Contemporary SOA, Part I

605.702 Service Oriented Architecture
Johns-Hopkins University

Lecture 3 Goals

- Part one of two to become familiar with the concepts behind Contemporary SOA
  - Message Exchange Patterns (MEP's)
  - Linking business processes to services and SOA
  - Control Services: Coordination, Orchestration, Choreography

Session #3 Today's Agenda

- Required Reading (to be read before class)
  - Erl Chapter 6: Web Services and Contemporary SOA I
    - Activity Management and Composition
- Today's Lecture
  - Ch 6: Web Services, Activity Management and Composition

Chapter 6: Contemporary SOA Part I

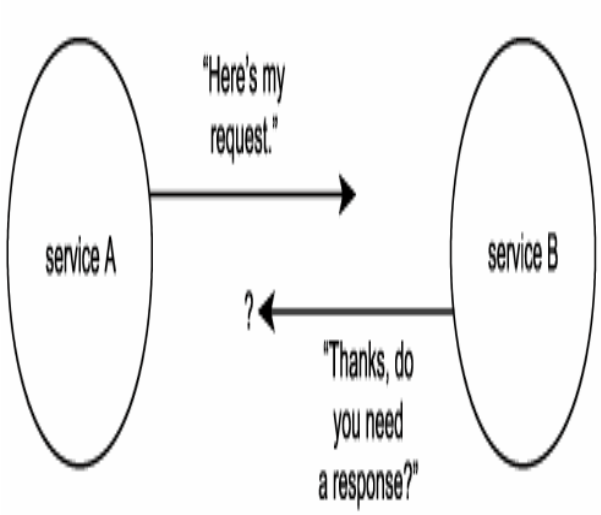
- 6.1 Message Exchange Patterns
- 6.2 Service Activity
- 6.3 Coordination
- 6.4 Atomic Transactions
- 6.5 Business Activities
- 6.6 Orchestration
- 6.7 Choreography

6.1 Message Exchange Patterns (MEP's)

- How services can share and cooperate in processing messages
  - Primitive MEP's
    - Request response
    - Fire and forget
      - Single destination, multi-cast or broadcast
  - Complex MEP's
    - Based on Primitive MEP's
    - Example: Publish and subscribe
  - WSDL & MEP's:
    - Request-response, solicit-response, one-way, notification

Message exchange patterns

Figure 6.2. Not all message exchanges require both requests and responses.



- Message exchange patterns (MEPs) represent a set of templates
- The most common example is a request and response pattern.
- Many MEPs have been developed, each addressing a common message exchange requirement.

Request-response

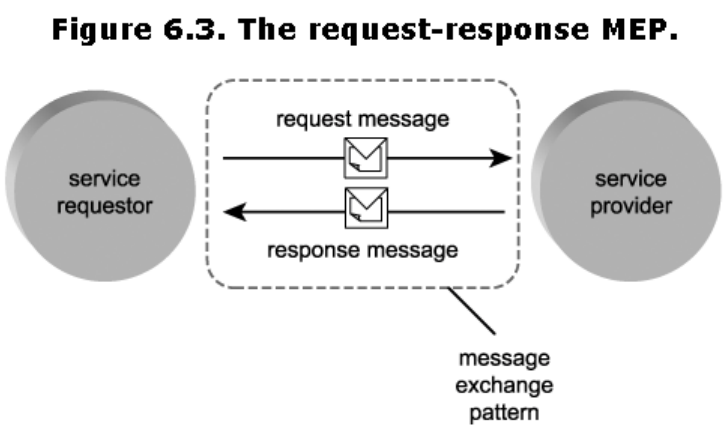
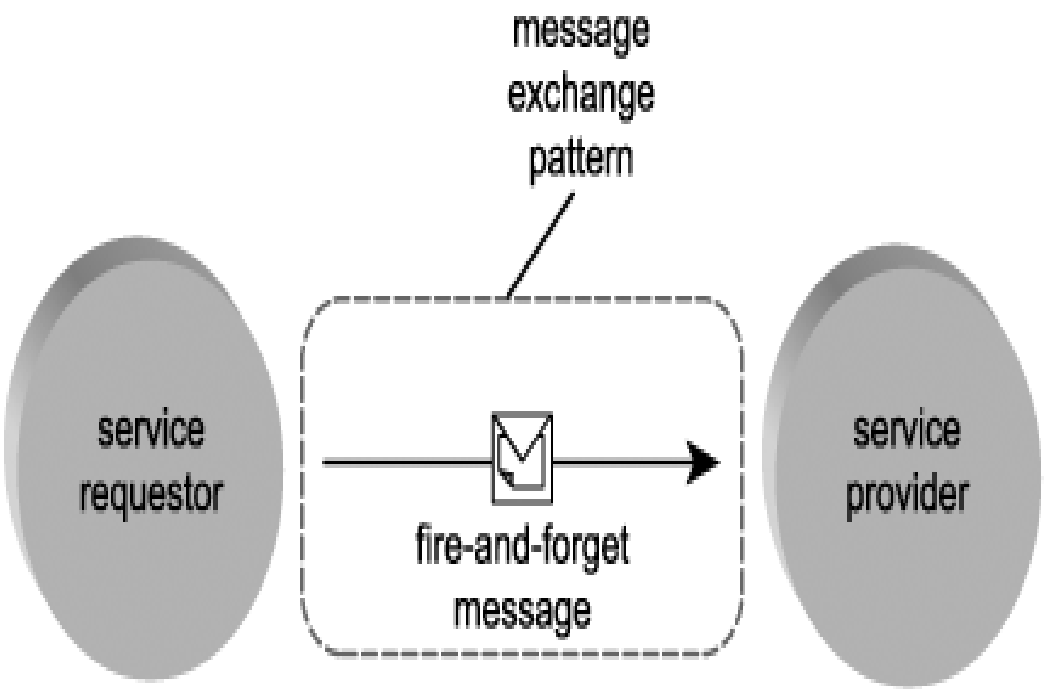


Figure 6.3. The request-response MEP.

- The request-response MEP (Figure 6.3) establishes a simple exchange in which a message is first transmitted from a source (service requestor) to a destination (service provider). Upon receiving the message, the destination (service provider) then responds with a message back to the source (service requestor).

