Service-Oriented Architecture

Slides are based on the book:
"Service-Oriented Architecture: Concepts, Technology, and Design"
By Thomas Erl,
Publisher: Prentice Hall PTR

Web Services and Contemporary SOA

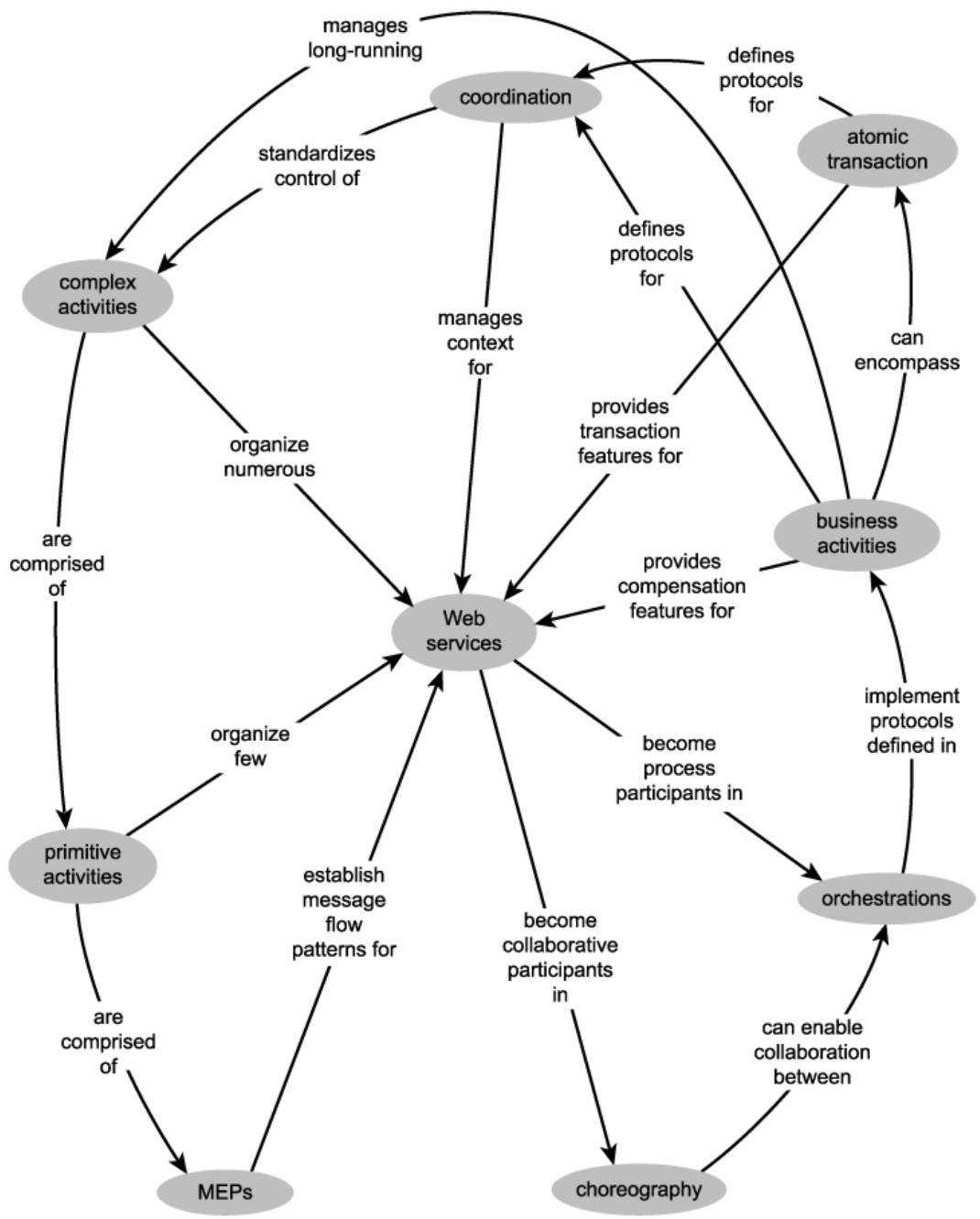
(Part I: Activity Management and Composition)

What is "WS-*"?

- The term "WS-*" has become a commonly used abbreviation that refers to the second-generation Web services specifications. These are extensions to the basic Web services framework established by first-generation standards represented by WSDL, SOAP, and UDDI.
- The term "WS-*" became popular because the majority of titles given to second-generation Web services specifications have been prefixed with "WS-".
- (See http://www.specifications.ws for examples of WS-* specifications.)

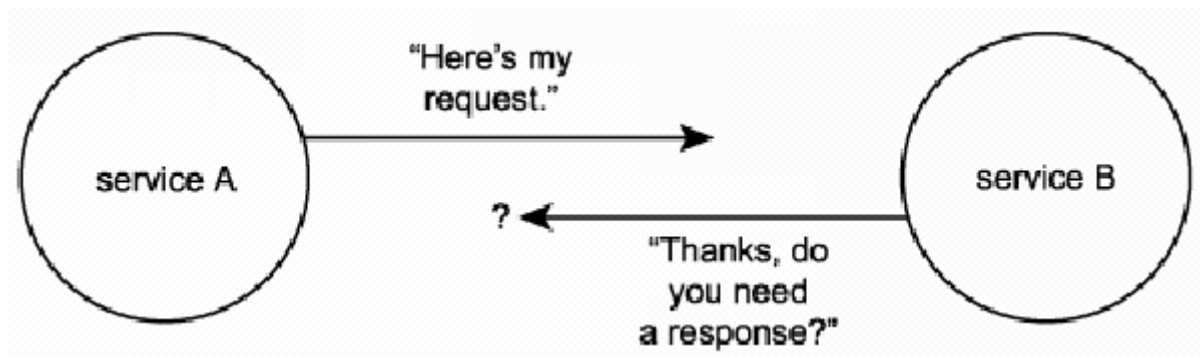
Web Services and Contemporary SOA

- To execute an automated business task with any substance, we need to be able to coordinate and compose a set of available services.
- To successfully perform this type of coordination requires that we have control over messaging that occurs between our composed services. Both conversational characteristics inherent in the Web services framework and specific features provided by key WS-* specifications provide a means to assert this control.
- Key concepts that define the messaging-based services activity model and also the role and context of concepts derived from WS-* specifications that implement standardized activity management and composition mechanisms are discussed.
- Collectively, these concepts (and the specifications from which they are derived) contribute to a robust, coordinated, and transaction-capable service-oriented architecture that is characteristically composable and extensible.
- (Next figure illustrates the concepts discussed on a high-level, how they can inter-relate.)


manages long-running
defines protocols for
coordination
atomic transaction
standardizes control of
complex activities
defines protocols for
manages context for
can encompass
provides transaction features for
organize numerous
are comprised of
business activities
provides compensation features for
Web services
implement protocols defined in
organize few
become process participants in
primitive activities
establish message flow patterns for
become collaborative participants in
orchestrations
are comprised of
can enable collaboration between
MEPs
choreography

Message exchange patterns

- Every task automated by a Web service can differ in both the nature of the application logic being executed and 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. The challenge lies in coordinating these messages in a particular sequence so that the individual actions performed by the message are executed properly and in alignment with the overall business task.


"Here's my request."
service A
service B
?
"Thanks, do you need a response?"

Message exchange patterns (MEPs)

- Message exchange patterns (MEPs) represent a set of templates that provide a group of already mapped out sequences for the exchange of messages. The most common example is a request and response pattern. Here the MEP states that upon successful delivery of a message from one service to another, the receiving service responds with a message back to the initial requestor.
- Many MEPs have been developed, each addressing a common message exchange requirement. It is useful to have a basic understanding of some of the more important MEPs, as you will no doubt be finding yourself applying MEPs to specific communication requirements when designing service-oriented solutions.

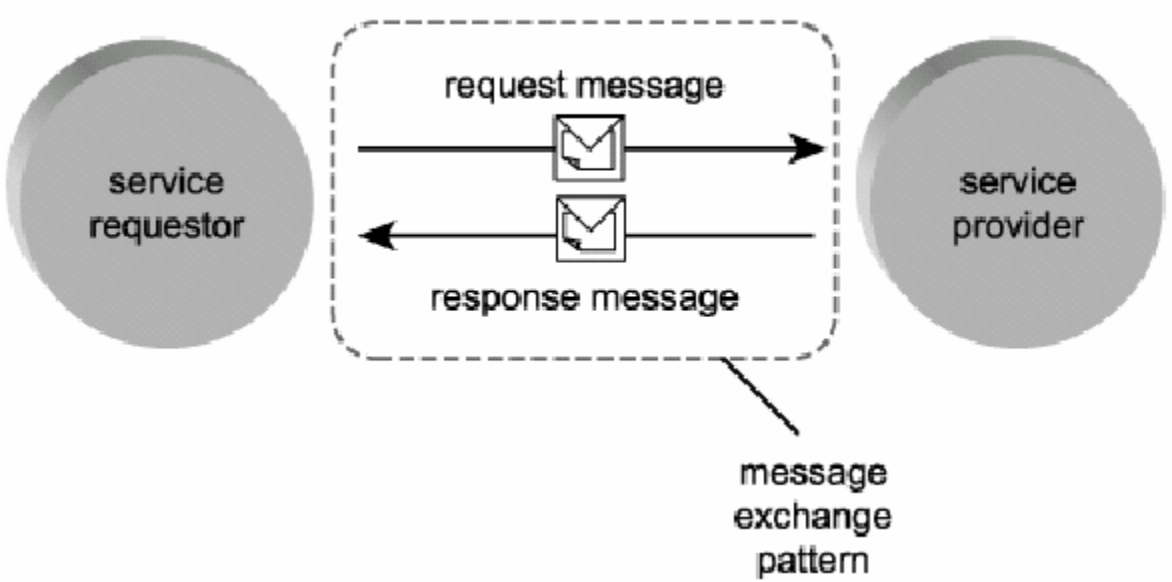
Primitive MEPs

- Before the arrival of contemporary SOA, messaging frameworks were already well used by various messaging-oriented middleware products. As a result, a common set of primitive MEPs has been in existence for some time.

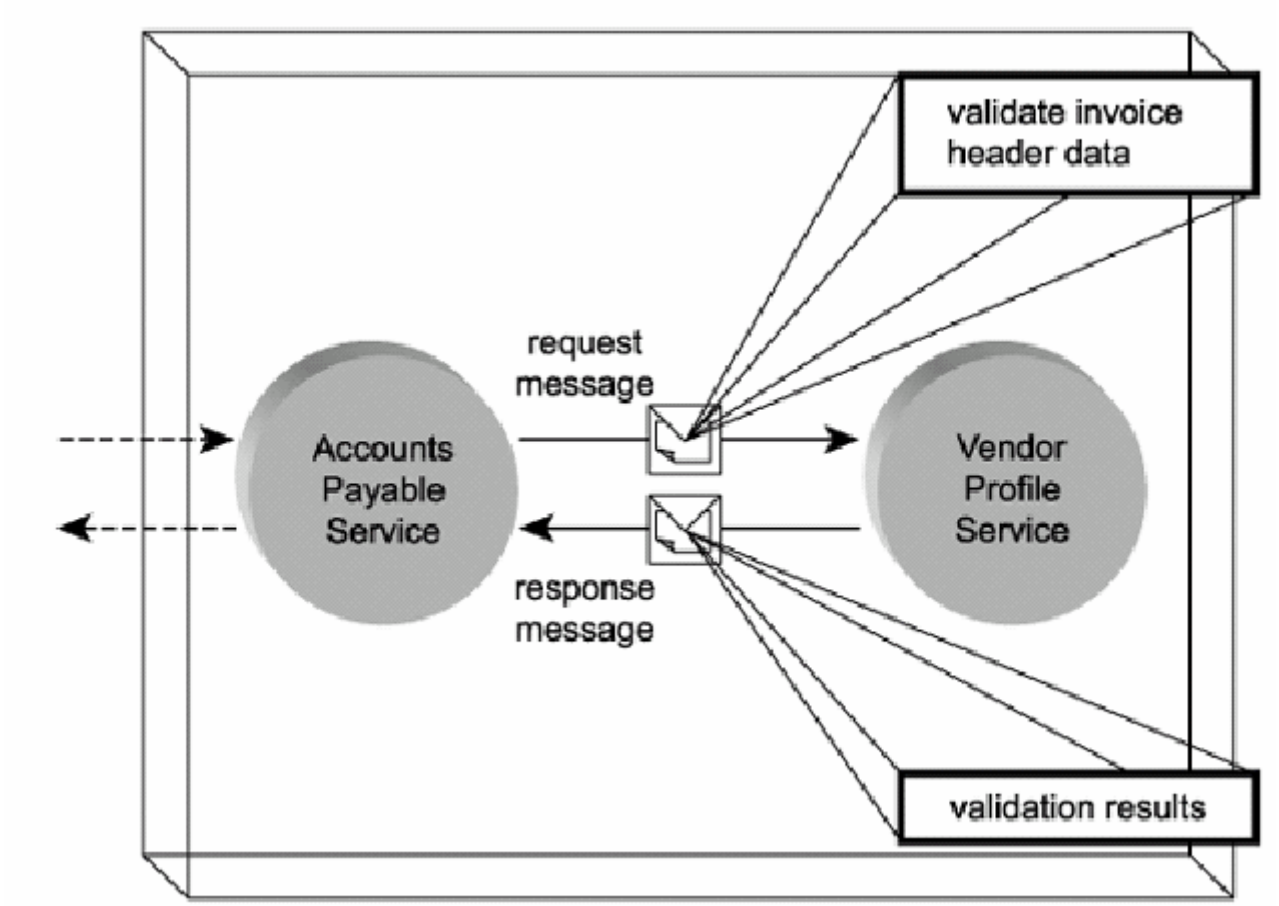
Request-response

- This is the most popular MEP in use among distributed application environments and the one pattern that defines synchronous communication (although this pattern also can be applied asynchronously).
- The request-response MEP 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).
- Note that within this MEP, services typically require a means of associating the response message with the corresponding request message. This can be achieved through the concept of correlation.

The request-response MEP



A sample request-response exchange between the Accounts Payable and Vendor Profile Services

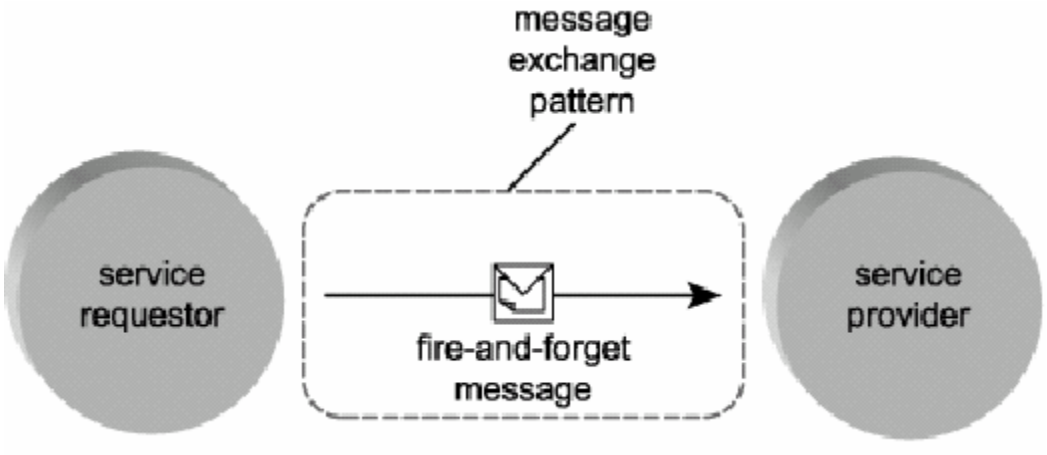


Primitive MEPs

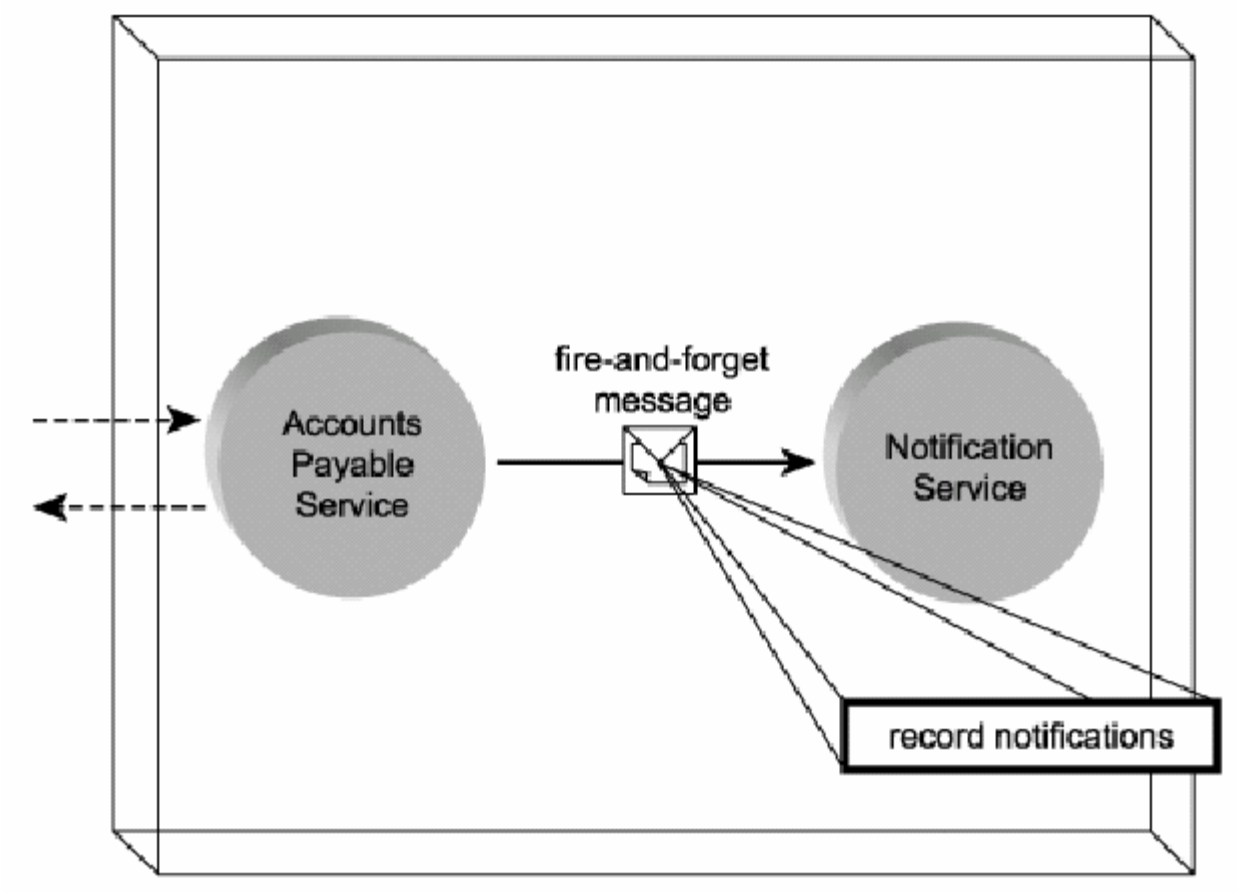
Fire-and-forget

- This simple asynchronous pattern is based on the unidirectional transmission of messages from a source to one or more destinations.
- 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.

