sometimes collectively referred to as Web services or Web-based services.
- Note also that REST services are often referred to as RESTful services.
- In this program, the term Web service and SOAP-based Web service are synonymous and the term REST service is used instead of RESTful service.
- In this program, we occasionally also refer to SOAP-based Web services and REST services collectively as Web-based services.

Web-based Services and Traditional Distributed Architecture

In traditional distributed solutions:

- Solution logic is partitioned into components.
- Components exist as self-contained units of software capable of being aggregated.
- Component development technology is generally proprietary.
- Component communication technology is generally proprietary.

Web-based Services and Traditional Distributed Architecture

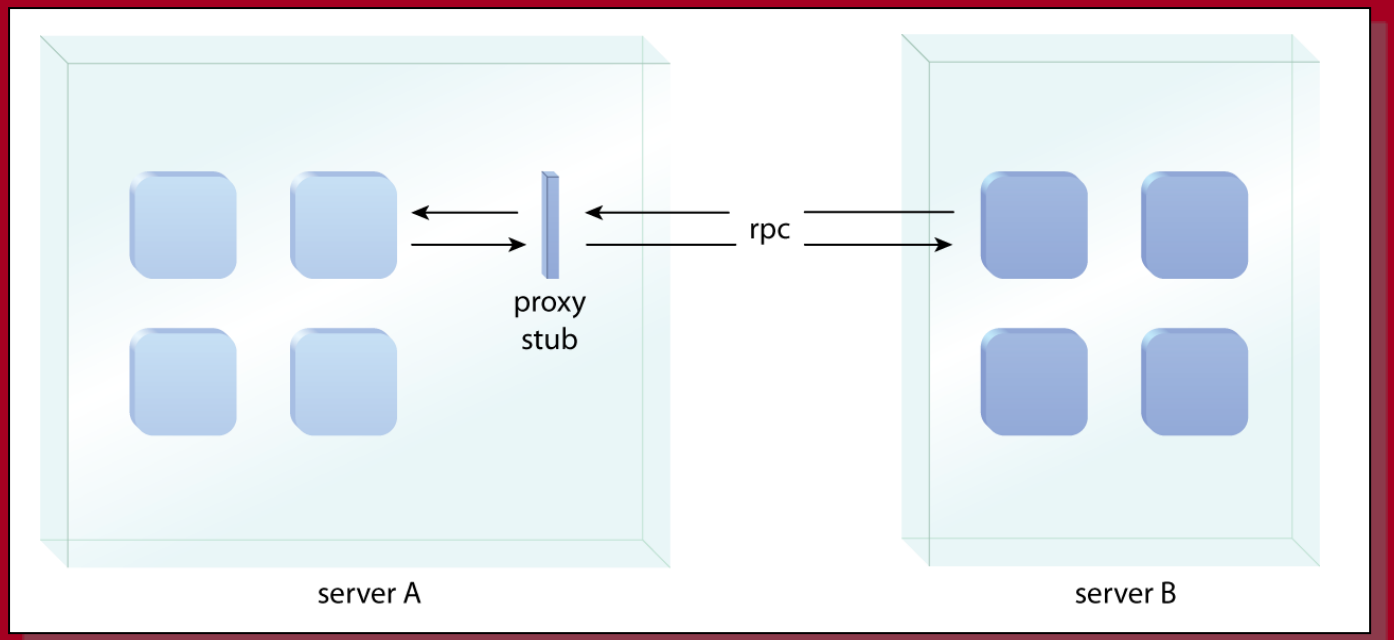
- Like components, Web-based services are self-contained units of solution logic capable of being aggregated.
- Like components, Web-based services can be developed using proprietary technology and the underlying solution logic may or may not be component-based.
- Unlike components, Web-based services are often designed to communicate with non-proprietary technologies based on industry standards.

Web-based Services and Traditional Distributed Architecture

- Components in traditional distributed applications can reside on one or more physical servers.
- Components on the same server typically communicate using proprietary APIs.
- Components that need to communicate across server boundaries have generally relied on Remote Procedure Call (RPC) technology.

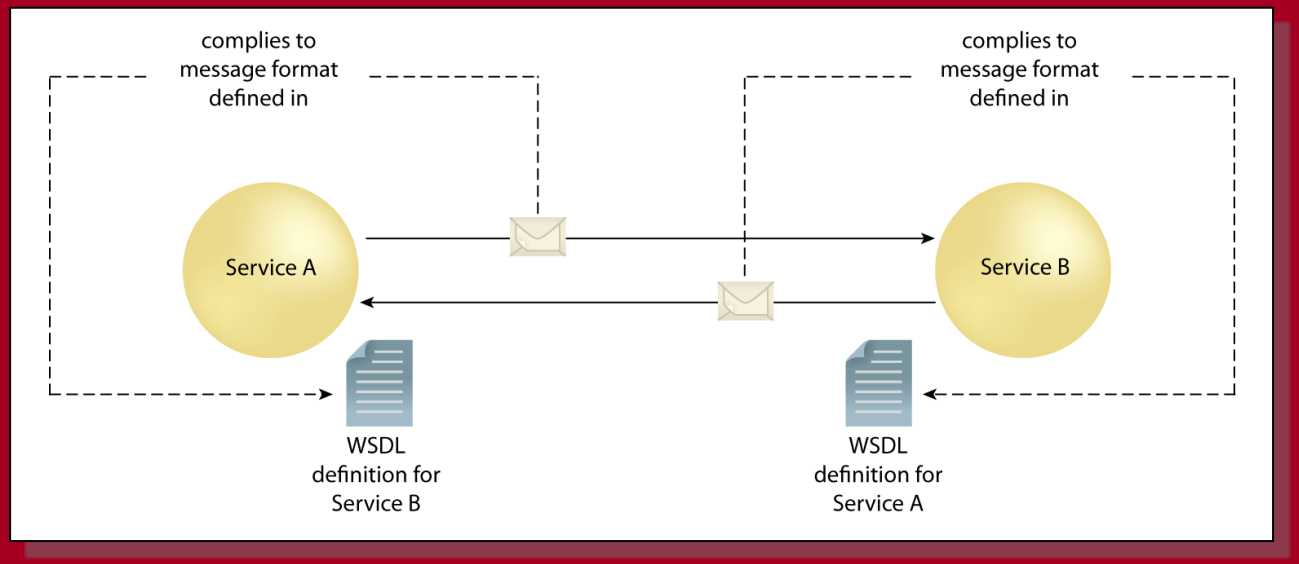
Web-based Services and Traditional Distributed Architecture

RPC protocols enable remote component communication through the use of proxy stubs that represent remote components on local servers.



Web-based Services and Traditional Distributed Architecture

- Like components, Web-based services are capable of communicating across server boundaries.
- Unlike components, Web-based services generally do not use proprietary APIs or RPC technology for remote communication.
- Web-based services rely on the exchange of data defined by their service contracts.



Web-based Services and SOA

- A service-oriented architecture is a form of distributed architecture in which solution logic is exposed via standardized service contracts.
- Different service implementation mediums can use different technologies or industry standards to define service contracts.
- For example, SOAP-based Web services utilize the WSDL language to express service contracts, while REST services rely on the uniform contract provided by HTTP.

Web-based Services and SOA

- Web-based services can be designed for use within or as extensions to traditional distributed solutions.
- Or, Web-based services can be designed for use in service-oriented solutions.
- This flexibility is one of the reasons Web-based services have been so popular and widely used.
- It also reveals the fact that the mere use of Web-based services does not result in service-oriented solutions.

Services as Components

- SOA as an architectural model is neutral to any one technology platform.
- Essentially any implementation technology that can be used to create distributed systems may be suitable for service-orientation.
- Common implementation mediums for building services are:
  - components
  - Web services (or SOAP-based Web services)
  - REST services

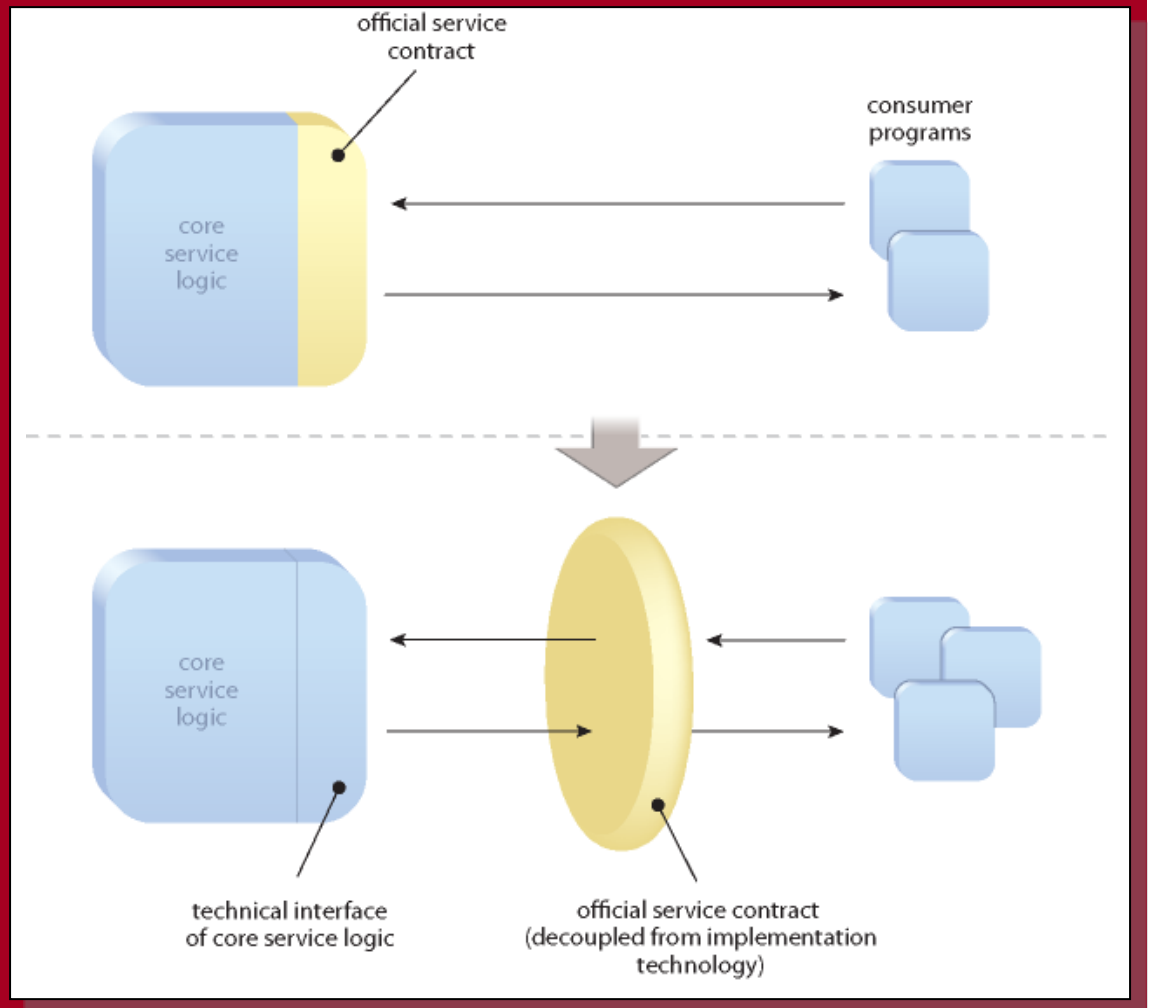
Services as Components

- Even though they are not Web-based, components can be designed as services.
- To design a component as a service, the component is shaped by the application of service-orientation design principles to whatever extent possible.
- It can be beneficial from a performance perspective to have services exist as components; however, because the service contract is not decoupled it can inhibit the application of certain principles.

Services as Components

With components the service contract is physically coupled to the solution logic.

With SOAP-based Web services and REST services, the contract is physically decoupled. This can be beneficial for the application of service-orientation.



Service Roles

Service Roles

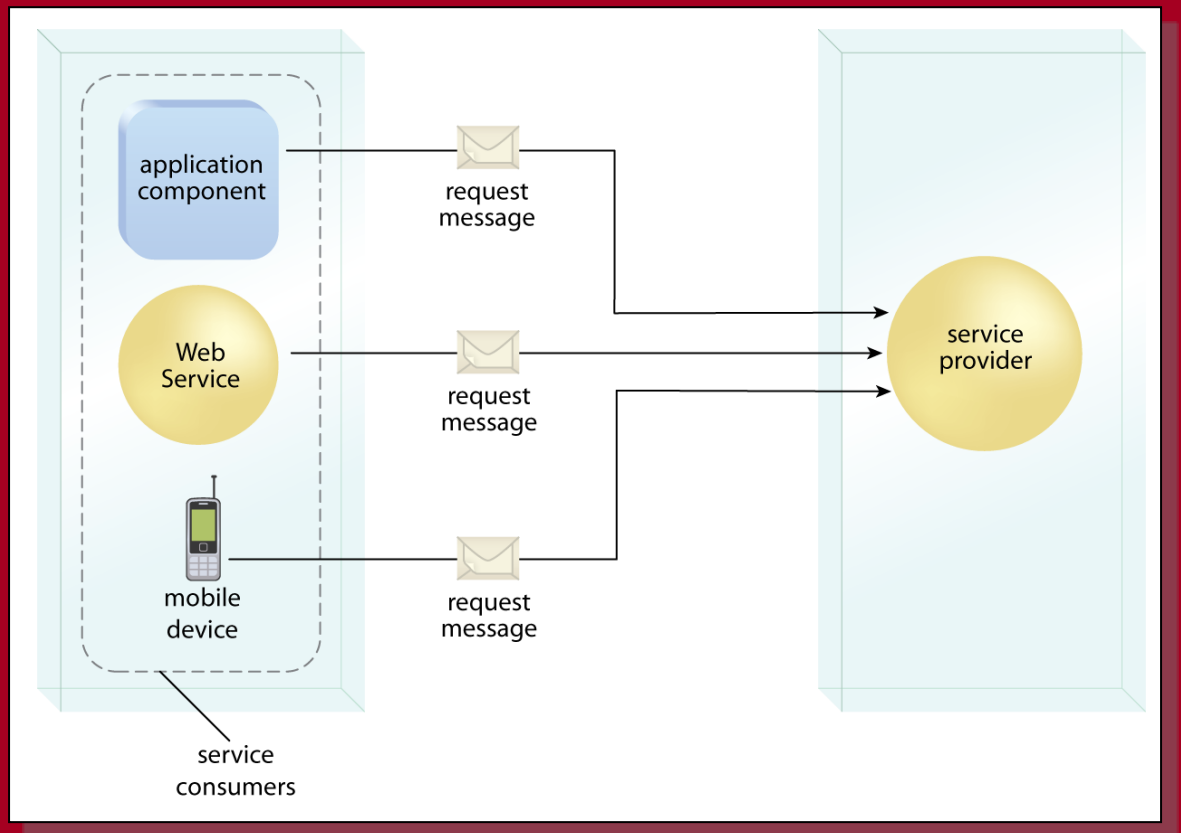
- A service is capable of assuming different roles, depending on the runtime context within which it is utilized.
- For example, a service can act as the initiator, relayer, or the recipient of a message.
- A service is therefore not labeled exclusively as a client or server, but instead as a unit of software capable of altering its role.

Service Roles

- The service provider role is assumed by a service that offers its capabilities to service consumers.
- A service is most commonly associated with this role, which is why this term is often not used when referring to a service in a runtime scenario.
- The service consumer role is assumed by any software program invoking a service provider.
- A service temporarily acts as a service consumer when it invokes another service.
- A service consumer is also called a "service requestor" (a term that originated with the W3C).

Service Roles

A service provider can be accessed by a variety of service consumers.



Service Roles

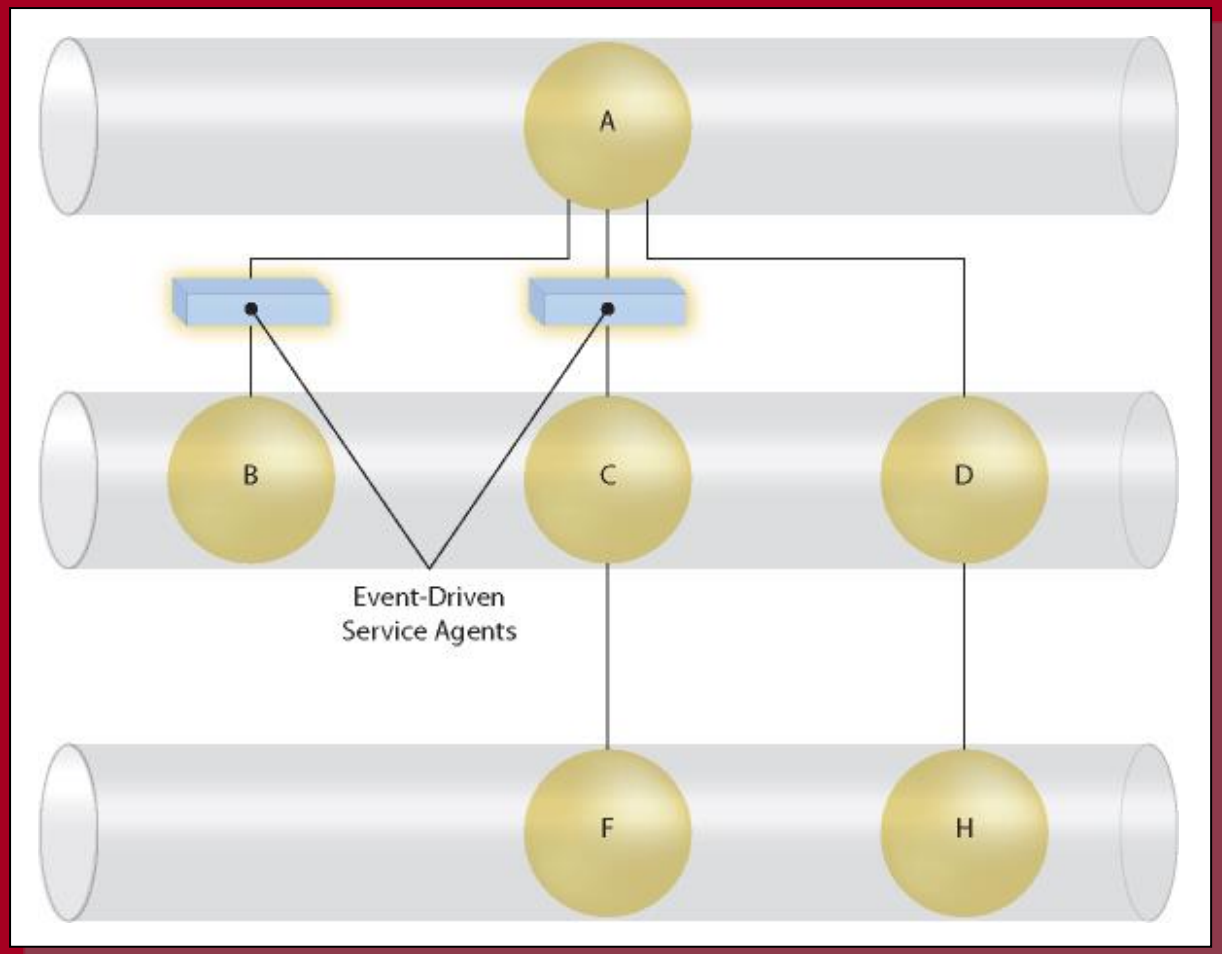
- Unlike traditional point-to-point channels, communication involving services is not limited to two points.
- A service can receive a message and then forward it to another service, and so on… (which is better described as point-to-*).
- A chain of services can be involved in the processing of a single message and as part of a single task.

Service Roles

- A message can be processed by multiple intermediate routing and processing programs before it arrives at its ultimate destination.
- The route a message takes is the message path.
- A message path may be predetermined or dynamically determined at runtime.
- Intermediaries are programs that perform intermediate processing tasks.
- There are two types of intermediaries:
  - service agents
  - intermediary services

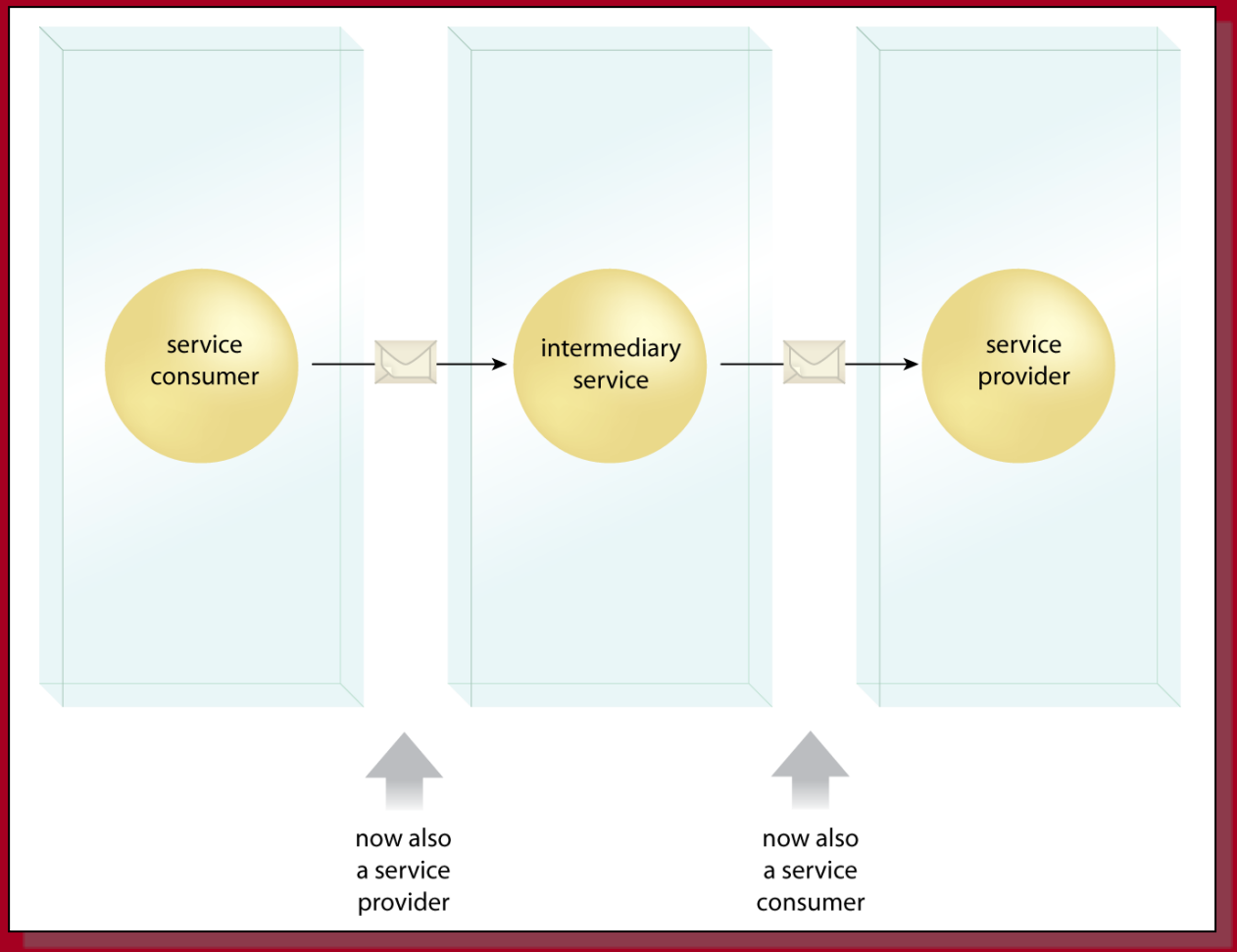
Service Roles

Service agents exist as lightweight, event-driven programs that automatically intercept and process messages.



