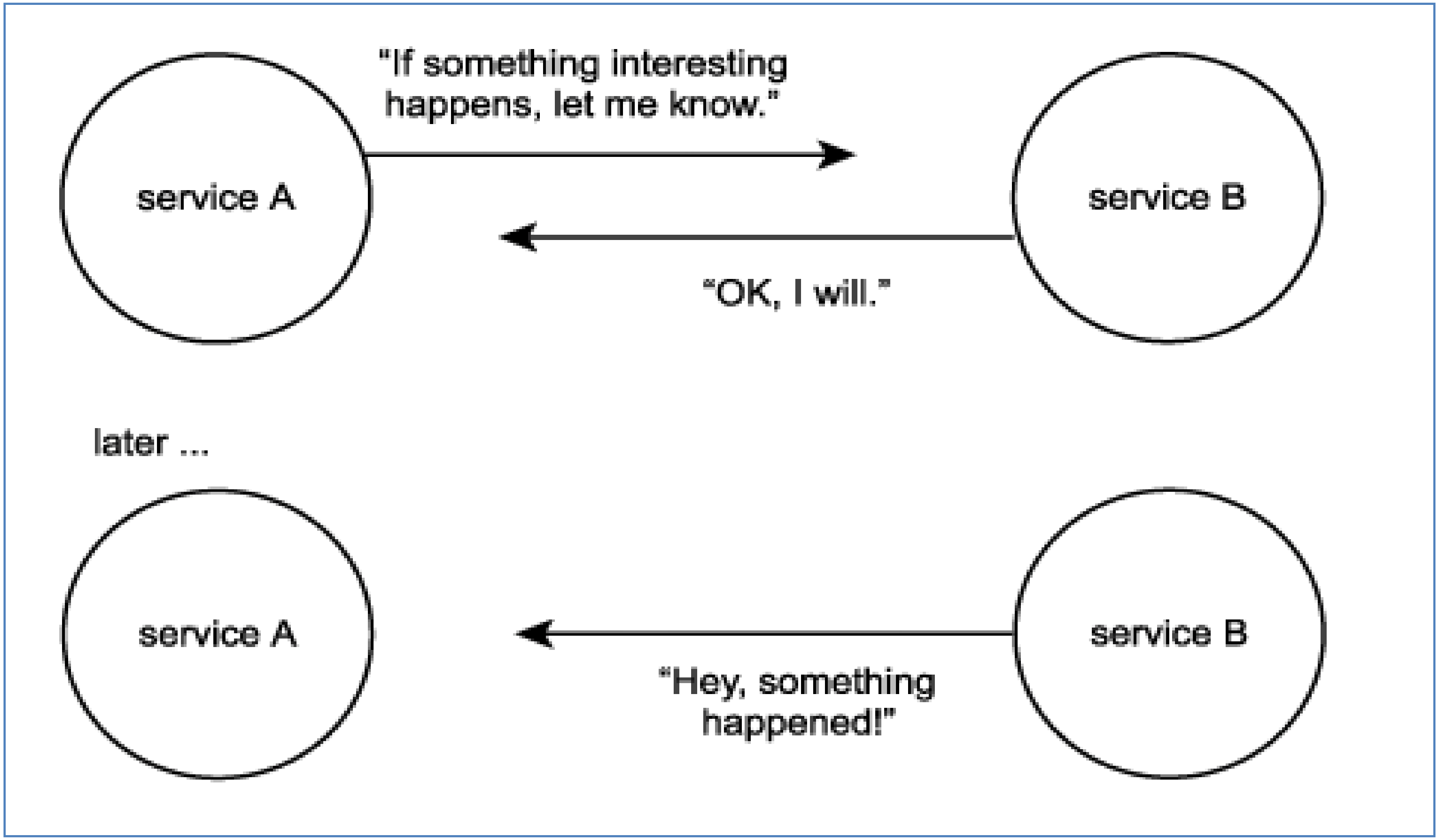
Notification and Eventing


"If something interesting happens, let me know."
service A
service B
"OK, I will."
later ...
service A
service B
"Hey, something happened!"

Publish-and-Subscribe in abstract

- This messaging pattern can be classified as a complex MEP assembled from a series of primitive MEPs.
- It involves a publisher service that makes information categorized by different topics available to registered subscriber services.
- Subscribers can choose which topics they want to register for, either by interacting with the publisher directly or by communicating with a separate broker service.
  - A topic is an item of interest and often is tied to the occurrence of an event.

Publish-and-Subscribe in abstract

- When a new piece of information on a given topic becomes available, a publisher broadcasts this information to all those services that have subscribed to that topic.
- Alternatively, a broker service can be used to perform the broadcast on the publisher's behalf.
  - This decouples the publisher from the subscriber, allowing each to act independently and without knowledge of each other.

One Concept, Two Specifications

- The WS-Notification framework (IBM)
- The WS-Eventing Specification (Microsoft)

WS-Notification

- The notification process typically is tied to an event that is reported by the publisher.
- This event is referred to as a situation.
- Situations can result in the generation of one or more notification messages.
- These messages contain information about the situation, and are categorized according to an available set of topics.
  - Through this categorization, notification messages can be delivered to services that have subscribed to corresponding topics.

WS-Notification

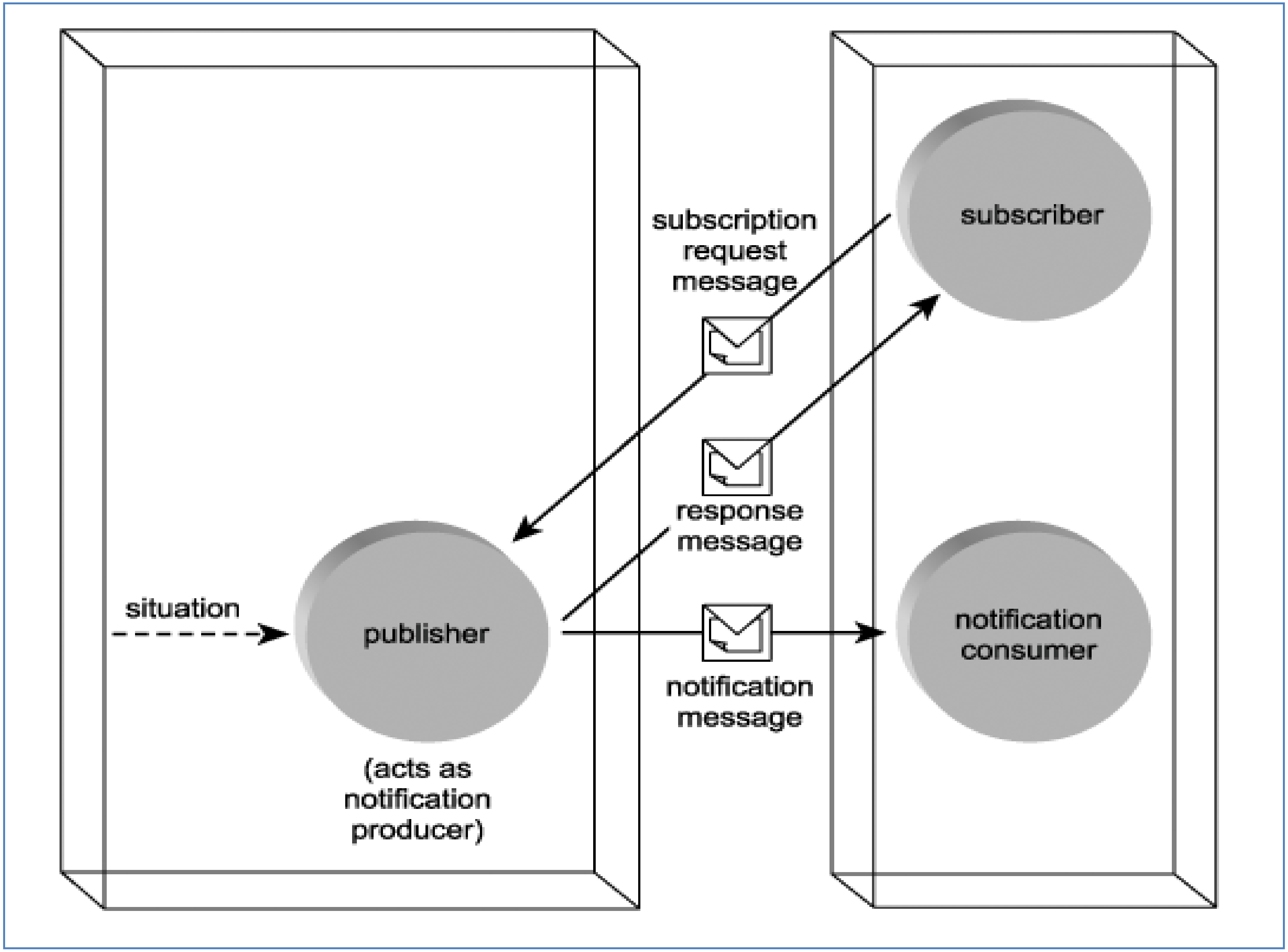
- Basic Terms:
  - Publisher
  - Notification Producer
  - Subscriber
  - Notification Consumer

