Module Five: Service Oriented Architecture: architectural patterns and modelling methods of SOA, designing a distributed service system with SOA

Assignment deadlines reminder

- April 2nd – hot topic study journal
- April 10th – web service programming tutorial

Architecture

- The process of planning, designing and constructing structures

Software Architecture

- Structures of a software system and the discipline of creating such structures and systems
- Each structure compromises
  - Software elements
  - Relations among the elements
  - Properties of elements
  - Properties of relations

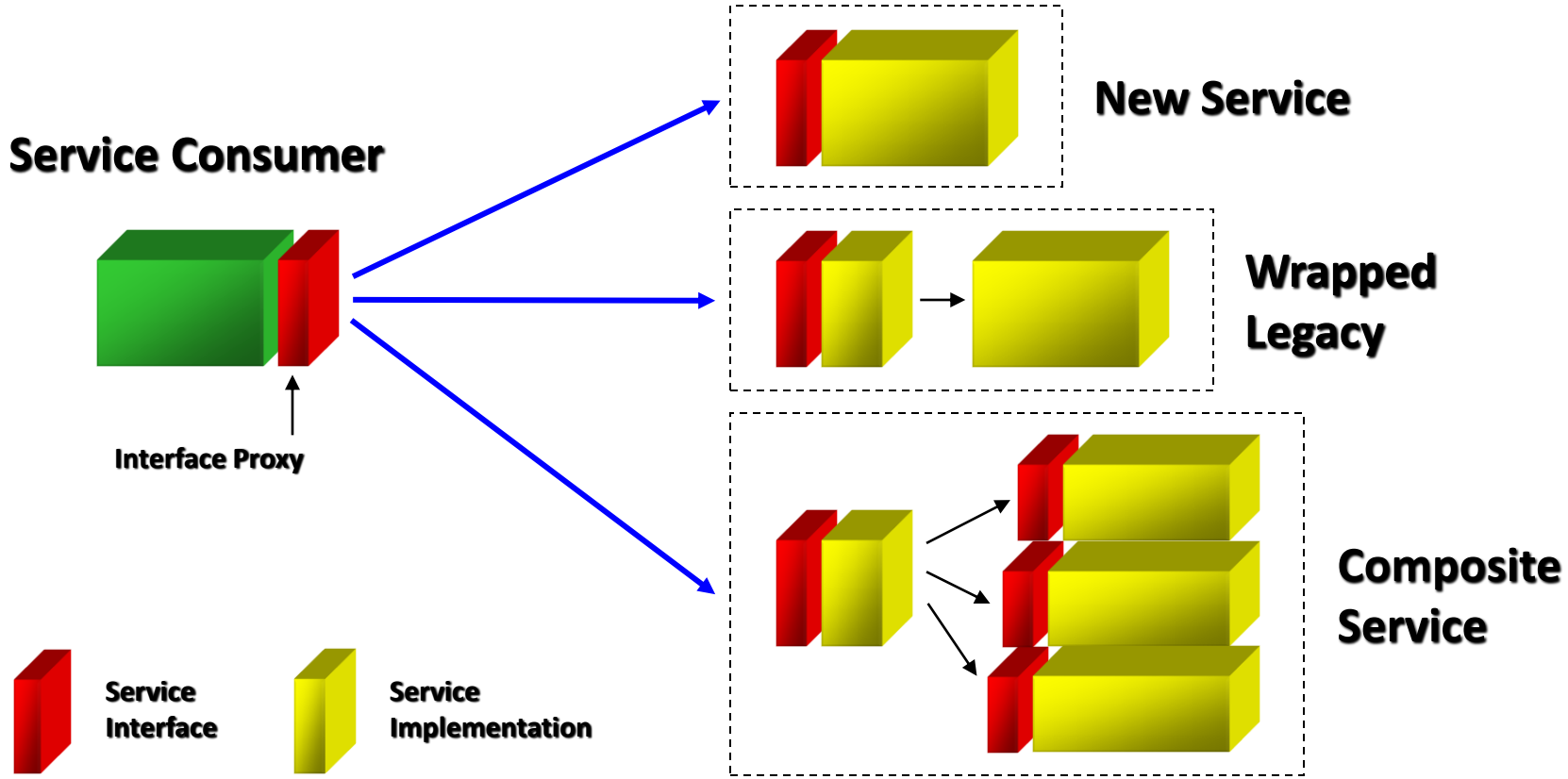
What is SOA?

- Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services.
- These services are well-defined business functionalities that are built as software components that can be reused for different purposes.

SOA Service

- A discrete unit of functionality that can be accessed remotely and acted upon and updated independently
- A repeatable task, for example:
  - Open an account
  - Perform a credit check
  - retrieving a credit card statement online

Anatomy of a Service



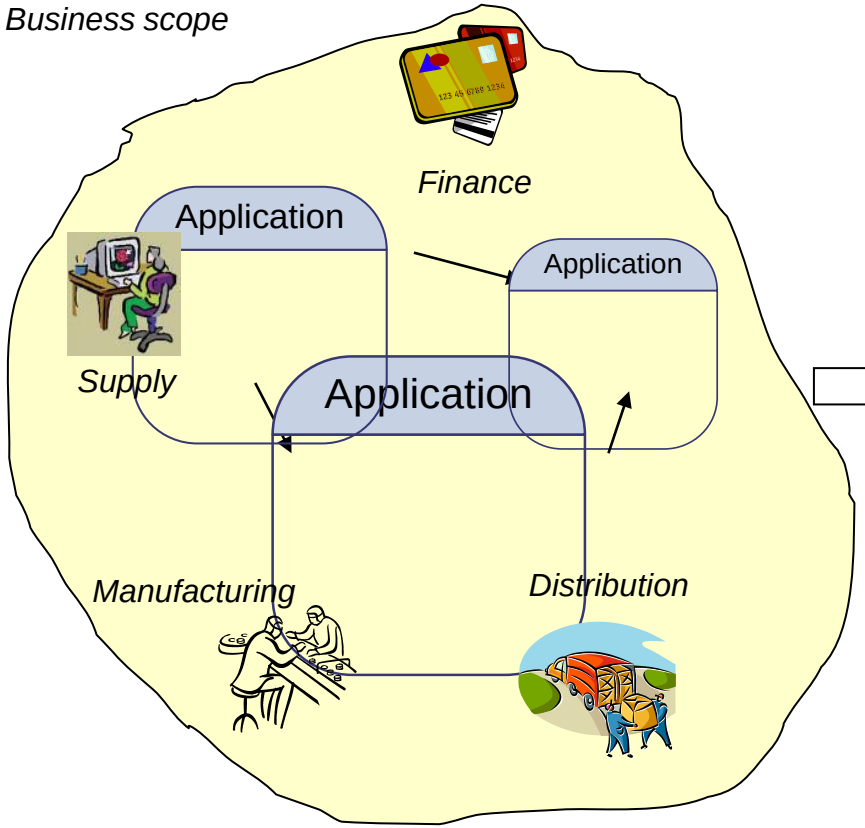
Why SOA in Enterprise?

