SERVICE ORIENTED ARCHITECTURE

To provide business value to the solutions we build

INTRODUCING SOA

- Fundamental SOA
- Common characteristics of contemporary SOA
- Common tangible benefits of using SOA

FUNDAMENTAL SOA

Fundamental SOA

- Represents a distinct approach for separating concerns
- logic required to solve a large problem can be better constructed, carried out, and managed if it is decomposed into a collection of smaller, related pieces. Each of these pieces addresses a concern or a specific part of the problem

A service-oriented analogy

How services encapsulate logic

How services relate

How services communicate

How services are designed

How services are built

Primitive SOA

A SERVICE ORIENTED ANALOGY

- "Service-oriented architecture" is a term that represents a model in which automation logic is decomposed into smaller, distinct units of logic. Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed.

- To allow outlets to interact and leverage each other's services, we want to avoid a model in which outlets form tight connections that result in constructive inter-dependencies.

- By empowering businesses to self-govern their individual services, allow them to evolve and grow relatively independent from each other.

A SERVICE ORIENTED ANALOGY

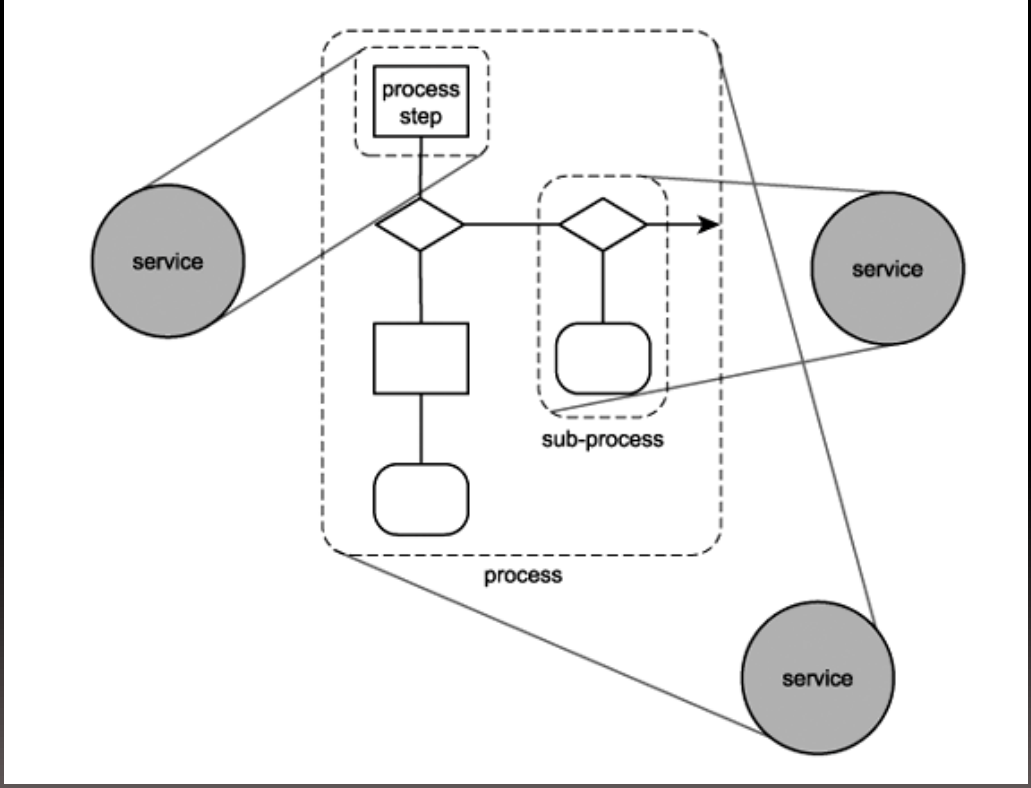
- service-oriented architecture (SOA) encourages individual units of logic to exist autonomously yet not isolated from each other.
- Units of logic are still required to conform to a set of principles that allow them to evolve independently, while still maintaining a sufficient amount of commonality and standardization.
- Within SOA, these units of logic are known as services.

HOW SERVICES ENCAPSULATE LOGIC

- To retain their independence, services encapsulate logic within a distinct context. This context can be specific to a business task, a business entity, or some other logical grouping
- service logic can encompass the logic provided by other services. In this case, one or more services are composed into a collective.
- when building an automation solution consisting of services, each service can encapsulate a task performed by an individual step or a sub-process comprised of a set of steps.
- A service can even encapsulate the entire process logic.

HOW SERVICES ENCAPSULATE LOGIC

Services can encapsulate varying amounts of logic.

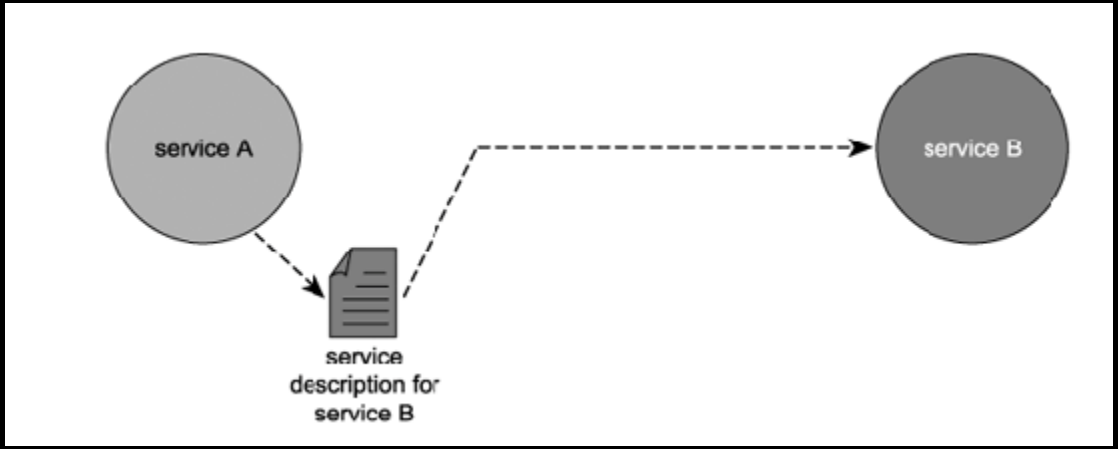


HOW SERVICES RELATE

- Within SOA, services can be used by other services or other programs
- relationship between services is based on an understanding that for services to interact, they must be aware of each other. This awareness is achieved through the use of service descriptions.
- A service description in its most basic format establishes the name of the service and the data expected and returned by the service
- services use service descriptions results in a relationship classified as loosely coupled.

HOW SERVICES RELATE

service A is aware of service B because service A is in possession of service B's service description.

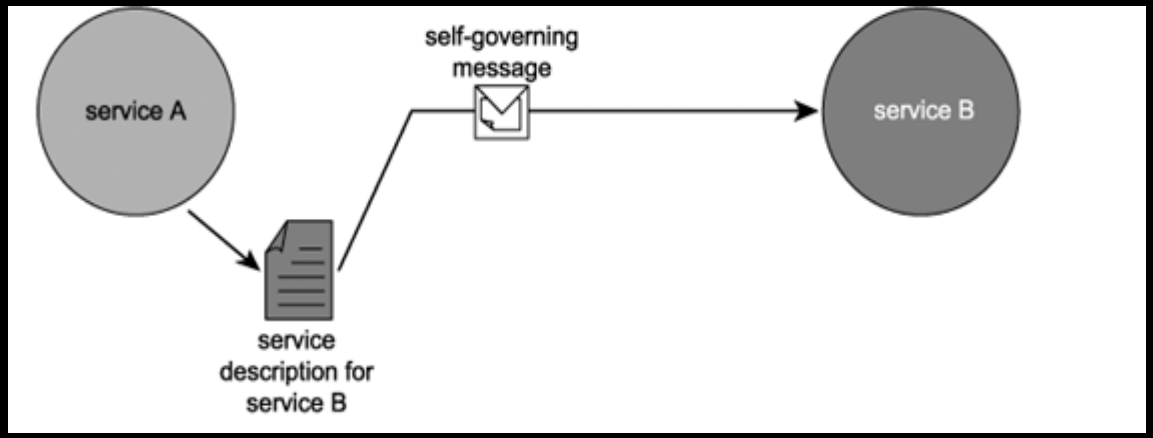


Access to service B's service description, service A has all of the information it needs to communicate with service B.

HOW SERVICES COMMUNICATE

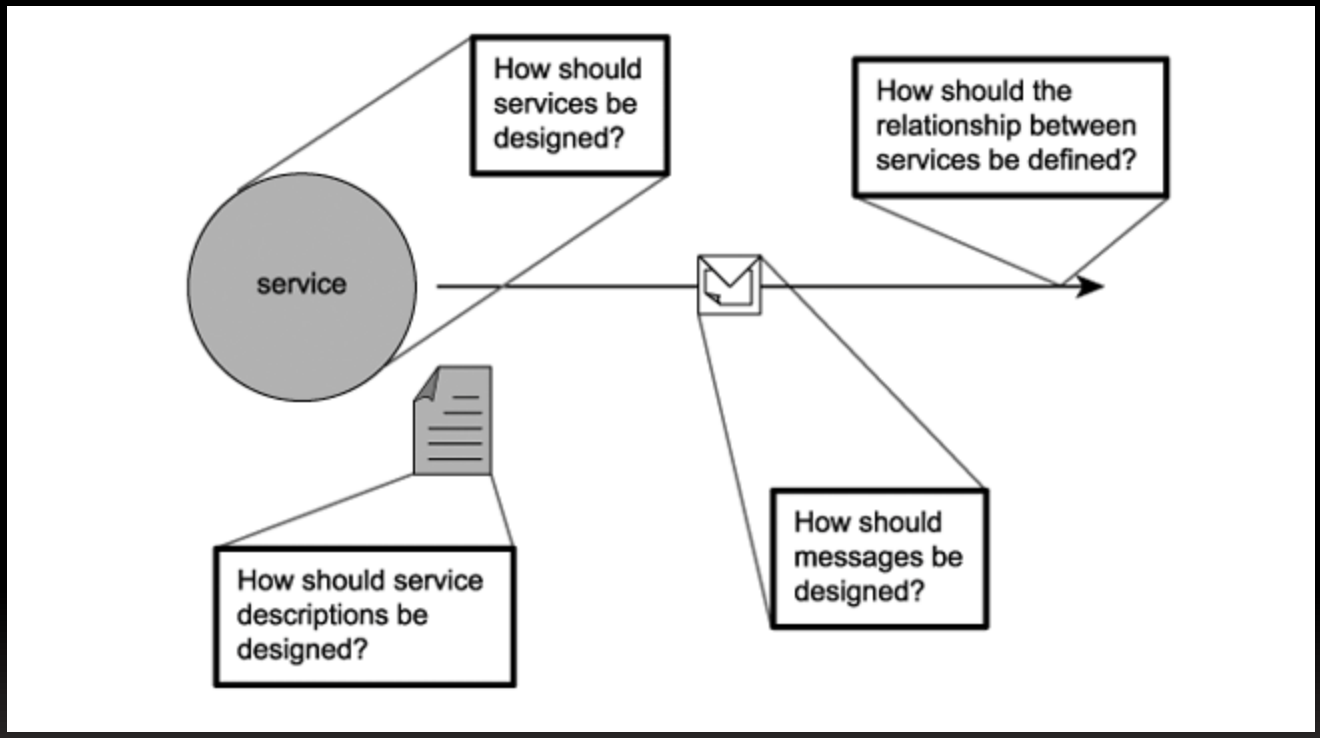
- After a service sends a message on its way, it loses control of what happens to the message thereafter
- requires messages to exist as "independent units of communication."
- messages, like services, should be autonomous
- messages can be outfitted with enough intelligence to self-govern their parts of the processing logic.
- Services that provide service descriptions and communicate via messages form a basic architecture

HOW SERVICES COMMUNICATE



A message existing as an independent unit of communication

HOW SERVICES ARE DESIGNED



Service-orientation principles address design issues

HOW SERVICES ARE DESIGNED

- After a service sends a message on its way, it loses control of what happens to the message thereafter

- When a solution is comprised of units of service-oriented processing logic, it is refered as a service-oriented solution

SOA PRINCIPLES

- Loose coupling: Services maintain a relationship that minimizes dependencies and only requires that they retain an awareness of each other.
- Service contract: Services adhere to a communications agreement, as defined collectively by one or more service descriptions and related documents.
- Autonomy: Services have control over the logic they encapsulate.
- Abstraction: Beyond what is described in the service contract, services hide logic from the outside world.

SOA PRINCIPLES

- Reusability: Logic is divided into services with the intention of promoting reuse.
- Composability: Collections of services can be coordinated and assembled to form composite services.
- Statelessness: Services minimize retaining information specific to an activity.
- Discoverability: Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

CONTEMPORARY SOA

- Major software vendors are continually conceiving new Web services specifications and building increasingly powerful XML and Web services support into current technology platforms.

- The result is an extended variation of service-oriented architecture refered as contemporary SOA.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

- Contemporary SOA is at the core of the service-oriented computing platform.
- Contemporary SOA increases quality of service.
- Contemporary SOA is fundamentally autonomous.
- Contemporary SOA is based on open standards.
- Contemporary SOA supports vendor diversity.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

- Contemporary SOA fosters intrinsic interoperability.
- Contemporary SOA promotes discovery.
- Contemporary SOA promotes federation.
- Contemporary SOA promotes architectural composability.
- Contemporary SOA fosters inherent reusability.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

- Contemporary SOA emphasizes extensibility.
- Contemporary SOA supports a service-oriented business modeling paradigm.
- Contemporary SOA implements layers of abstraction.
- Contemporary SOA promotes loose coupling throughout the enterprise.
- Contemporary SOA promotes organizational agility.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

- Contemporary SOA is a building block.
- Contemporary SOA is an evolution.
- Contemporary SOA is still maturing.
- Contemporary SOA is an achievable ideal.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is at the core of the service-oriented computing platform:

- The terms "client-server" or "n-tier," for example, can be used to classify a tool, an administration infrastructure, or an application architecture.
- Contemporary SOA represents an architecture that promotes service-orientation through the use of Web services.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA increases quality of service :

- SOA can implement enterprise-level functionality as safely and reliably as the more established distributed architectures

Common quality of service requirements (QoS):

- The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.
- Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed.
- Performance requirements to ensure that the overhead imposed by SOAP message and XML content processing does not inhibit the execution of a task.
- Transactional capabilities to protect the integrity of specific business tasks with a guarantee that should the task fail, exception logic is executed.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is fundamentally autonomous :

- Individual services exists as independent and self-contained as possible with respect to the control they maintain over their underlying logic
- It is realized through message-level autonomy where messages passed between services are sufficiently intelligence-heavy that they can control the manner in which they are processed by recipient services.
- Applications comprised of autonomous services, can themselves be viewed as composite, self-reliant services that exercise their own self-governance within service-oriented integration environments.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is based on open standards :

- The most significant characteristic of Web services is the fact that data exchange is governed by open standards.
- After a message is sent from one Web service to another it travels via a set of protocols that is globally standardized and accepted.
- The message itself is standardized, both in format and in how it represents its payload.
- The use of SOAP, WSDL, XML, and XML Schema allow for messages to be fully self-contained and support the underlying agreement that to communicate.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

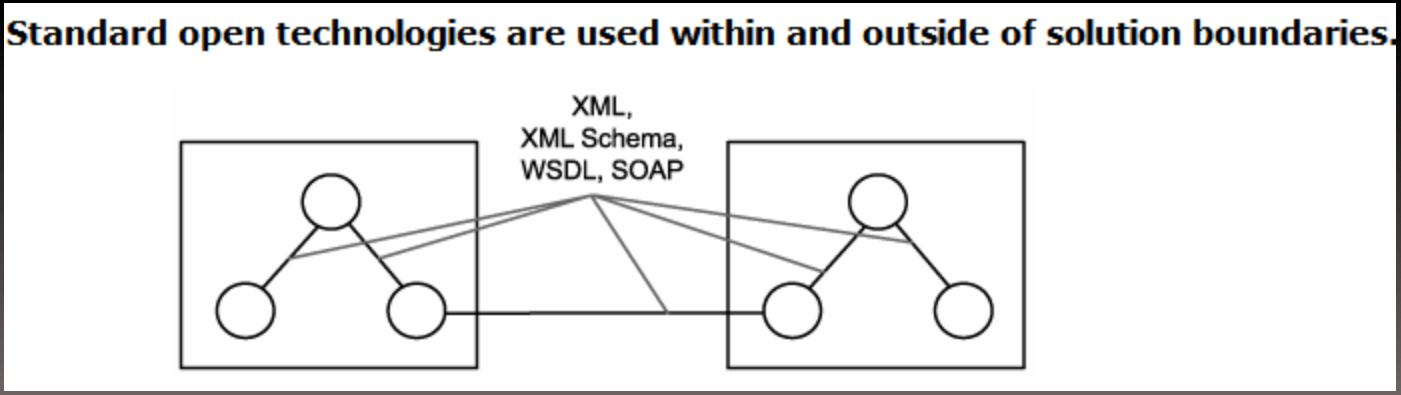
Contemporary SOA is based on open standards :

- Services require nothing more than a knowledge of each other's service descriptions.
- The use of an open, standardized messaging model eliminates the need for underlying service logic to share type systems and supports the loosely coupled paradigm.
- Contemporary SOAs fully leverage and reinforce this open, vendor-neutral communications framework

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is based on open standards :

- An SOA limits the role of proprietary technology to the implementation and hosting of the application logic encapsulated by a service.
- The opportunity for inter-service communication is therefore always an option.



COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA supports vendor diversity:

- The open communications framework has significant implications for bridging much of the heterogeneity within (and between) corporations, but it also allows organizations to choose best-of-breed environments for specific applications.
- opensup interoperability opportunities with other, service-capable applications .
- It can encapsulate legacy logic through service adapters, and leverage middleware advancements based on Web services

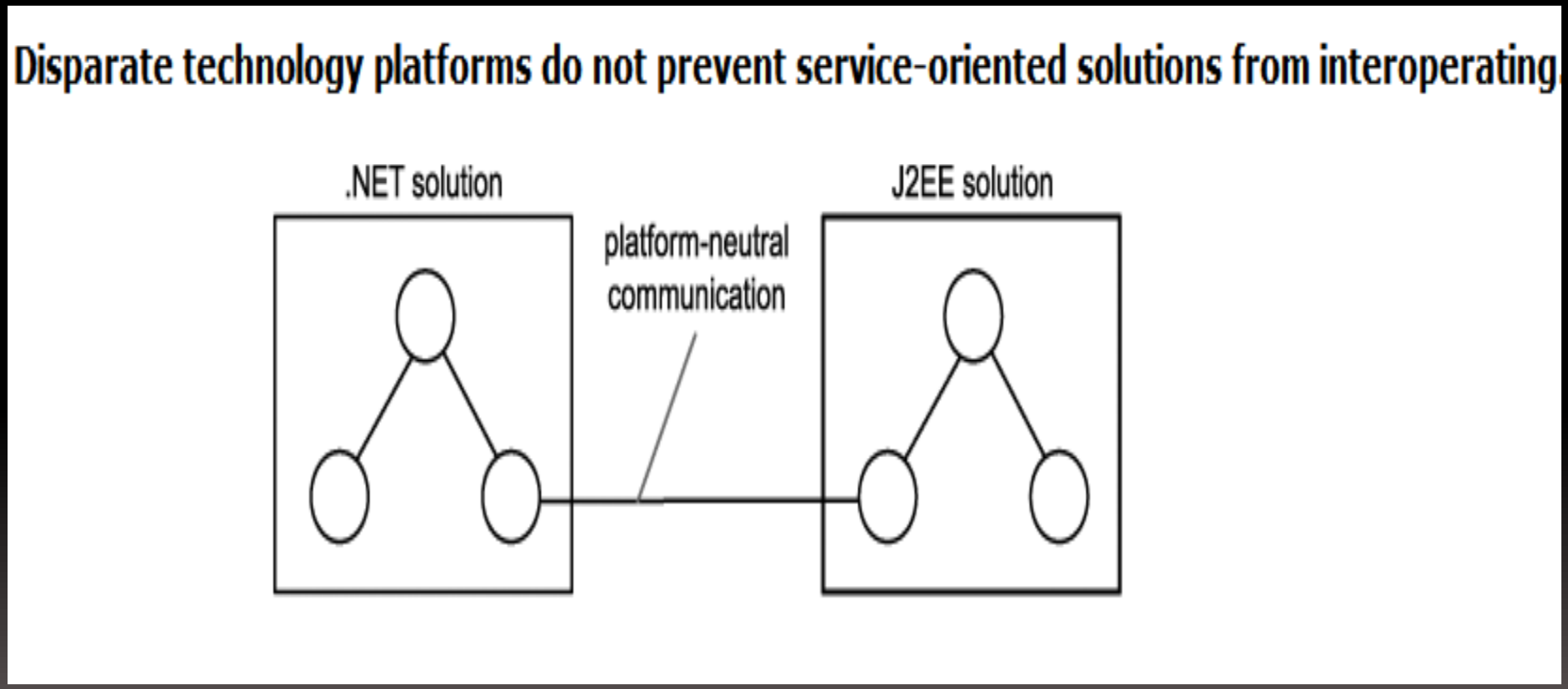
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA supports vendor diversity:

- Organizations can certainly continue building solutions with existing development tools and server products.
- opensup interoperability opportunities with other, service-capable applications .
- To explore the offerings of new vendors is made possible by the open technology provided by the Web services framework and is made more attainable through the standardization and principles introduced by SOA.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA supports vendor diversity:

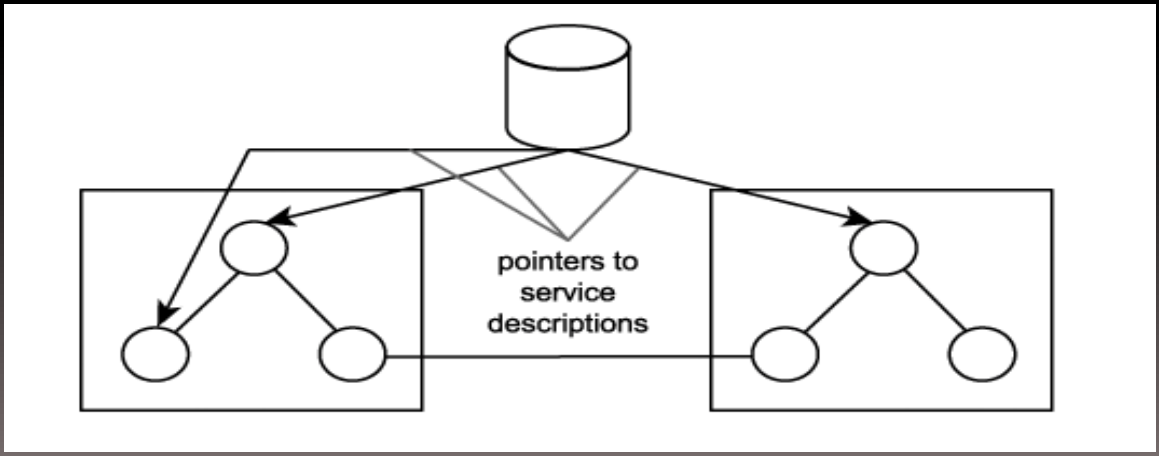


COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes discovery:

- First generation of Web services standards included UDDI, few of the early implementations actually used service registries as part of their environments.
- SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond.

Registries enable a mechanism for the discovery of services

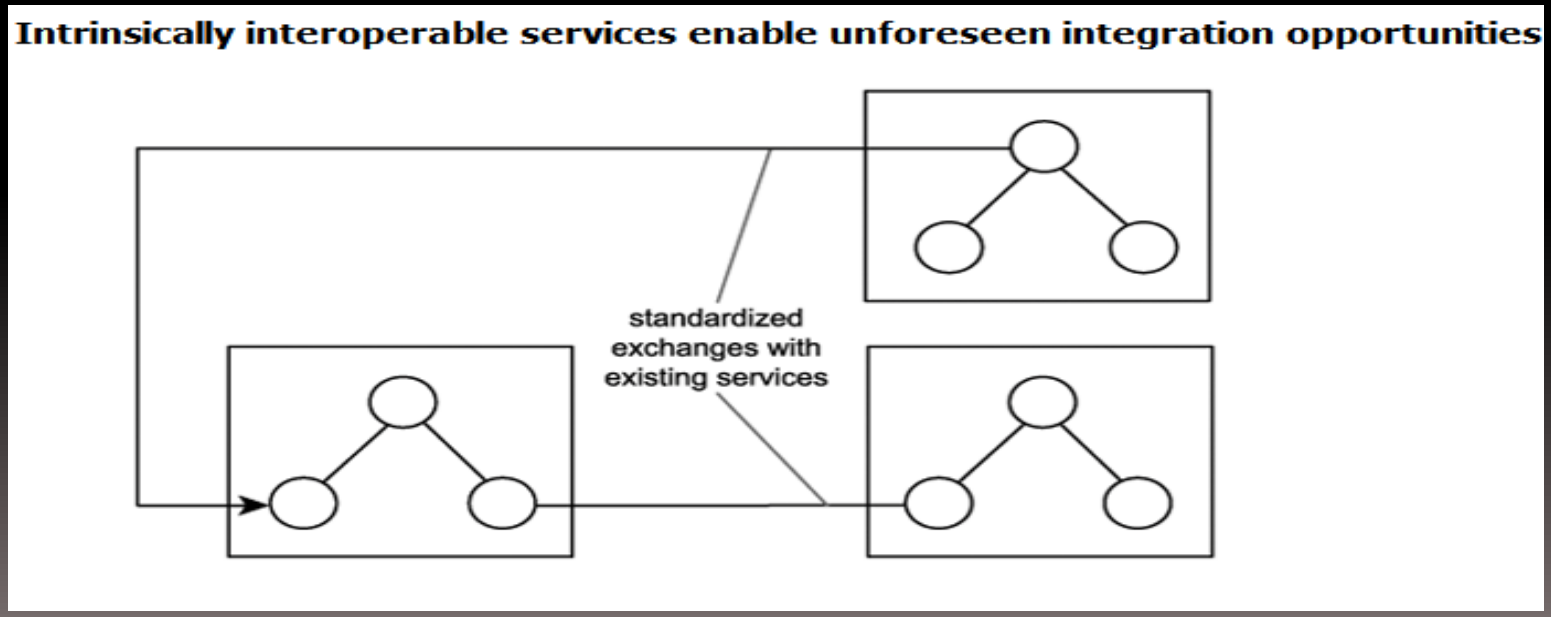


COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA fosters intrinsic interoperability:

- When building an SOA application from the ground up, services with intrinsic interoperability become potential integration endpoints

- Fostering this characteristic can significantly alleviate the cost and effort of fulfilling future cross-application integration requirements

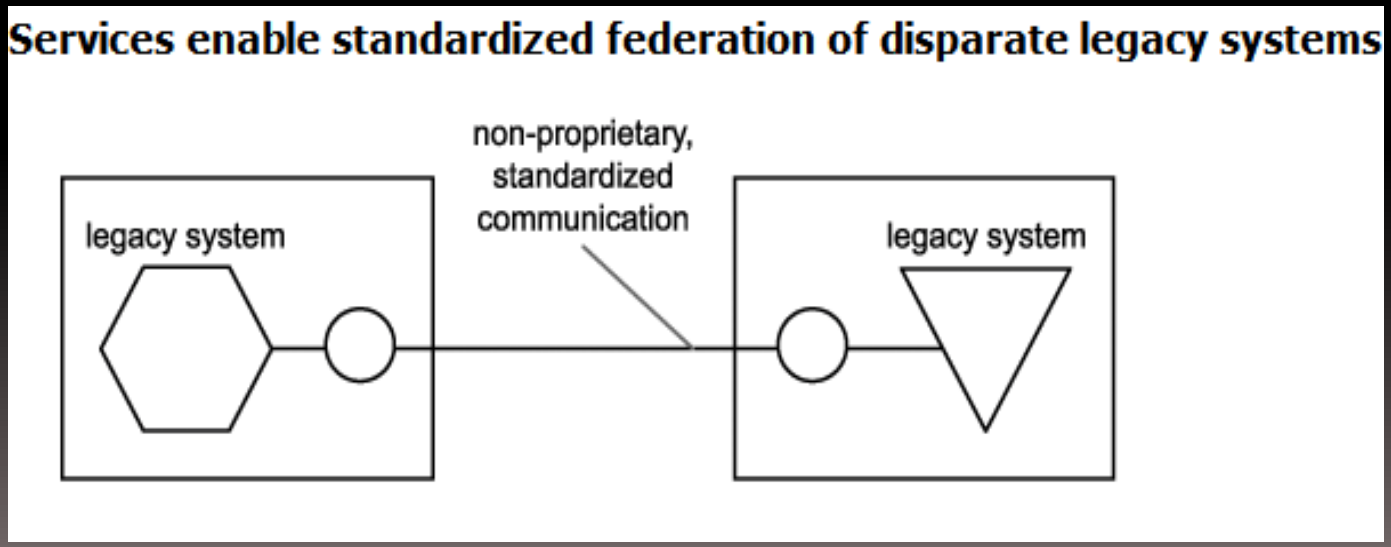


COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes federation:

- Ability to introduce unity across previously non-federated environments

- While Web services enable federation, SOA promotes this by establishing and standardizing the ability to encapsulate legacy and non-legacy application logic and by exposing it via a common, open, and standardized communications framework



COMMON CHARACTERISTICS OF CONTEMPORARY SOA

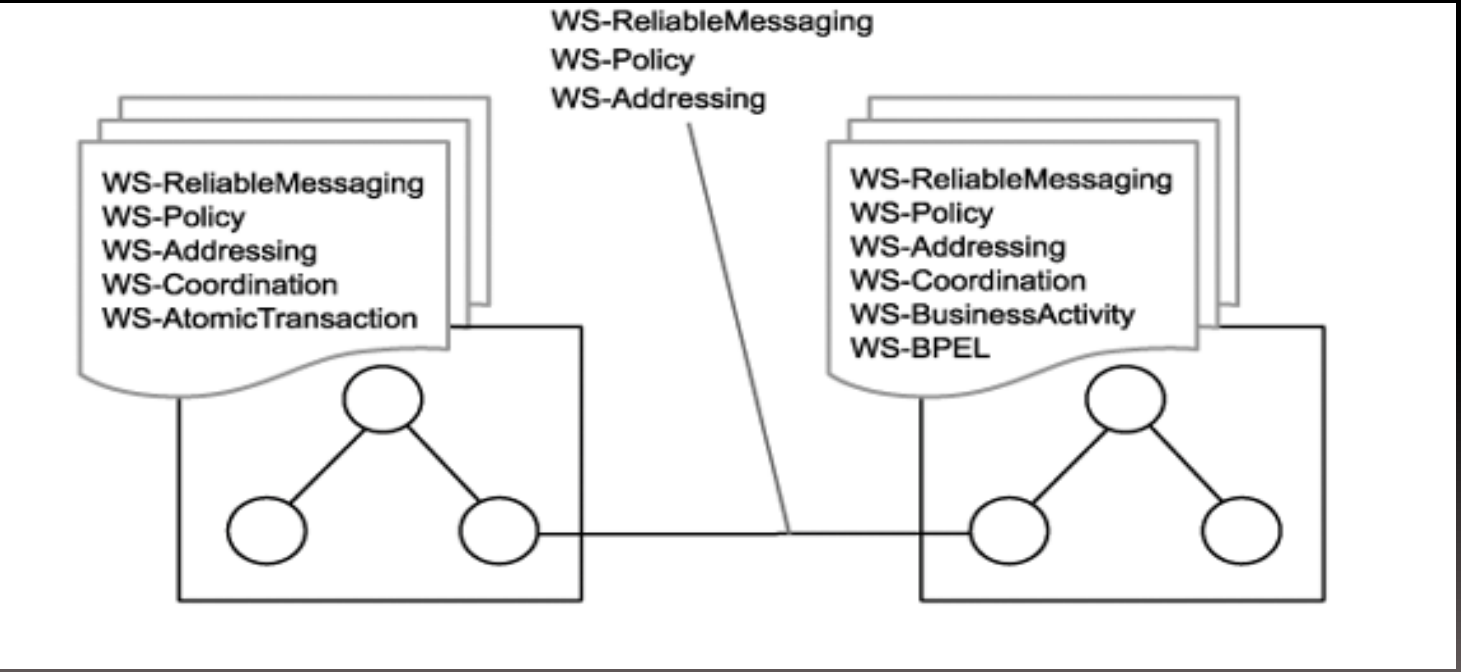
Contemporary SOA promotes architectural composability:

- Composability is a deep-rooted characteristic of SOA that can be realized on different levels
- A business process can be broken down into a series of services, each responsible for executing a portion of the process.
- Composability is represented by the second-generation Web services framework that is evolving out of the release of the numerous WS-* specifications.
- The modular nature of these specifications allows an SOA to be composed of only the functional building blocks it requires.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes architectural composability:

Different solutions can be composed of different extensions and can continue to interoperate as long as they support the common extensions required.



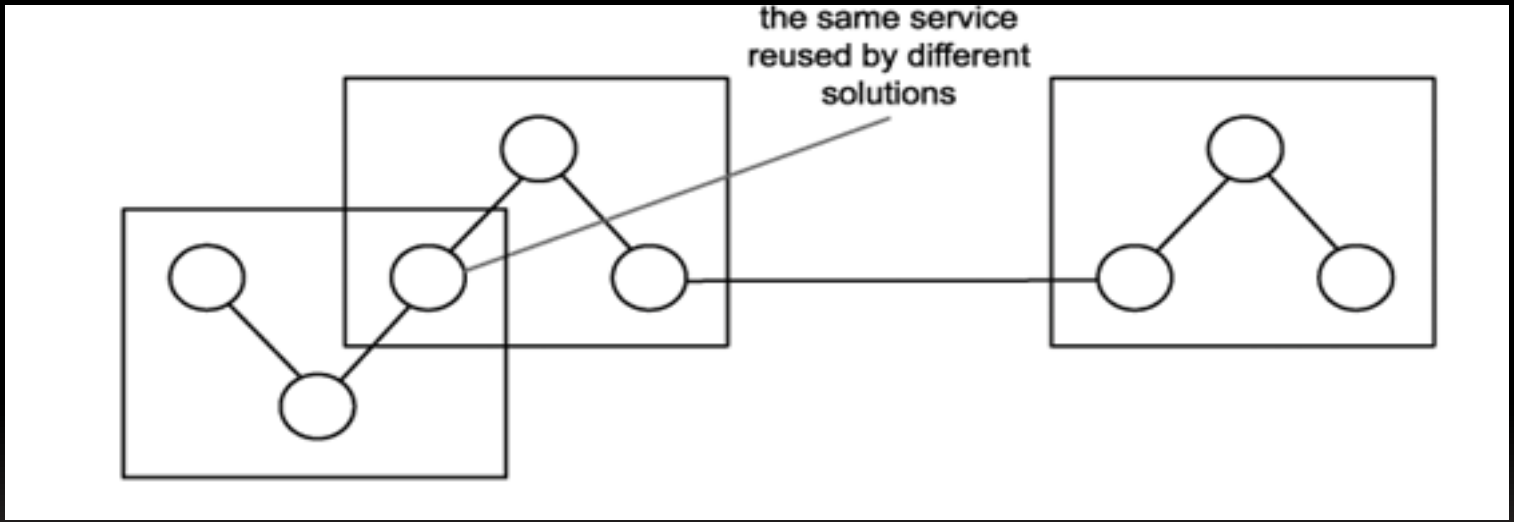
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA fosters inherent reusability:

- SOA establishes an environment that promotes reuse on many levels
- services designed according to service-orientation principles are encouraged to promote reuse, even if no immediate reuse requirements exist.
- Collections of services that form service compositions can themselves be reused by larger compositions.
- inherent reuse can be fostered when building service-oriented solutions

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA fosters inherent reusability:



Inherent reuse accommodates unforeseen reuse opportunities

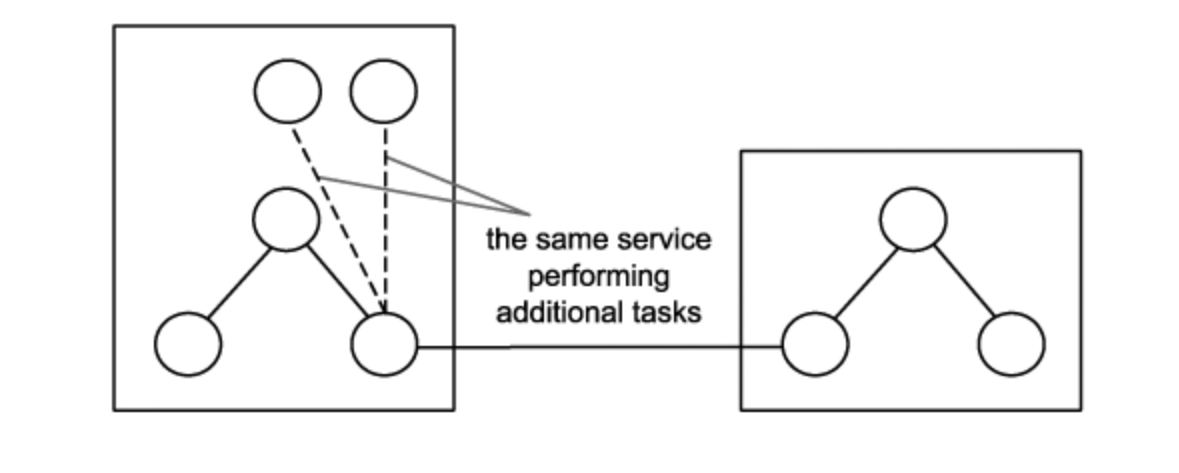
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA emphasizes extensibility:

- When service logic is properly partitioned via an appropriate level of interface granularity, the scope of functionality offered by a service can sometimes be extended without breaking the established interface
- Extending entire solutions can be accomplished by adding services or by merging with other service-oriented applications
- The loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA emphasizes extensibility:



Extensible services can expand functionality with minimal impact

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA emphasizes extensibility:

Time to redefine the characteristics we've covered:

Contemporary SOA represents an open, extensible, federated, composable architecture that promotes service-orientation and is comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.