Fire-and-forget

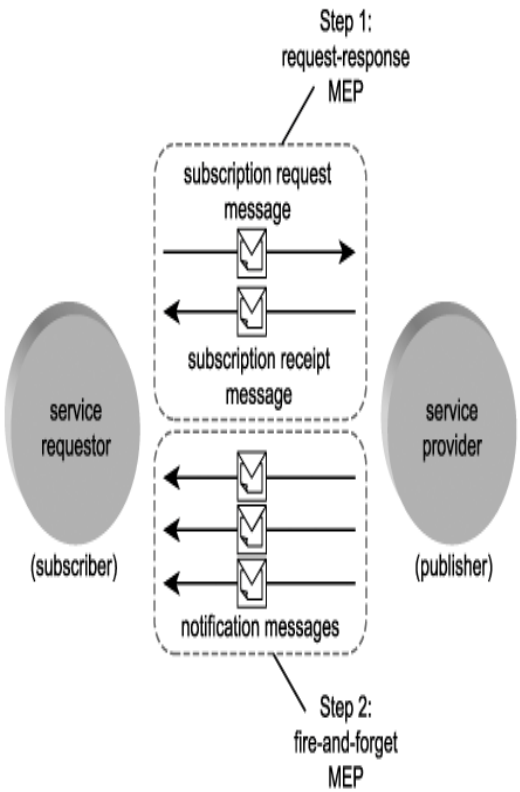
Figure 6.5. The fire-and-forget MEP.



- A number of variations of the fire-and-forget MEP exist, including:
  - The single-destination pattern, where a source sends a message to one destination only.
  - The multi-cast pattern, where a source sends messages to a predefined set of destinations.
  - The broadcast pattern, which is similar to the multi-cast pattern, except that the message is sent out to a broader range of recipient destinations.
- The fundamental characteristic of the fire-and-forget pattern is that a response to a transmitted message is not expected.

Complex MEPs

Figure 6.7. The publish-and-subscribe messaging model is a composite of two primitive MEPs.



A classic example is the publish-and-subscribe model..

The publish-and-subscribe pattern introduces new roles for the services involved with the message exchange. They now become **publishers** and **subscribers**, and each may be involved in the transmission and receipt of messages.

- This asynchronous MEP accommodates a requirement for a publisher to make its messages available to a number of subscribers interested in receiving them.

The steps involved are generally similar to the following:

- **Step 1.** The subscriber sends a message to notify the publisher that it wants to receive messages on a particular topic.

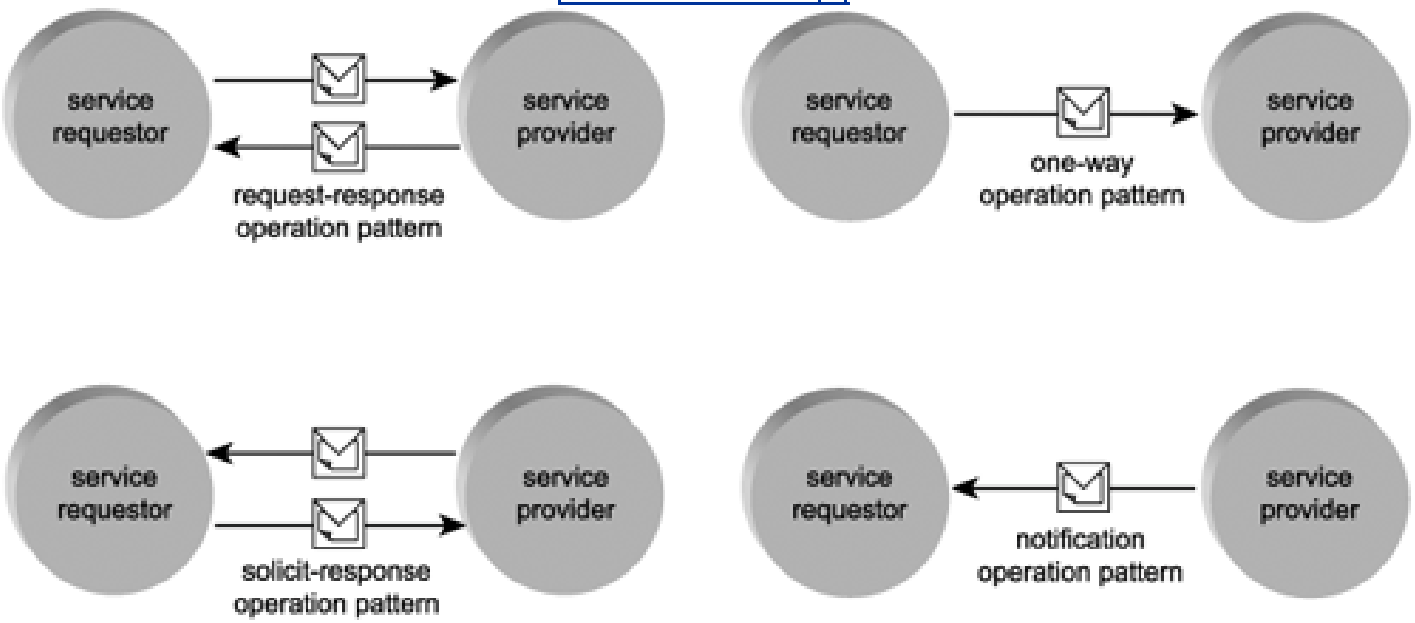
- **Step 2.** Upon the availability of the requested information, the publisher broadcasts messages on the particular topic to all of that topic's subscribers.

MEPs and WSDL

- Request-response operation
  - Upon receiving a message, the service must respond with a standard message or a fault message.
- Solicit-response operation
  - Upon submitting a message to a service requestor, the service expects a standard response message or a fault message.
- One-way operation
  - The service expects a single message and is not obligated to respond.
- Notification operation
  - The service sends a message and expects no response.

Figure 6.9. The four basic patterns supported by WSDL 1.1.

[View full size image]



WSDL specification extends MEP support to eight patterns

- **The in-out pattern**, comparable to the request-response MEP (and equivalent to the WSDL 1.1 request-response operation).
- **The out-in pattern**, which is the reverse of the previous pattern where the service provider initiates the exchange by transmitting the request. (Equivalent to the WSDL 1.1 solicit-response operation.)
- **The in-only pattern**, which essentially supports the standard *fire-and-forget* MEP. (Equivalent to the WSDL 1.1 one-way operation.)
- **The out-only pattern**, which is the reverse of the in-only pattern. It is used primarily in support of *event notification*. (Equivalent to the WSDL 1.1 notification operation.)