Service Roles

Intermediary services are services that transition through the roles of service provider and service consumer.



Service Roles

There are two further categories for intermediaries:

- passive intermediary - Routes messages without modifying them. Service agents are commonly passive intermediaries.
- active intermediary - Routes, processes, and modifies messages. Both services and service agents can act as active intermediaries.

Service Roles

Additional roles exist to label the position of a service along a message path:

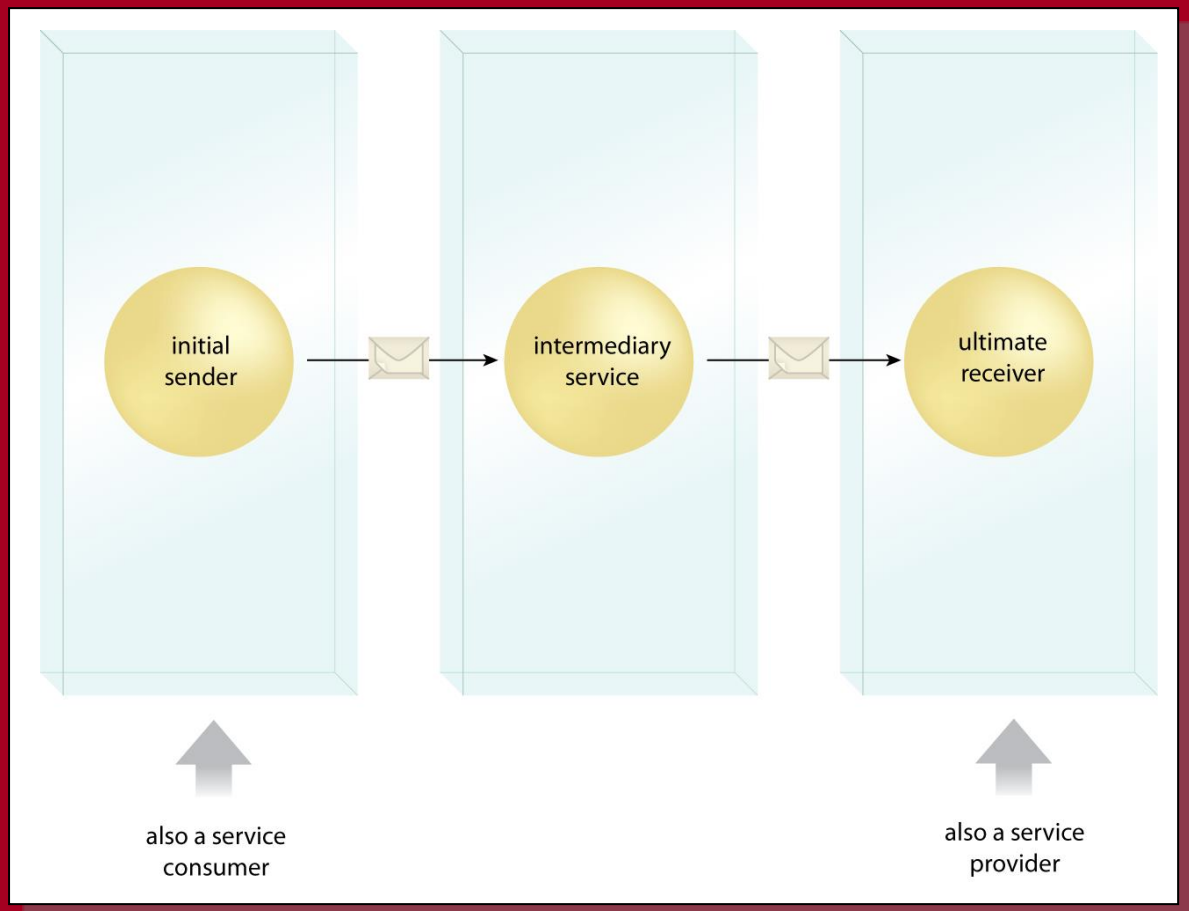
- initial sender – This role is assigned to the service consumer that initiates the message transmission (the starting point of the message path).
- ultimate receiver – A role assigned to the service that is a message's ultimate destination (for example, the last service along the message path).

Service Roles

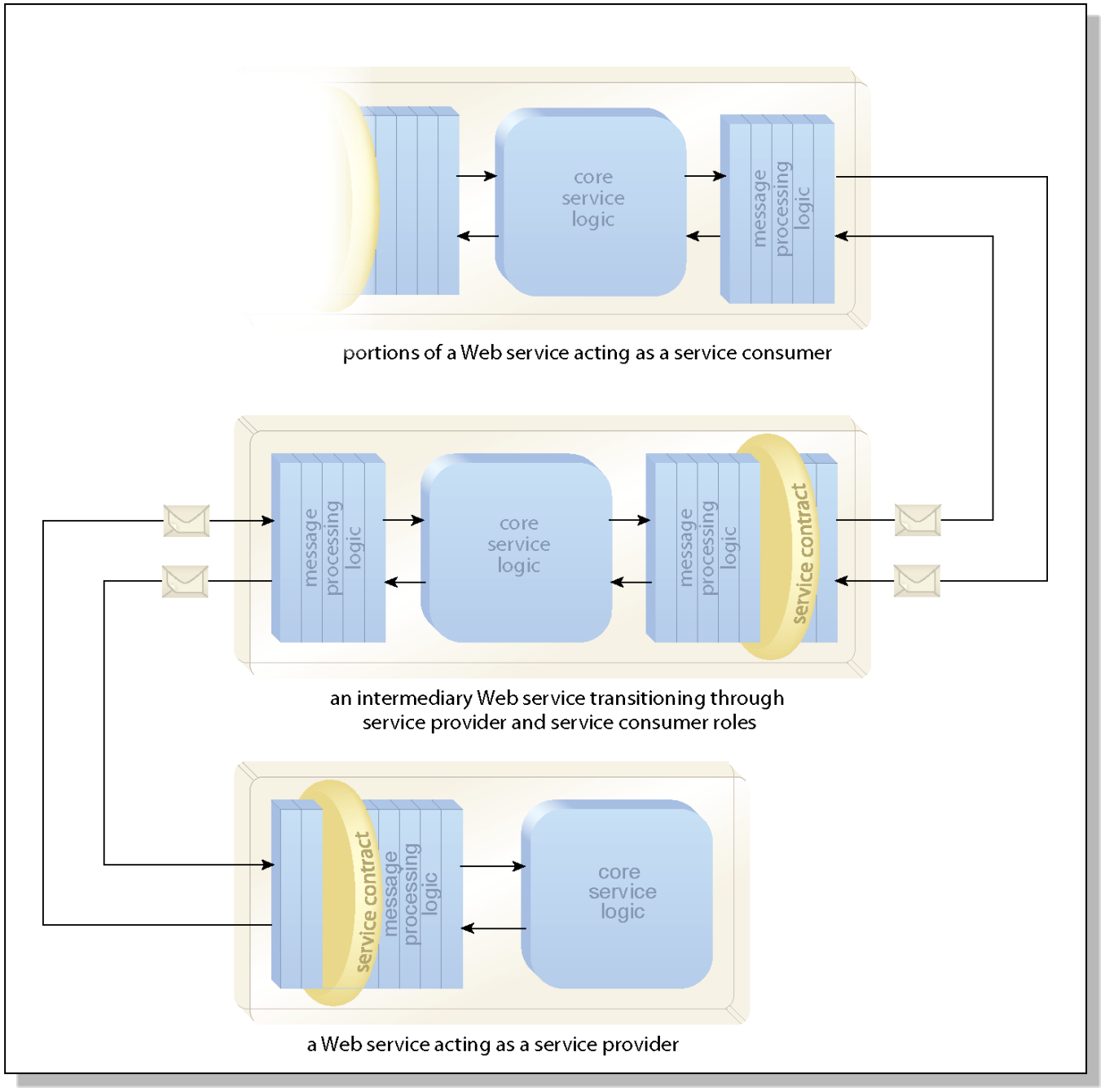
An initial sender is always also a service consumer.

An ultimate receiver is always also a service provider.

An intermediary is never an initial sender or an ultimate receiver.



An example of Web services assuming different roles as part of a message exchange.



Service Roles

Whereas service roles represent temporary classifications based on a service's runtime utilization, service models are permanent classifications based on the nature of logic encapsulated by a service.

Exercises

Exercise 2.4, 2.5, 2.6

Identify Service Roles, Intermediary Types, and Role Transitions

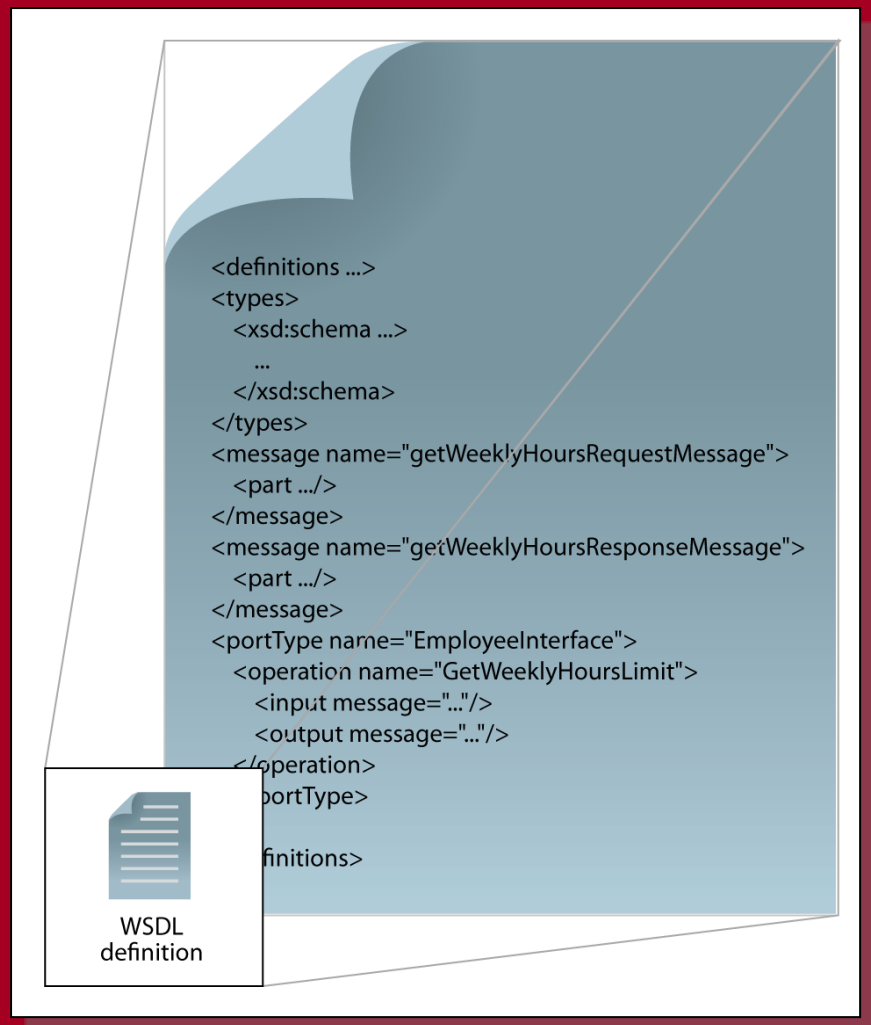
Web Services Overview

A Brief History

- In 2000, the W3C received a submission for the Simple Object Access Protocol (SOAP) specification.
- SOAP essentially established a standardized format for XML messages.
- SOAP was originally intended to unify (and in some cases replace) binary RPC-style communication through XML and HTTP.
- After SOAP was in use, the industry began to recognize the potential of standardized Web-based data exchange.
- This led to the idea of creating an industry-standard, vendor-neutral, Web-based communications framework.
- This concept was called Web services.

A Brief History

- One of the first initiatives in support of Web services was the Web Service Description Language (WSDL), also by the W3C.
- WSDL established a standardized means of expressing a messaging-based technical interface (or a technical contract) via the XML language.



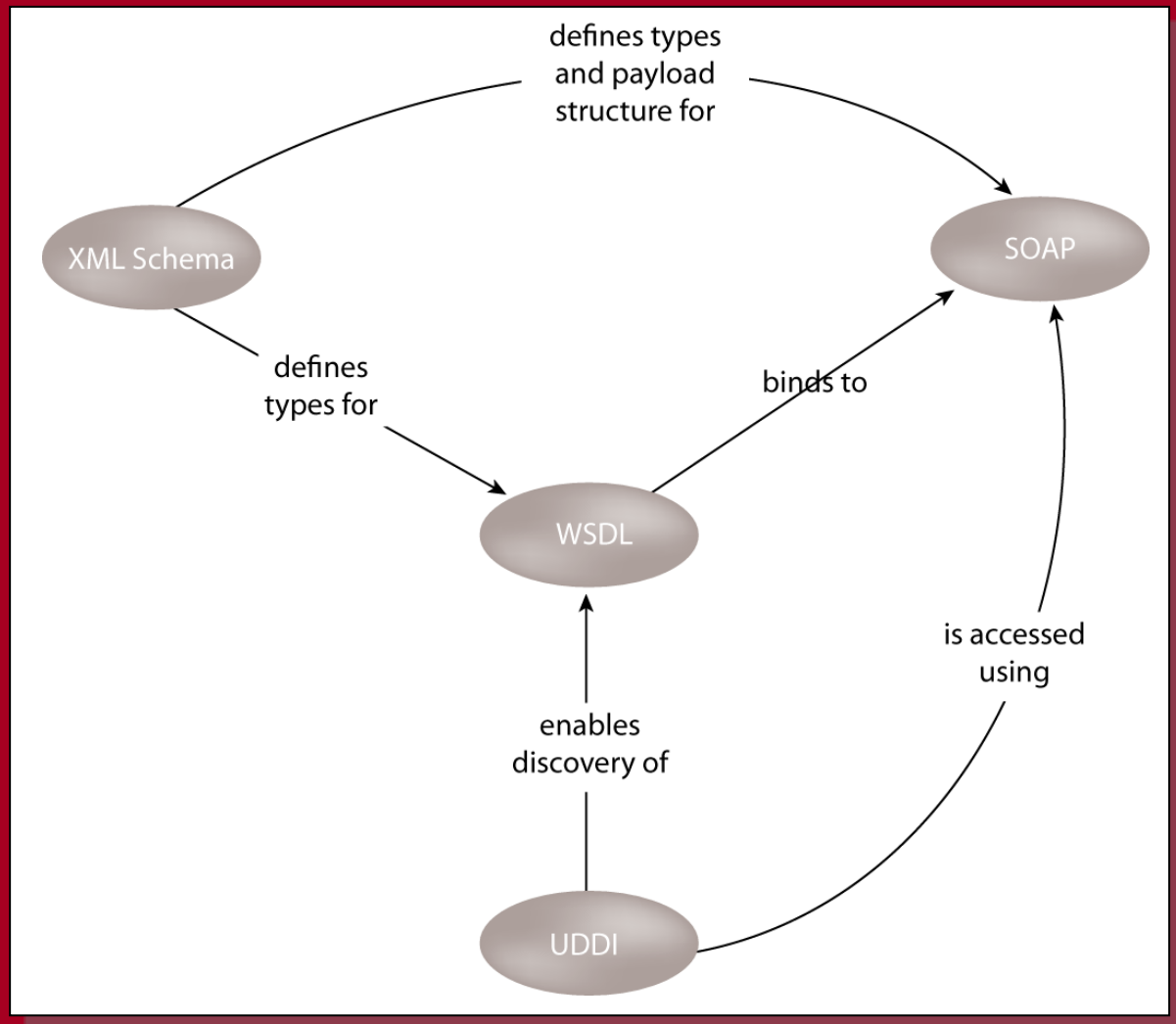
A Brief History

- After WSDL became somewhat developed, the industry looked for a standard messaging format to be used with WSDL interfaces.
- Although alternatives, such as XML-RPC, were considered, SOAP won out as the industry favorite.
- The combination of WSDL and SOAP established a core, industry-standard communications framework.
- The Universal Description, Discovery, and Integration (UDDI) specification was also created.
- Its purpose was to allow organizations to discover Web services published by others.
- UDDI was designed in support of wide-spread use of Web services, but was not as commonly used as WSDL and SOAP.

The First-Generation, Web Services Framework

WSDL, SOAP, and UDDI established the first-generation SOAP-based Web services framework.

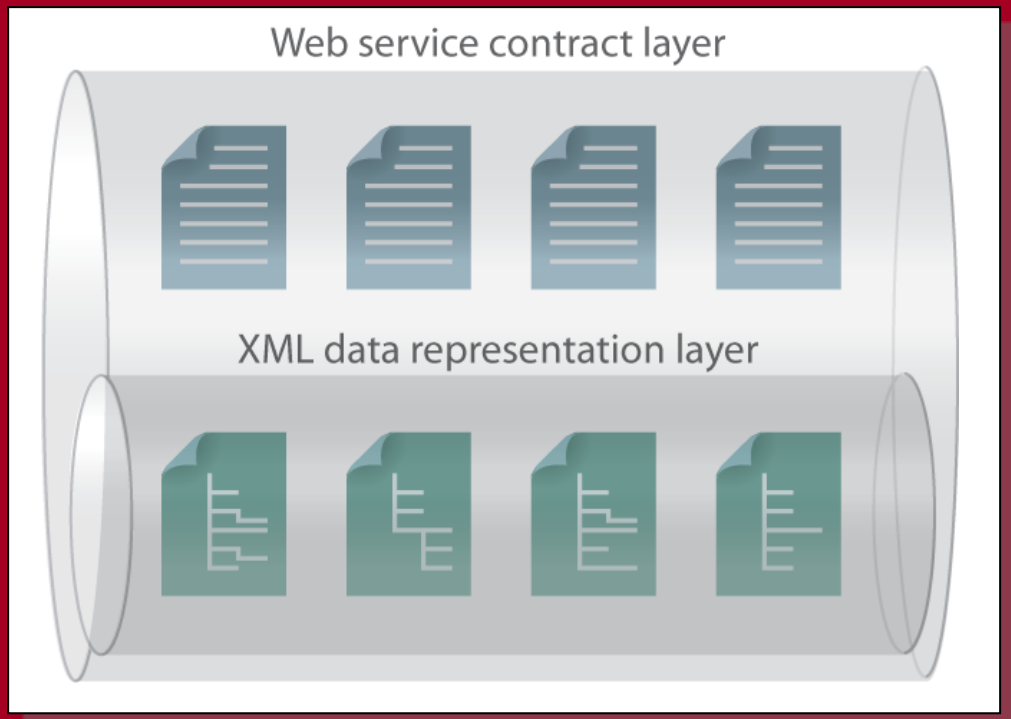
Although its non-proprietary communication made SOAP-based Web services intriguing, this framework was relatively primitive, lacking essential QoS features.



The Web Service Layer

Web services introduce technical contracts that are generally based upon the use of XML schema definitions.

The Web service layer therefore builds upon the XML data representation layer established by XML schemas.



What is a SOAP-based Web Service?

- A SOAP-based Web service is solution logic that has a published WSDL definition and exchanges SOAP-compliant messages.
- The solution logic might be part of a larger system or it may exist as a self-contained component.
- Logic made available as a service does not need to be exclusively accessed via WSDL.

Web Services and Standards Organizations

- Standards organizations are responsible for developing specifications into industry standards.
- The three primary standards organizations influencing the Web services framework are:
  - W3C
  - OASIS
  - WS-I
- These organizations consist of a membership comprised primarily of vendor representatives.