- Why to introduce SOA in an organization? What are the benefits for enterprises?

Why SOA?

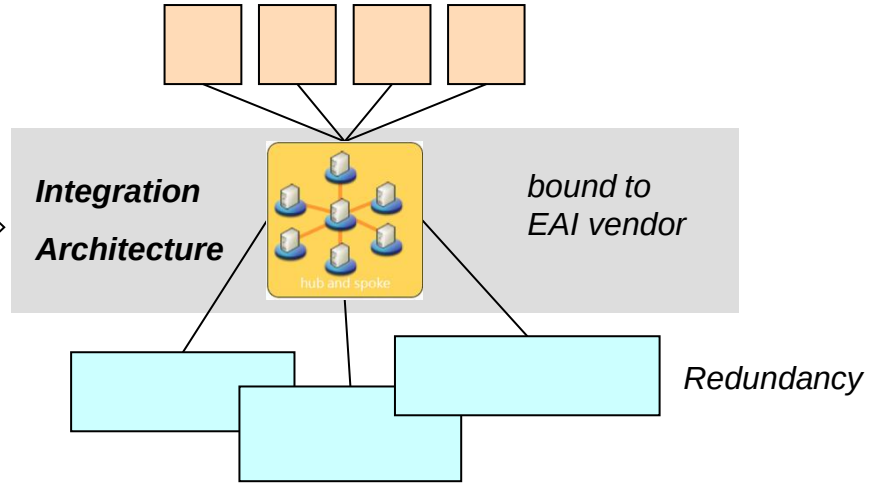
- From application centric to service centric environment

Application Centric



Business functionality is duplicated in each application that requires it.

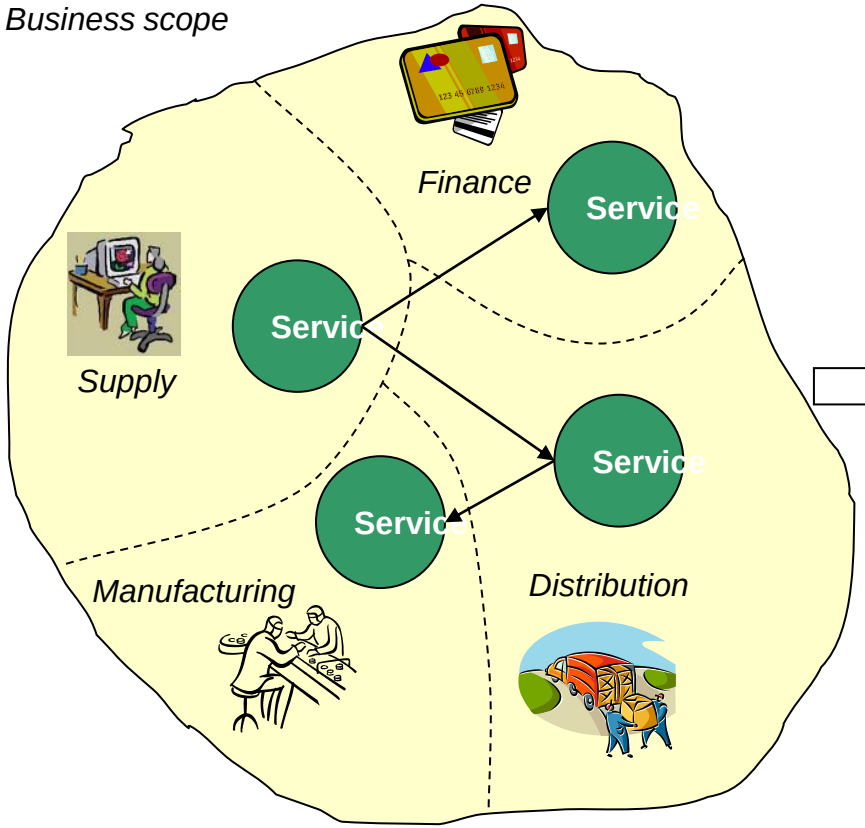
Narrow Consumers
Limited Business Processes



Overlapped resources
Overlapped providers

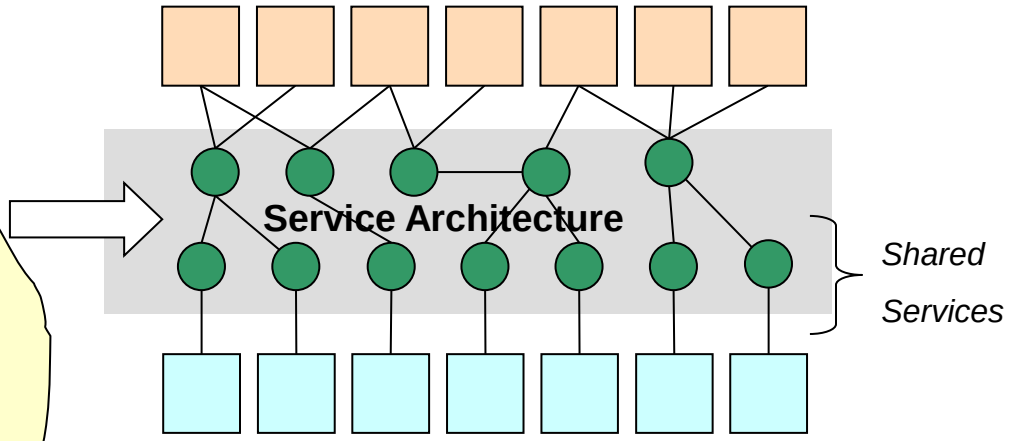
EAI ‘leverage’ application silos with the drawback of data and function redundancy.

Service Centric



SOA structures the business and its systems as a set of capabilities that are offered as Services, organized into a Service Architecture