WSDL specification extends MEP support to eight patterns

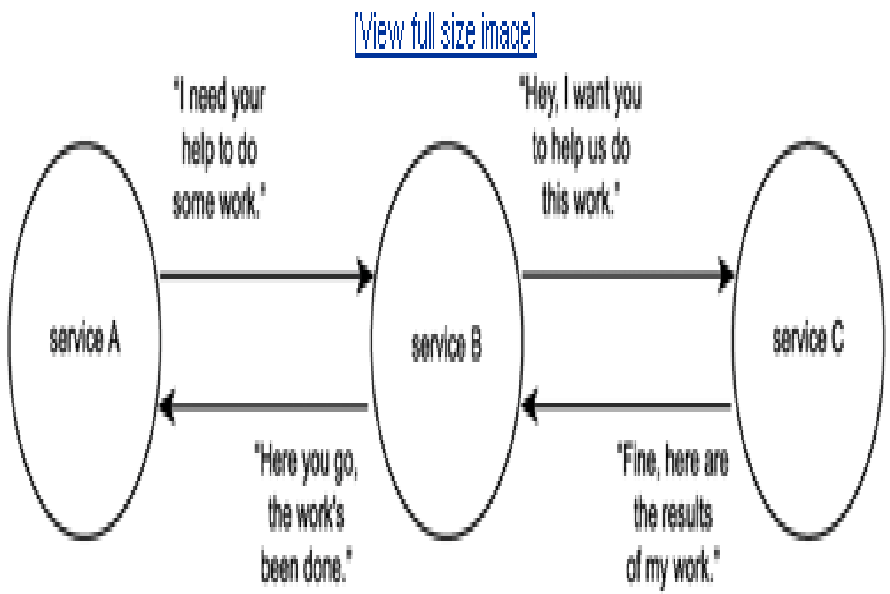
- **The robust in-only pattern**, a variation of the in-only pattern that provides the option of launching a fault response message as a result of a transmission or processing error.
- **The robust out-only pattern**, which, like the out-only pattern, has an outbound message initiating the transmission. The difference here is that a fault message can be issued in response to the receipt of this message.
- **The in-optional-out pattern**, which is similar to the in-out pattern with one exception. This variation introduces a rule stating that the delivery of a response message is optional and should therefore not be expected by the service requestor that originated the communication. This pattern also supports the generation of a fault message.
- **The out-optional-in pattern** is the reverse of the in-optional-out pattern, where the incoming message is optional. Fault message generation is again supported.

6.2 Service Activity

- Interaction of group of services to complete a task
  - Sort of a MEP plus the pattern of messages exchanged to perform a specific task
    - One MEP can perform multiple tasks
  - Primitive activity ~= Simple MEP
  - Complex Activity~= Multiple MEP’s or a large complex MEP

Service Activity

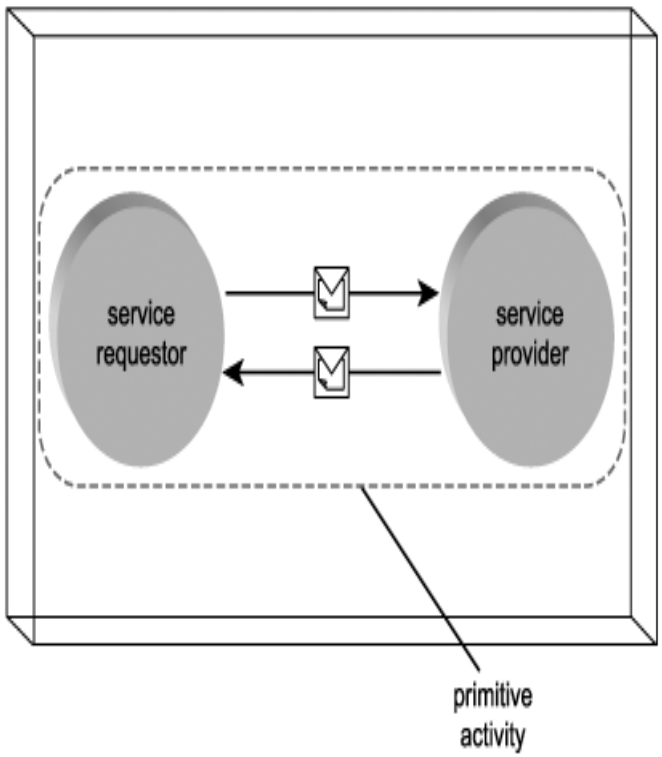
Figure 6.10. In an activity, multiple Web services collaborate to do a specific piece of work.



- The interaction of a group of services working together to complete a task can be referred to as a service activity

Simple activity

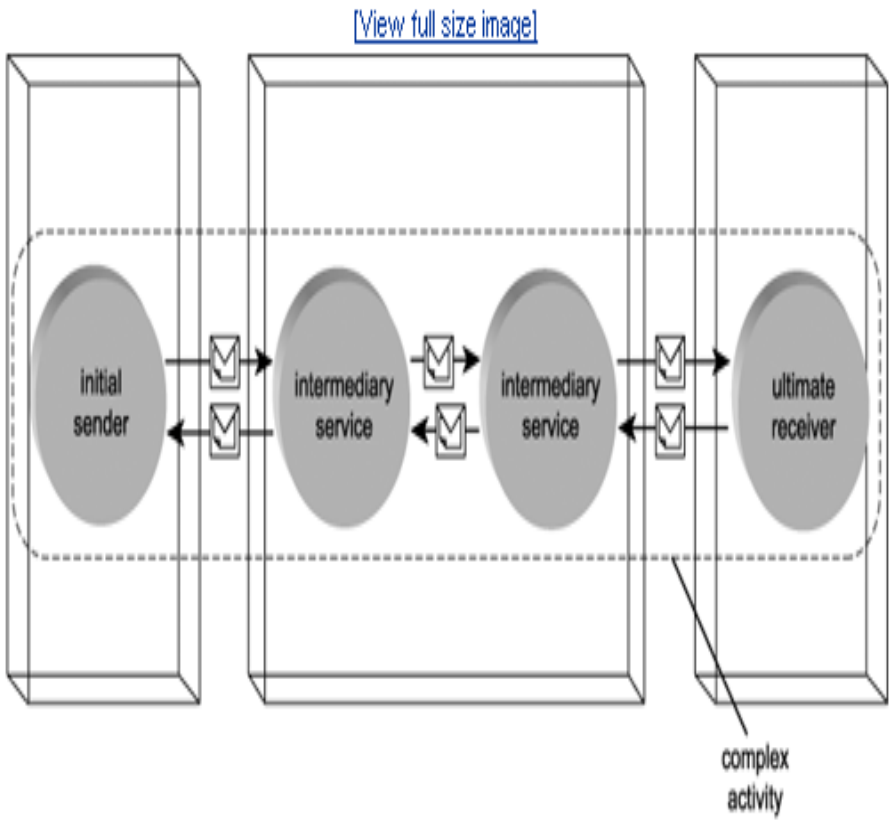
Figure 6.11. A primitive service activity consisting of a simple MEP.



- A simple or primitive activity is typified by synchronous communication and therefore often consists of two services exchanging information using a standard request-response MEP

Complex activities

Figure 6.12. A complex activity involving four services.