The fire-and-forget MEP



Accounts Payable Service sending off a one-way notification message



Primitive MEPs

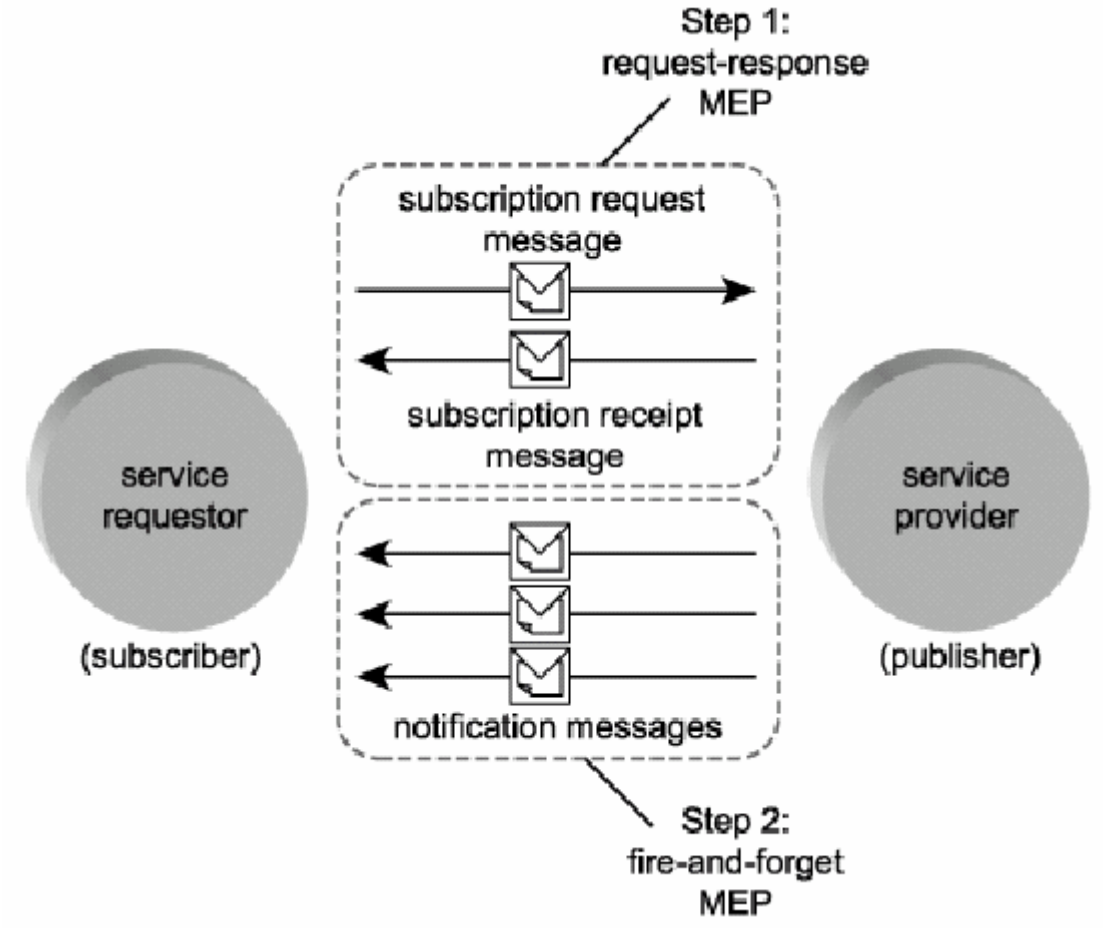
Complex MEPs

- Even though a message exchange pattern can facilitate the execution of a simple task, it is really more of a building block intended for composition into larger patterns. Primitive MEPs can be assembled in various configurations to create different types of messaging models, sometimes called complex MEPs.
- A classic example is the publish-and-subscribe model. Although we explain publish-and-subscribe approaches in more detail later on, let's briefly discuss it here as an example of a complex MEP.
- 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.

Primitive MEPs

- 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.
- This pattern is a great example of how to aggregate primitive MEPs, as shown in the next figure and explained here:
- Step 1 in the publish-and-subscribe MEP could be implemented by a request-response MEP, where the subscriber's request message, indicating that it wants to subscribe to a topic, is responded to by a message from the publisher, confirming that the subscription succeeded or failed.
- Step 2 then could be supported by one of the fire-and-forget patterns, allowing the publisher to broadcast a series of unidirectional messages to subscribers.

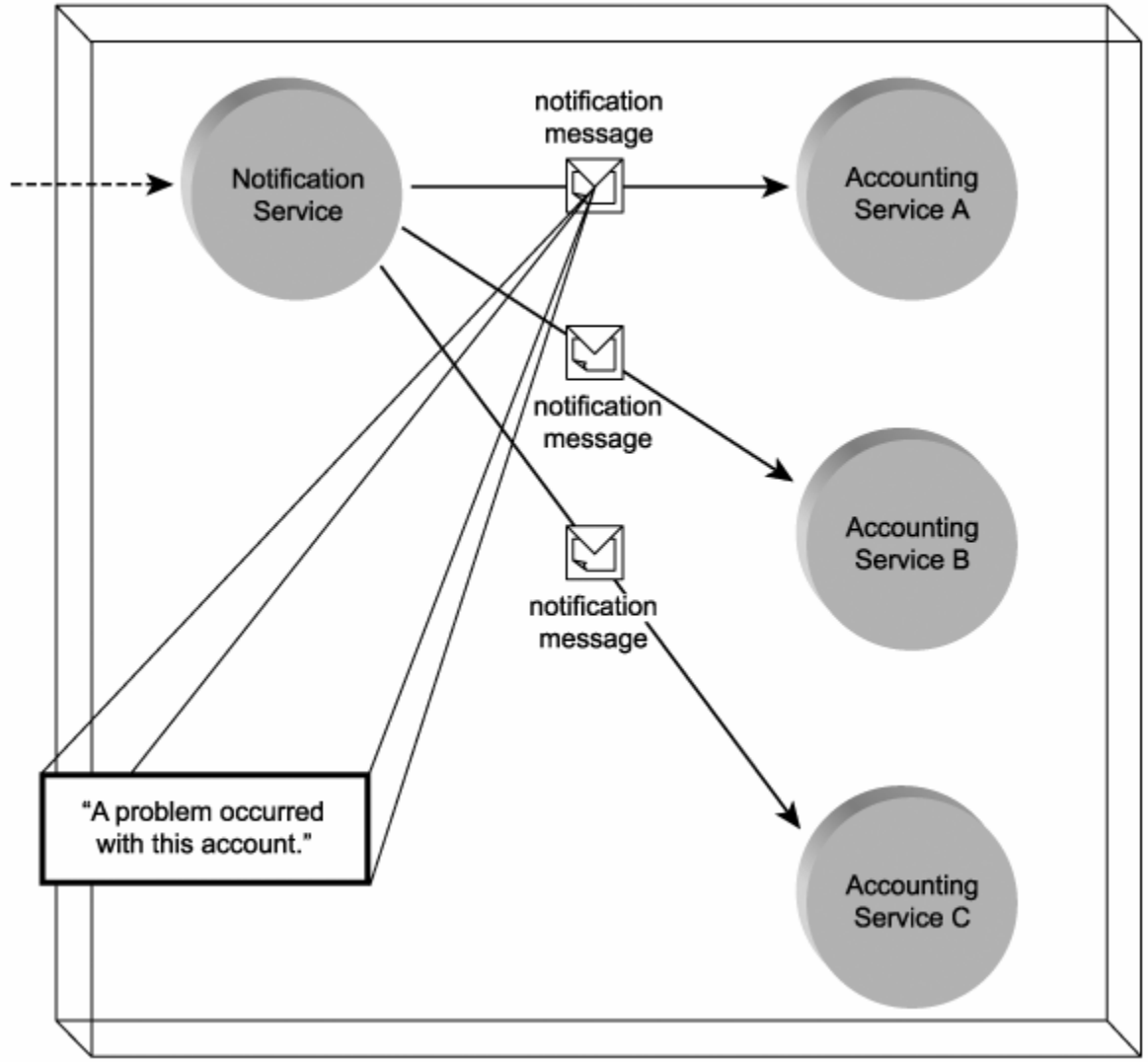
The publish-and-subscribe messaging model is a composite of two primitive MEPs



Primitive MEPs

- WS-* specifications that incorporate this messaging model include:
  - WS-BaseNotification
  - WS-BrokeredNotification
  - WS-Topics
  - WS-Eventing

Notification Service notifying subscribers about a problem condition via a notification broadcast



MEPs and SOAP

- On its own, the SOAP standard provides a messaging framework designed to support single-direction message transfer. The extensible nature of SOAP allows countless messaging characteristics and behaviors (MEP-related and otherwise) to be implemented via SOAP header blocks.
- The SOAP language also provides an optional parameter that can be set to identify the MEP associated with a message. (Note that SOAP MEPs also take SOAP message compliance into account.)

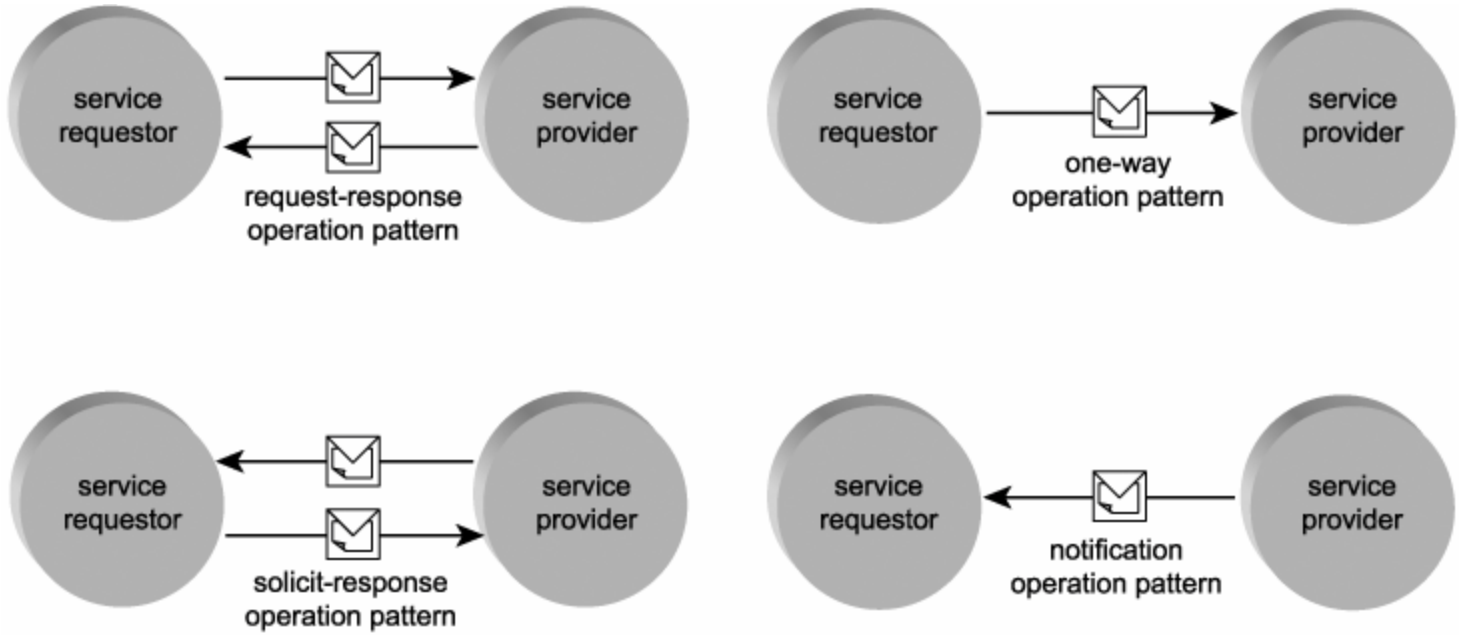
MEPs and WSDL

- Operations defined within service descriptions are comprised, in part, of message definitions. The exchange of these messages constitutes the execution of a task represented by an operation. MEPs play a larger role in WSDL service descriptions as they can coordinate the input and output messages associated with an operation. The association of MEPs to WSDL operations thereby embeds expected conversational behavior into the interface definition.
- WSDL operations support different configurations of incoming, outgoing, and fault messages. These configurations are equivalent to message exchange patterns, but within the WSDL specification, they often are referred to simply as patterns. It is important to note that WSDL definitions do not restrict an interface to these patterns; they are considered minimal conversational characteristics that can be extended.

MEPs and WSDL

- Release 1.1 of the WSDL specification provides support for four message exchange patterns that roughly correspond to the MEPs we described in the previous section. These patterns are applied to service operations from the perspective of a service provider or endpoint. In WSDL 1.1 terms, they are represented as follows:
  - 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.
- Of these four patterns (also illustrated in the next figure), only the request-response operation and one-way operation MEPs are recommended by the WS-I Basic Profile.

The four basic patterns supported by WSDL 1.1



MEPs and WSDL

- Not only does WSDL support most traditional MEPs, recent revisions of the specification have extended this support to include additional variations. Specifically, release 2.0 of the WSDL specification extends MEP support to eight patterns (and also changes the terminology) as follows.
  - 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 patternwhere 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.)

MEPs and WSDL

- 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 patternwith 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.

- Until version 2.0 of WSDL becomes commonplace, these new patterns will be of limited importance to SOA. Still, it is useful to know in what direction this core standard is heading.

MEPs and WSDL

- Note:
- Version 2.0 of the WSDL specification was originally labeled 1.2. However, the working group responsible for the new specification decided that the revised feature set constituted a full new version number. Therefore, 1.2 was changed to 2.0. However, you still may find references to version 1.2 in some places. WSDL 2.0 is not yet widely used, and details regarding this version of the specification are provided here as they demonstrate the broadening applicability of MEPs.

MEPs and SOA

- MEPs are highly generic and abstract in nature. Individually, they simply relate to an interaction between two services.
- Their relevance to SOA is equal to their relevance to the abstract Web services framework. They are therefore a fundamental and essential part of any Web services-based environment, including SOA.

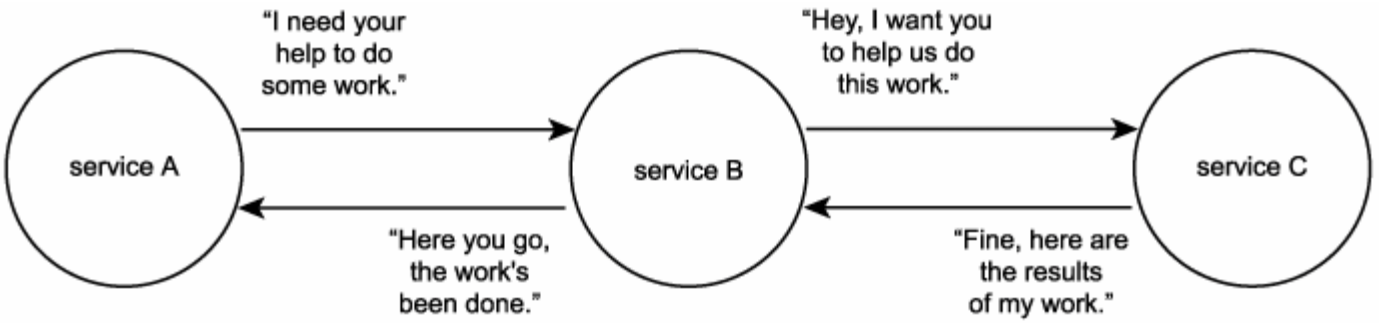
Summary of Key Points

- An MEP is a generic interaction pattern that defines the message exchange between two services.
- MEPs have been around for as long as messaging-based middleware products have been used. As a result, some common patterns have emerged.
- MEPs can be composed to support the creation of larger, more complex patterns.
- The WSDL and SOAP specifications support specific variations of common MEPs.

Service activity

- The completion of business tasks is an obvious function of any automated solution. Tasks are comprised of processing logic that executes to fulfill a number of business requirements. In service-oriented solutions, each task can involve any number of services. The interaction of a group of services working together to complete a task can be referred to as a service activity.
- Note:
- Though there is no formal definition of the term "activity," it is used by several of the specifications. It is a generic term that is most frequently associated with a logical unit of work completed by a collection of services.

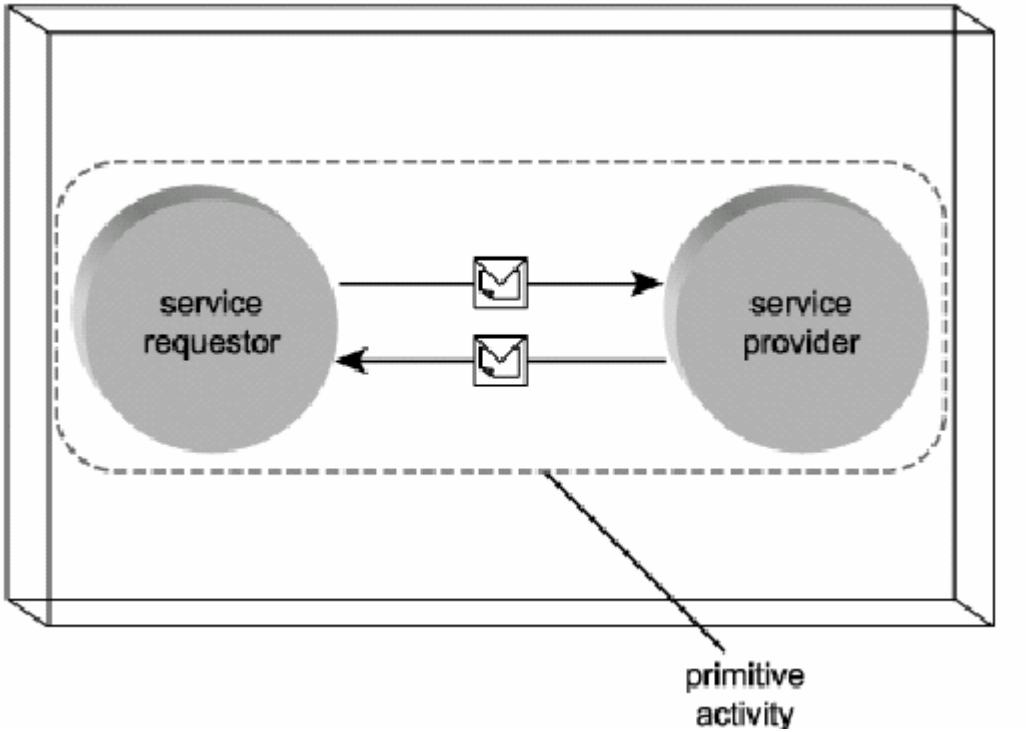
In an activity, multiple Web services collaborate to do a specific piece of work



Primitive and complex service activities

- The scope of an activity can drastically vary. 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.
- Primitive activities are almost always short-lived; the execution of a single MEP generally constitutes the lifespan of a primitive activity.