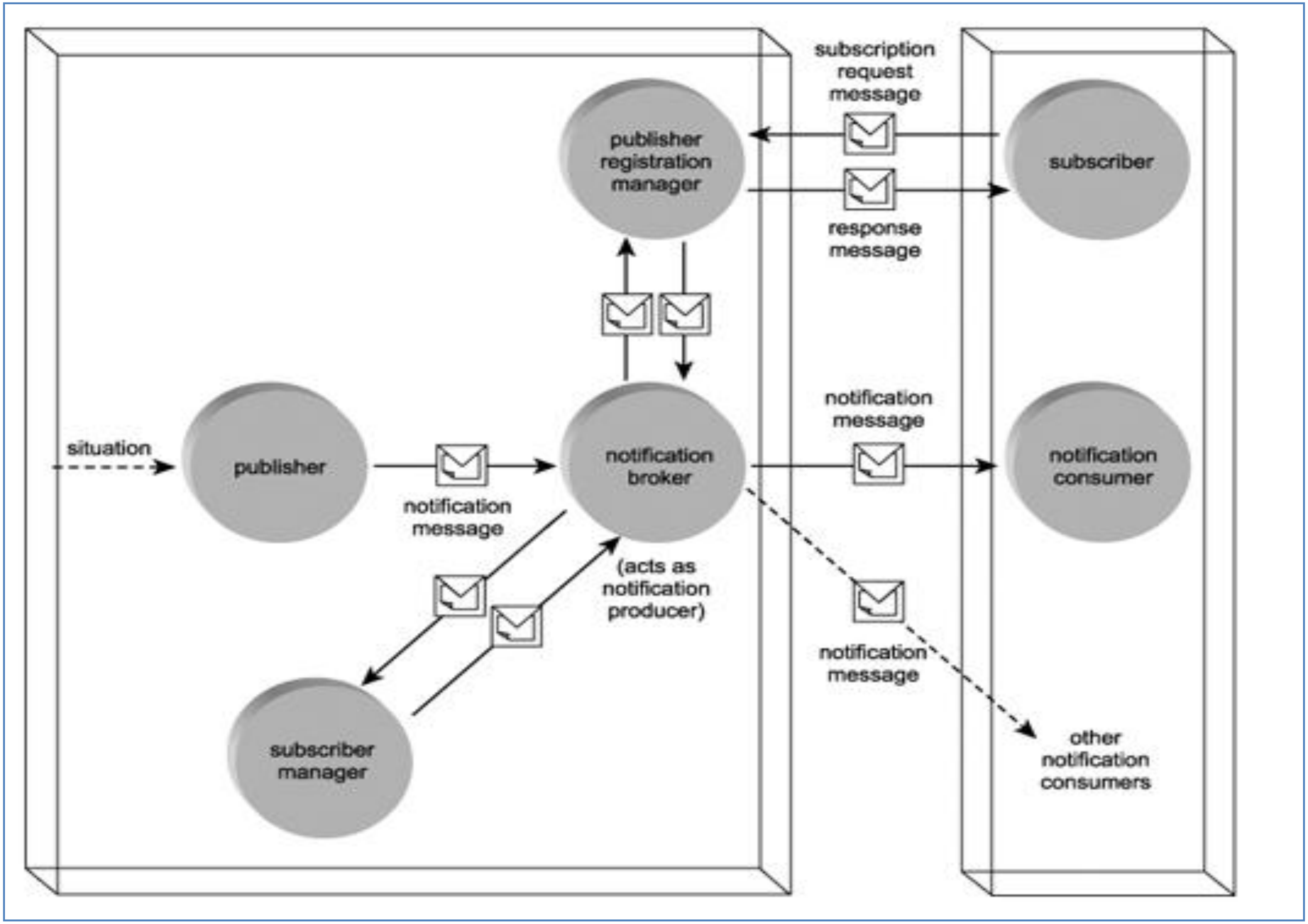
Notification Producers and Publishers

- The term "**Publisher**" represents the part of the solution that responds to situations and is responsible for generating notification messages.
- However, a publisher is not necessarily required to distribute these messages.
- Distribution of notification messages is the task of the "**Notification Producer**".
  - This service keeps track of subscriptions and corresponds directly with subscribers.
  - It ensures that notification messages are organized by topic and delivered accordingly.

Notification Consumers and Subscribers

- A "**Subscriber**" is the part of the application that submits the subscription request message to the notification producer.
- This means that the subscriber is not necessarily the recipient of the notification messages transmitted by the notification producer.
  - The recipient is the "**Notification Consumer**", the service to which the notification messages are delivered


subscription request message
subscriber
response message
situation
publisher
notification message
notification consumer
(acts as notification producer)



subscription request message
publisher registration manager
subscriber
response message
situation
publisher
notification message
notification broker
(acts as notification producer)
notification message
notification consumer
notification message
other notification consumers
subscriber manager

Notification Broker, Publisher Registration Manager, and Subscription manager

- **The Notification Broker**
  - A Web service that performs the role of the notification producer.
  - This isolates the publisher from any contact with subscribers.

Notification Broker, Publisher Registration Manager, and Subscription manager

- **The Publisher Registration Manager**
  - A Web service that provides an interface for subscribers to search through and locate items/topics available for registration.

Notification Broker, Publisher Registration Manager, and Subscription manager

- **The Subscription Manager**
  - A Web service that allows notification producers to access and retrieve required subscriber information for a given notification message broadcast.

The WS-Eventing Specification

- WS-Eventing addresses publish-and-subscribe requirements by focusing on an event-oriented messaging model.

- When an event related to one Web service occurs, any other services that have expressed interest in the event are subsequently notified.

The WS-Eventing Specification

- **Basic Terms:**
  - Event Source
  - Event Sink
  - Subscriber

Event Sources

- The term ~~“Publisher"~~ is never actually mentioned in the WS-Eventing specification.
- Instead, its role is assumed by a Web service, known as the **Event Source**.
- This part of the eventing architecture is responsible for both receiving subscription requests and for issuing corresponding notification messages that report information about occurred events.

Event Sinks and Subscribers

- On the subscription end of the eventing model, separate Web services manage the processing of notification and subscription messages.
- An **"Event Sink"** is a service designed to consume (receive) notification messages from the event source.
- **"Subscribers"** are services capable of issuing various types of subscription requests.

Subscription Managers

- An **Event Source**, by default, assumes the responsibility of managing subscriptions and transmitting notifications.
- In *high volume environments* it may be desirable to split these roles into separate services.
- To manage the demands on the Event Source, *intermediate services*, known as "**Subscription Managers**", optionally can be used to distribute publisher-side processing duties.

Notification Messages

- When an event occurs, it is reported by the event source via the issuance of a notification.
- These are standard SOAP messages that contain WS-Eventing-compliant headers to convey event details.