- *Complex activities*, on the other hand, can involve many services (and MEPs) that collaborate to complete multiple processing steps over a long period of time

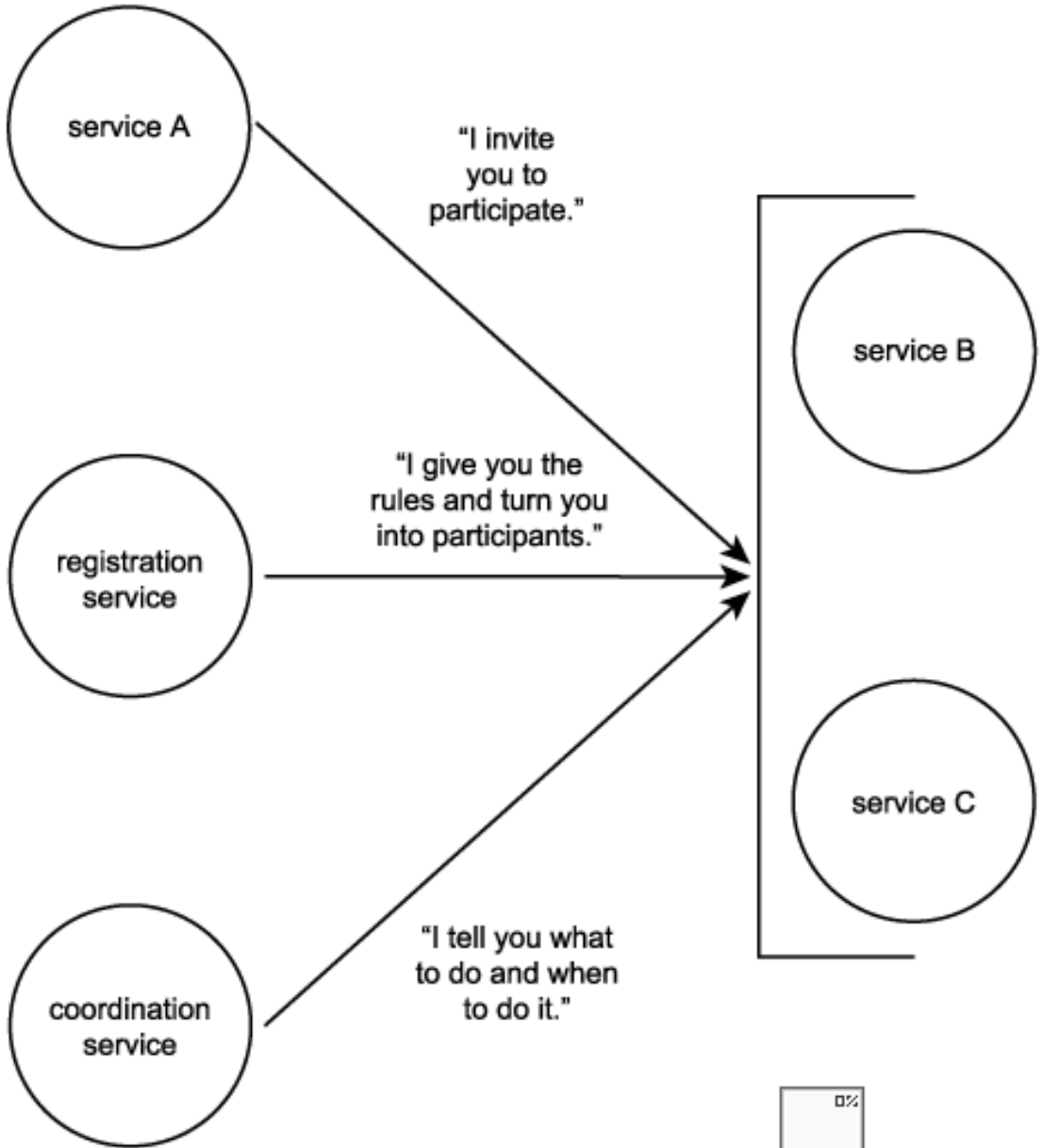
6.3 Coordination

- Coordination establishes a framework for complex activities to be managed and distributed to activity participants
- Can be extended to the concepts of choreography and orchestration
- Usually requires context state (information about the state of the complex activity) to be retained

Coordination

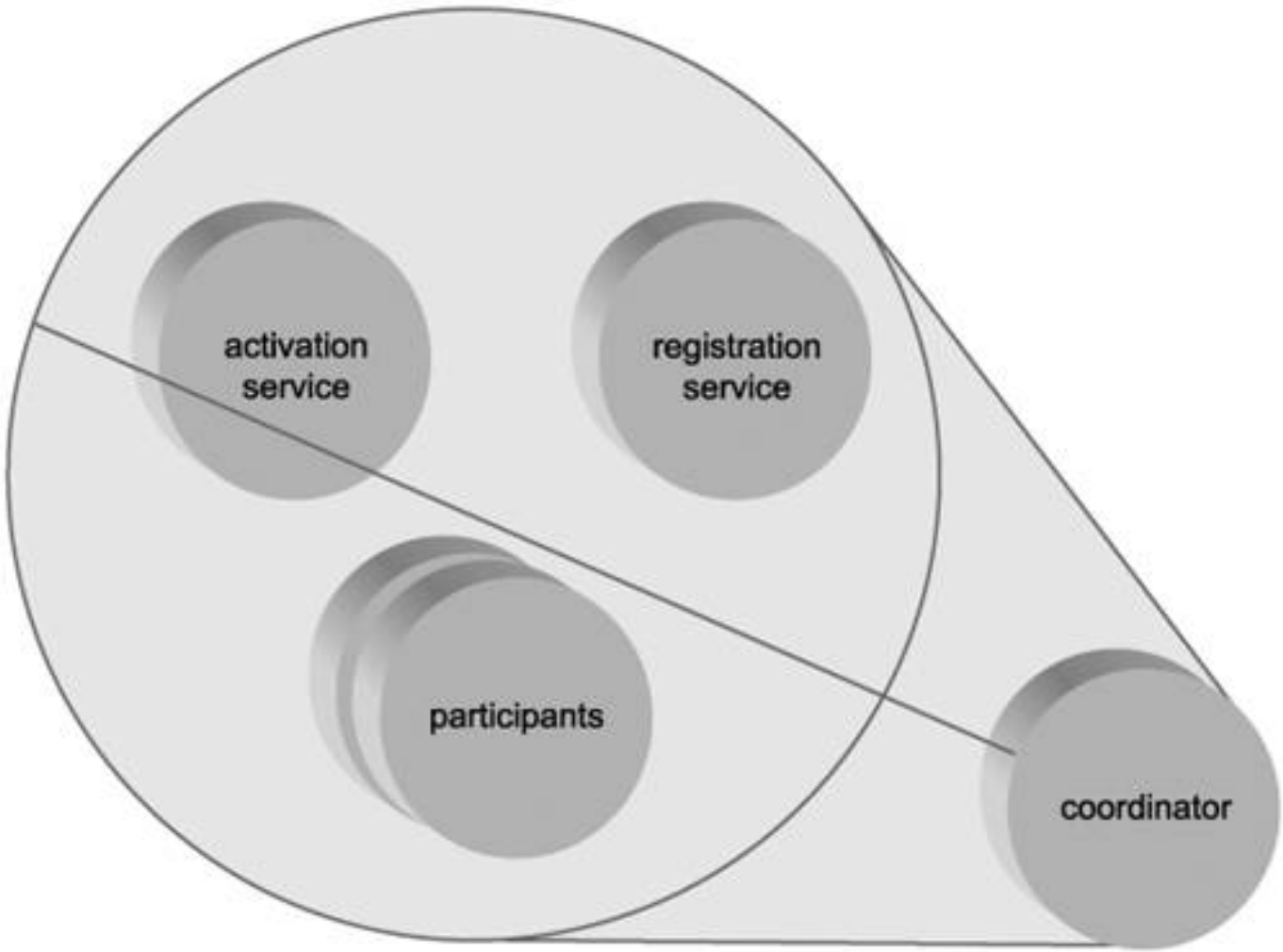
- The complexity of an activity can relate to a number of factors, including:
  - the amount of services that participate in the activity
  - the duration of the activity
  - the frequency with which the nature of the activity changes
  - whether or not multiple instances of the activity can concurrently exist
- A framework is required to provide a means for context information in complex activities to be managed, preserved and/or updated, and distributed to activity participants.