Multiple Service Consumers
Multiple Business Processes

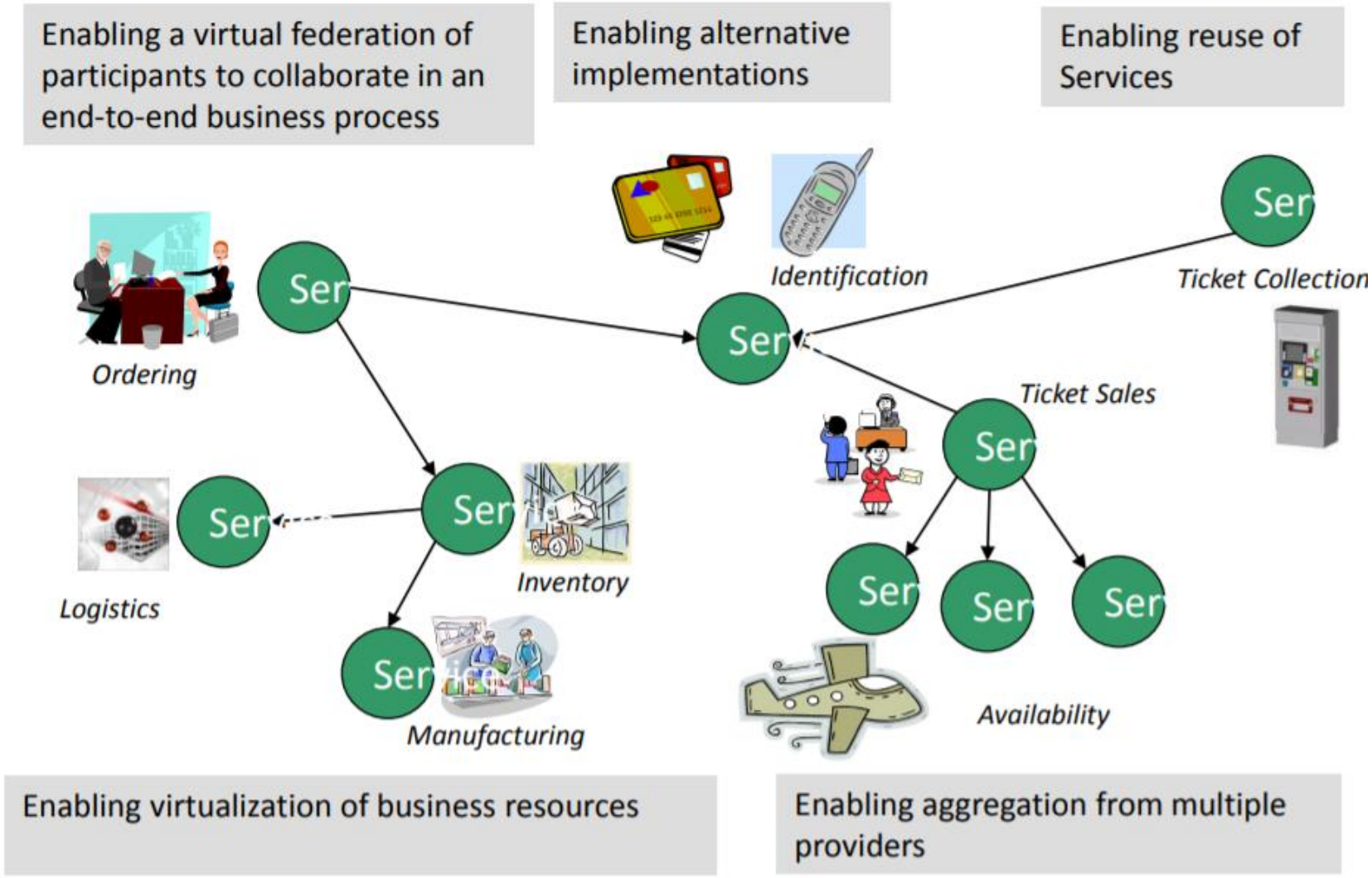


Multiple Discrete Resources
Multiple Service Providers

Service virtualizes how that capability is performed, and where and by whom the resources are provided, enabling multiple providers and consumers to participate together in shared business activities.

Why SOA in Enterprises? ENABLE FLEXIBLE, FEDERATED BUSINESS PROCESSES

- Enable flexible, federated business processes



Business benefits – decreased cost

- Decreased cost:
  - Add value to core investments by leveraging existing assets
  - New systems can be built faster for less money
    - Reducing integration expense
    - Built for flexibility
    - Long term value of interoperability

Business benefits – increased productivity

- Increased employee productivity:
  - Built on existing skills
  - Consolidate duplicate functionality

Business benefits - partnership

- Built for partnerships:
  - Standards based
  - Business relationships expressed via service interactions
  - Integration is driven by what is needed, not what is technically possible

Business benefits – agility

- Agility - Built for change
  - Helps applications evolve over time and last
  - Abstract the backend and replace over time
  - Focusing on core-competencies
  - Incremental implementation approach is supported
  - Service Outsourcing – new business model!

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication
- Platform independent use

Technical Benefits

- Services Scale
  - Build scalable, evolvable systems
  - Scale down to mobile devices
  - Scale up to for large systems or across organizations
- Manage complex systems
  - Does not require centralized services
  - Empowers users with high end communication
- Platform independent use
- Loose Coupling allows flexibility

Web Services vs SOA

- Are SOA and Web Services the same thing?

Open Question Points: 10

Setting

Based on your understanding, are SOA and Web Services the same thing? Please explain.

Answer

Web Services vs SOA

- Web services and SOA are not synonymous.

Web Services vs SOA

- Web services and SOA are not synonymous.
- SOA is a design principle, whereas web services is an implementation technology.