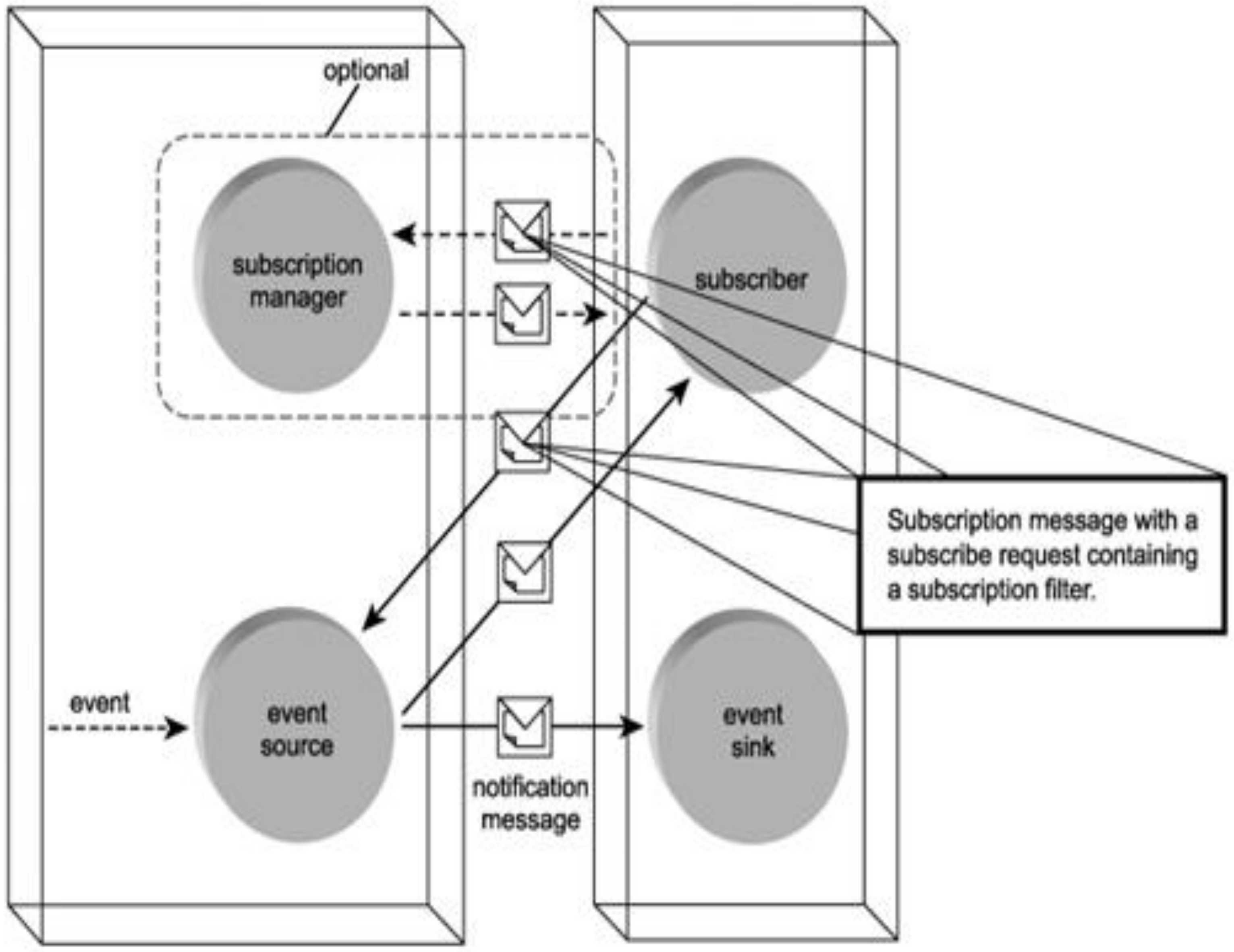
- WS-Eventing also allows for an expiry date to be attached to subscriptions.
  - This requires that subscribers issue renewal requests for the subscription to continue.

Subscription End Messages

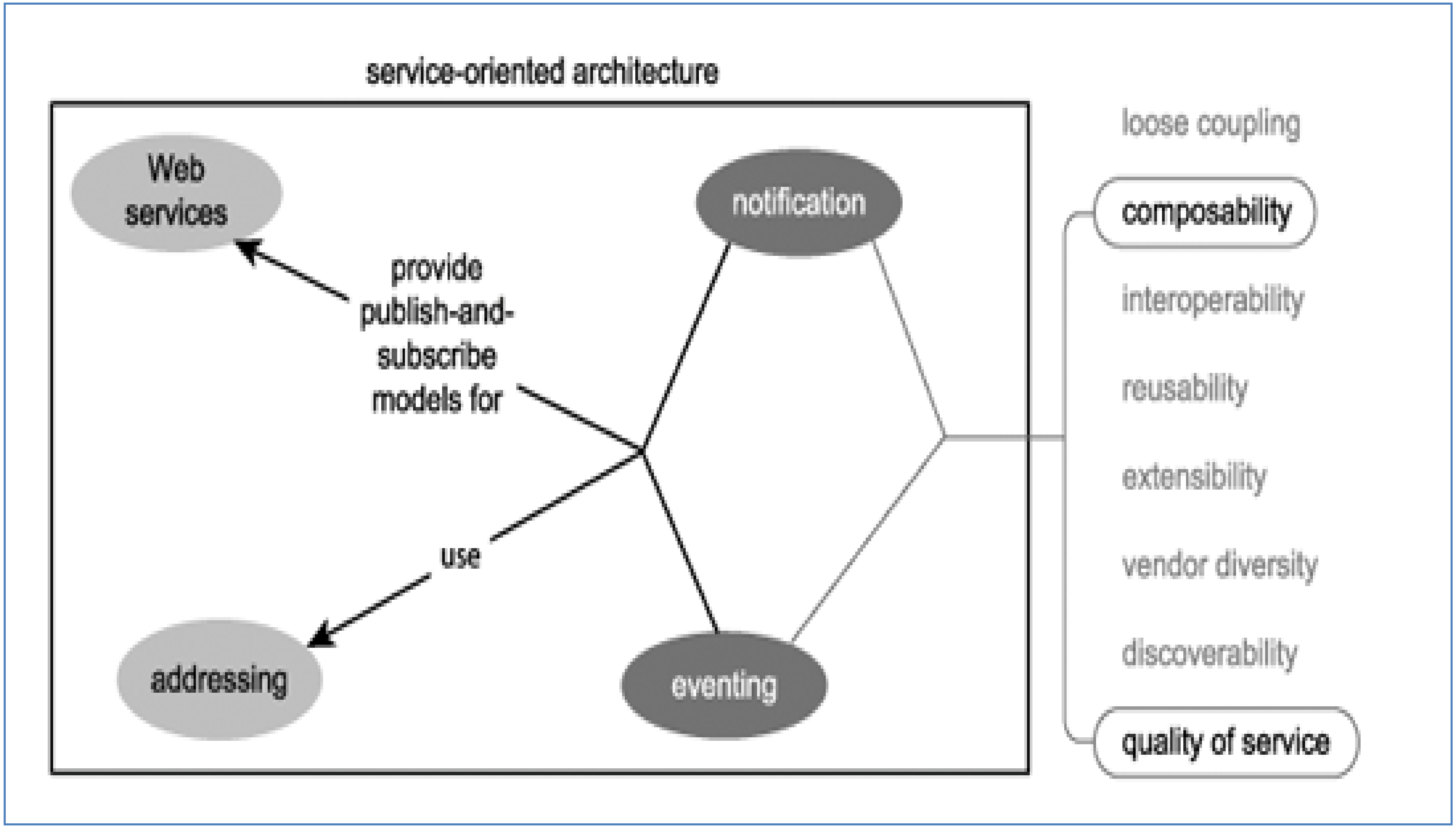
- If a subscription is left to expire, it is the event source that often is expected to send a special type of notification to the corresponding event sink, called a subscription end message.