Figure 6.14. Coordination provides services that introduce controlled structure into activities.



Coordinator composition

Figure 6.15. The coordinator service composition.



WS-Coordination

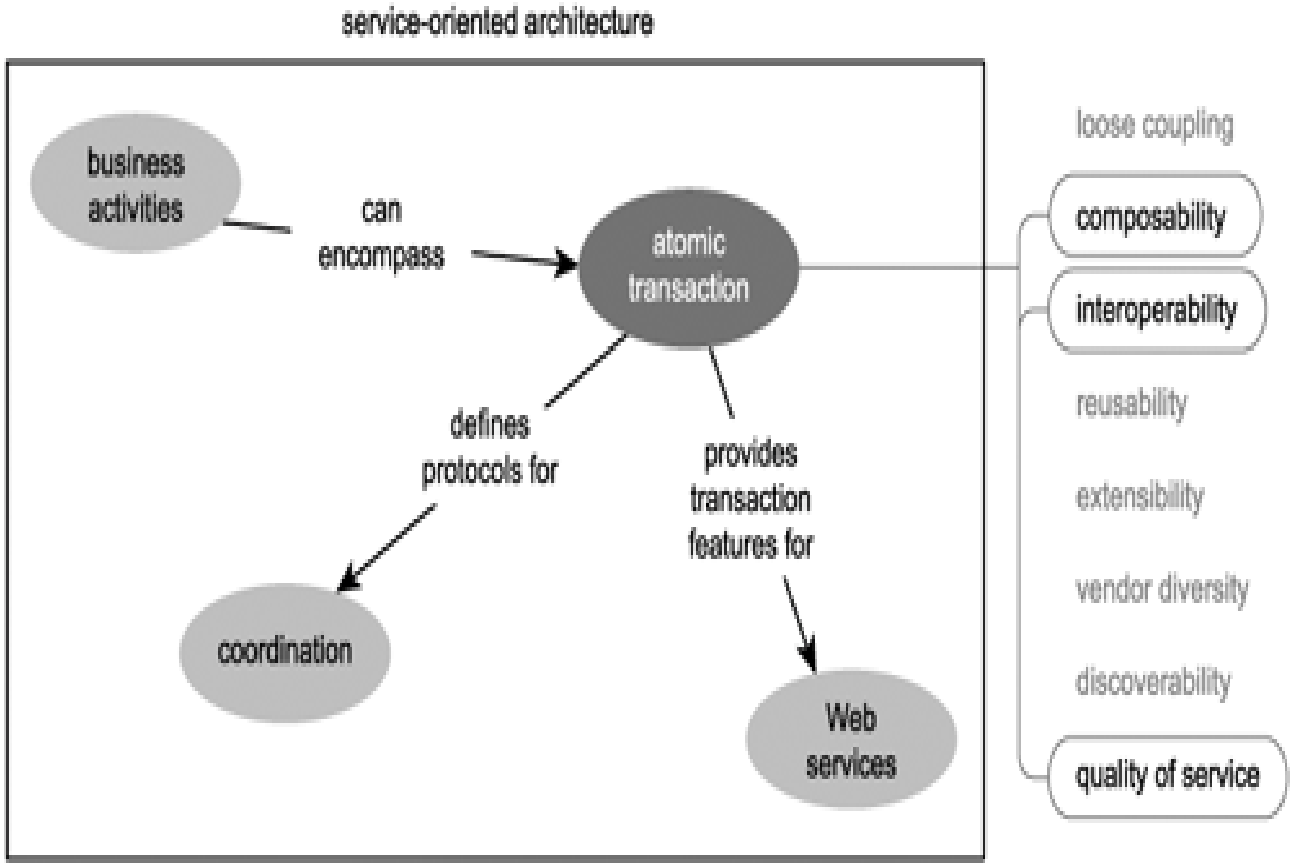
- WS-Coordination establishes a framework that introduces a generic service based on the coordinator service model
  - Activation service
    - Responsible for the creation of a new context and for associating this context to a particular activity.
  - Registration service
    - Allows participating services to use context information received from the activation service to register for a supported context protocol.
  - Protocol-specific services
    - These services represent the protocols supported by the coordinator's coordination type.
  - Coordinator
    - The controller service of this composition, also known as the coordination service.

6.4 Atomic Transaction

- WAKE UP – this section is complicated, boring and important
- ACID
  - Atomic - All or nothing, you can't do part of a transaction
  - Consistent – system data models must remain so
  - Isolated – transactions can't interfere with each other
  - Durable – Changes made by a transaction can survive subsequent system failures

Figure 6.25. Atomic transaction relating to other parts of SOA.

[View full size image]



6.5 Business Activities

- Long running, complex service activities
- Do not offer rollback capabilities
  - Not ACID-type transaction functionality
- 6.5.2 Business Activity States
  - Active state
  - Completed state
  - Cancelled state

Business activities

- *Business activities*
  - govern long-running, complex service activities.
  - Hours, days, or even weeks can pass before a business activity is able to complete.
  - During this period, the activity can perform numerous tasks that involve many participants.
- required to follow specific rules defined by protocols.
  - Business activities primarily differ from the also protocol-based atomic transactions
  - exceptions
  - the constraints introduced.
- business activity protocols do not offer rollback capabilities.
- not ACID-type transaction functionality.
- optional *compensation* process

Business activities

Figure 6.27. A business activity controls the integrity of a service activity by providing participants with a "plan B" (a compensation).