W3C

World Wide Web Consortium (W3C)

- Established 1994
- Approximate membership 400
- Overall goal is to further the evolution of the Web, by providing fundamental standards that improve online business and information sharing.
- Prominent deliverables: XML, XML Schema, XQuery, XML Encryption, XML Signature, XPath, XSLT, WSDL, SOAP, WS-CDL, WS-Addressing

OASIS

Organization for the Advancement of Structured Information Standards (OASIS)

- Established: 1993 as the SGML Open, 1998 as OASIS
- Approximate membership: 600
- Original goal is to promote online commerce via "specialized" Web services standards.
- Prominent deliverables: WS-BPEL, WS-Security, UDDI, ebXML, SAML, others…

WS-I

Web Services Interoperability Organization (WS-I)

- Established: 2002
- Approximate membership: 200
- Overall goal is to foster standardized interoperability using Web services standards.
- Prominent deliverables: Basic Profile 1.2, 2.0, Reliable Security Profile 1.0
- In 2010 the WS-I closed down and transferred all of its deliverables to OASIS

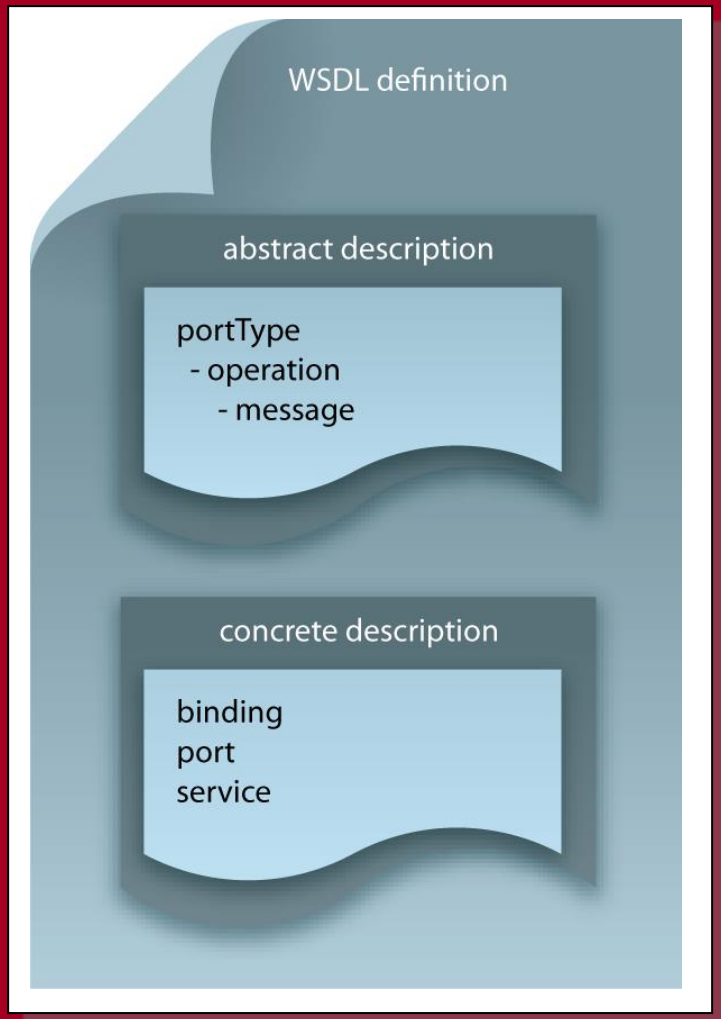
Web Service Contracts, Messaging & Registries

Web Service Contracts

- Service descriptions are required for any Web service to act as a service provider.
- The primary service description documents for Web services are the WSDL definition and XML schema.
- Other service description documents also exist, such as WS-Policy definitions (explained later).
- Collectively, service description documents form a service contract (or technical service contract).

Web Service Contracts

- A WSDL definition is an XML document.
- A WSDL definition consists of abstract and concrete descriptions.
- This separation allows for an implementation-neutral interface definition.



Web Service Contracts

A WSDL document defines a series of operations, each of which represents a function the Web service is capable of performing. Each operation is defined through a set of input and/or output messages. The existence and sequence of these messages determines the operation's message exchange patterns.

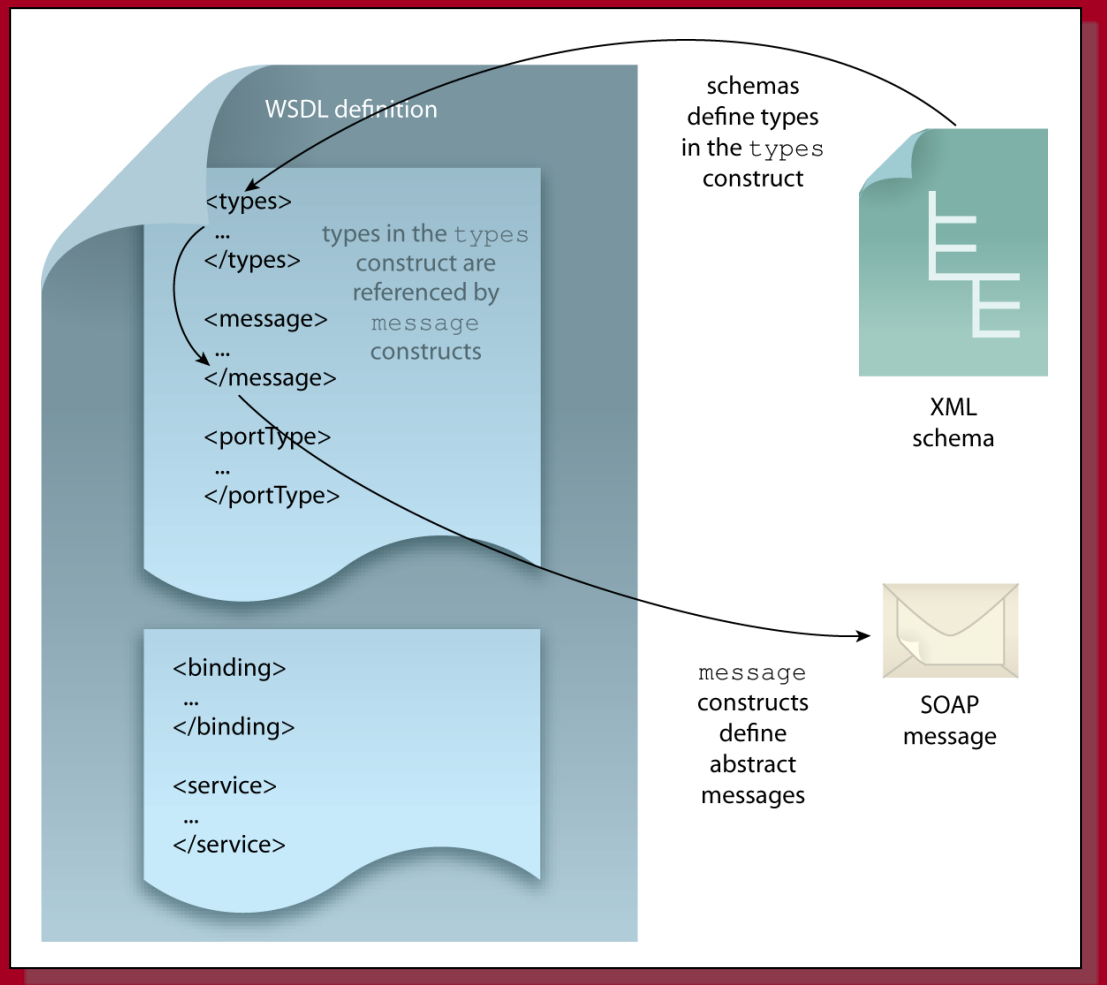
When comparing a Web service to a component:

- An operation is comparable to a component method.
- Input and output messages are comparable to input and output method parameters.
- The WSDL definition is comparable to the component's class interface.

Web Service Contracts

WSDL definitions rely on XML schemas to define the types used by input and output messages.

Schemas can be embedded within or linked to the WSDL document.



Web Service Contracts

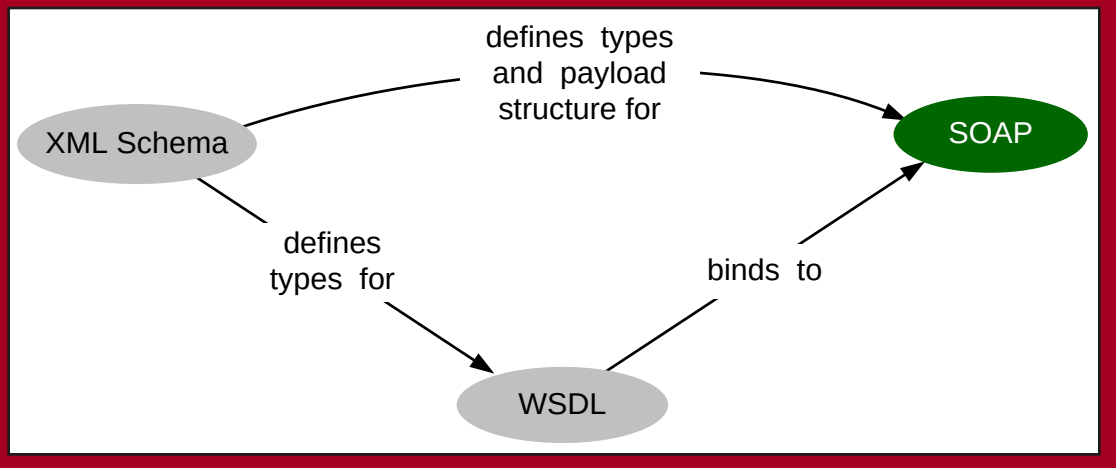
- Entity-centric XML schemas can be designed to represent specific information sets that correspond to business entity models.
- Entity schemas are generally used together with services based on the entity service model.
- Ideally entity schemas are standardized separately from Web service contracts so that they establish an independent data architecture.

Web Service Messaging

- Web services rely solely on messages as a means of exchanging information.
- This is a departure from traditional components that used binary RPC communication.
- Messaging-based communication brings with it increased flexibility but also performance and reliability challenges.
- The foremost standard for messages exchanged by Web services is SOAP.

Web Service Messaging

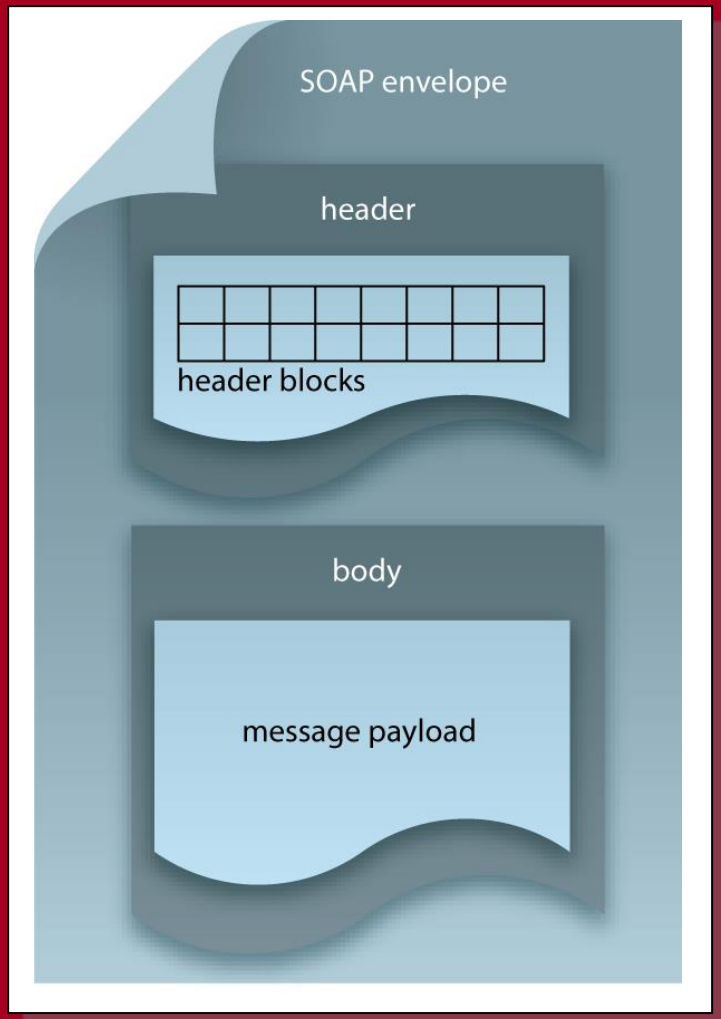
- A SOAP message is an XML document that essentially acts as a standardized container designed to transport XML data.
- An abstract WSDL definition is bound to the SOAP protocol via its concrete definition.
- The structure of message content is defined by XML schema definitions.



Web Service Messaging

A SOAP message is structured as an envelope document that holds an optional header and a required body section.

The data being transported by the SOAP message resides in the body section and is often referred to as the message payload.

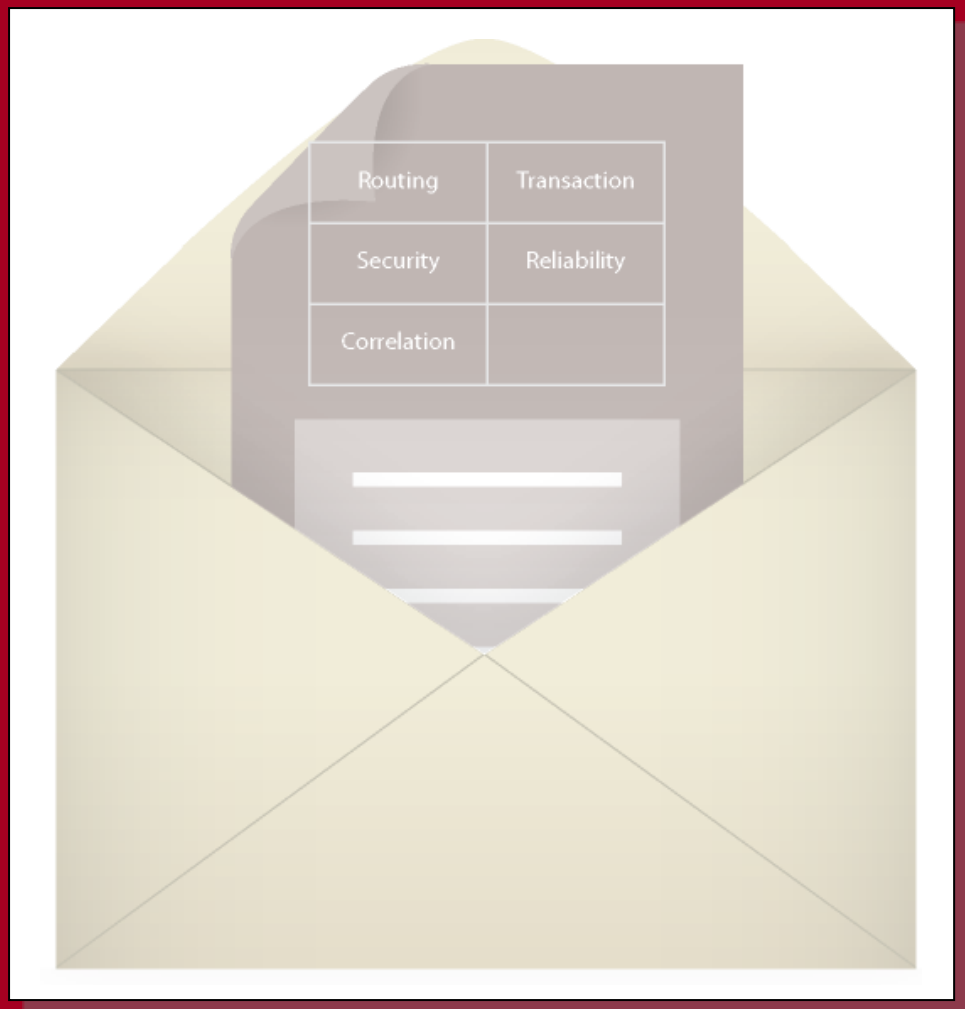


Web Service Messaging

- The header section can be used to hold groups of meta information called header blocks or SOAP headers.
- A classic SOAP header is a correlation identifier.
- Other types of metadata placed in header blocks can pertain to transactions, security, routing, etc.

Web Service Messaging

- SOAP headers are a cornerstone part of the Web services world.
- Almost all WS-* specifications are implemented via SOAP headers.



Exercises

Exercise 2.7

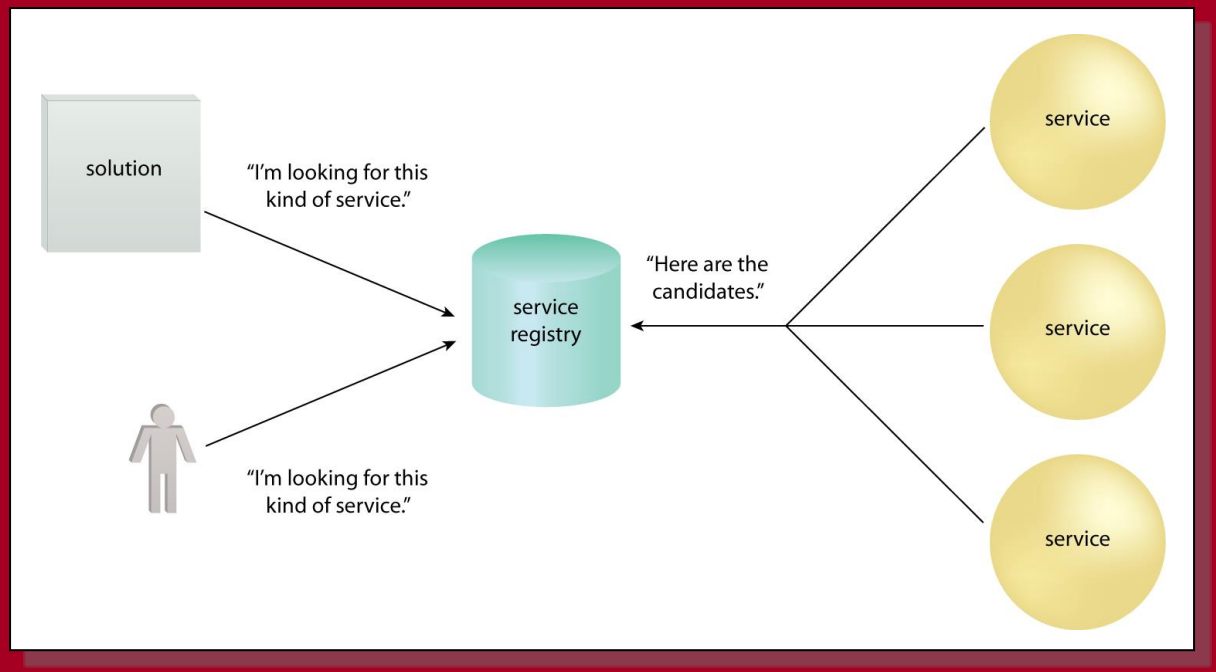
Define Headers

Explain Intermediary Header Processing

Service Registries

Public registries allow service provider owners to register their services for discovery by other organizations.

Private registries allow an organization to establish a central service repository to support discovery within the enterprise.



Web Services and UDDI

- UDDI (Universal Description, Discovery, and Integration) is a well-known registry model standard.
- UDDI was originally positioned as a core Web services technology but has since lost its prominence.
- Organizations have sought alternative directory technologies to create service registries but interest in UDDI has recently increased.
- UDDI records are XML documents that essentially contain discoverable meta information about services.

Web Service Anatomy

Web Service Anatomy

- From an implementation perspective, a Web service consists of different physical parts that collectively fulfill the processing requirements associated with its role.
- These processing tasks can be separated into two categories:
  - message processing logic
  - core service logic

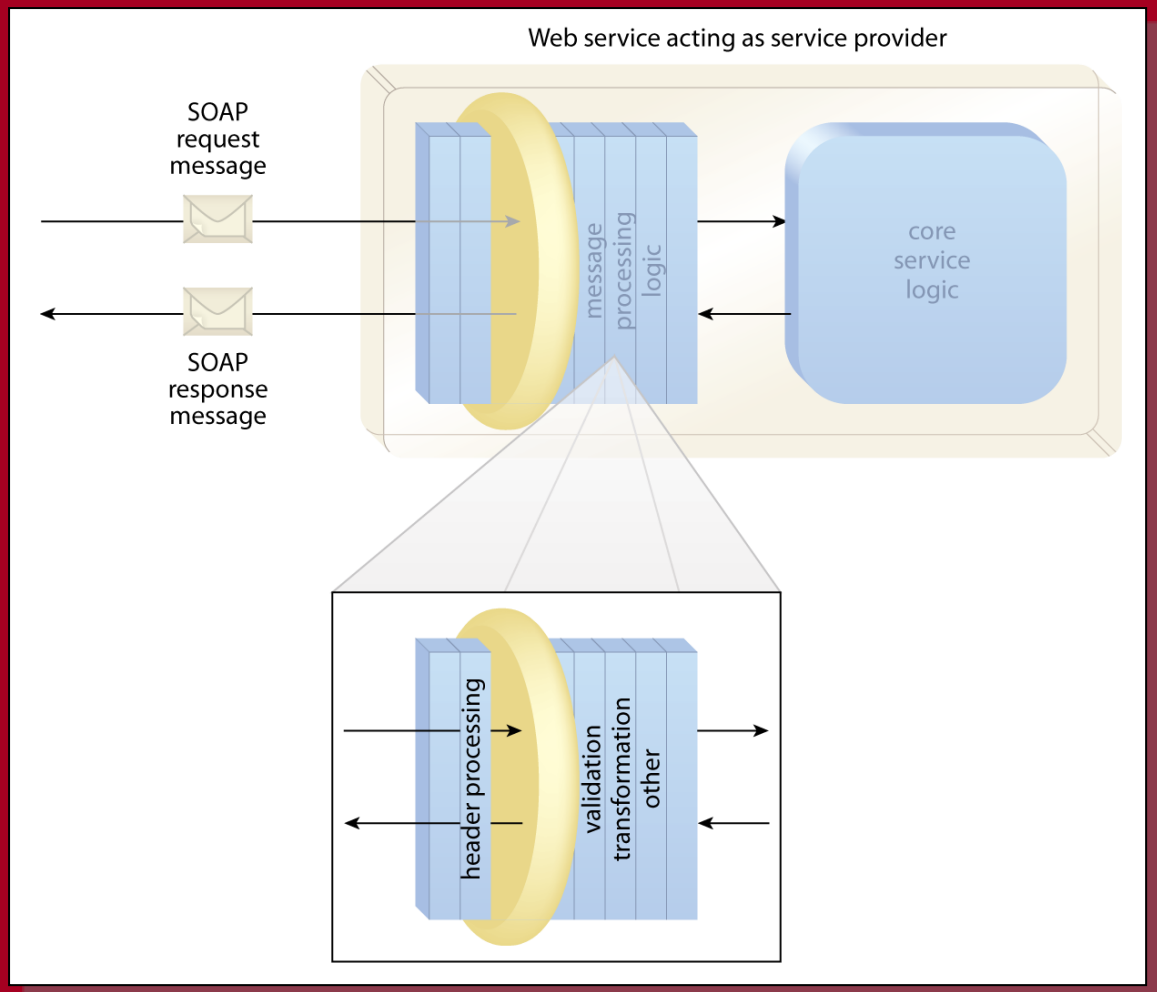
Web Service Anatomy

- Message processing logic represents a combination of runtime system services, service agents, and service logic related to the processing of the SOAP message and the WSDL definition.
- Core service logic is the back-end part of a Web service that carries out whatever task is associated with the operations expressed in the Web service contract.