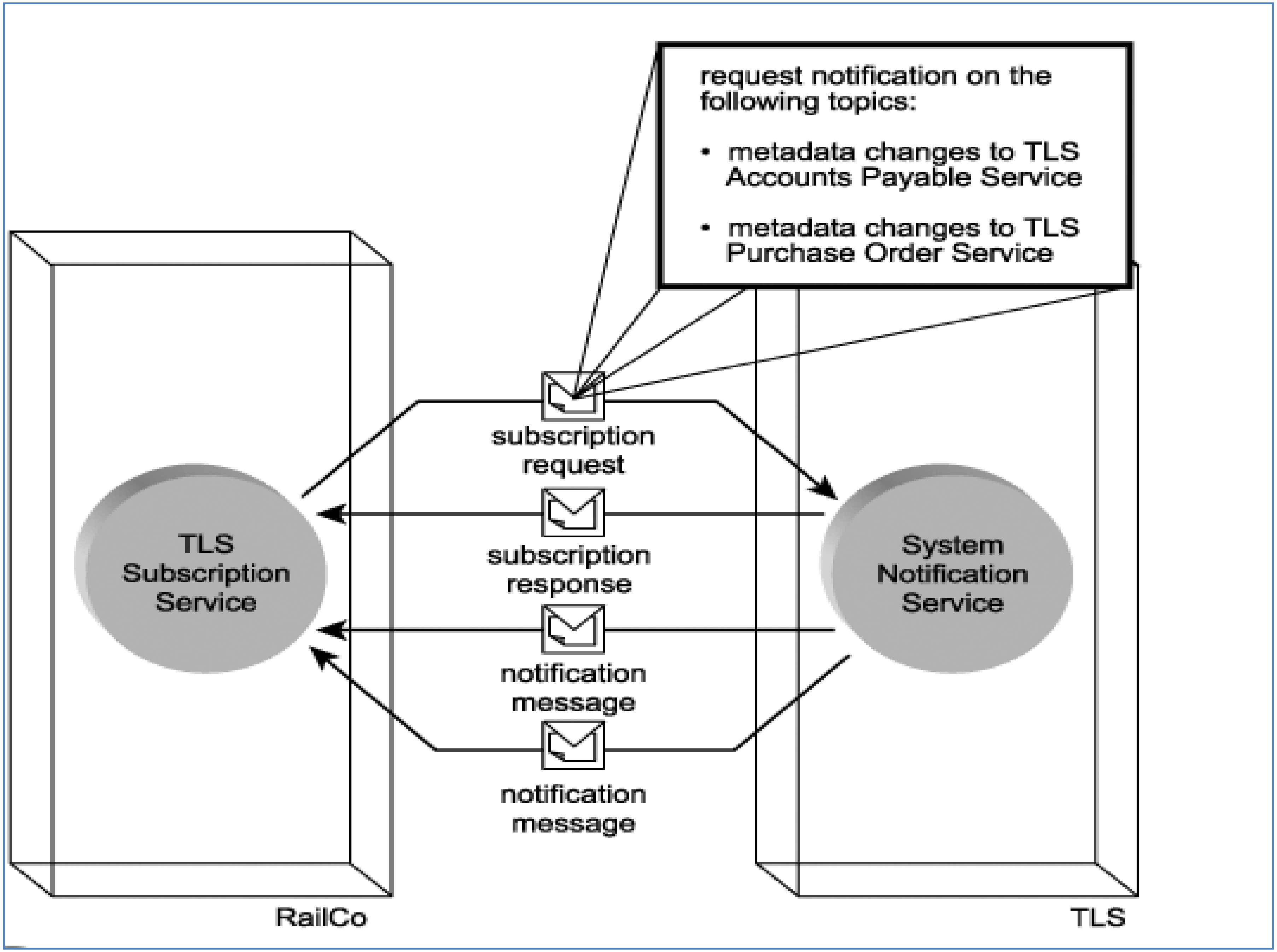
Types of Messages

- The following specific requests are supported:
  - Subscribe Requests for a new subscription to be created.
  - Unsubscribe Requests for an existing subscription to be canceled.
  - Renew Requests for an existing subscription scheduled to expire be renewed.
  - GetStatus Requests for the status of a subscription to be retrieved.


optional
subscription manager
subscriber
Subscription message with a subscribe request containing a subscription filter.
event
event source
event sink
notification message

Notification, Eventing and SOA




request notification on the following topics:
• metadata changes to TLS Accounts Payable Service
• metadata changes to TLS Purchase Order Service
subscription request
subscription response
notification message
notification message
TLS Subscription Service
System Notification Service
RailCo
TLS

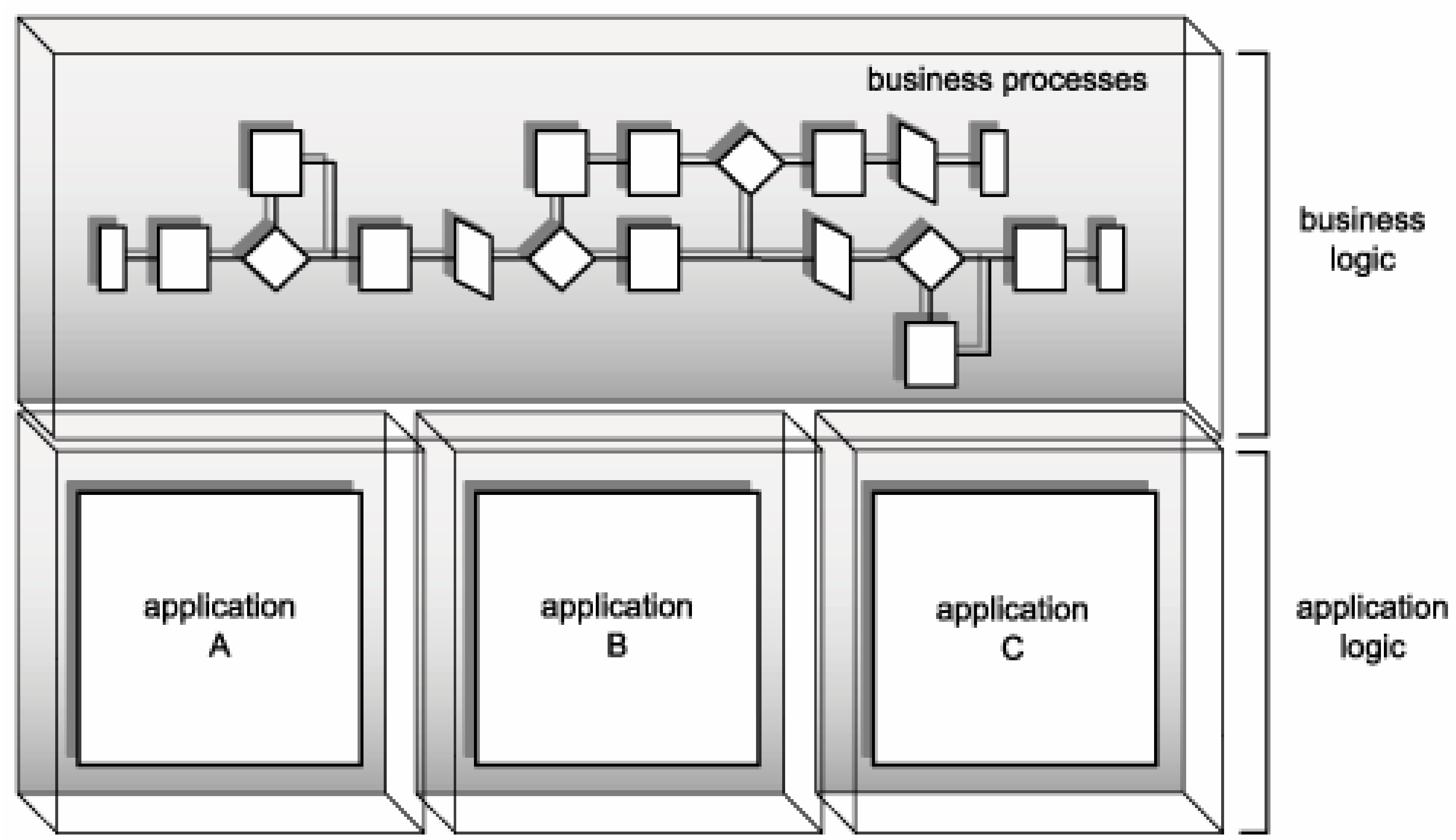
So far,

- The traditional publish-and-subscribe messaging model can be implemented with the WS-Notification framework or the WS-Eventing specification.
- WS-Notification consists of the WS-BaseNotification, WS-Topics, and WS-BrokeredNotification specifications that collectively establish a subscription and notification system.
- The WS-Eventing specification provides similar functionality but is based on a moderately different architecture.
- Notification and eventing realize the popular publish-and-subscribe messaging model within SOA.

How can we build Web services that are truly service-oriented?

Which are the primitive components of an SOA ?