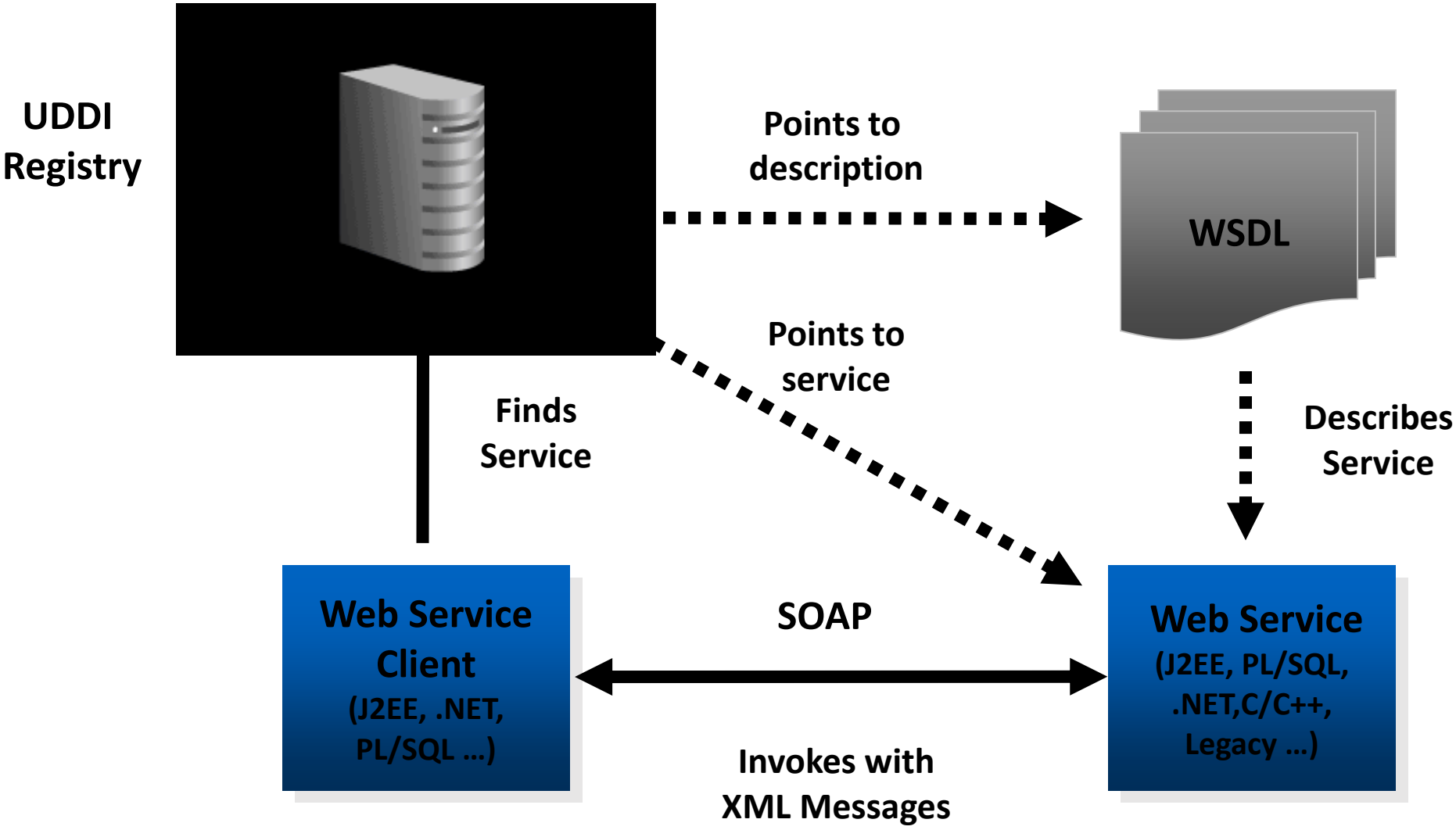
Web Services vs SOA

- Web services and SOA are not synonymous.
- SOA is a design principle, whereas web services is an implementation technology.
- You can build a service-oriented application without using web services

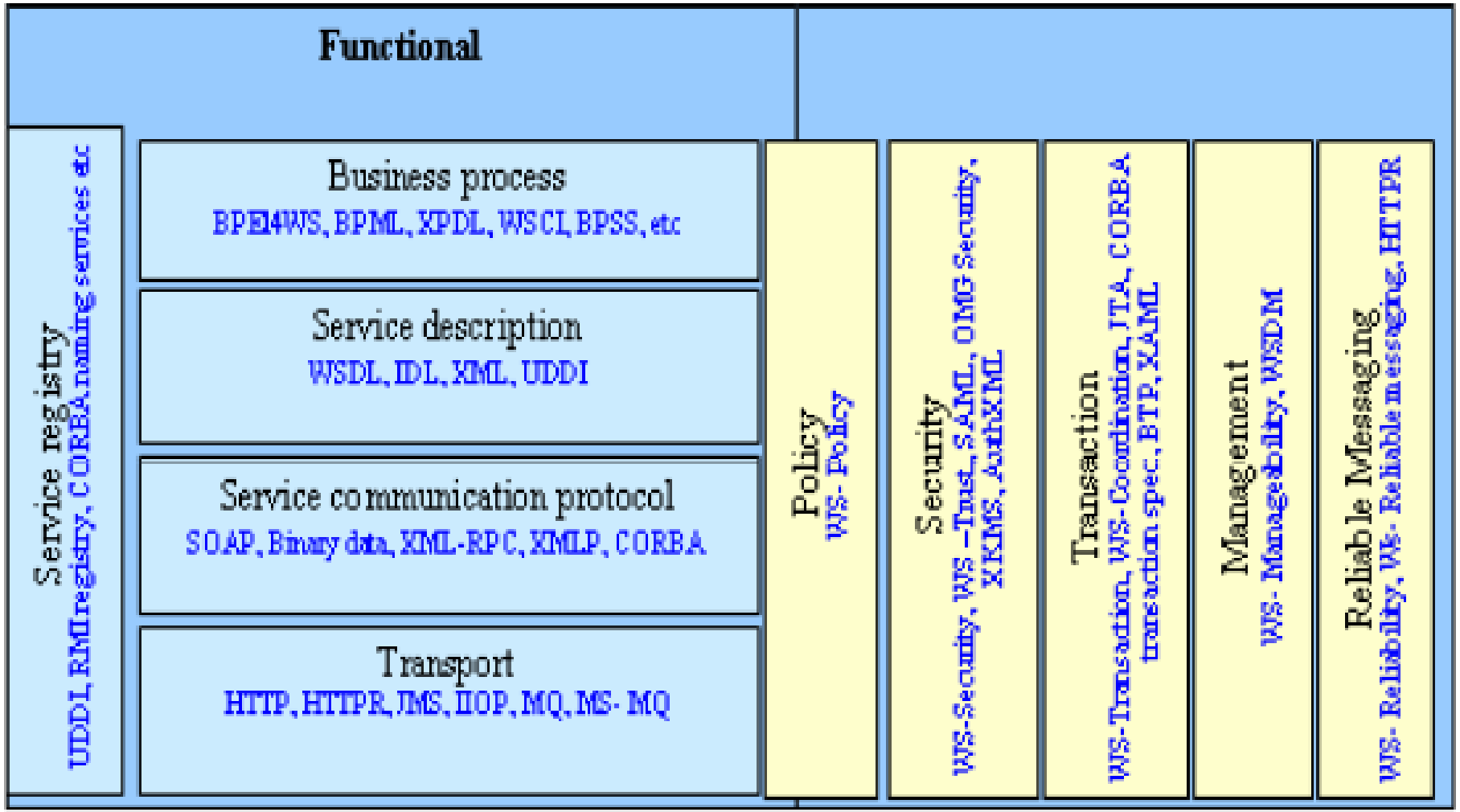
Web service implementation of SOA

- Service-oriented architecture can be implemented with web services.
- Functional building blocks are accessible over standard internet protocols
  - Independent of platforms and programming languages
- Web services can represent
  - New applications
  - Wrappers around existing legacy systems
- SOA can be implemented using other technologies