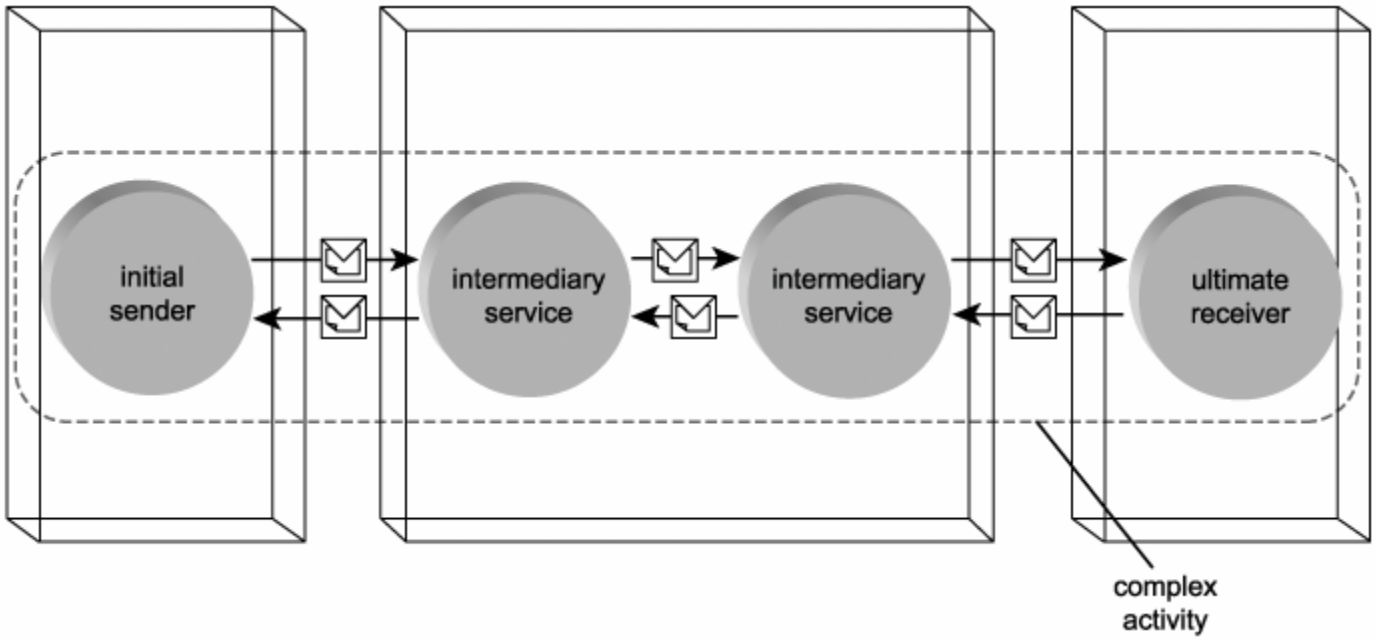
A primitive service activity consisting of a simple MEP



Primitive and complex service activities

- 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.
- These more elaborate types of activities are generally structured around extension-driven and composition-oriented concepts, such as choreography and orchestration.
- However, a business task automated by a series of custom-developed services and without the use of a composition extension can just as easily be classified a complex activity.

A complex activity involving four services



Service activities and SOA

- In SOAs, activities represent any service interaction required to complete business tasks. The scope of a service activity is primarily concerned with the processing and communication between services only.
- The underlying application 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. Complex activities are common place in larger service-oriented solutions and can involve numerous participating services.
- Note:
- The term "Web services activity" also happens to represent the ongoing Web services-related work being performed by numerous W3C working groups.

Case Study

- The message path traveled by a RailCo invoice submission is a great example of a complex activity.
- Invoice Submission Process:

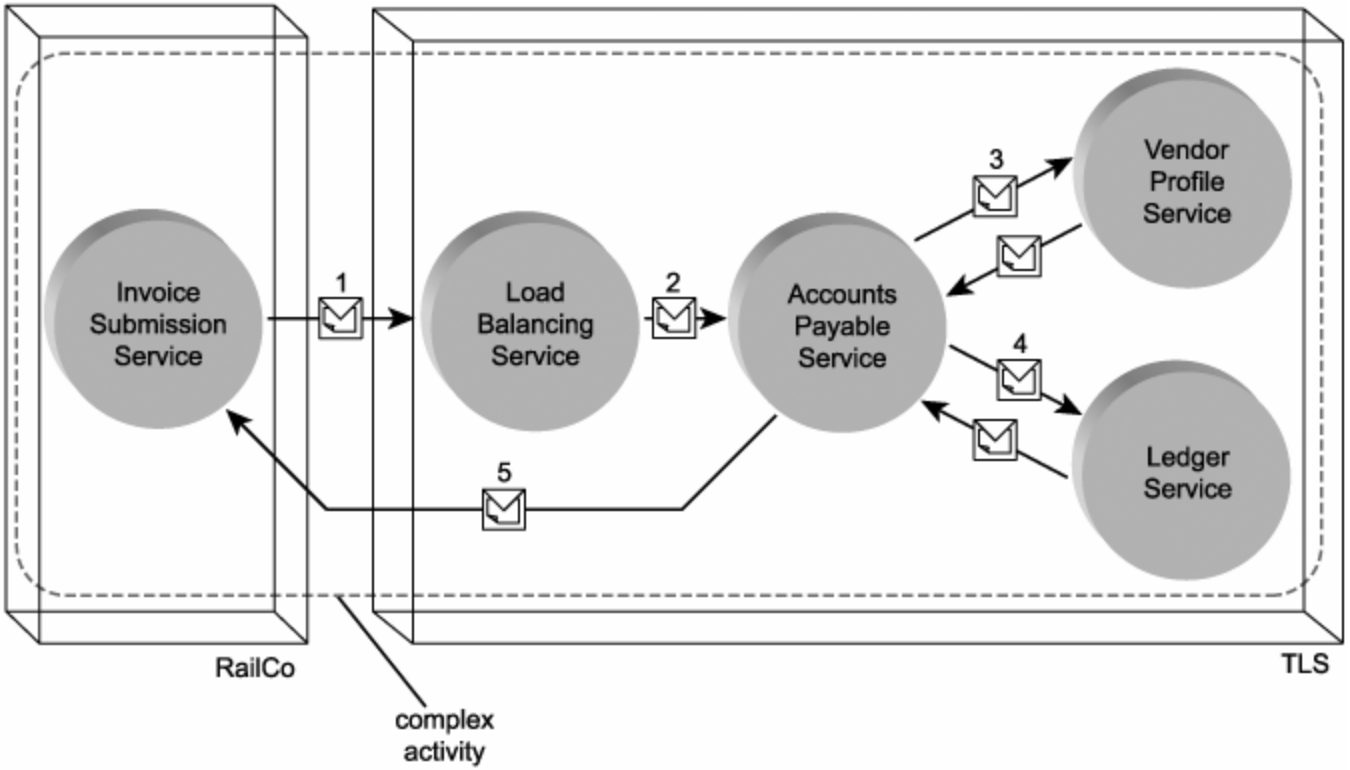
1. The initial sender, RailCo's Invoice Submission Service, transmits the invoice message.
2. The message is first received by a passive intermediary, TLS's Load Balancing Service, which routes the message according to environmental conditions. The message subsequently arrives at TLS's Accounts Payable Service.
3. The Accounts Payable Service acts as a controller and initiates a service composition to begin processing the message contents. It begins by interacting with the Vendor Profile Service to validate invoice header data and attaches the invoice document to the vendor account.

Case Study

4. The Accounts Payable Service then extracts taxes, shipping fees, and the invoice total. It passes these values to the Ledger Service, which updates various ledger accounts, including the General Ledger.
5. Finally the activity ends, as the Accounts Payable Service completes its processing cycle by sending a response message back to RailCo.

- Collectively, these processing steps constitute a complex activity involving five services

A sample complex activity spanning RailCo and TLS boundaries



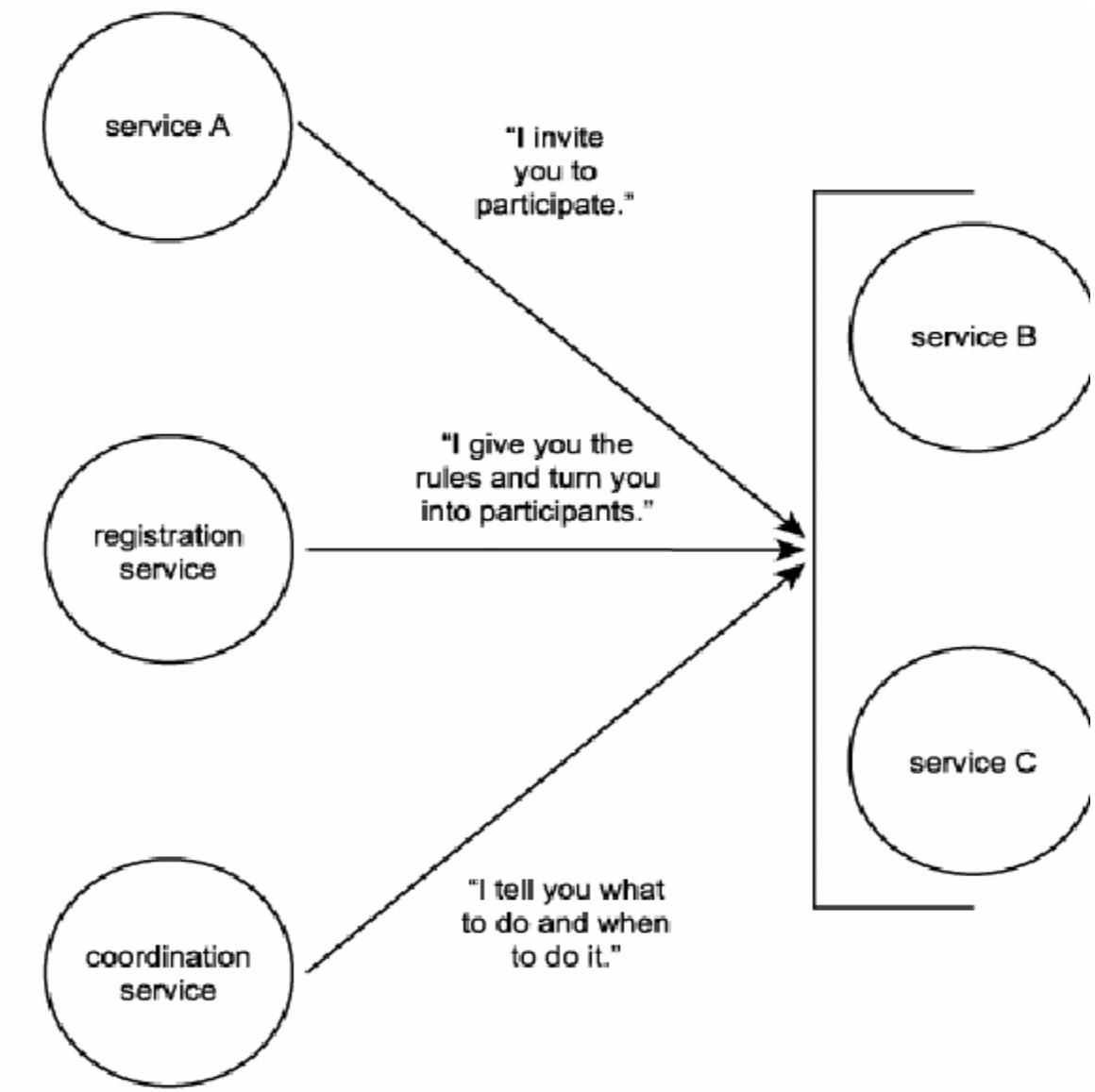
Summary of Key Points

- An activity is a generic concept used to represent a task or a unit of work performed by a set of services.
- The scope of primitive activities can be limited to the completion of simple MEPs.
- Complex activities are common within SOAs and exist as part of any non-trivial service-oriented application.

Coordination

- Every activity introduces a level of context into an application runtime environment. Something that is happening or executing has meaning during its lifetime, and the description of its meaning (and other characteristics that relate to its existence) can be classified as context information.
- The more complex an activity, the more context information it tends to bring with it. 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. Coordination establishes such a framework.

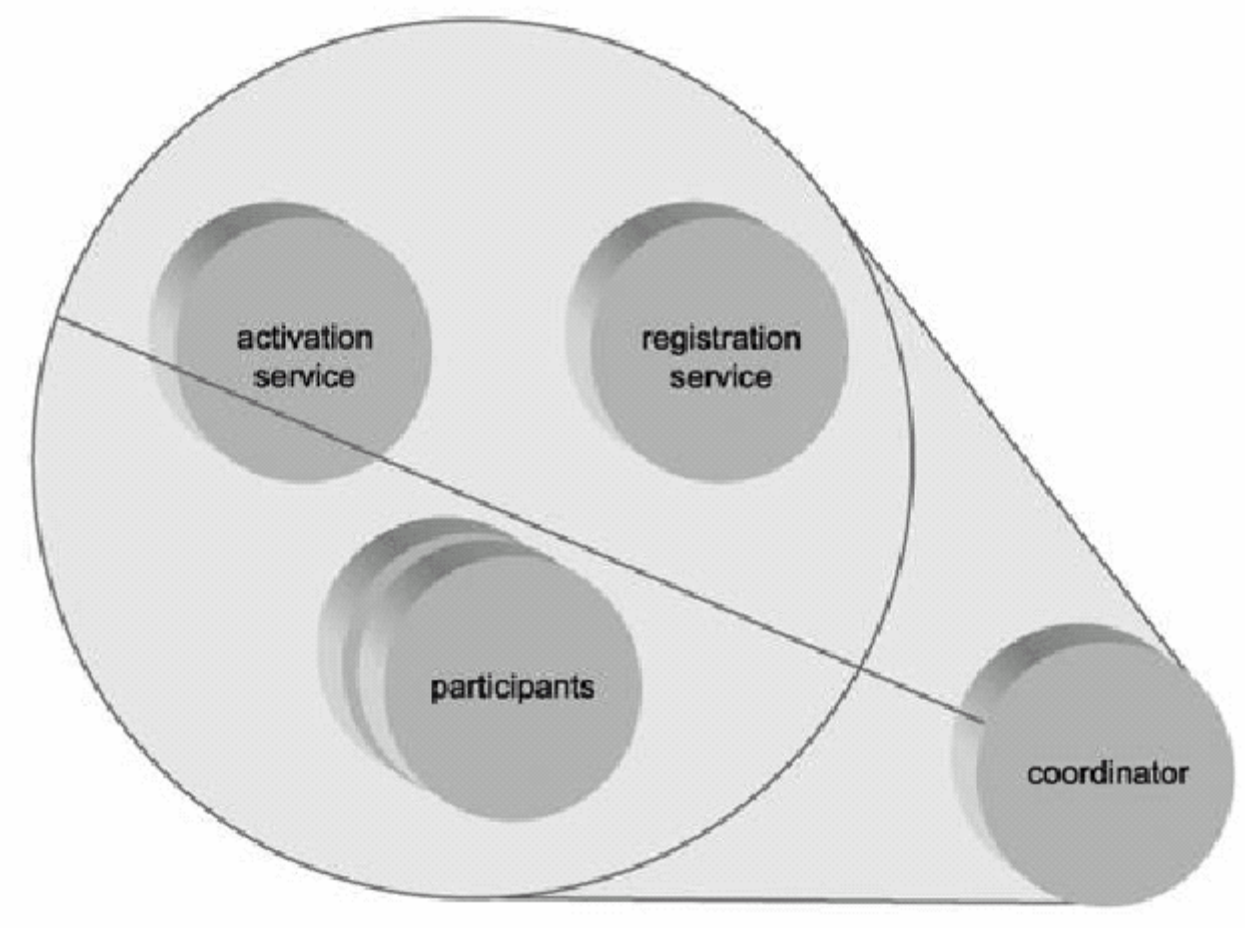
Coordination provides services that introduce controlled structure into activities



Coordinator composition

- WS-Coordination establishes a framework that introduces a generic service based on the coordinator service model.
- This service controls a composition of three other services that each play a specific part in the management of context data.

The coordinator service composition



Coordinator composition

- The coordinator composition consists of the following services:
- **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. (This is further explained in the next sections.)
- **Coordinator** The controller service of this composition, also known as the coordination service.

Coordination types and coordination protocols

- Each coordinator is based on a coordination type, which specifies the nature and underlying logic of an activity for which context information is being managed.
- Coordination types are specified in separate specifications.
- The WS-Coordination framework is extensible and can be utilized by different coordination types, including custom variations.
- However, the two coordination types most commonly associated with WS-Coordination are
  - WS-AtomicTransaction and
  - WS-BusinessActivity.
- (Concepts relating to these specifications are explained in the upcoming Atomic transactions and Business activities sections.)
- Coordination type extensions provide a set of coordination protocols, which represent unique variations of coordination types and consist of a collection of specific behaviors and rules.
- A protocol is best viewed as a set of rules that are imposed on activities and which all registered participants must follow.