Service-Orientation and The Enterprise



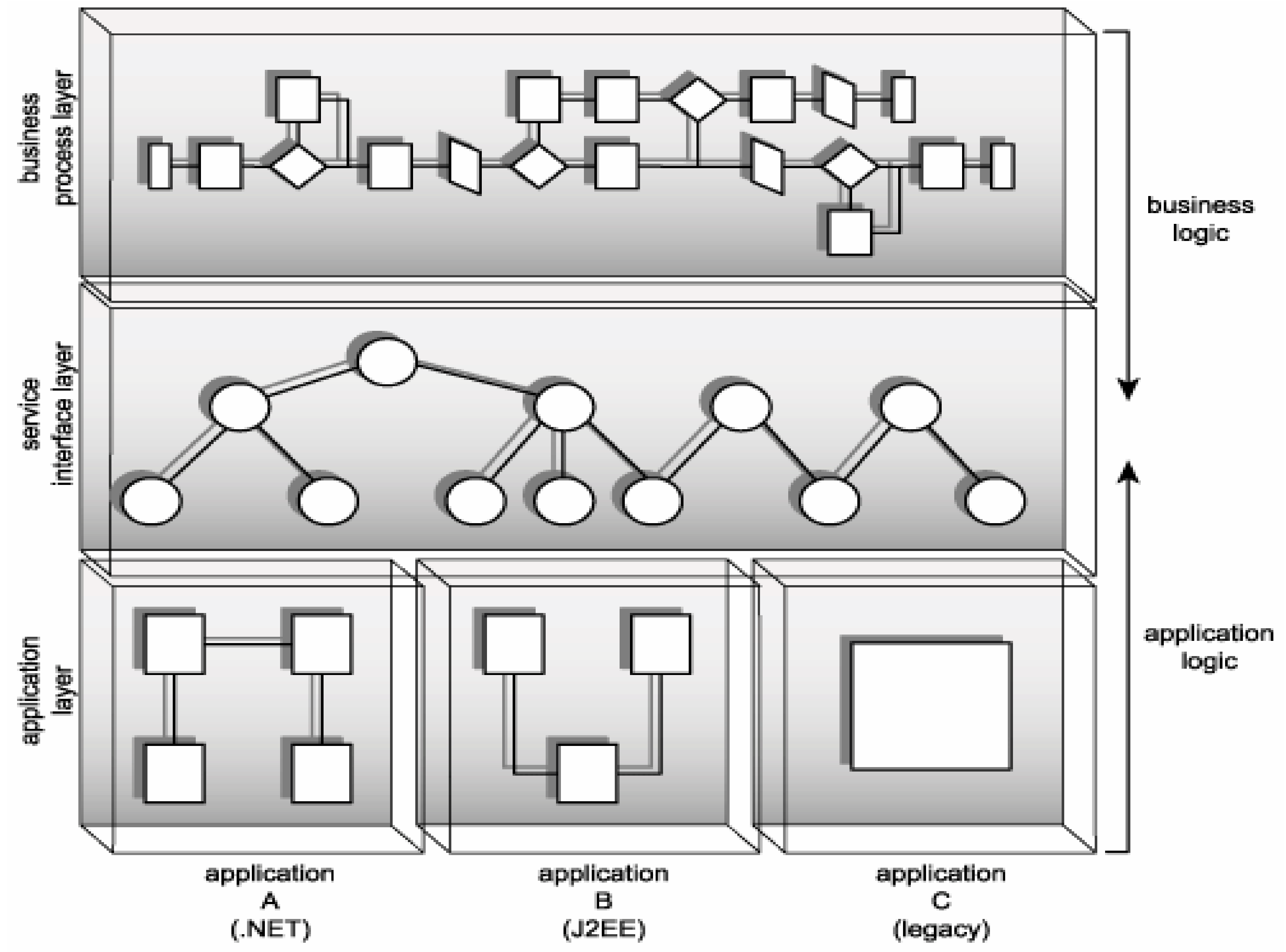
The Business and Application logic

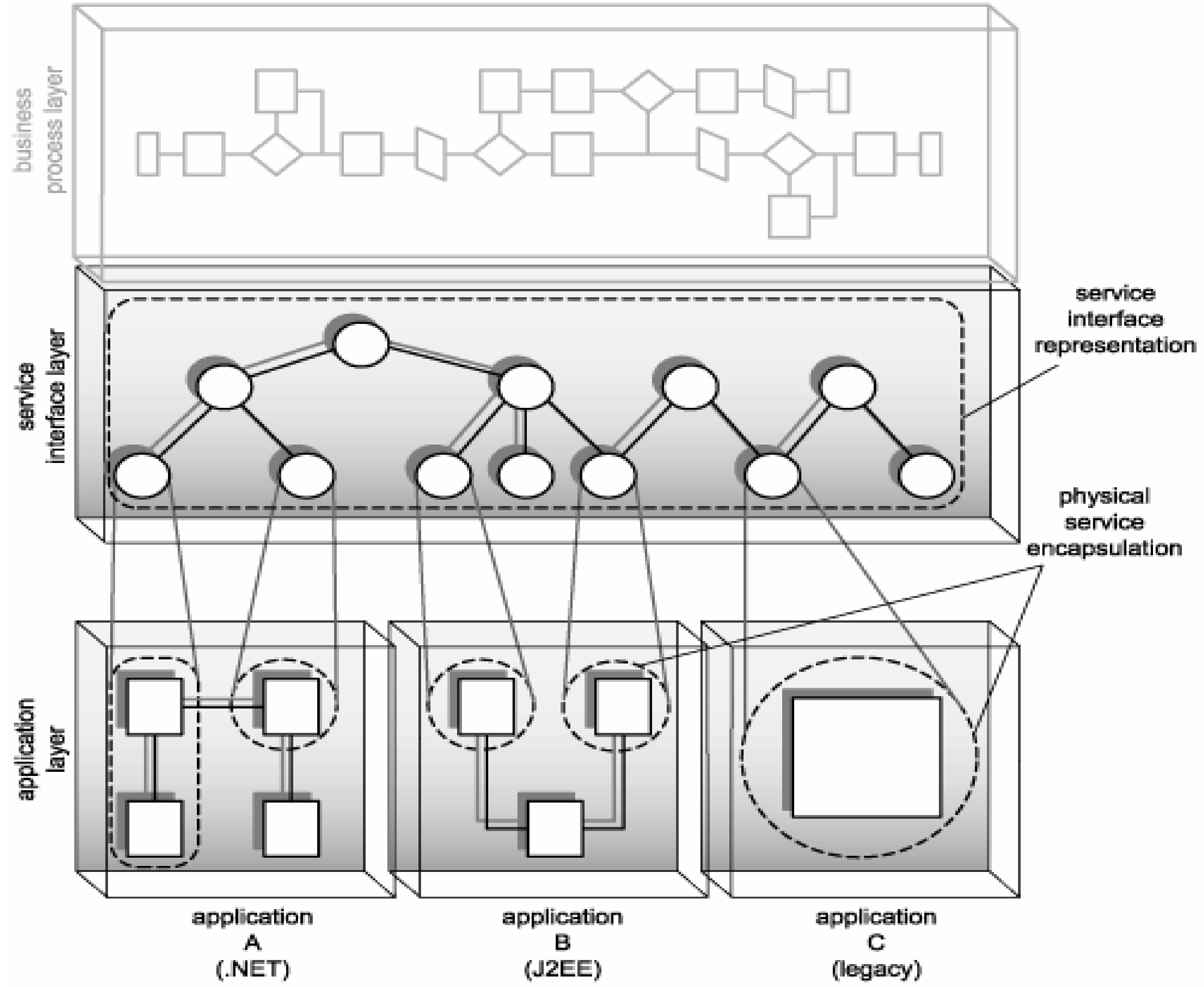
- **Business logic** is a documented implementation of the business requirements that originate from an enterprise's business areas.
- Business logic is generally expressed as requirements, along with any associated constraints, dependencies, and outside influences.

The Business and Application logic

- **Application logic** is an automated implementation of business logic organized into various technology solutions.