Web Service Anatomy

A physical representation of a Web service acting as a service provider.

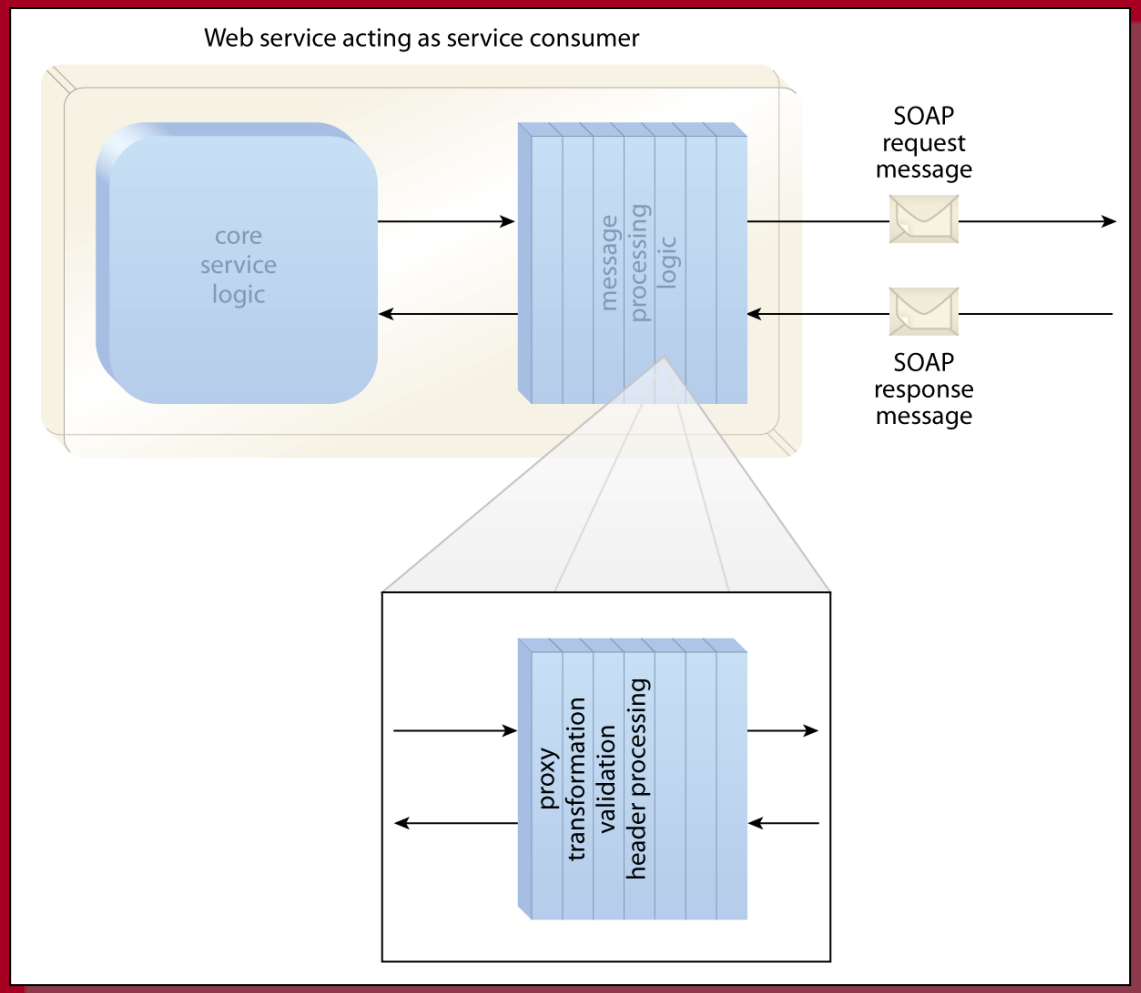
Note the layers of message processing logic.



Web Service Anatomy

A physical representation of a Web service acting as a service consumer.

Note that "proxy" refers to a service proxy.



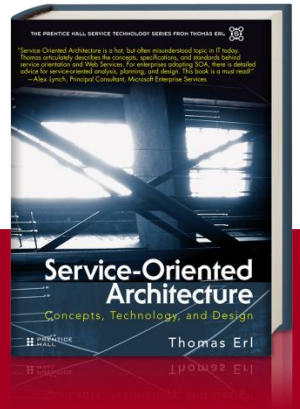
Exercises

Exercise 2.8

Identify Physical Service Roles

Identify Intermediaries

Reading

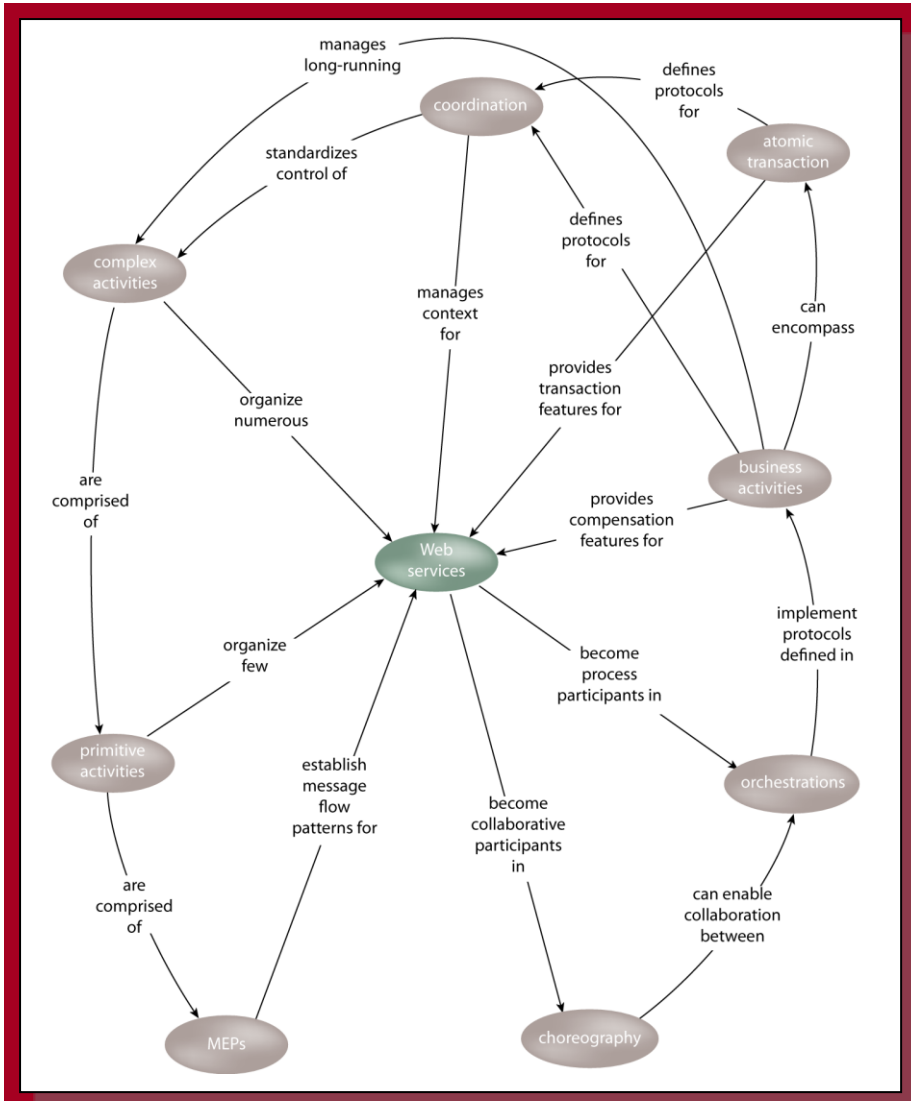


"Service-Oriented Architecture: Concepts, Technology & Design"

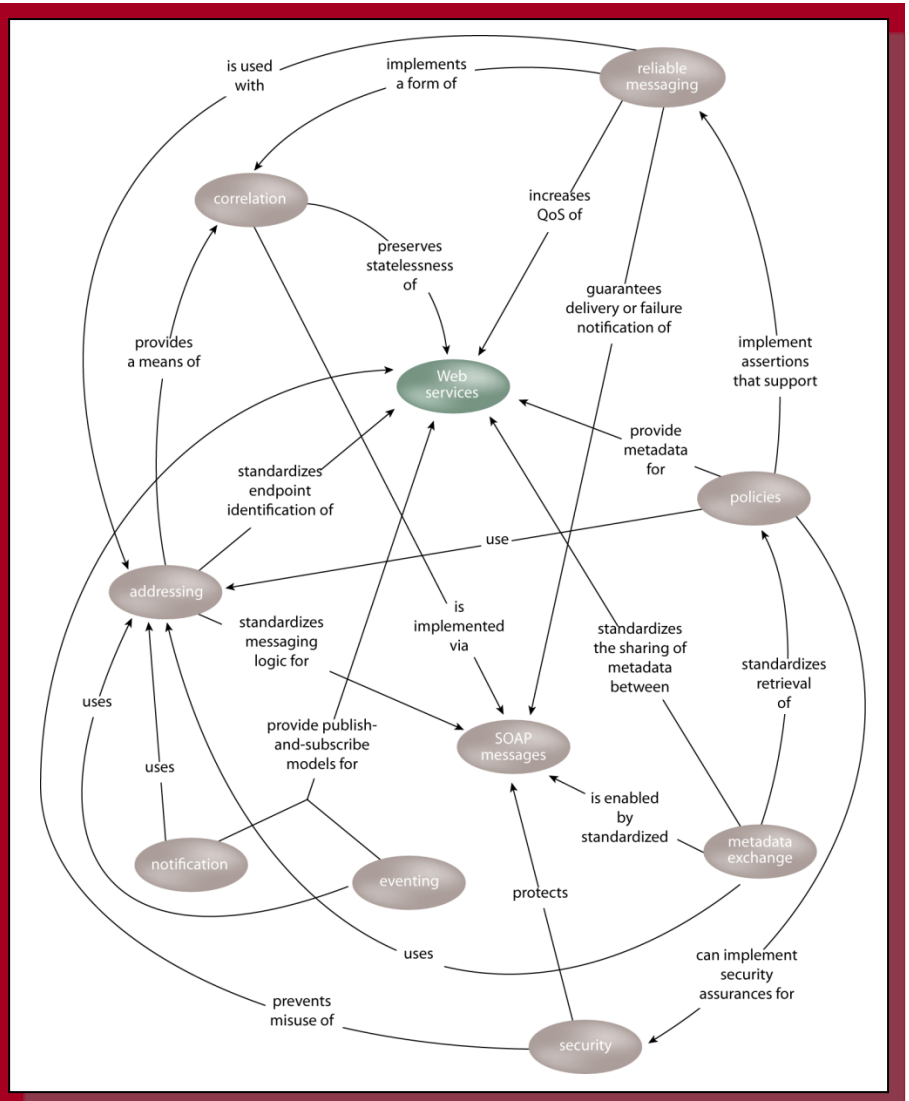
Chapter 2,
Case Study Background (pages 22-28)

WS-* Technologies

Introduction to WS-*



SOA-CTD Figure 6.1



SOA-CTD Figure 7.1

Introduction to WS-*

- The Web services framework is ever-growing due to the concepts and technology introduced by many additional Web service technology specifications.
- These specifications represent the second-generation Web services platform and are collectively referred to as WS-* (because most of their names are prefixed with "WS-").
- Though the Web services framework is vendor neutral it remains very much a vendor-driven initiative.

Introduction to WS-*

- Message Exchange Patterns
- Service Activities
- Coordination (WS-Coordination)
- Transactions (WS-AtomicTransaction & WS-BusinessActivity)
- Orchestration (WS-BPEL)
- Addressing (WS-Addressing)
- Reliable Messaging (WS-RM)
- Policies (WS-Policy)
- Security (WS-Security)

Message Exchange Patterns

Message Exchange Patterns

- Every task automated by a Web service can differ in:
  - The nature of the logic being executed
  - The role played by the service in the overall execution of the business task.
- Regardless of how complex a task is, almost all require the transmission of multiple messages.

Message Exchange Patterns

- Message exchange patterns (MEPs) represent a set of templates that provide already mapped out sequences for the exchange of messages.
- There are two types of message exchange patterns:
  - Primitive MEPs that govern simple message exchanges.
  - Complex MEPs which are generally comprised of multiple primitive MEPs.

Message Exchange Patterns

- The most popular primitive MEP is the request-response pattern.
- This synchronous pattern requires that upon delivery of a message from consumer to provider, the provider responds with a message to the consumer.
- Another common primitive MEP is the one-way pattern where a single message is sent from consumer to provider without a response.
- Both request-response and one-way are considered inbound MEPs.

Message Exchange Patterns

- Version 1.1 of the WSDL specification supports two additional MEPs (solicit-response, notification) that are considered outbound MEPs.
- Outbound MEPs were intended to provide call-back functionality, but they were not adopted by the industry and are not recommended by the WS-I.
- WSDL 2.0 adds one new MEP and does not support outbound MEPs.

Service Activities

Service Activities

- In service-oriented solutions, the automation of a given task can involve any number of services.
- This typically results in the formation of a service composition.
- The interaction of a group of composed services working together to complete a task can be referred to as a service activity.

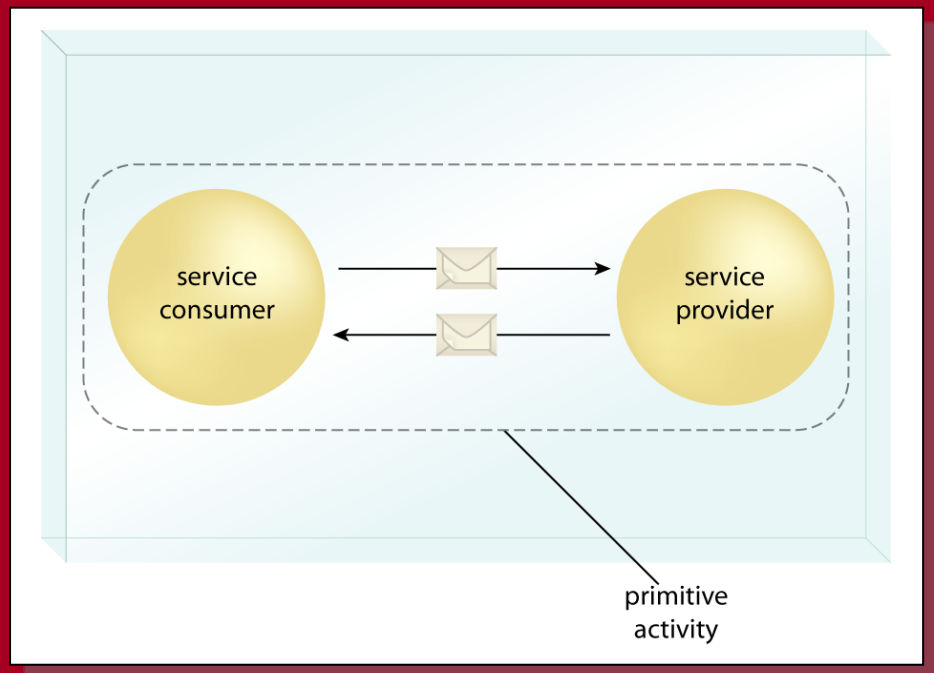
Service Activities

- Whereas MEPs are design-time patterns, service activities represent runtime execution scenarios.
- A service activity can be considered an instance of an MEP.
- As with MEPs, there are primitive and complex variations of service activities.

Service Activities

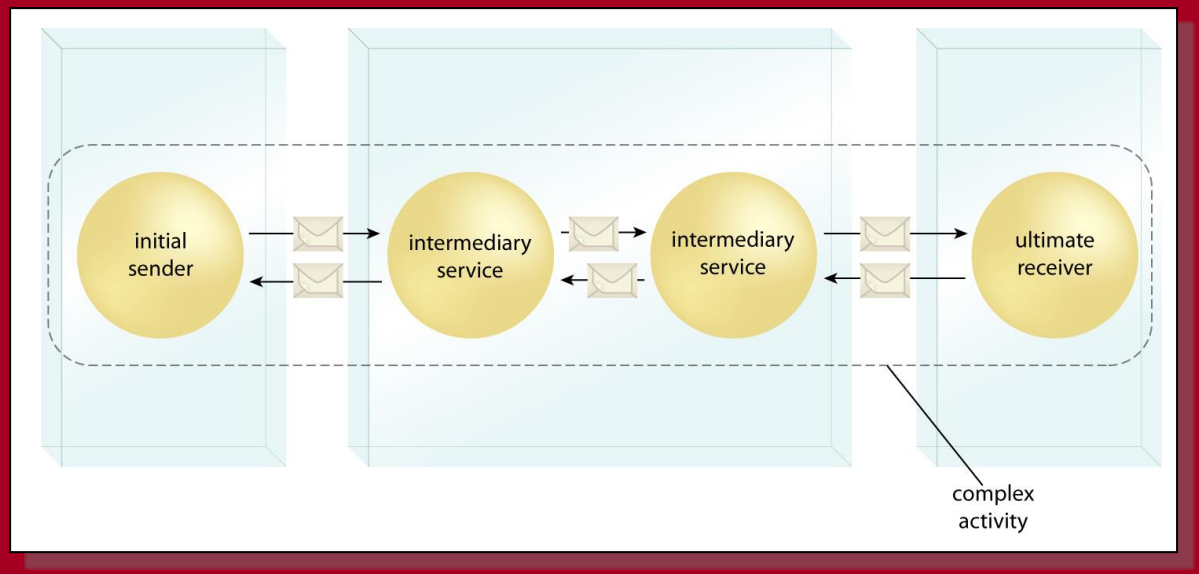
A primitive activity is typified by a consumer and provider exchanging information (as per a standard MEP).

An example of a task with a scope limited to a primitive service activity is a point-to-point data exchange.



Service Activities

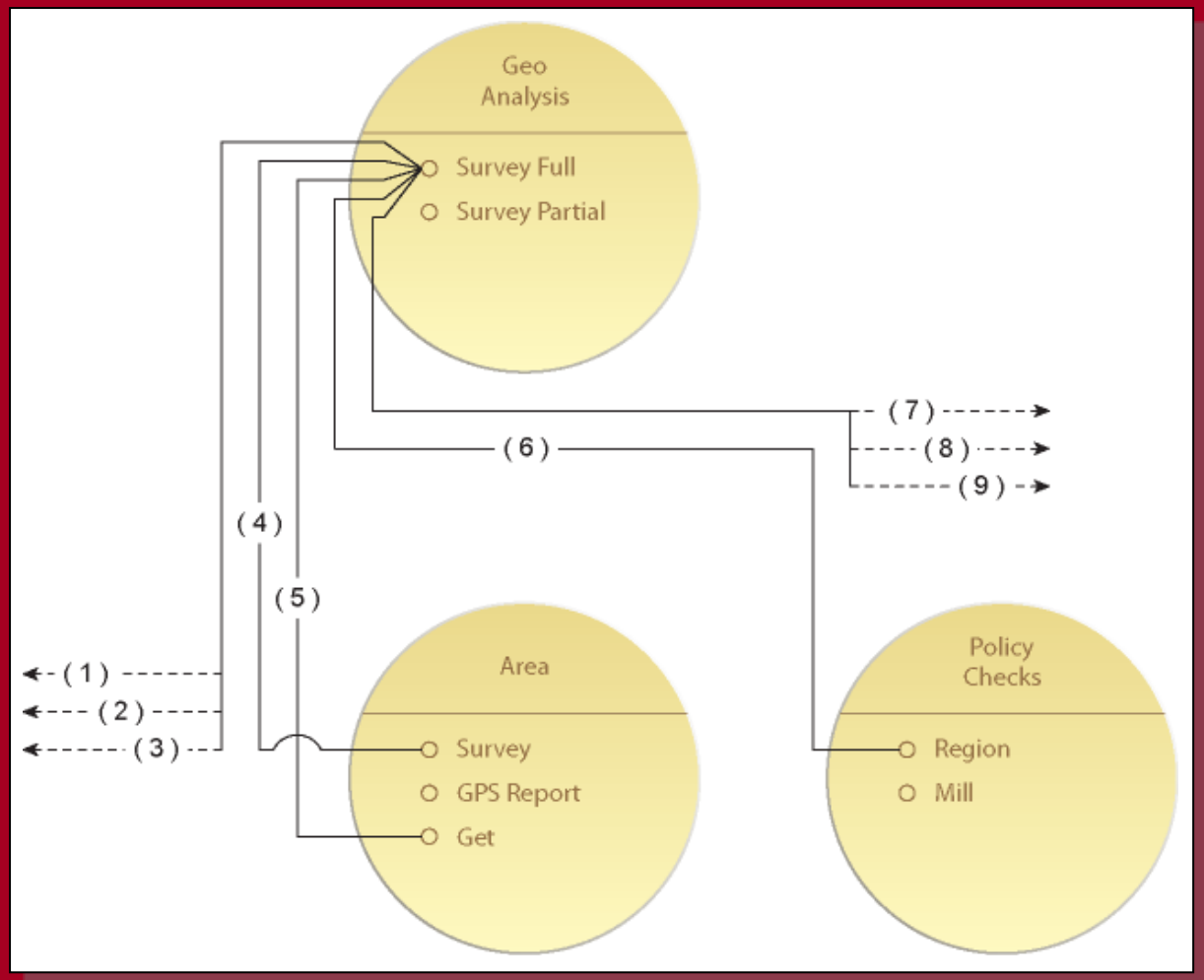
Complex activities involve more than two services (and MEPs) as part of a service composition.