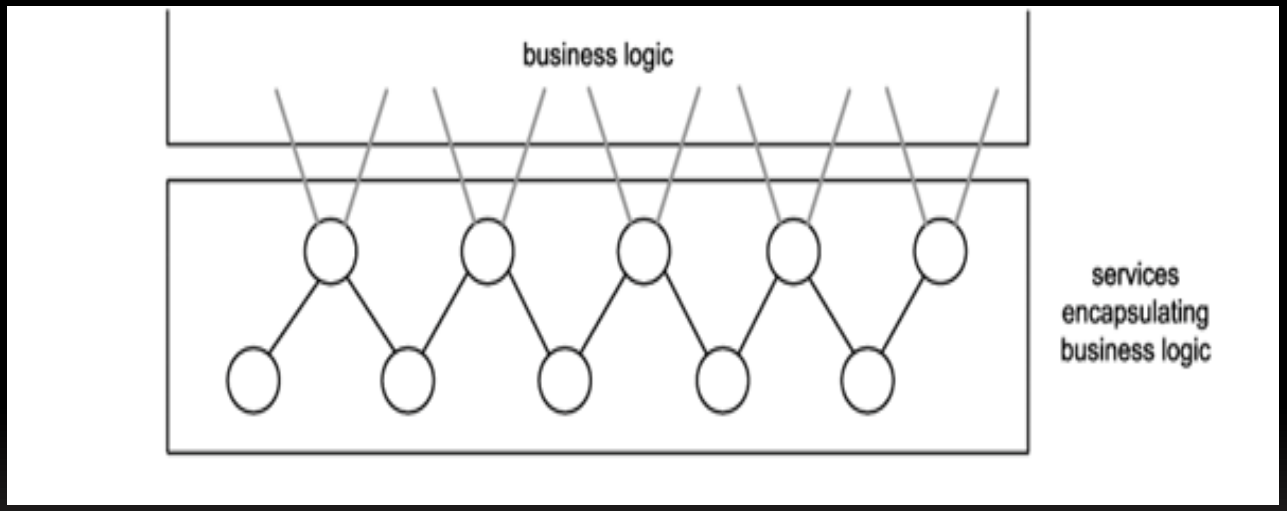
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA supports a service-oriented business modeling paradigm:

- services can be designed to express business logic
- Partitioning business logic into services that can then be composed has significant implications as to how business processes can be modeled
- Analysts can leverage these features by incorporating an extent of service-orientation into business processes for implementation through SOAs.
- BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA supports a service-oriented business modeling paradigm:



A collection (layer) of services encapsulating business process logic

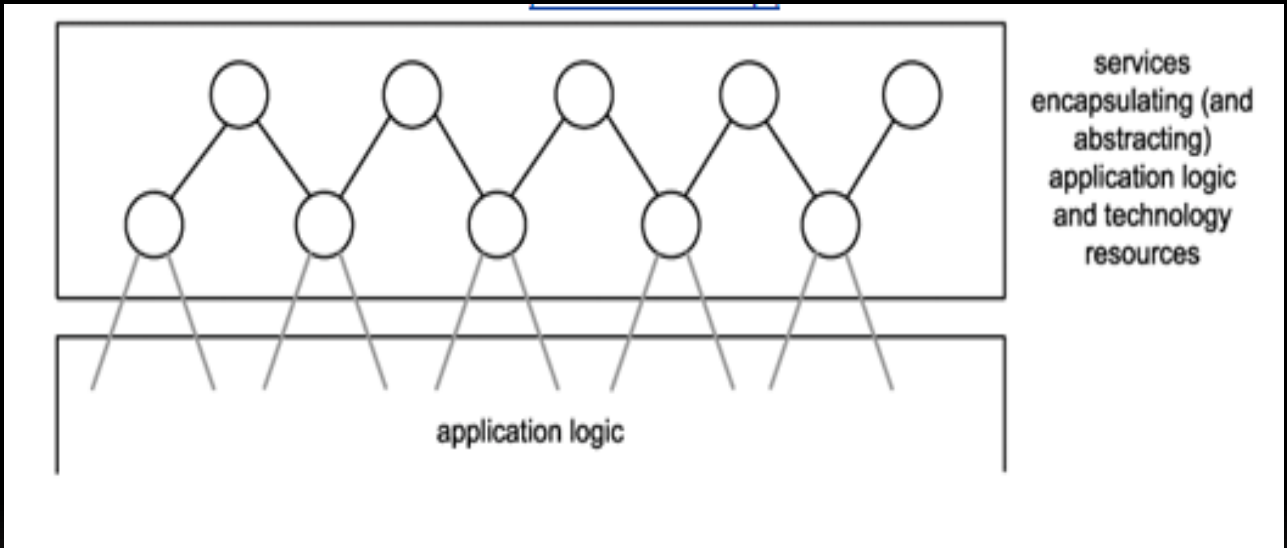
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA implements layers of abstraction:

- Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.
- By establishing a layer of endpoints that represent entire solutions and technology platforms, all of the proprietary details associated with these environments disappear
- The only remaining concern is the functionality offered via the service interfaces.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA implements layers of abstraction:



Application logic created with proprietary technology can be abstracted through a dedicated service layer

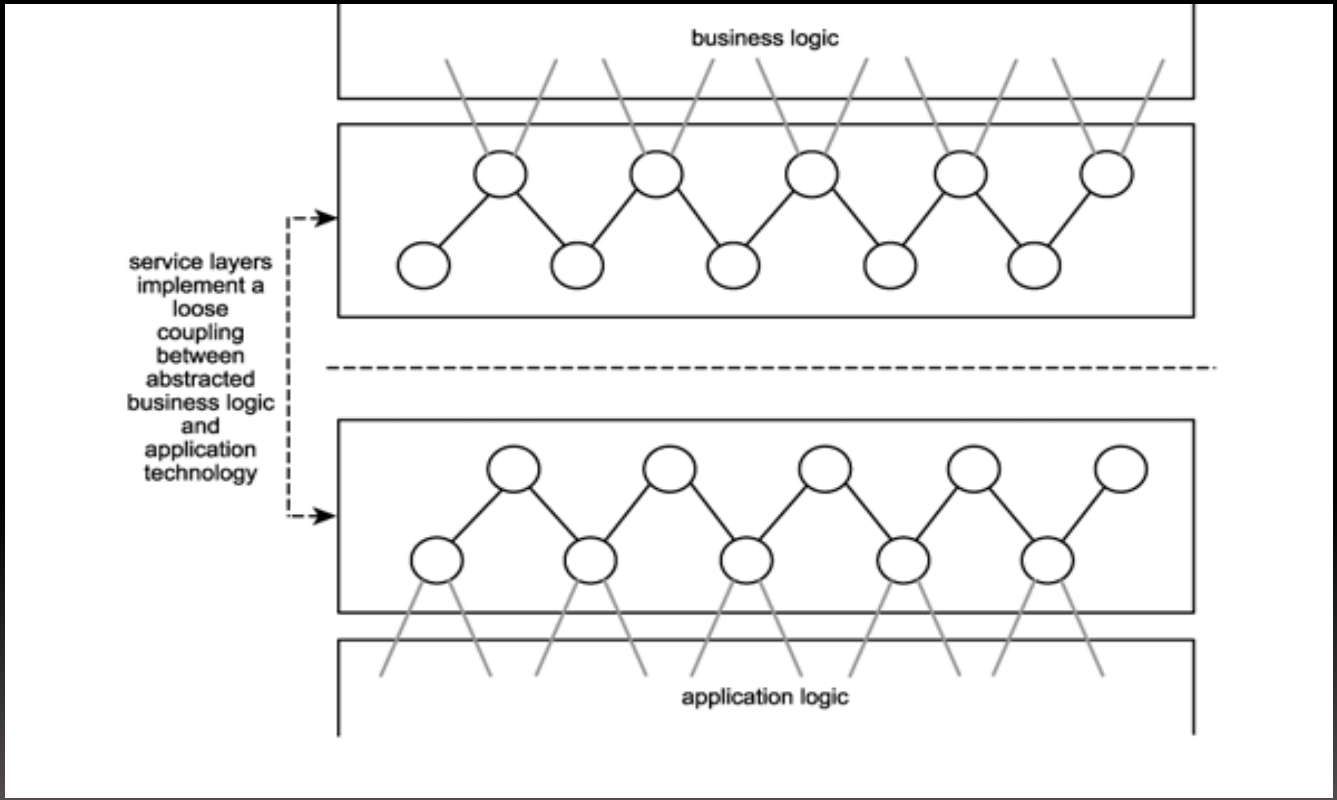
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes loose coupling throughout the enterprise :

- core benefit to building a technical architecture with loosely coupled services is the resulting independence of service logic.
- By implementing standardized service abstraction layers, a loosely coupled relationship also can be achieved between the business and application technology domains of an enterprise
- Each end only requires an awareness of the other, therefore allowing each domain to evolve more independently.
- The result is an environment that can better accommodate business and technology-related change- a quality known as organizational agility

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes loose coupling throughout the enterprise:



Through the implementation of service layers that abstract business and application logic, the loose coupling paradigm can be applied to the enterprise as a whole

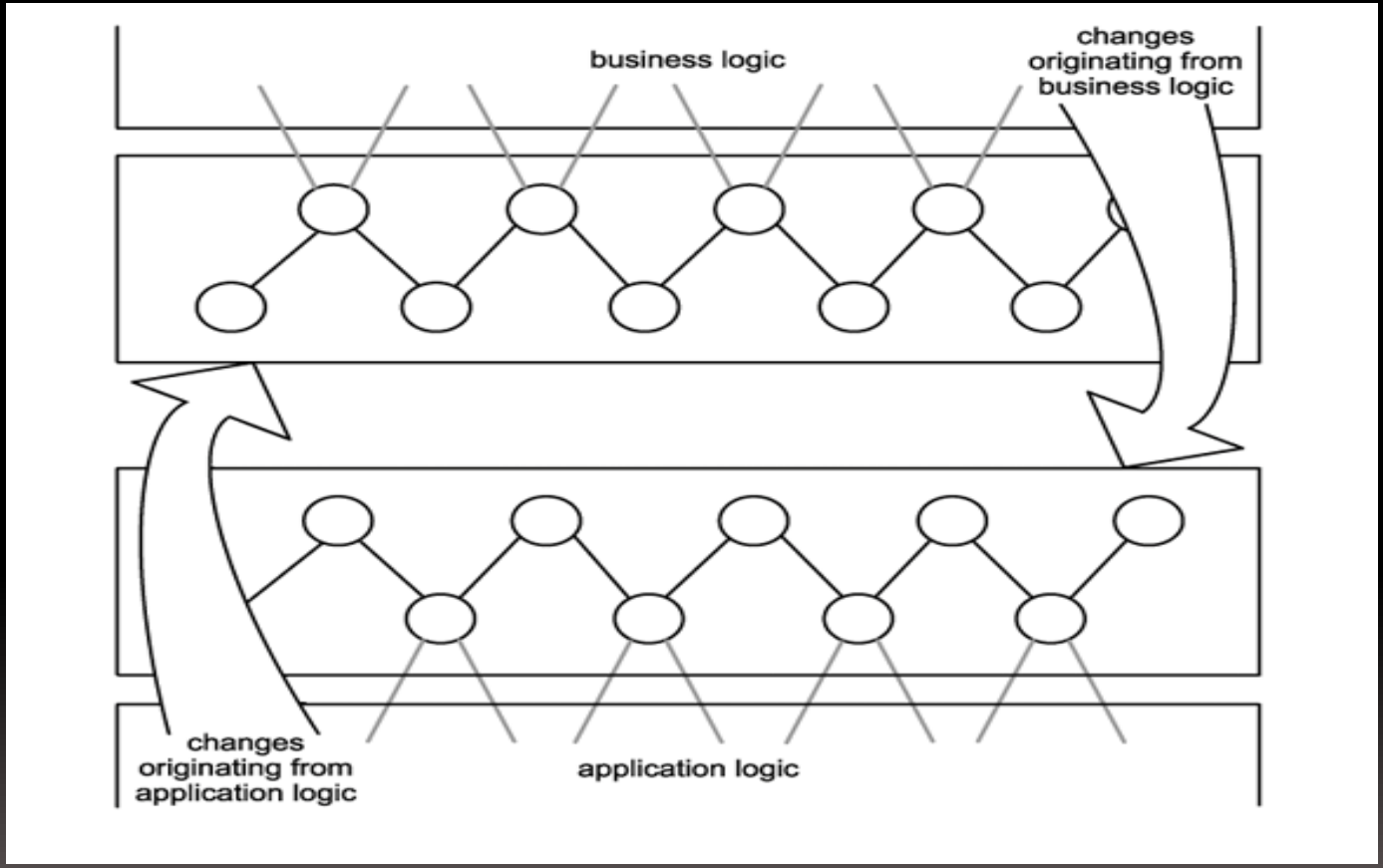
COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes organizational agility:

- an organization's ability to accommodate change determines the efficiency with which it can respond to unplanned events.
- Change in an organization's business logic can impact the application technology that automates it.
- Change in an organization's application technology infrastructure can impact the business logic automated by this technology.
- The more dependencies that exist between these two parts of an enterprise, the greater the extent to which change imposes disruption and expense.
- By leveraging service business representation, service abstraction, and the loose coupling between business and application logic provided through the use of service layers, SOA offers the potential to increase organizational agility

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA promotes organizational agility:



A loosely coupled relationship between business and application technology allows each end to more efficiently respond to changes in the other

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is a building block:

- Organizations standardizing on SOA known as the service-oriented enterprise (SOE), where all business processes are composed of and exist as services, both logically and physically.
- When viewed in the context of SOE, the functional boundary of an SOA represents either as a standalone unit of business automation or as a service encapsulating some or all of the business automation logic.
- An SOA consists of services within services within services, to the point that a solution based on SOA itself is one of many services within an SOE.

Broadened definition:

SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models. These changes foster service-orientation in support of a service-oriented enterprise.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is an evolution:

- SOA defines an architecture that is related to but still distinct from its predecessors.
- SOA supports and promotes reuse, as well as the componentization and distribution of application logic.
- These and other established design principles that are commonplace in traditional distributed environments are still very much a part of SOA.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is still maturing:

- Even though SOA is being positioned as the next standard application computing platform, this transition is not yet complete.
- Web services are being used to implement a great deal of application functionality, the support for a number of features necessary for enterprise-level computing is not yet fully available.
- When SOA platforms and tools reach an adequate level of maturity, the utilization of Web services can be extended to support the creation of enterprise SOA solutions, making the ideal of a service-oriented enterprise attainable.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Contemporary SOA is an achievable ideal:

- A standardized enterprise-wide adoption of SOA is a state to which many organizations would like to fast-forward.
- The reality is that the process of transitioning to this state demands an enormous amount of effort, discipline, and, depending on the size of the organization, a good amount of time.
- Every technical environment will undergo changes during such a migration, and various parts of SOA will be phased in at different stages and to varying extents.
- The majority of the contemporary SOA characteristics covered are attainable today

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Defining SOA:

- Contemporary SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.

- SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models.

- SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise.

- SOA is ideally standardized throughout an enterprise, but achieving this state requires a planned transition and the support of a still evolving technology set.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Separating concrete characteristics:

Remove the following items from list by exploring the concrete

Characteristics only.

- Contemporary SOA is at the core of the service-oriented computing platform.
- Contemporary SOA is a building block.
- Contemporary SOA is an evolution.
- Contemporary SOA is still maturing.
- Contemporary SOA is an achievable ideal.

COMMON CHARACTERISTICS OF CONTEMPORARY SOA

Separating concrete characteristics:

Concrete characteristics:

Contemporary SOA is generally:

- based on open standards
- architecturally composable
- capable of improving QoS

Contemporary SOA supports, fosters, or promotes:

- vendor diversity
- intrinsic interoperability
- discoverability
- federation
- inherent reusability
- extensibility
- service-oriented business modeling
- layers of abstraction
- enterprise-wide loose coupling
- organizational agility

COMMON TANGIBLE BENEFITS OF SOA

Improved integration (and intrinsic interoperability):

- SOA can result in the creation of solutions that consist of inherently interoperable services.
- Utilizing solutions based on interoperable services is part of service-oriented integration (SOI) and results in a service-oriented integration architecture.
- Because of the vendor-neutral communications framework established by Web services-driven SOAs, the potential is there for enterprises to implement highly standardized service descriptions and message structures.
- The net result is intrinsic interoperability, which turns a cross-application integration project into less of a custom development effort, and more of a modeling exercise.

The bottom line: The cost and effort of cross-application integration is significantly lowered when applications being integrated are SOA-compliant.

COMMON TANGIBLE BENEFITS OF SOA

Inherent reuse:

- Service-orientation promotes the design of services that are inherently reusable.
- Designing services to support reuse from the get-go opens the door to increased opportunities for leveraging existing automation logic.

The bottom line:

Building services to be inherently reusable results in a moderately increased development effort and requires the use of design standards. Subsequently leveraging reuse within services lowers the cost and effort of building service-oriented solutions.

COMMON TANGIBLE BENEFITS OF SOA

Streamlined architectures and solutions:

- The WS-* platform is based in its entirety on the principle of composability.
- This composability aspect of service-oriented architecture can lead to highly optimized automation environments, where only the technologies required actually become part of the architecture.

The bottom line:

Benefits of streamlined solutions and architectures include the potential for reduced processing overhead and reduced skill-set requirements (because technical resources require only the knowledge of a given application, service, or service extension).

COMMON TANGIBLE BENEFITS OF SOA

Leveraging the legacy investment :

- The industry-wide acceptance of the Web services technology enabling many legacy environments to participate in service-oriented integration architectures.

- This allows IT departments to work toward a state of federation, where previously isolated environments now can interoperate without requiring the development of expensive and sometimes fragile point-to-point integration channels.

The bottom line:

The cost and effort of integrating legacy and contemporary solutions is lowered. The need for legacy systems to be replaced is potentially lessened.

COMMON TANGIBLE BENEFITS OF SOA

Establishing standardized XML data representation :

- SOA is built upon and driven by XML.
- Adoption of SOA leads to the opportunity to fully leverage the XML data representation platform.
- A standardized data representation format (once fully established) can reduce the underlying complexity of all affected application environments..

The bottom line:

The cost and effort of application development is reduced after a proliferation of standardized XML data representation is achieved.

COMMON TANGIBLE BENEFITS OF SOA

Focused investment on communications infrastructure :

- Web services establish a common communications framework, SOA can centralize inter-application and intra-application communication as part of standard IT infrastructure.

- This allows organizations to evolve enterprise-wide infrastructure by investing in a single technology set responsible for communication.

The bottom line:

The cost of scaling communications infrastructure is reduced, as only one communications technology is required to support the federated part of the enterprise.

COMMON TANGIBLE BENEFITS OF SOA

"Best-of-breed" alternatives:

- Feature of service-oriented enterprise environments is the support of "best-of-breed" technology. Because SOA establishes a vendor-neutral communications framework, it frees IT departments from being chained to a single proprietary development and/or middleware platform.

The bottom line:

The potential scope of business requirement fulfillment increases, as does the quality of business automation.

COMMON TANGIBLE BENEFITS OF SOA

Organizational agility :

- A simple algorithm, a software component, a solution, a platform, a process all of these parts contain a measure of agility related to how they are constructed, positioned, and leveraged.
- How building blocks such as these can be realized and maintained within existing financial and cultural constraints ultimately determines the agility of the organization as a whole.
- When accommodating change becomes the norm in distributed solution design, qualities such as reuse and interoperability become commonplace. The predictability of these qualities within the enterprise leads to a reliable level of organizational agility.
- Building automation solutions and supporting infrastructure with the anticipation of change seems to make a great deal of sense. A standardized technical environment comprised of loosely coupled, composable, and interoperable and potentially reusable services establishes a more adaptive automation environment that empowers IT departments to more easily adjust to change.

The bottom line:

The cost and effort to respond and adapt to business or technology-related change is reduced.

AN SOA TIMELINE (FROM XML TO WEB SERVICES TO SOA)

XML: a brief history:

- HTML, the Extensible Markup Language (XML) was a W3C creation derived from the popular Standard Generalized Markup Language (SGML) that has existed since the late 60s.
- XML gained popularity during the eBusiness movement of the late 90s, where server-side scripting languages made conducting business via the Internet viable
- The XML Schema Definition Language (XSD) and the XSL Transformation Language (XSLT) were both authored using XML
- The XML data representation architecture represents the foundation layer of SOA.
- XSD schemas preserve the integrity and validity of message data, and XSLT is employed to enable communication between disparate data representations through schema mapping

AN SOA TIMELINE (FROM XML TO WEB SERVICES TO SOA)

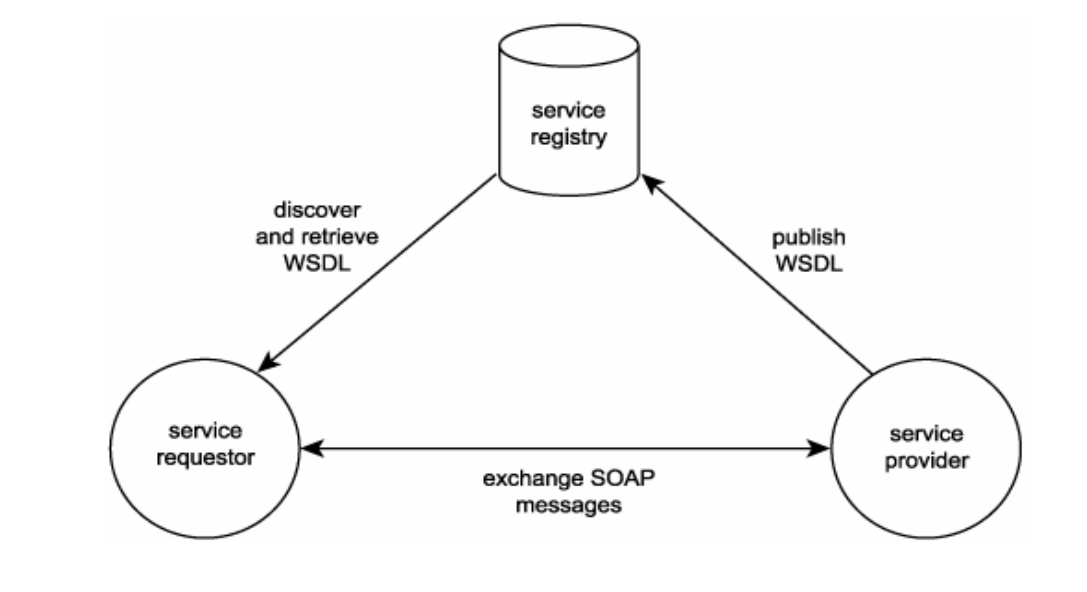
Web services: a brief history:

- In 2000, the W3C received a submission for the Simple Object Access Protocol (SOAP) specification. This specification was originally designed to unify (and in some cases replace) proprietary RPC communication
- The idea was for parameter data transmitted between components to be serialized into XML, transported, and then deserialized back into its native format.
- the idea of creating a pure, Web-based, distributed technology one that could leverage the concept of a standardized communications framework to bridge the enormous disparity that existed between and within organizations. This concept was called Web services.
- Although alternatives, such as XML-RPC, were considered, SOAP won out as the industry favorite and remains the foremost messaging standard for use with Web services.
- Completing the first generation of the Web services standards family was the UDDI specification. Originally developed by UDDI.org, it was submitted to OASIS, which continued its development in collaboration with UDDI.org.

AN SOA TIMELINE (FROM XML TO WEB SERVICES TO SOA)

SOA : a brief history:

- initial set of Web services standards, defined SOA as an architecture modeled around three basic components: t
  1. the service requestor
  2. the service provider
  3. the service registry



An early incarnation of SOA

AN SOA TIMELINE (FROM XML TO WEB SERVICES TO SOA)

SOA : a brief history:

- First-generation Web services standards fulfilled this model as follows:
  - WSDL described the service.
  - SOAP provided the messaging format used by the service and its requestor.
  - UDDI provided the standardized service registry format.

How SOA is re-shaping XML and Web services:

1. SOA requires that data representation and service modeling standards now be kept in alignment.

2. SOA relies on SOAP messaging for all inter-service communication. As a result, any place that XML needs to go, SOAP messages are generally there, taking care of transportation, interim processing and routing, and the ultimate delivery. XML documents and associated XSD schemas constantly need to be modeled with SOAP messaging in mind.

AN SOA TIMELINE (FROM XML TO WEB SERVICES TO SOA)

How SOA is re-shaping XML and Web services:

3. SOA standardizes the use of a document-style messaging. The shift from RPC-style to document-style messages imposes change on the design of service descriptions. Specifically, interface characteristics need to be expressed in more generic terms, and the overall operation granularity increases.

4. SOA relies on SOAP messaging for all inter-service communication. As a result, any place that XML needs to go, SOAP messages are generally there, taking care of transportation, interim processing and routing, and the ultimate delivery. XML documents and associated XSD schemas constantly need to be modeled with SOAP messaging in mind.

5. Due to this emphasis on document-style SOAP messages, SOA promotes a content and intelligence-heavy messaging model. This supports service statelessness and autonomy, and minimizes the frequency of message transmissions.

6. Until the advanced messaging capabilities of WS-* extensions become commonplace, many applications will need to be outfitted with custom SOAP headers to implement interim solutions to manage complex message exchanges.

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

"Standards" vs. "Specifications" vs. "Extensions" :

- A specification is a document that proposes a standard
- It is not officially an industry standard until the specification is submitted to, accepted by, and released as such by a recognized standards organization.
- **Definition:**

Standard:

An accepted industry standard. All of the first-generation Web services specifications are considered standards, as are a number of XML specifications.

Specification:

A proposed or accepted standard, described in a specification. XML standards, first-generation Web services standards, and WS-* extensions all exist within specifications.

Extension:

An extension typically represents a WS-* specification or a feature provided by a WS-* specification.

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Standards organizations that contribute to SOA :

- **The World Wide Web Consortium (W3C)**

  - Originally founded by Tim Berners-Lee in 1994, the W3C has been hugely responsible for furthering the World Wide Web as a global, semantic medium for information sharing.
  - It began with the release of HTML, one of the most popular technical languages the IT industry has ever produced.
  - When the use of the Internet broadened to include eBusiness initiatives, the W3C responded by producing key foundation standards based on XML, such as XML Schema and XSLT.
  - SOAP and WSDL standards, which have now become the signature specifications associated with Web services.
  - W3C has produced the Web Services Choreography Description Language (WS-CDL), a specification that governs standardized inter-service exchange patterns.
  - The W3C is known for its formal and rigorous approach to standards development.
  - Its process requires that specifications be subjected to numerous review and revision stages, with each new version being published to their public Web site. T
  - he thoroughness of its process comes at the cost of time.
  - Standards can take two to three years to be completed.

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Standards organizations that contribute to SOA :

Organization for the Advancement of Structured Information Standards (OASIS):

- Originally established in 1993 as the SGML Open, OASIS changed its name five years later to represent a shift in focus from SGML to XML-related standards
- OASIS assumed ownership of the prominent WS-BPEL specification and is also known for its development of ebXML (a specification that aims to establish a standardized means of B2B data interchange) and its contributions to the UDDI specification, one of the core standards associated with the first-generation Web services platform.
- Security Assertion Markup Language (SAML) and the Extensible Access Control Markup Language (XACML) provide important features in the areas of single sign-on and authorization
- the standards development processes used by OASIS are noticeably shorter

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Standards organizations that contribute to SOA :

The Web Services Interoperability Organization (WS-I):

- Established in 2002, The primary objective of the WS-I is not to create new standards, but to ensure that the ultimate goal of open interoperability is realized.
- The WS-I is best known for releasing the Basic Profile, a recommendation-based document that establishes which of the available standards should be collectively used to form the most desirable interoperability architecture.
- By formally positioning specific versions of WSDL, SOAP, UDDI, XML, and XML Schema, the Basic Profile has become an important document within the IT community. Those organizations wanting to ensure that the SOAs they develop are fully interoperable with others can guarantee a high-level of acceptance with compliance to the Basic Profile.
- More recently, the WS-I developed the Basic Security Profile. Essentially the same concept as the Basic Profile, this document establishes the most important collection of Web services and XML security technologies.

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Standards organizations that contribute to SOA :

A Comparison of Standards Organizations

	W3C	OASIS	WS-I
Established	1994	1993 as the SGML Open, 1998 as OASIS	2002
Approximate membership	400	600	200
Overall goal (as it relates to SOA)	To further the evolution of the Web, by providing fundamental standards that improve online business and information sharing.	To promote online trade and commerce via specialized Web services standards.	To foster standardized interoperability using Web services standards.
Prominent deliverables (related to SOA)	XML, XML Schema, XQuery, XML Encryption, XML Signature, XPath, XSLT, WSDL, SOAP, WS-CDL, WS-Addressing, Web Services Architecture	UDDI, ebXML, SAML, XACML, WS-BPEL, WS-Security	Basic Profile, Basic Security Profile

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Major vendors that contribute to SOA:

- Though standards organizations have their own culture and philosophies around how standards should be developed, they are all heavily influenced by the commercial market.
- Even though these organizations exist as independent entities, their membership includes all major software vendors.
- same vendors supply a significant portion of the contributors that actually end up developing the standards.
- Some of the companies that have participated in the standards development processes include Microsoft, IBM, BEA Systems, Sun Microsystems, Oracle, Tibco, Hewlett-Packard, Canon, Commerce One, Fujitsu, Software AG, Nortel, Verisign, and WebMethods.

THE CONTINUING EVOLUTION OF SOA (STANDARDS ORGANIZATIONS AND CONTRIBUTING VENDORS)

Why standards are being developed in support of SOA:

- No one person or organization owns or controls SOA.
- Having evolved from proprietary platforms into an architecture that promotes and supports open standards and vendor-neutral protocols, SOA will likely remain an important architecture for as long as the major software vendors choose to support it.
- the benefits of SOA can only be realized as long as it continues to receive the global acceptance it does now.
- Regardless, SOA today is foremost on every major software organization's priority list.
- Non-compliance with SOA is not even being considered, as it would mean cutting yourself out of an ever-growing market.

THE ROOTS OF SOA (COMPARING SOA TO PAST ARCHITECTURES)

Application architecture:

- Application architecture is to an application development team like what a blueprint is to a team of construction workers.
- Different organizations document different levels of application architecture.
- Some keep it high-level, providing abstract physical and logical representations of the technical blueprint. Others include more detail, such as common data models, communication flow diagrams, application-wide security requirements, and aspects of infrastructure.
- It is not uncommon for an organization to have several application architectures.
- A single architecture document typically represents a distinct solution environment.
- an organization that houses both .NET and J2EE solutions would very likely have separate application architecture specifications for each.
- A key part of any application-level architecture is that it reflects immediate solution requirements, as well as long-term, strategic IT goals.

THE ROOTS OF SOA (COMPARING SOA TO PAST ARCHITECTURES)

Enterprise architecture:

- In larger IT environments, the need to control and direct IT infrastructure is critical.
- When numerous, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous (tedious).
- a master specification need to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure.
- an enterprise architecture specification is to an organization what an urban plan is to a city.
- changes to enterprise architectures directly affect application architectures, which is why architecture specifications often are maintained by the same group of individuals.
- enterprise architectures often contain a long-term vision of how the organization plans to evolve its technology and environments.
- document also may define the technology and policies behind enterprise-wide security measures.
- However, these often are isolated into a separate security architecture specification.

THE ROOTS OF SOA (COMPARING SOA TO PAST ARCHITECTURES)

Service-oriented architecture:

- service-oriented architecture spans both enterprise and application architecture domains.
- The benefit potential offered by SOA can only be truly realized when applied across multiple solution environments.
- This is where the investment in building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged.
- This does not mean that the entire enterprise must become service-oriented.
- SOA belongs in those areas that have the most to gain from the features and characteristics it introduces.
- the term "SOA" does not necessarily imply a particular architectural scope.
- An SOA can refer to an application architecture or the approach used to standardize technical architecture across the enterprise.
- Because of the composable nature of SOA (meaning that individual application-level architectures can be comprised of different extensions and technologies), it is absolutely possible for an organization to have more than one SOA.

SOA VS. CLIENT-SERVER ARCHITECTURE

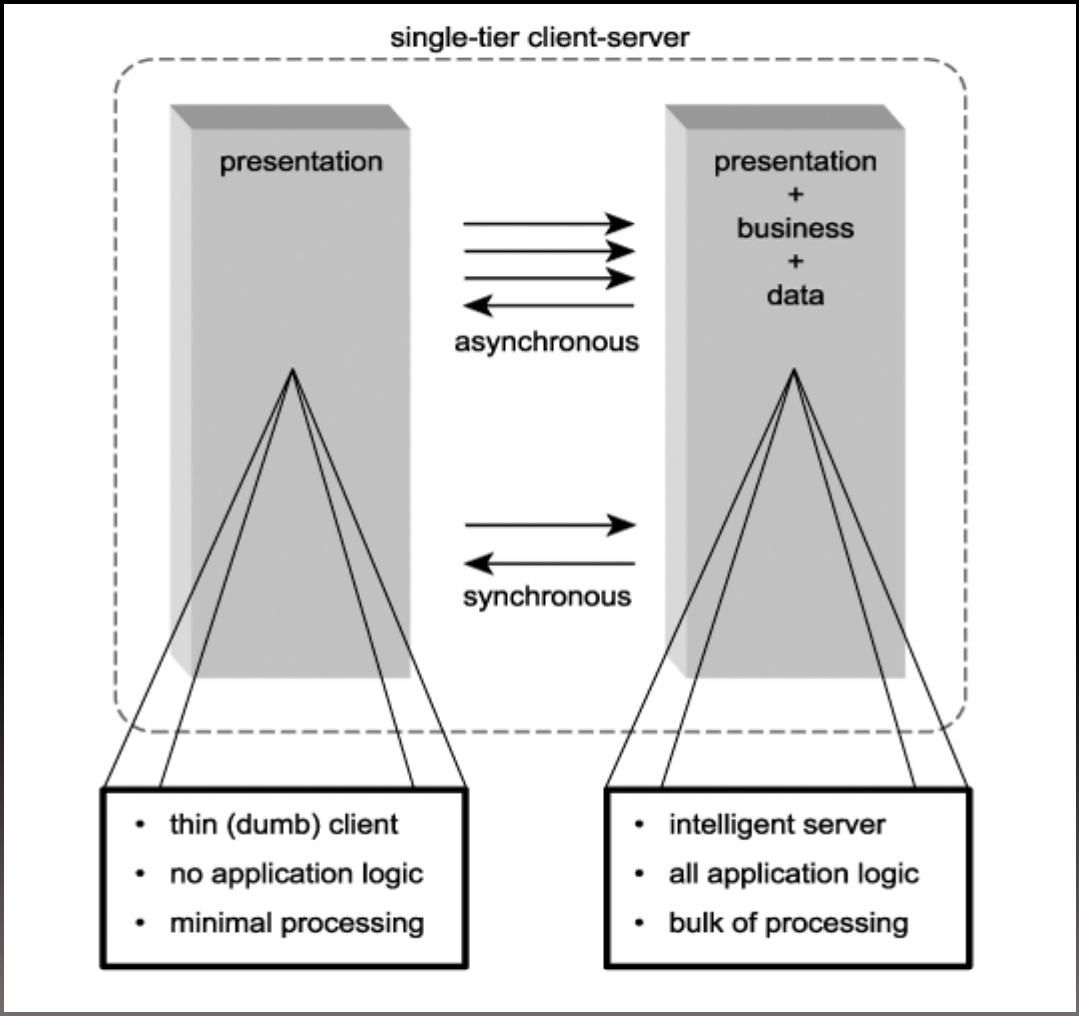
Client-server architecture: a brief history:

- bulky mainframe back-ends served thin clients, are considered an implementation of the single-tier client-server architecture .
- Mainframe systems natively supported both synchronous and asynchronous communication. The latter approach was used primarily to allow the server to continuously receive characters from the terminal in response to individual key-strokes. Only upon certain conditions would the server actually respond.
- new approach introduced the concept of delegating logic and processing duties onto individual workstations, resulting in the birth of the fat client.
- support by the innovation of the graphical user-interface (GUI), two-tier client-server was considered a huge step forward and went on to dominate the IT world for years during the early 90s.
- The common configuration of this architecture consisted of multiple fat clients, each with its own connection to a database on a central server. Client-side software performed the bulk of the processing, including all presentation-related and most data access logic

SOA VS. CLIENT-SERVER ARCHITECTURE

Client-server architecture: a brief history:

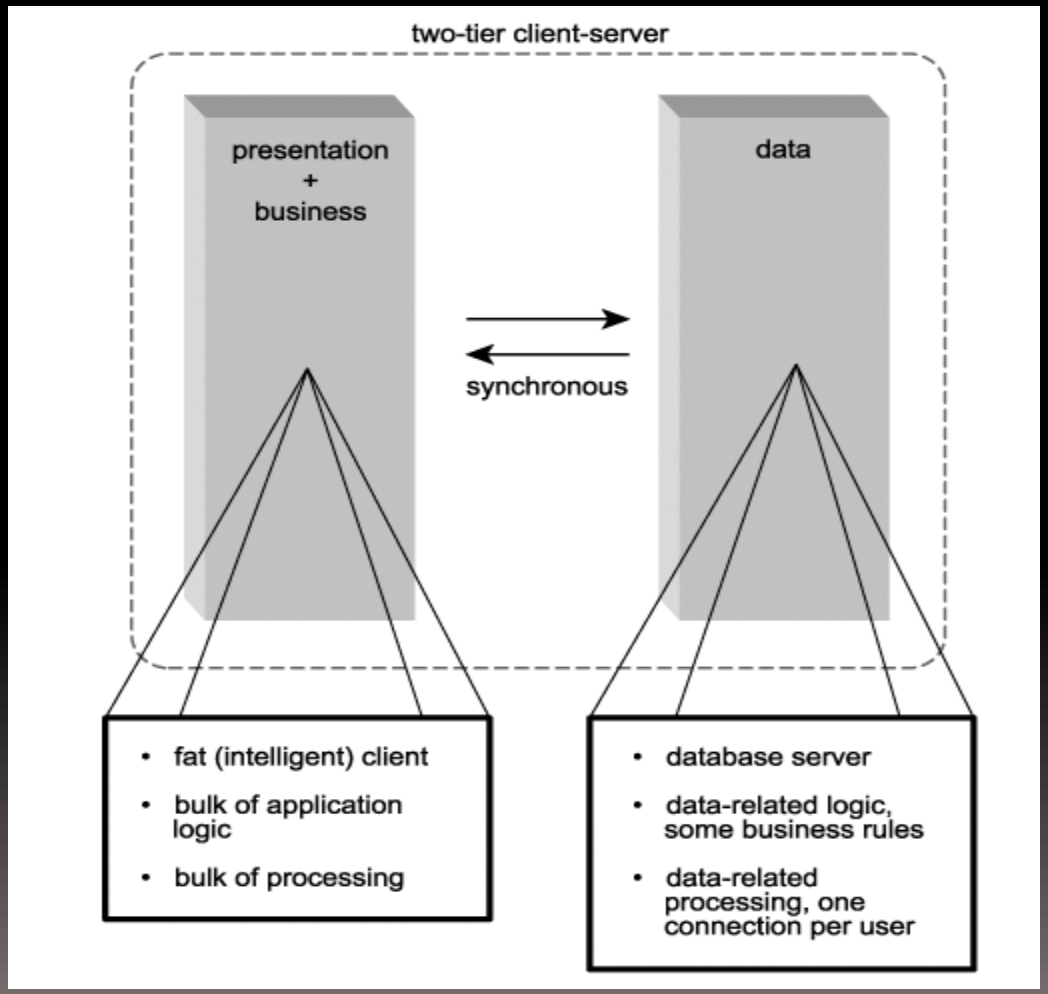
A typical single-tier client-server architecture



SOA VS. CLIENT-SERVER ARCHITECTURE

Client-server architecture: a brief history:

A typical two-tier client-server architecture



SOA VS. CLIENT-SERVER ARCHITECTURE

Application logic

- Client-server environments place the majority of application logic into the client software. This results in a monolithic executable that controls the user experience, as well as the back-end resources.
- One exception is the distribution of business rules.
- A popular trend was to embed and maintain business rules relating to data within stored procedures and triggers on the database.
- This abstracted a set of business logic from the client and simplified data access programming.
- the client ran the show.
- The presentation layer within contemporary service-oriented solutions can vary.
- Any piece of software capable of exchanging SOAP messages according to required service contracts can be classified as a service requestor.
- While it is commonly expected for requestors to be services as well, presentation layer designs are completely open and specific to a solution's requirements.