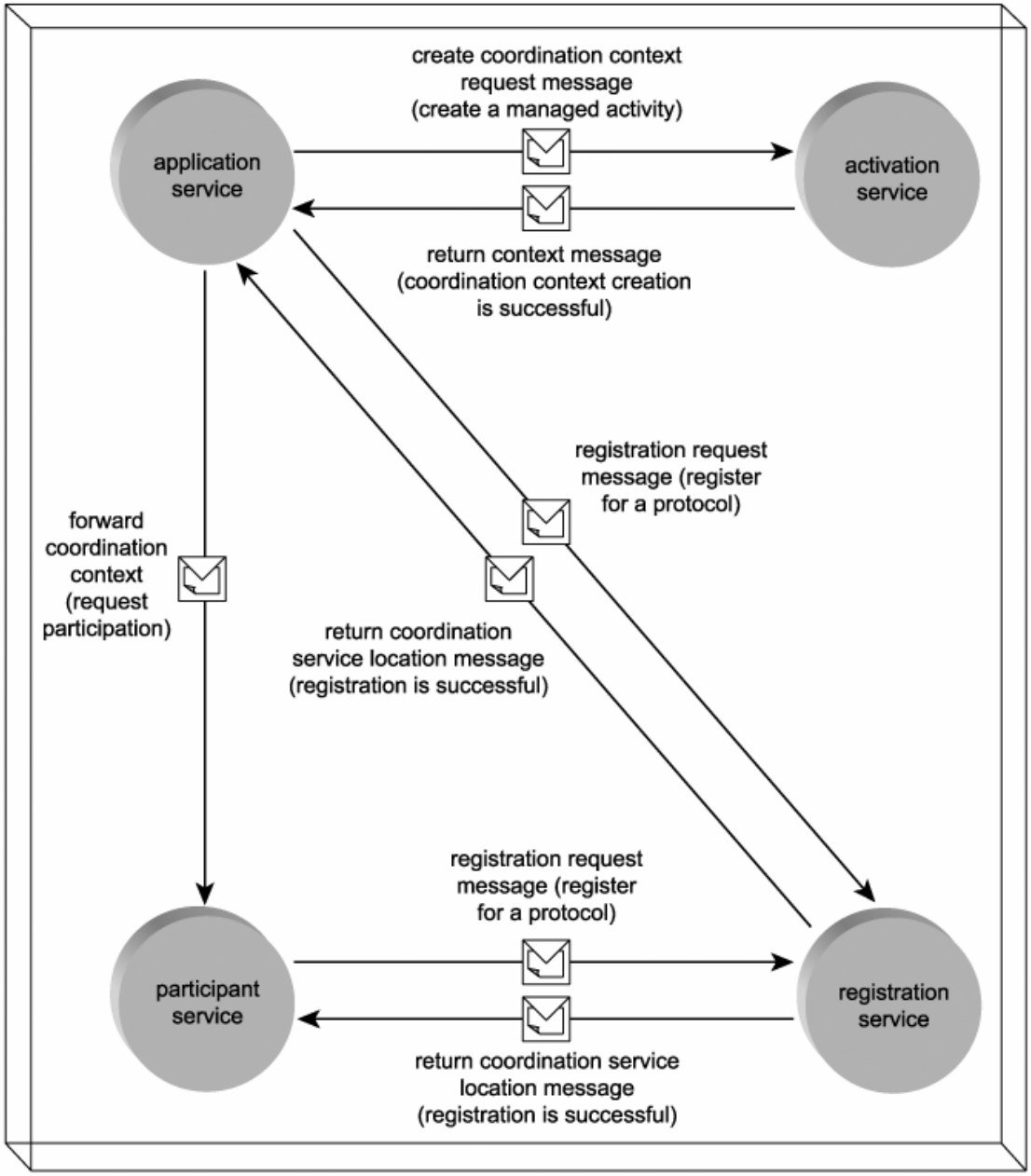
Coordination contexts and coordination participants

- A context created by the activation service is referred to as a coordination context. It contains a collection of information that represents the activity and various supplementary data.
- Examples of the type of data held within a coordination context include:
  - a unique identifier that represents the activity
  - an expiration value
  - coordination type information
- A service that wants to take part in an activity managed by WS-Coordination must request the coordination context from the activation service. It then can use this context information to register for one or more coordination protocols.
- A service that has received a context and has completed registration is considered a participant in the coordinated activity.

The activation and registration process

- The coordination service composition is instantiated when an application service contacts the activation service as given in the next figure.
- Via a CreateCoordinationContext request message, it asks the activation service to generate a set of new context data.
- Once passed back with the ReturnContext message, the application service now can invite other services to participate in the coordination. This invitation consists of the context information the application service originally received from the activation service.

The WS-Coordination registration process



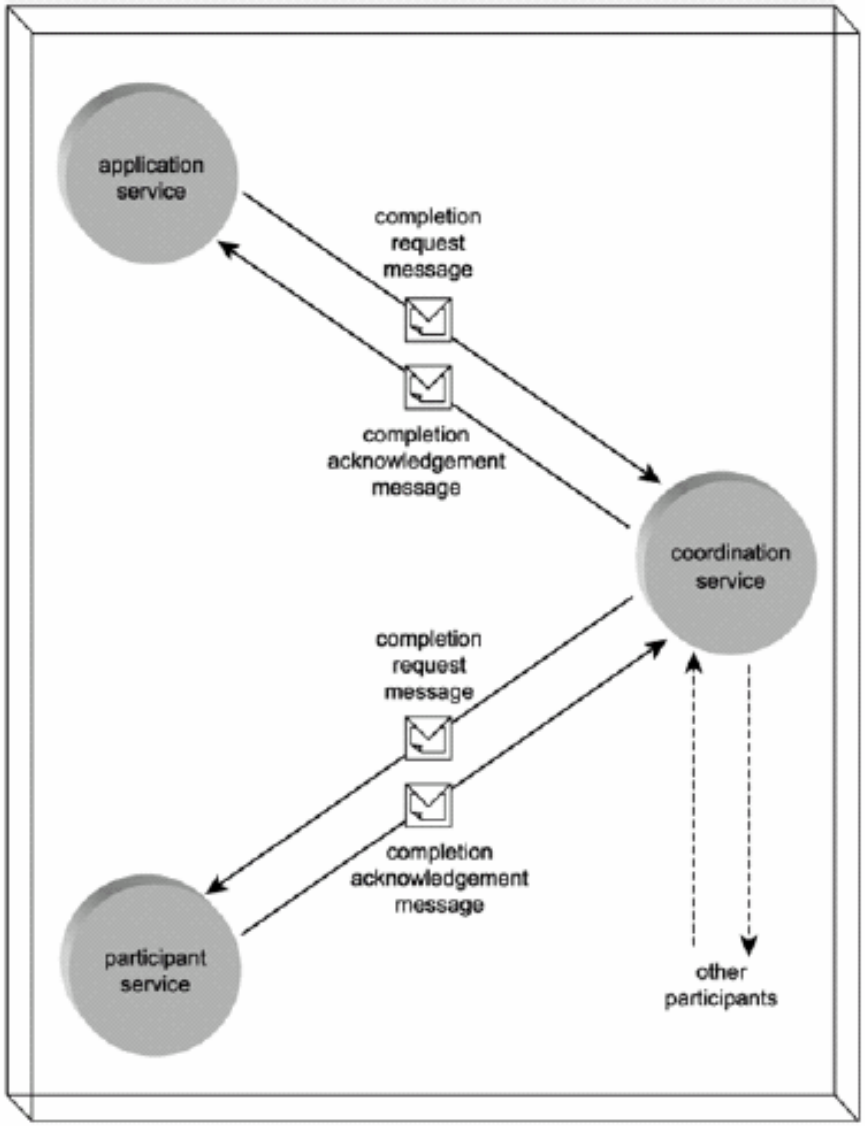
The activation and registration process

- Any Web service in possession of this context information may issue a registration request to the registration service.
- This allows the service to enlist in a coordination based on a specific protocol. (Protocols are provided by separate specifications and are discussed later on as part of the Atomic transaction and Business activities sections.)
- Upon a successful registration, a service is officially a participant.
- The registration service passes the service the location of the coordinator service, with which all participants are required to interact.
- At this time, the coordination service is also sent the address of the new participant.

The completion process

- The application service can request that a coordination be completed by issuing a completion request message to the coordination service.
- The coordinator, in turn, then issues its own completion request messages to all coordination participants.
- Each participant service responds with a completion acknowledgement message.

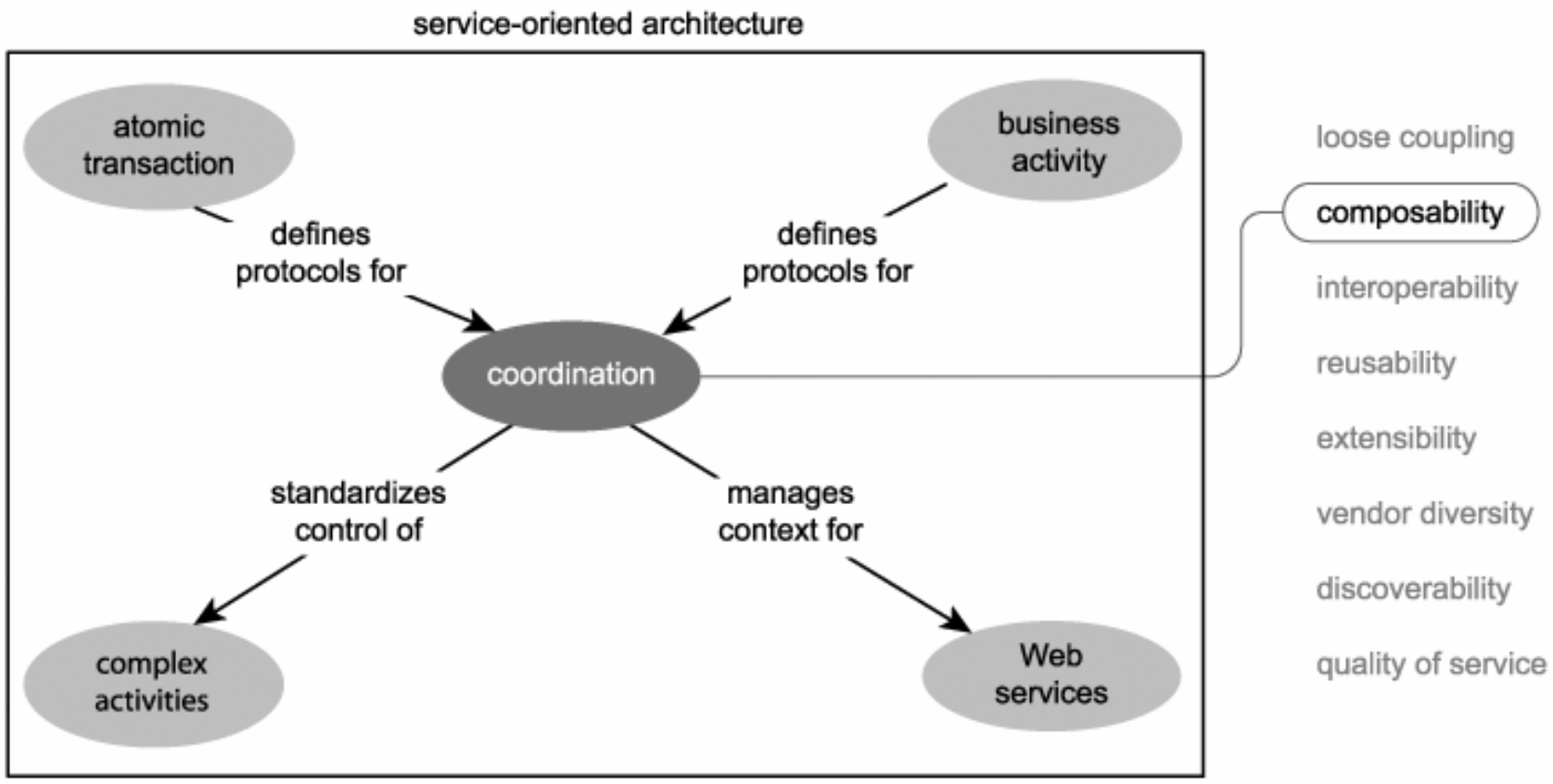
The WS-Coordination completion process



Coordination and SOA

- A coordinator-based context management framework, as provided by WS-Coordination and its supporting coordination types, introduces a layer of composition control to SOAs.
- It standardizes the management and interchange of context information within a variety of key business protocols.
- Coordination also alleviates the need for services to retain state.
- Statelessness is a key service-orientation principle applied to services for use within SOAs.
- Coordination reinforces this quality by assuming responsibility for the management of context information.

Coordination as it relates to other parts of SOA

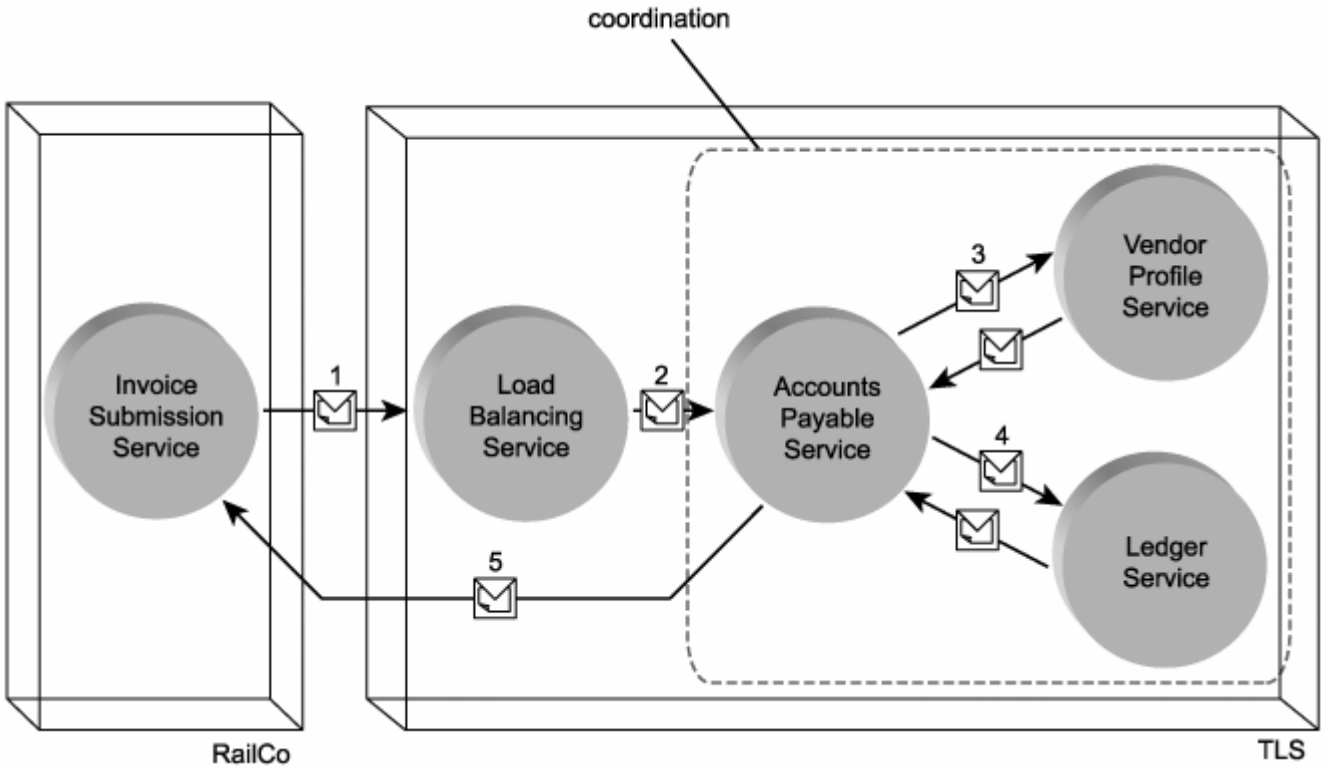


Case Study

- In the previous case study example, we established the individual process steps that comprised a complex activity. Once the processing of this activity enters the TLS environment, TLS employs a context management system to coordinate the flow of the message through its internal services.
- As shown in the next figure, coordination is applied to the following steps:

3. The Accounts Payable Service uses the Vendor Profile Service to validate the invoice header data. If the data is valid, the invoice document is attached to the vendor account.
4. The Accounts Payable Service extracts taxes and shipping fees from the invoice document. These values, along with the invoice total, are submitted to the Ledger Service. The Ledger Service is responsible for updating the General Ledger and numerous sub-ledgers, such as the Accounts Payable Ledger.

The TLS Accounts Payable, Vendor Profile, and Ledger Services being managed by a coordination



Case Study

- The atomic transaction coordination type is used to coordinate these processing steps, as explained in the next two case studies.

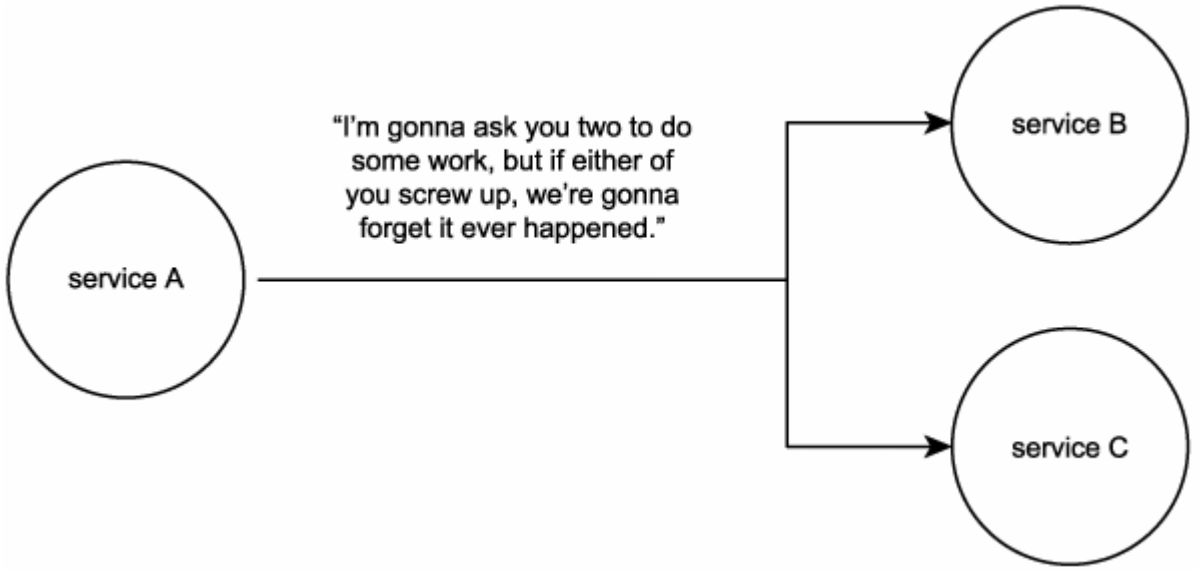
Summary of Key Points

- Complex activities tend to introduce the requirement for context data and the subsequent need for this data to be managed and coordinated at runtime.
- WS-Coordination provides a context management framework using a standardized service composition spearheaded by a coordinator service.
- Specialized implementations of this framework are realized through the use of coordination types, such as WS-AtomicTransaction and WS-BusinessActivity.
- By introducing an activity management layer to SOA, coordination promotes service statelessness and supports the controlled composition of complex activities.

Atomic transactions

- Transactions have been around for almost as long as automated computer solutions have existed.
- When managing certain types of corporate data, the need to wrap a series of changes into a single action is fundamental to many business process requirements.
- Atomic transactions implement the familiar commit and rollback features to enable cross-service transaction support.
- The concepts discussed in this section were derived from the WS-AtomicTransaction specification, which defines protocols for use with WS-Coordination.
- (For a simple example of how the WS-AtomicTransaction coordination type is referenced as part of a SOAP header, see the WS-Coordination overview section)

Atomic transactions apply an all-or-nothing requirement to work performed as part of an activity



ACID transactions

- The protocols provided by the WS-AtomicTransaction specification enable cross-service transaction functionality comparable to the ACID-compliant transaction features found in most distributed application platforms.
- For those of you who haven't yet worked with ACID transactions, let's quickly recap this important standard. The term "ACID" is an acronym representing the following four required characteristics of a traditional transaction:
- **Atomic** Either all of the changes within the scope of the transaction succeed, or none of them succeed. This characteristic introduces the need for the rollback feature that is responsible for restoring any changes completed as part of a failed transaction to their original state.
- **Consistent** None of the data changes made as a result of the transaction can violate the validity of any associated data models. Any violations result in a rollback of the transaction.
- **Isolated** If multiple transactions occur concurrently, they may not interfere with each other. Each transaction must be guaranteed an isolated execution environment.
- **Durable** Upon the completion of a successful transaction, changes made as a result of the transaction can survive subsequent failures.