These more elaborate types of activities often require the use of WS-* extensions, especially those associated with activity management.

Service Activities

Complex activities are commonplace in service-oriented solutions and rely on the ability of Web services to act as effective composition members.



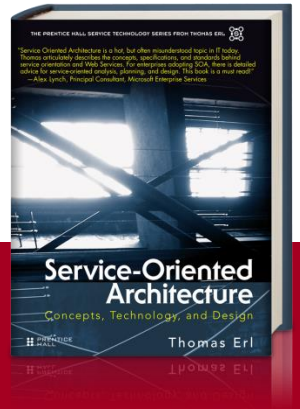
Service Activities

Note

The underlying solution logic of each Web service, whether it consists of a single component or an entire legacy system, is generally not mapped as part of a service activity.

Service activities represent service interaction only.

Reading



"Service-Oriented Architecture: Concepts, Technology & Design"

Chapter 5,
Case Study Example (pages 125-126)

Chapter 6,
Case Study Examples (pages 164, 165-166, 175-176)

Coordination (WS-Coordination)

Coordination (WS-Coordination)

- Every service activity introduces a measure of activity-specific information into the runtime environment.
- This information is specific to the context of an activity and is therefore classified as context information.
- The more complex an activity, the more context information it tends to bring with it.

Coordination (WS-Coordination)

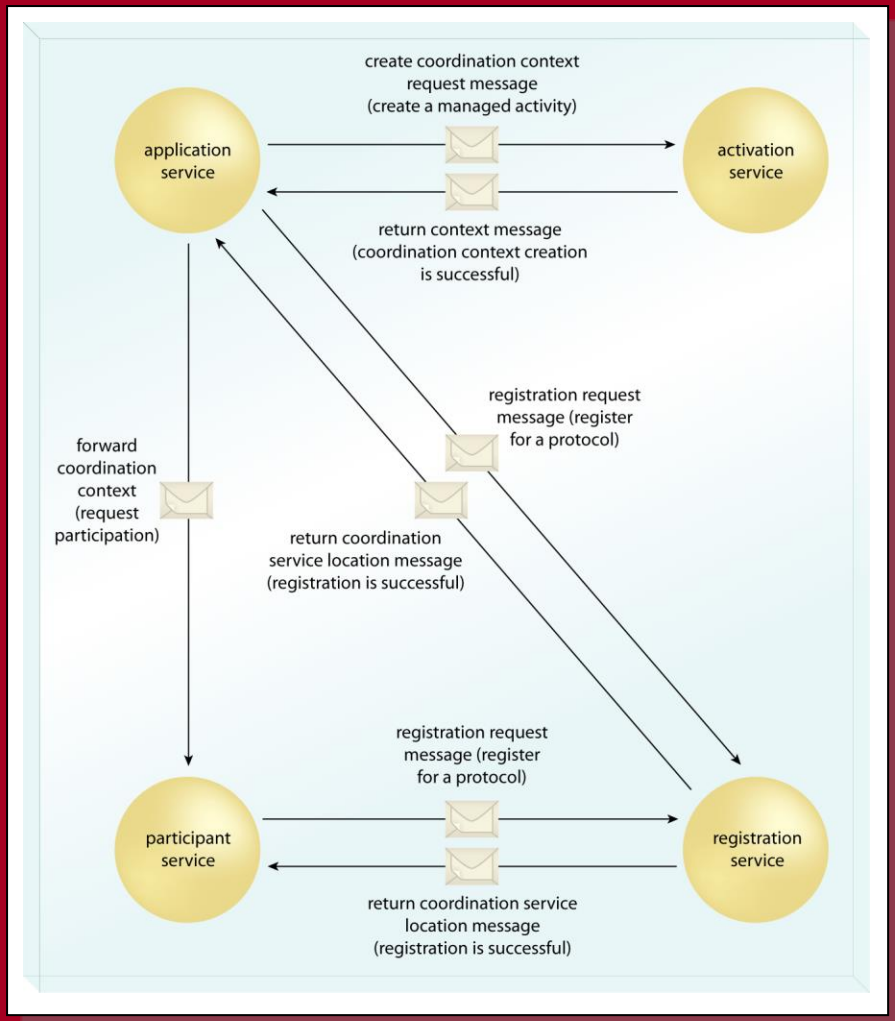
Examples of context information:

- the number of services participating in the activity
- the duration of the activity
- the number of activity instances that concurrently exist
- rules associated with the activity

Coordination (WS-Coordination)

- WS-Coordination is a specification that introduces a generic coordinator service model.
- The coordinator service controls a composition of three other system services that each play a part in the management of context data.
- Collectively, these services carry out an industry-standard means of managing a service activity.
- Context information is represented by standardized SOAP headers.

Coordination (WS-Coordination)

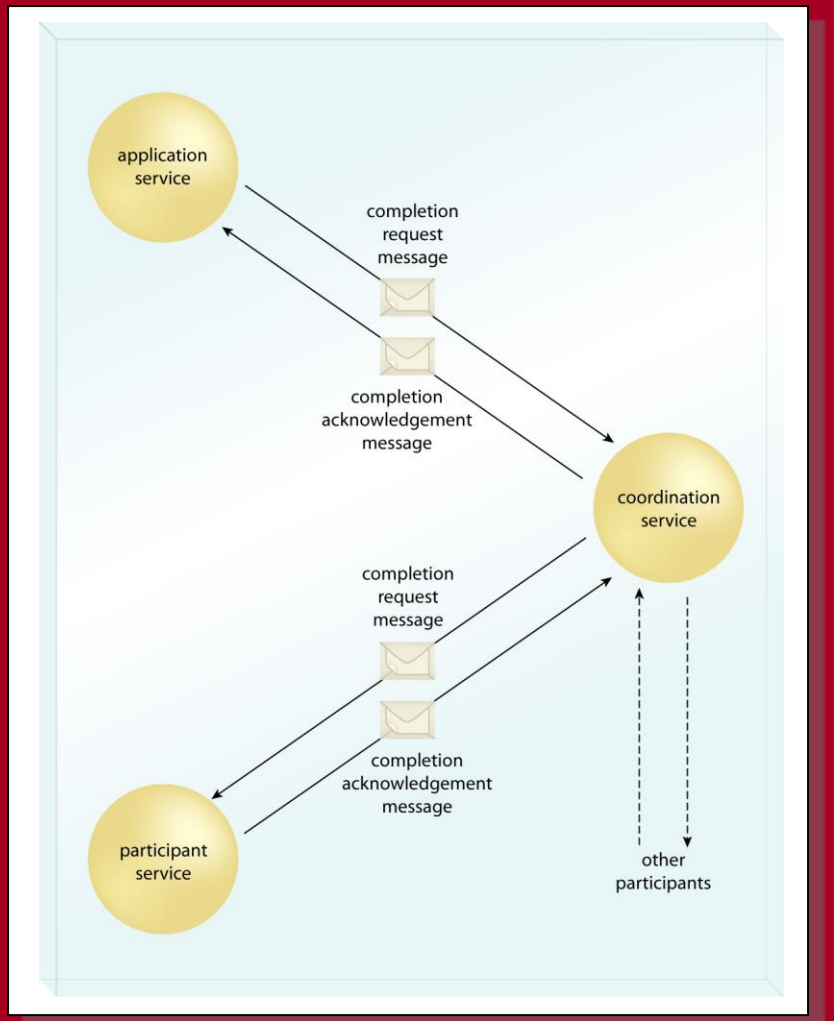


The activation service supplies a coordination context to the application service, which shares it with other services as a means of inviting them to participate in the coordination.

All participants register with the registration service which provides the address of the coordination service (not shown).

Coordination (WS-Coordination)

A coordination service manages the interaction between the application service and participant services.



Transactions (WS-AtomicTransaction & WS-BusinessActivity)

Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

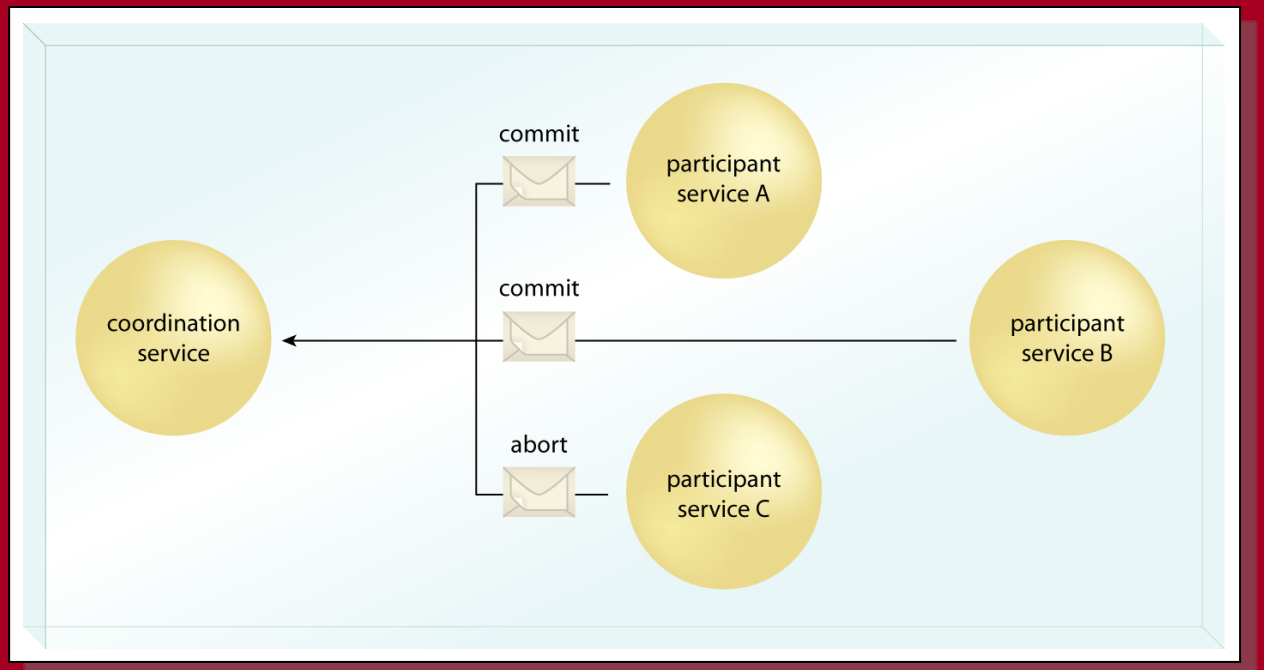
- In the past, transaction capability was limited to logic within Web service boundaries or relied on proprietary features.
- However, the need to support common complex compositions requires a means of propagating a transaction across Web service boundaries.
- For this reason, the WS-AtomicTransaction and WS-BusinessActivity standards were developed (both of which were contributed to the WS-TX technical committee).

Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

- The WS-AtomicTransaction specification provides industry-standard protocols (rules) that emulate ACID transaction functionality.
- The protocols are implemented via the WS-Coordination context management framework.
- Services registered for a transaction participate by voting on the outcome in order to determine whether to commit or abort (rollback) changes.
- If any one service within the transaction votes to abort or if a vote from a participating service is not received, the changes are rolled back.

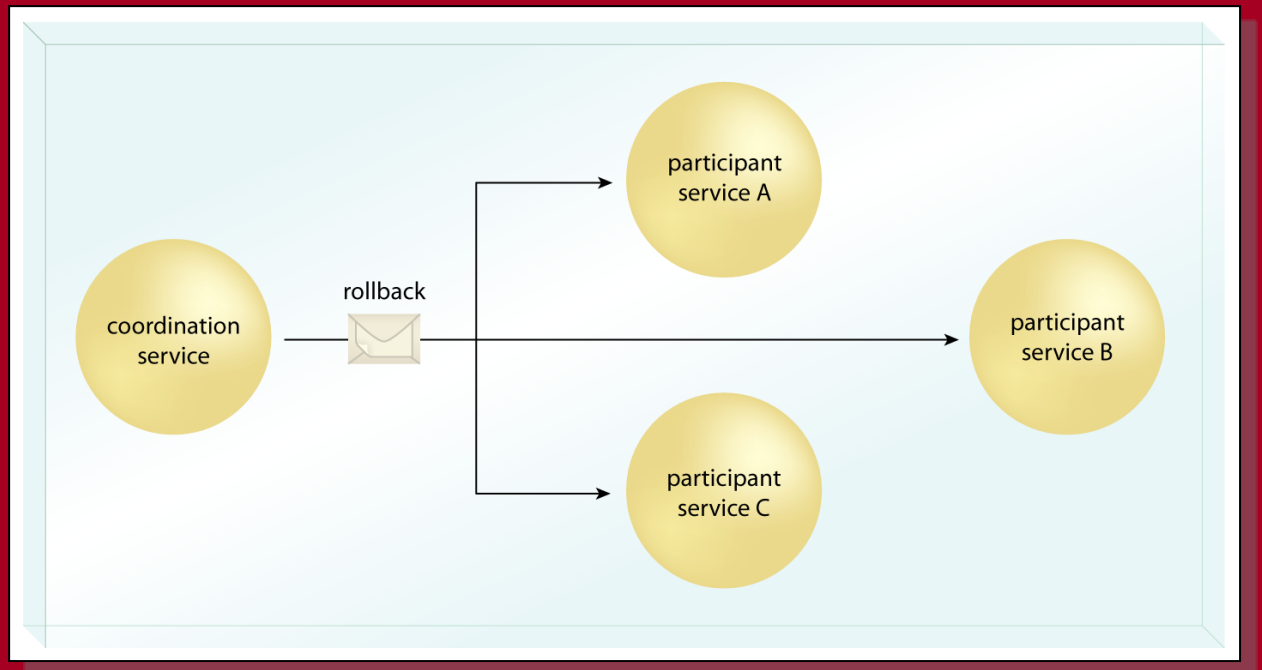
Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

The transaction participants vote on the outcome of the atomic transaction, but one participant votes to abort.



Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

Because one abort vote was received, the coordinator service aborts the transaction and notifies all participants to rollback changes.

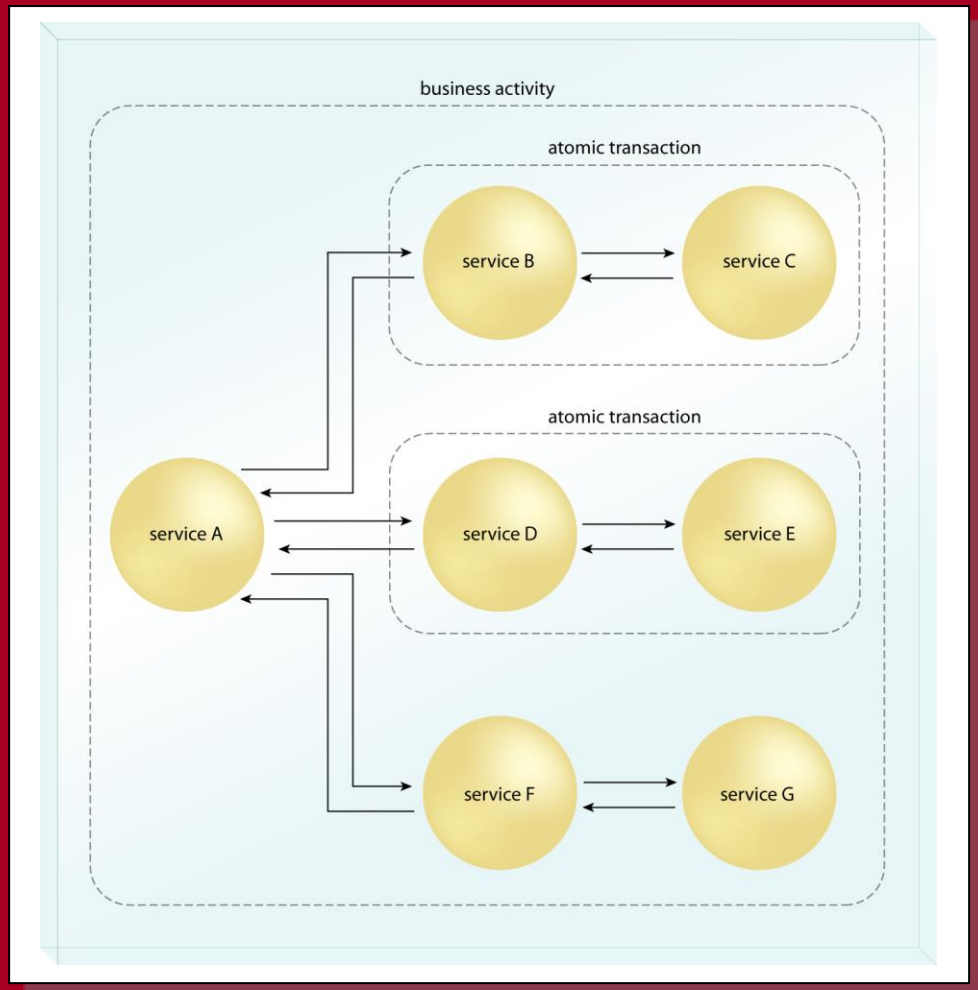


Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

- WS-BusinessActivity supplies protocols in support of long-running complex activities, also based on the WS-Coordination framework.
- Long-running business activities can span minutes, hours, or even days and can involve manual steps completed by humans.
- Failed transactions are not rolled back; instead they invoke a separate compensation process.

Transactions
(WS-AtomicTransaction & WS-BusinessActivity)

A service activity managed by WS-BusinessActivity can encapsulate one or more atomic transactions.



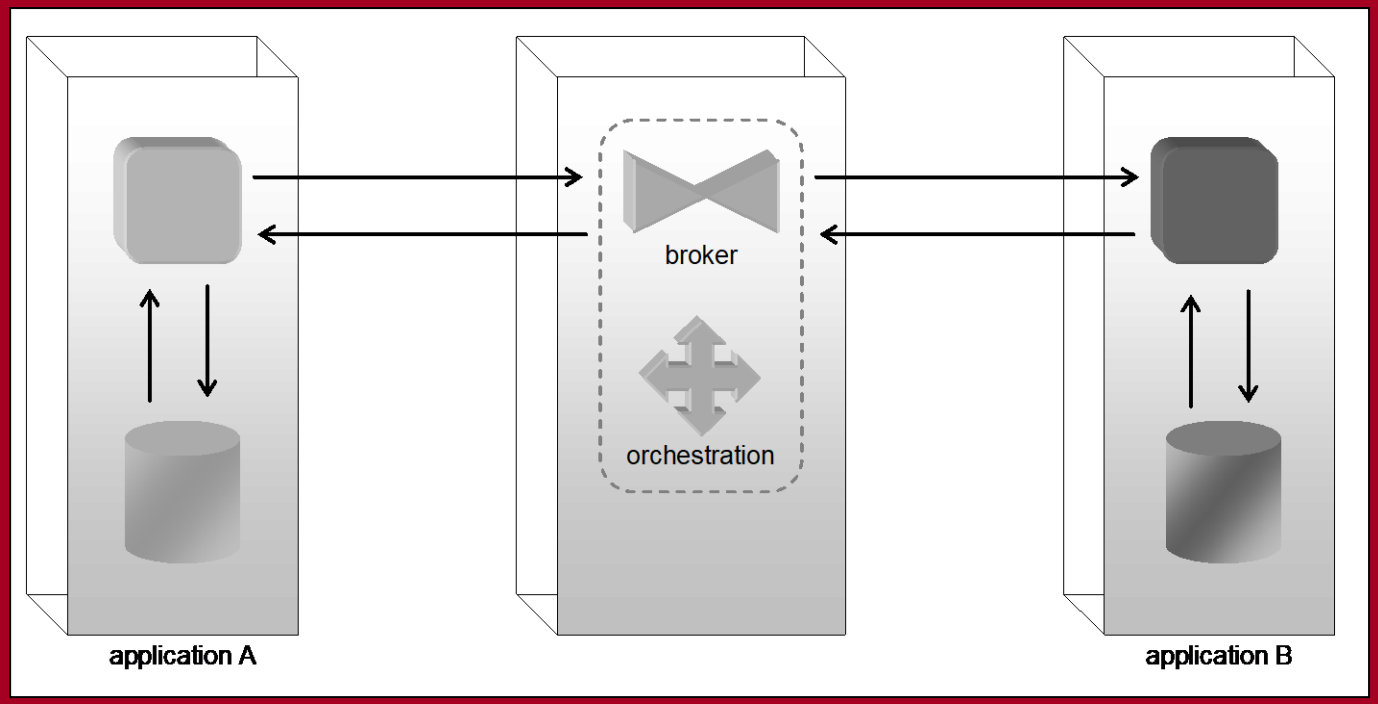
Orchestration (WS-BPEL)

Orchestration (WS-BPEL)

- Orchestration emerged from the enterprise application integration (EAI) era as a prominent form of middleware.
- Its primary role was to facilitate connectivity between disparate systems by establishing a hub in which business process logic is centrally defined and executed.
- Within the hub, brokerage services (such as data transformation) would also be carried out to overcome system incompatibilities.