SOA VS. CLIENT-SERVER ARCHITECTURE

Application logic

- service-oriented design principles often dictates the partitioning of processing logic into autonomous units.
- This facilitates specific design qualities, such as service statelessness and interoperability, as well as future composability and reusability.
- Additionally, it is more common within an SOA for these units of processing logic to be solution-agnostic.
- This supports the ultimate goal of promoting reuse and loose coupling across application boundaries.

SOA VS. CLIENT-SERVER ARCHITECTURE

Application processing

- most client-server application logic resides in the client component, the client workstation is responsible for the bulk of the processing.
- The 80/20 ratio often is used as a rule of thumb, with the database server typically performing twenty percent of the work.
- it is the database that frequently becomes the performance bottleneck in these environments.
- A two-tier client-server solution with a large user-base generally requires that each client establish its own database connection.
- Communication is predictably synchronous, and these connections are often persistent (meaning that they are generated upon user login and kept active until the user exits the application).

SOA VS. CLIENT-SERVER ARCHITECTURE

Application processing

- Proprietary database connections are expensive, and the resource demands sometimes overwhelm database servers, imposing processing latency on all users.
- the clients are assigned the majority of processing responsibilities, they too often demand significant resources.
- Client-side executables are fully stateful and consume a steady chunk of PC memory.
- User workstations are required to run client programs exclusively so that all available resources can be offered to the application.
- Processing in SOA is highly distributed. Each service has an explicit functional boundary and related resource requirements.
- In modeling a technical service-oriented architecture, you have many choices as to how you can position and deploy services.
- Enterprise solutions consist of multiple servers, each hosting sets of Web services and supporting middleware.

SOA VS. CLIENT-SERVER ARCHITECTURE

Application processing

- Services can be distributed as required, and performance demands are one of several factors in determining the physical deployment configuration.
- Communication between service and requestor can be synchronous or asynchronous.
- This flexibility allows processing to be further streamlined, especially when asynchronous message patterns are utilized.
- placing a large amount of intelligence into the messages, options for achieving message-level context management are provided.
- This promotes the stateless and autonomous nature of services and further alleviates processing by reducing the need for runtime caching of state information.

SOA VS. CLIENT-SERVER ARCHITECTURE

Technology

- The emergence of client-server applications promoted the use of 4GL programming languages, such as Visual Basic and PowerBuilder.
- These development environments took better advantage of the Windows operating system by providing the ability to create aesthetically rich and more interactive user-interfaces.
- Regardless, traditional 3GL languages, such as C++, were also still used, especially for solutions that had more rigid performance requirements.
- On the back-end, major database vendors, such as Oracle, Informix, IBM, Sybase, and Microsoft, provided robust RDBMSs that could manage multiple connections, while providing flexible data storage and data management features.
- The technology set used by SOA actually has not changed as much as it has expanded.
- Newer versions of older programming languages, such as Visual Basic, still can be used to create Web services, and the use of relational databases still is commonplace.
- In addition to the standard set of Web technologies (HTML, CSS, HTTP, etc.) contemporary SOA brings with it the absolute requirement that an XML data representation architecture be established, along with a SOAP messaging framework, and a service architecture comprised of the ever-expanding Web services platform.

SOA VS. CLIENT-SERVER ARCHITECTURE

Security

- Besides the storage and management of data and the business rules embedded in stored procedures and triggers, the one other part of client-server architecture that frequently is centralized at the server level is security.
- Databases are sufficiently sophisticated to manage user accounts and groups and to assign these to individual parts of the physical data model.
- Security also can be controlled within the client executable, especially when it relates to specific business rules that dictate the execution of application logic (such as limiting access to a part of a user-interface to select users).
- operating system-level security can be incorporated to achieve a single sign-on, where application clearance is derived from the user's operating system login account information.
- Though one could boast about the advantages of SOA, most architects envy the simplicity of client-server security.
- Corporate data is protected via a single point of authentication, establishing a single connection between client and server.
- In the distributed world of SOA, this is not possible.
- Security becomes a significant complexity directly relational to the degree of security measures required.
- Multiple technologies are typically involved, many of which comprise the WS-Security framework

SOA VS. CLIENT-SERVER ARCHITECTURE

Administration

- One of the main reasons the client-server era ended was the increasingly large maintenance costs associated with the distribution and maintenance of application logic across user workstations.
- Because each client housed the application code, each update to the application required a redistribution of the client software to all workstations.
- In larger environments, this resulted in a highly burdensome administration process.
- Maintenance issues spanned both client and server ends.
- Client workstations were subject to environment-specific problems because different workstations could have different software programs installed or may have been purchased from different hardware vendors. Further, there were increased server-side demands on databases, especially when a client-server application expanded to a larger user base.
- Because service-oriented solutions can have a variety of requestors, they are not necessarily immune to client-side maintenance challenges.
- While their distributed back-end does accommodate scalability for application and database servers, new administration demands can be introduced.
- For example, once SOAs evolve to a state where services are reused and become part of multiple service compositions, the management of server resources and service interfaces can require powerful administration tools, including the use of a private registry.

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

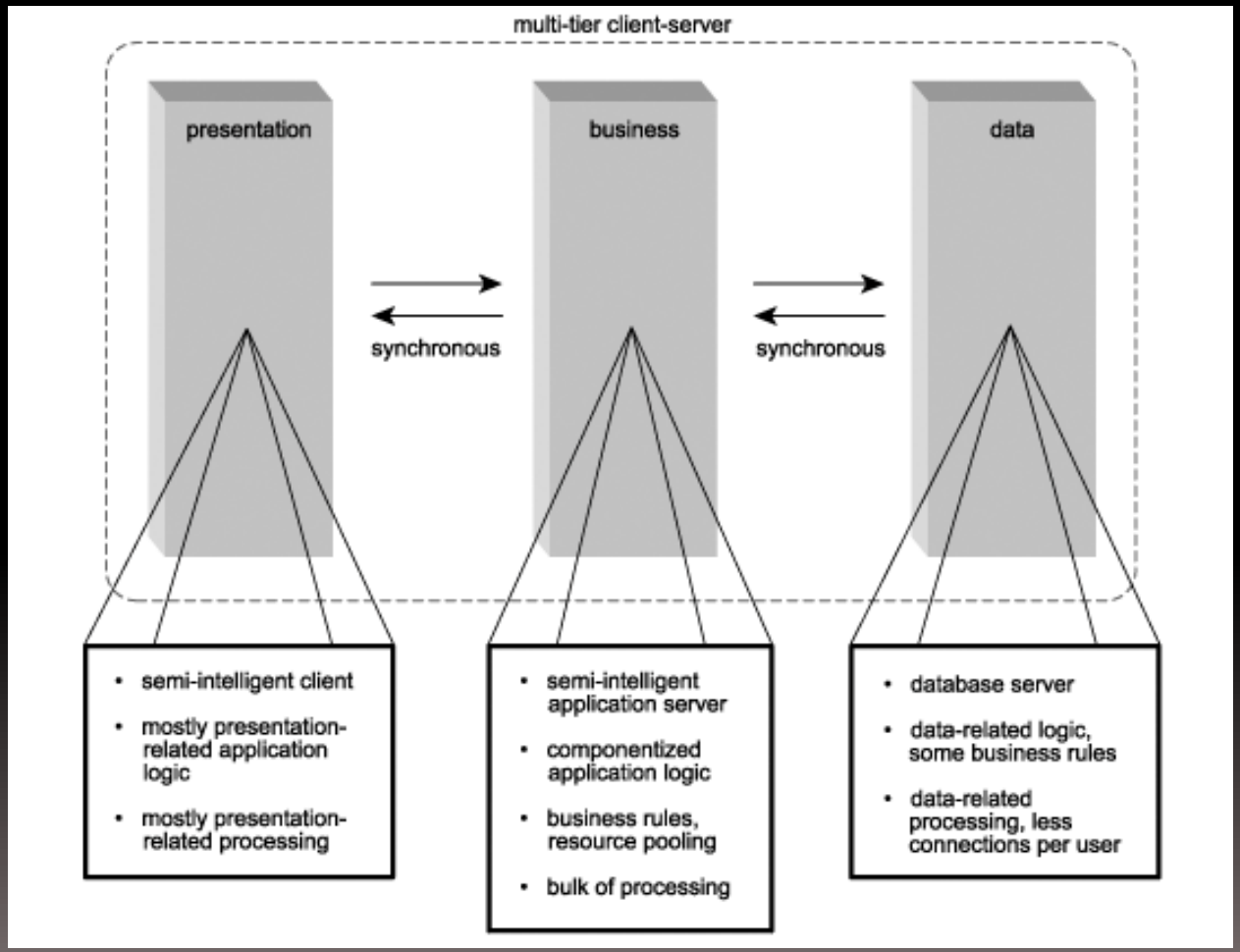
Distributed Internet architecture: a brief history

- costs and limitations associated with the two-tier client server architecture, the concept of building component-based applications hit the mainstream
- Multi-tier client-server architectures surfaced, breaking up the monolithic client executable into components designed to varying extents of compliance with object-orientation.
- Distributing application logic among multiple components (some residing on the client, others on the server) reduced deployment headaches by centralizing a greater amount of the logic on servers.
- Server-side components, now located on dedicated application servers, would then share and manage pools of database connections, alleviating the burden of concurrent usage on the database server
- A single connection could easily facilitate multiple users

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Distributed Internet architecture: a brief history

A typical multi-tier client-server architecture



SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

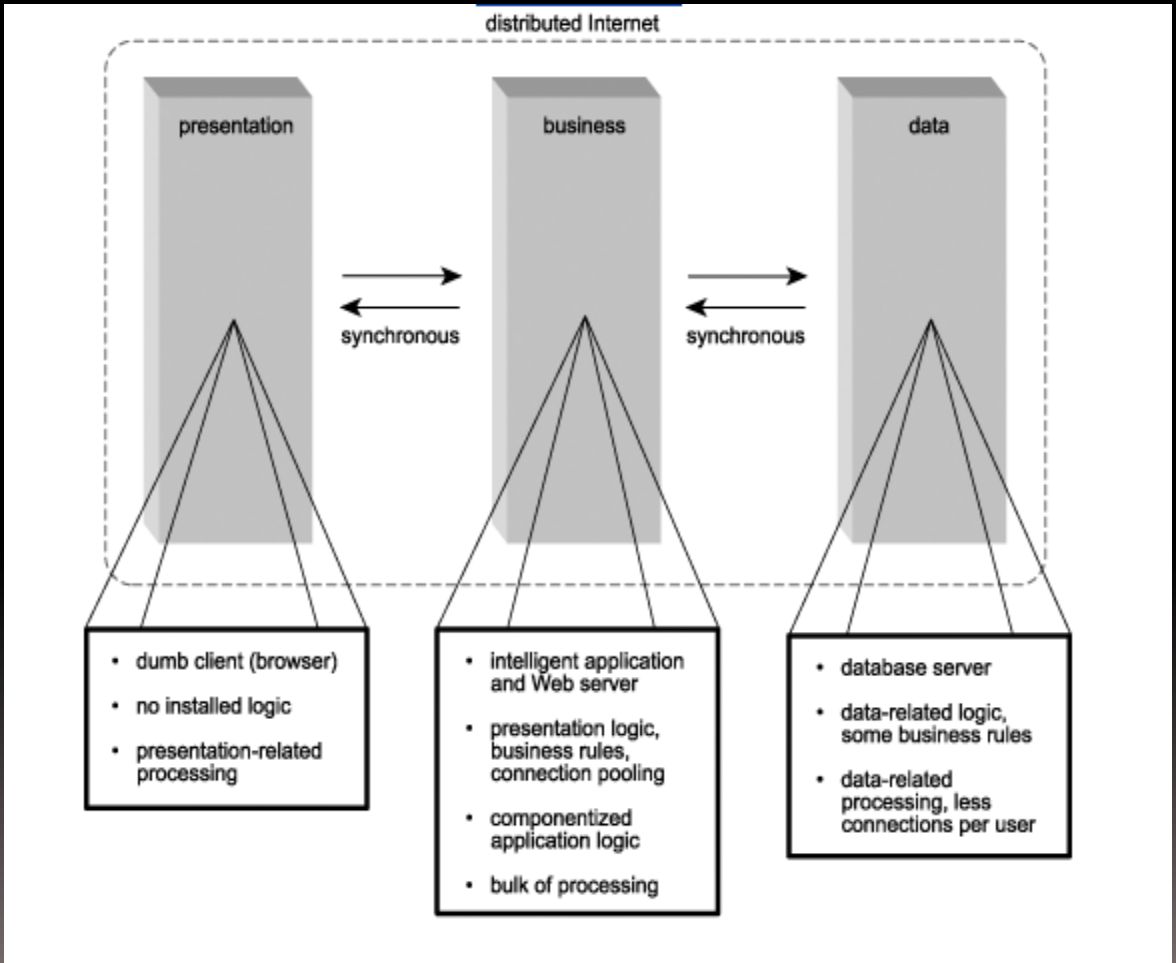
Distributed Internet architecture: a brief history

- benefits came at the cost of increased complexity and ended up shifting expense and effort from deployment issues to development and administration processes.
- Building components capable of processing multiple, concurrent requests was more difficult and problem-ridden than developing a straight-forward executable intended for a single user.
- replacing client-server database connections was the client-server remote procedure call (RPC) connection.
- RPC technologies such as CORBA and DCOM allowed for remote communication between components residing on client workstations and servers.
- Issues similar to the client-server architecture problems involving resources and persistent connections emerged.
- increased maintenance effort resulting from the introduction of the middleware layer. For example, application servers and transaction monitors required significant attention in larger environments.
- Upon the arrival of the World Wide Web as a viable medium for computing technology in the mid-to-late 90s, the multi-tiered client-server environments began incorporating Internet technology.
- the replacement of the custom software client component with the browser. Not only did this change radically alter (and limit) user-interface design, it practically shifted 100% of application logic to the server

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Distributed Internet architecture: a brief history

A typical distributed Internet architecture



SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Distributed Internet architecture: a brief history

- Distributed Internet architecture also introduced a new physical tier, the Web server.
- This resulted in HTTP replacing proprietary RPC protocols used to communicate between the user's workstation and the server.
- The role of RPC was limited to enabling communication between remote Web and application servers.

Application logic

- distributed Internet applications place all of their application logic on the server side.
- Even client-side scripts intended to execute in response to events on a Web page are downloaded from the Web server upon the initial HTTP request.
- With none of the logic existing on the client workstation, the entire solution is centralized.

The emphasis is therefore on:

1. how application logic should be partitioned
2. where the partitioned units of processing logic should reside
3. how the units of processing logic should interact

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Distributed Internet architecture: a brief history

- From a physical perspective, service-oriented architecture is very similar to distributed Internet architecture.
- Provider logic resides on the server end where it is broken down into separate units.
- Traditional distributed applications consist of a series of components that reside on one or more application servers.
- Components residing on the same server communicate via proprietary APIs, as per the public interfaces they expose.
- RPC protocols are used to accomplish the same communication across server boundaries. This is made possible through the use of local proxy stubs that represent components in remote locations
- Contemporary SOAs still employ and rely on components
- creation of services that encapsulate some or all of these components.
- The purpose of wrapping functionality within a service is to expose that functionality via an open, standardized interface irrespective of the technology used to implement the underlying logic
- While traditional components provide methods that, once invoked, send and receive parameter data, Web services communicate with SOAP messages.
- Even though SOAP supports RPC-style message structures, the majority of service-oriented Web service designs rely on document-style messages.

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Application processing

- Distributed Internet architecture promotes the use of proprietary communication protocols, such as DCOM and vendor implementations of CORBA for remote data exchange.
- They can support the creation of stateful and stateless components that primarily interact with synchronous data exchanges
- SOA relies on message-based communication.
- This involves the serialization, transmission, and deserialization of SOAP messages containing XML document payloads.
- Processing steps can involve the conversion of relational data into an XML-compliant structure, the validation of the XML document prior and subsequent to transmission, and the parsing of the document and extraction of the data by the recipient.
- This messaging framework promotes the creation of autonomous services that support a wide range of message exchange patterns. Though synchronous communication is fully supported, asynchronous patterns are encouraged, as they provide further opportunities to optimize processing by minimizing communication

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Distributed Internet architecture: a brief history

Technology

- Initial architectures consisted of components, server-side scripts, and raw Web technologies, such as HTML and HTTP.
- Improvements in middleware allowed for increased processing power and transaction control.
- The emergence of XML introduced sophisticated data representation that actually gave substance to content transmitted via Internet protocols.
- The subsequent availability of Web services allowed distributed Internet applications to cross proprietary platform boundaries.

SOA VS. DISTRIBUTED INTERNET ARCHITECTURE:

Security

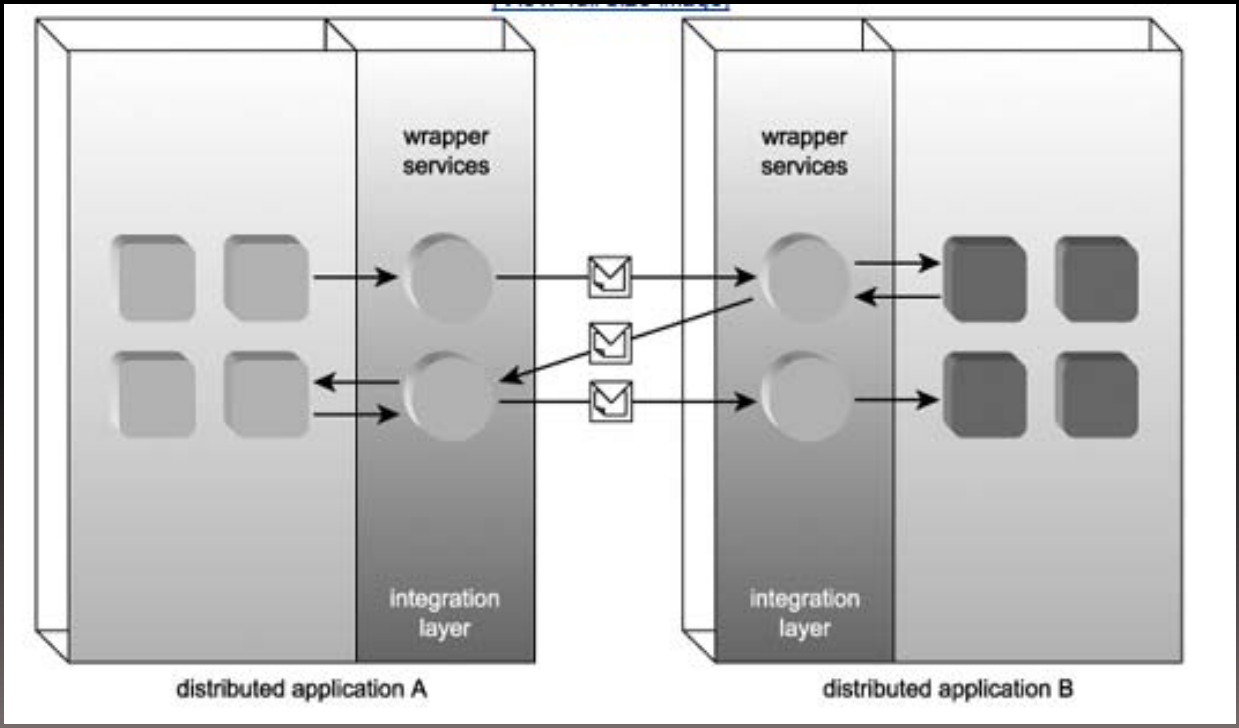
- Encryption also is added to the wide open HTTP protocol to allow data to be protected during transmission beyond the Web server.
- SOAs depart from this model by introducing wholesale changes to how security is incorporated and applied.
- Relying heavily on the extensions and concepts established by the WS-Security framework, the security models used within SOA emphasize the placement of security logic onto the messaging level.
- SOAP messages provide header blocks in which security logic can be stored.

SOA VS. HYBRID WEB SERVICE ARCHITECTURE

Web services as component wrappers

- The primary role of Web services is to introduce an integration layer that consists of wrapper services that enable synchronous communication via SOAP-compliant integration channels

Wrapper services encapsulating components



SOA VS. HYBRID WEB SERVICE ARCHITECTURE

Web services as component wrappers

- integration channels are primarily utilized in integration architectures to facilitate communication with other applications or outside partners.
- They also are used to enable communication with other (more service-oriented) solutions and to take advantage of some of the features offered by third-party utility Web services.
- SOA provides strong support for a variety of messaging models (based on both synchronous and asynchronous exchanges).
- Additionally, Web services within SOAs are subject to specific design requirements, such as those provided by service-orientation principles.
- These and other characteristics support the pursuit of consistent loose coupling.

SERVICE-ORIENTATION AND OBJECT-ORIENTATION

- Service-orientation emphasizes loose coupling between units of processing logic (services).
- Although object-orientation supports the creation of reusable, loosely coupled programming routines, much of it is based on predefined class dependencies, resulting in more tightly bound units of processing logic (objects).

- Service-orientation encourages coarse-grained interfaces (service descriptions) so that every unit of communication (message) contains as much information as possible for the completion of a given task.
- Object-oriented programming fully supports fine-grained interfaces (APIs) so that units of communication (RPC or local API calls) can perform various sized tasks.

- Service-orientation expects the scope of a unit of processing logic (service) to vary significantly.
- Object-oriented units of logic (objects) tend to be smaller and more specific in scope.

SERVICE-ORIENTATION AND OBJECT-ORIENTATION

- Service-orientation promotes the creation of activity-agnostic units of processing logic (services) that are driven into action by intelligent units of communication (messages).
- Object-orientation encourages the binding of processing logic with data, resulting in highly intelligent units (objects).

- Service-orientation prefers that units of processing logic (services) be designed to remain as stateless as possible.
- Object-orientation promotes the binding of data and logic, resulting in the creation of more stateful units (objects). (However, more recent component-based design approaches deviate from this tendency.)

- Service-orientation supports the composition of loosely coupled units of processing logic (services).
- Object-orientation supports composition but also encourages inheritance among units of processing logic (objects), which can lead to tightly coupled dependencies.

WEB SERVICES AND PRIMITIVE SOA

The Web services framework:

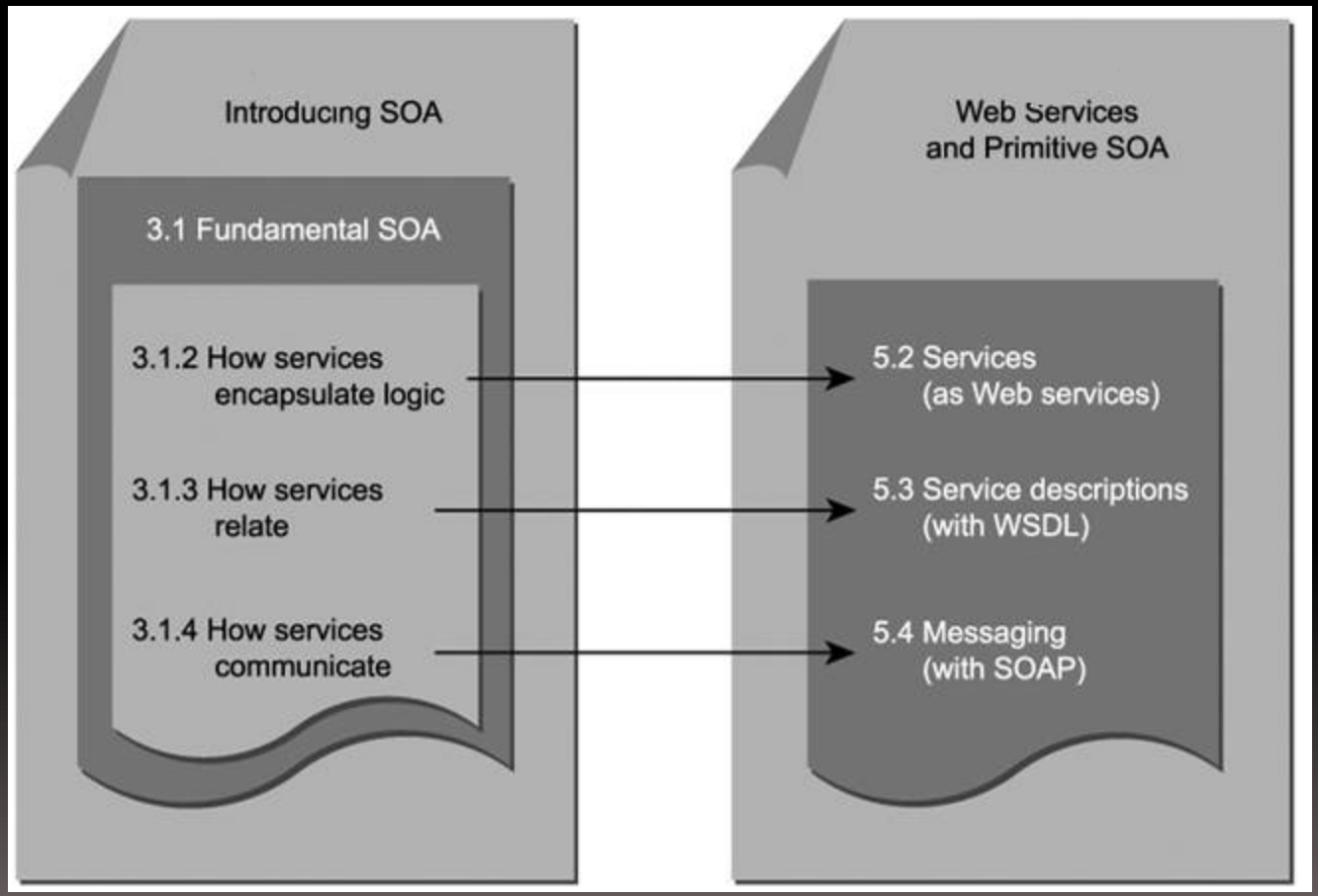
- A technology framework is a collection of things. It can include one or more architectures, technologies, concepts, models, and even sub-frameworks. The framework established by Web services is comprised of all of these parts.

Specifically, this framework is characterized by:

- an abstract (vendor-neutral) existence defined by standards organizations and implemented by (proprietary) technology platforms
- core building blocks that include Web services, service descriptions, and messages
- a communications agreement centered around service descriptions based on WSDL
- a messaging framework comprised of SOAP technology and concepts
- a service description registration and discovery architecture sometimes realized through UDDI
- a well-defined architecture that supports messaging patterns and compositions
- a second generation of Web services extensions (also known as the WS-* specifications) continually broadening its underlying feature-set

WEB SERVICES AND PRIMITIVE SOA

The structural relationship



SERVICES (AS WEB SERVICES)

- Web services provide the potential of fulfilling primitive requirements, but they need to be intentionally designed to do so. This is because the Web services framework is flexible and adaptable.

- Web services can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or they can be designed to be fully SOA-compliant. This flexibility has allowed Web services to become part of many existing application environments and has been one of the reasons behind their popularity.

- It also reveals the fact that Web services are not necessarily inherently service-oriented.

SERVICES (AS WEB SERVICES)

Most basic Web services design concepts:

Fundamentally, every Web service can be associated with:

- a temporary classification based on the roles it assumes during the runtime processing of a message
  - service roles (temporary classifications)

- a permanent classification based on the application logic it provides and the roles it assumes within a solution environment
  - service models (permanent classifications)

SERVICES (AS WEB SERVICES)

Service roles:

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

SERVICES (AS WEB SERVICES)

Service provider:

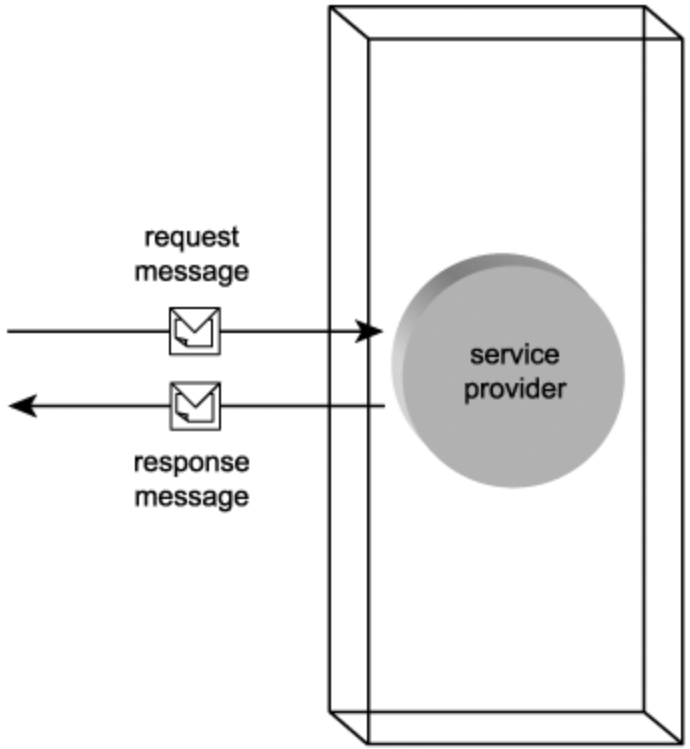
The service provider role is assumed by a Web service under the following conditions:

- The Web service is invoked via an external source, such as a service requestor
- The Web service provides a published service description offering information about its features and behavior.
- Depending on the type of message exchange used when invoking a service provider, the service provider may reply to a request message with a response message.

SERVICES (AS WEB SERVICES)

Service provider:

As the recipient of a request message, the Web service is classified as a service provider.



SERVICES (AS WEB SERVICES)

Service requestor:

- Any unit of processing logic capable of issuing a request message that can be understood by the service provider is classified as a service requestor.

- A Web service is always a service provider but also can act as a service requestor.

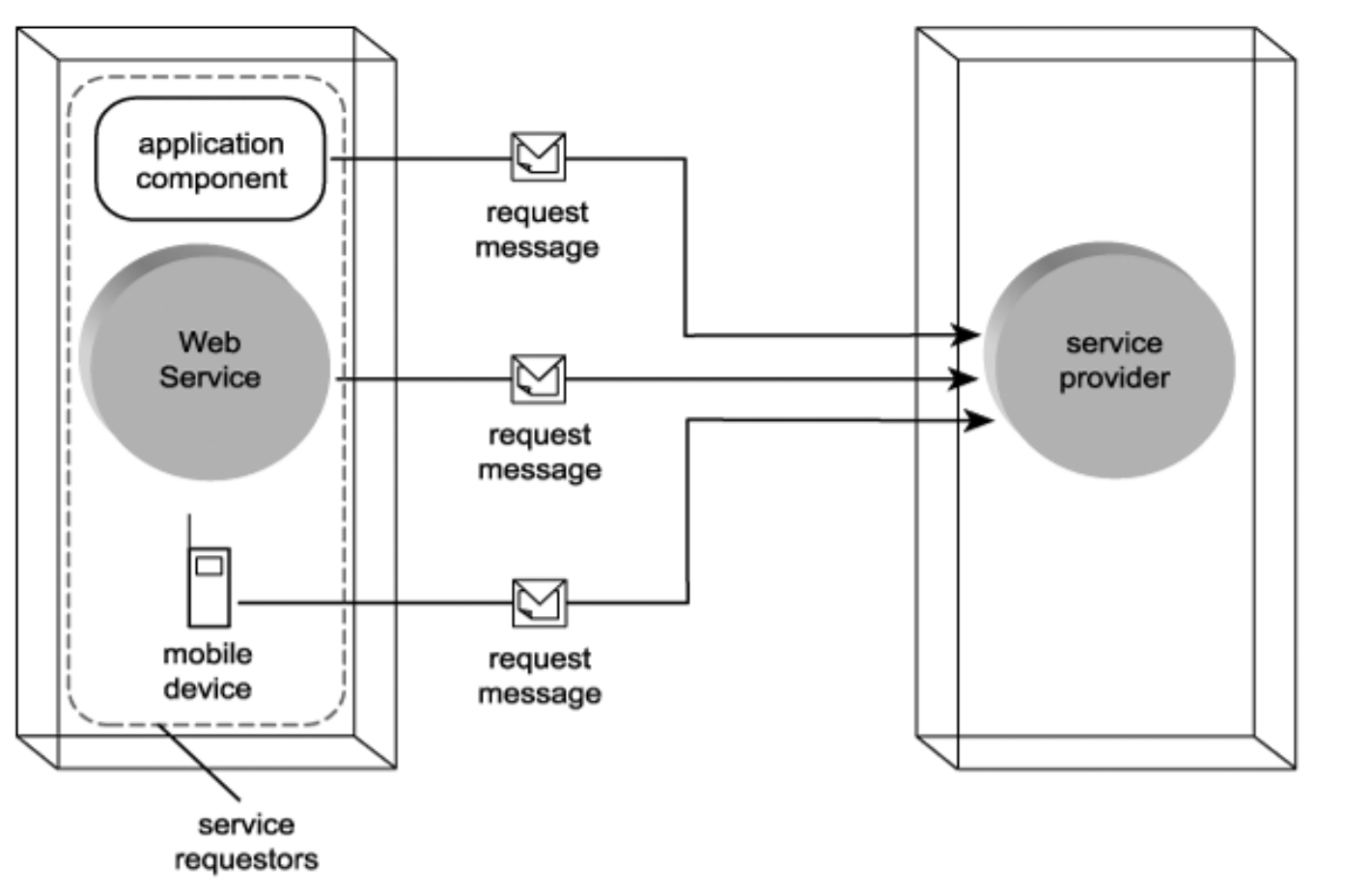
A Web service takes on the service requestor role under the following circumstances:

- The Web service invokes a service provider by sending it a message
- The Web service searches for and assesses the most suitable service provider by studying available service descriptions.

SERVICES (AS WEB SERVICES)

Service requestor:

The sender of the request message is classified as a service requestor



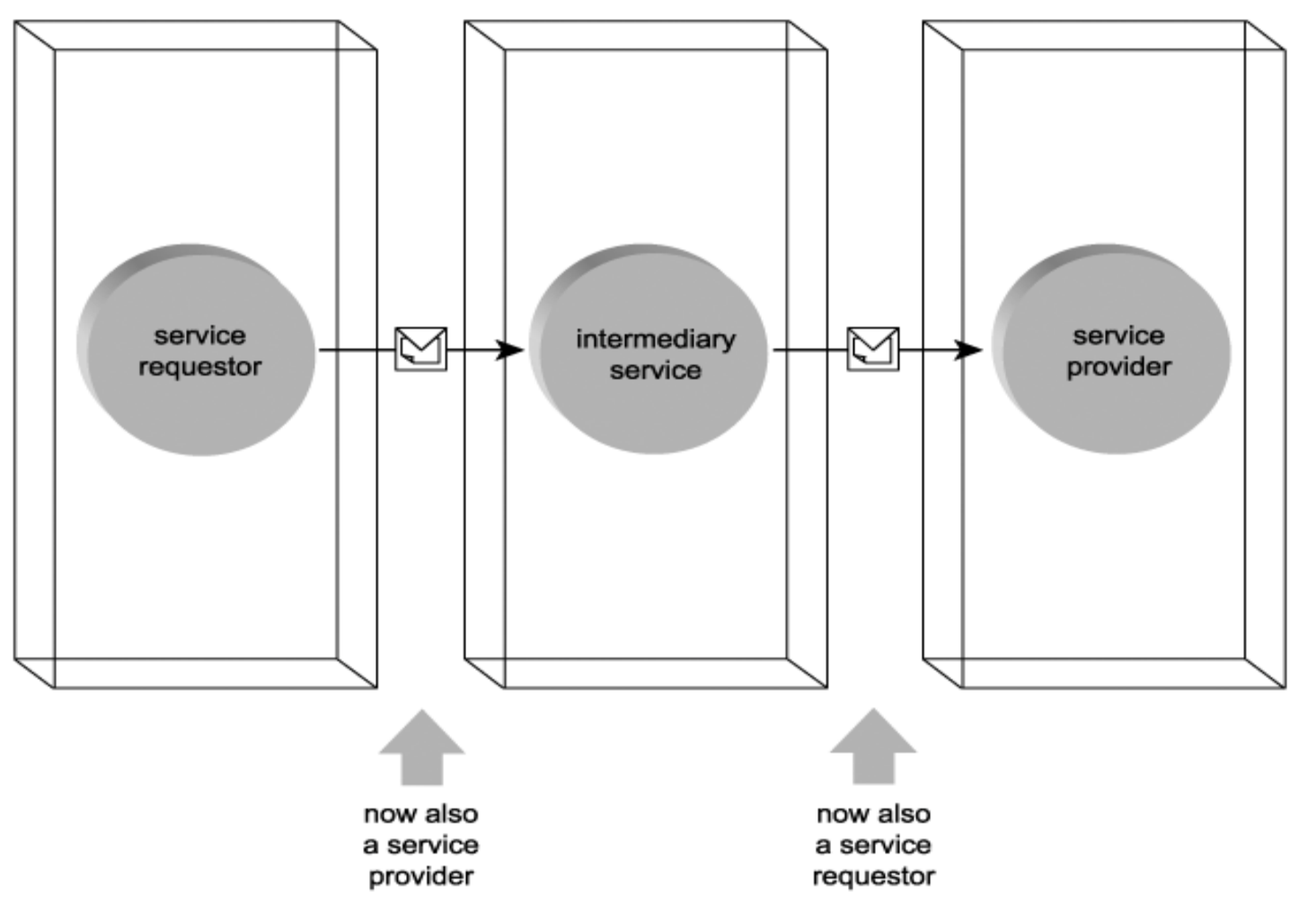
SERVICES (AS WEB SERVICES)

Intermediaries:

- The communications framework established by Web services contrasts the predictable nature of traditional point-to-point communications channels.
- Web services communication is based on the use of messaging paths, which can best be described as point-to-* paths.
- once a service provider submits a message, it can be processed by multiple intermediate routing and processing agents before it arrives at its ultimate destination.
- Web services and service agents that route and process a message after it is initially sent and before it arrives at its ultimate destination are referred to as intermediaries or intermediary services.
- Because an intermediary receives and submits messages, it always transitions through service provider and service requestor roles

SERVICES (AS WEB SERVICES)

Intermediaries:



The intermediary service transitions through service provider and service requestor roles while processing a message

SERVICES (AS WEB SERVICES)

Intermediaries:

Types of intermediaries.