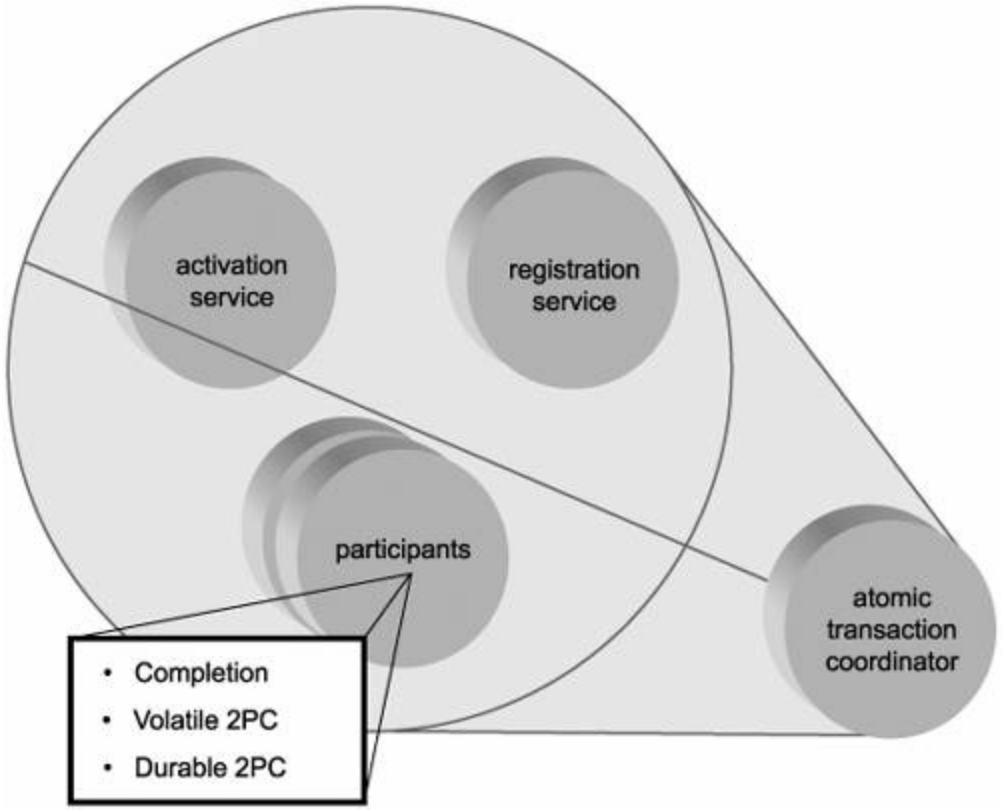
Atomic transaction protocols

- WS-AtomicTransaction is a coordination type, meaning that it is an extension created for use with the WS-Coordination context management framework we covered in the previous section.
- To participate in an atomic transaction, a service first receives a coordination context from the activation service. It can subsequently register for available atomic transaction protocols.
- The following primary transaction protocols are provided:
- A Completion protocol, which is typically used to initiate the commit or abort states of the transaction.
- The Durable 2PC protocol for which services representing permanent data repositories should register.
- The Volatile 2PC protocol to be used by services managing non-persistent (temporary) data.
- Most often these protocols are used to enable a two-phase commit (2PC) that manages an atomic transaction across multiple service participants.

The atomic transaction coordinator

- When WS-AtomicTransaction protocols are used, the coordinator controller service can be referred to as an atomic transaction coordinator.
- This particular implementation of the WS-Coordination coordinator service represents a specific service model.
- The atomic transaction coordinator plays a key role in managing the participants of the transaction process and in deciding the transaction's ultimate outcome.

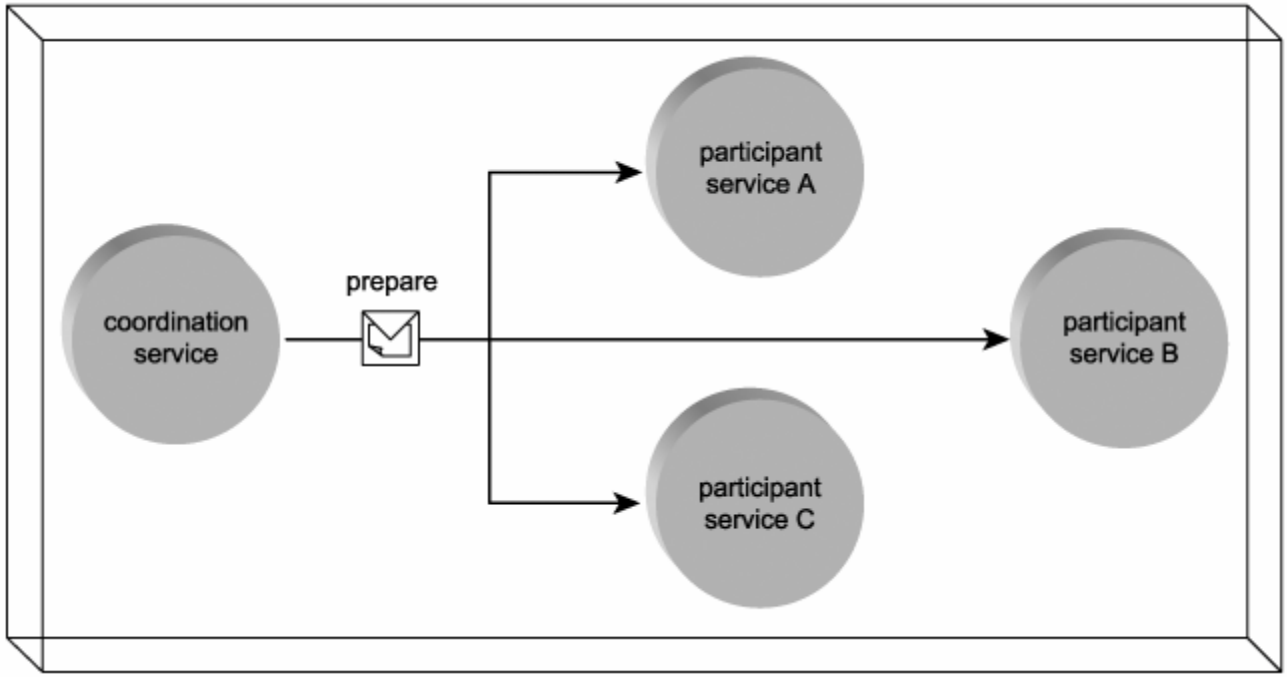
The atomic transaction coordinator service model



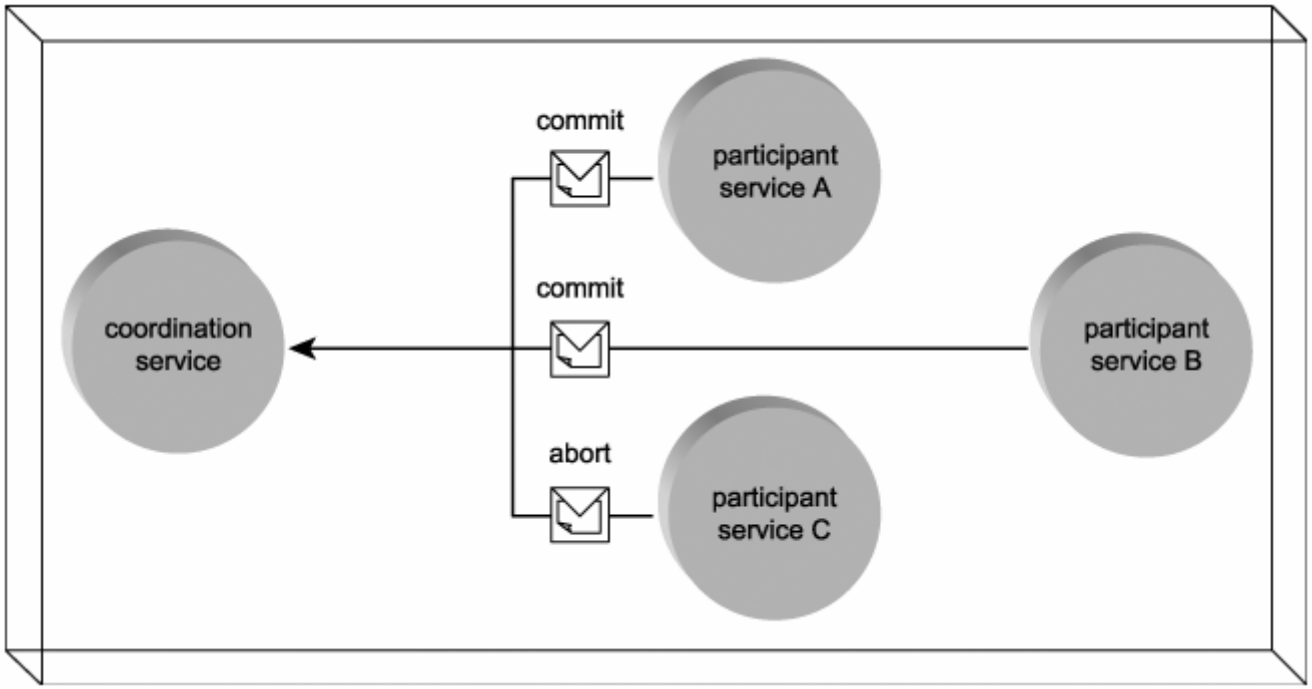
The atomic transaction process

- As previously mentioned, the atomic transaction coordinator is tasked with the responsibility of deciding the outcome of a transaction. It bases this decision on feedback it receives from all of the transaction participants.
- The collection of this feedback is separated into two phases.
- During the prepare phase, all participants are notified by the coordinator, and each is asked to prepare and then issue a vote. Each participant's vote consists of either a "commit" or "abort" request.

The coordinator requesting that transaction participants prepare to vote



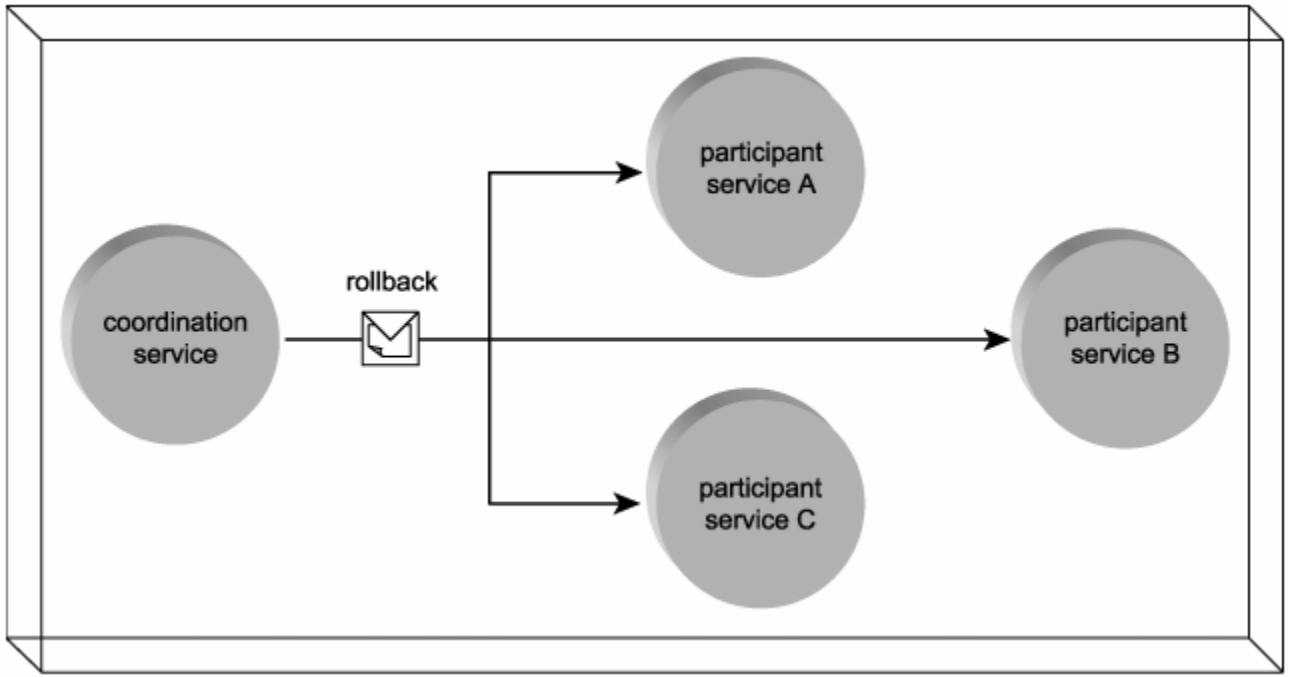
The transaction participants voting on the outcome of the atomic transaction



The atomic transaction process

- After the votes are collected, the atomic transaction coordinator enters the commit phase.
- It now reviews all votes and decides whether to commit or rollback the transaction.
- The conditions of a commit decision are simple:
  - if all votes are received and if all participants voted to commit, the coordinator declares the transaction successful, and the changes are committed.
  - However, if any one vote requests an abort, or if any of the participants fail to respond, then the transaction is aborted, and all changes are rolled back.

The coordinator aborting the transaction and notifying participants to rollback all changes



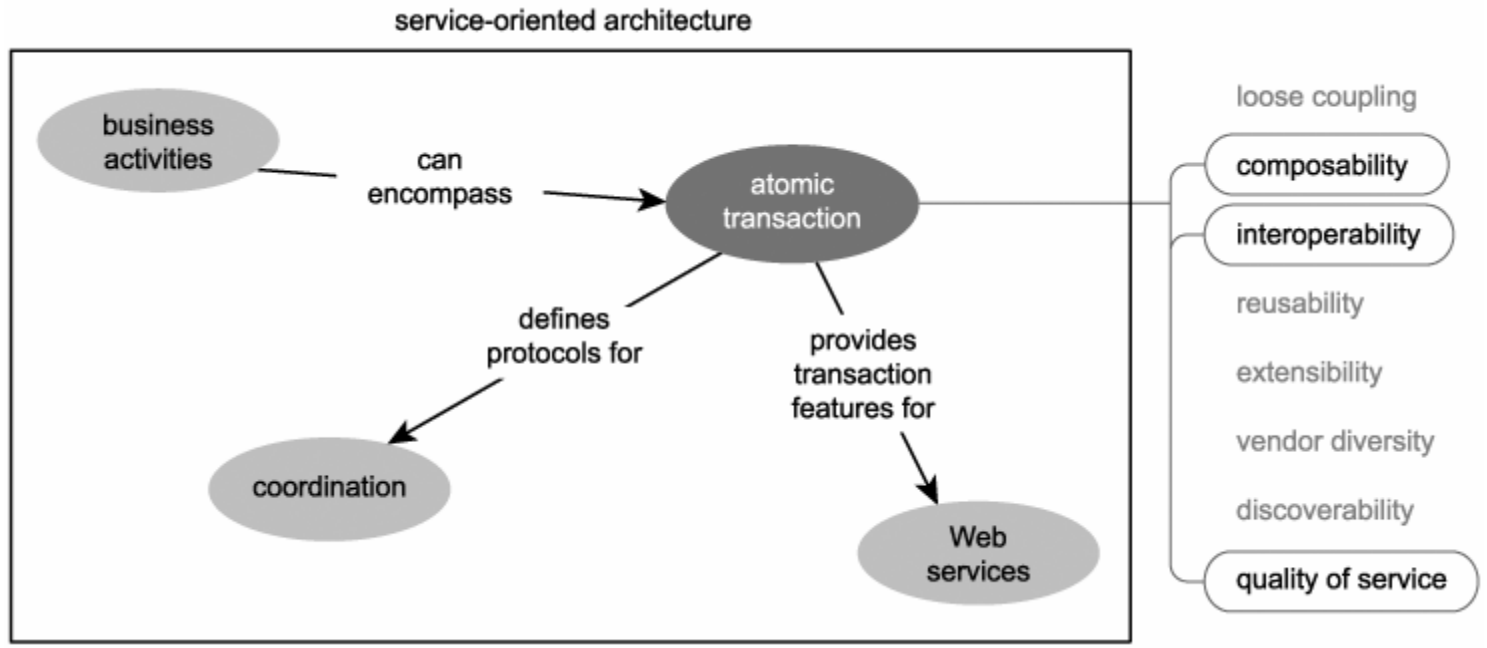
Atomic transactions and SOA

- Much of the transactional functionality implemented in service-oriented solutions is done so among the components that execute an activity on behalf of a single service.
- However, as more services emerge within an organization and as service compositions become more commonplace, the need to move transaction boundaries into cross-service interaction scenarios increases.
- Being able to guarantee an outcome of an activity is a key part of enterprise-level computing, and atomic transactions therefore play an important role in ensuring quality of service.

Atomic transactions and SOA

- Not only do atomic transactional capabilities lead to a robust execution environment for SOA activities, they promote interoperability when extended into integrated environments.
- This allows the scope of an activity to span different solutions built with different vendor platforms, while still being assured a guaranteed all-or-nothing outcome.
- Assuming, of course, that WS-AtomicTransaction is supported by the affected applications, this option broadens the application of the two-phase commit protocol beyond traditional application boundaries (thus, supporting service interoperability). Next figure illustrates how atomic transactions support these aspects of SOA.

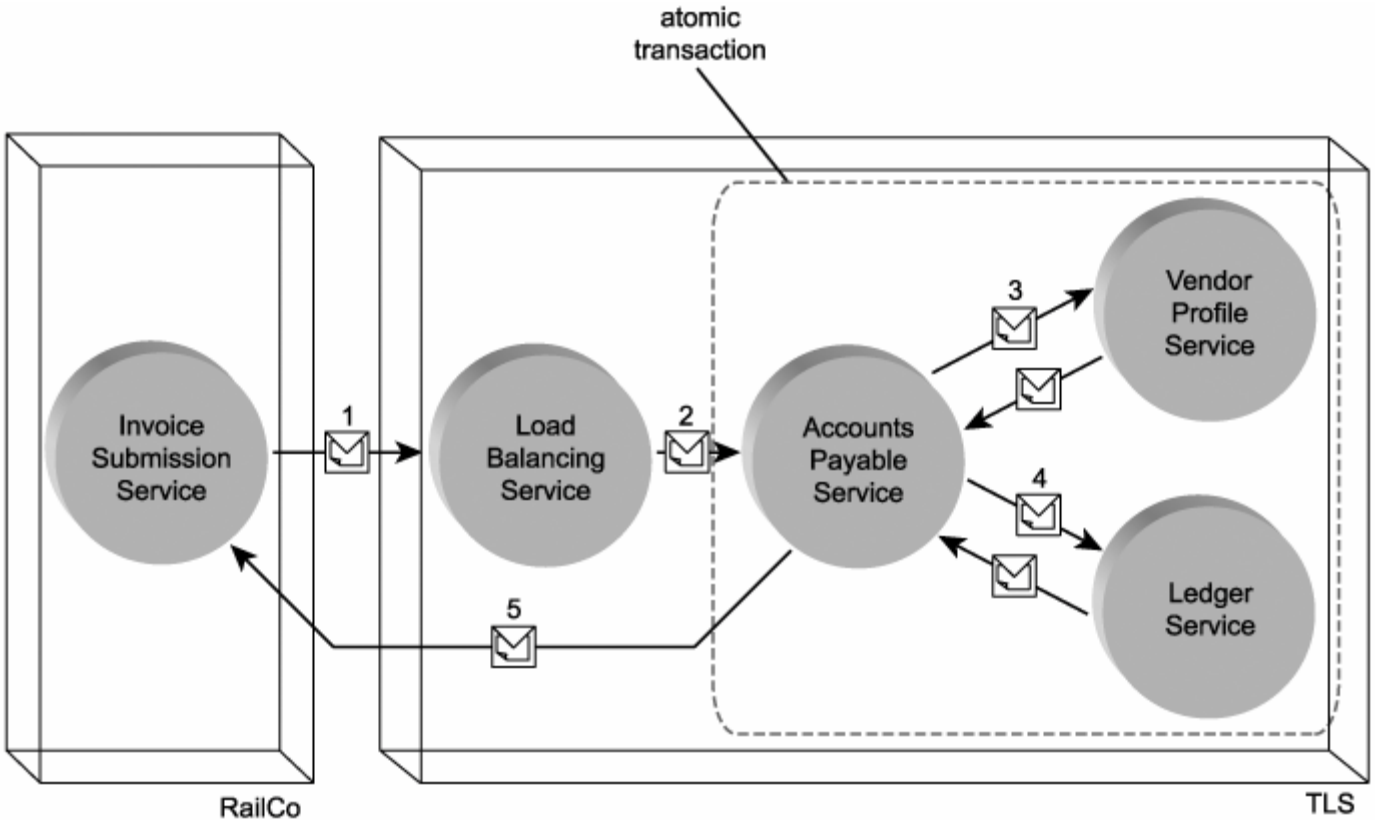
Atomic transaction relating to other parts of SOA



Case Study

- Continuing with our previous case study example, a look under the hood reveals that TLS actually wraps Steps 3 and 4 into an atomic transaction. This guarantees that should any one update fail by any of the services involved in this composition, all previous updates performed (as of Step 3) will be rolled back.
- To accomplish this, TLS relies on WS-Coordination, as implemented by the WS-AtomicTransaction coordination type. It utilizes the context coordinator along with the Complete transaction protocol to incorporate ACID-type transaction features into this complex activity.
- To accomplish this, TLS relies on WS-Coordination, as implemented by the WS-AtomicTransaction coordination type.
- It utilizes the context coordinator along with the Complete transaction protocol to incorporate ACID-type transaction features into this complex activity.