Orchestration (WS-BPEL)

A hub-and-spoke architecture in which the orchestration platform establishes a hub that becomes a primary contact point for disparate applications.



Orchestration (WS-BPEL)

- The ability of orchestration platforms to abstract and centralize business process logic can benefit SOA.
- A parent business process layer can be established as a location for all non-agnostic service logic.
- This supports the creation of a service inventory with a high percentage of agnostic services.

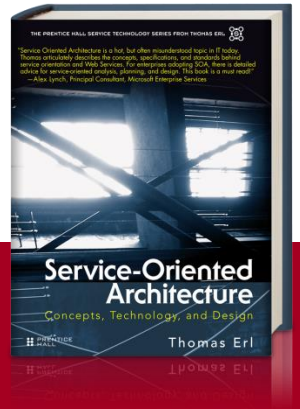
Orchestration (WS-BPEL)

- The Web Services Business Process Execution Language (WS-BPEL) provides a means of expressing business process logic in an industry-standard manner.
- WS-BPEL is essentially a Web service composition language.
- Its syntax allows for the invocation and interaction of Web services responsible for carrying out the business process logic.

Orchestration (WS-BPEL)

- WS-BPEL process definitions can also be encapsulated by and exposed via Web services.
- These services form the basis of the orchestrated task service model (also known as the process service model).
- WS-BPEL process definitions are typically defined using a front-end tool and then executed via an orchestration engine.
- WS-BPEL was formerly called BPEL4WS and has strong industry support.

Reading



"Service-Oriented Architecture: Concepts, Technology & Design"

Chapter 6, Case Study Examples
(pages 184-185, 192, 198-199, 206-207)

Addressing (WS-Addressing)

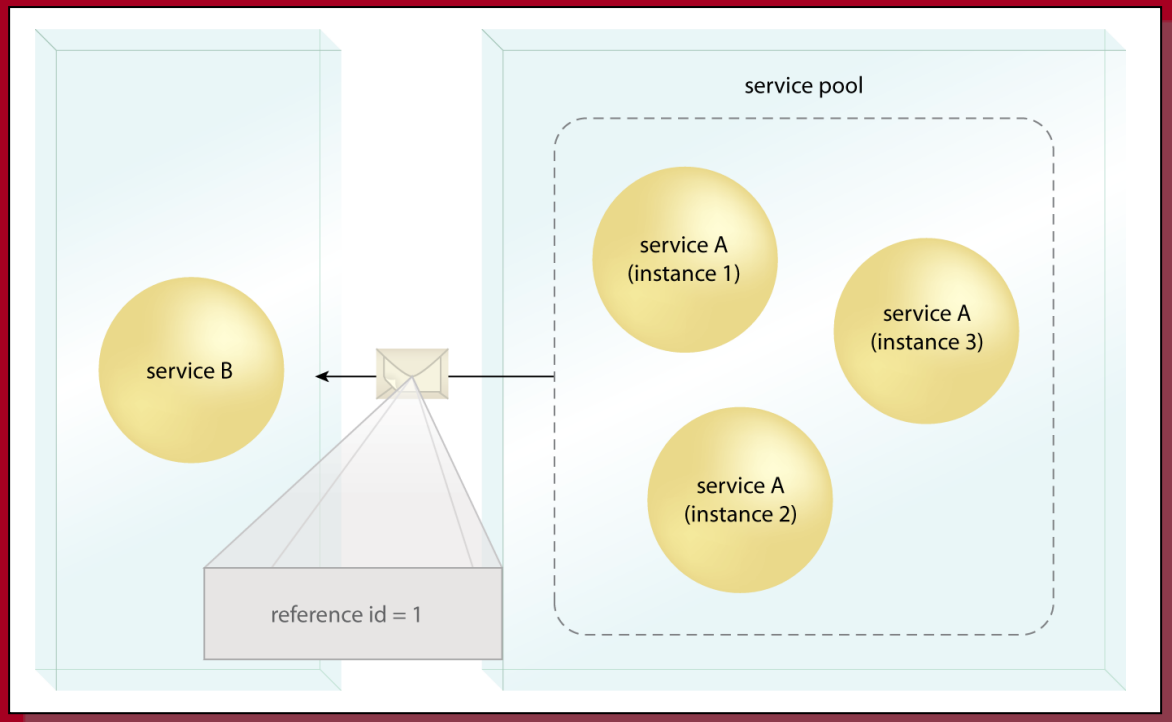
Addressing (WS-Addressing)

- The WS-Addressing specification provides industry standard message routing features via a set of pre-defined SOAP headers.
- There are two types of SOAP headers introduced by WS-Addressing:
  - Endpoint References (EPRs)
  - Messaging Information (MI) Headers
- Note: MI Headers have recently been renamed to Message Addressing Properties (MAP).

Addressing (WS-Addressing)

Endpoint References (EPRs) allow you to define further details pertaining to a Web service contract (endpoint).

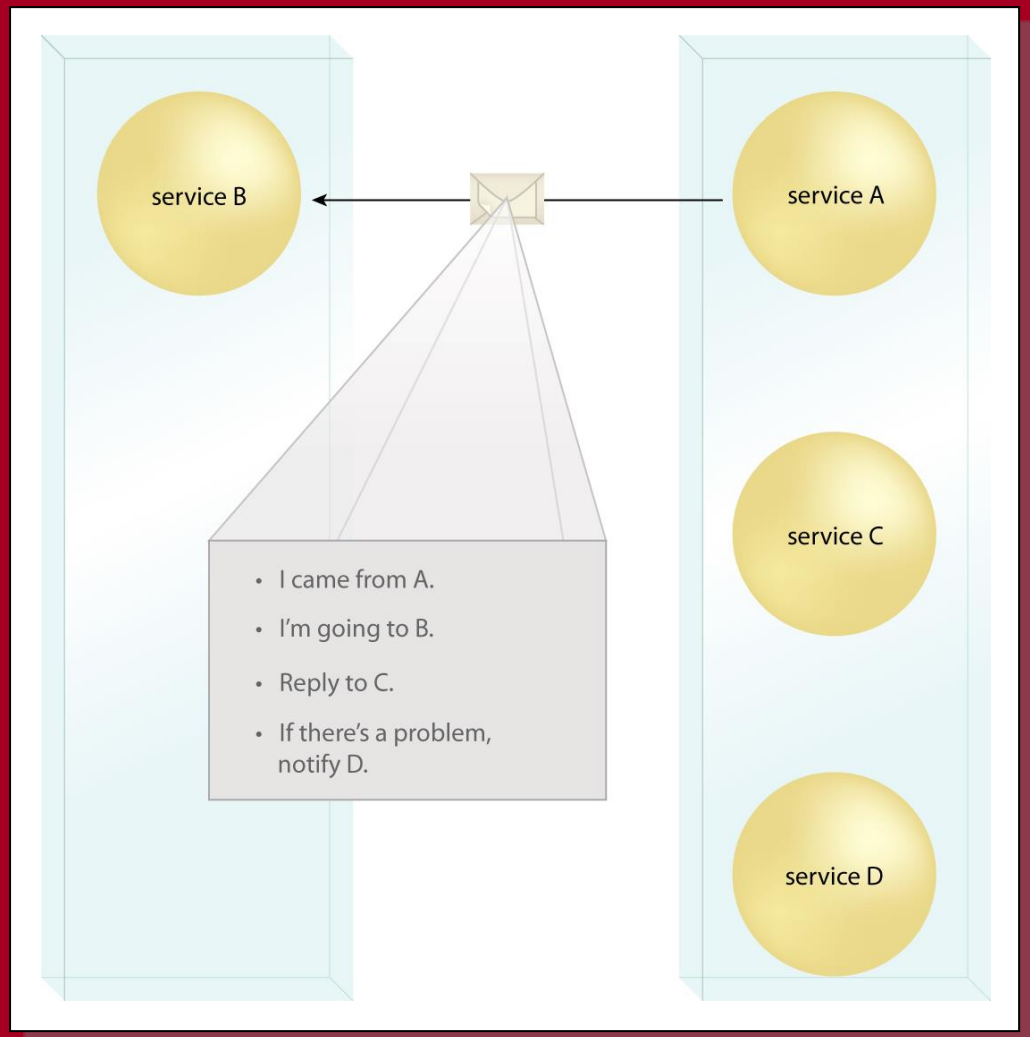
A common use of EPRs is to target messages to service instances through the use of special identifiers.



Addressing (WS-Addressing)

MI Headers are used to equip messages with self-governing routing and processing details (which can include EPRs).

What MI Headers bring to SOAP messaging is commonly compared to what a waybill brings to the shipping process.



Reliable Messaging (WS-RM)

Reliable Messaging (WS-RM)

WS-ReliableMessaging (WS-RM) establishes a framework that provides an industry-standard means of communicating:

- whether a message successfully arrived at its intended destination
- whether a message failed to arrive and therefore requires a re-transmission
- whether a series of messages arrived in the sequence they were intended to

Reliable Messaging (WS-RM)

There are two specifications that implement reliable messaging:

- WS-ReliableMessaging
- WS-Reliability

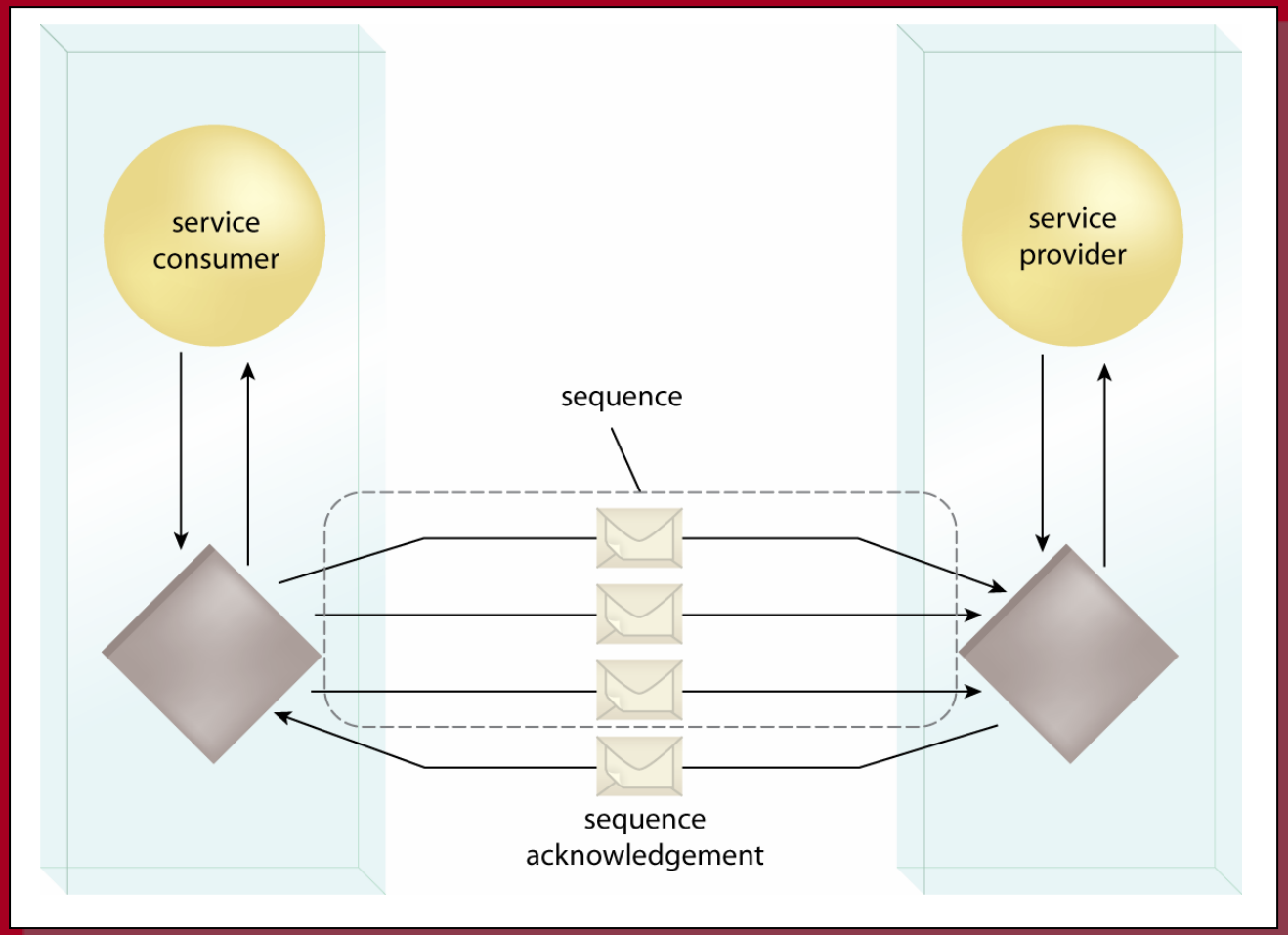
- Their feature-sets overlap and they are considered competing specifications.
- Both formed the basis for the subsequent WS-RX technical committee.

Reliable Messaging (WS-RM)

- Reliable messaging groups messages in a sequence.
- Positive or negative acknowledgement notifications are sent by the recipient of the sequence, depending on whether the messages were delivered as expected.
- Acknowledgements can also be delivered prior to a sequence being completed.

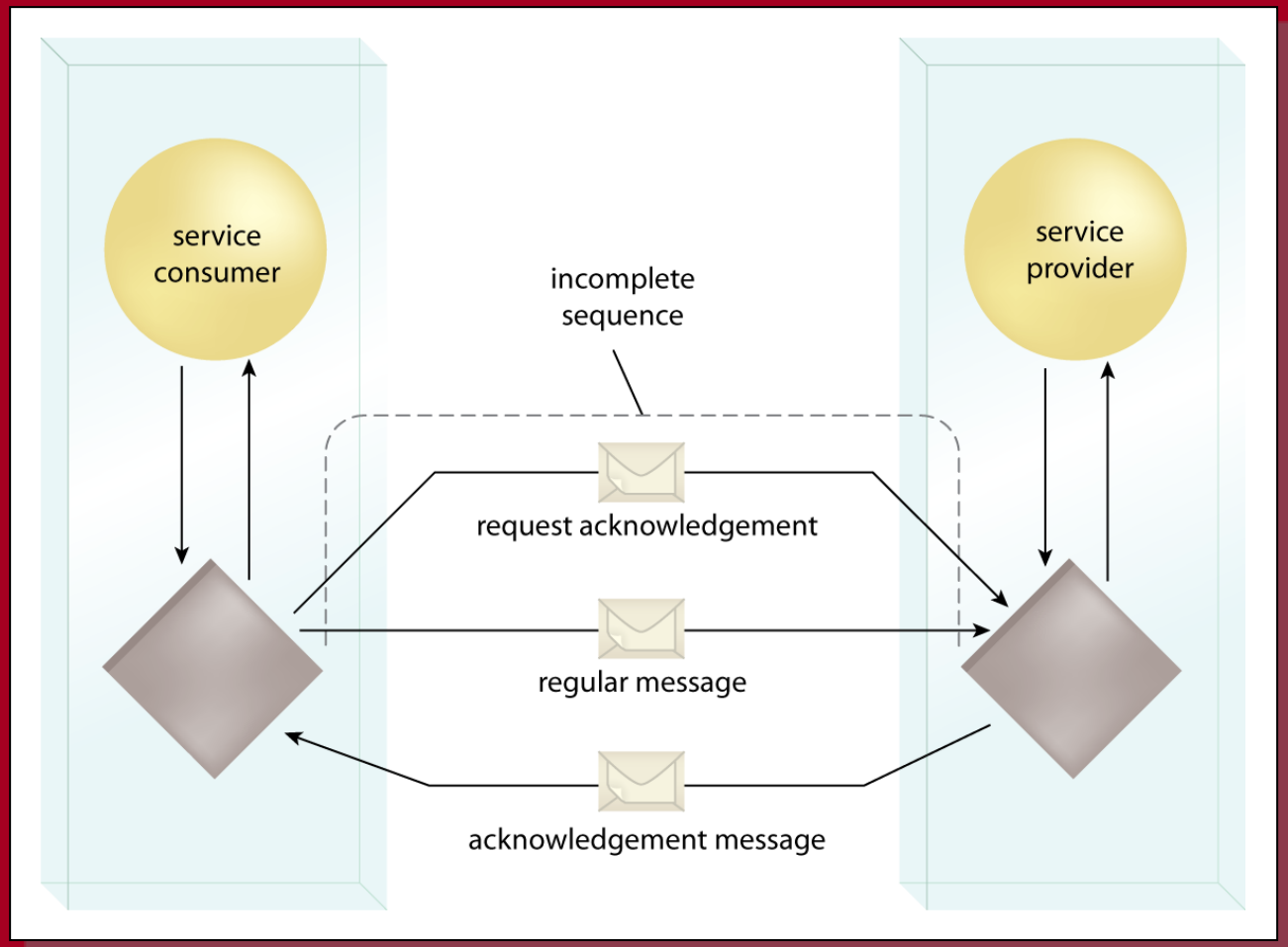
Reliable Messaging (WS-RM)

A sequence acknow-ledgement sent after the successful delivery of a sequence of messages.



Reliable Messaging (WS-RM)

A negative acknow-ledgement (bottom message) sent to indicate a failed delivery prior to the completion of the sequence.



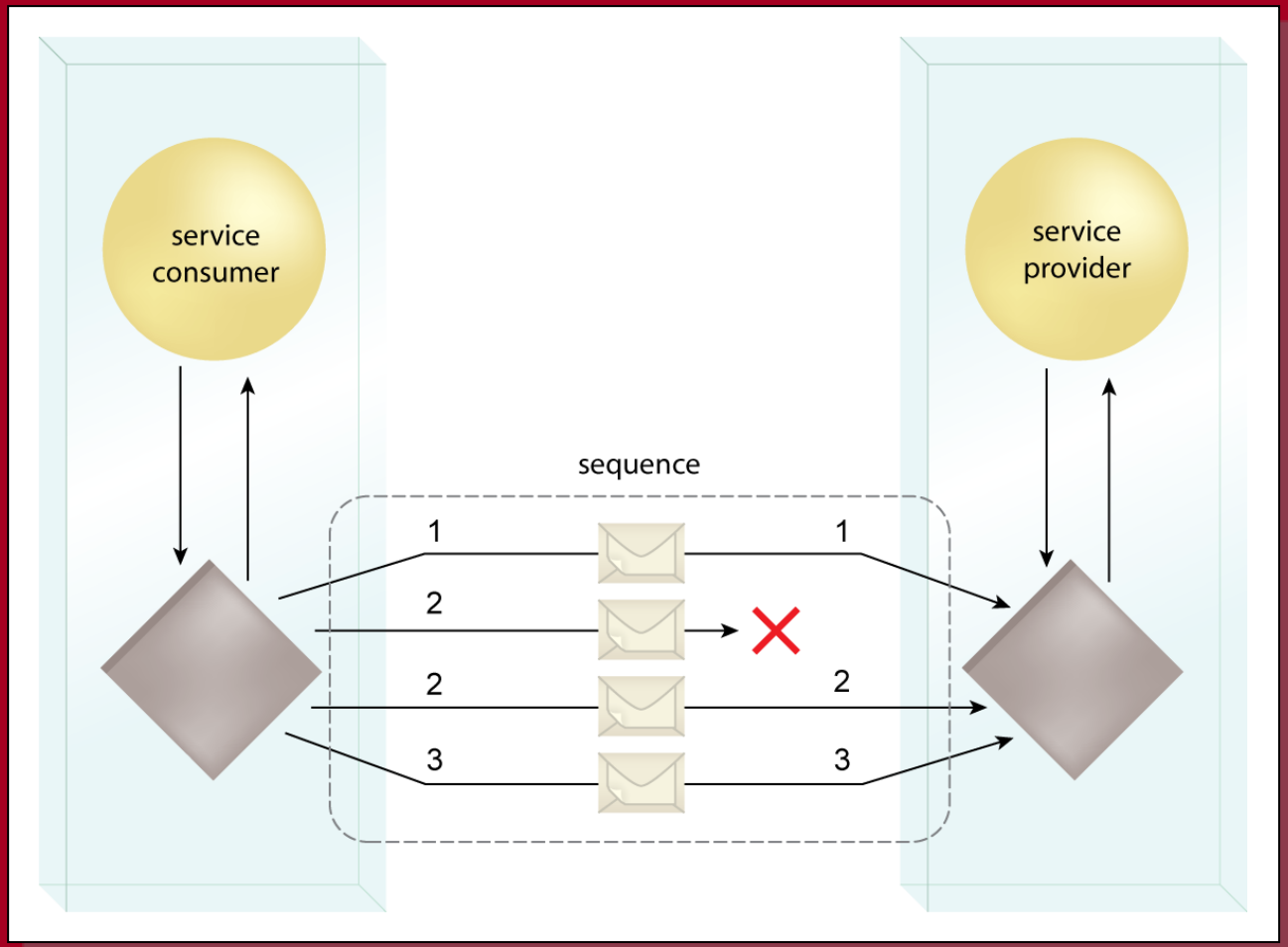
Reliable Messaging (WS-RM)

Delivery assurances are predefined rules that establish reliability delivery patterns:

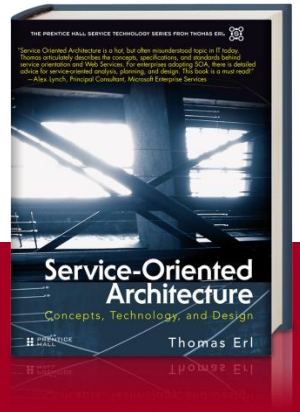
- AtMostOnce - Promises the delivery of one or zero messages.
- AtLeastOnce - Allows a message to be delivered once or several times.
- ExactlyOnce - Guarantees that a message will only be delivered once.

Reliable Messaging (WS-RM)

The InOrder delivery assurance is used to guarantee that messages are delivered in a specific sequence.