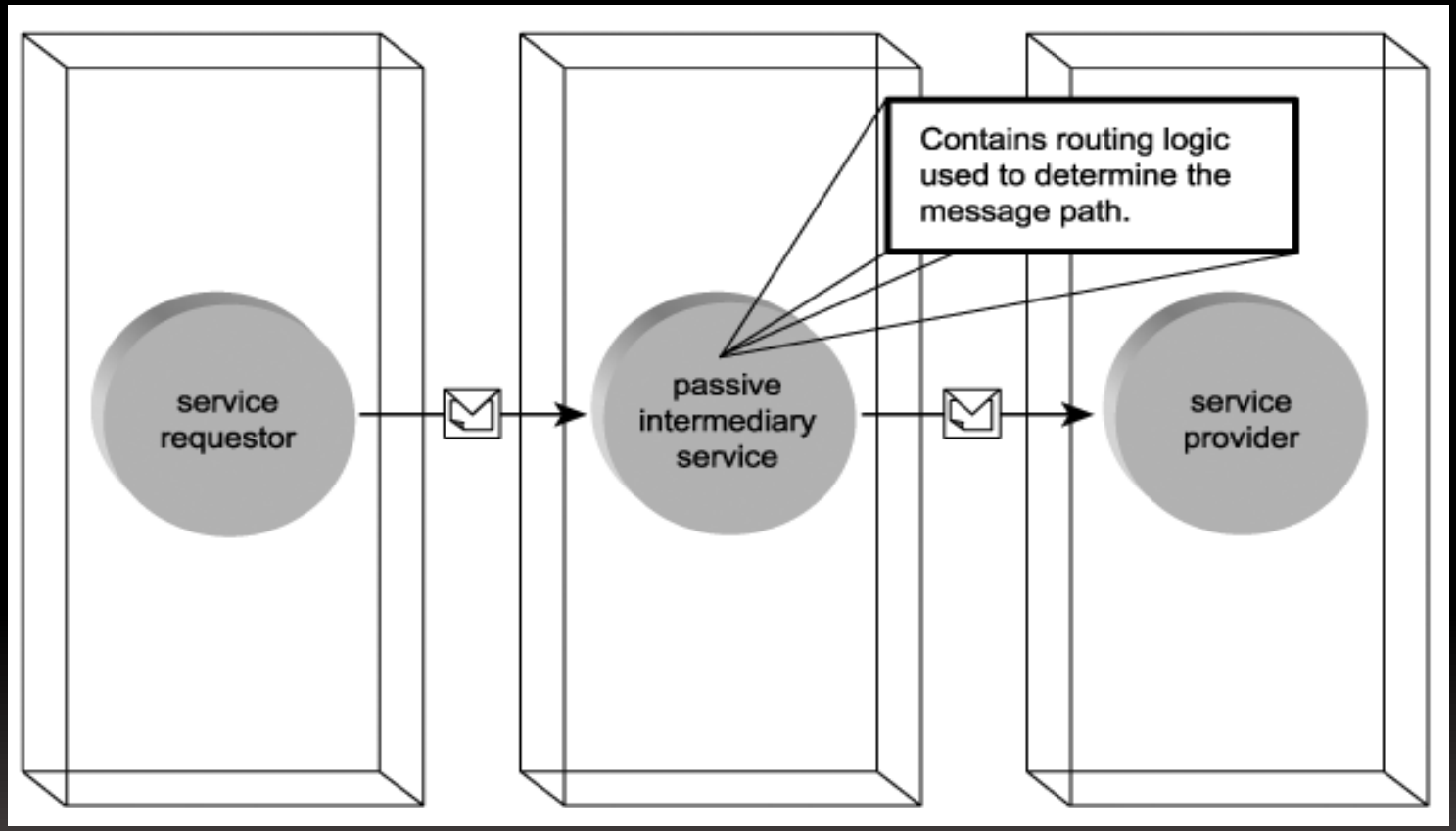
1. passive intermediary
2. active intermediaries

1. passive intermediary

- typically responsible for routing messages to a subsequent location .
- It may use information in the SOAP message header to determine the routing path, or it may employ native routing logic to achieve some level of load balancing.
- intermediary is because it does not modify the message.

SERVICES (AS WEB SERVICES)

passive **Intermediaries**:



A passive intermediary service processing a message without altering its contents

SERVICES (AS WEB SERVICES)

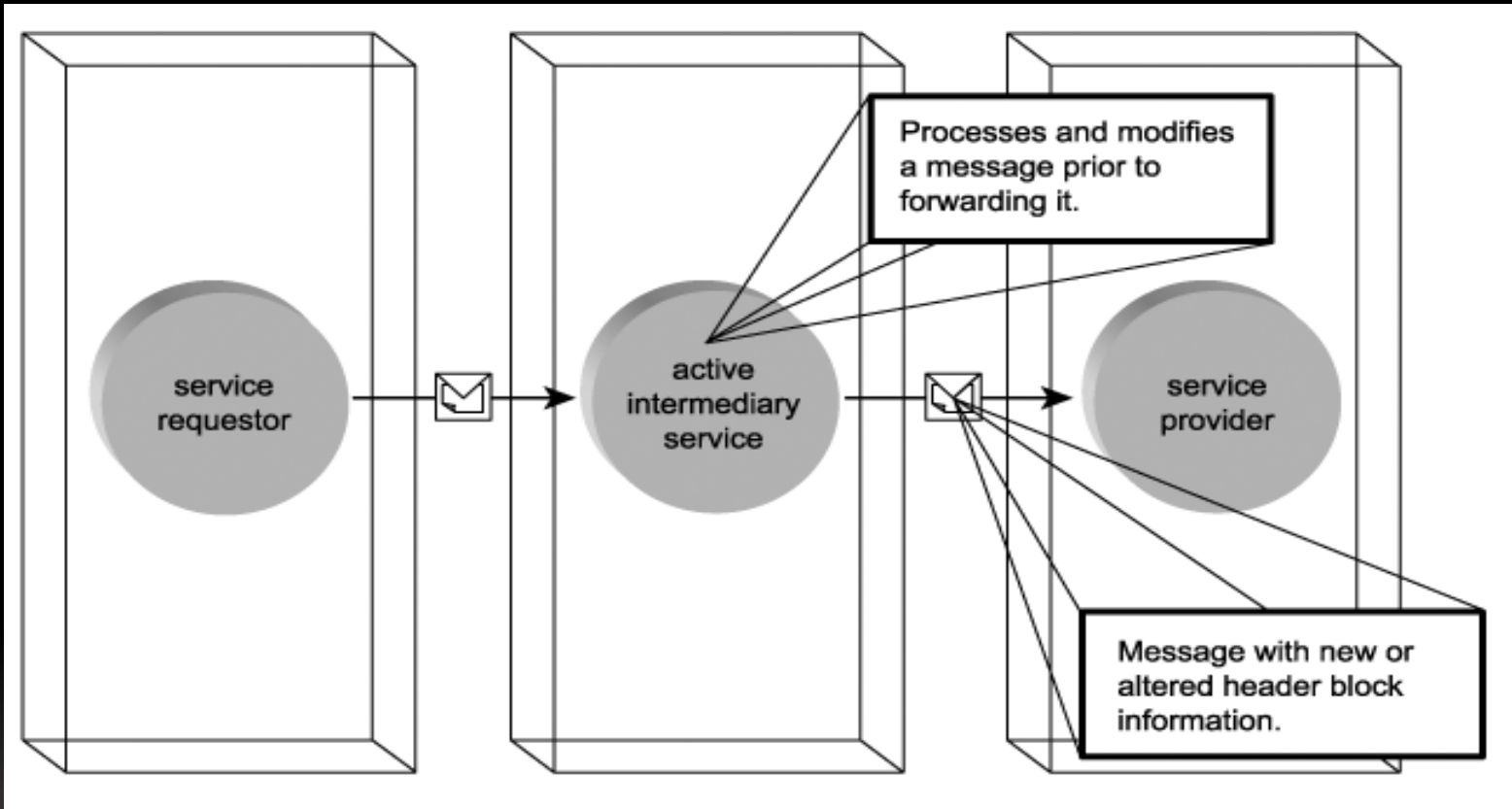
Intermediaries:

2. Active intermediary

- route messages to a forwarding destination.
- Prior to transmitting a message, however, these services actively process and alter the message contents .
- Typically, active intermediaries will look for particular SOAP header blocks and perform some action in response to the information they find there.
- They almost always alter data in header blocks and may insert or even delete header blocks entirely.

SERVICES (AS WEB SERVICES)

Active **Intermediaries**:



An active intermediary service

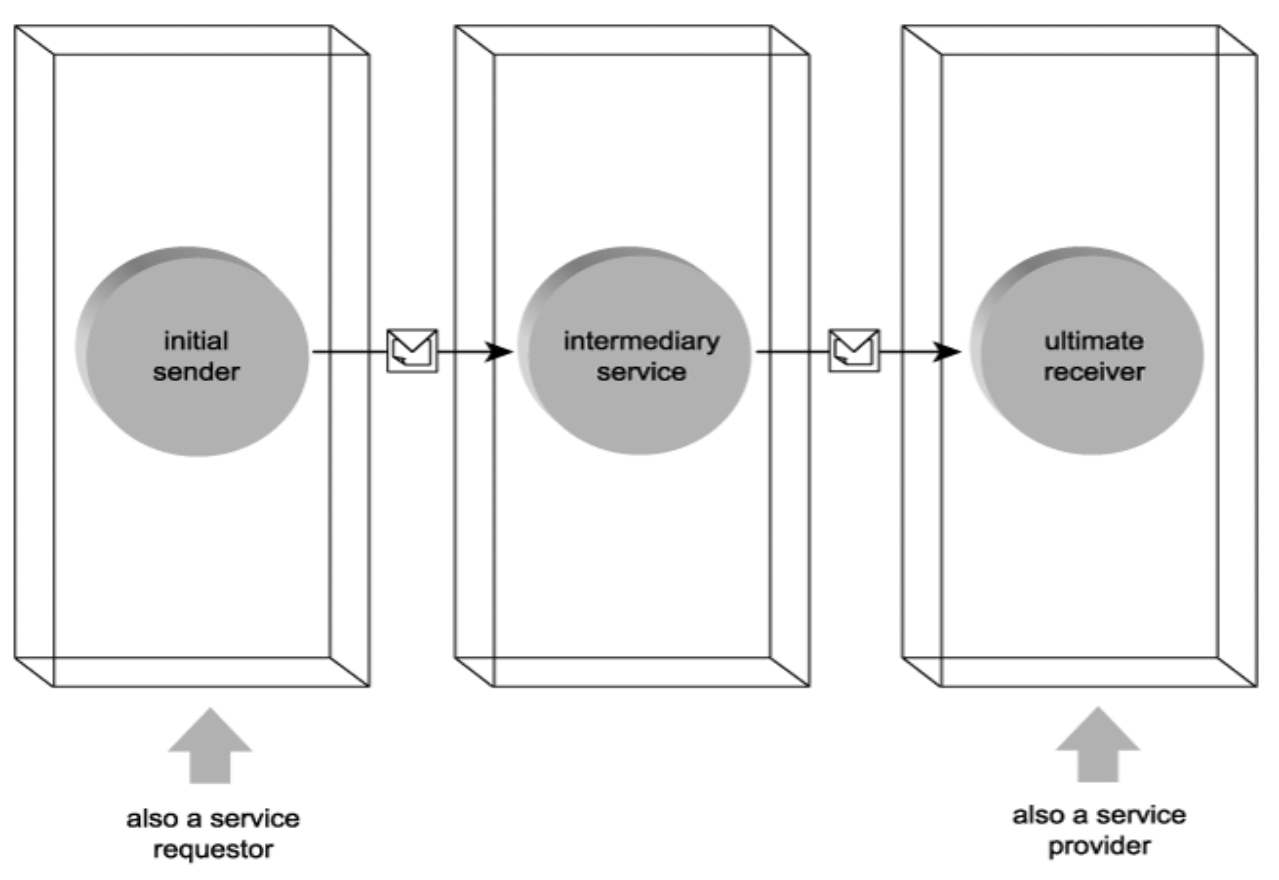
SERVICES (AS WEB SERVICES)

Initial sender and ultimate receiver:

- Initial senders are simply service requestors that initiate the transmission of a message.
- the initial sender is always the first Web service in a message path.
- The counterpart to this role is the ultimate receiver.
- This label identifies service providers that exist as the last Web service along a message's path
- intermediary services can never be initial senders or ultimate receivers within the scope of a service activity.

SERVICES (AS WEB SERVICES)

Initial sender and ultimate receiver :



Web services acting as initial sender and ultimate receiver

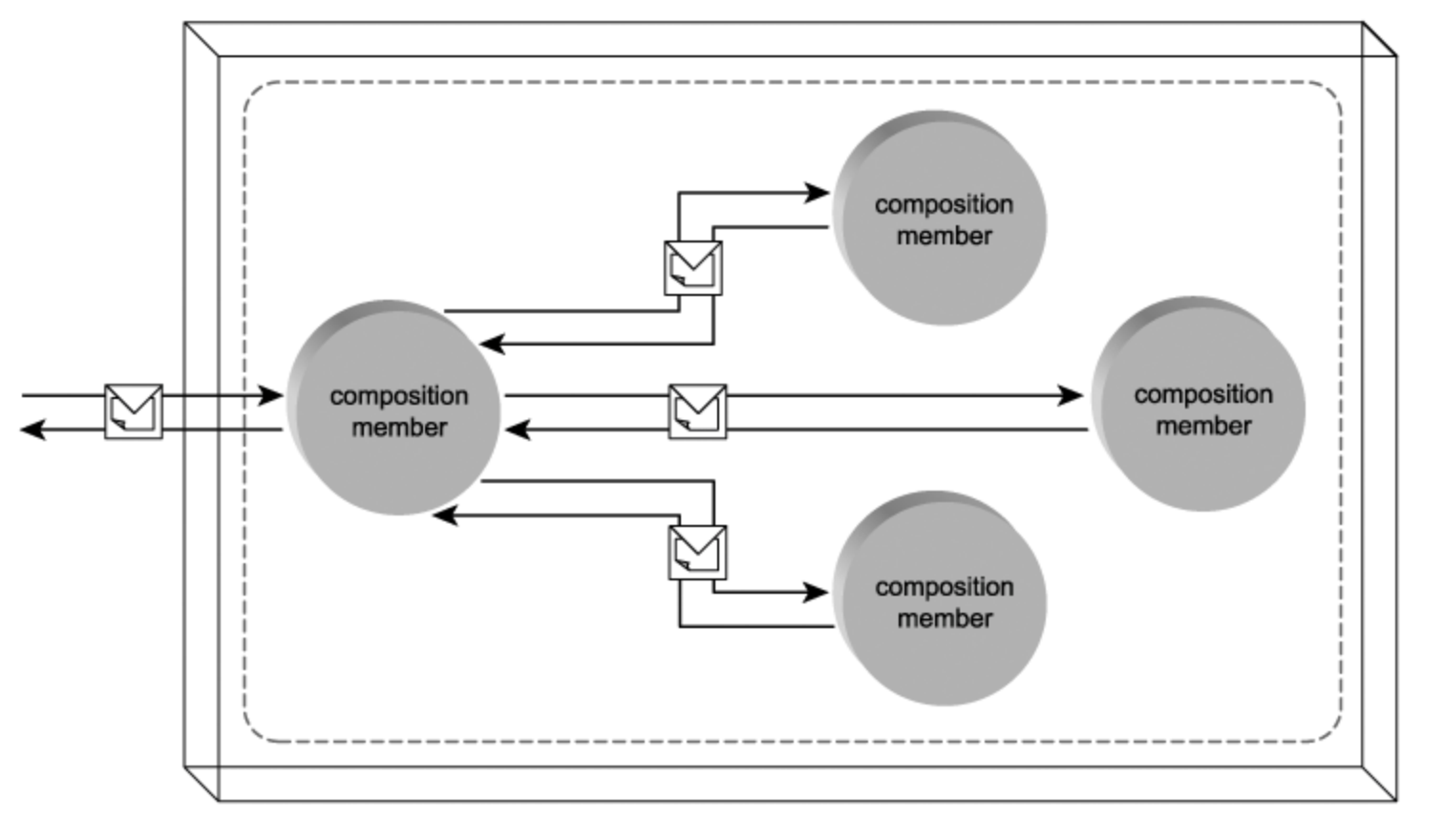
SERVICES (AS WEB SERVICES)

Service compositions

- The term does not apply to a single Web service, but to a composite relationship between a collection of services.
- Any service can enlist one or more additional services to complete a given task.
- any of the enlisted services can call other services to complete a given sub-task.
- each service that participates in a composition assumes an individual role of service composition member
- Service compositions also are referred to as service assemblies

SERVICES (AS WEB SERVICES)

Service compositions



A service composition consisting of four members

SERVICES (AS WEB SERVICES)

Service models

- The roles explored so far are agnostic to the nature of the functionality being provided by the Web service.

- They are generic states that a service can enter within a generic context.

- The manner in which services are being utilized in the real world, though, has led to a classification based on the nature of the application logic they provide, as well as their business-related roles within the overall solution. These classifications are known as service models.

SERVICES (AS WEB SERVICES)

Business service model

- Within an SOA, the business service represents the most fundamental building block.
- It encapsulates a distinct set of business logic within a well-defined functional boundary.
- It is fully autonomous but still not limited to executing in isolation, as business services are frequently expected to participate in service compositions.

Business services are used within SOAs as follows:

- as fundamental building blocks for the representation of business logic
- to represent a corporate entity or information set
- to represent business process logic
- as service composition members

SERVICES (AS WEB SERVICES)

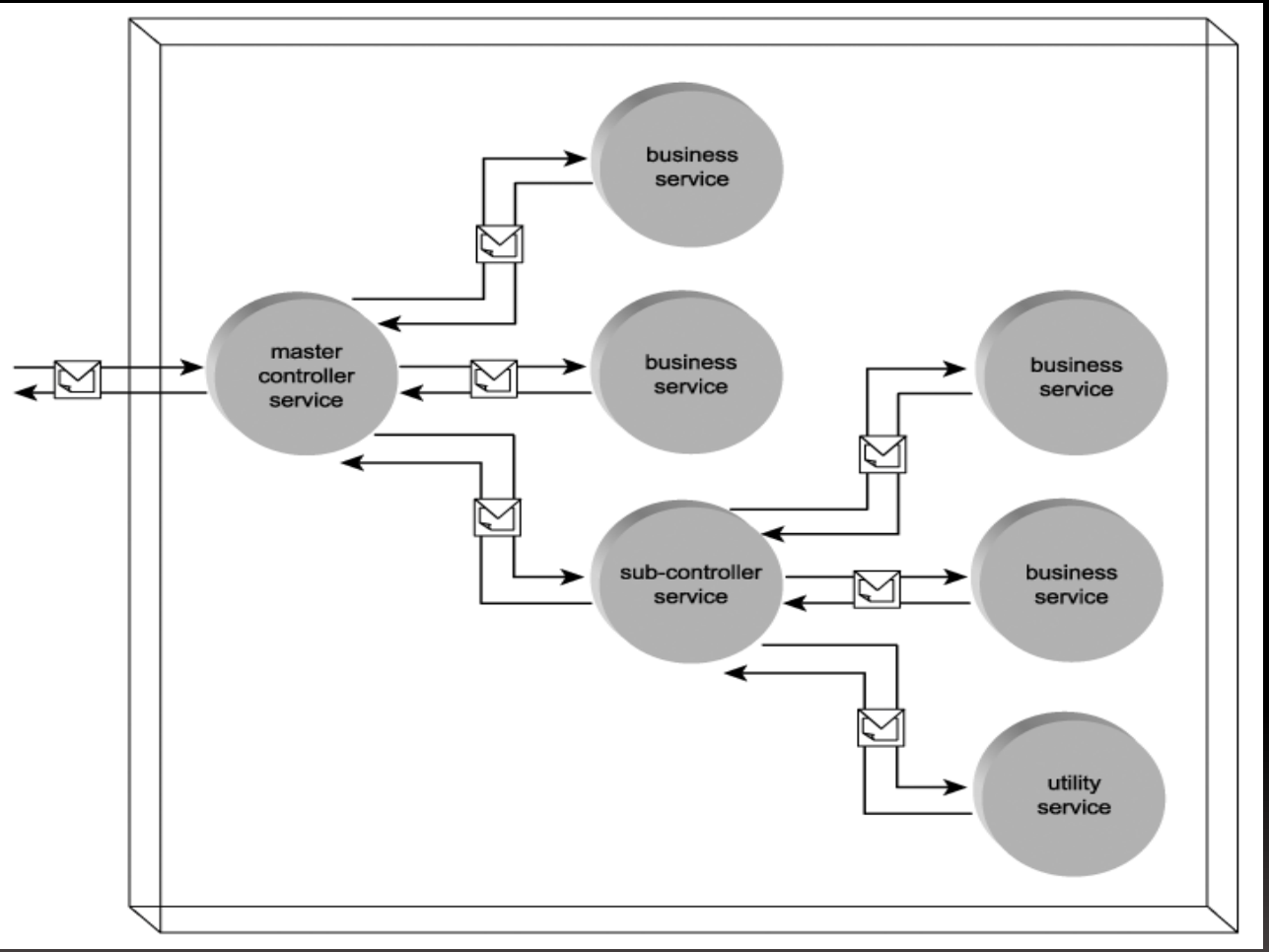
Utility service model

- Any generic Web service or service agent designed for potential reuse can be classified as a utility service.
- the reusable functionality be completely generic and non-application specific in nature.

Utility services are used within SOAs as follows:

- as services that enable the characteristic of reuse within SOA
- as solution-agnostic intermediary services
- as services that promote the intrinsic interoperability characteristic of SOA
- as the services with the highest degree of autonomy

SERVICES (AS WEB SERVICES)

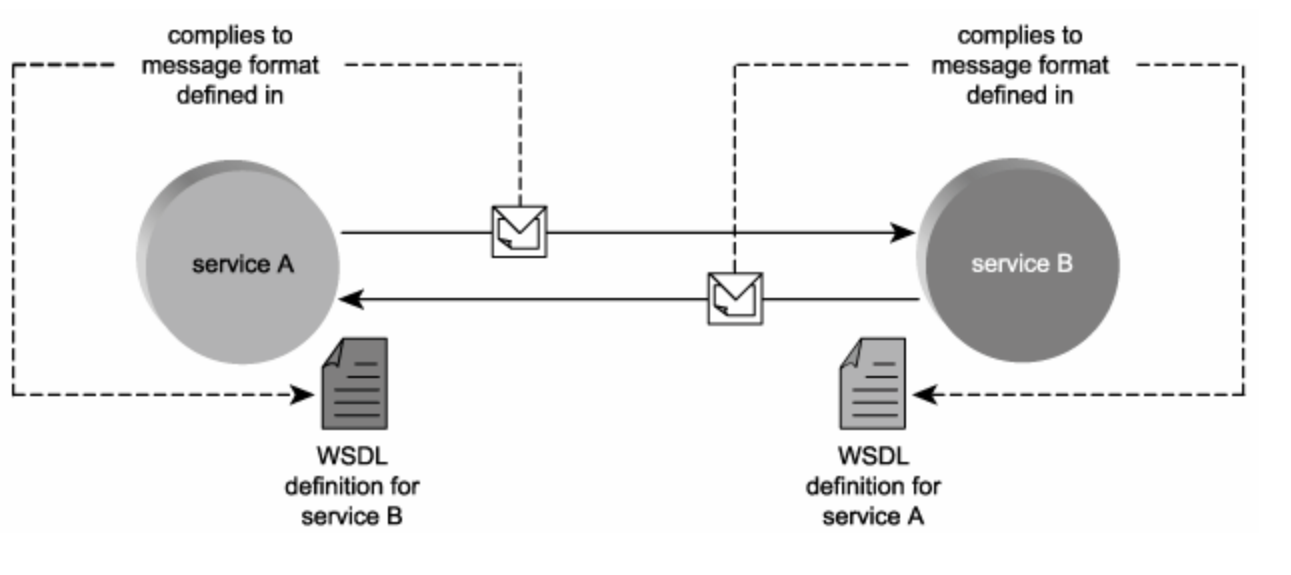


A service composition consisting of a master controller, a subcontroller, four business services, and one utility service

SERVICE DESCRIPTIONS (WITH WSDL)

- description documents are required to accompany any service wanting to act as an ultimate receiver.
- The primary service description document is the WSDL definition

WSDL definitions enable loose coupling between services



SERVICE DESCRIPTIONS (WITH WSDL)

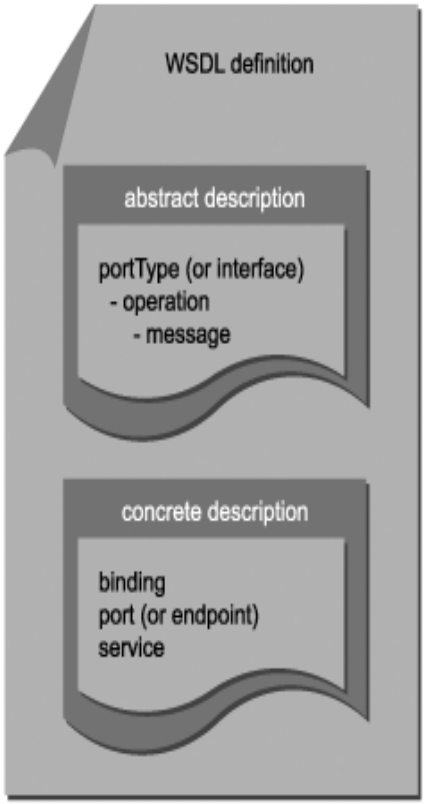
Service endpoints and service descriptions:

- A WSDL describes the point of contact for a service provider, also known as the service endpoint or just endpoint.
- It provides a formal definition of the endpoint interface (so that requestors wishing to communicate with the service provider know exactly how to structure request messages) and also establishes the physical location (address) of the service.
- A WSDL service description (also known as WSDL service definition or just WSDL definition) can be separated into two categories:
  - abstract description
  - concrete description

SERVICE DESCRIPTIONS (WITH WSDL)

Service endpoints and service descriptions:

WSDL document consisting of abstract and concrete parts that collectively describe a service endpoint



SERVICE DESCRIPTIONS (WITH WSDL)

Abstract description

- An abstract description establishes the interface characteristics of the Web service without any reference to the technology used to host or enable a Web service to transmit messages.
- By separating this information, the integrity of the service description can be preserved regardless of what changes might occur to the underlying technology platform.

Three main parts that comprise an abstract description.

- **portType,**
- **operation,**
- **and message**

SERVICE DESCRIPTIONS (WITH WSDL)

Abstract description

- The parent portType section of an abstract description provides a high-level view of the service interface by sorting the messages a service can process into groups of functions known as operations.
- Each operation represents a specific action performed by the service.
- A service operation is comparable to a public method used by components in traditional distributed applications.
- Like component methods, operations also have input and output parameters. Because Web services rely exclusively on messaging-based communication, parameters are represented as messages. Therefore, an operation consists of a set of input and output messages.

SERVICE DESCRIPTIONS (WITH WSDL)

Abstract description

- The transmission sequence of these messages can be governed by a predetermined message exchange pattern that also is associated with the operation.

- The term "portType" is being renamed to "interface" in version 2.0 of the WSDL specification

SERVICE DESCRIPTIONS (WITH WSDL)

Concrete description

- For a Web service to be able to execute any of its logic, it needs for its abstract interface definition to be connected to some real, implemented technology.

- Because the execution of service application logic always involves communication, the abstract Web service interface needs to be connected to a physical transport protocol.

- This connection is defined in the concrete description portion of the WSDL file, which consists of three related parts:
  - **binding,**
  - **port,**
  - **and service**

SERVICE DESCRIPTIONS (WITH WSDL)

Concrete description

- A WSDL description's binding describes the requirements for a service to establish physical connections or for connections to be established with the service.
- binding represents one possible transport technology the service can use to communicate.
- SOAP is the most common form of binding, but others also are supported. A binding can apply to an entire interface or just a specific operation.
- Related to the binding is the port, which represents the physical address at which a service can be accessed with a specific protocol.

SERVICE DESCRIPTIONS (WITH WSDL)

Concrete description

- This piece of physical implementation data exists separately to allow location information to be maintained independently from other aspects of the concrete description.
- Within the WSDL language, the term service is used to refer to a group of related endpoints.
- The term "port" is being renamed "endpoint" in version 2.0 of the WSDL specification.

SERVICE DESCRIPTIONS (WITH WSDL)

Metadata and service contracts

- abstract and concrete descriptions provided by a WSDL definition express technical information as to how a service can be interfaced with and what type of data exchange it supports.

- WSDL definitions frequently rely on XSD schemas to formalize the structure of incoming and outgoing messages.

- Another common supplemental service description document is a policy.

- Policies can provide
  rules,
  preferences,
  and processing details above and beyond what is expressed through the WSDL and XSD schema documents

SERVICE DESCRIPTIONS (WITH WSDL)

Three separate documents that each describe an aspect of a service:

1. WSDL definition
2. XSD schema
3. Policy

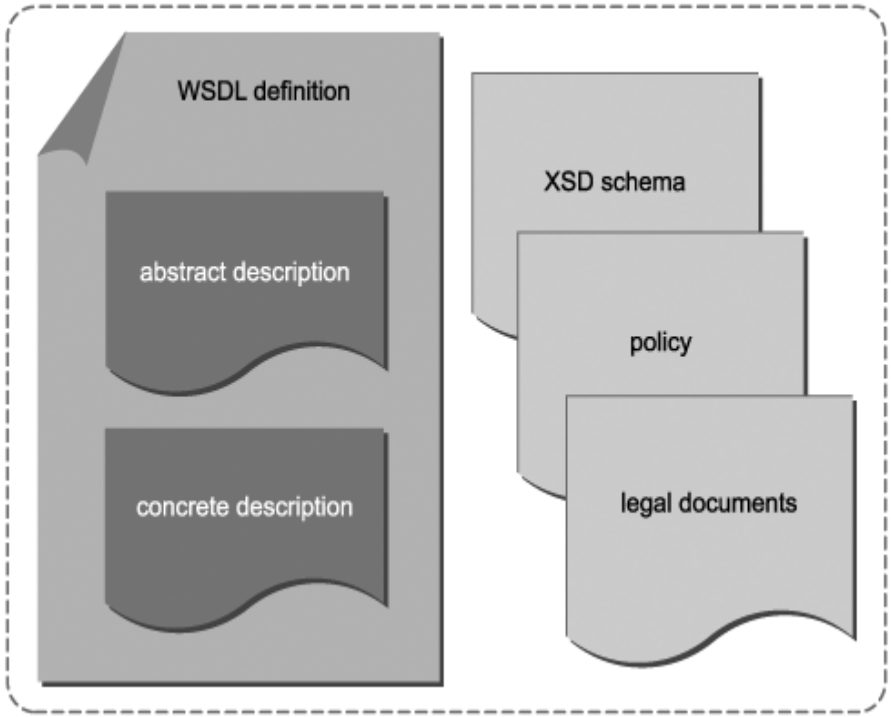
- Each of these three service description documents can be classified as service metadata, as each provides information about the service.

- Service description documents can be collectively viewed as establishing a service contracta set of conditions that must be met and accepted by a potential service requestor to enable successful communication

SERVICE DESCRIPTIONS (WITH WSDL)

Metadata and service contracts

A service contract comprised of a collection of service descriptions and possibly additional documents.



SERVICE DESCRIPTIONS (WITH WSDL)

Semantic descriptions

- Most of the metadata currently provided by services focuses on expressing technical information related to data representation and processing requirements.
- The most challenging part of providing a complete description of a Web service is in communicating its semantic qualities.

Examples of service semantics include:

1. how a service behaves under certain conditions
2. how a service will respond to a specific condition
3. what specific tasks the service is most suited for

SERVICE DESCRIPTIONS (WITH WSDL)

Semantic descriptions

- Most of the time service semantics are assessed by humans, either verbally by discussing the qualities of a service with its owner, or by reading supplementary documentation published alongside service descriptions.
- The ultimate goal is to provide sufficient semantic information in a structured manner so that, in some cases, service requestors can go as far as to evaluate and choose suitable service providers independently.
- Semantic information is important when dealing with external service providers, where your knowledge of another party's service is limited to the information the service owner decides to publish.
- But even within organizational boundaries, semantic characteristics tend to take on greater relevance as the amount of internal Web services grows.

SERVICE DESCRIPTIONS (WITH WSDL)

Service description advertisement and discovery

- The requirement for one service to contact another is access to the other service's description.
- As the amount of services increases within and outside of organizations, mechanisms for advertising and discovering service descriptions may become necessary.
- For example, central directories and registries become an option to keep track of the many service descriptions that become available.
- These repositories allow humans (and even service requestors) to:
  - locate the latest versions of known service descriptions
  - discover new Web services that meet certain criteria

.

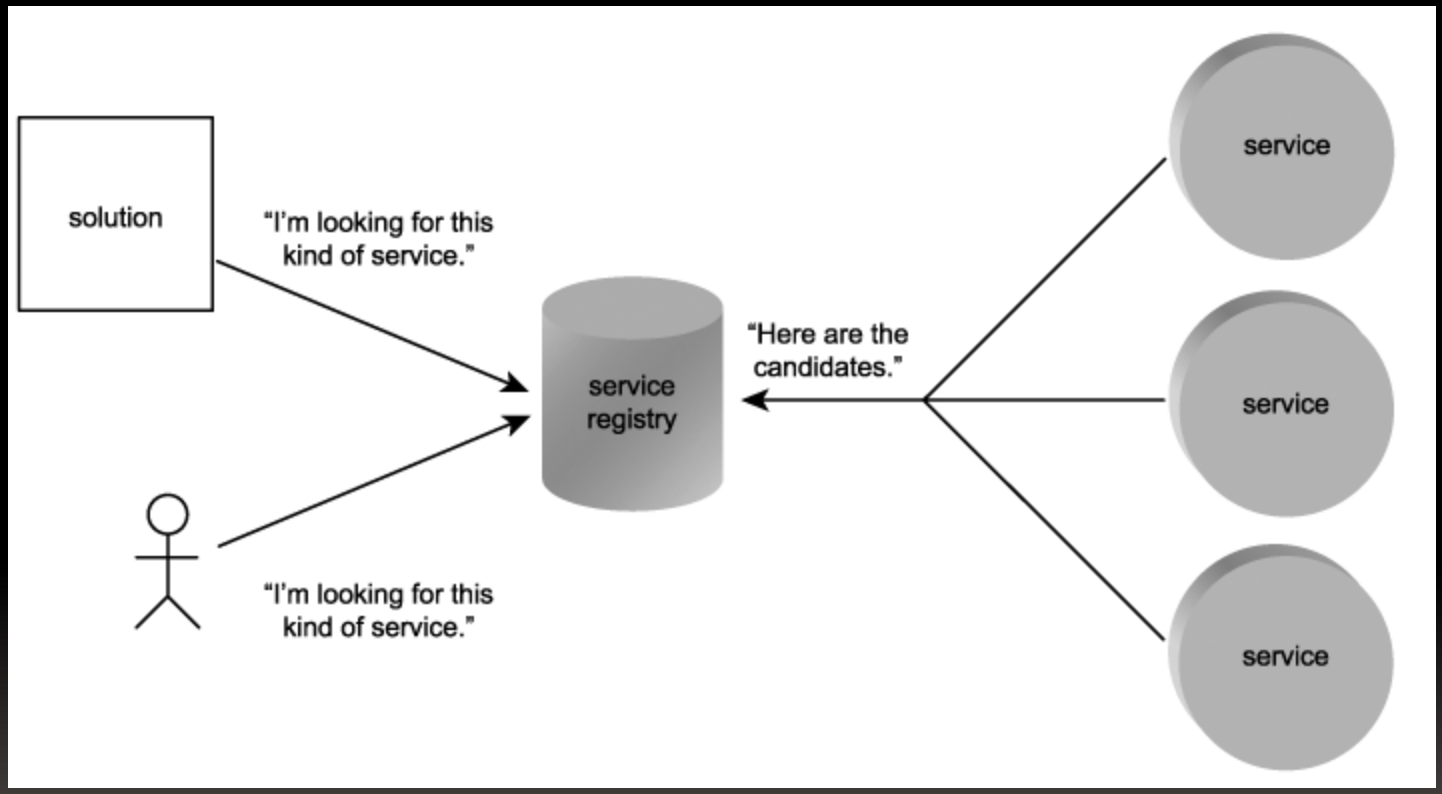
SERVICE DESCRIPTIONS (WITH WSDL)

Private and public registries

- UDDI specifies a relatively accepted standard for structuring registries that keep track of service descriptions .
- These registries can be searched manually and accessed programmatically via a standardized API.
- Public registries accept registrations from any organizations, regardless of whether they have Web services to offer. Once signed up, organizations acting as service provider entities can register their services.
- Private registries can be implemented within organization boundaries to provide a central repository for descriptions of all services the organization develops, leases, or purchases.

SERVICE DESCRIPTIONS (WITH WSDL)

Service description advertisement and discovery



SERVICE DESCRIPTIONS (WITH WSDL)

Descriptions of the primary parts that comprise UDDI registry records:

Business entities and business services

- Each public registry record consists of a business entity containing basic profile information about the organization (or service provider entity).
- Included in this record are one or more business service areas, each of which provides a description of the services offered by the business entity.
- Business services may or may not be related to the use of Web services.