All changes made by the TLS Accounts Payable, Vendor Profile, and Ledger Services are under the control of an atomic transaction



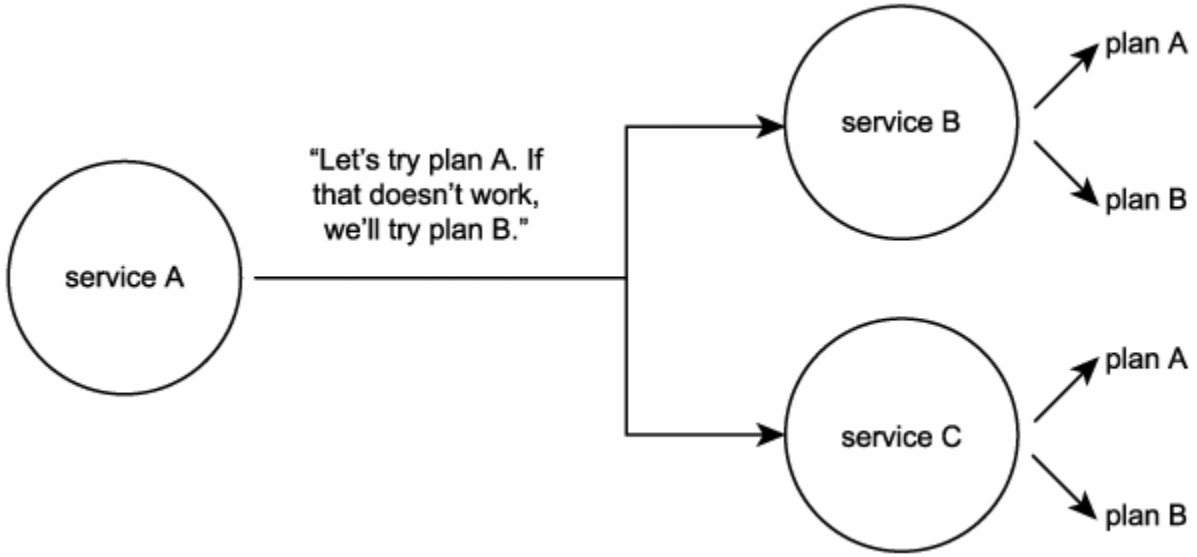
Summary of Key Points

- WS-AtomicTransaction is a coordination type that supplies three coordination protocols that can be used to achieve two-phase commit transactions across multiple service participants.
- The atomic transaction coordinator makes the ultimate decision to commit or rollback a transaction. This decision is based on votes collected from participants.
- Contemporary SOAs can incorporate cross-service, ACID-type transaction features by using WS-AtomicTransaction.

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.
- What distinguishes a business activity from a regular complex activity is that its participants are required to follow specific rules defined by protocols. Business activities primarily differ from the also protocol-based atomic transactions in how they deal with exceptions and in the nature of the constraints introduced by the protocol rules.
- For instance, business activity protocols do not offer rollback capabilities. Given the potential for business activities to be long-running, it would not be realistic to expect ACID-type transaction functionality. Instead, business activities provide an optional compensation process that, much like a "plan B," can be invoked when exception conditions are encountered.
- Note: The concepts discussed in this section are derived from the WS-BusinessActivity specification, which (like WS-AtomicTransaction) provides protocols for use with WS-Coordination. The WS-Coordination overview section contains a brief example of how a coordination type referencing WS-BusinessActivity exists within a SOAP header.

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



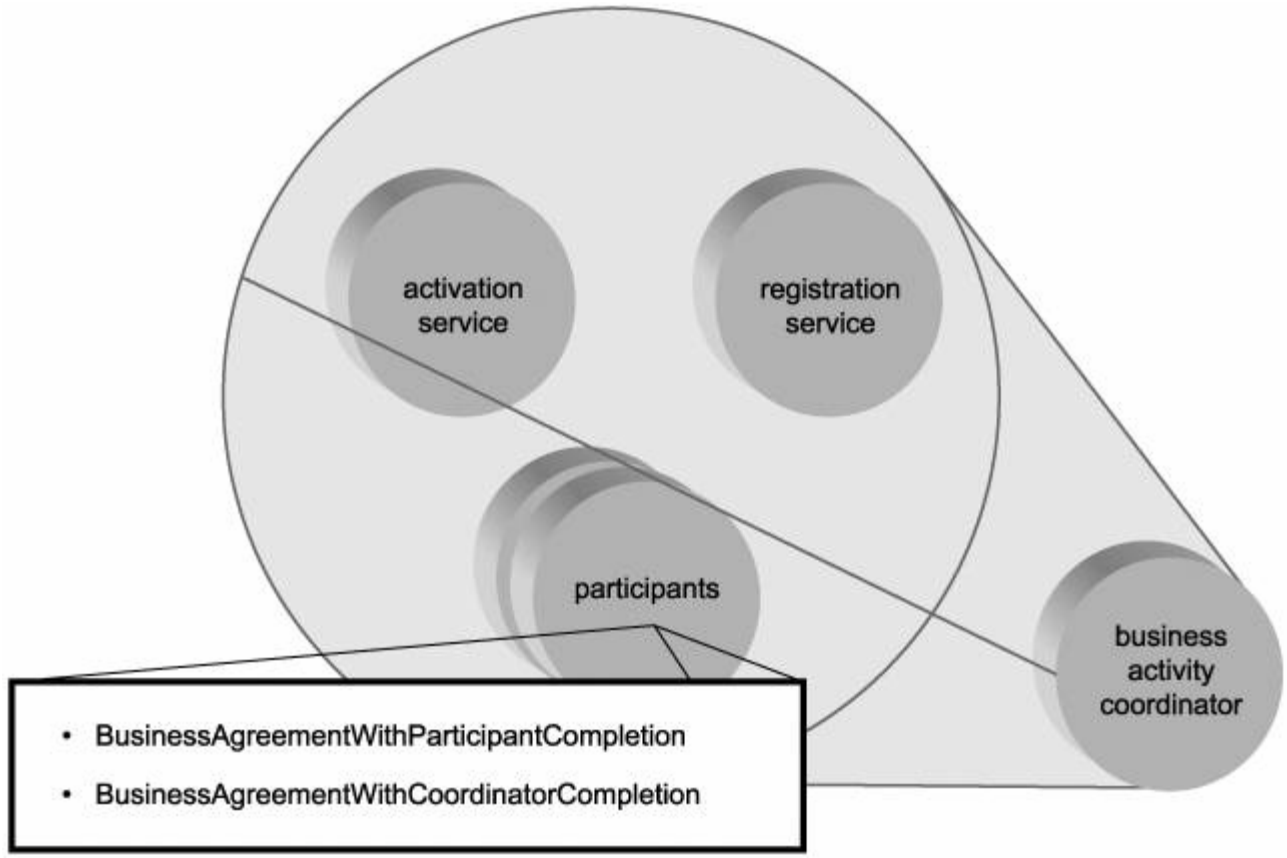
Business activity protocols

- As with WS-AtomicTransaction, WS-BusinessActivity is a coordination type designed to leverage the WS-Coordination context management framework.
- It provides two very similar protocols, each of which dictates how a participant may behave within the overall business activity.
- The BusinessAgreementWithParticipantCompletion protocol, which allows a participant to determine when it has completed its part in the business activity.
- The BusinessAgreementWithCoordinatorCompletion protocol, which requires that a participant rely on the business activity coordinator to notify it that it has no further processing responsibilities.
- Business activity participants interact with the standard WS-Coordination coordinator composition to register for a protocol, as was explained in the previous Coordination section.

The business activity coordinator

- When its protocols are used, the WS-Coordination controller service assumes a role specific to the coordination type in this case it becomes a business activity coordinator.
- As explained in the previous section, this coordinator has varying degrees of control in the overall activity, based on the coordination protocols used by the participants.

The business activity coordinator service model



Business activity states

- During the lifecycle of a business activity, the business activity coordinator and the activity participants transition through a series of states. The actual point of transition occurs when special notification messages are passed between these services.
- For example, a participant can indicate that it has completed the processing it was required to perform as part of the activity by issuing a completed notification. This moves the participant from an active state to a completed state. The coordinator may respond with a close message to let the participant know that the business activity is being successfully completed.
- However, if things don't go as planned during the course of a business activity, one of a number of options are available.
- Participants can enter a compensation state during which they attempt to perform some measure of exception handling.
- This generally invokes a separate compensation process that could involve a series of additional processing steps.
- A compensation is different from an atomic transaction in that it is not expected to rollback any changes performed by the participating services; its purpose is generally to execute plan B when plan A fails.

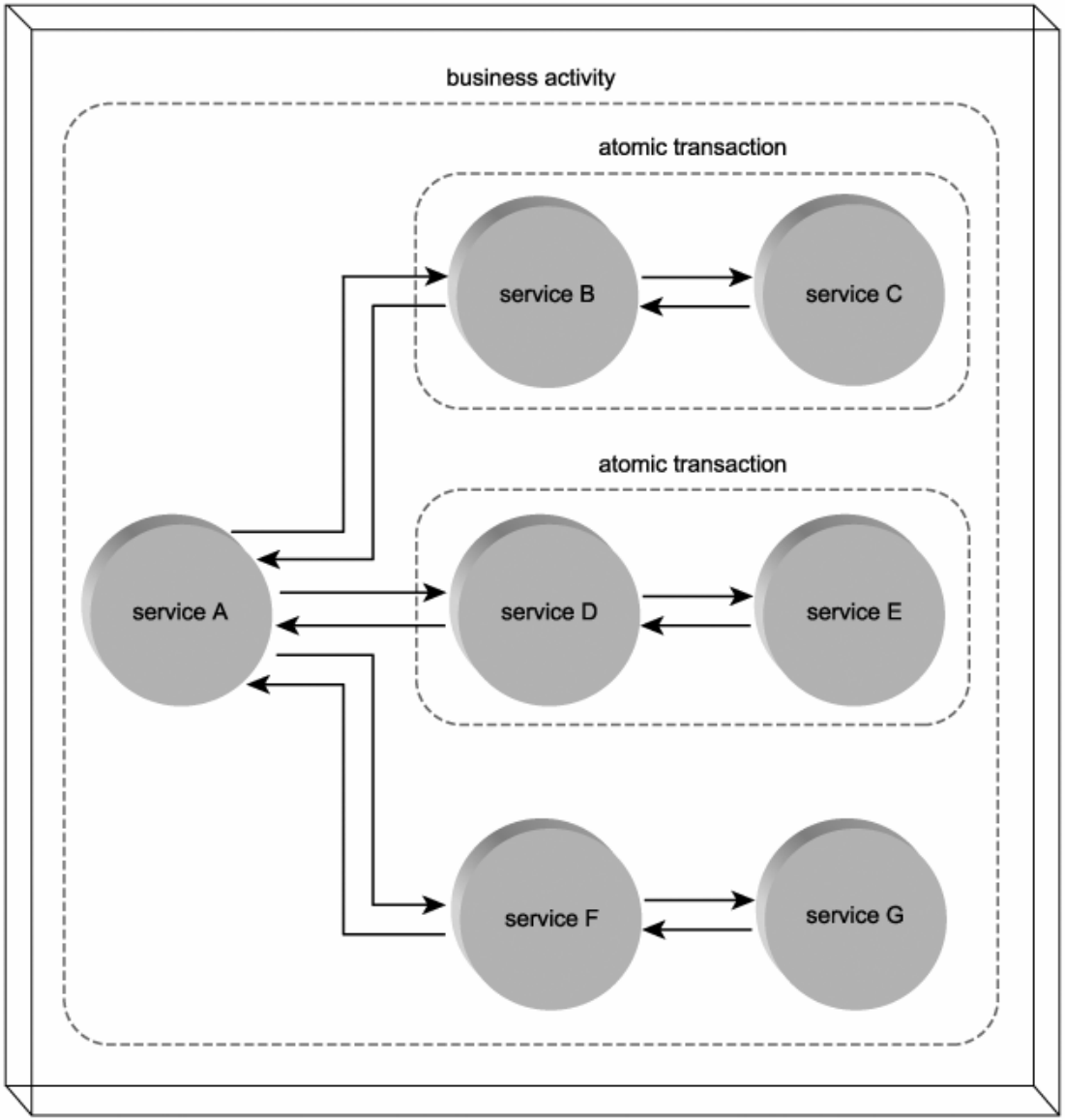
Business activity states

- Alternatively, a cancelled state can be entered. This typically results in the termination of any further processing outside of the cancellation notifications that need to be distributed.
- What also distinguishes business activities from atomic transactions is the fact that participating services are not required to remain participants for the duration of the activity.
- Because there is no tight control over the changes performed by services, they may leave the business activity after their individual contributions have been performed.
- When doing so, participants enter an exit state by issuing an exit notification message to the business activity coordinator.
- These and other states are defined in a series of state tables documented as part of the WS-BusinessActivity specification. These tables establish the fundamental rules of the business activity protocols by determining the sequence and conditions of allowable states.

Business activities and atomic transactions

- It is important to note that the use of a business activity does not exclude the use of atomic transactions.
- In fact, it is likely that a long-running business activity will encompass the execution of several atomic transactions during its lifetime.

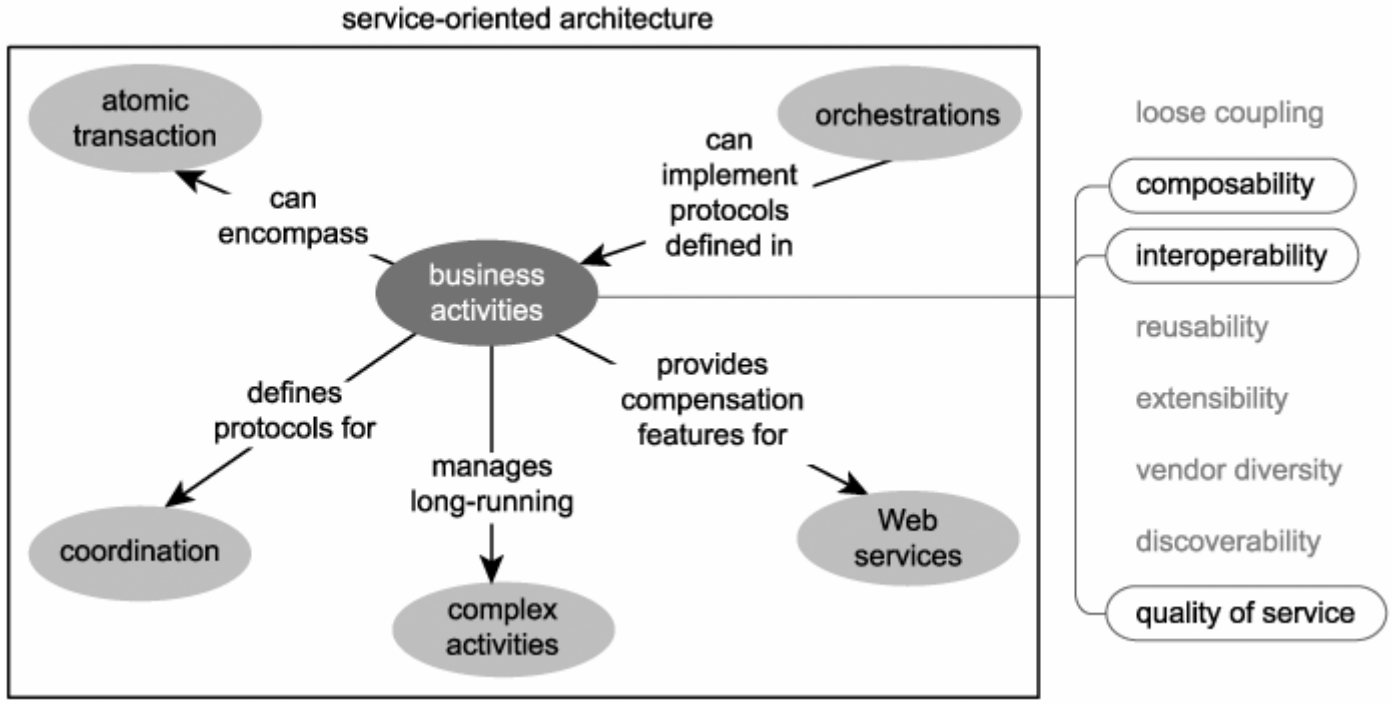
Two atomic transactions residing within the scope of a business activity



Business activities and SOA

- Business activities fully complement the composable nature of SOA by tracking and regulating complex activities while also allowing them to carry on for long periods of time.
- Service autonomy and statelessness are preserved by permitting services to participate within an activity for only the duration they are absolutely required to.
- This also allows for the design of highly adaptive business activities wherein the participants can augment activity or process logic to accommodate changes in the business tasks being automated.
- Through the use of the compensation process, business activities increase SOA's quality of service by providing built-in fault handling logic.