Reading



"Service-Oriented Architecture: Concepts, Technology & Design"

Chapter 7, Case Study Examples
(pages 226-227, 236-237)

Policies (WS-Policy)

Policies (WS-Policy)

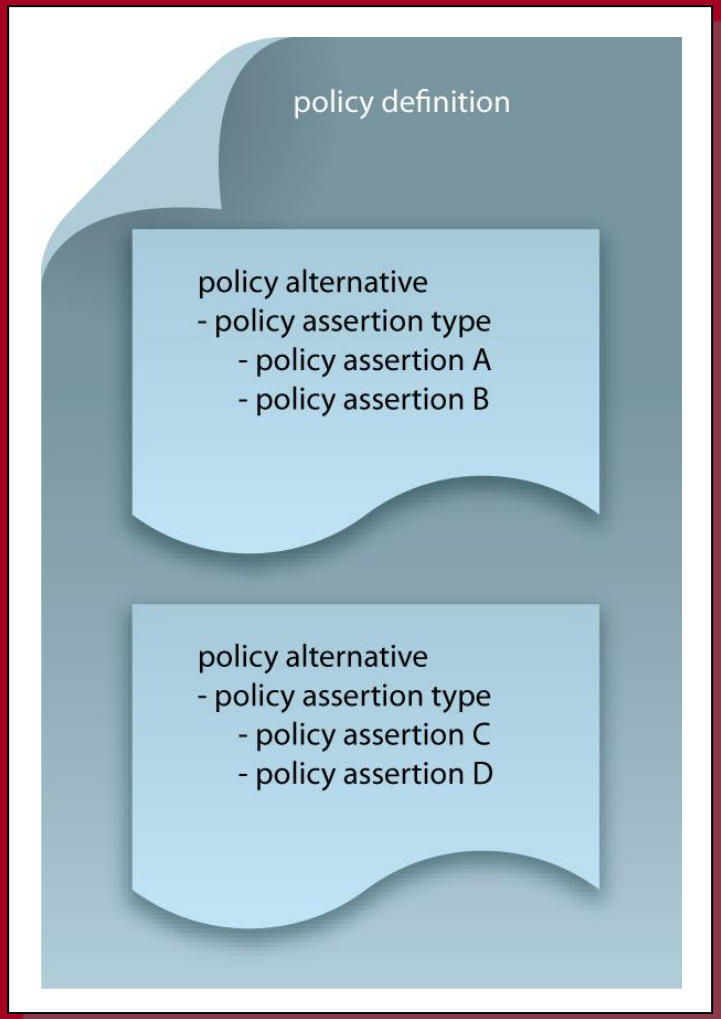
- The WS-Policy framework provides a set of specifications that define the assembly and structure of policy description documents
- The use of policies allows a service to express characteristics and preferences beyond its technical interface.
- This alleviates the service from having to implement and enforce policy rules and constraints in a custom manner.

Policies (WS-Policy)

- The policy document is referred to as the policy definition. Policies are represented individually by required or optional policy assertions.
- Industry standard policy assertions exist, but they can also be custom-defined.
- Industry standards that provide policy assertions include WS-ReliableMessaging, WS-Addressing, WS-SecurityPolicy.
- An example of a custom policy assertion could be something like "Provide Log Details".
- WS-Policy allows for the definition of policies that can be ignored by service consumers, which allows for the addition of policy assertions that won't "break" service contracts.

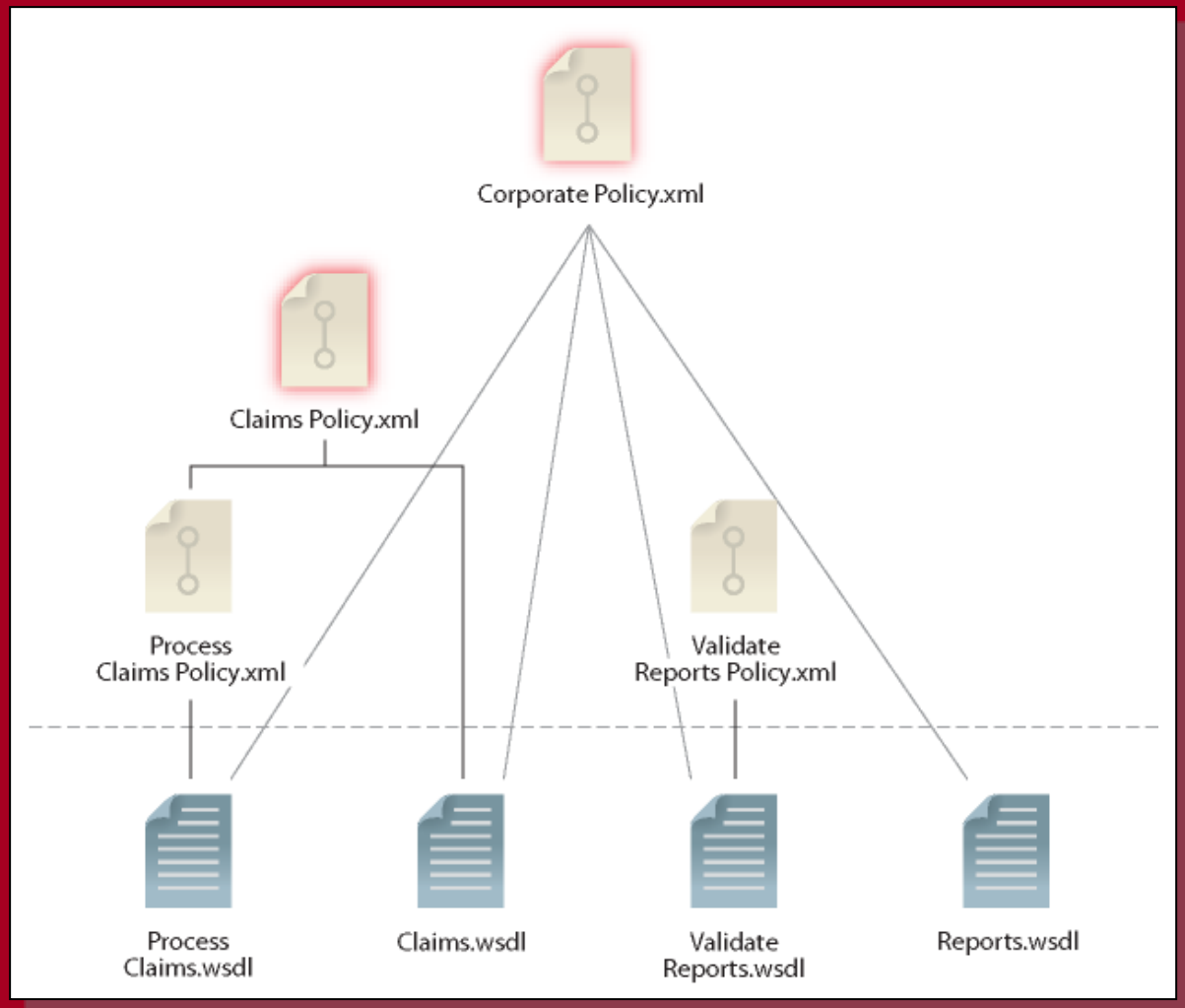
Policies (WS-Policy)

- Policy assertions can be grouped into policy alternatives.
- Each policy alternative represents one acceptable (or allowable) combination of policy assertions.
- This gives a service provider the ability to offer service consumers a choice of policies.



Policies (WS-Policy)

WS-Policy definitions can be shared across WSDL definitions allowing for the creation of global or domain-specific policy assertions.



Note

Module 10: Advanced Web-Based Service Technology provides further coverage of the WS-Policy markup language.

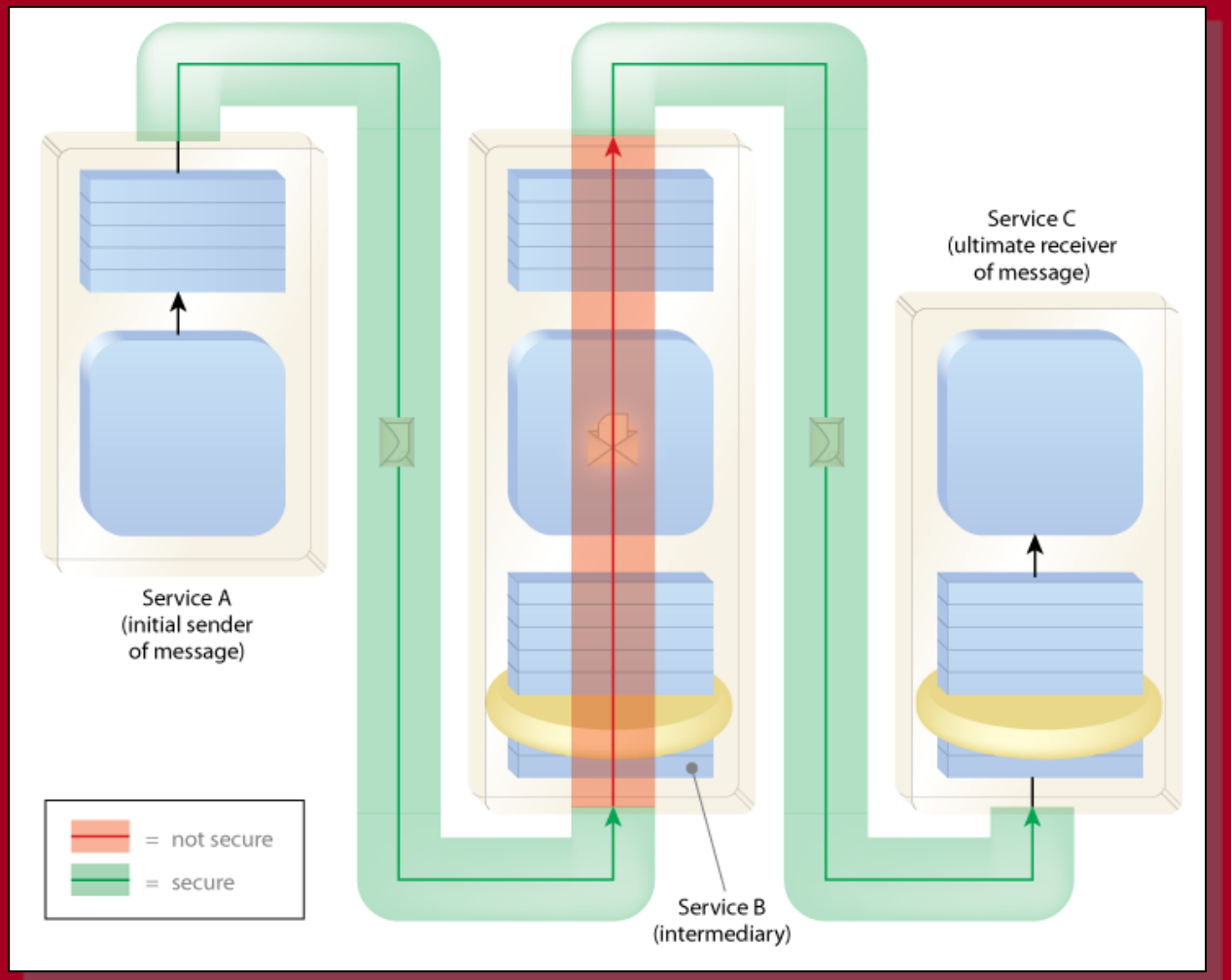
Security (WS-Security)

Security (WS-Security)

- The WS-Security framework provides extensions that can be used to implement message-layer security.
- This enables message contents to be protected during transport and during processing by service intermediaries.
- WS-Security, in conjunction with additional technologies, can provide authentication and authorization control.

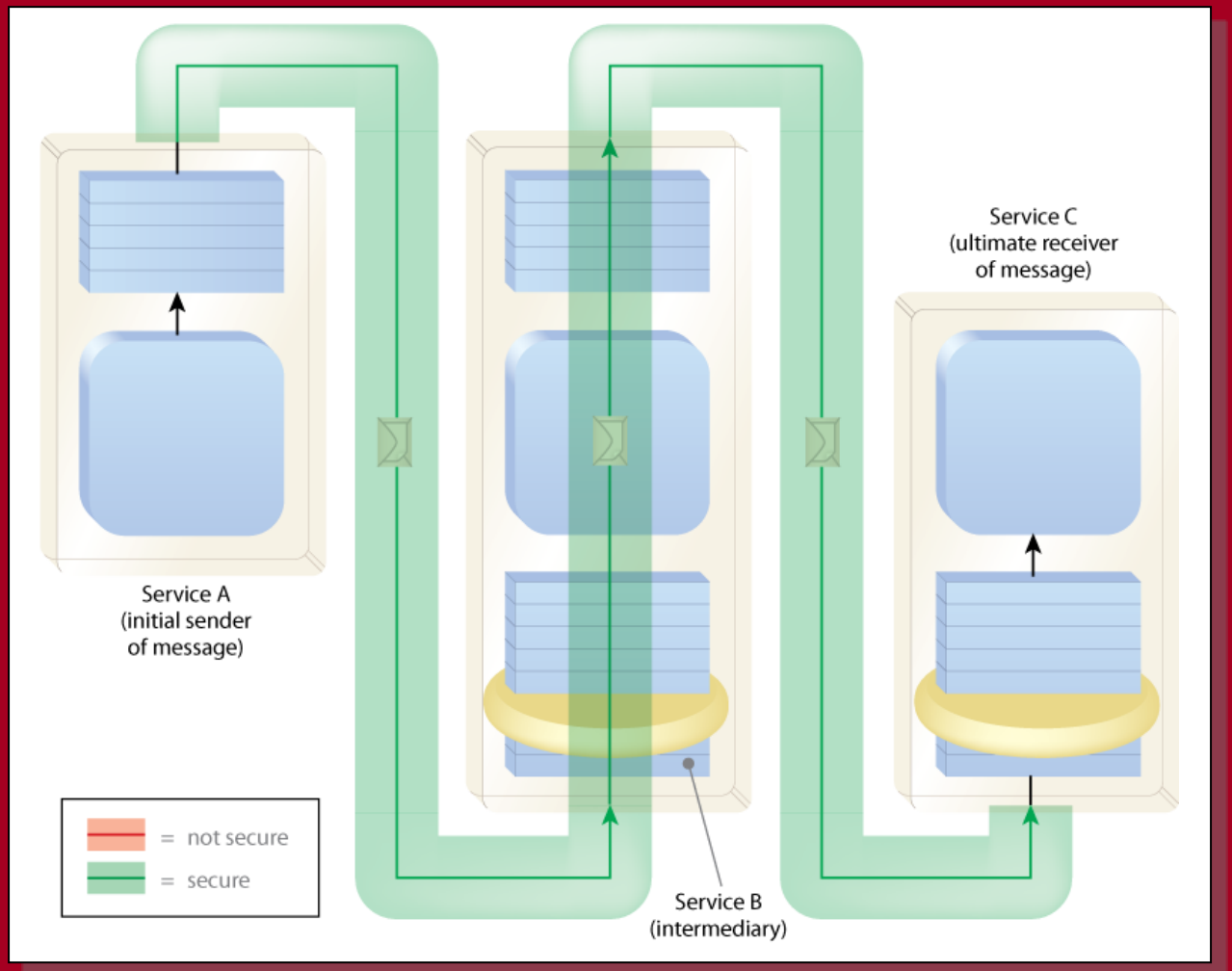
Security (WS-Security)

Transport-layer security is often insufficient when intermediaries are involved because the message is not protected while being processed by intermediaries.



Security (WS-Security)

WS-Security works with the XML-Signature and XML-Encryption standards to guarantee message integrity, authenticity, and confidentiality.



Security (WS-Security)

SOA security is a large topic that is covered in the following separate modules:

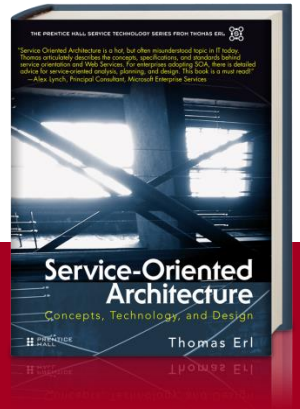
Module 18: Fundamental SOA Security
Module 19: Advanced SOA Security
Module 20: SOA Security Lab

Exercise

Exercise 2.9

WS-* Pop Quiz

Reading



"Service-Oriented Architecture: Concepts, Technology & Design"

Chapter 7, Case Study Examples (pages 247, 265-266)

Enterprise Service Bus (ESB) Overview

Enterprise Service Bus (ESB)

- An enterprise service bus (ESB) is considered middleware that establishes an intermediate layer of processing with a variety of message processing features.
- An ESB can help resolve common service composition design problems related to reliability, scalability, compatibility, and message routing.
- Different vendors provide different ESB products and platforms.

Enterprise Service Bus (ESB)

An ESB is commonly expected to provide:

- Asynchronous Queuing
- Intermediate Routing
- Reliable Messaging
- Policy Centralization
- Rules Centralization
- Event-Driven Messaging
- Data Format Transformation (Service Broker)
- Data Model Transformation (Service Broker)
- Protocol Bridging (Service Broker)

These are explained individually as design patterns in Module 8.

Enterprise Service Bus (ESB)

The latter three items on the preceding list are related to Service Broker functionality in that they carry out transformation and bridging functions.

For example:

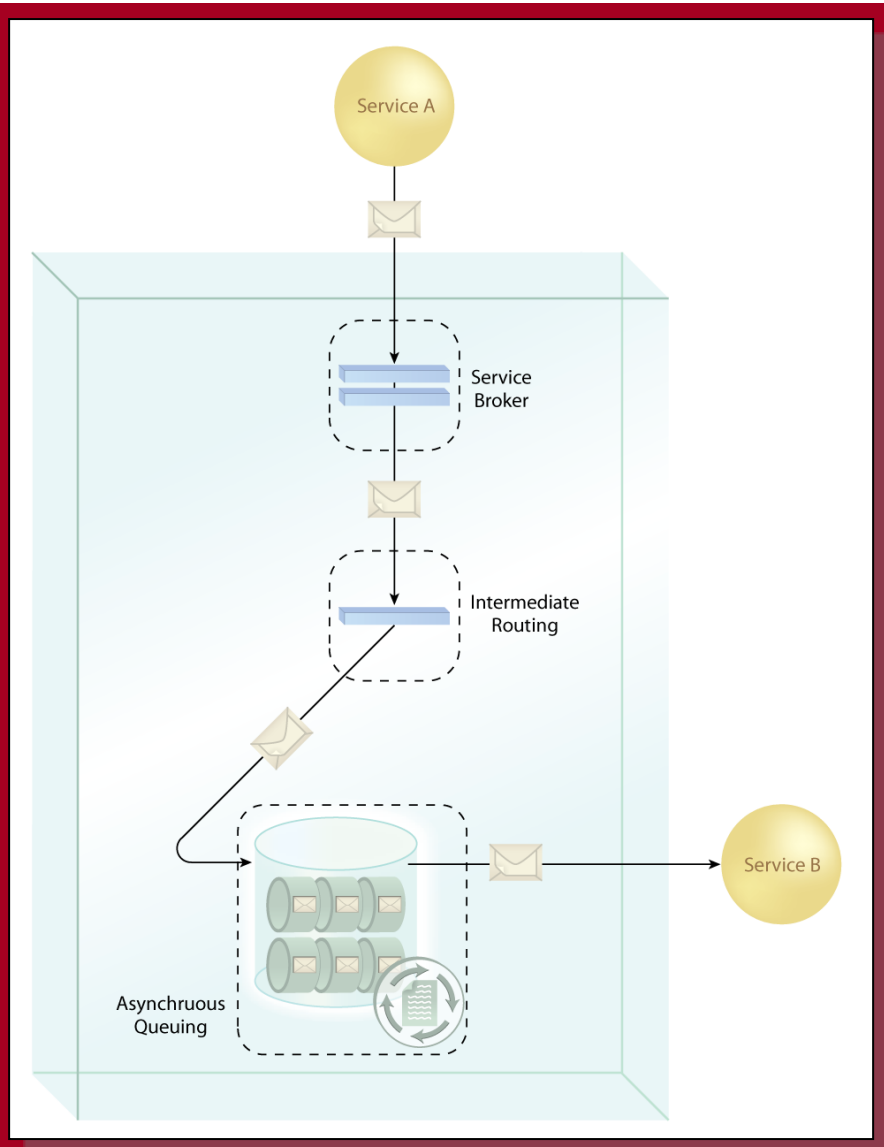
- They enable services using different data models (that represent the same business document) to communicate with each other. XSLT is commonly used for this purpose.
- They enable services using different communication protocols to communicate with each other.
- They enable services using different data formats (such as XML and ASCII) to communicate with each other.

These Service Broker functions can be used individually or at the same time for a given data exchange.

Enterprise Service Bus (ESB)

A message under-going various processing steps carried out by an ESB.

In this case, the message is first transformed by a Service Broker, then dynamically routed, and finally stored in a queue before it is delivered.



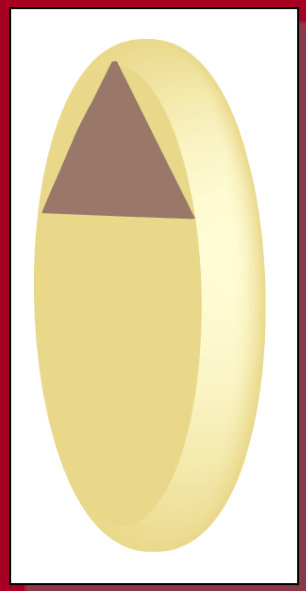
REST Services: Contracts, Resources & Messaging

REST Overview

- REST (Representational State Transfer) is an architectural style based on the underlying architecture of the World Wide Web and originated in a Ph.D. dissertation by Roy Fielding in 2000.
- Unlike SOAP-based Web services architecture, which is primarily based on the combined usage of specific technologies and industry standards, REST introduces an architectural style defined by goals and constraints.
- Architectural goals represent the properties that need to exist for an architecture to be considered RESTful, and constraints can be viewed as the design decisions that need to be applied in order to realize these goals.
- REST architectural goals and constraints are covered separately in upcoming sections.