- Application logic expresses business process workflows through purchased or custom-developed systems within the confines of an organization's IT infrastructure, security constraints, technical capabilities, and vendor dependencies.


business process layer
service interface layer
application layer
business logic
application logic
application A (.NET)
application B (J2EE)
application C (legacy)

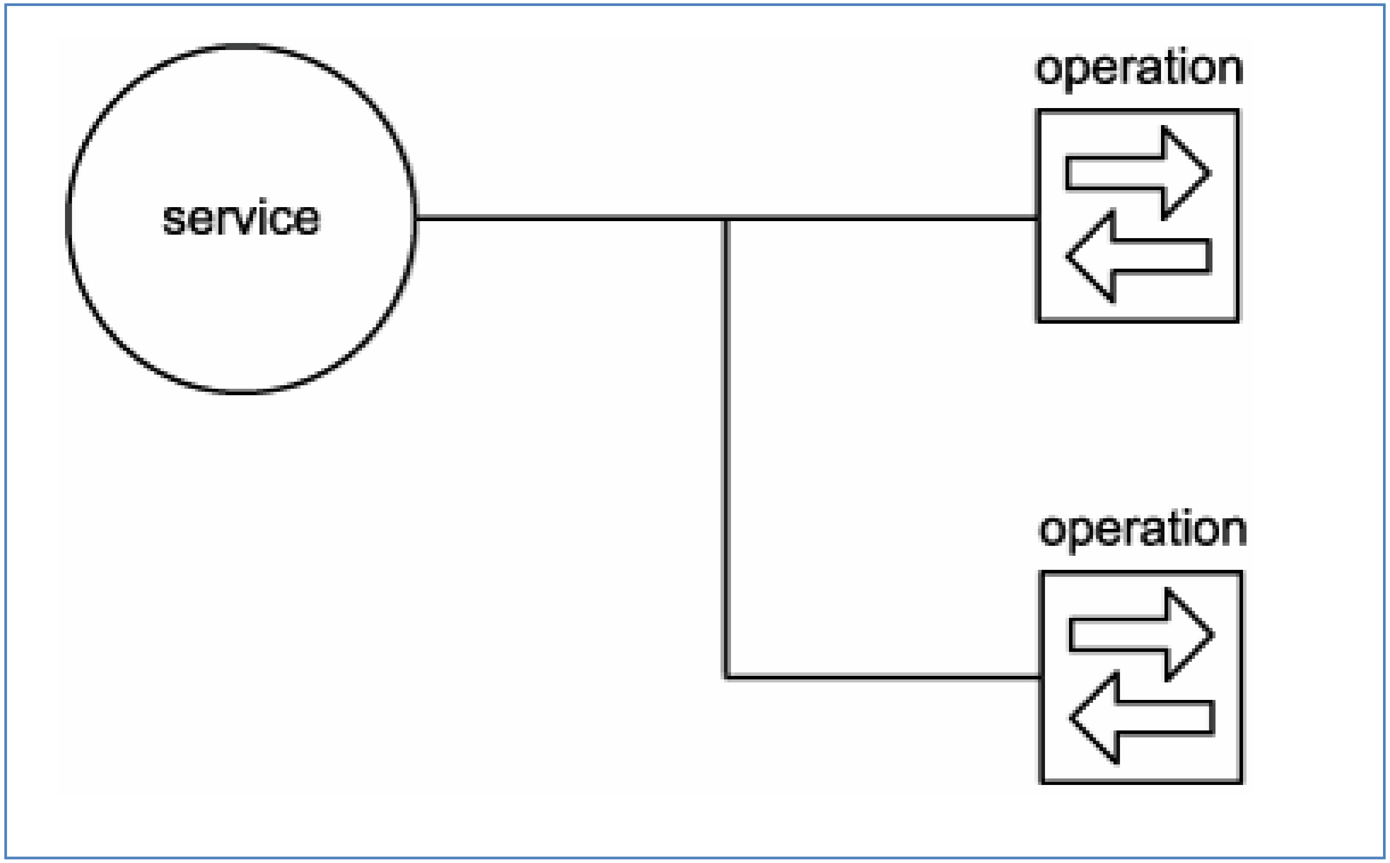

business process layer
service interface layer
service interface representation
physical service encapsulation
application layer
application A (.NET)
application B (J2EE)
application C (legacy)

Logical components of the Web Services Framework

- Logical components of the Web services framework include:
  - Services
  - Operations
  - Messages
  - Activities

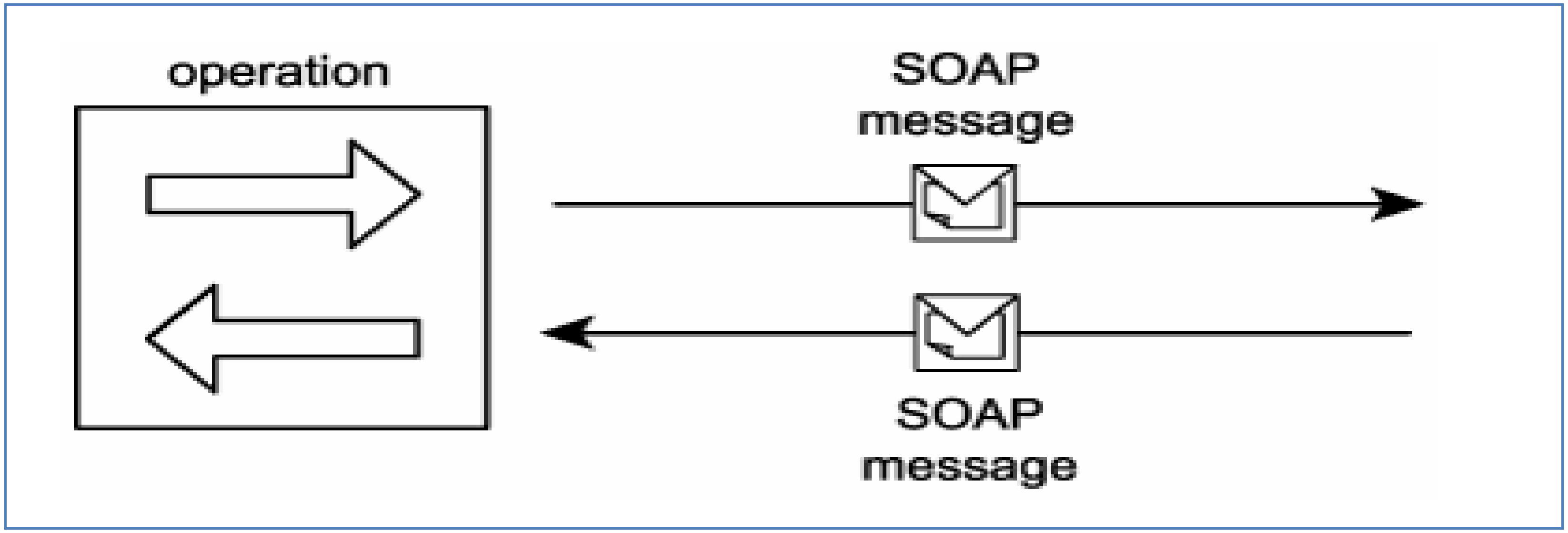
Cont.

- Each Web service contains one or more operations.



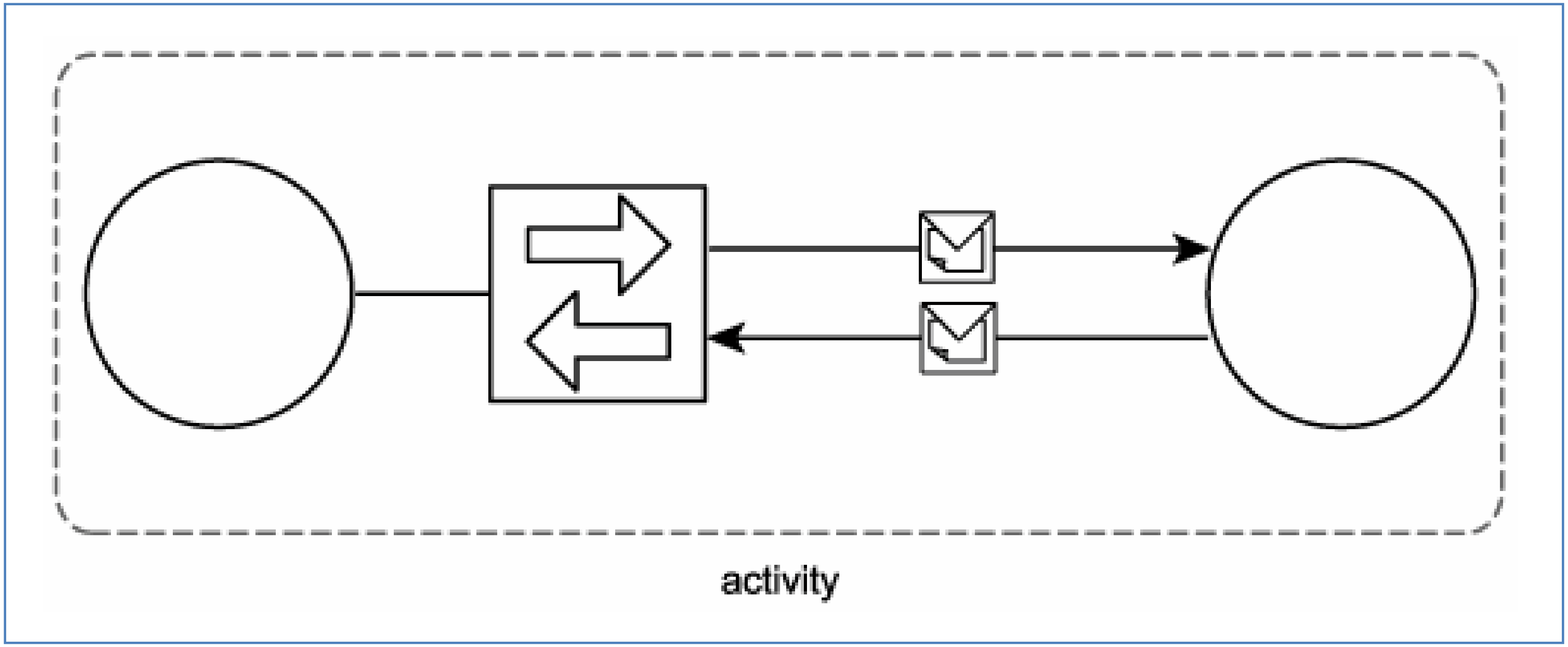
Cont.