A business activity relating to other parts of SOA



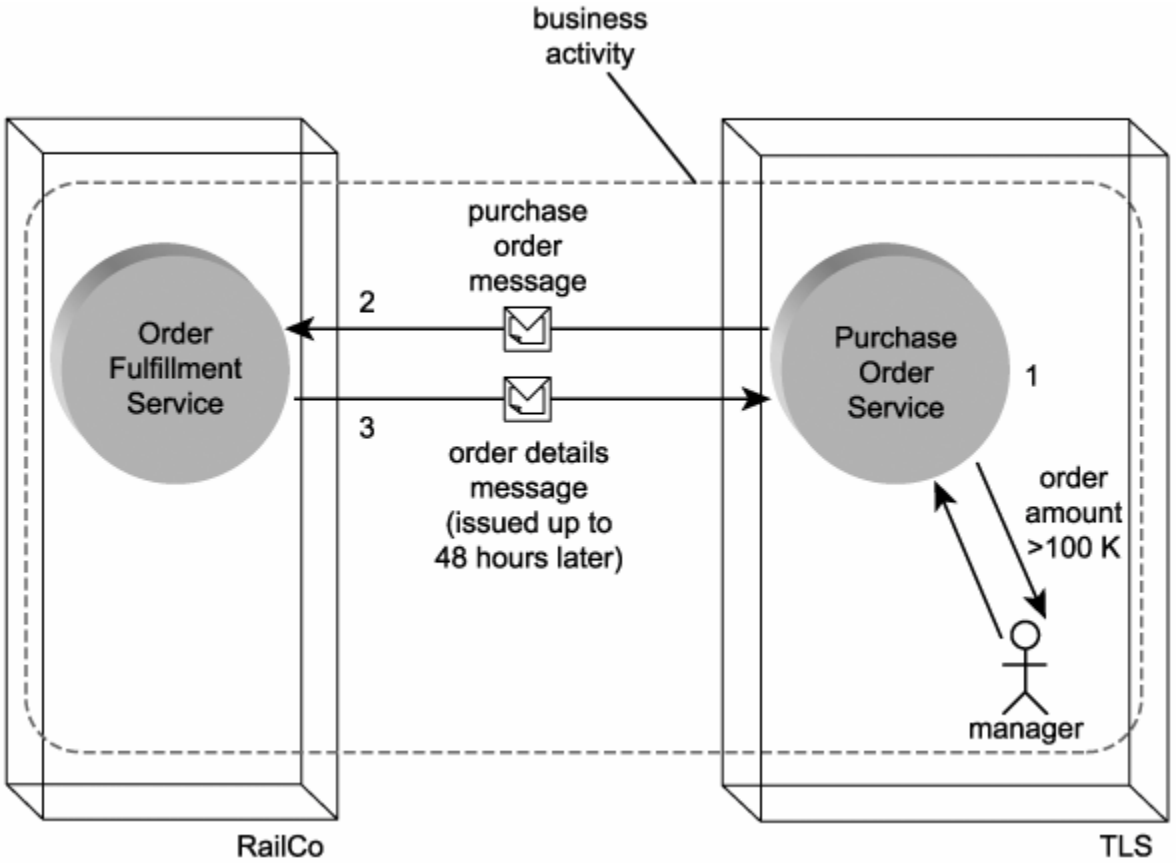
Business activities and SOA

- As with WS-AtomicTransaction, support of the WS-BusinessActivity extension by multiple solutions promotes inherent interoperability and can greatly simplify integration architectures.
- Business activities take this a few steps further, though, by allowing the scope of the activity to include interaction with outside business partners.
- (Note that there is nothing restricting atomic transactions from being utilized across organizations. However, business activities are typically more suitable for this type of communication.)

Case Study

- The TLS Purchase Order Submission Process involves the TLS Purchase Order Service acting as the initial sender, responsible for transmitting a SOAP message containing PO details to the RailCo Order Fulfillment Service (Step 2).
- Further complicating this process is the fact that purchase orders for amounts exceeding 100,000 require a separate approval step by a TLS manager (Step 1).

The TLS Purchase Order Submission Process wrapped in a long-running business activity and spanning two organizations (two participants).



Case Study

- Vendors are given a period of 48 hours during which they are expected to check for the availability of the requested inventory and respond with either an acknowledgement message indicating that the order will be shipped or a message explaining that an order cannot be filled (or only partially filled), along with any relevant back-order information (Step 3).
- Even though the PO Submission Process also can be classified as a complex activity, it is different from our previous case study example, as follows:
  - It can take a long time for this activity to complete.
  - It includes a manual review step (which can result in an approval or a rejection of the purchase order request).
- To best manage this complex activity, TLS has used the same WS-Coordination context management framework applied in the previous case study example, only this time the WS-BusinessActivity coordination type is chosen instead.

Summary of Key Points

- Business activities manage complex, long-running activities that can vary in scope and in the amount of participating services.
- WS-BusinessActivity builds on the WS-Coordination context management framework by providing two protocols for which activity participants can register.
- Participants and the business activity coordinator progress through a series of states during the lifespan of a business activity. State transition is accomplished through the exchange of notification messages.
- Long-running activities are commonplace in contemporary SOAs, which positions WS-BusinessActivity as an important specification for the controlled management of logic that underlies these types of complex activities.

Orchestration

- Organizations that already have employed enterprise application integration (EAI) middleware products to automate business processes or to integrate various legacy environments will likely already be familiar with the concept of orchestration.
- In these systems, a centrally controlled set of workflow logic facilitates interoperability between two or more different applications.
- A common implementation of orchestration is the hub-and-spoke model that allows multiple external participants to interface with a central orchestration engine.
- One of the driving requirements behind the creation of these solutions was to accommodate the merging of large business processes.
- With orchestration, different processes can be connected without having to redevelop the solutions that originally automated the processes individually.
- Orchestration bridges this gap by introducing new workflow logic.
- Further, the use of orchestration can significantly reduce the complexity of solution environments.
- Workflow logic is abstracted and more easily maintained than when embedded within individual solution components.

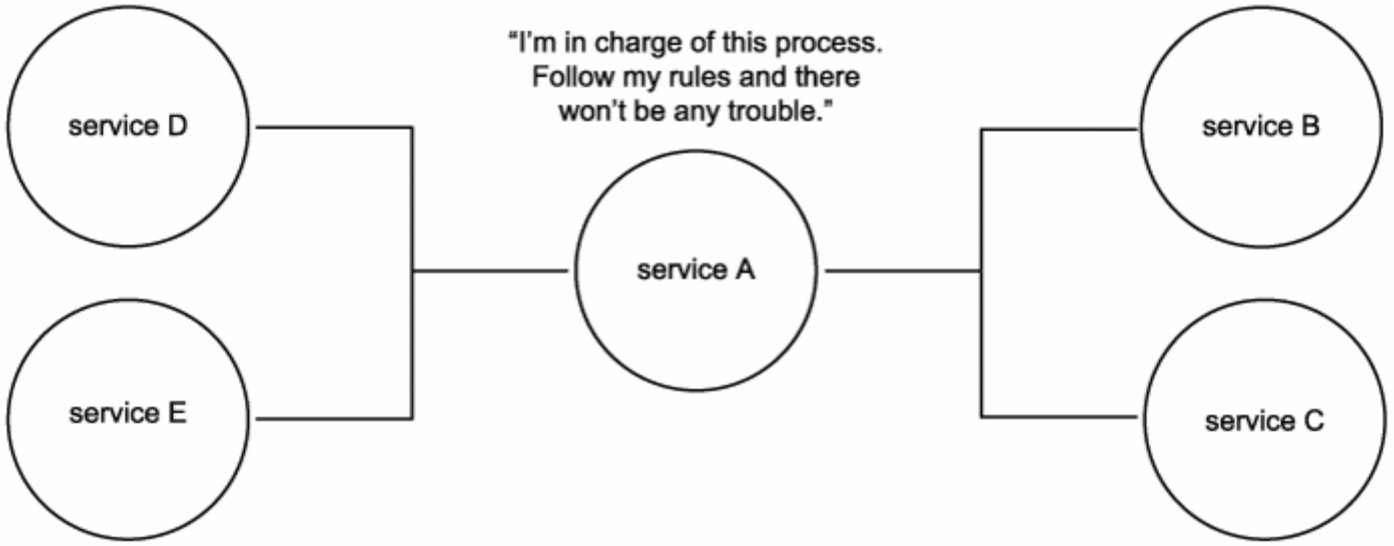
Orchestration

- The role of orchestration broadens in service-oriented environments. Through the use of extensions that allow for business process logic to be expressed via services, orchestration can represent and express business logic in a standardized, services-based venue.
- When building service-oriented solutions, this provides an extremely attractive means of housing and controlling the logic representing the process being automated.
- Orchestration further leverages the intrinsic interoperability sought by service designs by providing potential integration endpoints into processes. A key aspect to how orchestration is positioned within SOA is the fact that orchestrations themselves exist as services.
- Therefore, building upon orchestration logic standardizes process representation across an organization, while addressing the goal of enterprise federation and promoting service-orientation.

Orchestration

- A primary industry specification that standardizes orchestration is the Web services Business Process Execution Language (WS-BPEL).
- WS-BPEL is a key second-generation extension and therefore uses its concepts and terminology as the basis for a number of discussions relating to business process modeling.
- Note:
- WS-BPEL is the most recent name given to this specification, which also is known as BPEL4WS and just BPEL. For an overview of the primary parts of the WS-BPEL language and a discussion of how the name change came about.

An orchestration controls almost every facet of a complex activity



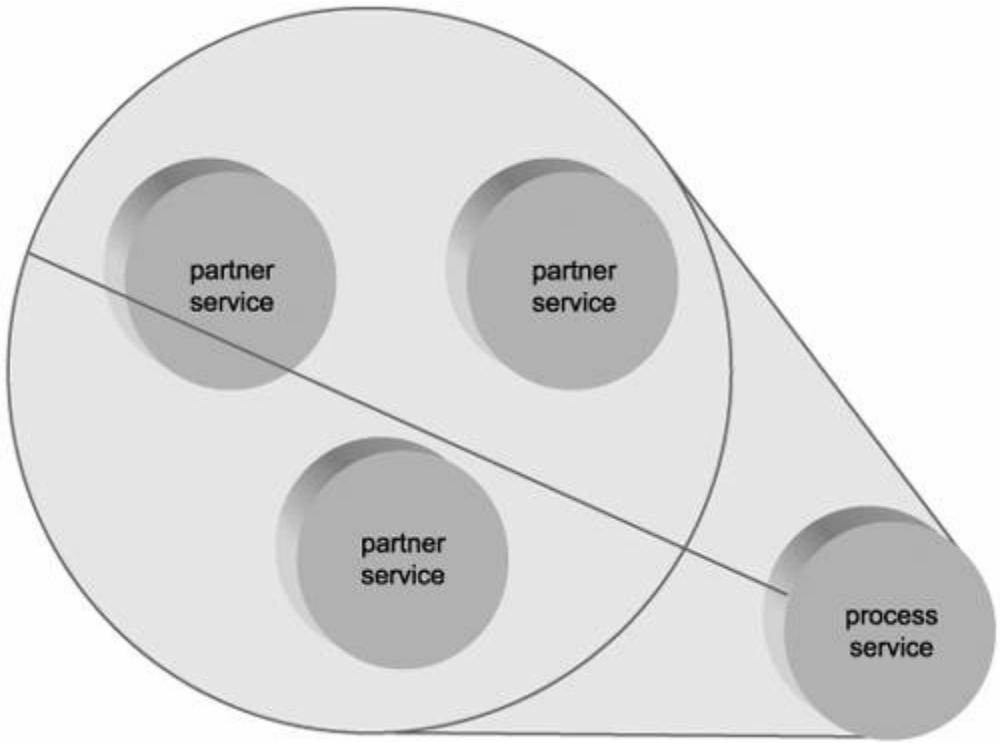
Business protocols and process definition

- The workflow logic that comprises an orchestration can consist of numerous business rules, conditions, and events.
- Collectively, these parts of an orchestration establish a business protocol that defines how participants can interoperate to achieve the completion of a business task.
- The details of the workflow logic encapsulated and expressed by an orchestration are contained within a process definition.

Process services and partner services

- Identified and described within a process definition are the allowable process participants.
- First, the process itself is represented as a service, resulting in a process service (which happens to be another one of our service models)

A process service coordinating and exposing functionality from three partner services

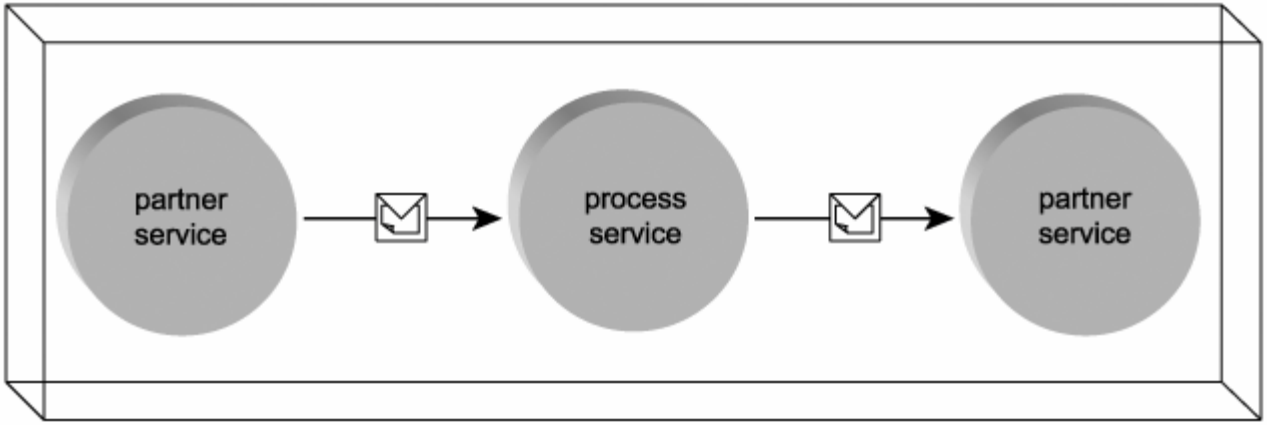


Process services and partner services

Partner services or partner links:

- Other services allowed to interact with the process service are identified as partner services or partner links.
- Depending on the workflow logic, the process service can be invoked by an external partner service, or it can invoke other partner services.

The process service, after first being invoked by a partner service, then invokes another partner service



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).

Sequences, flows, and links

- Basic and structured activities can be organized so that the order in which they execute is predefined.

A sequence

- aligns groups of related activities into a list that determines a sequential execution order. Sequences are especially useful when one piece of application logic is dependent on the outcome of another.

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.

Sequences, flows, and links

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.

Orchestrations and activities

- As we defined earlier, an activity is a generic term that can be applied to any logical unit of work completed by a service-oriented solution.
- The scope of a single orchestration, therefore, can be classified as a complex, and most likely, long-running activity.

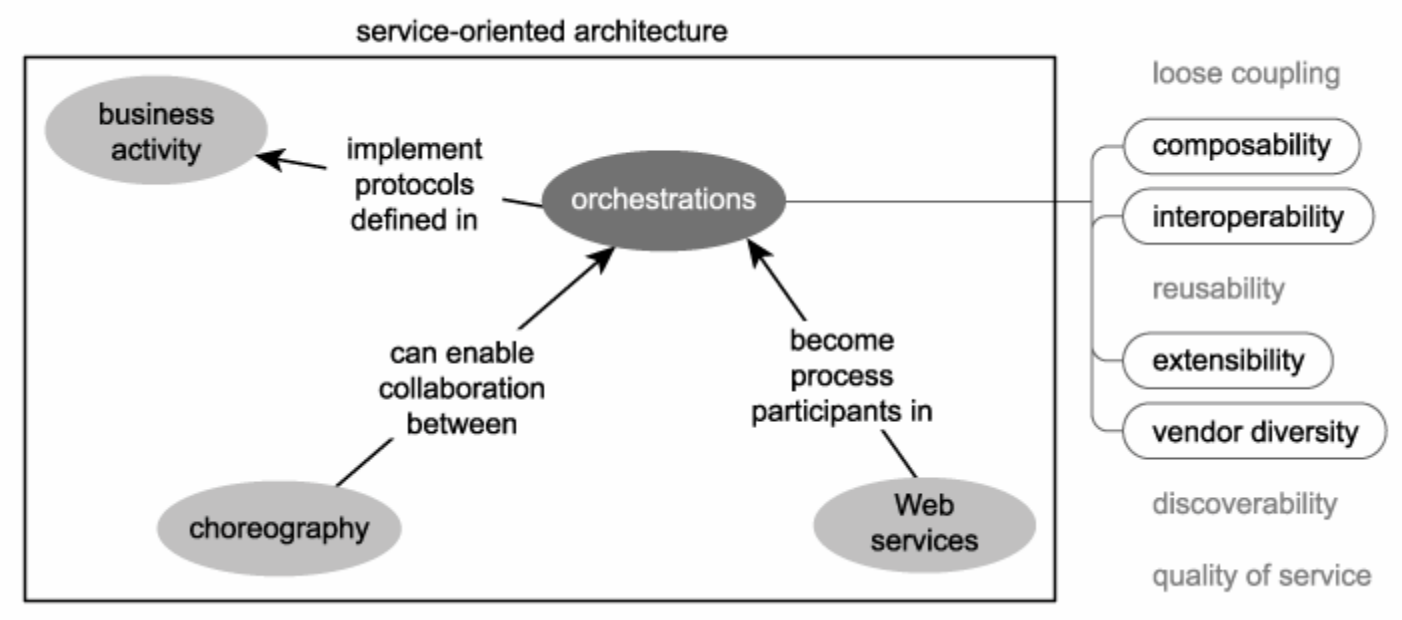
Orchestration and coordination

- Orchestration, as represented by WS-BPEL, can fully utilize the WS-Coordination context management framework by incorporating the WS-BusinessActivity coordination type.
- This specification defines coordination protocols designed to support complex, long-running activities.

Orchestration and SOA

- Business process logic is at the root of automation solutions. Orchestration provides an automation model where process logic is centralized yet still extensible and composable.
- Through the use of orchestrations, service-oriented solution environments become inherently extensible and adaptive.
- Orchestrations themselves typically establish a common point of integration for other applications, which makes an implemented orchestration a key integration enabler.

Orchestration relating to other parts of SOA



Orchestration and SOA

- These qualities lead to increased organizational agility because:
  - The workflow logic encapsulated by an orchestration can be modified or extended in a central location.
  - Positioning an orchestration centrally can significantly ease the merging of business processes by abstracting the glue that ties the corresponding automation solutions together.
  - By establishing potentially large-scale service-oriented integration architectures, orchestration, on a fundamental level, can support the evolution of a diversely federated enterprise.