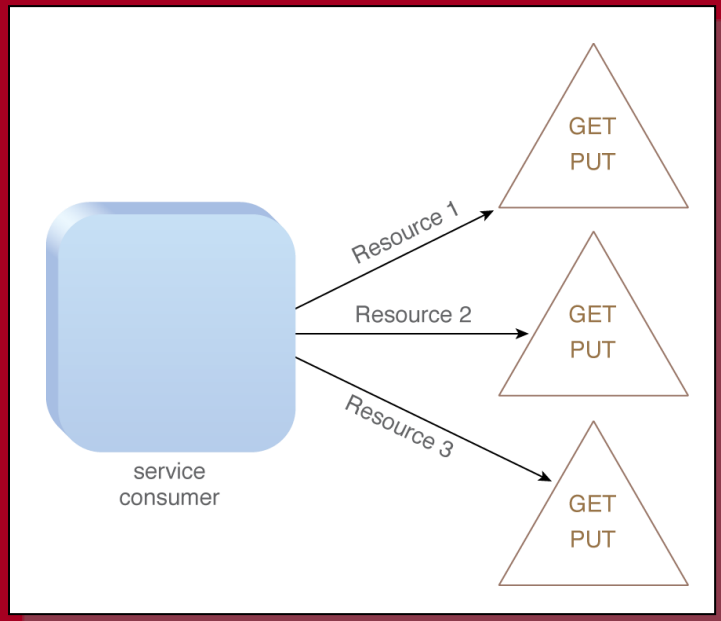
REST Services and Service Contracts

- REST services do not have individual service contracts, but instead share a uniform contract, most commonly via the standard HTTP protocol.
- REST services typically communicate with each other using HTTP methods (such as GET, PUT, POST, DELETE, HEAD, OPTIONS) provided by the uniform contract.
- Each REST service can be designed to support only a subset of available HTTP methods.
- Unlike SOAP-based Web services, where each service usually has its own service contract, many REST services generally share the same uniform contract.



REST Services and Resources

- Uniform contract methods are extremely generic and multi-purpose and therefore have little meaning beyond basic, high-level functions (such as Get Something or Update Something).
- The uniform contract alone does not constitute a REST service contract.
- The high-level methods provided by the uniform contract are used to access specific resources, each of which represents a custom function or logic that the REST service can provide.



REST Services and Resources

- In order for the service consumer to communicate to the REST service which resource it needs to access, it provides a resource identifier.
- The resource identifier determines the specific function requested by the service consumer (such as Invoice Total). Resource identifiers are expressed using URLs or URIs, such as: `www.example.com/invoicetotal`
- Together, these two parts of the service consumer request message give the REST service what it needs to carry out a specific function (such as Get Invoice Total).
- Therefore, a REST service capability can be considered a combination of a uniform contract method and a resource identifier.

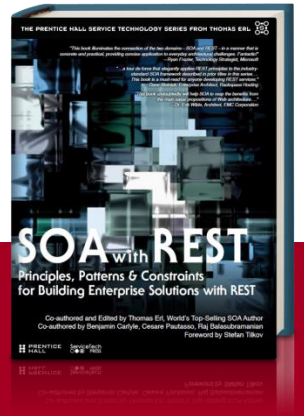
REST Service Messaging

- When REST services exchange messages, HTTP headers can be used to carry metadata, such as authentication tokens, response codes, and error information.
- Resources can be represented using different formats and protocols, such as XML, POX (Plain Old XML), or JSON (JavaScript Object Notation).
- Message data exchanged by REST services can be based on XML and can be defined using XML schemas.

Media Types

- Media types define the encoding of data sent in messages to and from REST services.
- Media types are a common and established part of Web-based communication. Examples of media types include: application, audio, image, text, and video.
- Media types have unique identifiers that are used by message recipients to invoke the correct parser for the message contents.
- For example, the `image/jpeg` identifier communicates that the message contains a JPEG image file.
- Media types can optionally encompass XML Schema definitions.

Optional Reading



"SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST"

Chapter 6: Service Contracts with REST

Note

The following sections that cover REST Architectural Goals and REST Constraints contain content that relates goals to constraints and vice versa.

This mapping of goals with constraints is provided for informational purposes only and is not covered on Exam S90.02.

REST Architectural Goals

Architectural Goals

The design goals of REST correspond to the realization of the following key architectural properties:

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Reliability

The purpose of these properties is to achieve an architecture that resembles the World Wide Web.

Performance

Three primary performance concerns are:

- network performance (throughput, overhead, bandwidth)
- user-perceived performance (latency and completion time of user-based tasks)
- network efficiency (quantity of service interactions, quantity of data exchanged with services)

REST aims to improve performance via the use of caches to keep data close to where it is being processed, and by keeping each interaction with services simple and self-contained (as a request/response pair).

Scalability

Scalability refers to the ability of an architecture to support a large number of instances or concurrent interactions.

REST addresses four types of scalability:

- scaling up (increasing the capacity of services, consumers, and other network components)
- scaling out (distributing load across multiple programs)
- smoothing out (evening out the number of interactions over peak and non-peak periods)
- decoupling (separating the consumption of finite resources, such as memory, from concurrent consumers)

Simplicity

- REST aims to simplify distributed service architectures in order to foster flexibility.
- A major contribution of REST to architectural simplicity is its usage of a uniform contract (as well as the Uniform Interface constraint, explained later).
- The Client-Server and Layered System constraints (explained in the upcoming *REST Constraints* section) also contribute to simplifying architectures.
- REST-style architecture is also further simplified by discouraging variability. For example reusable REST services and media types reduce variation within a given architecture.

Reliability

- Reliability is the degree to which an architecture is susceptible to failure.
- Reliability can be improved by avoiding single points of failure with service implementations and connectivity.
- An important architectural reliability feature is the ability for service consumers to continue operating correctly over partial or total failures of their service.
- This requires the state of their session with the service to be persistent in a way that it is unlikely to be lost due to the failure of the service.

Reliability (Idempotency)

- It is also important that any interaction between a service and its consumer is supported by a feature to detect interaction failures and to recover from them.
- This type of feature generally relies on the ability to repeat request messages without resulting in unexpected side-effects.
- A request message that can be safely transmitted more than once without being processed multiple times by a service is considered to be idempotent.
- Repeating idempotent requests is supported by ensuring service consumers have a copy of all session state through the Stateless constraint (explained shortly).

Modifiability

Modifiability is about the ease with which changes can be made within a REST architecture. This property is addressed as:

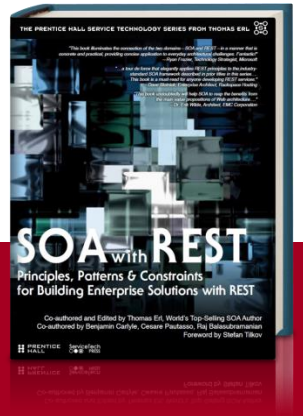
- Evolvability (the ability to change an implementation without impacting others)
- Extensibility (the ability to add functionality to the architecture, even while services are running)
- Customizability (the ability to temporarily modify services to do a special task)
- Configurability (the ability to permanently modify services)
- Reusability (the ability to add new solutions that reuse existing parts of the architecture without modification)

The Client-Server, Layered System, and Interface constraints contribute to these elements of modifiability.

Visibility

- The property of visibility relates to how much generic information a service agent can extract from a message without knowing any service-specific contract details.
- Generic middleware-based service agents need to be able to understand messages that pass through them just enough to monitor and mediate without forming design-time dependencies on specific services or their contacts.
- The focus of these service agents is commonly on improving administrative control and optimizing performance.
- REST supports visibility through the Interface constraint. (The same resource identifier syntax is used across all services and resources and the same methods and the same media types are reused.)

Optional Reading



"SOA with REST: Principles, Patterns & Constraints for Building Enterprise Solutions with REST"

Chapter 5, Section 5.2 Goals of the REST Architectural Style

REST Constraints

Overview

The following are the formal REST constraints, which act as rules used to define the distinct characteristics of REST architecture:

- Client-Server
- Stateless
- Cache
- Layered System
- Code-On-Demand
- Interface (Uniform Contract)

Each constraint can be viewed as a pre-determined design decision that can have both positive and negative impacts.

The intent is for the positives of each constraint to balance the negatives to produce an overall architecture that resembles the Web.

Client-Server

- The Client-Server constraint enforces the separation of concerns in the form of a client-server architecture.
- The separation of clients from the server helps to establish a fundamental distributed architecture, thereby supporting the independent evolution of the server logic and its clients.
- The Client-Server constraint has a positive impact on the Simplicity and Modifiability goals.

Stateless

- REST services are stateless between requests.
- A service can retain data relevant to its function, but it cannot store session state on behalf of service consumers.
- Any data that relates to a service consumer needs to be returned so that it can be stored by the service consumer.
- This constraint has a negative impact on the Performance goal, while having a positive impact on the Scalability, Simplicity, Visibility, and Reliability goals.

Layered System

- The Layered System constraint requires that a solution be comprised of multiple architectural layers where no one layer can "see past" the next.
- Layers can be added, removed, modified, or reordered without changing the logic of other layers.
- Middleware layers can be inserted transparently so that pre-existing interaction between a service and its consumer continues to occur in the same way, regardless of whether the consumer is talking to the original service or to newly added middleware.
- This constraint has positive impacts on Scalability, Simplicity, Modifiability, and Reliability goals. It can have both positive and negative impacts on the Performance goal.

Cache

- A cache acts as mediator between a service and its consumers.
- Requests passing through the cache can reuse previous responses to partially or completely eliminate some interactions over the network.
- Response messages from the service to its consumers are explicitly labeled as cacheable or non-cacheable.
- Caching is generally used to recover some of the performance lost as a result of the Statelessness constraint.
- This constraint can have a positive impact on the Performance and Scalability goals, but a negative impact on the Reliability goal.

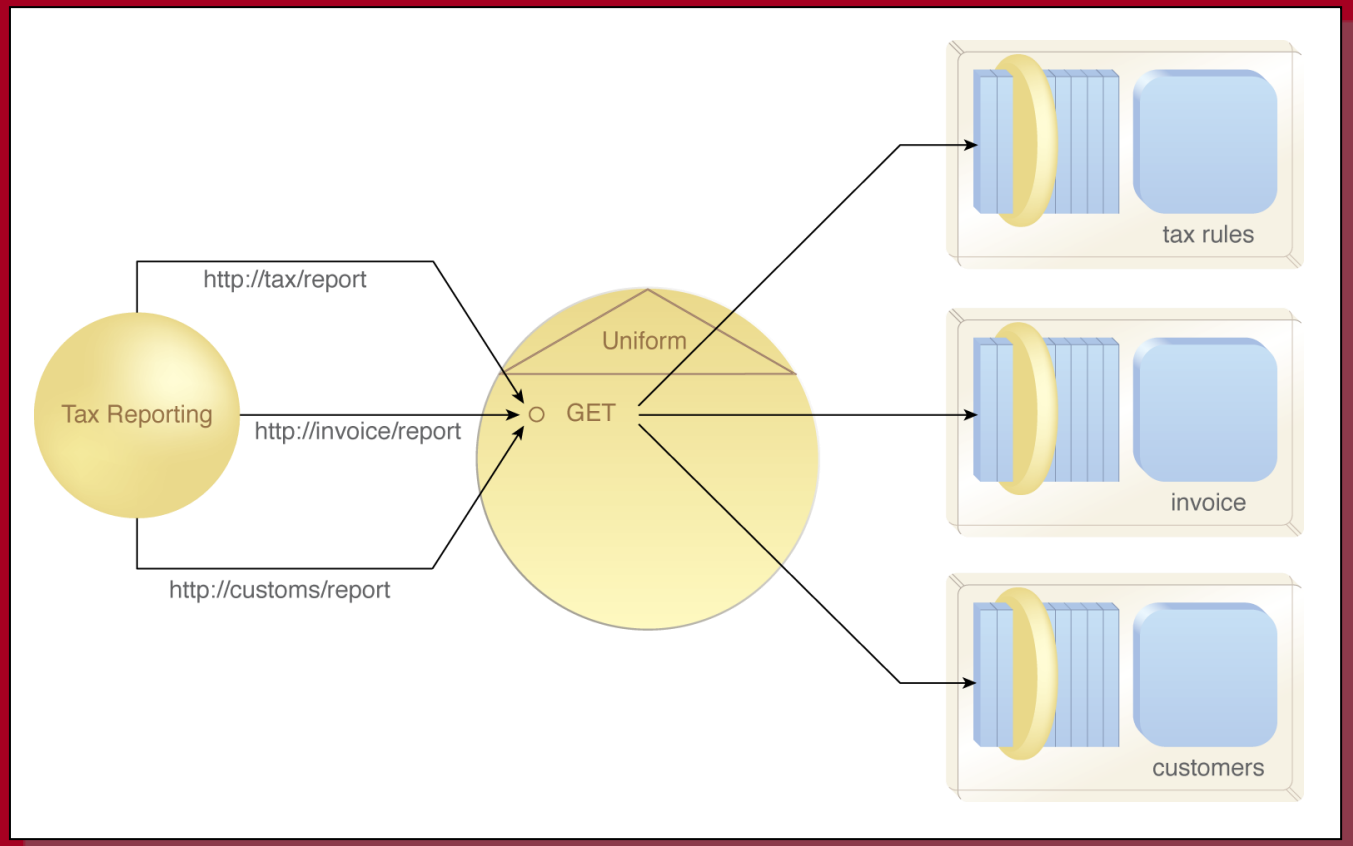
Code-On-Demand

- The Code-on-Demand constraint enables service logic as well as data to be sent back to service consumers in response messages.
- This allows services to dynamically extend consumers with new capabilities (a concept comparable to Web browser plug-ins).
- It also allows services to choose whether to execute logic on the service or consumer side of a message exchange.
- This constraint can have a positive impact on the Performance, Scalability, and Modifiability goals. It can have a negative impact on the Visibility and Reliability goals, and it can have both positive and negative impacts on the Simplicity goal.

Interface (Uniform Contract)

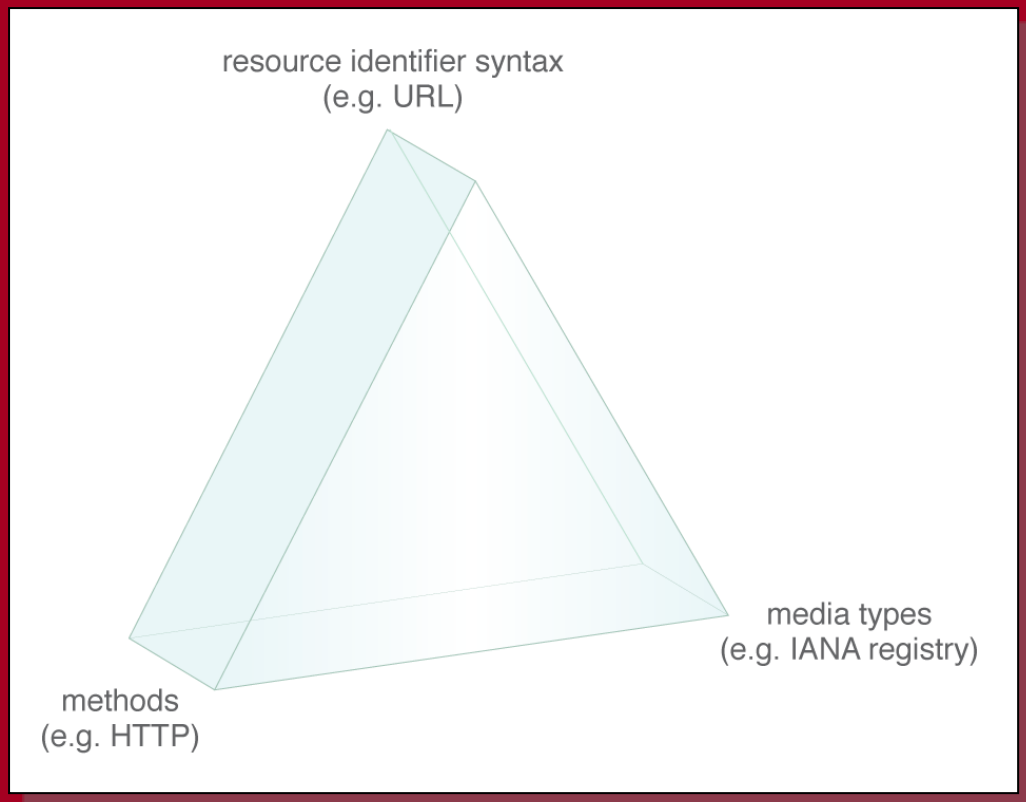
The Interface constraint corresponds to the uniform contract explained earlier.

This constraint requires that an overarching technical contract providing a set of generic methods be shared by multiple REST services.



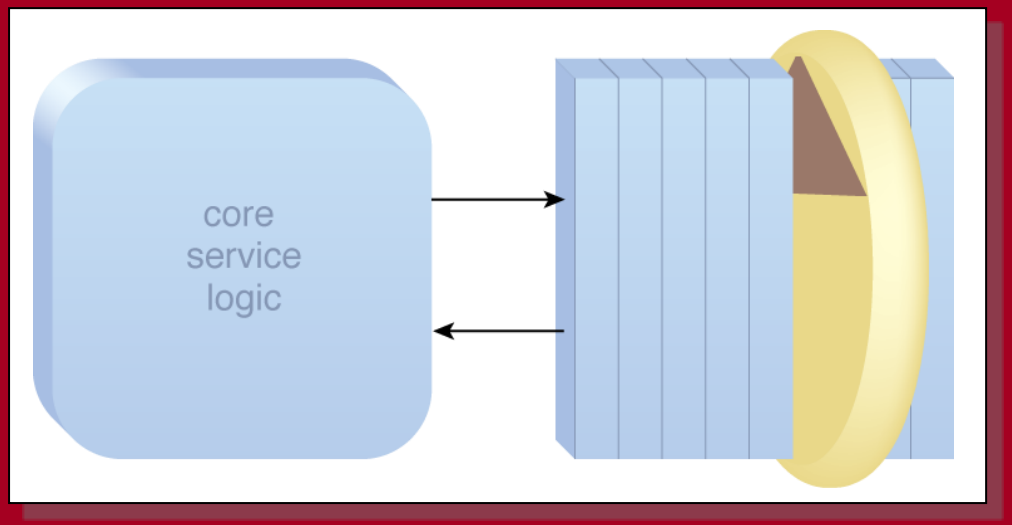
Interface (Uniform Contract)

- A uniform contract must support generic methods, resource identifiers, and media types.
- This "triangle" of facets forms the basis of the triangle notation used to distinguish REST service contracts from non-REST service contracts.
- This constraint has a positive impact on the Simplicity, Modifiability, and Visibility goals, but a negative impact on the Performance goal.



Uniform Contract Symbol

To distinguish a service architecture that uses a uniform contract, the triangle notation is used with the service contract symbol.

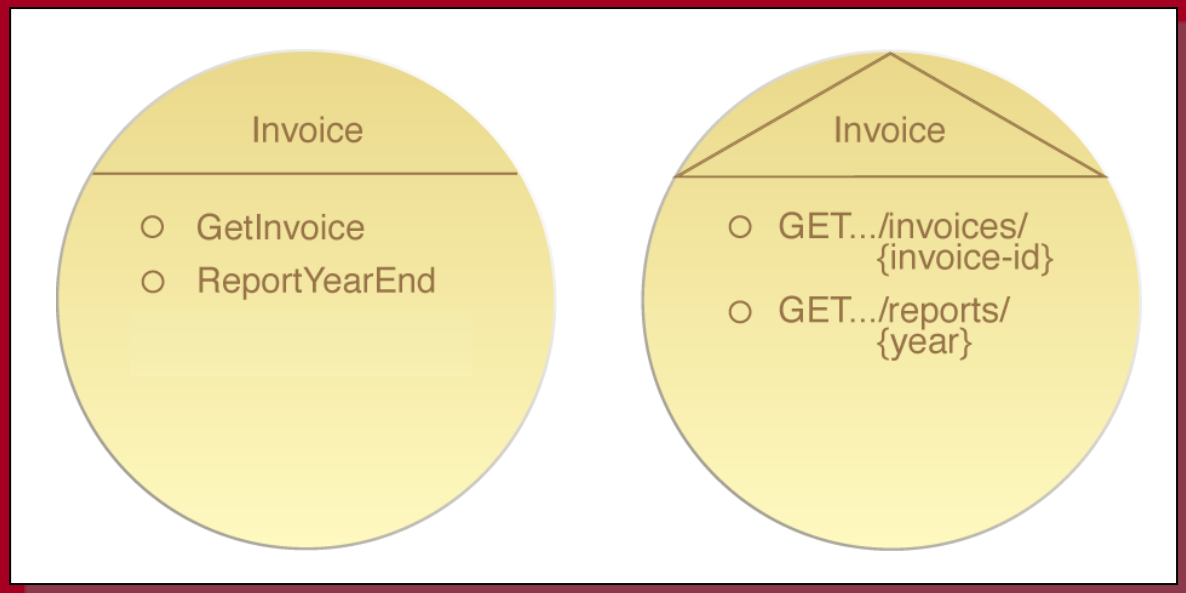


As shown in the diagram, the anatomy of a REST service architecture can be similar to that of a Web service architecture (as explained earlier in this module).

However, there are unique features and characteristics of REST service architectures. These are covered in Module 10.

REST Service Contract Symbol

From a modeling and design view, a REST service contract displays service capabilities as combinations of methods and resource identifiers (right). The overall symbol notation is similar to a SOAP-based Web service contract (left), except that the triangle is shown to indicate the usage of a uniform contract.



Exercise

Exercise 2.10

REST - Pop Quiz