SERVICE DESCRIPTIONS (WITH WSDL)

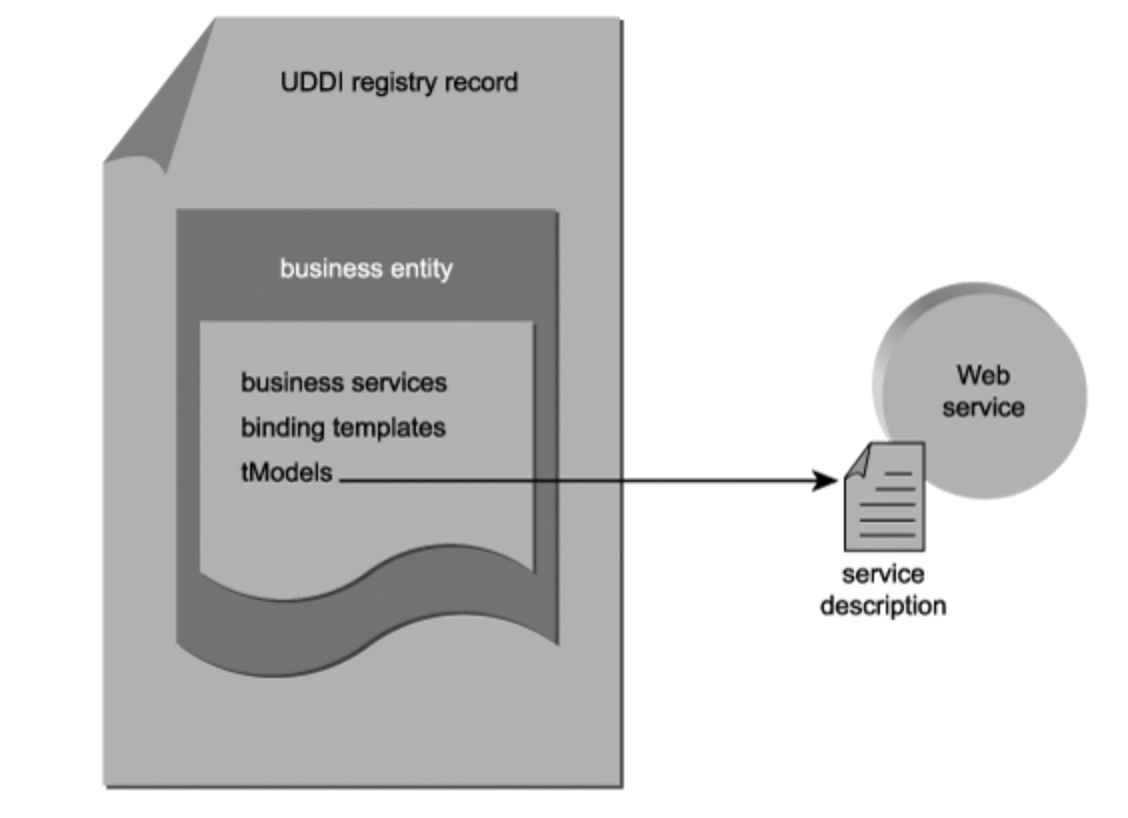
Binding templates and tModels

- WSDL definitions stored implementation information separately from the actual interface design.
- This resulted in an interface definition that existed independently from the transport protocols to which it was eventually bound.
- Registry records follow the same logic in that they store binding information in a separate area, called the binding template.
- Each business service can reference one or more binding templates.
- The information contained in a binding template may or may not relate to an actual service.
- For example, a binding template may simply point to the address of a Web site. However, if a Web service is being represented, then the binding template references a tModel.
- The tModel section of a UDDI record provides pointers to actual service descriptions

SERVICE DESCRIPTIONS (WITH WSDL)

Binding templates and tModels

The basic structure of a UDDI business entity record



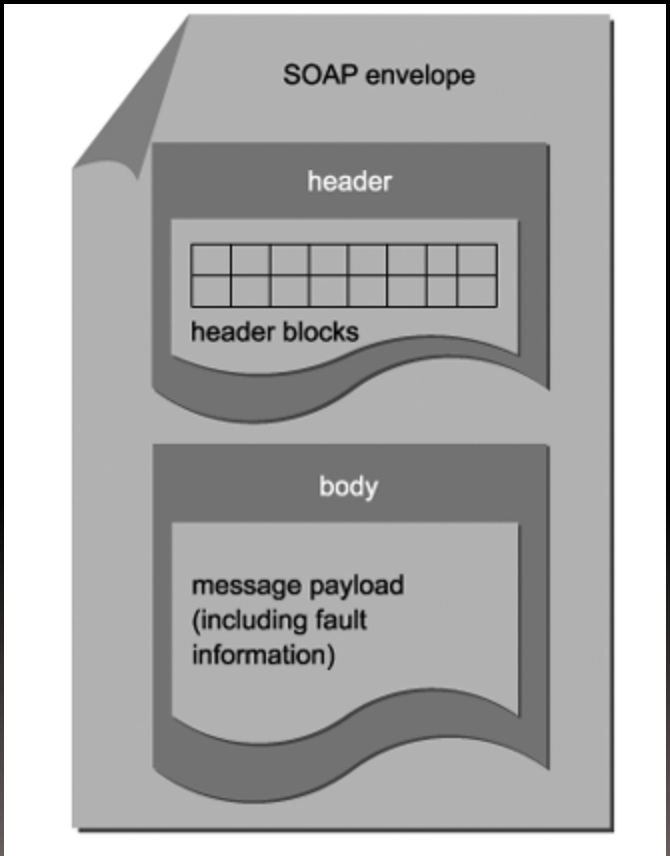
MESSAGING (WITH SOAP)

- within SOAs ,emphasis is placed on a message-centric application design that increases amount of business and application logic is embedded into messages.
- The SOAP specification has been universally accepted as the standard transport protocol for messages processed by Web services.
- A phone book is commonly compared to a service registry, and a mailing address is the equivalent of a WSDL endpoint (or port).
- An envelope represents a standardized medium for transporting mail much like SOAP represents a standardized format for transporting messages.

MESSAGING (WITH SOAP)

Messages

- Simple Object Access Protocol, the SOAP specification's main purpose is to define a standard message format.



The basic structure of a SOAP message

MESSAGING (WITH SOAP)

Envelope, header, and body

- Every SOAP message is packaged into a container known as an envelope.
- The envelope is responsible for housing all parts of the message
- Each message can contain a header, an area dedicated to hosting meta information.
- header section is a vital part of the overall architecture, and though optional, it is rarely omitted.
- Its importance relates to the use of header blocks through which numerous extensions can be implemented.
- The actual message contents are hosted by the message body, which typically consists of XML formatted data.
- The contents of a message body are often referred to as the message payload.

MESSAGING (WITH SOAP)

Header blocks

- A primary characteristic of the SOAP communications framework used by SOAs is an emphasis on creating messages that are as intelligence-heavy and self-sufficient as possible.
- This results in SOAP messages achieving a level of independence that increases the robustness and extensibility of this messaging framework qualities that are extremely important when relying on communication within the loosely coupled environment that Web services require.
- Message independence is implemented through the use of header blocks, packets of supplementary meta information stored in the envelope's header area.
- Header blocks outfit a message with all of the information required for any services with which the message comes in contact to process and route the message in accordance with its accompanying rules, instructions, and properties

MESSAGING (WITH SOAP)

Header blocks

- the use of header blocks, SOAP messages are capable of containing a large variety of supplemental information related to the delivery and processing of message contents.
- This alleviates services from having to store and maintain message-specific logic.
- It further reinforces the characteristics of contemporary SOA related to fostering reuse, interoperability, and composability.
- Web services can be designed with generic processing functionality driven by various types of meta information the service locates in the header blocks of the messages it receives.
- The use of header blocks has elevated the Web services framework to an extensible and composable enterprise-level computing platform. Practically all WS-* extensions are implemented using header blocks.

MESSAGING (WITH SOAP)

Header blocks

Examples of the types of features a message can be outfitted with using header blocks include:

1. processing instructions that may be executed by service intermediaries or the ultimate receiver
2. routing or workflow information associated with the message
3. security measures implemented in the message
4. reliability rules related to the delivery of the message
5. context and transaction management information
6. correlation information (typically an identifier used to associate a request message with a response message)

- SOAP allows the recognition and processing of header blocks to be marked as optional.
- This way messages can be safely outfitted with header blocks that implement non-critical features from newer extensions.

MESSAGING (WITH SOAP)

Message styles

- The SOAP specification was originally designed to replace proprietary RPC protocols by allowing calls between distributed components to be serialized into XML documents, transported, and then deserialized into the native component format upon arrival.
- As a result, much in the original version of this specification centered around the structuring of messages to accommodate RPC data.
- This RPC-style message runs contrary to the emphasis SOA places on independent, intelligence-heavy messages.
- SOA relies on document-style messages to enable larger payloads, coarser interface operations, and reduced message transmission volumes between services.

MESSAGING (WITH SOAP)

Attachments

- To facilitate requirements for the delivery of data not so easily formatted into an XML document, the use of SOAP attachment technologies exist.
- Each provides a different encoding mechanism used to bundle data in its native format with a SOAP message.
- SOAP attachments are commonly employed to transport binary files, such as images.

- Example:

purchase orders are not allowed to be issued in the standard electronic format, as the signature is required to be an ever-present part of the document.

MESSAGING (WITH SOAP)

Faults

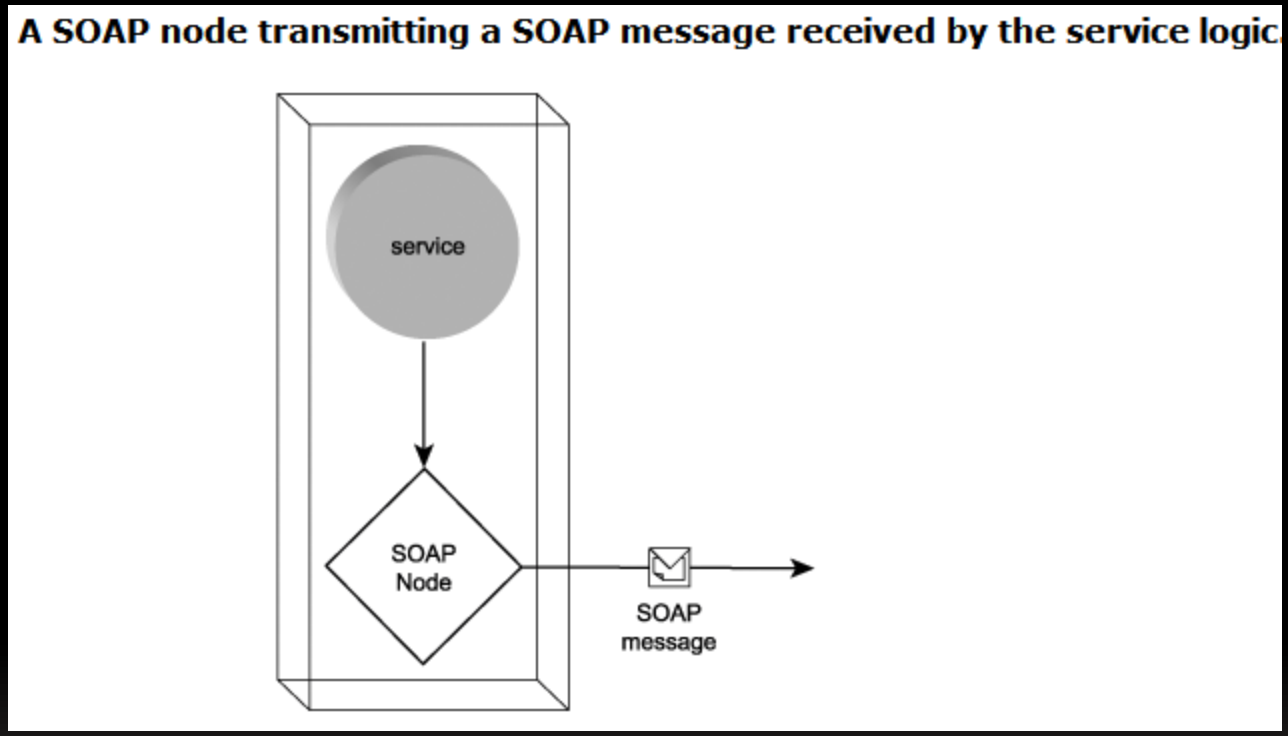
- SOAP messages offer the ability to add exception handling logic by providing an optional fault section that can reside within the body area.
- The typical use for this section is to store a simple message used to deliver error condition information when an exception occurs.

Nodes

- Although Web services exist as self-contained units of processing logic, they are reliant upon a physical communications infrastructure to process and manage the exchange of SOAP messages.
- Every major platform has its own implementation of a SOAP communications server, and as a result each vendor has labeled its own variation of this piece of software differently.
- In abstract, the programs that services use to transmit and receive SOAP messages are referred to as SOAP nodes

MESSAGING (WITH SOAP)

Nodes



- SOAP message sent by the SOAP node from service A can be received and processed by a SOAP node (supporting the same version of the SOAP standard) from any other service.

MESSAGING (WITH SOAP)

Node types

- As with the services that use, the underlying SOAP nodes are given labels that identify their type, depending on what form of processing they are involved with in a given message processing scenario.
- Below is a list of type labels associated with SOAP nodes (in accordance with the standard SOAP Processing Model).
- Notice that these names are very similar to the Web service roles. The SOAP specification has a different use for the term "role" and instead refers to these SOAP types or labels as concepts.
  - SOAP sender- a SOAP node that transmits a message
  - SOAP receiver- a SOAP node that receives a message
  - SOAP intermediary- a SOAP node that receives and transmits a message, and optionally processes the message prior to transmission
  - initial SOAP sender -the first SOAP node to transmit a message
  - ultimate SOAP receiver-the last SOAP node to receive a message

MESSAGING (WITH SOAP)

SOAP intermediaries

- The same way service intermediaries transition through service provider and service requestor roles, SOAP intermediary nodes move through SOAP receiver and SOAP sender types when processing a message
- SOAP nodes acting as intermediaries can be classified as forwarding or active.
- When a SOAP node acts as a forwarding intermediary, it is responsible for relaying the contents of a message to a subsequent SOAP node. In doing so, the intermediary will often process and alter header block information relating to the forwarding logic it is executing.
- For example, it will remove a header block it has processed, as well as any header blocks that cannot be relayed any further.

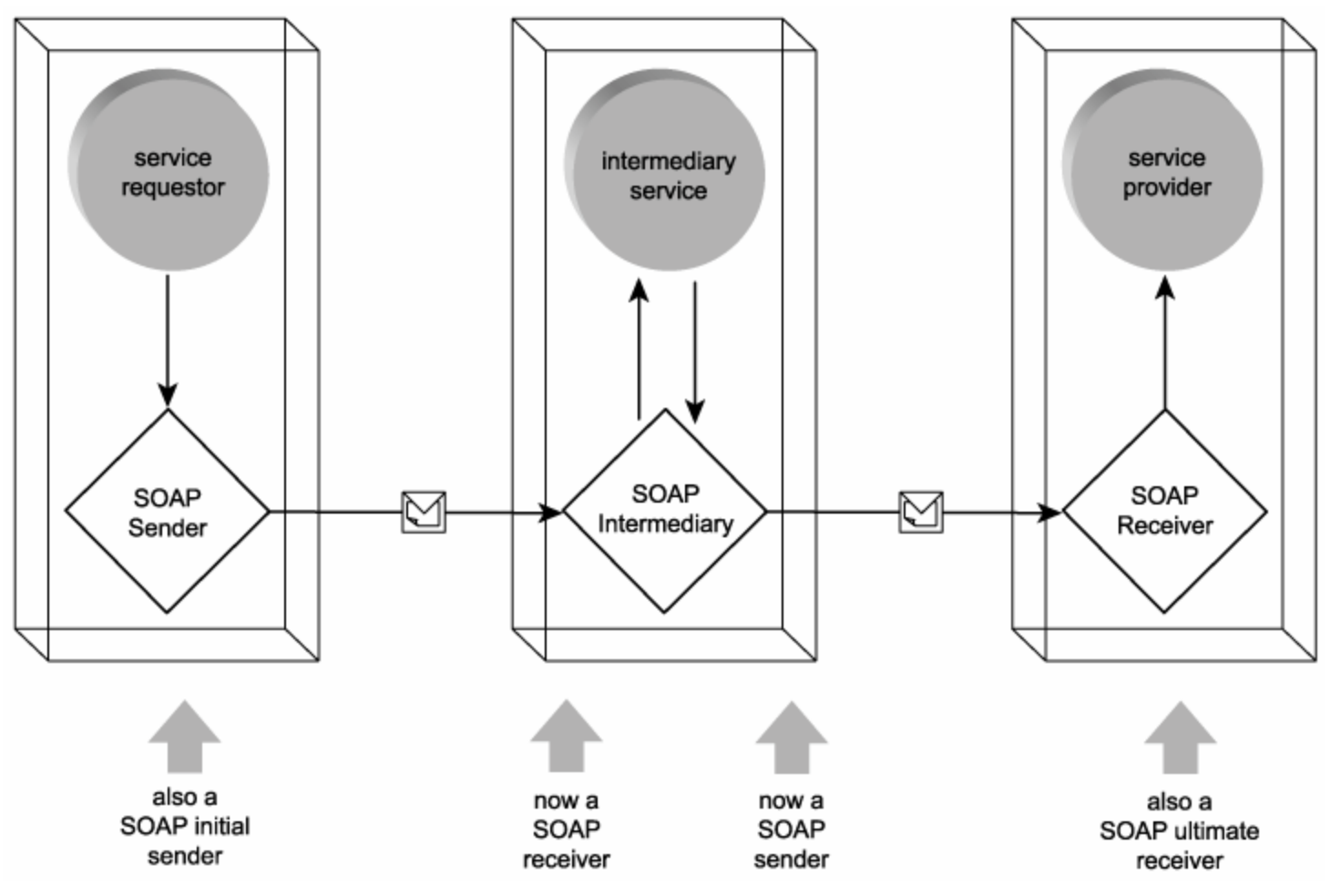
MESSAGING (WITH SOAP)

SOAP intermediaries

- Active intermediary nodes are distinguished by the type of processing they perform above and beyond forwarding-related functions.
- An active intermediary is not required to limit its processing logic to the rules and instructions provided in the header blocks of a message it receives.
- It can alter existing header blocks, insert new ones, and execute a variety of supporting actions.

MESSAGING (WITH SOAP)

SOAP intermediaries



Different types of SOAP nodes involved with processing a message.

MESSAGING (WITH SOAP)

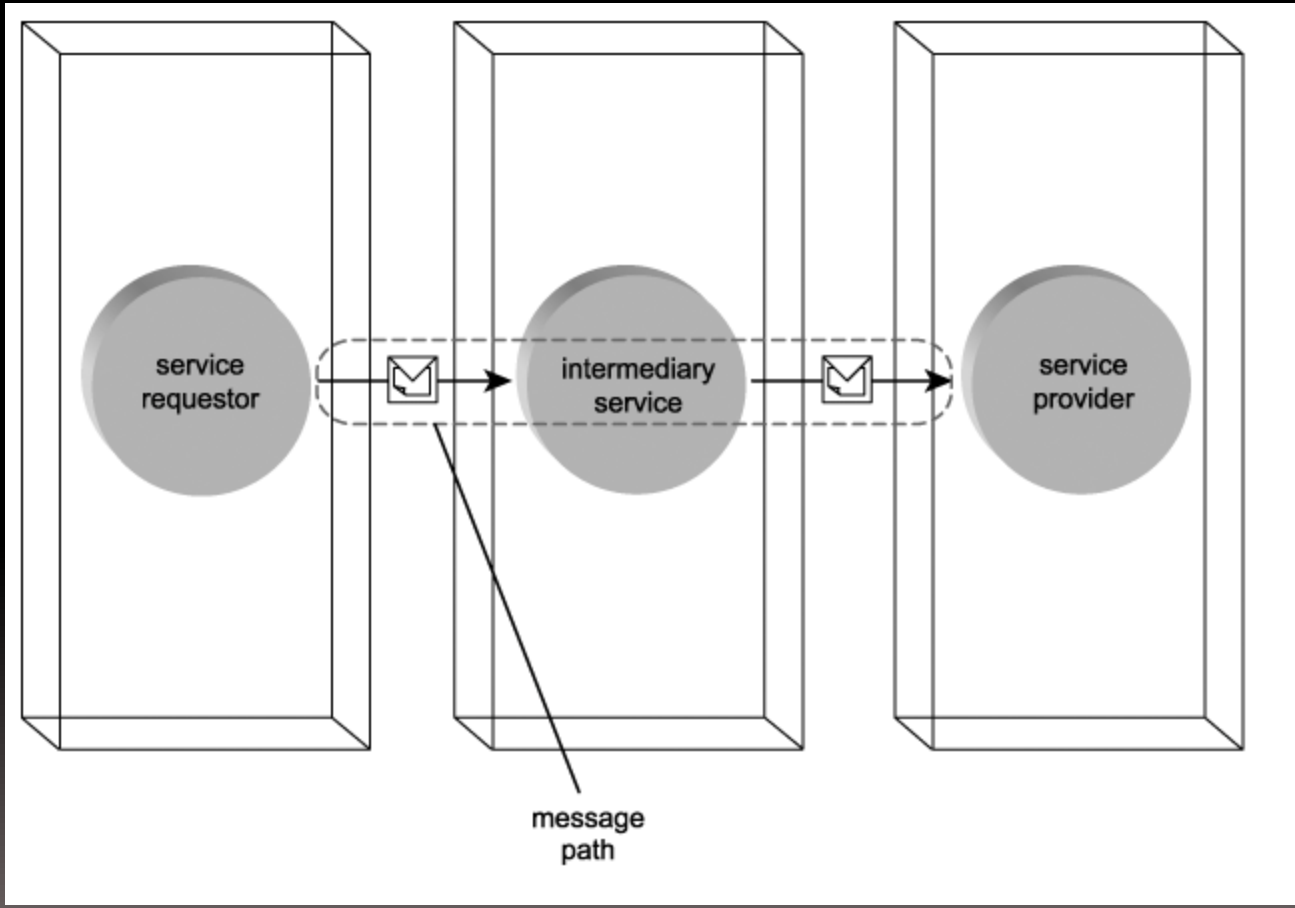
Message paths

- A message path refers to the route taken by a message from when it is first sent until it arrives at its ultimate destination.
- A message path consists of at least one initial sender, one ultimate receiver, and zero or more intermediaries .
- Mapping and modeling message paths becomes an increasingly important exercise in SOAs, as the amount of intermediary services tends to grow along with the expansion of a service-oriented solution.
- Design considerations relating to the path a message is required to travel often center around performance, security, context management, and reliable messaging concerns
- A message path is sometimes not predetermined.
- The use of header blocks processed by intermediaries can dynamically determine the path of a message. This may be the result of routing logic, workflow logic, or environmental conditions

MESSAGING (WITH SOAP)

Message paths

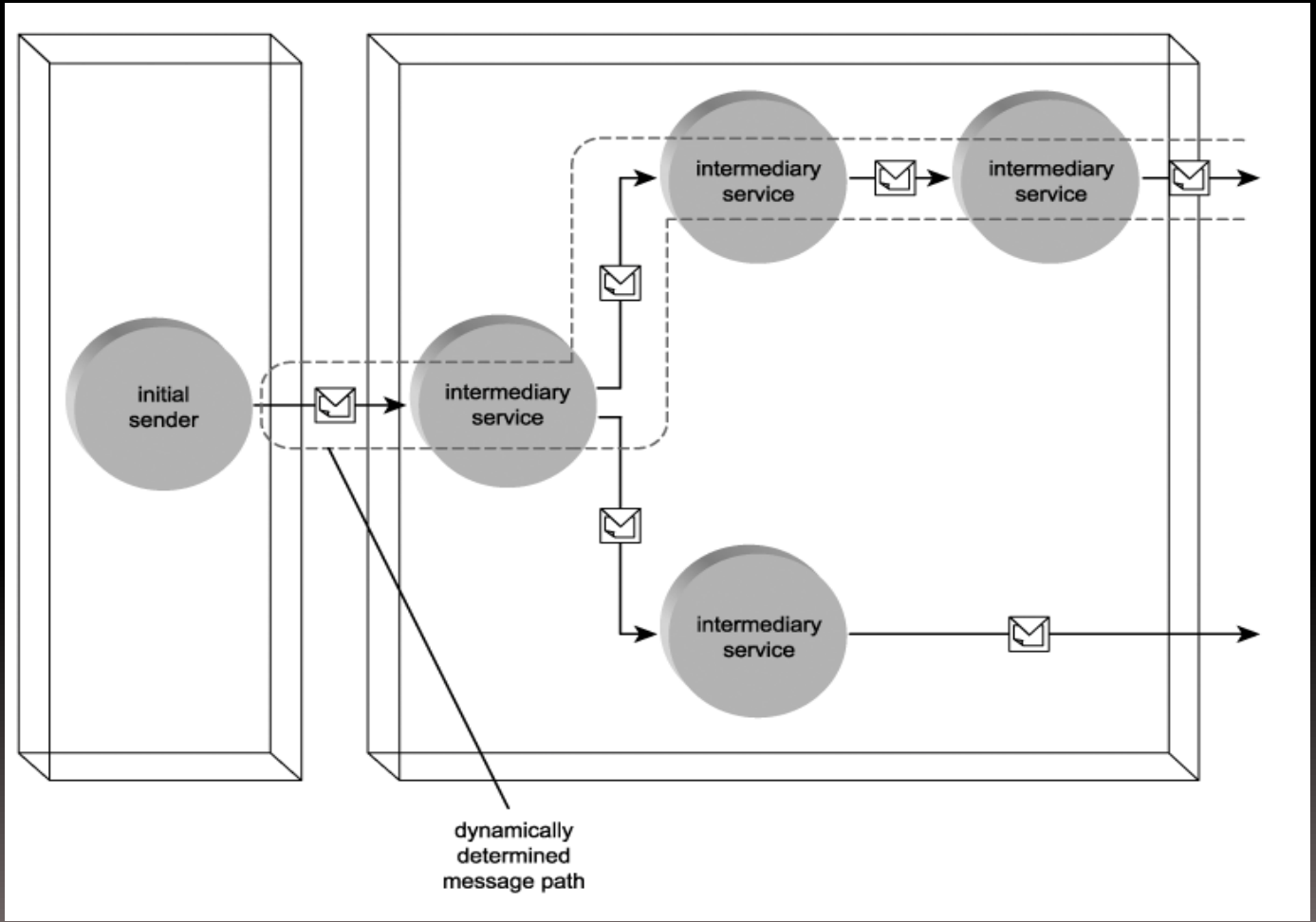
A message path consisting of three Web services



MESSAGING (WITH SOAP)

Message paths

A message path determined at runtime



MESSAGING (WITH SOAP)

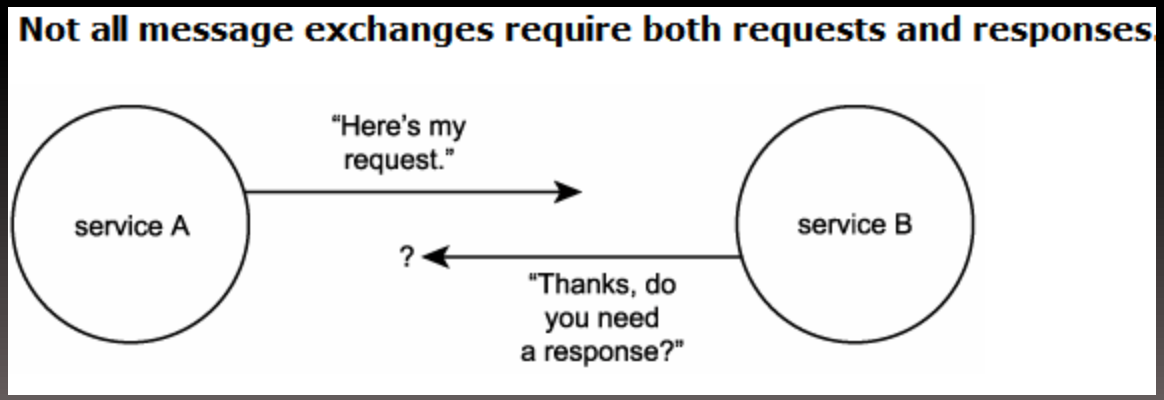
Message paths

- When used within the context of SOAP nodes, it is referred to as a SOAP message path.
- While a message path in abstract can be purely logical, the SOAP node perspective is always focused on the actual physical transport route.
- A SOAP message path is comprised of a series of SOAP nodes, beginning with the initial SOAP sender and ending with the ultimate SOAP receiver.
- Every node refers to a physical installation of SOAP software, each with its own physical address

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Message exchange patterns (MEP)

- 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



Message exchange patterns (MEP)

- 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 no doubt be finding while applying MEPs to specific communication requirements when designing service-oriented solutions.
- An MEP is like a type of conversation. It's not a long conversation; it actually only covers one exchange between two parties

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Primitive MEPs

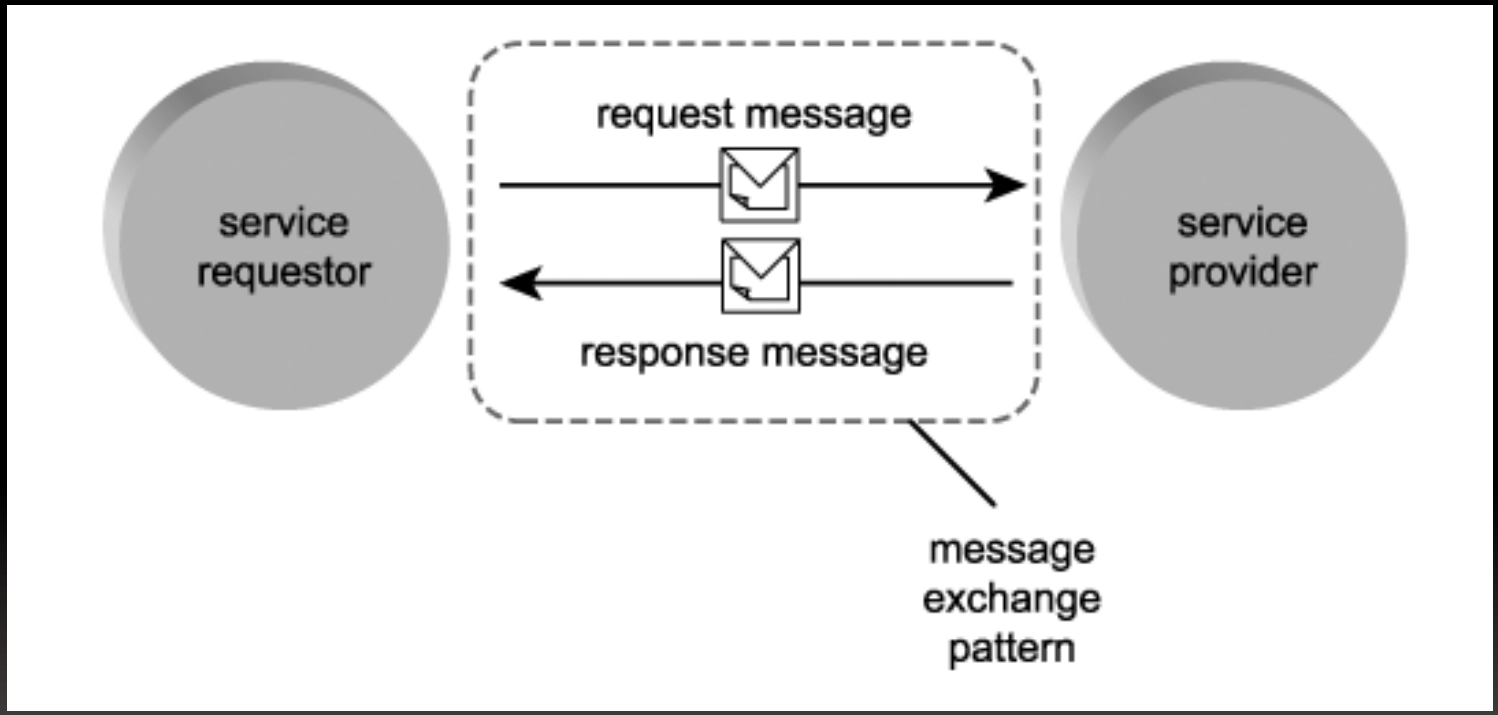
- Before the arrival of contemporary SOA, messaging frameworks were already well used by various messaging-oriented middleware products. As a result, a common set of primitive MEPs has been in existence for some time.

Request-response MEP

- 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).

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

The request-response MEP



Fire-and-forget MEP

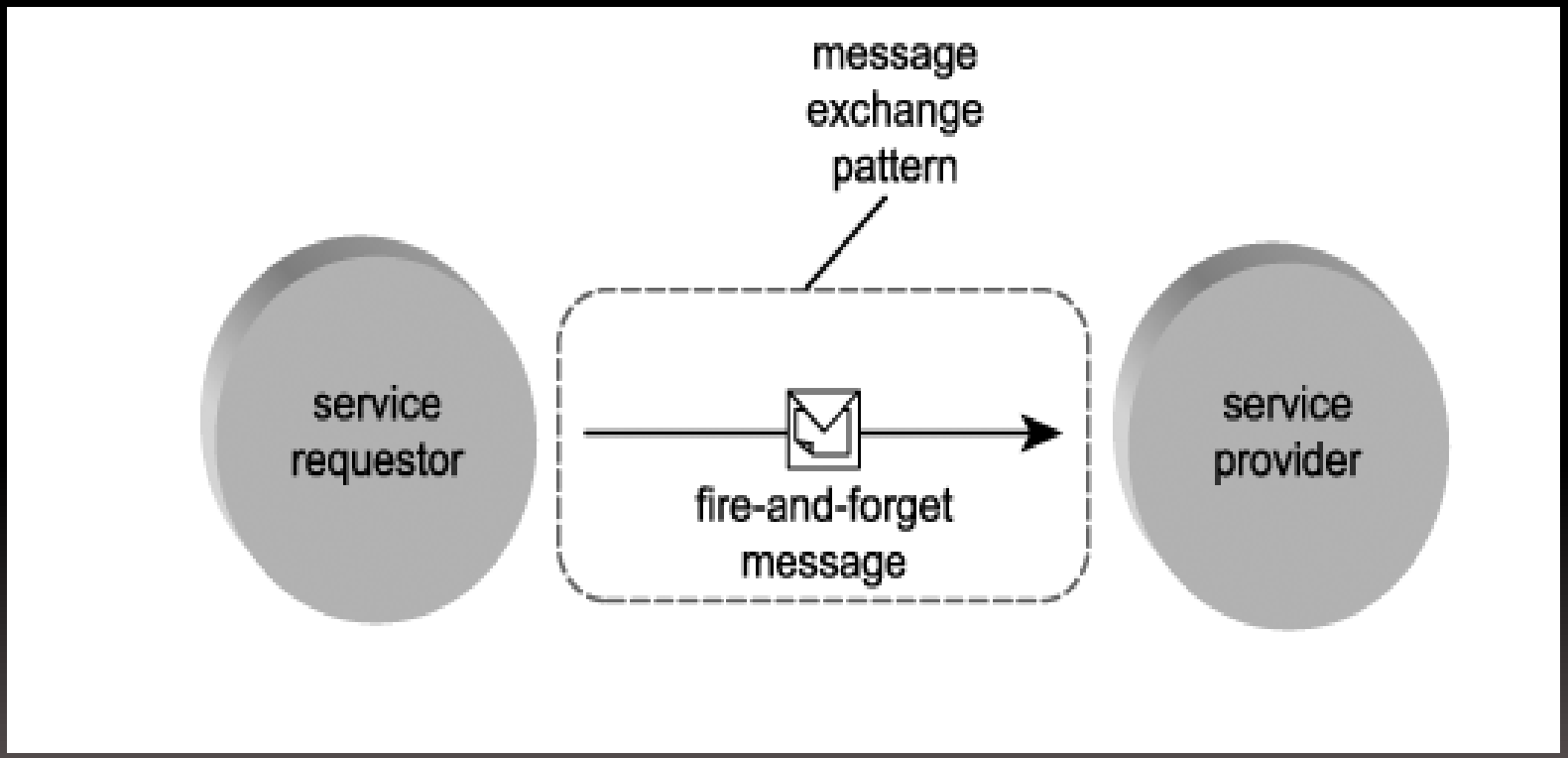
- simple asynchronous pattern is based on the unidirectional transmission of messages from a source to one or more destinations

Variations of the fire-and-forget :

- 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

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Fire-and-forget MEP



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.
- 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.

Complex MEPs

The steps involved :

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.