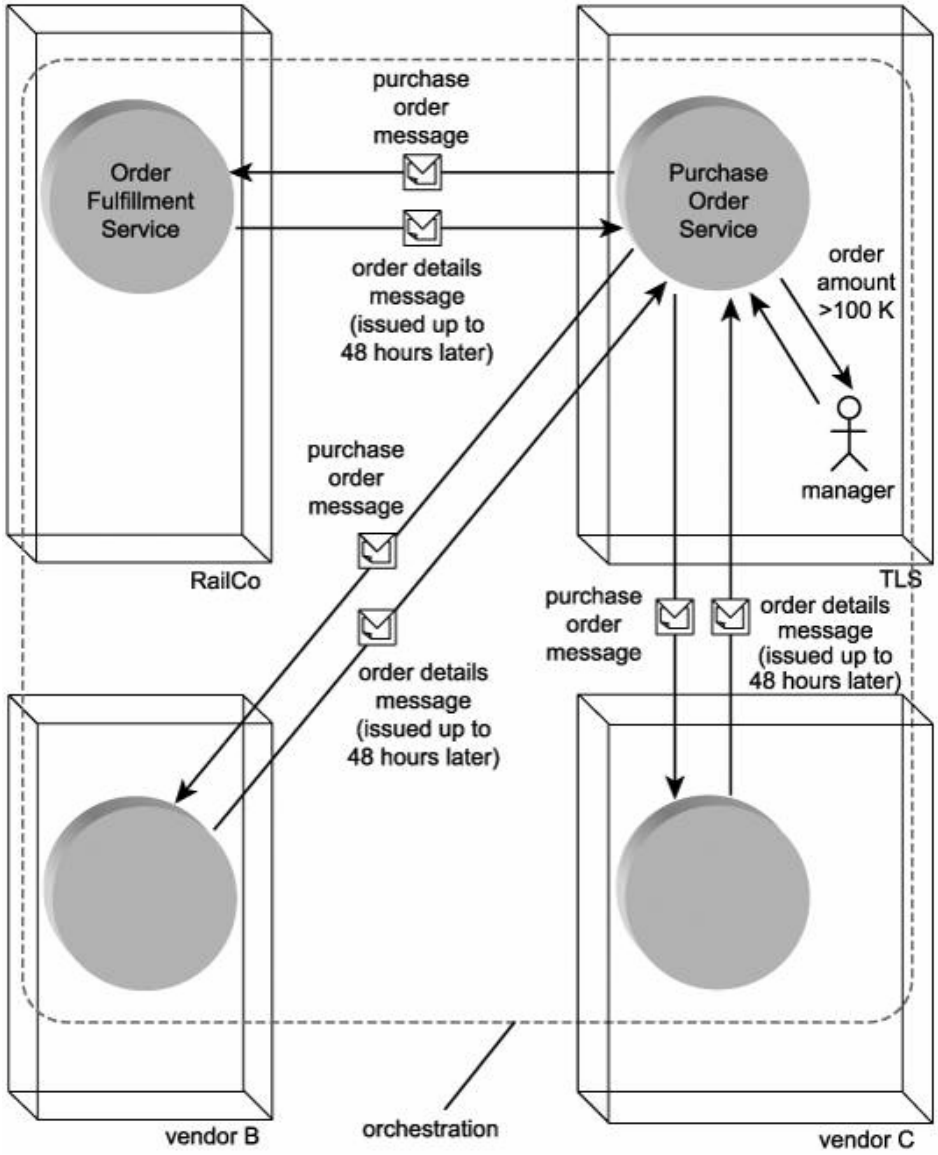
Orchestration and SOA

- Orchestration is a key ingredient to achieving a state of federation within an organization that contains various applications based on disparate computing platforms.
- Advancements in middleware allow orchestration engines themselves to become fully integrated in service-oriented environments.
- The concept of service-oriented orchestration fully leverages all of the concepts we've discussed so far.
- For many environments, orchestrations become the heart of SOA.

Case Study

- The series of steps we wrapped into a business activity in the previous case study example demonstrated how TLS used the WS-BusinessActivity protocol to add context management and exception handling to a long-running, complex activity.
- Even though the scope of a business activity can constitute a business process, it does not provide TLS with a standard means of expressing the underlying workflow logic.
- For that, TLS employs a WS-BPEL orchestration

The extended TLS Purchase Order Submission Process managed by an orchestration and involving numerous potential partner organizations



Case Study

- The orchestration establishes comprehensive process logic that encompasses the business activity and extends it even further to govern additional interaction scenarios with multiple vendor services.
  - For example, when one vendor cannot fulfill an order, the next vendor in line is sent the same purchase order.
  - This cycle is repeated until either one vendor can complete the order in its entirety (within certain price limitations) or until all vendors have been queried.
  - In the latter situation, the system simply assesses the best deal on the table by applying a formula that takes into account the price, percentage of order to be filled, and backorder terms.
- The orchestration logic manages all aspects of the process, including the involvement of multiple vendor partner services, as well as the business activity that kicks in when a PO is processed.

Summary of Key Points

- An orchestration expresses a body of business process logic that is typically owned by a single organization.
- An orchestration establishes a business protocol that formally defines a business process definition.
- The workflow logic within an orchestration is broken down into a series of basic and structured activities that can be organized into sequences and flows.
- Orchestration has been called the "heart of SOA," as it establishes a means of centralizing and controlling a great deal of inter and intra-application logic through a standardized service model.

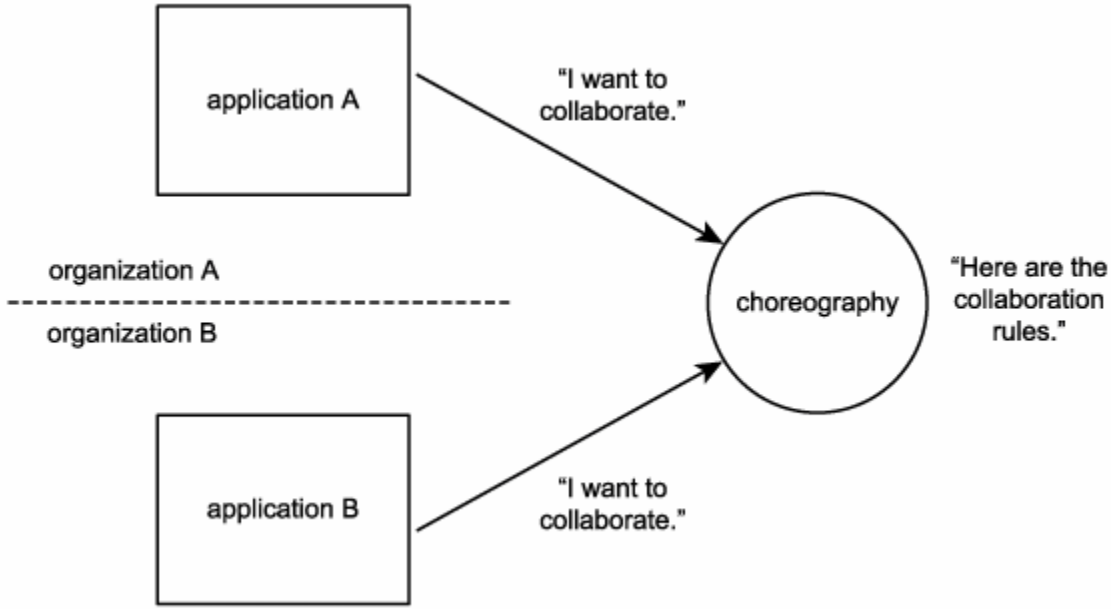
Choreography

- In a perfect world, all organizations would agree on how internal processes should be structured, so that should they ever have to interoperate, they would already have their automation solutions in perfect alignment.
- Though this vision has about a zero percent chance of ever becoming reality, the requirement for organizations to interoperate via services is becoming increasingly real and increasingly complex.
- This is especially true when interoperation requirements extend into the realm of collaboration, where multiple services from different organizations need to work together to achieve a common goal.

Choreography

- The 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.
- It is used to represent the concept of choreography and also the specification from which many of the terms discussed in this section have been derived.

A choreography enables collaboration between its participants



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.
- Note: 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

- Within any given choreography, a Web service assumes one of a number of 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

- Every action that is mapped out within a choreography can be broken down into a series of 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.
- Now that we've defined who can talk with each other, we require a means of establishing the nature of the conversation. Channels do exactly that by defining the characteristics of the message exchange between two specific roles.
- 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.

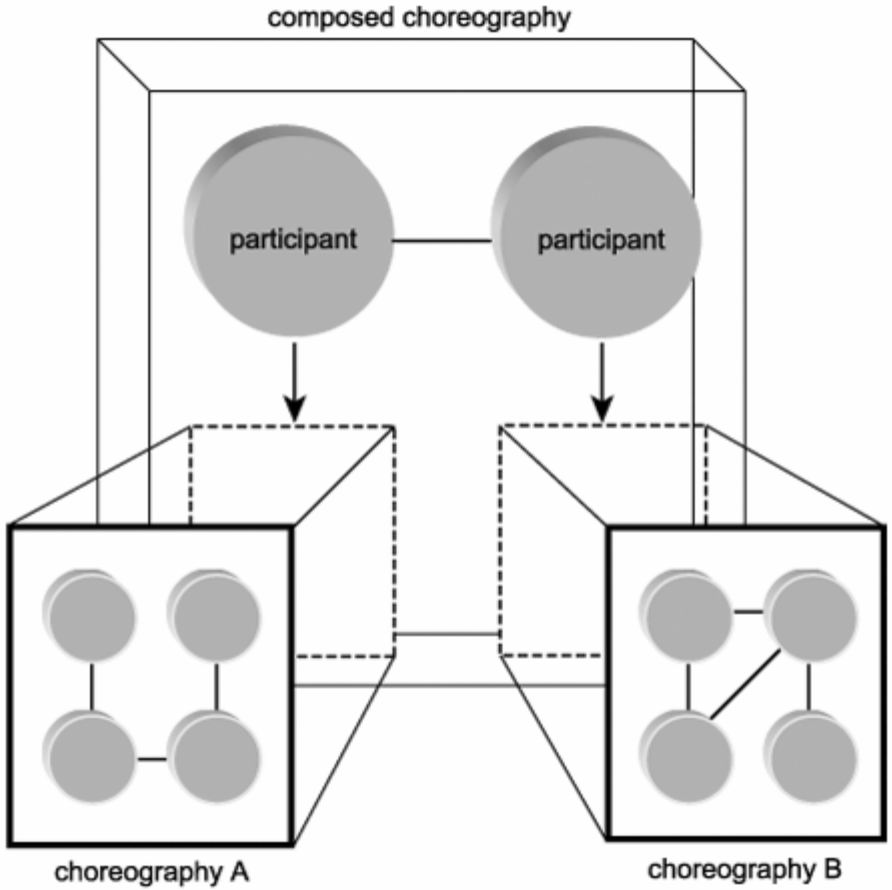
Interactions and work units

- 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. They 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.
- Finally, even though a choreography in effect composes a set of non-specific services to accomplish a task, choreographies themselves can be assembled into larger compositions.

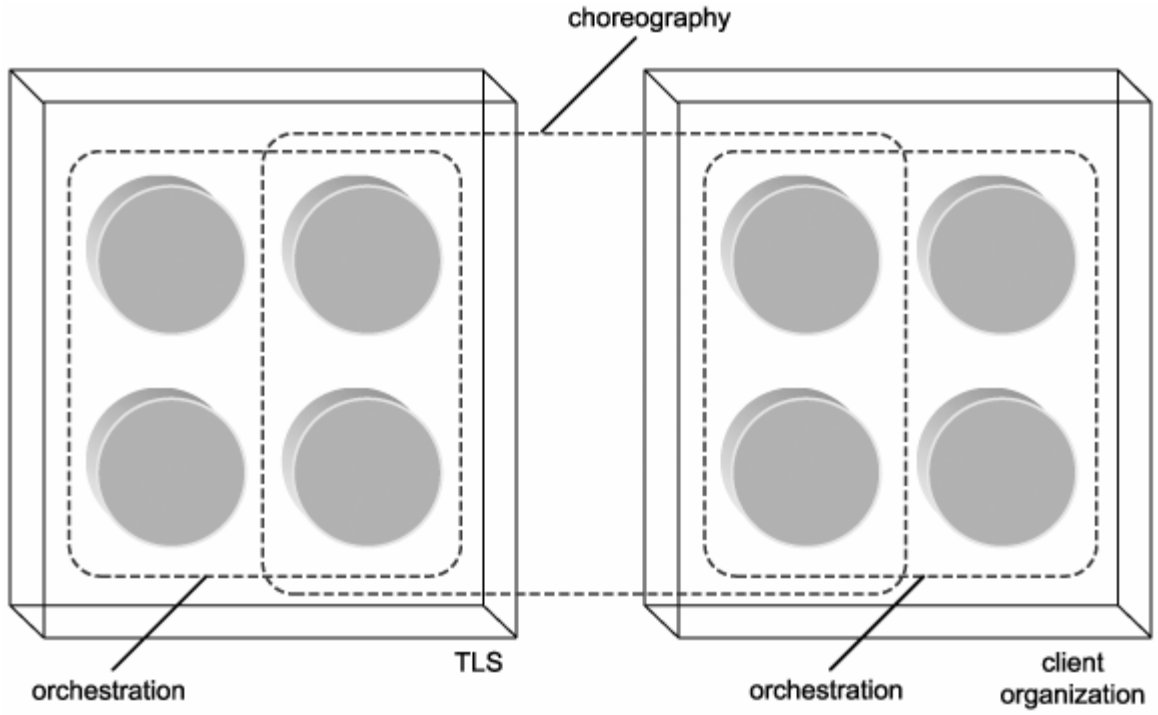
A choreography composed of two smaller choreographies



Orchestrations and choreographies

- While both represent complex message interchange patterns, there is a common distinction that separates the terms "orchestration" and "choreography."
- An orchestration expresses organization-specific business workflow.
- This means that an organization owns and controls the logic behind an orchestration, even if that logic involves interaction with external business partners.
- A choreography, on the other hand, is not necessarily owned by a single entity. It acts as a community interchange pattern used for collaborative purposes by services from different provider entities.

A choreography enabling collaboration between two different orchestrations



Orchestrations and choreographies

- One can view an orchestration as a business-specific application of a choreography.
- This view is somewhat accurate, only it is muddled by the fact that some of the functionality provided by the corresponding specifications (WS-CDL and WS-BPEL) actually overlaps.
- This is a consequence of these specifications being developed in isolation and submitted to separate standards organizations (W3C and OASIS, respectively).

Orchestrations and choreographies

- An orchestration is based on a model where the composition logic is executed and controlled in a centralized manner.
- A choreography typically assumes that there is no single owner of collaboration logic.
- However, one area of overlap between the current orchestration and choreography extensions is the fact that orchestrations can be designed to include multi-organization participants.
- An orchestration can therefore effectively establish cross-enterprise activities in a similar manner as a choreography.
- Again, though, a primary distinction is the fact that an orchestration is generally owned and operated by a single organization.

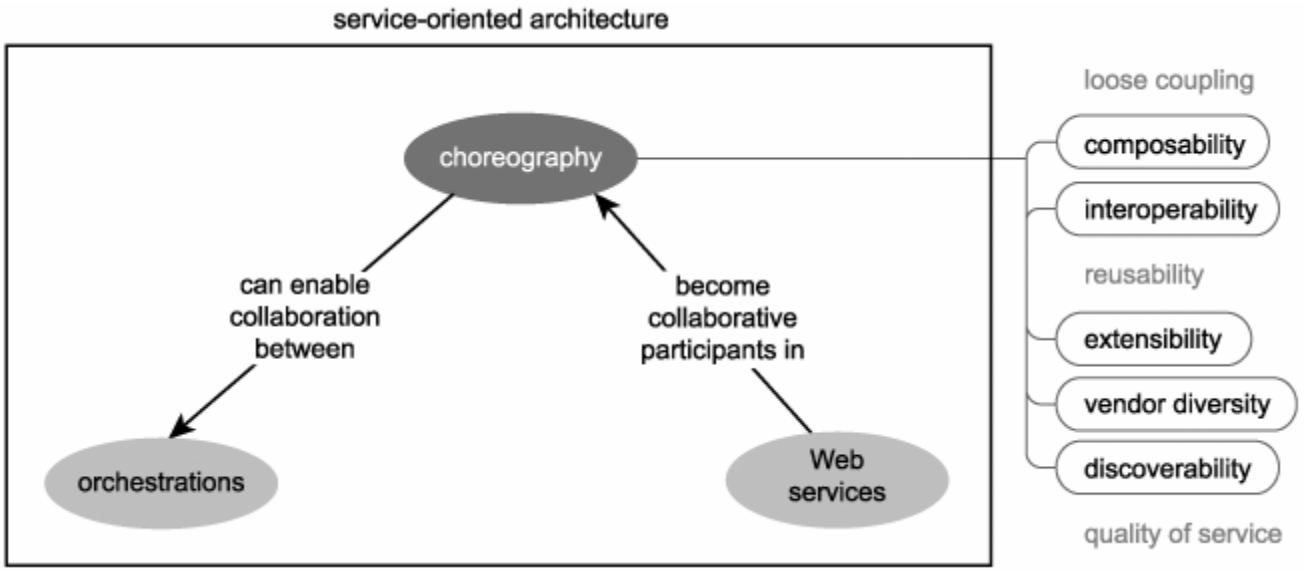
Choreography and SOA

- The fundamental concept of exposing business logic through autonomous services can be applied to just about any implementation scope.
- Two services within a single organization, each exposing a simple function, can interact via a basic MEP to complete a simple task.
- Two services belonging to different organizations, each exposing functionality from entire enterprise business solutions, can interact via a basic choreography to complete a more complex task.
- Both scenarios involve two services, and both scenarios support SOA implementations.

Choreography and SOA

- Choreography therefore can assist in the realization of SOA across organization boundaries.
- While it natively supports 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 relating to other parts of SOA



Case Study

- TLS owns the Sampson Steel manufacturing plant, a factory that originally produced various metal parts for automobile and airline companies.
- TLS uses this factory to build parts for its own railways but also continues to support the manufacture of custom parts for other clients.
- A relatively significant client has surfaced over the past year, requiring specific types of steel parts for its line of products.
- To determine the exact design specifications of a single part, the client needs to collaborate with the manufacturing specialists that were formerly employed by Sampson Steel and that now work for TLS.

Case Study

- To achieve the design specification of a single part, numerous factors are taken into consideration, including:
  - the complexity of the design
  - the cost of materials
  - the quantity of parts required
  - the availability of the necessary machines within the plant
  - the durability requirements of the part
  - environmental conditions to which the part may be exposed

Case Study

- As a result, many drafts of a specification go back and forth between the client and the TLS specialists.
- These documents undergo automated processing steps during each review cycle, relating to privacy, patents, chemical composition, and the processing of mathematical formulas that pertain to the actual part design.
- To facilitate this process, TLS and the client agree to bridge their respective automation environments with a choreography.

Case Study

- The participants of this choreography govern:
  - the transmission and routing of the messages containing the part specification documents
  - the automatic validation of specification data
  - the processing of privacy and security-related policies
  - the calculation of complex mathematical formulas
- The choreography achieves a cross-organization process that automates the collaboration cycle required by TLS and its client to negotiate and finalize specifications for custom manufactured steel parts.

Summary of Key Points

- A choreography is a complex activity comprised of a service composition and a series of MEPs.
- Choreographies consist of multiple participants that can assume different roles and that have different relationships.
- Choreographies are reusable, composable, and can be modularized.
- The concept of choreography extends the SOA vision to standardize cross-organization collaboration.