[View full size image]

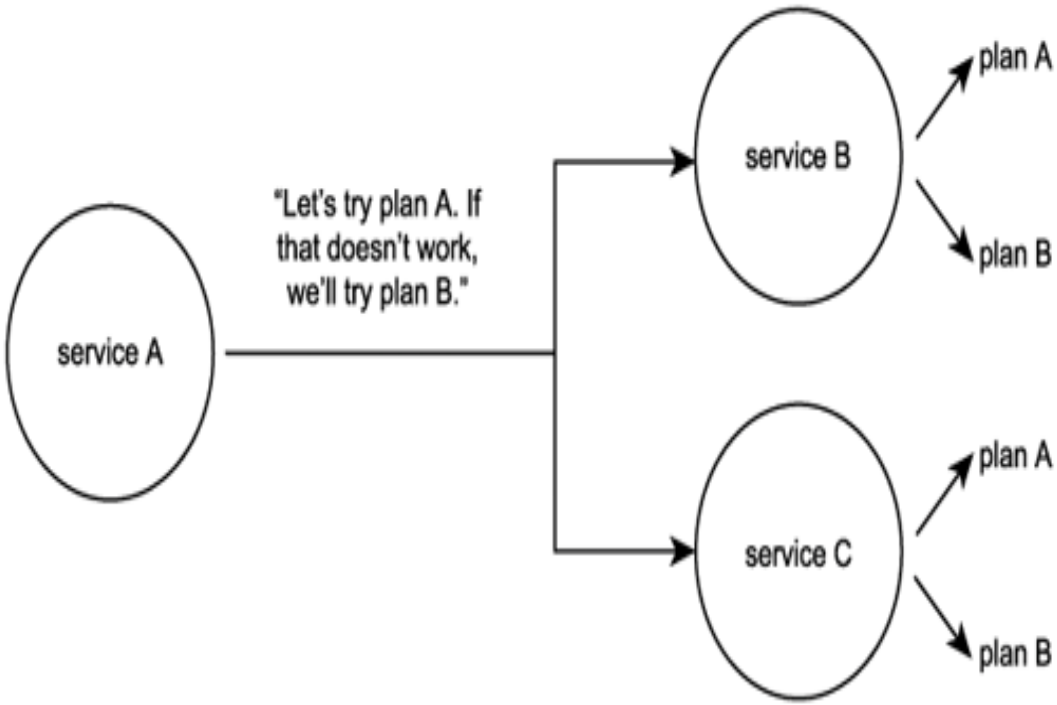
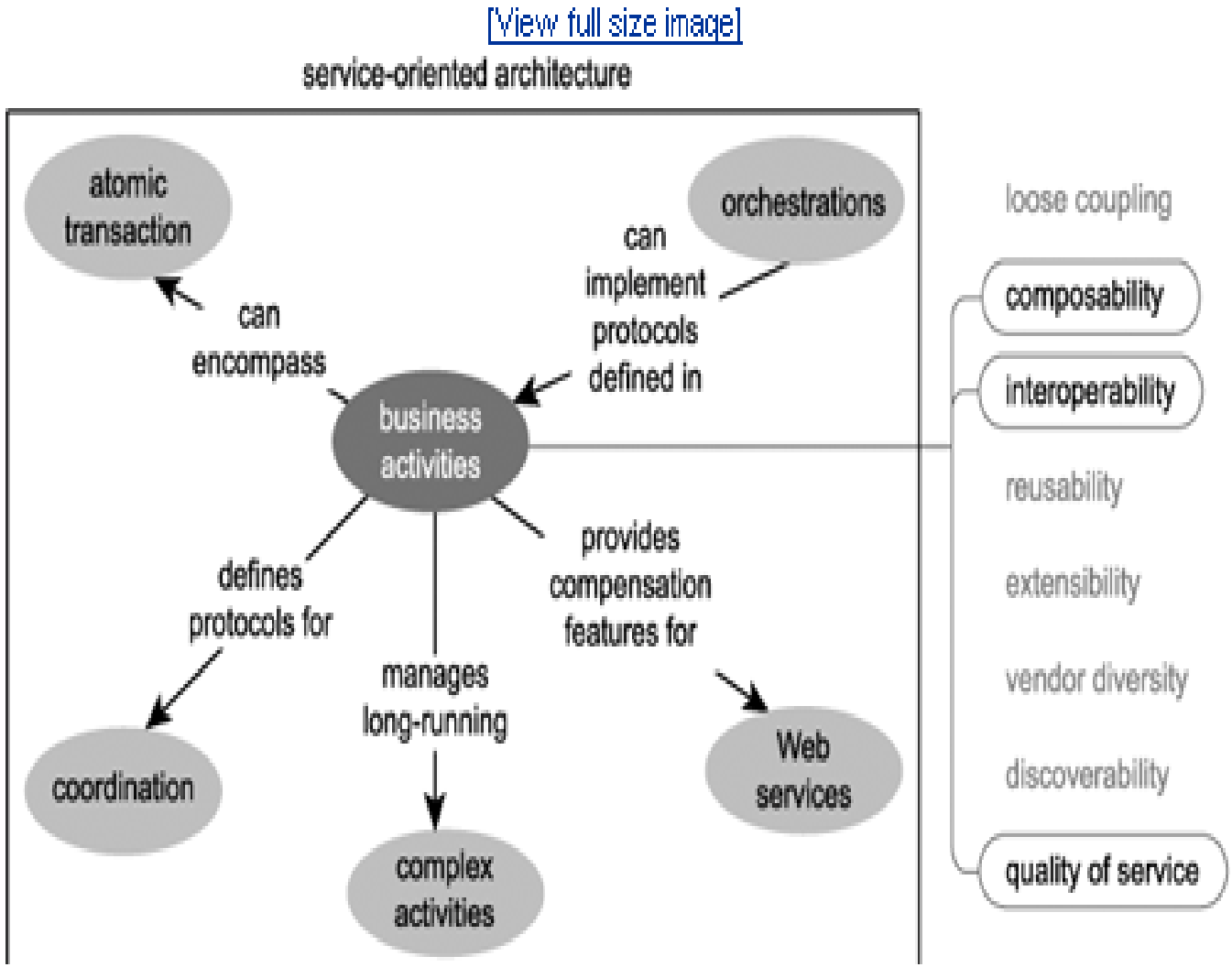


Figure 6.30. A business activity relating to other parts of SOA.