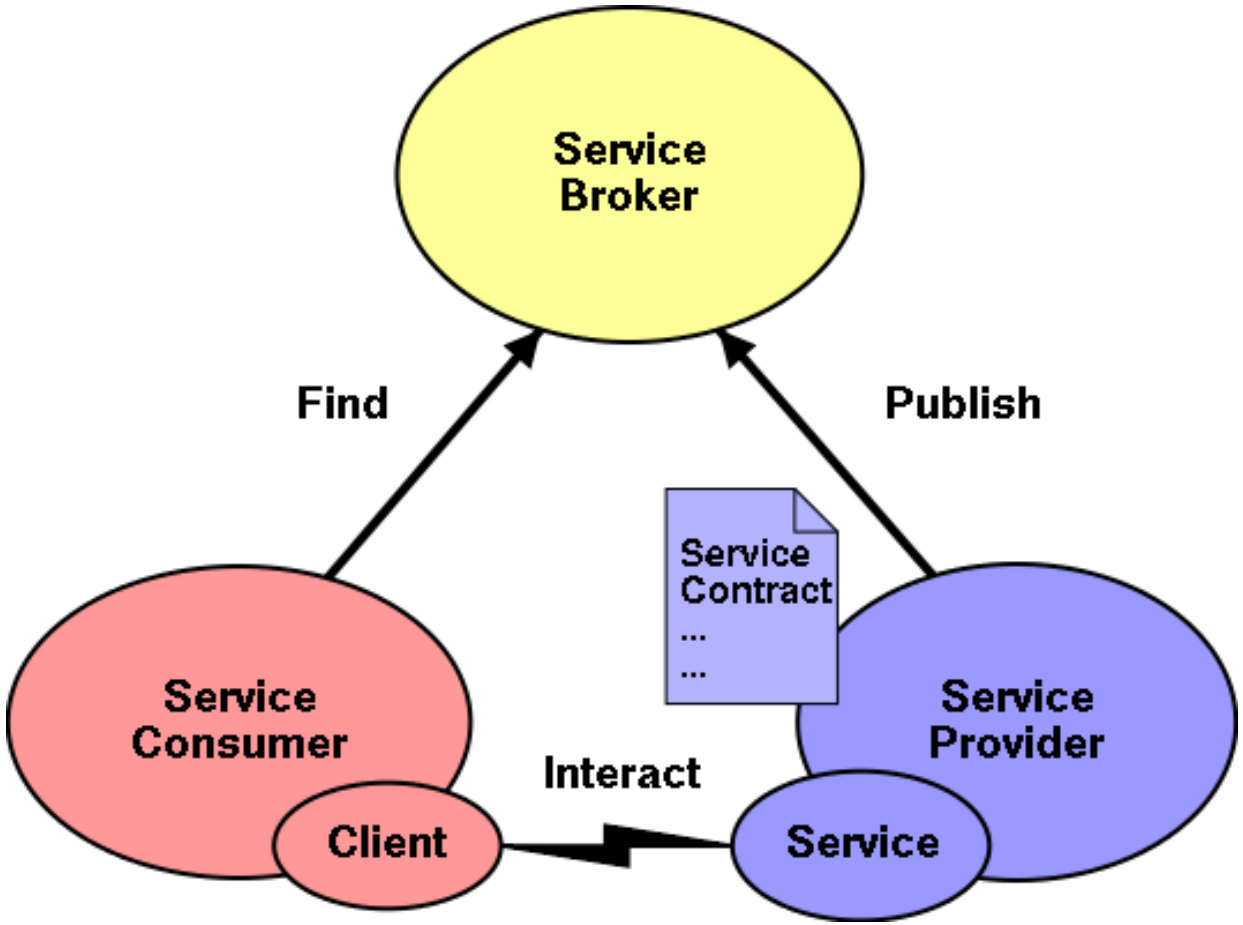
Basic Web Services



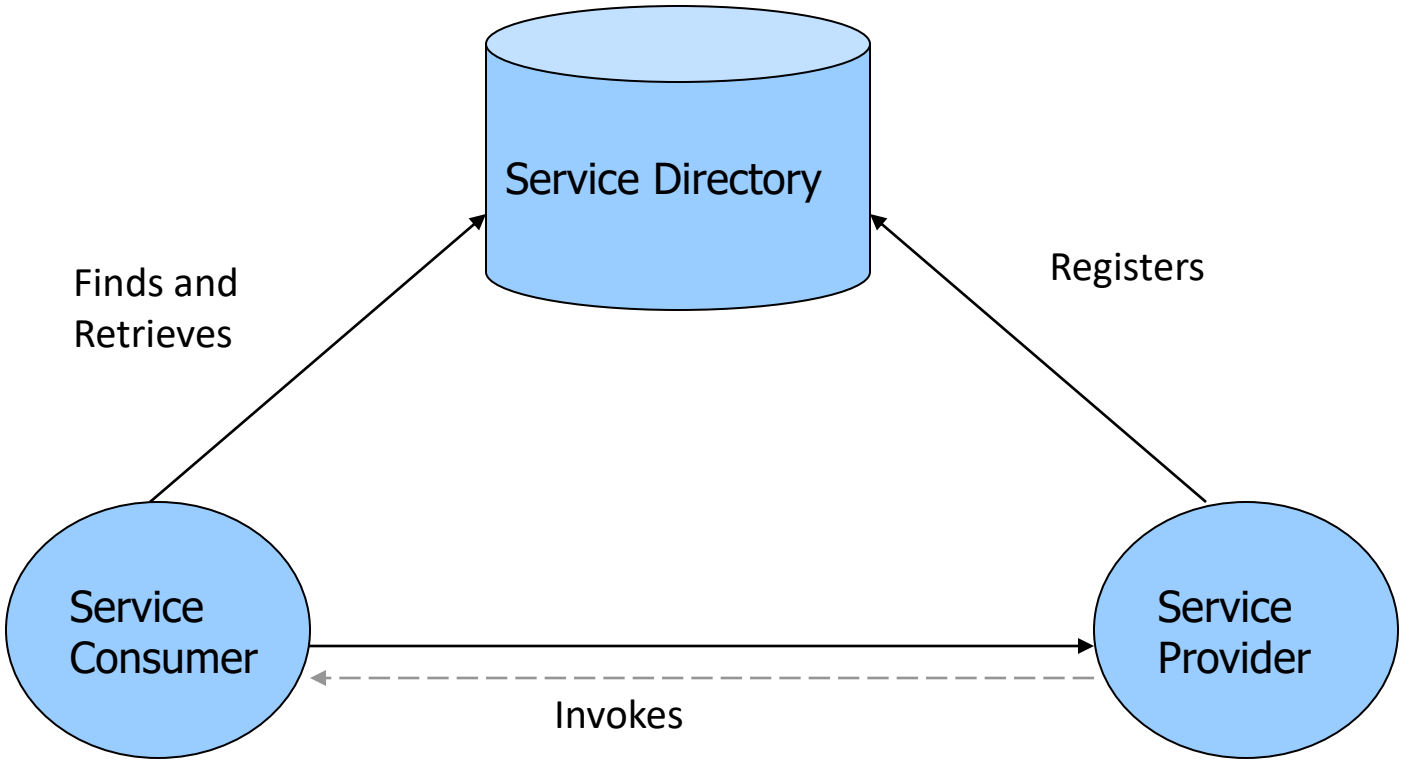
SOA Standards Stack



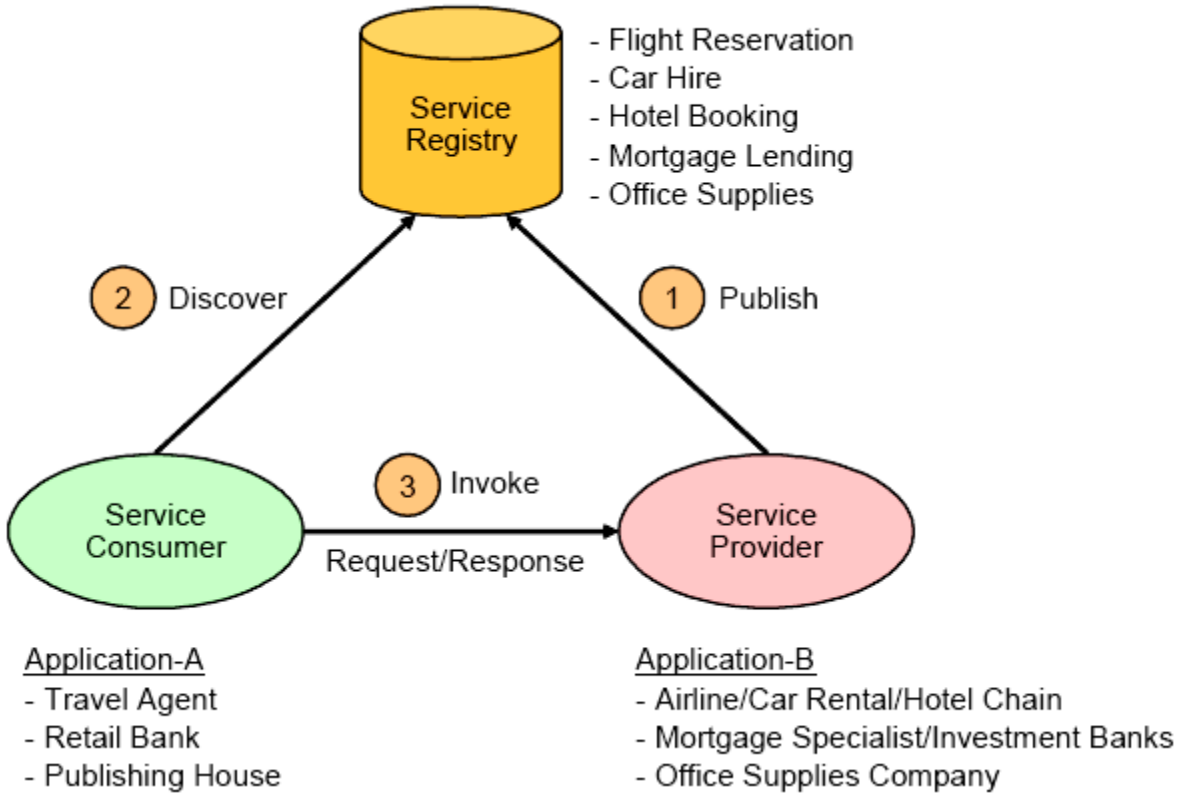
SOA Architecture



SOA Architecture



SOA Components and Operations



Basic Components of an SOA

- SOA consists of the following three components:
  - Service provider
  - Service consumer
  - Service registry
- Each component can also act as one of the two other components.
  - For instance, if a service provider needs additional information that it can only acquire from another service, it acts as a service consumer.

Poll 1 answer(s) at most

Setting

In your opinion, if the enterprise decides to use SOA paradigm, is it necessary for all the IT functions in this company to become services?

A Yes, only then the SOA will be successful

B No, we should select carefully which IT functions should become services

Submit

Characteristics of service-oriented architectures

- Any function can be a service but need not be.

Open Question Points: 10

Setting

Please complete this sentence:

The criteria for whether a function needs to be a service should be based on .....

Open Question is only supported on Version 2.0 or newer.

Answer

Recall our definition - What is SOA?

- Service-Oriented Architecture (SOA) is a set of principles and methodologies for designing and developing software in the form of interoperable services.
- These services are well-defined business functionalities that are built as software components that can be reused for different purposes.