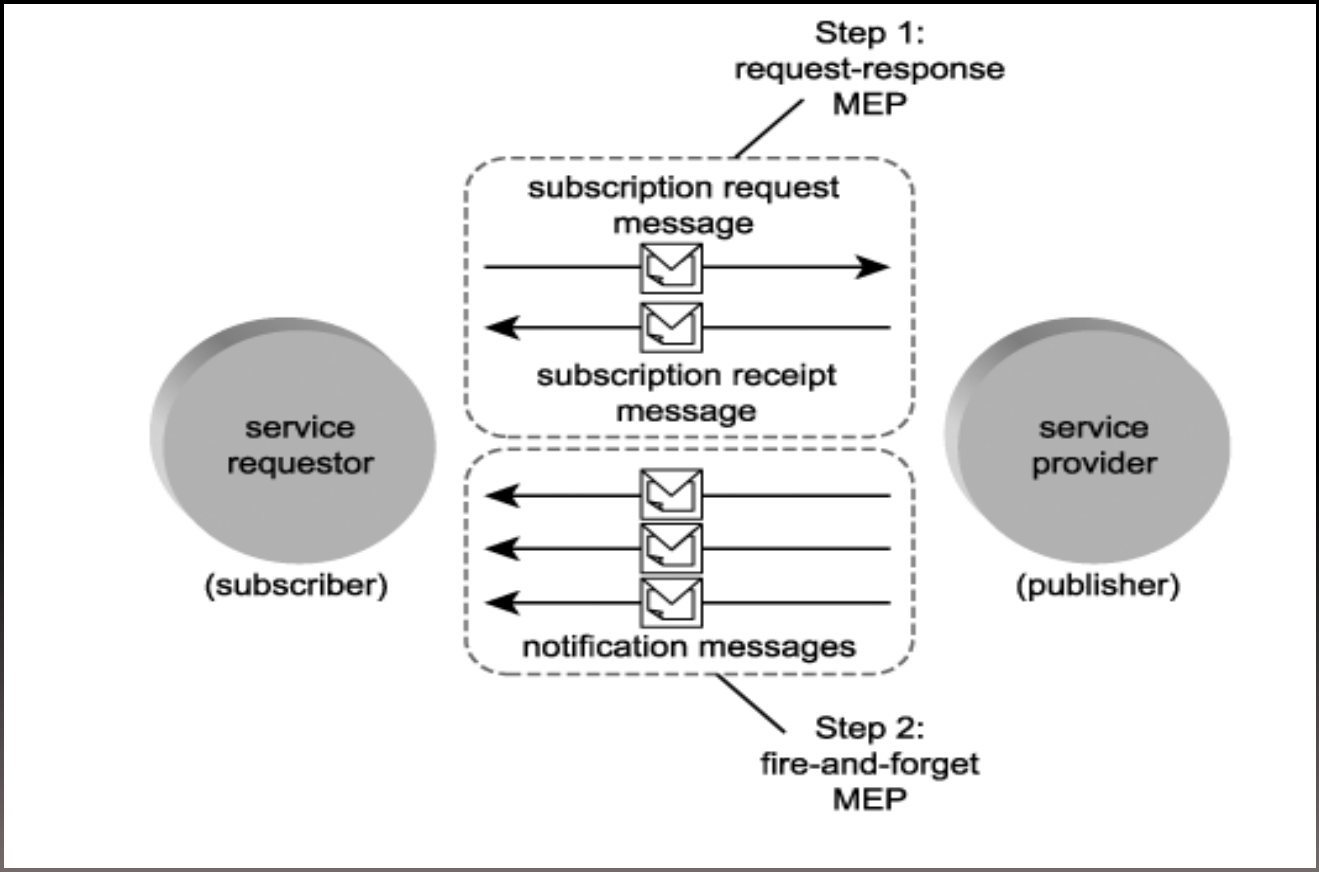
Complex MEPs

- This pattern is a great example of how to aggregate primitive MEPs
- 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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Complex MEPs

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



WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Complex MEPs

WS-* specifications that incorporate this messaging model include:

- WS-BaseNotification
- WS-BrokeredNotification
- WS-Topics
- WS-Eventing

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

MEPs and WSDL

- Release 1.1 of the WSDL specification provides support for four message exchange patterns that roughly correspond to the MEPs. 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 :

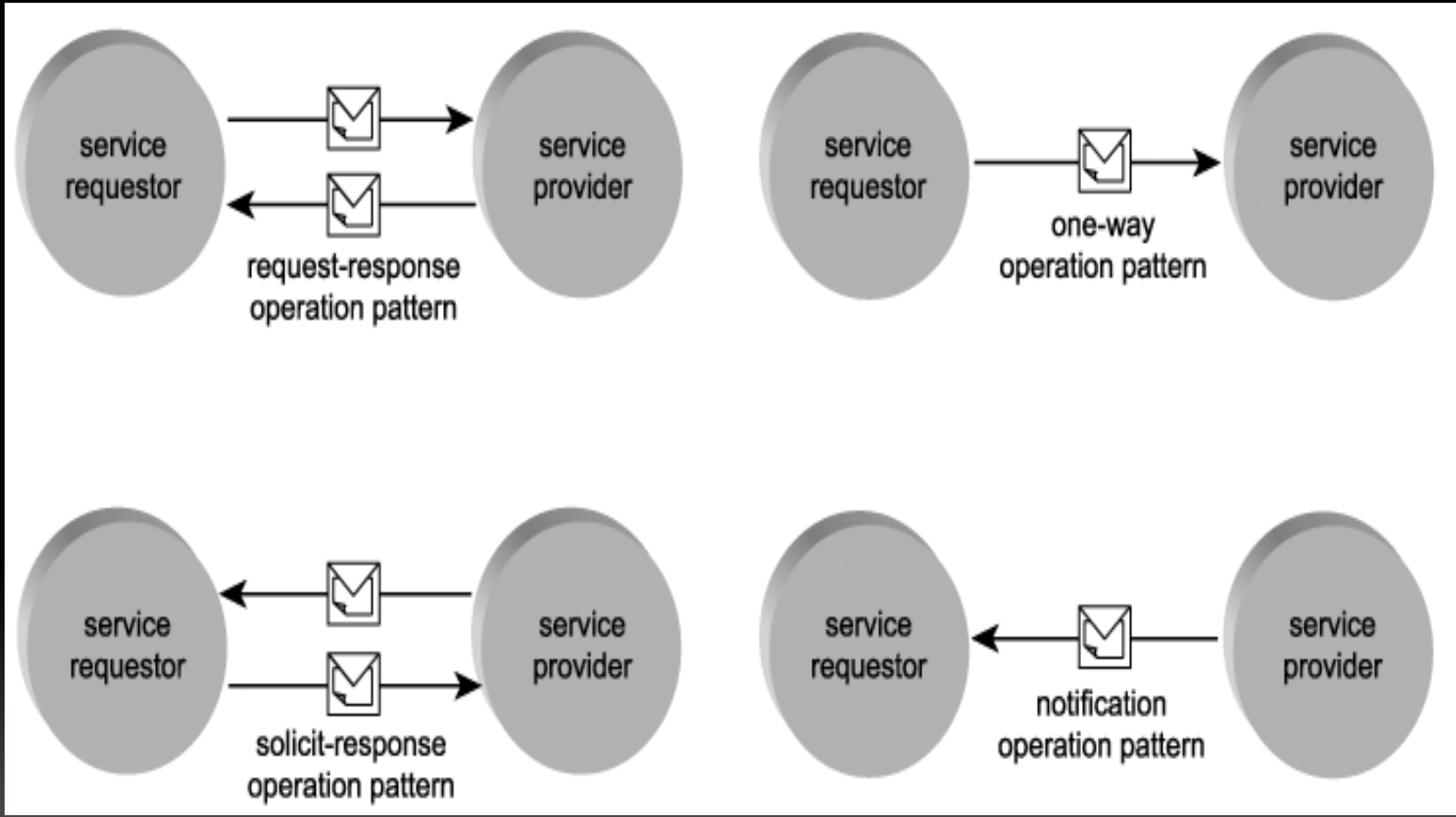
- The service expects a single message and is not obligated to respond.

Notification operation:

- The service sends a message and expects no response.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

The four basic patterns supported by WSDL 1.1



WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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.)

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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

which is the reverse of the in-optional-out pattern, where the incoming message is optional. Fault message generation is again supported.

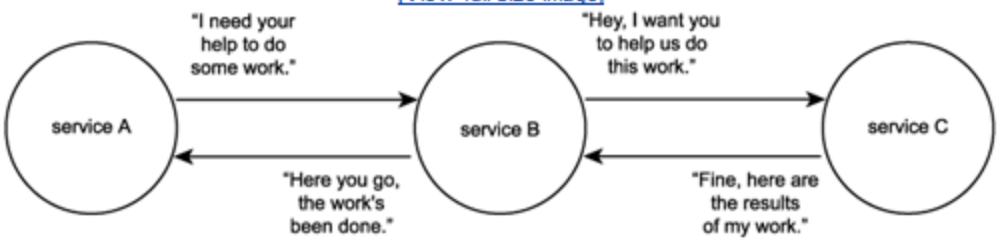
WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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.

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

[View full size image]



WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

In Plain English

Any type of task will consist of one or more steps. The task of turning on the TV is relatively simple, consisting of perhaps the following two steps:

1. Pick up the remote control.
2. Press the "Power" button.

The task of washing a car, on the other hand, can be a bit more omplicated. It could exist of a series of steps, including:

1.Locate bucket.

2. Locate sponge.

3. Locate hose.

4. Fill bucket with warm water.

5. Add soap to water.

6. Soak sponge in water.

7.Rub sponge on car....and so on.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

The steps that comprise this more complex task could be summarized into a series of simple (or primitive) tasks, as follows:

1. Gather required equipment.
2. Prepare water.
3. Wash car.

- Each simple task consists of a smaller number of steps.
- Collectively these simple tasks represent a larger, logical unit of work.
- Individually, simple tasks do not accomplish anything of relevance, primarily because each subsequent task is dependent on the completion of the former. It is only when they are assembled into a complex task that they represent a useful unit of work.

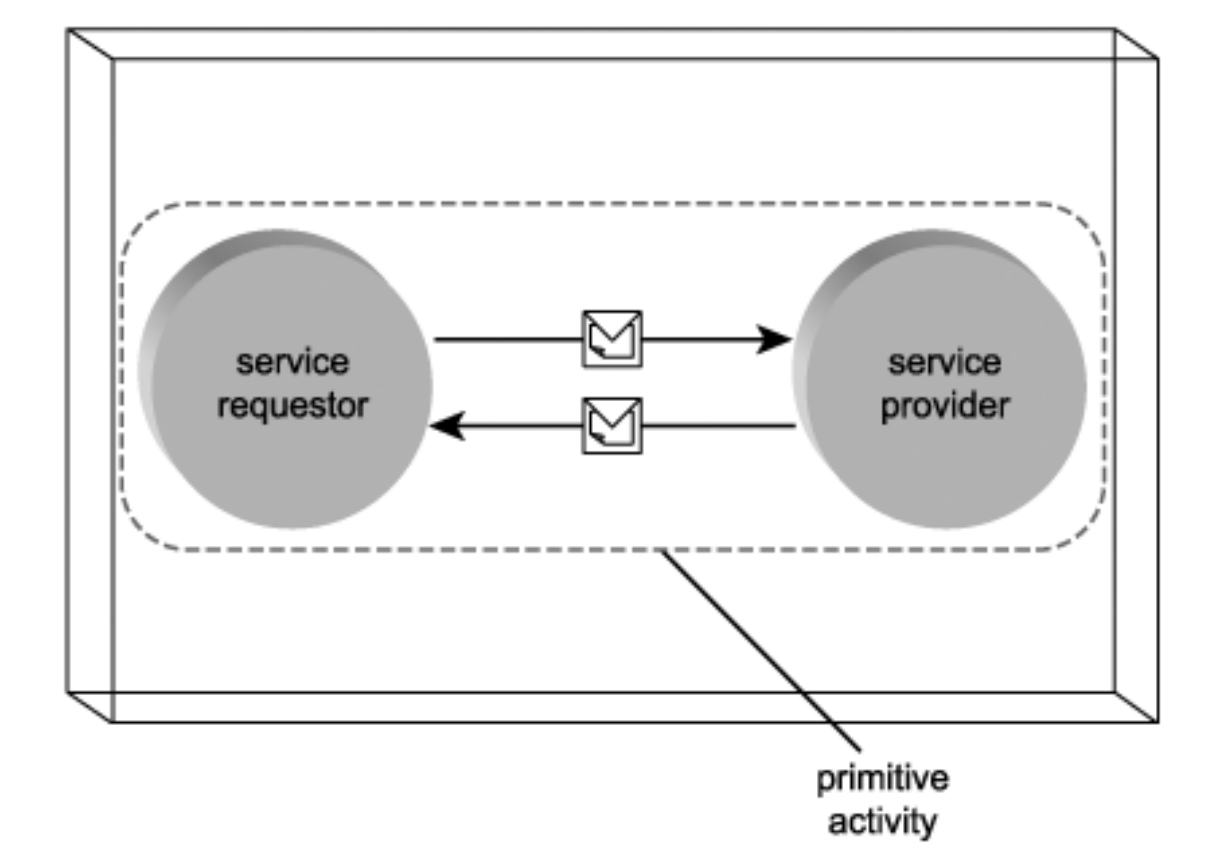
WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Primitive and complex service activities

- The scope of an activity can drastically vary.
- A simple or primitive activity is typified by synchronous communication and 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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

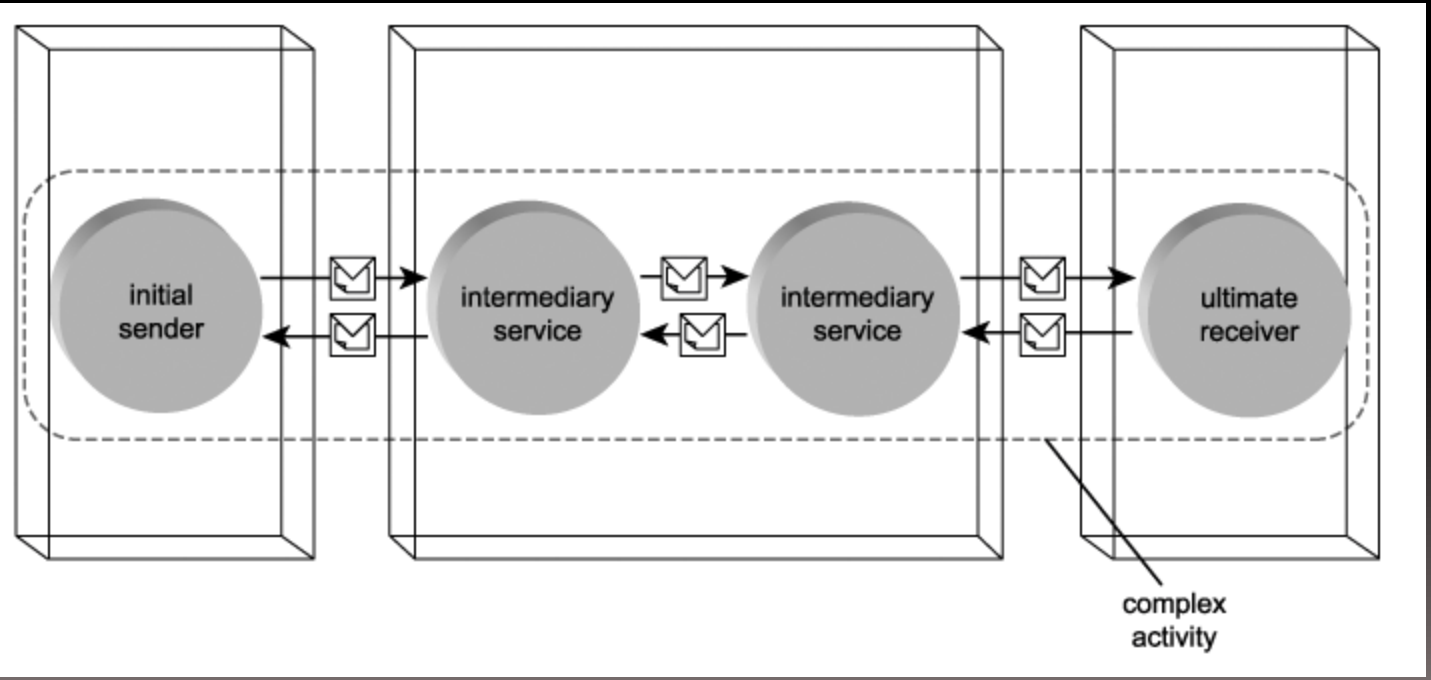
A primitive service activity consisting of a simple MEP



WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

- Complex activities 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.

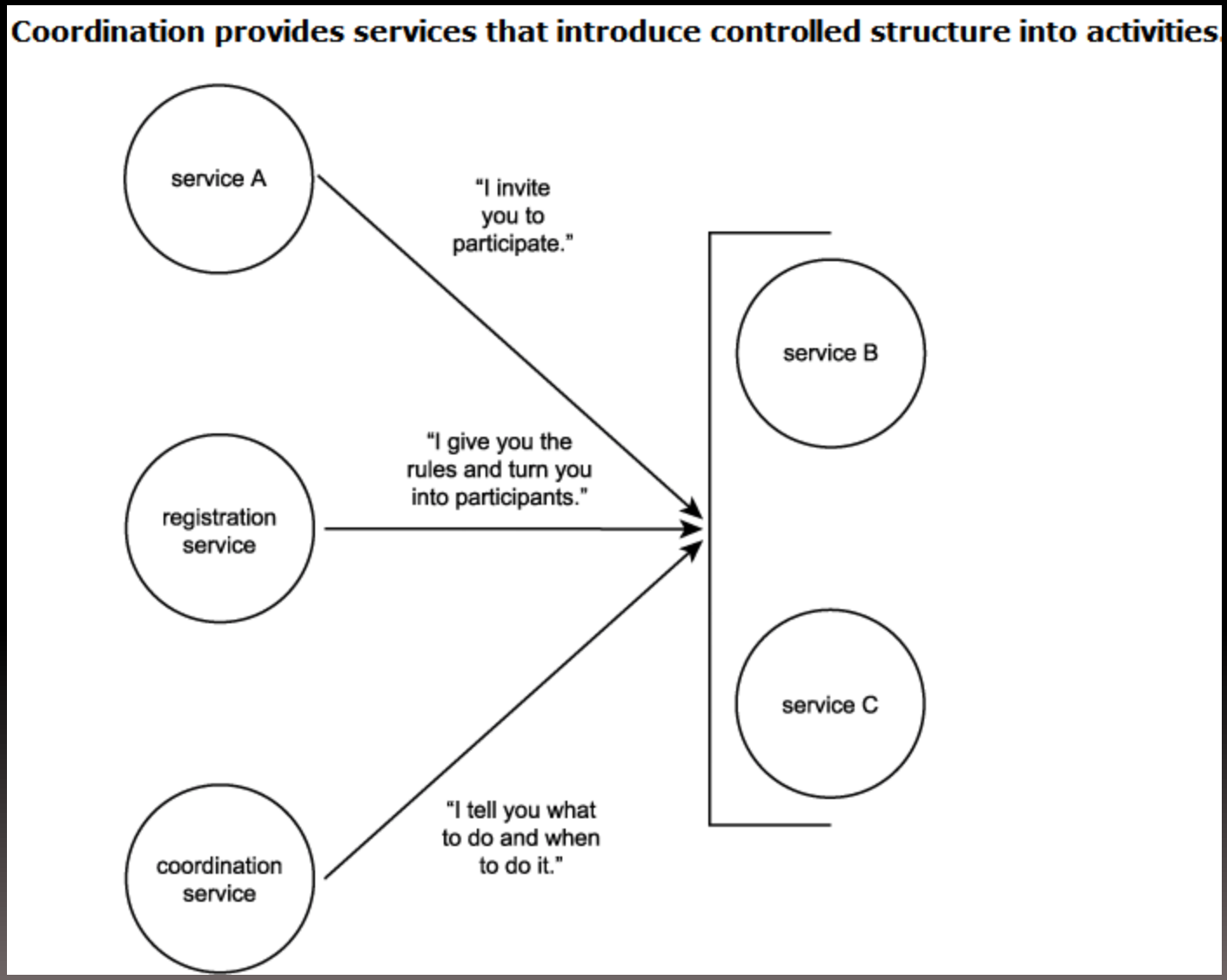
A complex activity involving four services



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

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)



In Plain English

I decide to get Chuck, Bob, and Jim to wash my car. When we're all ready to go, I assign each of them a simple task, as follows:

ChuckGather equipment.

BobPrepare the water.

JimWash the car.

You might recall each of these simple tasks consists of a series of steps. Chuck's task, for instance, is comprised of the following steps:

1. Locate bucket.
2. Locate sponge.
3. Locate hose.

Completion of the first step is required before Bob can begin his task of preparing the water. Completion of Chuck's and Bob's tasks is required for Jim to start washing the car. However, allowing Bob to start on his task after Chuck completes only the first step of his task will allow both Chuck and Bob to complete their respective tasks at around the same time. The benefit here is that Jim can start his task sooner than if Bob had to wait for Chuck to fully complete all steps in his task.

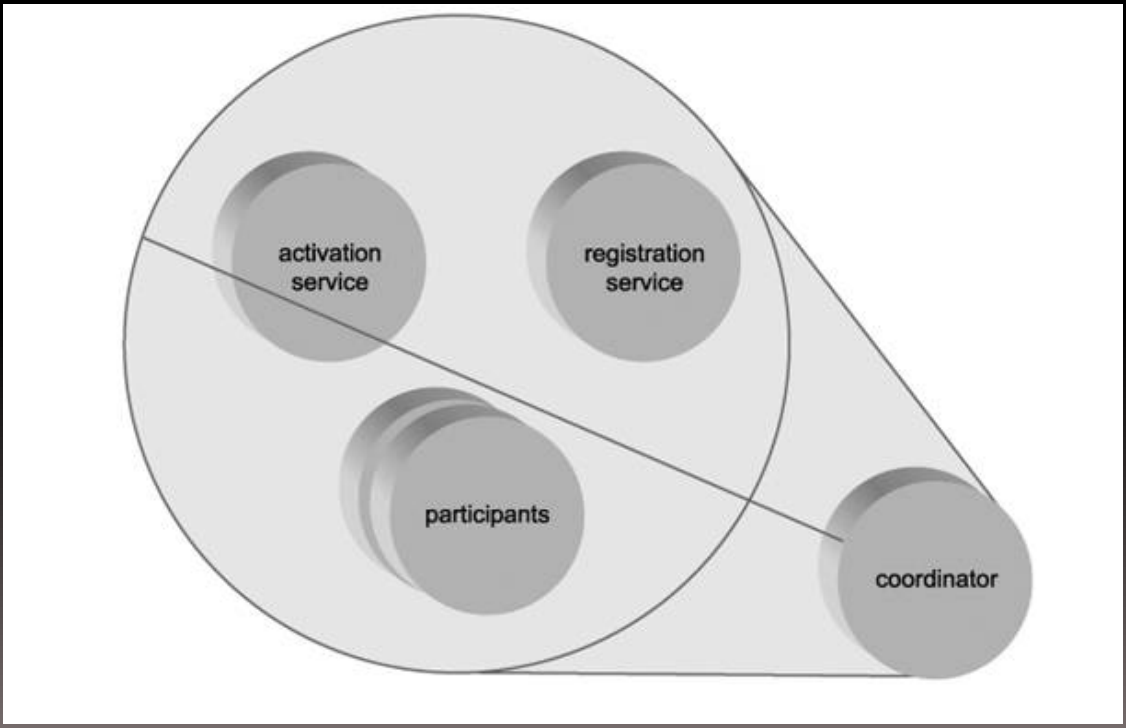
To coordinate the complex activity of car washing in an efficient manner, Chuck needs to communicate to Bob when he has located the bucket. Because Bob may not be immediately available and because Chuck doesn't want to go looking for Bob to hand over the bucket, Chuck tells me that the bucket is ready. Chuck then continues with his other work, and when I see Bob next, I can relay the status of the bucket availability to him.

In this scenario, the bucket availability status is considered context information that I managed. Because a separate context manager (me) was in place, Chuck was alleviated of the responsibility of remembering and communicating the context information to Bob. This freed Chuck to continue with his other work. It also spared Bob from having to directly locate and communicate with Chuck to get the context information. Finally, my knowledge of who was doing what in this car washing process also would be classified as context information.

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:

1. Activation service:
   - Responsible for the creation of a new context and for associating this context to a particular activity.
2. Registration service:
   - Allows participating services to use context information received from the activation service to register for a supported context protocol.
3. Protocol-specific services:
   - These services represent the protocols supported by the coordinator's coordination type.
4. Coordinator :
   - The controller service of this composition, also known as the coordination service.

Coordinator composition

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.
- The two coordination types most commonly associated with WS-Coordination are WS-AtomicTransaction and WS-BusinessActivity.
- 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.

Coordinator composition

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 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.

Coordinator composition

The activation and registration process

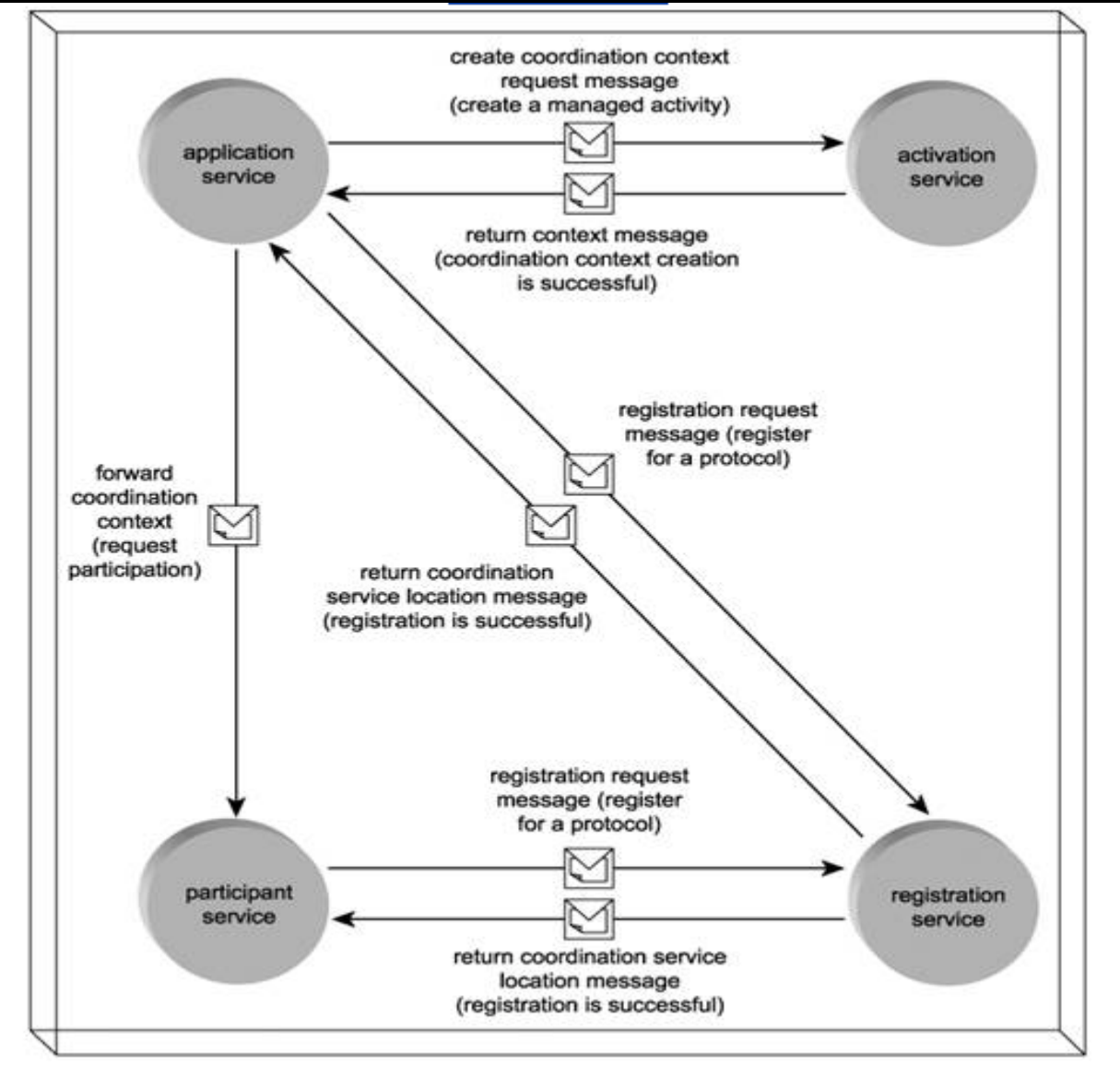
- The coordination service composition is instantiated when an application service contacts the activation service.
- Via a CreateCoordinationContext request message, it asks the activation service to generate a set of new context data.
- Once passed back with the ReturnContext message, the application 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.

Coordinator composition

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.
- 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.

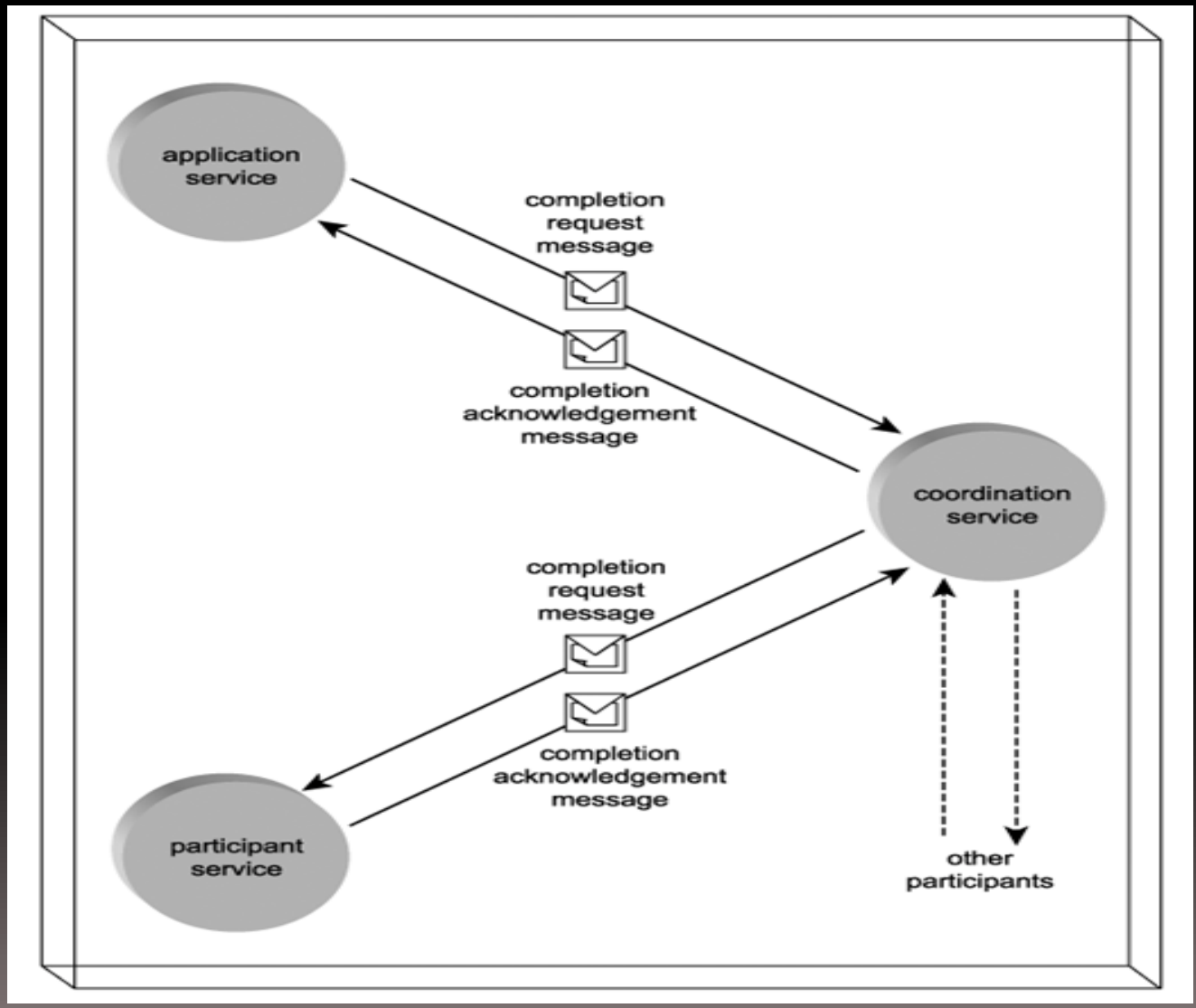
The WS-Coordination registration process



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 .

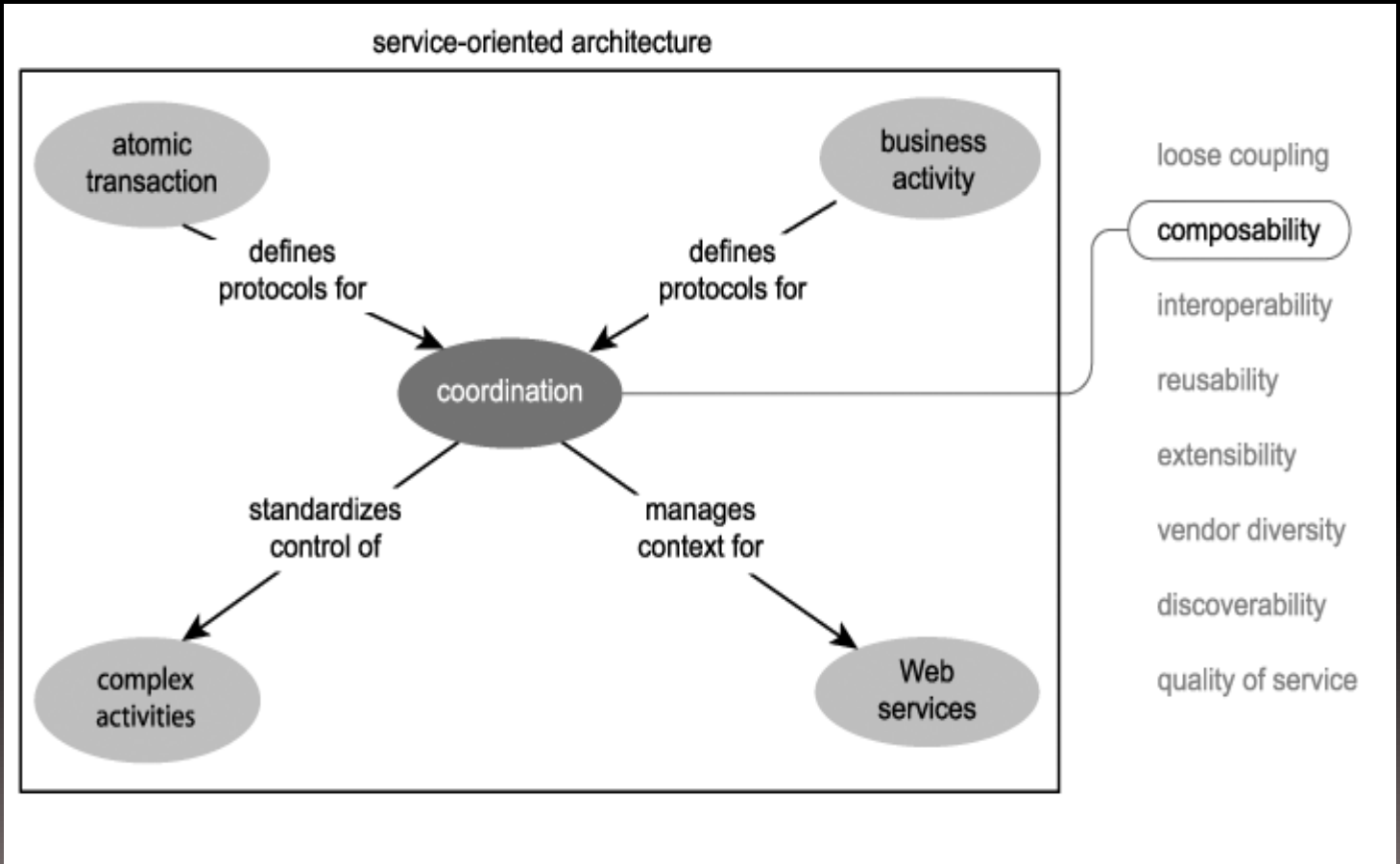
- It standardizes the management and interchange of context information within a variety of key business protocols.

- 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.

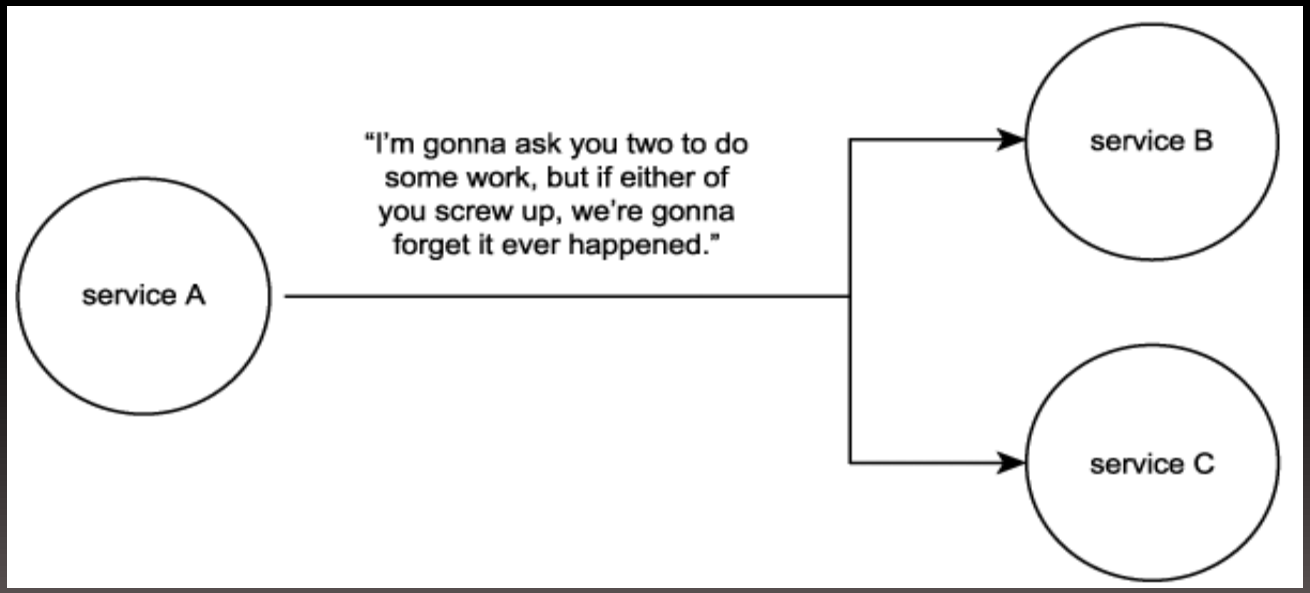
Coordination and SOA

Coordination as it relates to other parts of SOA.



Atomic transactions

- 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



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

In Plain English

While we were washing my car, a neighbor stops by and offers us some money to wash his car. Having nothing else planned for the afternoon, we oblige and apply the same car washing process to the neighbor's car.

In the following weeks, word of our car washing abilities gets around, and others from the neighborhood start requesting our services. We soon find ourselves washing four or five cars every weekend.

We then come across a car with a special metallic finish. Not knowing any better, we proceed to apply our standard car washing process. When the car dries, though, we notice spots of discoloration throughout its exterior. A subsequent investigation leads us to find out that the soap we've been using is not suitable for some finishes (including metallic paint).

This turns into an expensive lesson, as we subsequently fund a new paint job. To prevent this from happening again, we decide to take measures. We proceed to purchase some specialized soaps for use in our water. The choice of soap is dependent on the finish of the car we are washing.

Sometimes the use of these new soaps requires us to carefully mix two or more cleaning solutions together. We determine a correct mixture by assessing the resulting color of the water. However, because we tend to eyeball this process, it can sometimes go wrong. We agree that if the correct color is not attained, we empty the contents of the bucket and start again.

This change to our process affects the following two steps:

4. Fill bucket with warm water.

5. Add soap to water.

Originally, these steps were simply performed in sequence as a continuation of the overall process. Now we have a requirement that dictates that should the resulting soap mixture be unacceptable, the bucket needs to be reset to its original state (empty). This requirement emulates an atomic transaction, where at the completion of Step 5, the process is either rolled back to the beginning of Step 4, or the quality of water is accepted (committed) so that it can be applied to washing the car.

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.
- 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.

ACID transactions

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, is an extension created for use with the WS-Coordination context management framework.
- 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.