6.6 Orchestration

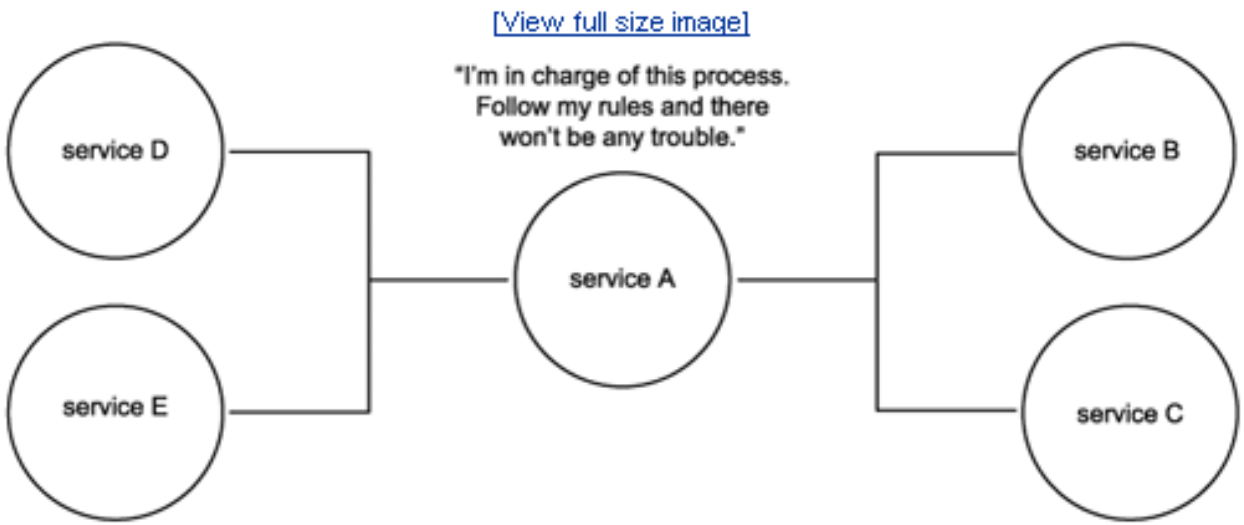
- 6.6.4 Sequences, flows and links
- 6.6.7 Orchestration and SOA

Sequences, flows, and links

- **Flows** also contain
- groups of related activities, but they introduce different execution requirements.
  - Pieces of application logic can execute concurrently within a flow, meaning that there is not necessarily a requirement for one set of activities to wait before another finishes.
  - However, the flow itself does not finish until all encapsulated activities have completed processing.
  - This ensures a form of synchronization among application logic residing in individual flows.
- **Links** are used to establish formal dependencies between activities that are part of flows.
  - Before an activity fully can complete, it must ensure that any requirements established in outgoing links first are met.
  - Similarly, before any linked activity can begin, requirements contained within any incoming links first must be satisfied.
  - Rules provided by links are also referred to as synchronization dependencies.

Orchestration

Figure 6.32. An orchestration controls almost every facet of a complex activity.



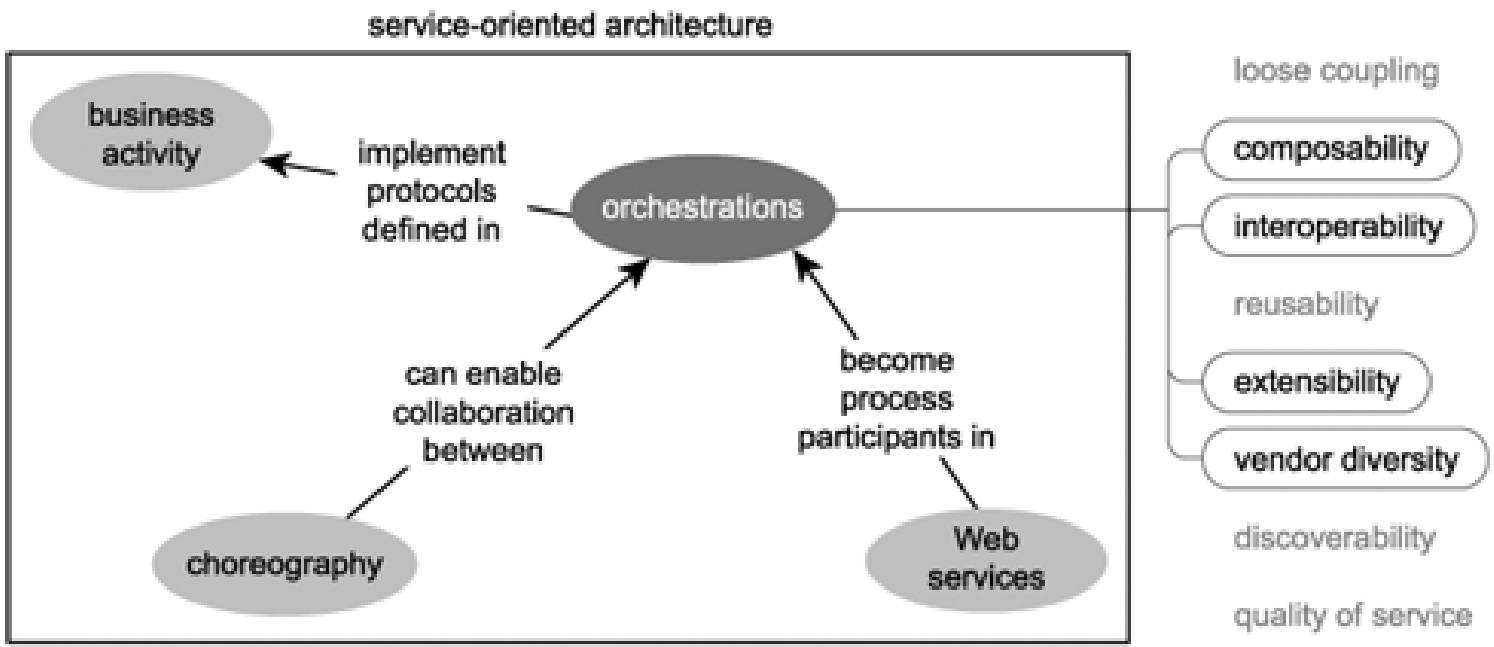
Basic activities and structured activities

- WS-BPEL breaks down workflow logic into a series of predefined primitive activities.
- Basic activities
  - **receive, invoke, reply, throw, wait**
  - represent fundamental workflow actions which can be assembled using the logic supplied by structured activities
    - **sequence, switch, while, flow, pick.**

Orchestration and SOA

Figure 6.35. Orchestration relating to other parts of SOA.