- Each operation governs the processing of a specific function the Web service is capable of performing.
- The processing consists of sending and receiving SOAP messages



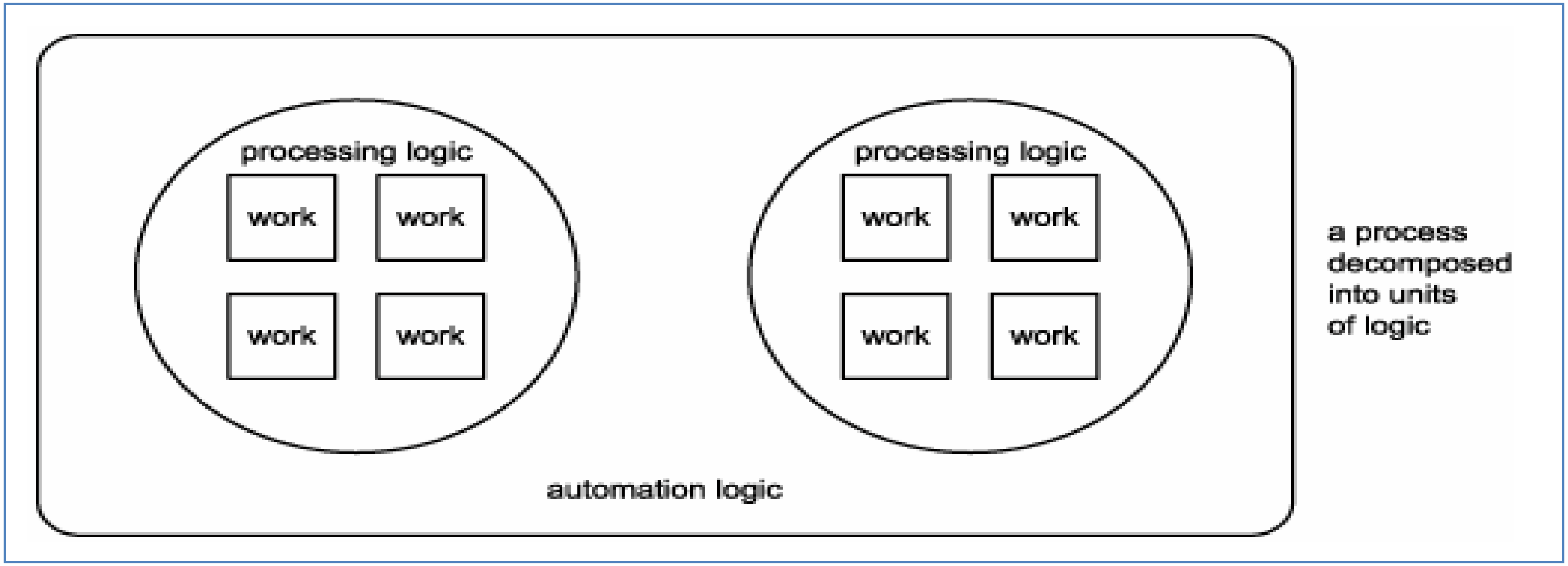
Cont.

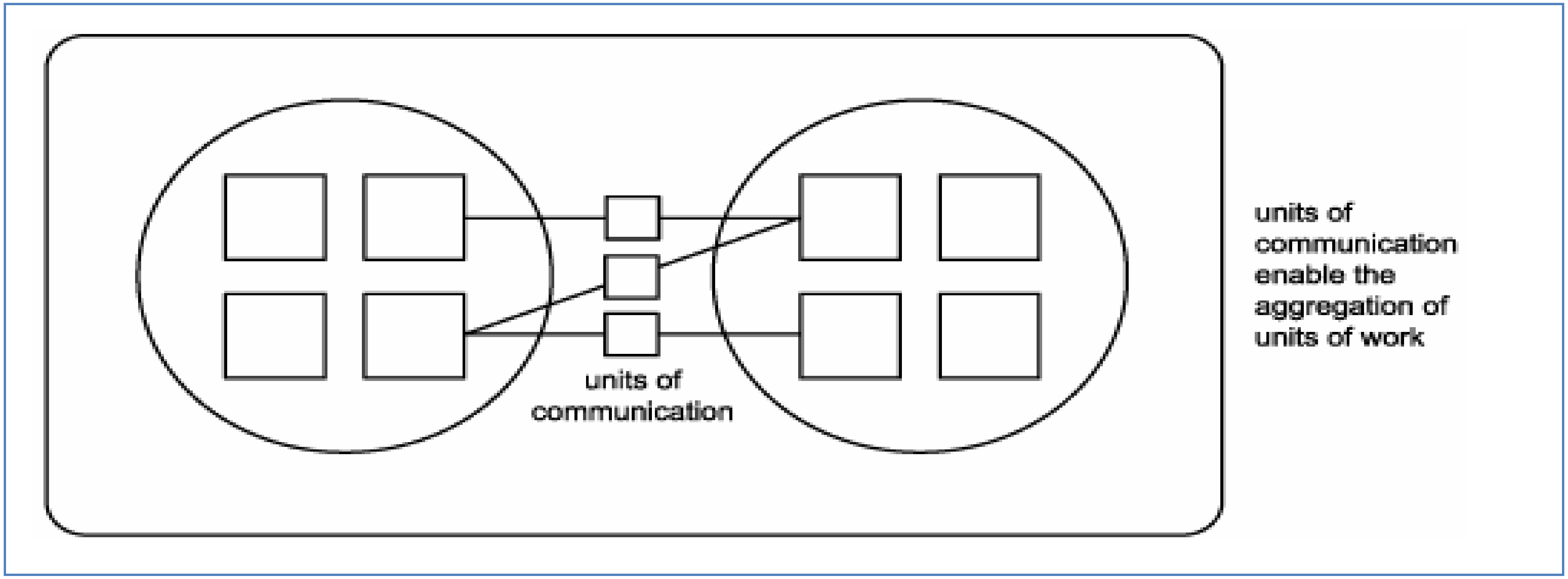
- By composing these parts, Web services form an **activity** through which they can collectively automate a task.



Logical components of Automation Logic

- SOAP messages → messages → units of communication
- Web service operations → operations → units of work
- Web services → services → units of processing logic (collections of units of work)
- Processes → Activities (and process instances) →units of automation logic (coordinated aggregation of units of work)


processing logic
work
work
work
work
processing logic
work
work
work
work
automation logic
a process decomposed into units of logic


units of communication
units of communication enable the aggregation of units of work

Components of an SOA

- A **message** represents the data required to complete some or all parts of a unit of work.

- An **operation** represents the logic required to process messages in order to complete a unit of work.

Components of an SOA

- A **service** represents a logically grouped set of operations capable of performing related units of work.