- SOA design principles are used during the phases of systems development and integration.

Recall our definition - SOA Service

- A discrete unit of functionality that can be accessed remotely and acted upon and updated independently
- A repeatable task, for example:
  - Open an account
  - Perform a credit check
  - retrieving a credit card statement online

Characteristics of service-oriented architectures

- Any function can be a service but need not be. The criteria for whether a function needs to be a service is based on its reuse potential
  - Business process services
    - createStockOrder, reconcileAccount, renewPolicy
  - Business transaction services
    - checkOrderAvailability, createBillingRecord
  - Business function services
    - calculateDollarValueFromYen, getStockPrice
  - Technical function services
    - auditEvent, checkUserPassword,checkUserAuthorisation

Service Design Principles

- The principles of service-orientation provide a means of supporting and achieving a foundation paradigm based upon building of SOA characteristics.

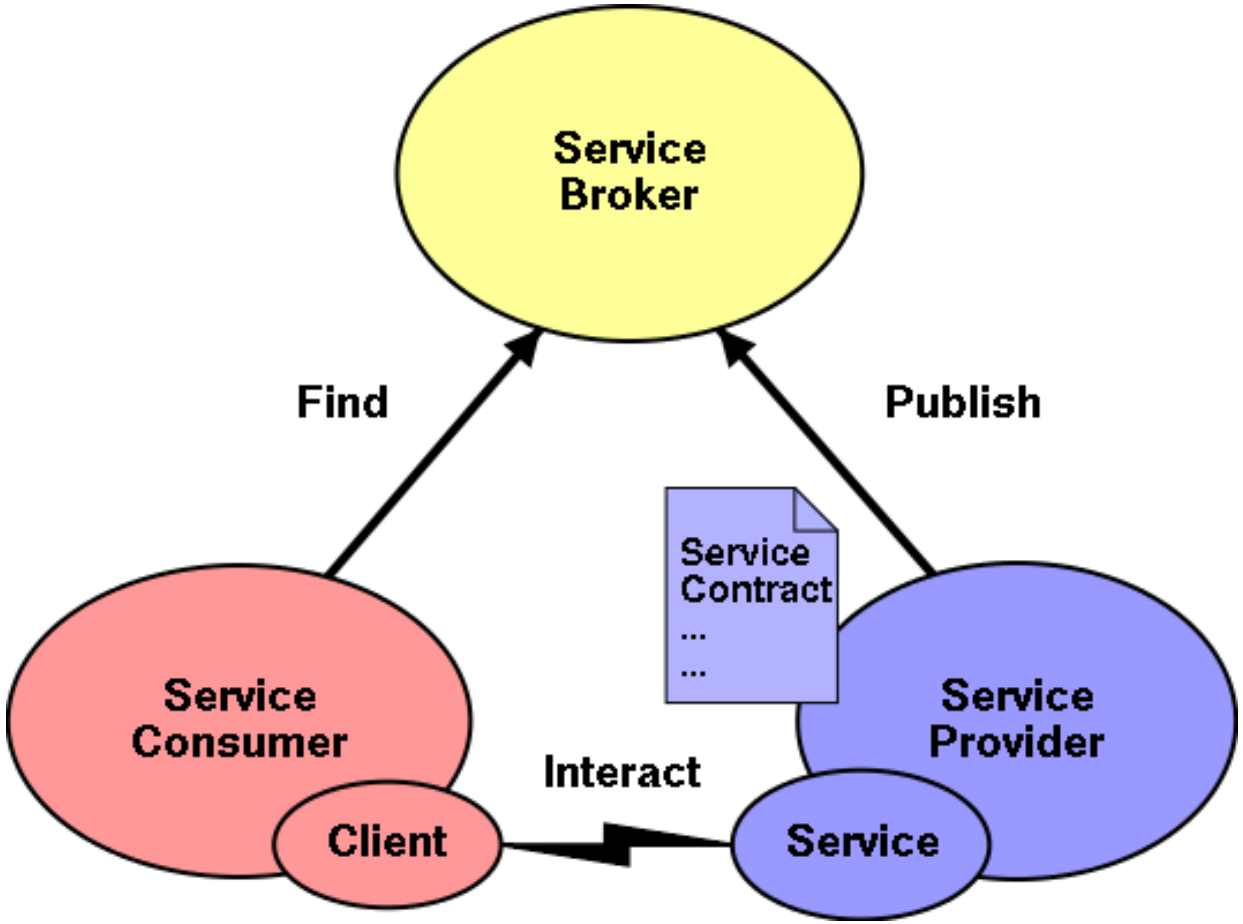
Common Service Design Principles

- Standardized Service Contracts
- Loose Coupling
- Abstraction
- Reusability
- Autonomy
- Statelessness
- Discoverability
- Composability

Standardized Service Contracts

- Services share a formal contract
- Services adhere to a communications agreement as defined collectively by one or more service description documents
- For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.