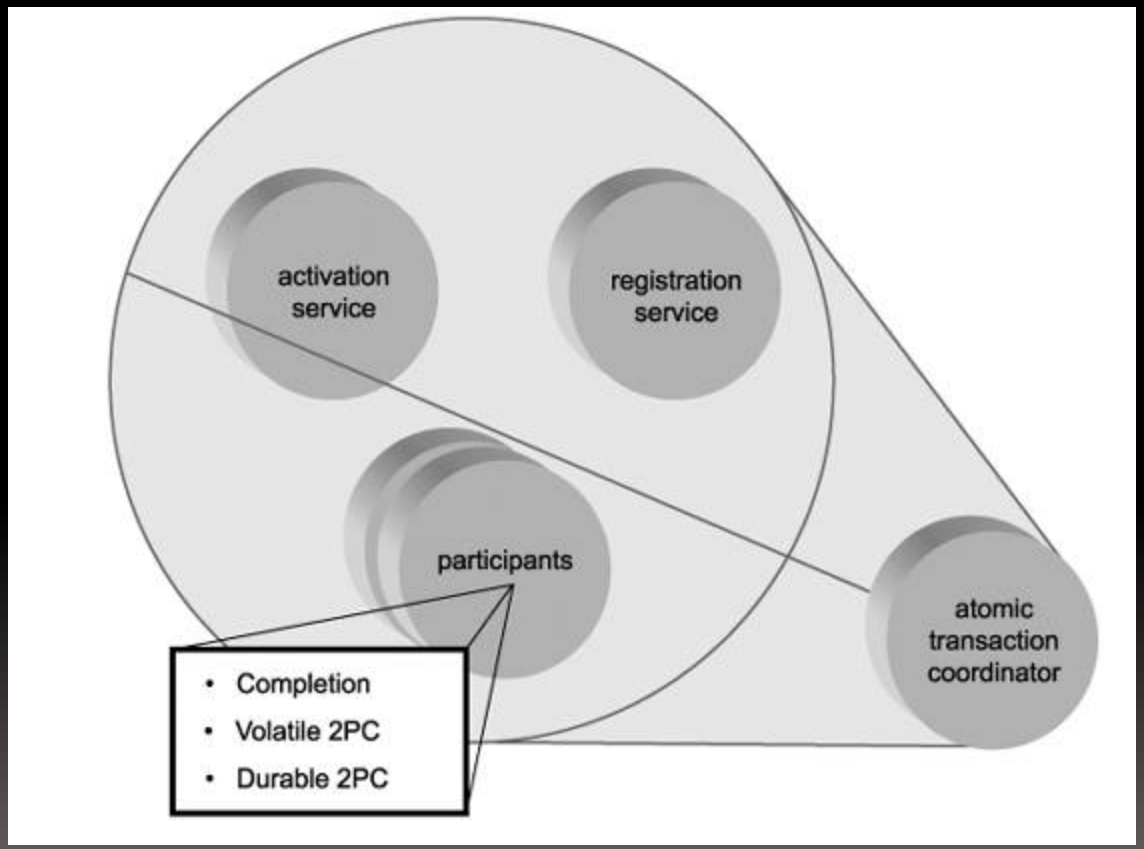
- Most often these protocols are used to enable a two-phase commit (2PC) that manages an atomic transaction across multiple service participants.

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

The atomic transaction coordinator service model



The atomic transaction process

- 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
- 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 atomic transaction process

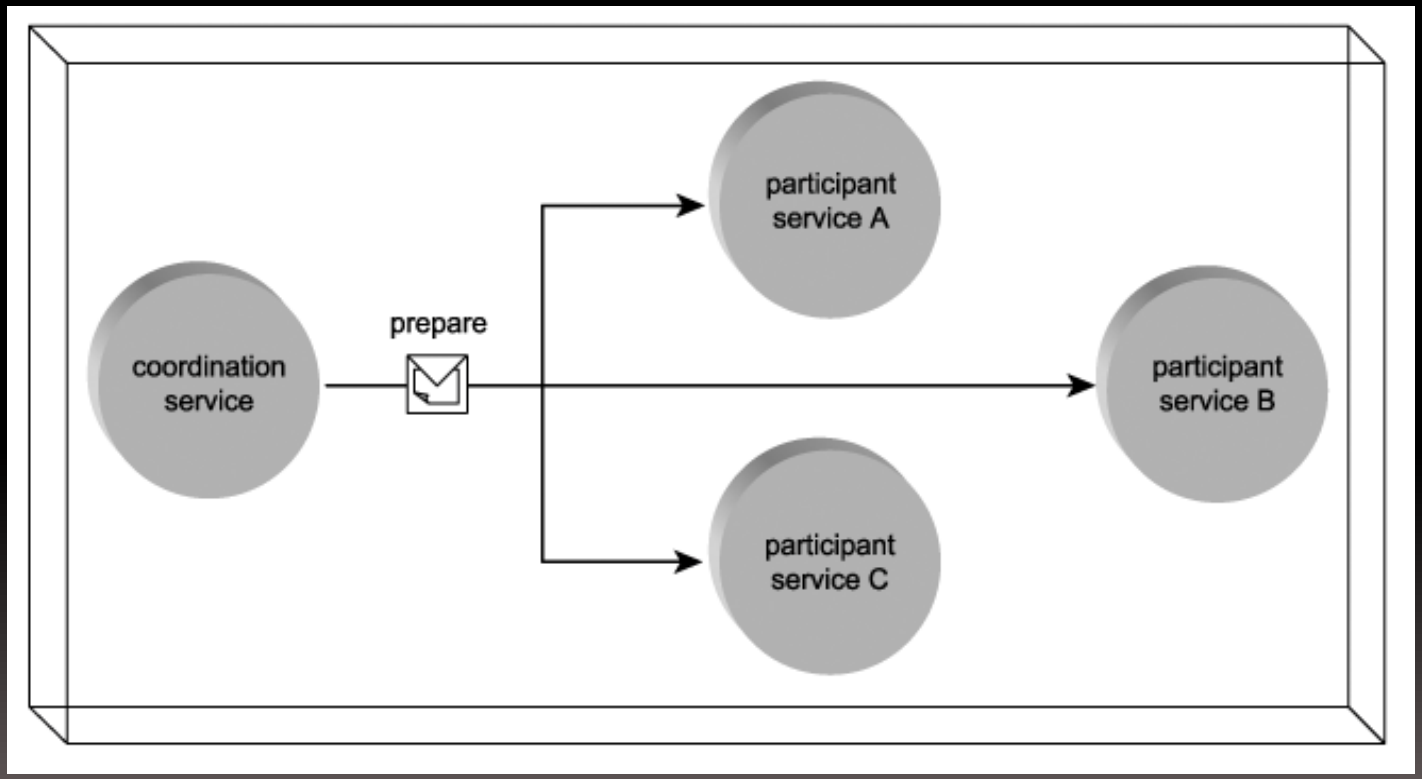
The conditions of a commit decision are simple:

- if all votes are received and if all participants voted to commit, the coordinator declares the transaction successful, and the changes are committed.

- 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

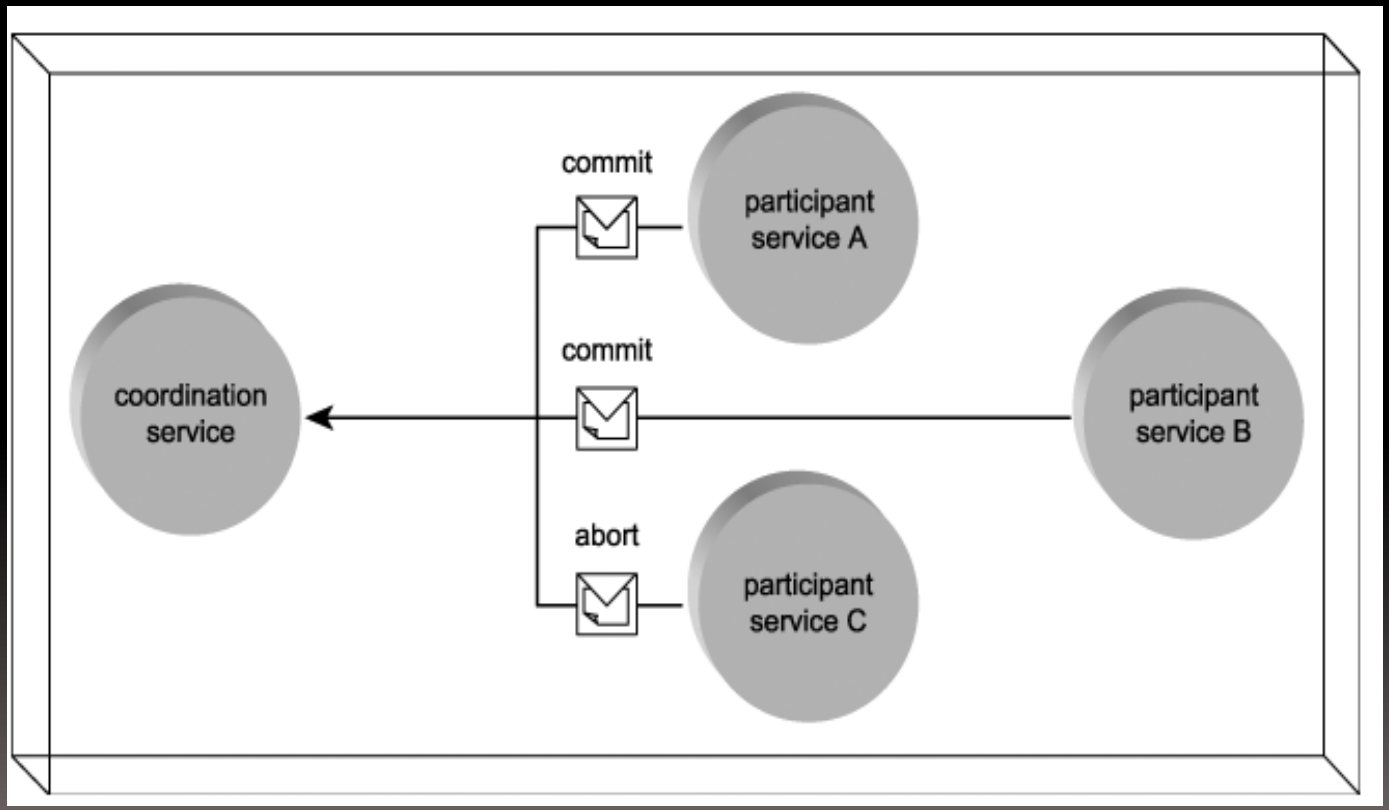
The atomic transaction process

The coordinator requesting that transaction participants prepare to vote



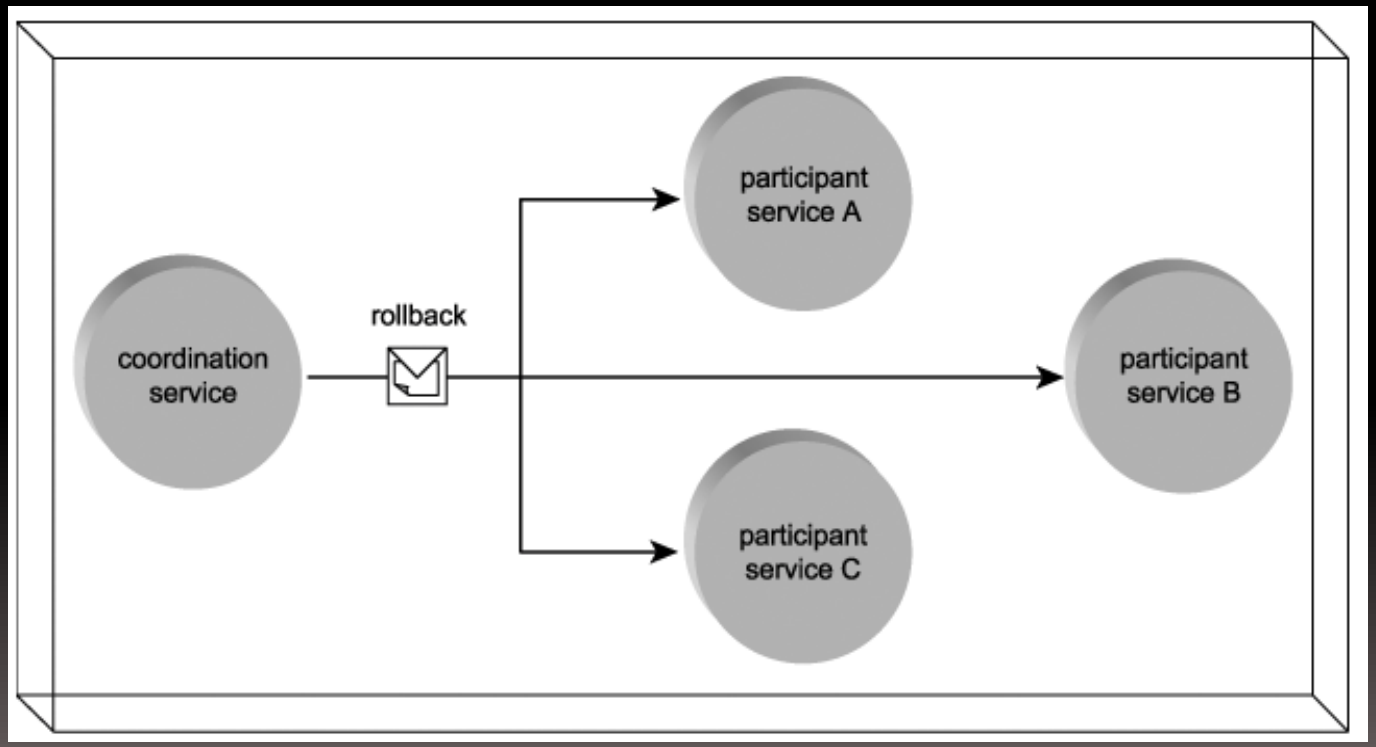
The atomic transaction process

The transaction participants voting on the outcome of the atomic transaction



The atomic transaction process

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

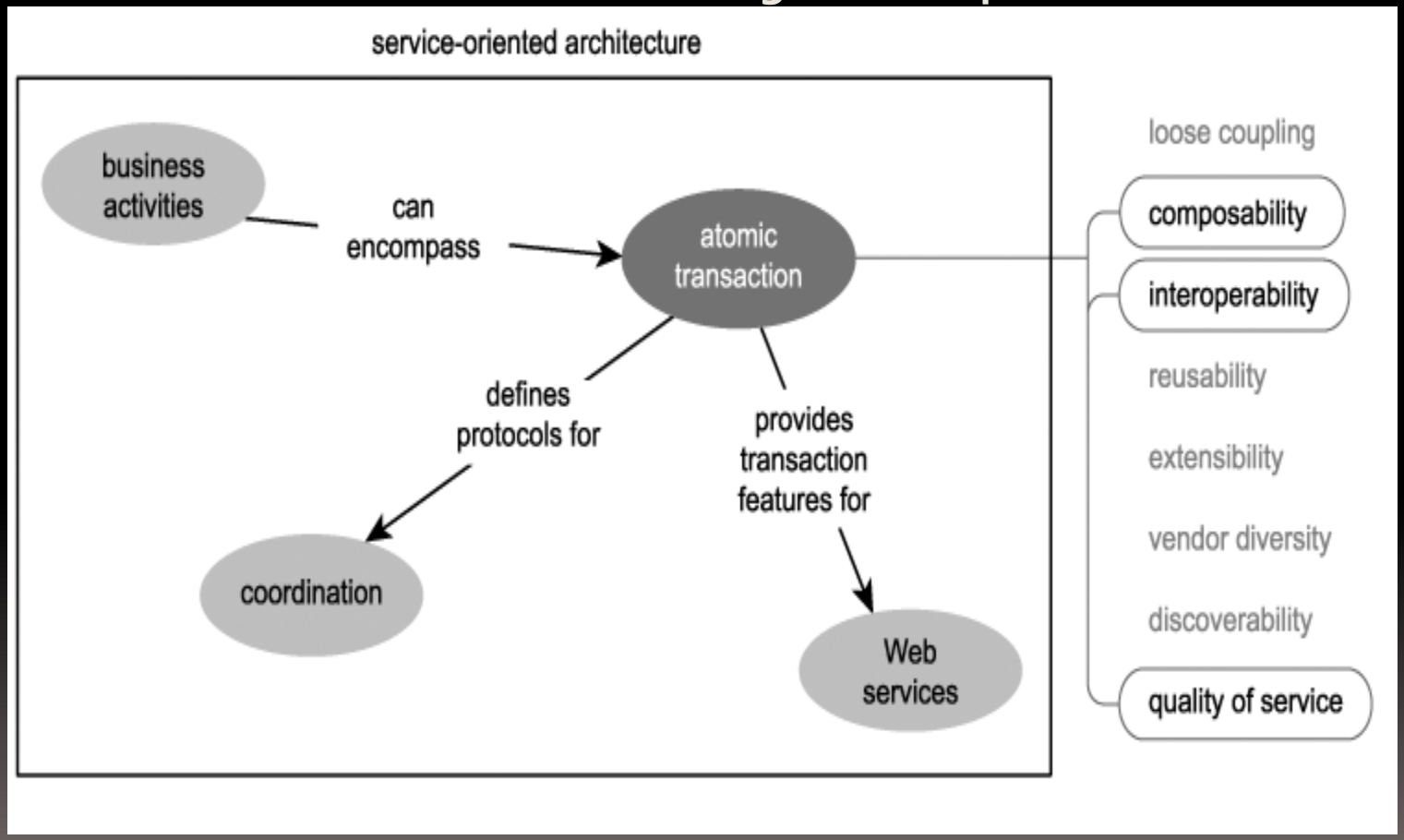


Atomic transactions and SOA

- 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.
- 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.

Atomic transactions and SOA

Atomic transaction relating to other parts of SOA



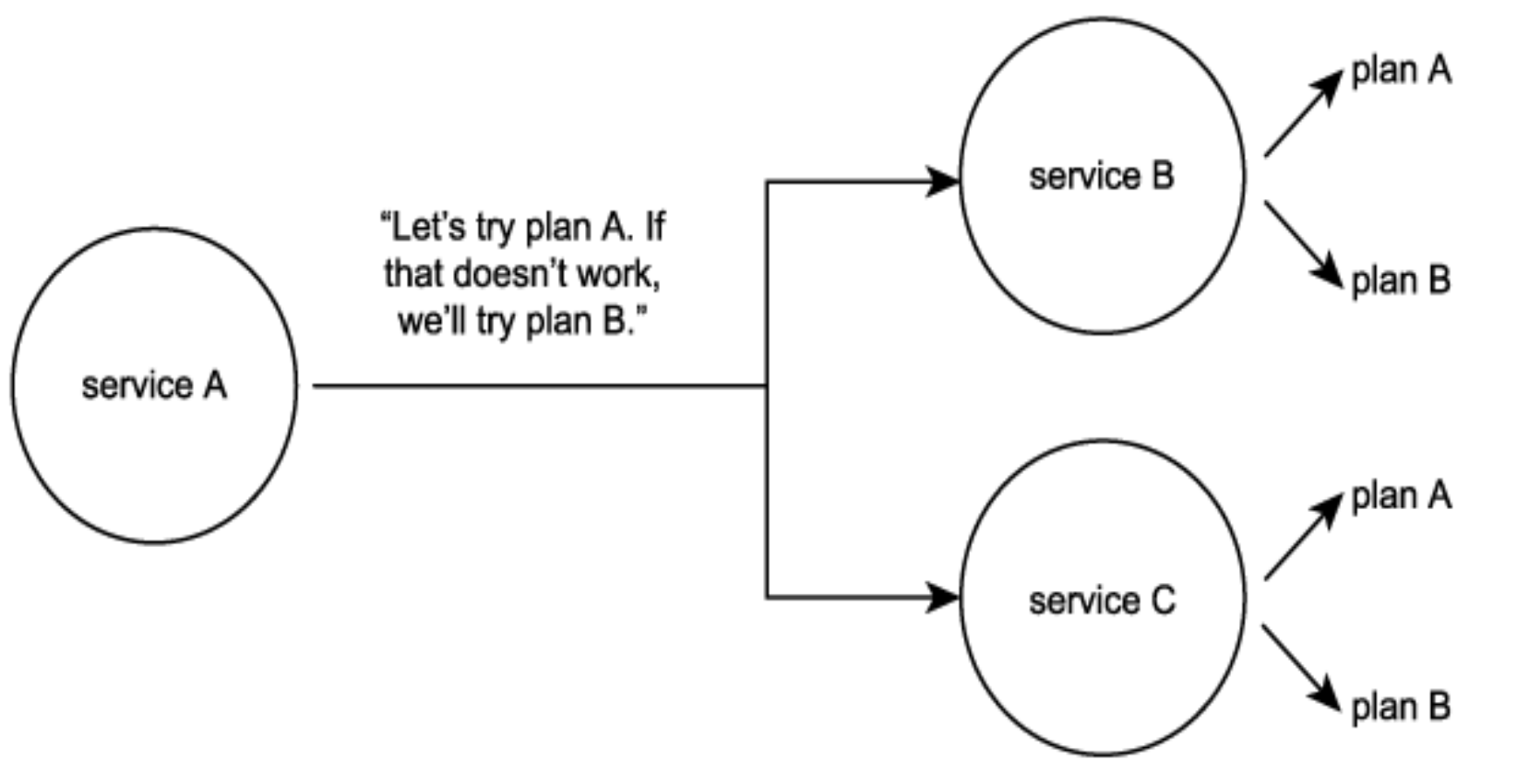
WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

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.
- business activity differs from a regular complex activity is that its participants are required to follow specific rules defined by protocols.
- Business activities differ from the protocol-based atomic transactions in how they deal with exceptions and in the nature of the constraints introduced by the protocol rules.business activity protocols do not offer rollback capabilities.
- The potential for business activities to be long-running, it would not be realistic to expect ACID-type transaction functionality.
- business activities provide an optional compensation process that, much like a "plan B," can be invoked when exception conditions are encountered

Business activities

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



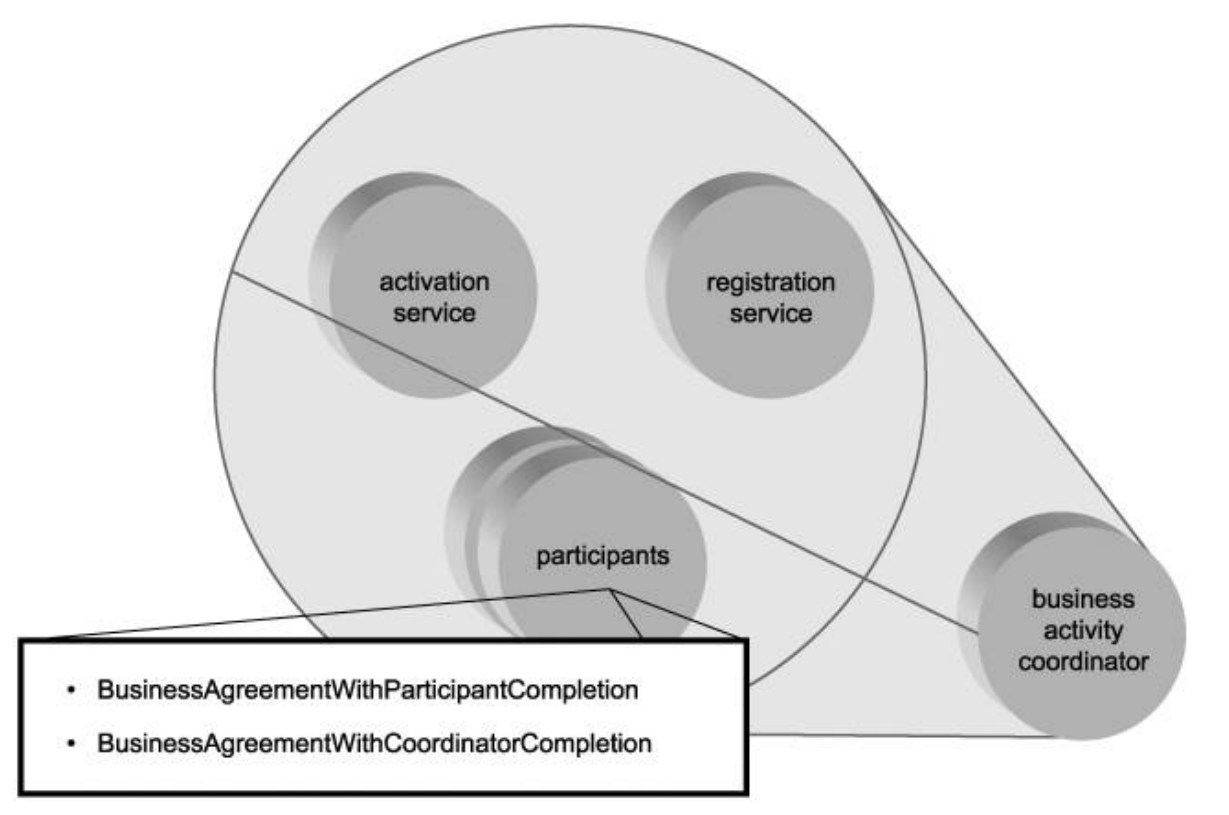
Business activity protocols

- 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

The business activity coordinator

When its protocols are used, the WS-Coordination controller service assumes a role specific to the coordination typein this case it becomes a business activity coordinator .

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.
- 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.
- 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.

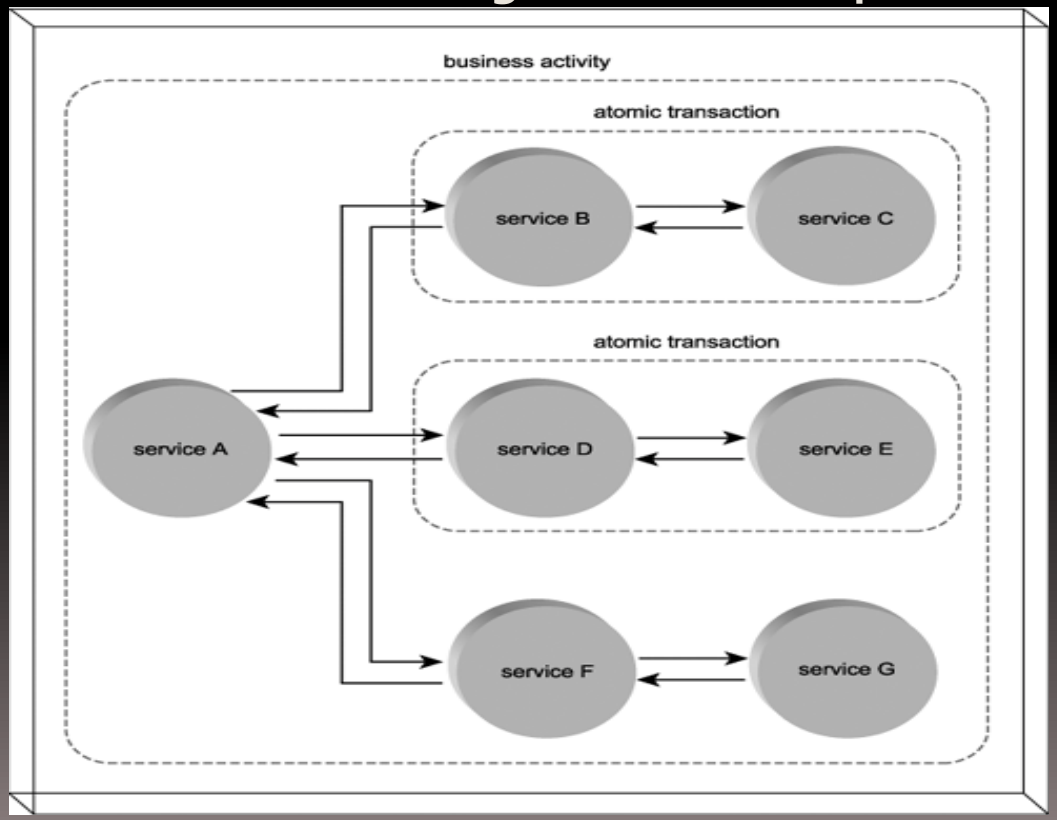
Business activity states

- 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.
- 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.

Business activities and atomic transactions

- the use of a business activity does not exclude the use of atomic transactions. 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

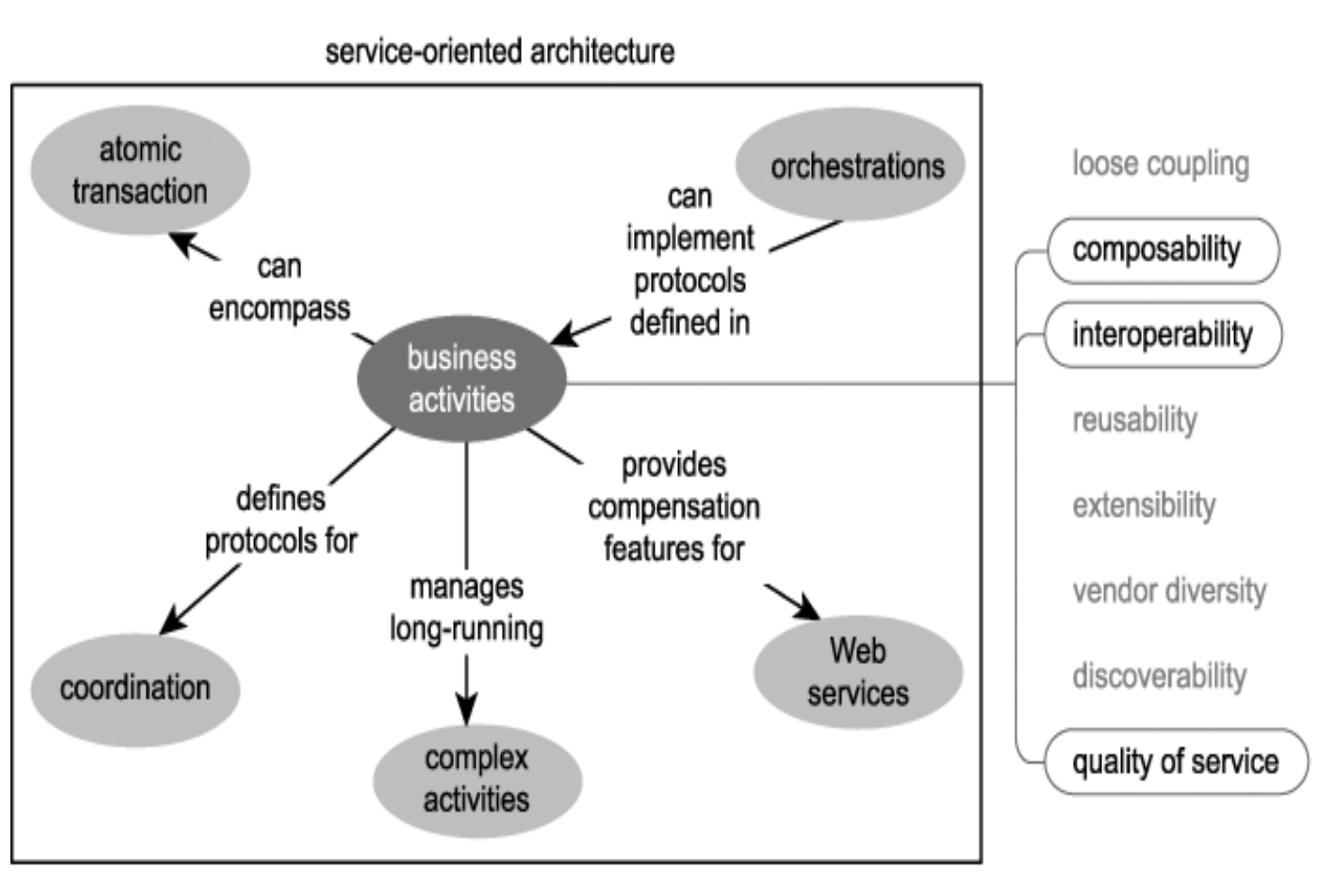


Business activities and SOA

- 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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

A business activity relating to other parts of SOA



Orchestration

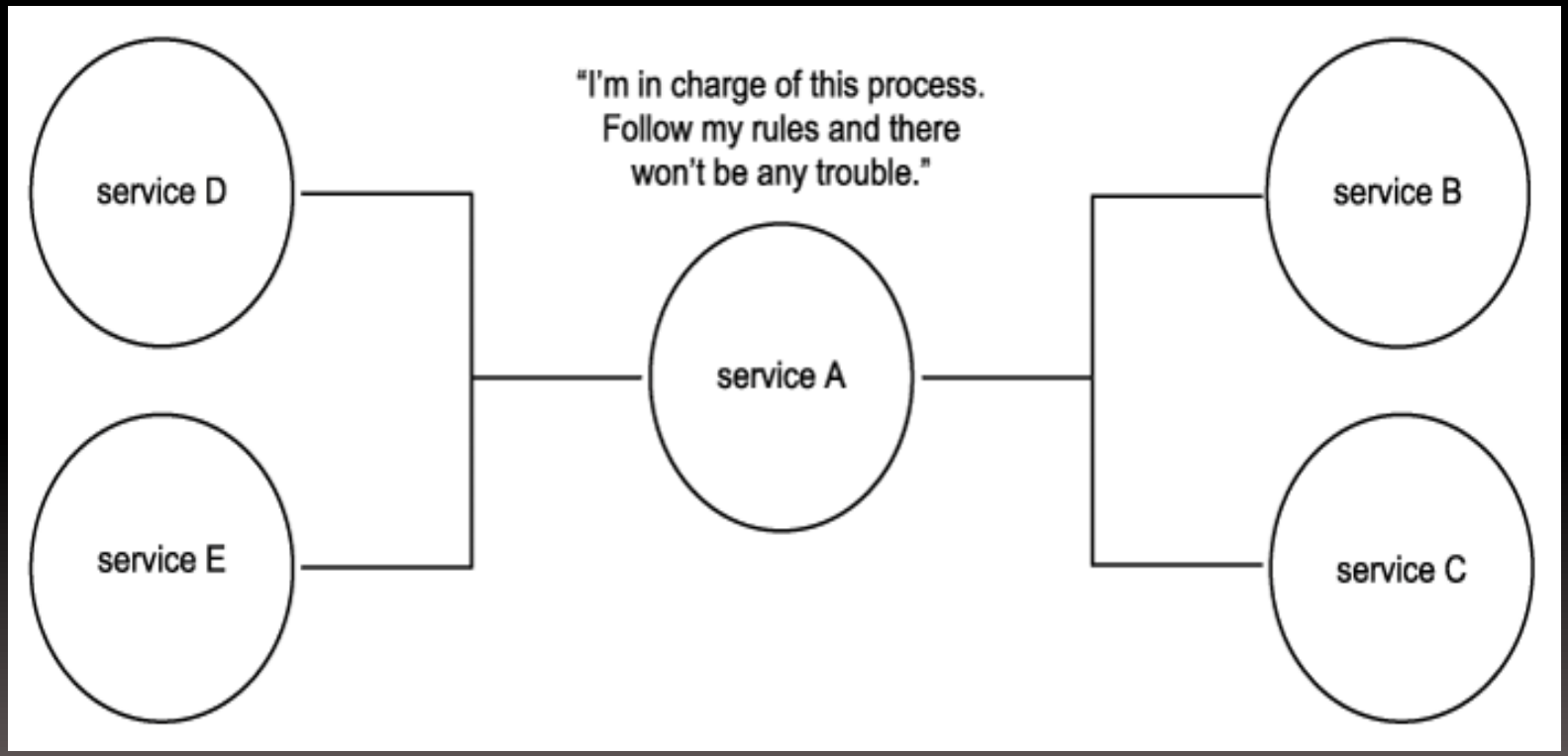
- A centrally controlled set of workflow logic facilitates interoperability between two or more different applications.
- A common implementation of orchestration is the hub-and-spoke model that allows multiple external participants to interface with a central orchestration engine.
- One of the driving 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.
- 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

- 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.
- Building upon orchestration logic standardizes process representation across an organization, while addressing the goal of enterprise federation and promoting service-orientation.

Orchestration

An orchestration controls almost every facet of a complex activity



WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Orchestration

Business protocols and process definition

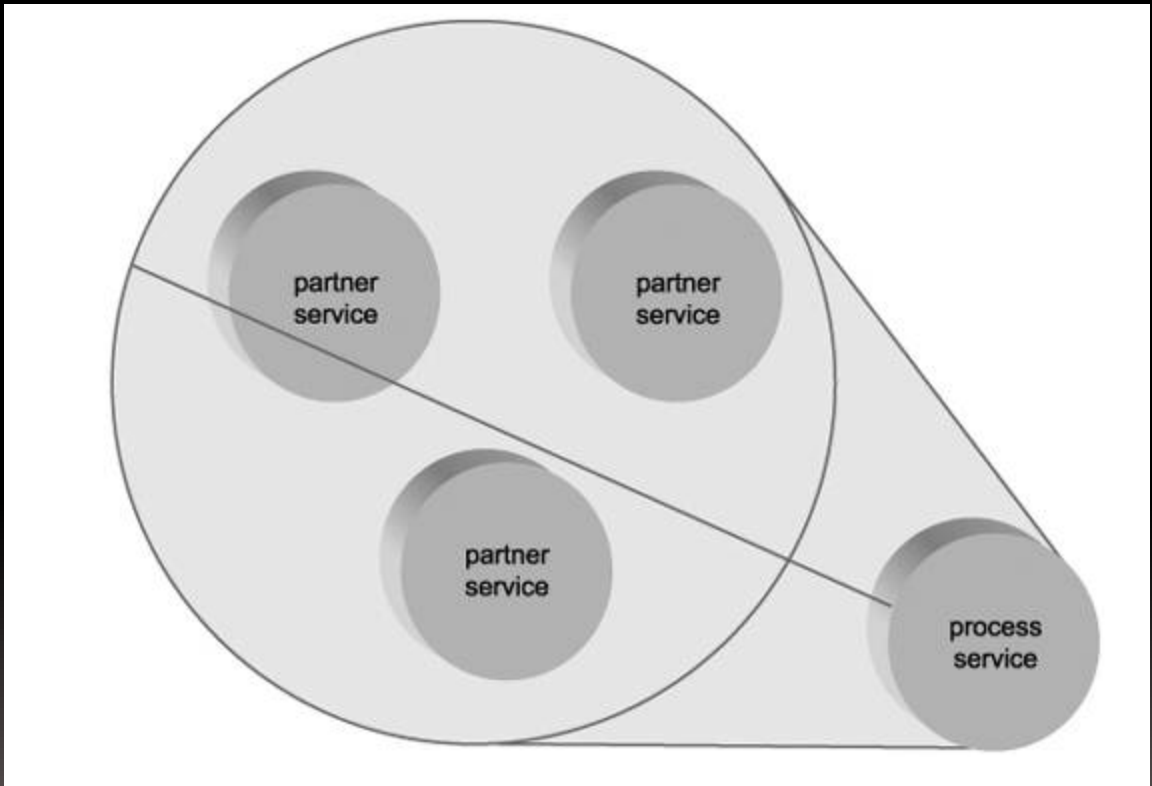
- The workflow logic that comprises an orchestration can consist of numerous business rules, conditions, and events.
- Collectively, these parts of an orchestration establish a business protocol that defines how participants can interoperate to achieve the completion of a business task.
- The details 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)

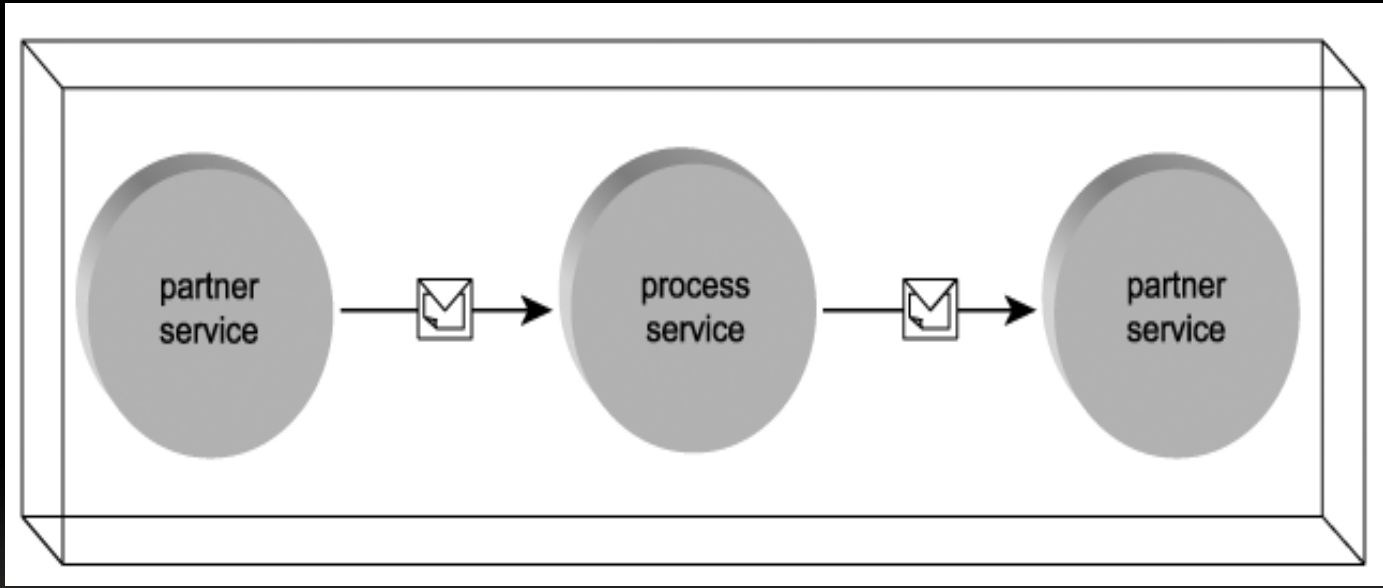
Orchestration

A process service coordinating and exposing functionality from three partner services



Orchestration

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



Orchestration

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).
- How these activities can be used to express actual business process logic

Orchestration

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.