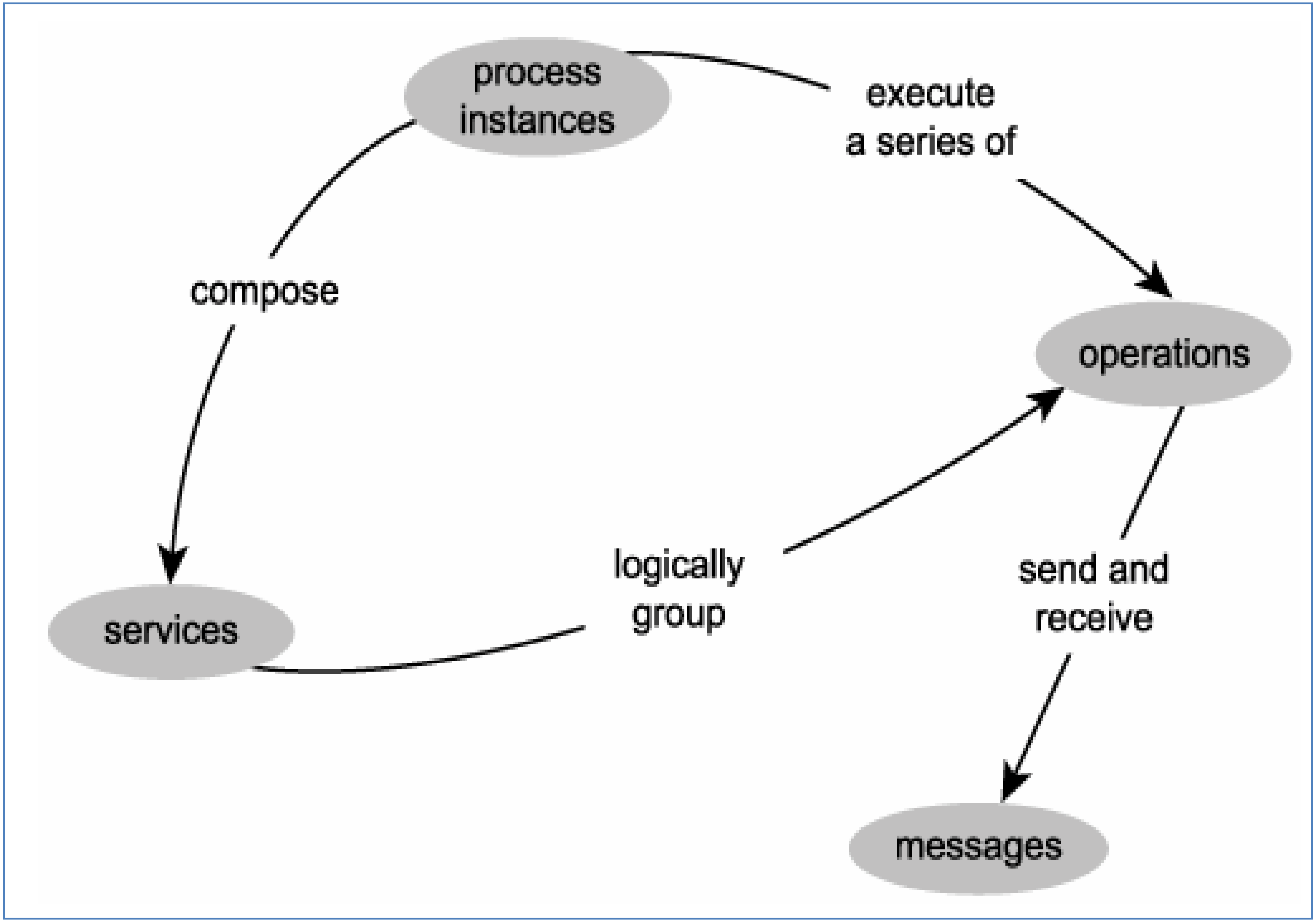
- A **process** contains the business rules that determine which service operations are used to complete a unit of automation.
  - In other words, a process represents a large piece of work that requires the completion of smaller units of work.

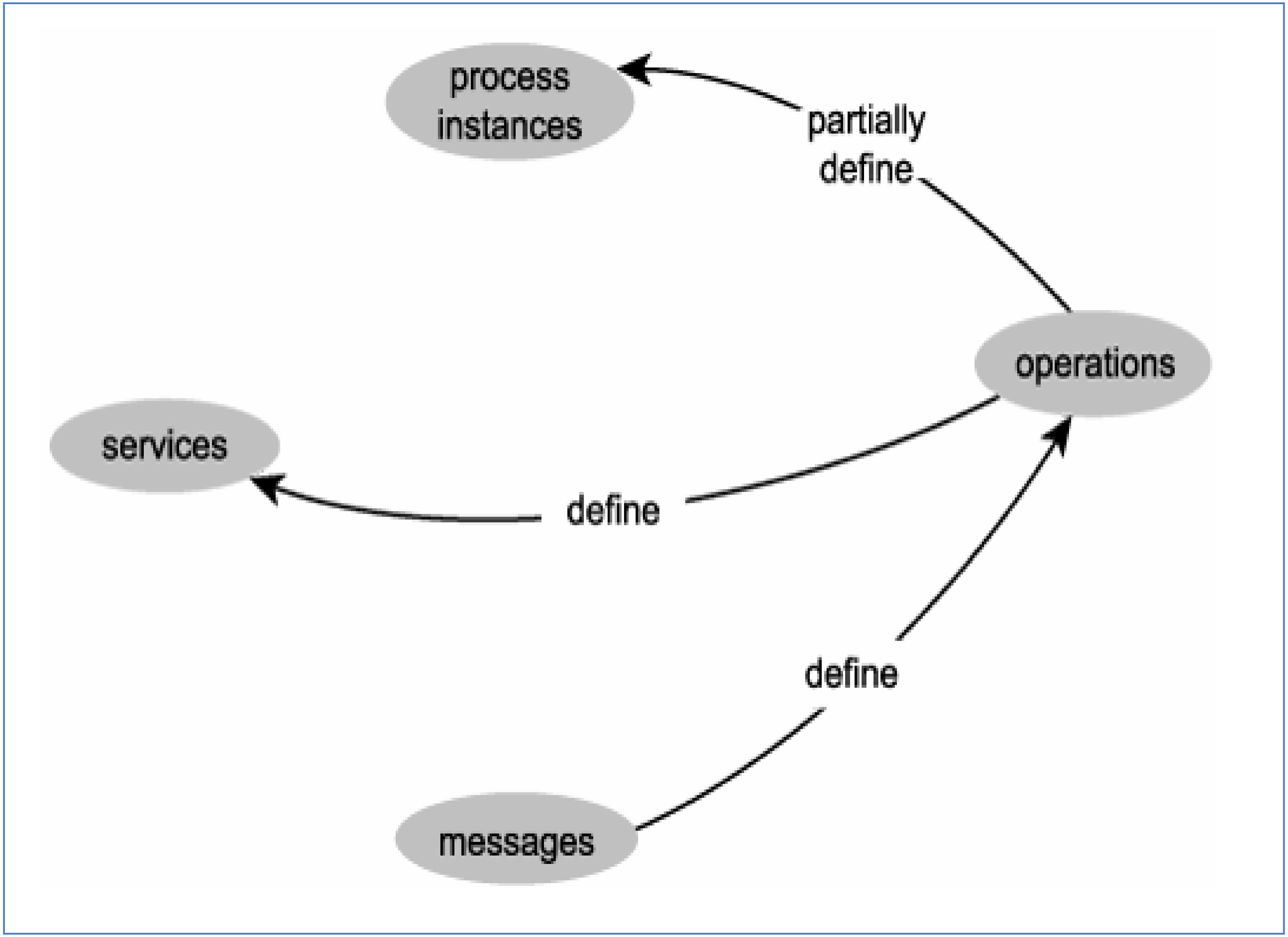
How components in an SOA Inter-relate?

- An operation sends and receives messages to perform work.
  - An operation is therefore mostly defined by the messages it processes.
- A service groups a collection of related operations.
  - A service is therefore mostly defined by the operations that comprise it.

How components in an SOA inter-relate

- A process instance can compose services.
  - A process instance is not necessarily defined by its services because it may only require a subset of the functionality offered by the services.
  - A process instance invokes a unique series of operations to complete its automation.
- Every process instance is therefore partially defined by the service operations it uses.


process instances
execute a series of
operations
compose
services
logically group
send and receive
messages


process instances
partially define
operations
services
define
define
messages

So far,

- The logical parts of an SOA can be mapped to corresponding components in the basic Web services framework.
- By viewing a service-oriented solution as a unit of automation logic, we establish that SOA consists of a sophisticated environment that supports a highly modularized separation of logic into differently scoped units.
- SOA further establishes specific characteristics, behaviors, and relationships among these components that provide a predictable environment in support of service-orientation.

So far,

- Service abstraction, composability, loose coupling, and the need for service contracts are native characteristics of Web services that are in full alignment with the corresponding principles of service-orientation.
- Service reusability, autonomy, statelessness, and discoverability are not automatically provided by Web services. Realizing these qualities requires a conscious modeling and design effort.