Standardized Service Contracts in SOA Architecture



Open Question Points: 10

Setting

How is the principle of standardized service contracts supported in the Web Services?

Open Question is only supported on Version 2.0 or newer.

Answer

Service contract goals

- Reduce the need for data transformations as two services interact with each other
  - If the services have been implemented as web services, this is achieved when the service contracts use standardized data models e.g. XML schemas (interoperability of the services will also be improved)
- Use a standardized way of expressing service capabilities so that their purpose and ability can be easily understood at design time
  - In Web Services, WSDL is a specification defining how to describe web services in a common XML grammar

WSDL – standardized service contracts support in Web Services

- WSDL is a specification defining how to describe web services in a common XML grammar.
- WSDL describes four critical pieces of data:
  - Interface information describing all publicly available functions
  - Data type information for all message requests and message responses
  - Binding information about the transport protocol to be used
  - Address information for locating the specified service

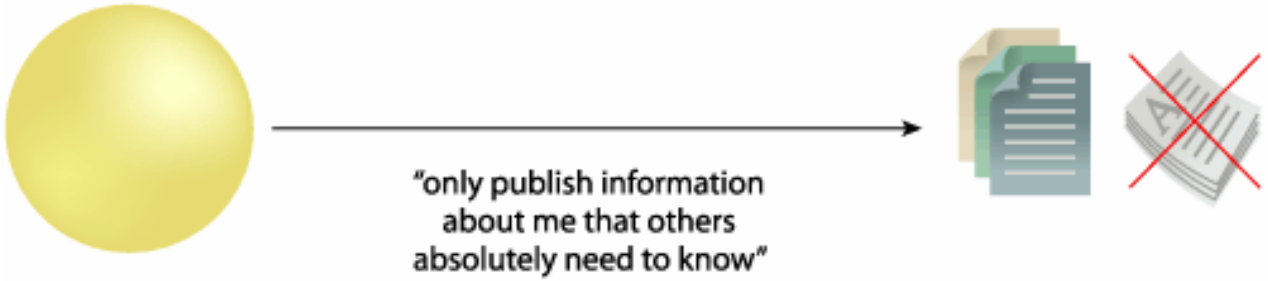
Loose Coupling

- Services are loosely coupled
- Services maintain a relationship that minimizes dependencies and only maintain an awareness of each other
- Services must be designed to interact without the need for tight, cross-service dependencies
- A service is defined solely by an implementation-independent interface
- Services should be able to change their implementation without impacting service consumers

What is loose coupling?

- SOA is an architectural style with characteristics such as *loose coupling, reuse, and simple and composite implementations*
- Loosely coupled services, even if they use incompatible system technologies, can be joined together dynamically to create composite services or disassembled just as easily into their functional components
- Service requesters depend on the interface and not on the service provider's implementation
- Various aspects of service interactions such as time (availability), protocol, message format, language, platform, or location are specified in the service interface separate from the service implementation

Abstraction

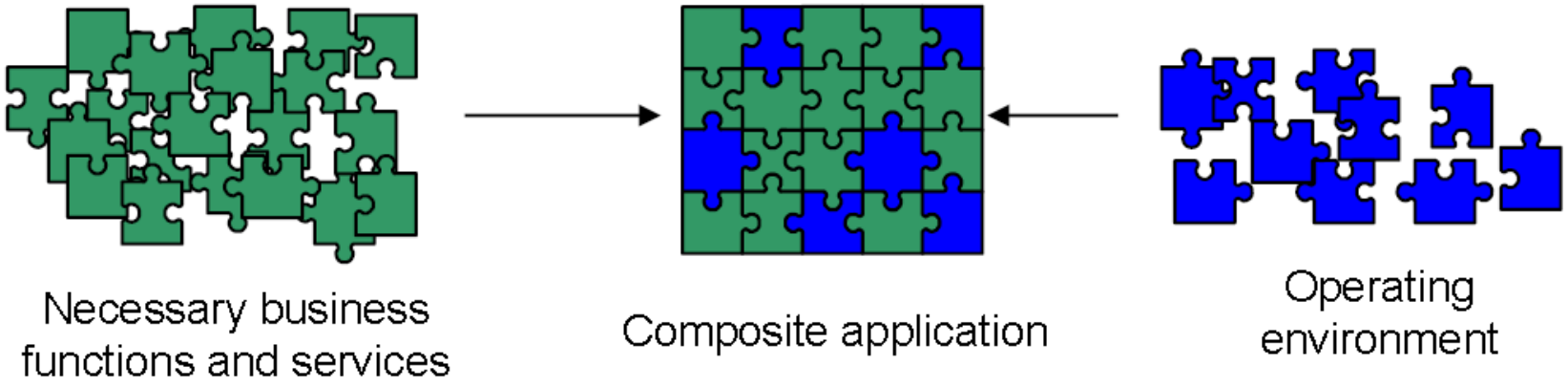


- Services abstract underlying logic
- Beyond what is described in the service contract, services hide logic from the outside world
- The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.

Reusability

- Services are reusable
- Logic is divided into services with the intention of promoting reuse
- Regardless of whether immediate reuse opportunities exist; services are designed to support potential reuse

Service design principles: Reusability



- Concept
  - A service interface should be designed with reuse in mind
  - Anticipate reuse scenarios

Reusability

- 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)

Open Question Points: 10

Setting

We just said that a service interface should be designed with reuse in mind. In Web services, WSDL describes the interface to the system. What can we do to enhance reusability when building applications based on Web Services?

Open Question is only supported on Version 2.0 or newer.

Answer

WSDL Authoring Style Recommendation helps with reusability of Web Services

- Maintain WSDL document in 3 separate parts
  - Data type definitions
  - Abstract definitions
  - Specific service bindings
- Use "*import*" element to import necessary part of WSDL document

Autonomy

- Services are autonomous
- Services have control over the logic they encapsulate
- The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.

Statelessness

- Services are stateless
- Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Service implementations should not hold conversational state across multiple requests.
  - Communicate complete information at each request.
- Each operation should be functionally isolated (separate, independent).