[View full size image]



6.7 Choreography

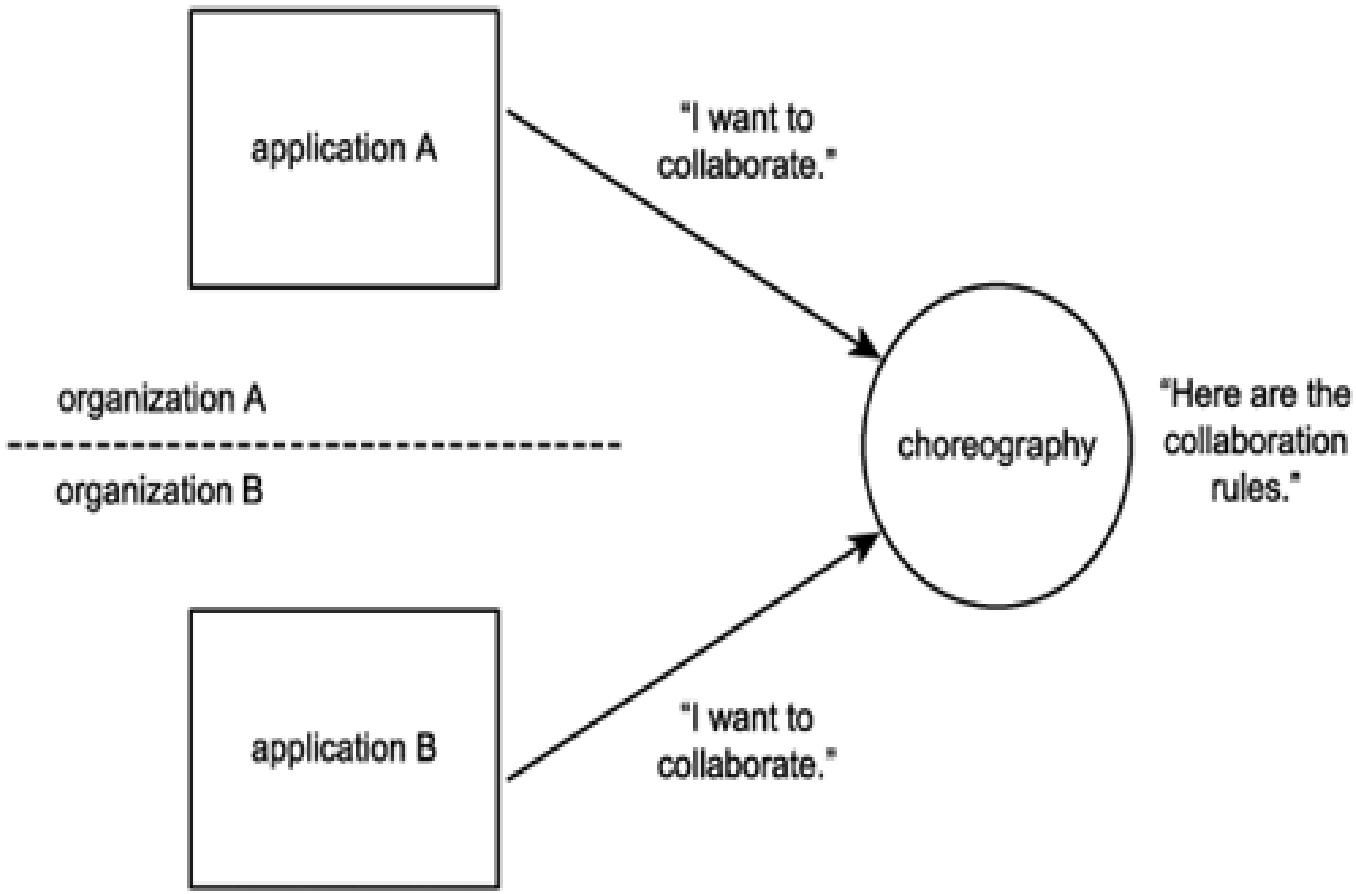
- 6.7.1 Collaboration
- 6.7.4 Interaction and work units
- 6.7.5 Reusability, composability and modularity
- 6.7.7 Choreography and SOA

Choreography

- Web Services Choreography Description Language (WS-CDL) is one of several specifications that attempts to organize information exchange between multiple organizations (or even multiple applications within organizations), with an emphasis on public collaboration

Figure 6.37. A choreography enables collaboration between its participants.

[View full size image]



Collaboration

- **Collaboration**
- An important characteristic of *choreographies* is that they are intended for public message exchanges. The goal is to establish a kind of organized collaboration between services representing different service entities, only no one entity (organization) necessarily controls the collaboration logic. Choreographies therefore provide the potential for establishing universal interoperability patterns for common inter-organization business tasks.
- While the emphasis on choreography is B2B interaction, it also can be applied to enable collaboration between applications belonging to a single organization. The use of orchestration, though, is far more common for this requirement.
-

Roles and participants

- **Roles and participants**
- predefined roles.
  - This establishes what the service does and what the service can do within the context of a particular business task.
  - Roles can be bound to WSDL definitions, and those related are grouped accordingly, categorized as participants (services).
- **Relationships and channels**
- message exchanges between two services.
- Each potential exchange between two roles in a choreography is therefore defined individually as a relationship. Every relationship consequently consists of exactly two roles.
- *Channels* do exactly that by defining the characteristics of the message exchange between two specific roles.

Roles and participants

- Further, to facilitate more complex exchanges involving **multiple participants**, channel information can actually be passed around in a message.
  - This allows one service to send another the information required for it to be communicated with by other services.
- This is a significant feature of the **WS-CDL** specification, as it fosters dynamic discovery and increases the number of potential participants within large-scale collaborative tasks.

Interactions and work units

- Finally, the actual logic behind a message exchange is encapsulated within an **interaction**.
- **Interactions** are the fundamental building blocks of choreographies because the completion of an interaction represents actual progress within a choreography.
- Related to interactions are work units.
  - These impose rules and constraints that must be adhered to for an interaction to successfully complete.

Reusability, composability, and modularity

- Each choreography can be designed in a reusable manner, allowing it to be applied to different business tasks comprised of the same fundamental actions.
- Further, using an import facility, a choreography can be assembled from independent modules.
- These modules can represent distinct sub-tasks and can be reused by numerous different parent choreographies

Choreography and SOA

- **Realization** of SOA across organization boundaries
- **Composability**, reusability, and extensibility, choreography also can increase organizational agility and discovery.
- Organizations are able to join into multiple online collaborations, which can dynamically extend or even alter related business processes that integrate with the choreographies.
- By being able to pass around channel information, participating services can make *third-party* organizations aware of other organizations with which they already have had contact.

Choreography and SOA

Figure 6.40. Choreography relating to other parts of SOA.