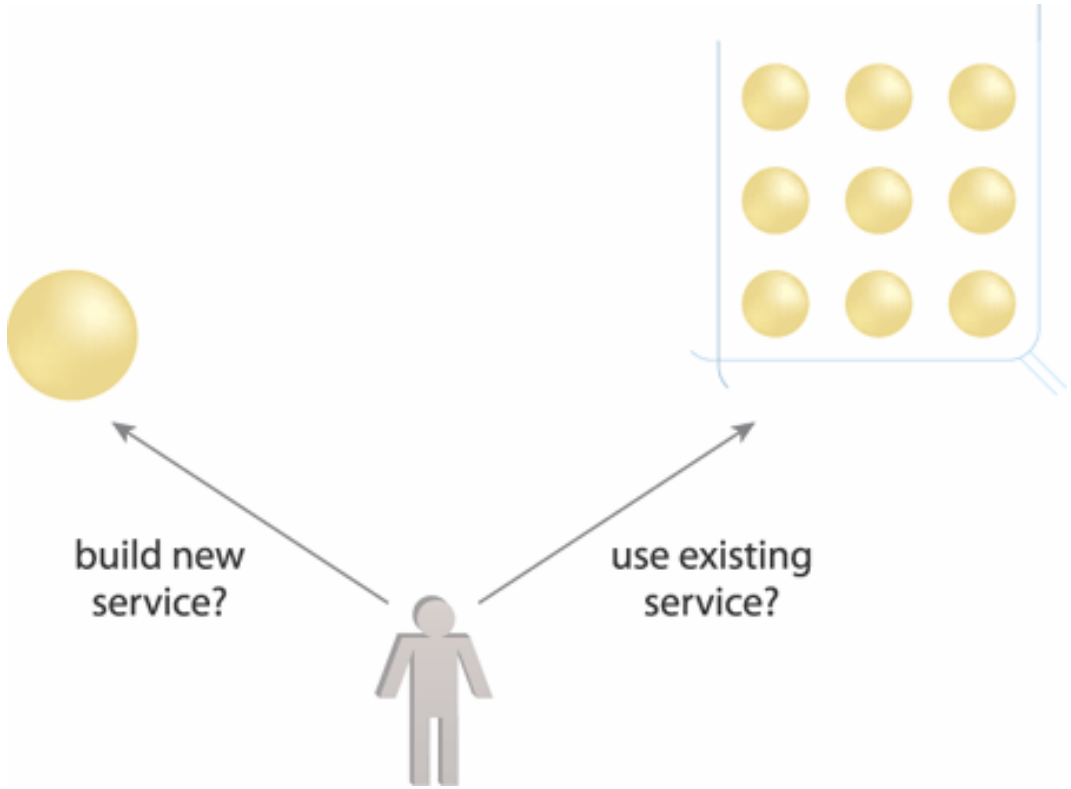
Statelessness--Consequences

- Stateless/connectionless services:
  - Benefit adaptability owing to the independence that exists between successive service requests of a client 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) using pooling and sharing of service instances (client-service independence).
- Stateful services:
  - Hinder adaptability as a consequence of tight dependency (coupling) between a clients successive service requests
    - There is then a need for a specific service instance to fulfil a particular request (client-service affinity).

Discoverability

- Services are discoverable
- Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms
- Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

Discoverability



- *"Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."*
- Service contracts contain appropriate meta data for discovery which also communicates purpose and capabilities to humans
- Store meta data in a service registry or profile documents

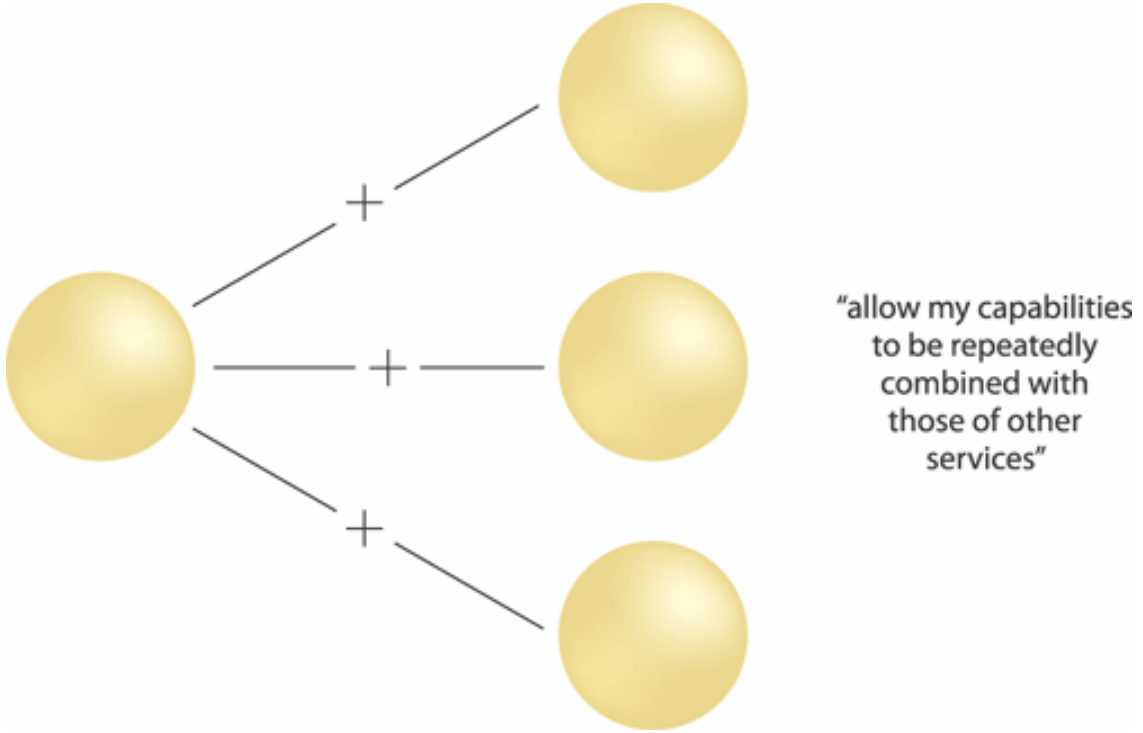
Source: Thomas Erl

Composability

- Services are composable
- Collections of services can be coordinated and assembled to form composite services
- Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.

Composability

- *"Services are effective composition participants, regardless of the size and complexity of the composition."*
- Ensures services are able to participate in multiple compositions to solve multiple larger problems
- Related to Reusability principle
- Service execution should be efficient in that individual processing should be highly tuned
- Flexible service contracts to allow different types of data exchange requirements for similar functions



Source: Thomas Erl

Poll(Anonymous) 1 answer(s) at most

Setting

Which service design principle is in your opinion the most important?

A Services share a formal contract

B Services are loosely coupled

C Services abstract underlying logic

D Services are reusable

E Services are autonomous

F Services are stateless

G Services are discoverable

H Services are composable

Submit