Orchestration

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

Orchestration

Orchestration and SOA

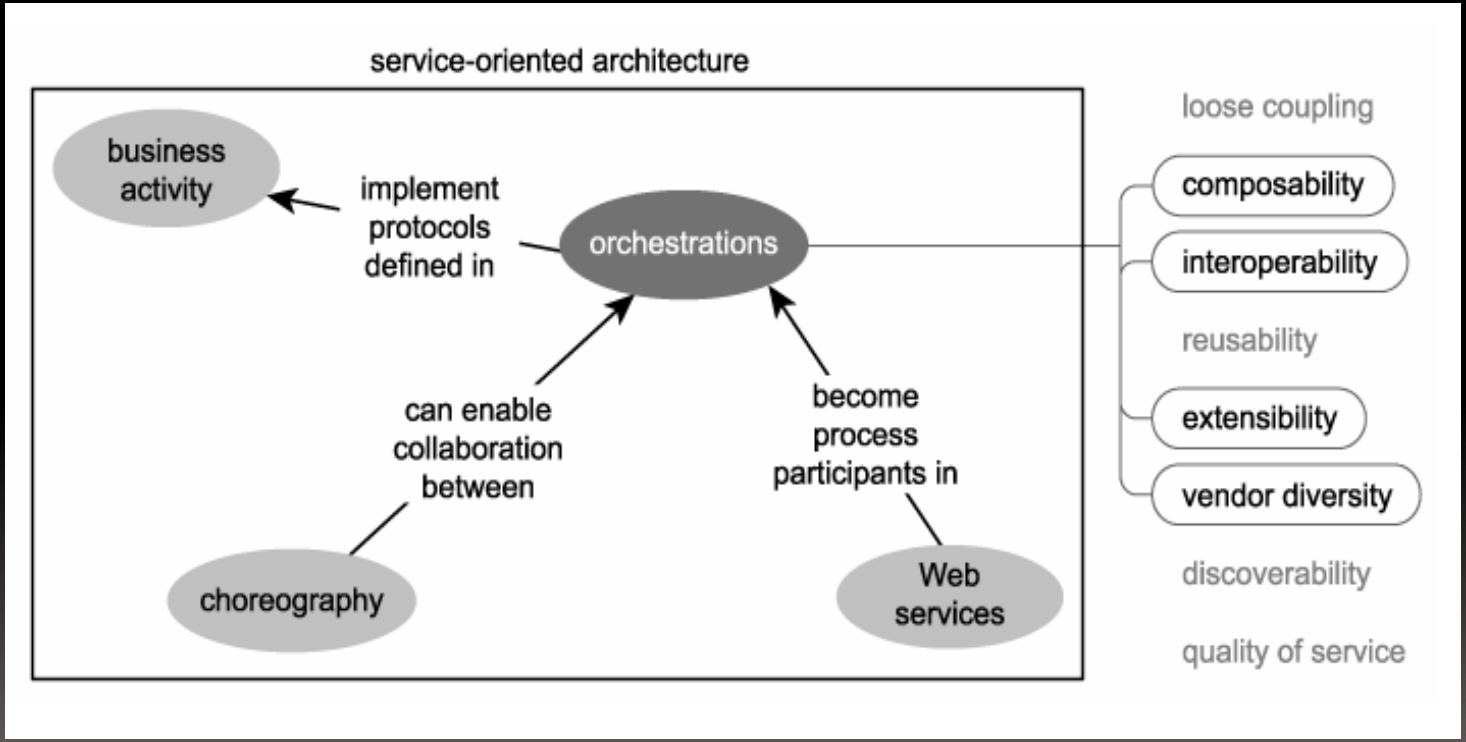
- Business process logic is at the root of automation solutions.
- Orchestration provides an automation model where process logic is centralized yet still extensible and composable .
- Through the use of orchestrations, service-oriented solution environments become inherently extensible and adaptive.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

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Orchestration

Orchestration and SOA

Orchestration relating to other parts of SOA



Orchestration

In Plain English

After successfully washing several cars together, Chuck, Bob, Jim, and I decide to start our own company. We formalize the steps in our car washing process so that we can handle different types of cars with different cleaning requirements.

Our process is therefore affected by the following new requirements:

- We decide to hire extra help during peak hours. This introduces up to two additional members that join our team.
- Because we have no venture capital for this business, we make an arrangement with a local gas station. In exchange for using a portion of their lot for our car washing operation, we agree to help out with the gas pumping duties during their peak hours.

Our simple car washing process now has become significantly more complicated. The process is no longer fixed in that it can change at any given time as a result of various conditions and events.

- When our extra workers arrive, the task allocation of the entire team is altered.
- When gas station personnel need extra help, we are obligated to send one or more of our car washing team members to assist them.

These examples relate to predictable conditions that occur on a daily basis. Our operation is further affected by some constraints:

- If our cash flow falls below a certain amount, we are unable to afford part-time workers.
- If it rains, all work is suspended (also leading to reduced cash flow).

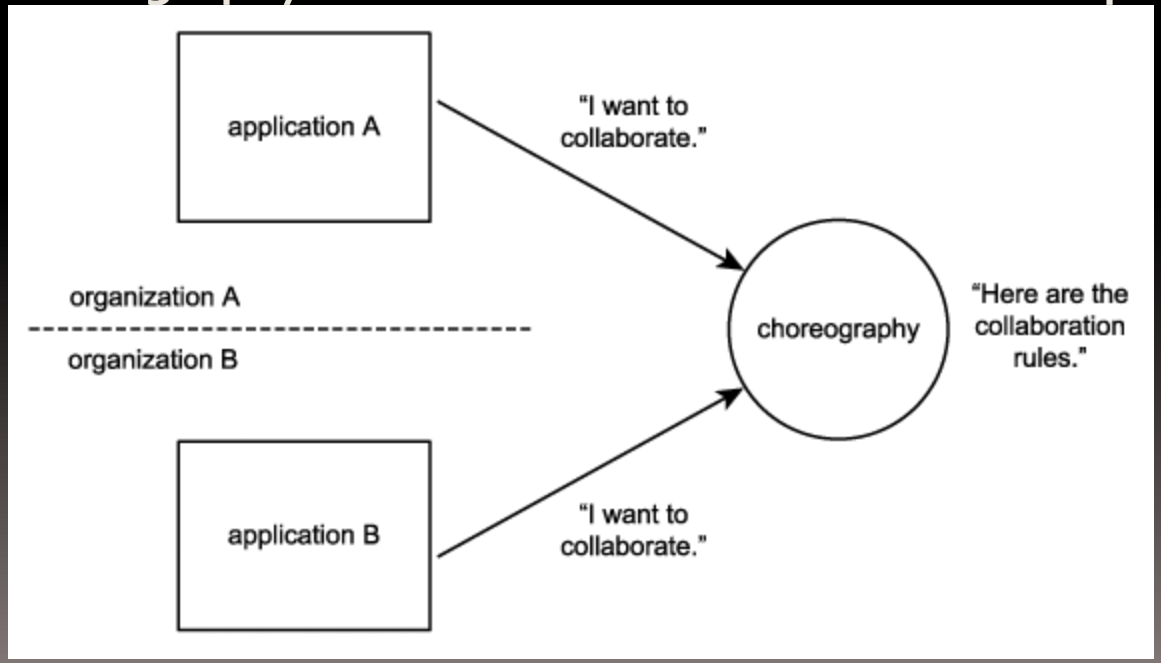
These constraints introduce conditions that are less common, but which we always need to take into consideration. To accommodate these potential situations, we come up with a plan that maps out our expanded process and provides alternative processes for dealing with both common and uncommon conditions.

This plan is essentially a workflow that joins individual steps with processes and sub-processes partitioned by decision points. This elaborate workflow incorporates our original process with the gas station's process and the extended process resulting from the arrival of our part-time workers. This workflow is essentially an orchestration that manages the individual process requirements and related resources, participants, events, business rules, and activities.

Choreography

- multiple services from different organizations need to work together to achieve a common goal
- 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

A choreography enables collaboration between its participants



Choreography

In Plain English

After a few months in operation, our little car washing enterprise achieves some success. With our flexible and adaptive system, we have been able to efficiently wash enough cars to make some profit. Once word in the car washing circles gets around, we are contacted by a nearby car washing company.

Even though this team of car washers is located only a kilometer away, they are not considered competitors. We positioned ourselves at a gas station located at the off ramp of a highway, and they are on the other side. Our customers originate from North-bound traffic, whereas theirs come from cars heading South. As a result, we have different peak hours corresponding directly to the traffic patterns of the highway. Our volume peaks during the morning rush hours, whereas theirs peaks in the afternoon.

It is suggested to us that we could form a partnership whereby we pool our respective resources (workers) to allow each of our companies to maximize the potential of each rush hour period. This form of collaboration appeals to us, as so far we've never been able to wash as many cars as we could at peak times. When customers entering the gas station grounds see a line up to our car wash, they often change their minds and drive away.

We decide to join forces with the other team. However, this arrangement soon affects our original business process. We now have to introduce a process that imposes new conditions and constraints. At the same time, though, we want to protect our existing system because it has been successful. After discussing these issues with our new partner, we come to an agreement that results in a flexible collaboration process.

A choreography is essentially a collaboration process designed to allow organizations to interact in an environment that is not owned by any one partner.

Choreography

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.

Roles and participants

- Within any given choreography, a Web service assumes one of a number of predefined roles. Roles can be bound to WSDL definitions, and those related are grouped accordingly, categorized as participants (services).

Choreography

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 it is defined who can talk with each other and requires establishing the nature of the conversation.
- Channels do exactly that by defining the characteristics of the message exchange between two specific roles.

Choreography

Relationships and channels

- 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.

Choreography

Interactions and work units

- 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 which impose rules and constraints that must be adhered to for an interaction to successfully complete.

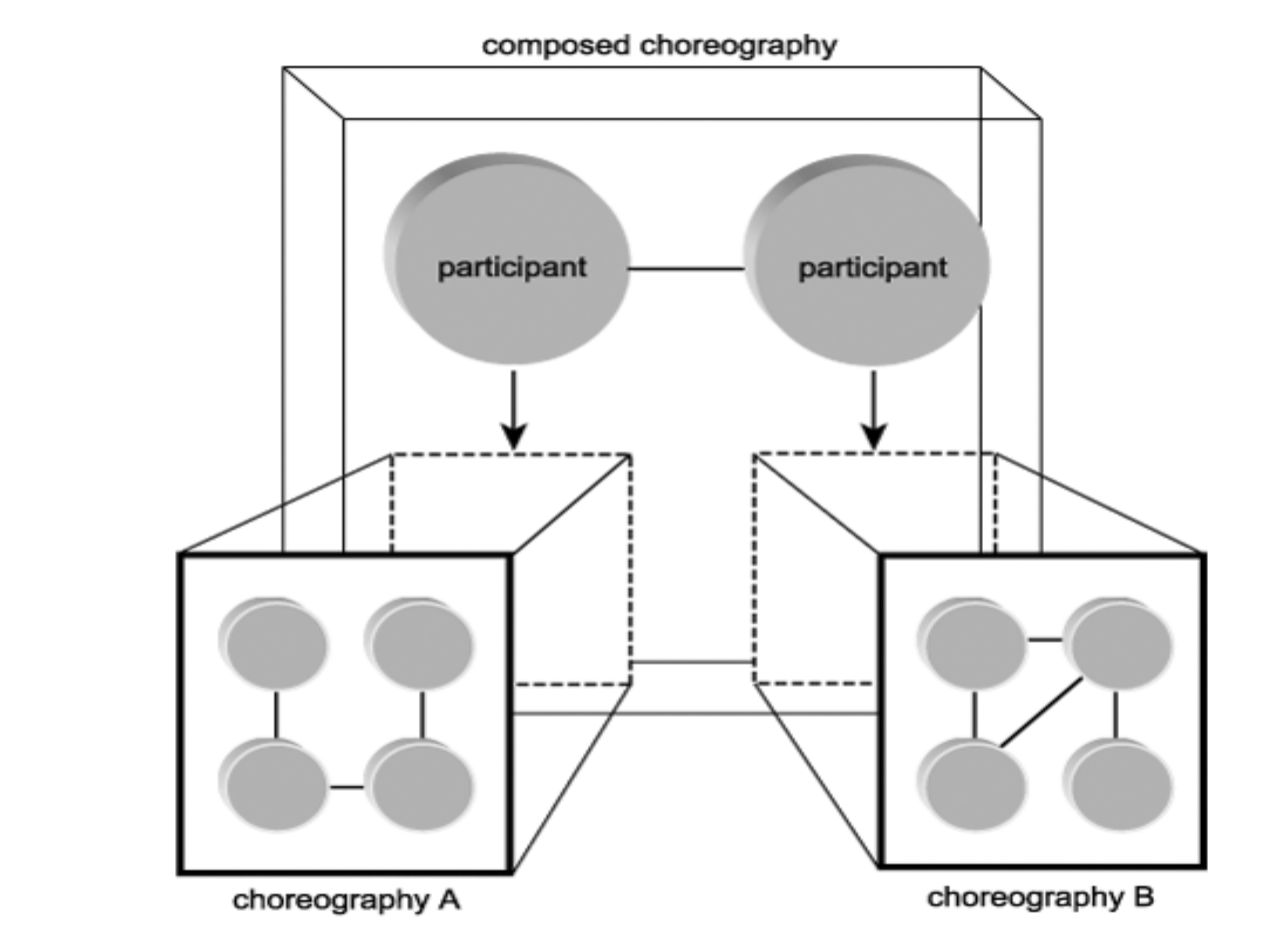
Choreography

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.
- 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

A choreography composed of two smaller choreographies



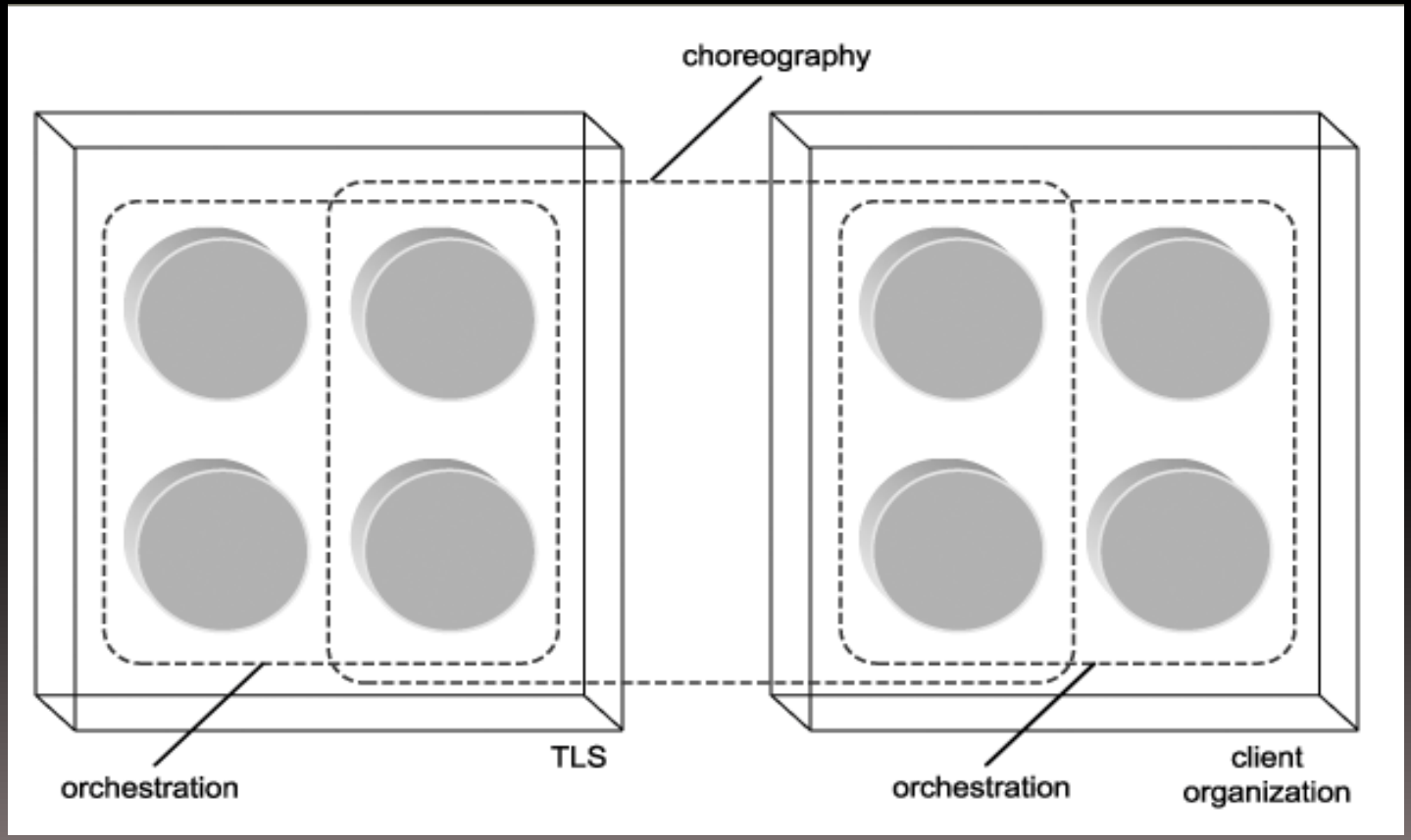
Choreography

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

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

- Choreography
- Orchestrations and choreographies
  - A choreography enabling collaboration between two different orchestrations



Choreography

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.
- One overlap between the current orchestration and choreography extensions is the fact that orchestrations can be designed to include multi-organization participants.
- An orchestration can effectively establish cross-enterprise activities in a similar manner as a choreography.
- primary distinction is the fact that an orchestration is generally owned and operated by a single organization

Choreography

Choreography and SOA

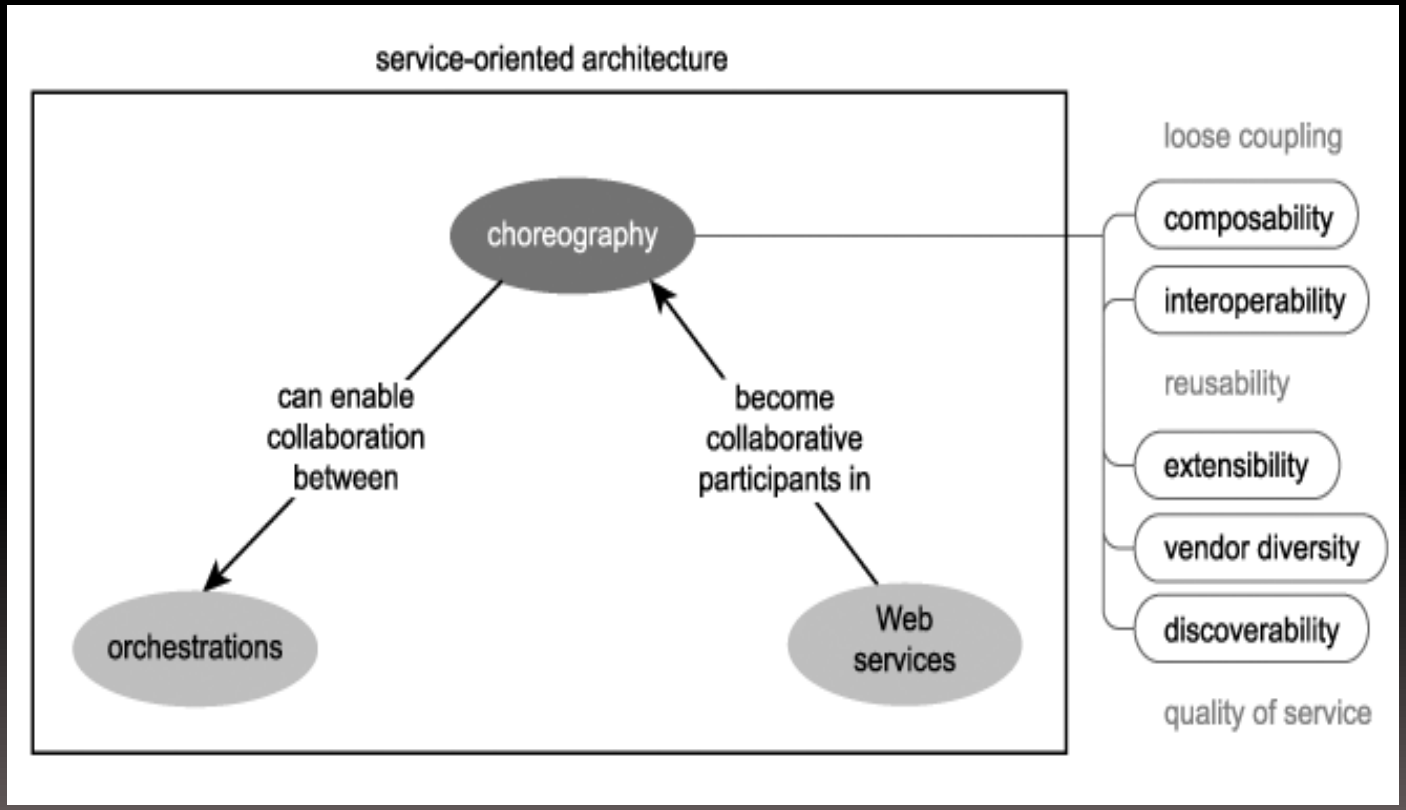
- Two services within a single organization, each exposing a simple function, can interact via a basic MEP to complete a simple task.
- Two services belonging to different organizations, each exposing functionality from entire enterprise business solutions, can interact via a basic choreography to complete a more complex task.
- Both scenarios involve two services, and both scenarios support SOA implementations.
- Choreography 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.

WEB SERVICES AND CONTEMPORARY SOA (PART I: ACTIVITY MANAGEMENT AND COMPOSITION)

Choreography

Choreography and SOA

Choreography relating to other parts of SOA.

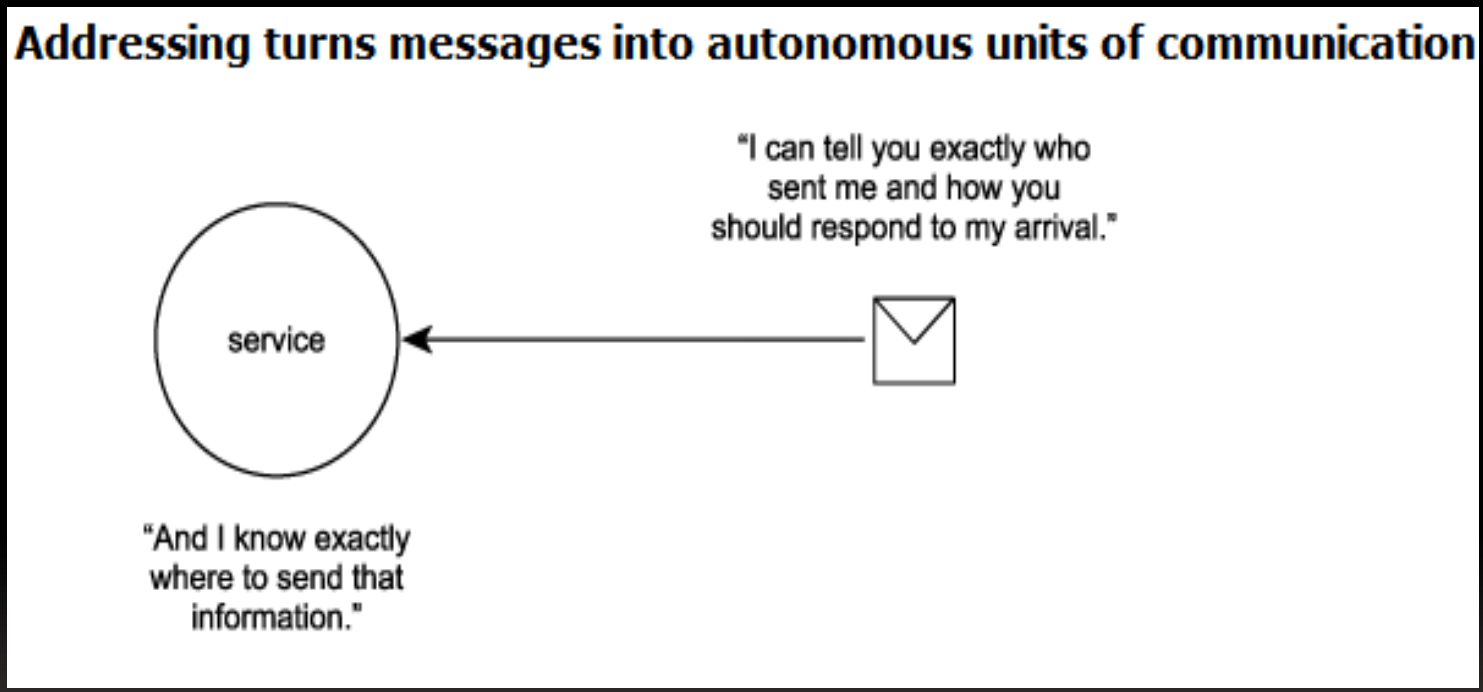


WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing

- What addressing brings to SOAP messaging is much like what a waybill brings to the shipping process.
- Regardless of which ports, warehouses, or delivery stations a package passes through en route to its ultimate destination, with a waybill attached to it, everyone it comes into contact with knows:
  - where it's coming from
  - the address of where it's supposed to go
  - the specific person at the address who is supposed to receive it
  - where it should go if it can't be delivered as planned
- The WS-Addressing specification implements these addressing features by providing two types of SOAP headers .
- Addressing extensions are integral to SOA's underlying messaging mechanics.
- Many other WS-* specifications implicitly rely on the use of WS-Addressing

Addressing


Addressing turns messages into autonomous units of communication
"I can tell you exactly who sent me and how you should respond to my arrival."
service
"And I know exactly where to send that information."

Addressing

In Plain English

As our car washing company grows, so do the administration duties. Every week Chuck reviews the mail and takes care of necessary paperwork. This week he receives two letters: one from our insurance company and the other from the tax office.

The first letter includes our renewed insurance policy statement, along with an invoice for another year of coverage. The "from" address on this letter is simply the name and location of the insurance company's head office. The enclosed statement contains a letter written by our account representative, outlining some of the changes in this year's policy and requesting that we mail our check directly to him. Chuck therefore encloses our payment in an envelope with a "to" address that includes an "attention" line stating that this letter should be delivered directly to the account representative.

The next letter contains another bill. This time, it's a tax statement accompanied by a letter of instruction and two return envelopes. According to the instructions, we are to use the first envelope (addressed to the A/R office) to mail a check if we are paying the full amount owing. If we cannot make a full payment, we need to use the second envelope (addressed to the collections department) to send whatever funds we have.

These scenarios, in their own crude way, demonstrate the fundamental concepts of endpoint references and message information headers, which are explained in the following sections.

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing

Endpoint references

- For a service requestor to contact a service provider, the provider's WSDL definition is needed.
- This document supplies the requestor with an address at which the provider can be contacted.
- What if, though, the service requestor needs to send a message to a specific instance of a service provider? In this case, the address provided by the WSDL is not sufficient.
- Traditional Web applications had different ways of managing and communicating session identifiers.
- The most common approach was to append the identifier as a query string parameter to the end of a URL.
- While easy to develop, this technique resulted in application designs that lacked security and were non-standardized.

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing

Endpoint references

- The concept of addressing introduces the endpoint reference, an extension used primarily to provide identifiers that pinpoint a particular instance of a service (as well as supplementary service metadata).

- The endpoint reference is expected to be almost always dynamically generated and can contain a set of supplementary properties.

Addressing- Endpoint references

An endpoint reference consists of the following parts:

- address
  - The URL of the Web service.

reference properties

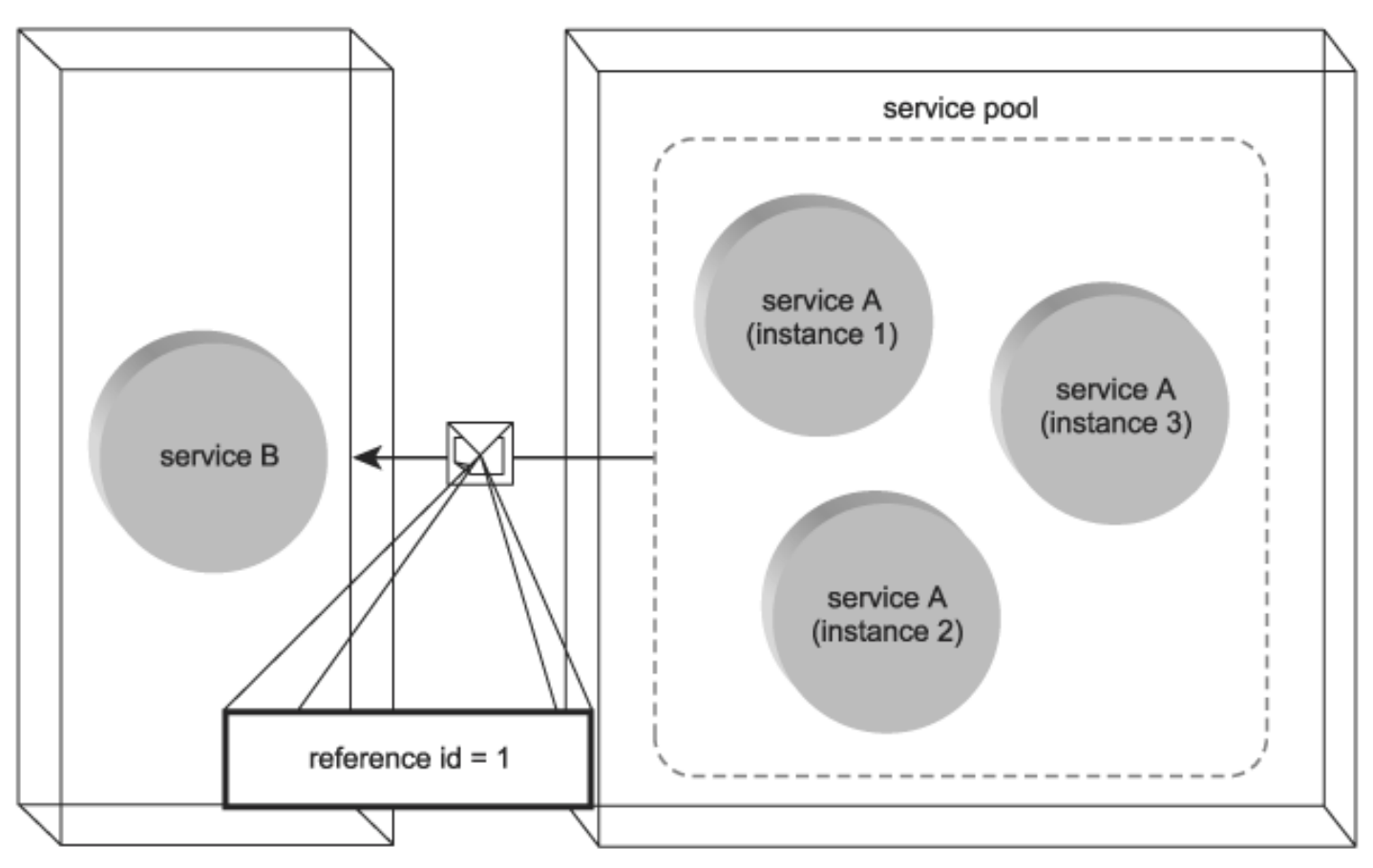
  - A set of property values associated with the Web service instance. (In Plain English example, the "attention" line used in the first scenario is representative of the reference ID property.)

- reference parameters
  - A set of parameter values that can be used to further interact with a specific service instance.
- service port type and port type
  - Specific service interface information giving the recipient of the message the exact location of service description details required for a reply.
- policy
  - A WS-Policy compliant policy that provides rules and behavior information relevant to the current service interaction

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing- Endpoint references

A SOAP message containing a reference to the instance of the service that sent it



WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing- Endpoint references

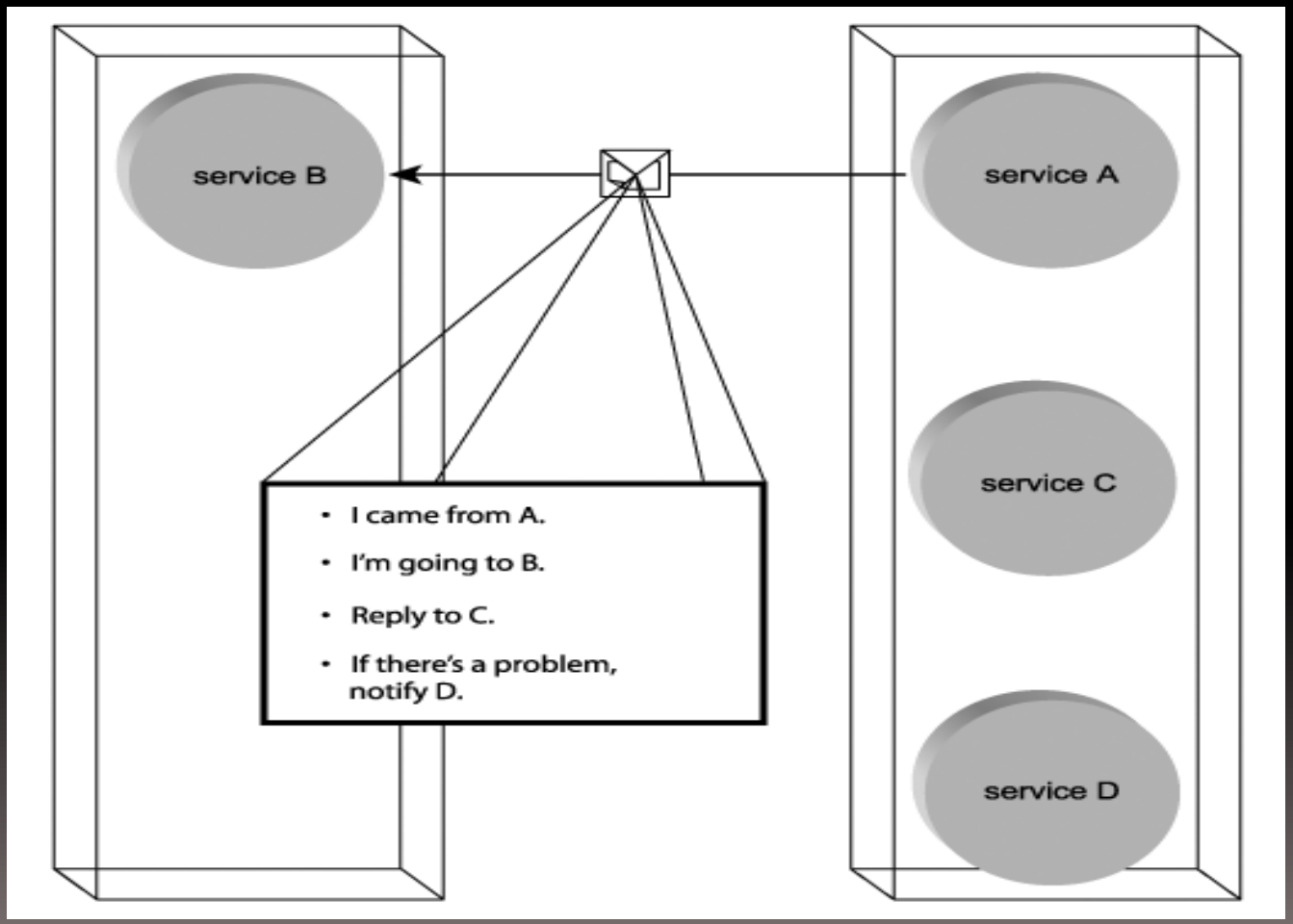
Message information headers

- MEPs have predictable characteristics that can ease the manner in which Web services are designed but also can limit the service interaction scenarios within which they participate.
- In service-oriented solutions, services often require the flexibility to break a fixed pattern.
- For example, they may want to dynamically determine the nature of a message exchange.
- The extensions provided by WS-Addressing were broadened to include new SOAP headers that establish message exchange-related characteristics within the messages themselves.
- This collection of standardized headers is known as the message information (or MI) headers

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Addressing- Endpoint references

A SOAP message with message information headers specifying exactly how the recipient service should respond to its arrival



Addressing- Endpoint references

The MI headers provided by WS-Addressing include:

- destination

  The address to which the message is being sent.

- source endpoint

  An endpoint reference to the Web service that generated the message.

- reply endpoint

  This important header allows a message to dictate to which address its reply should be sent.

- fault endpoint

  Further extending the messaging flexibility is this header, which gives a message the ability to set the address to which a fault notification should be sent.

- message id

  A value that uniquely identifies the message or the retransmission of the message (this header is required when using the reply endpoint header).

Addressing- Endpoint references

The MI headers provided by WS-Addressing include:

- relationship

  Most commonly used in request-response scenarios, this header contains the message id of the related message to which a message is replying (this header also is required within the reply message).

- action

  A URI value that indicates the message's overall purpose (the equivalent of the standard SOAP HTTP action value).

  [WS-Addressing specification provides an anonymous URI that allows MI headers to intentionally contain an invalid address]

Addressing- Endpoint references

- A SOAP message with these headers further increases its position as an independent unit of communication.
- Using MI headers, SOAP messages now can contain detailed information that defines the messaging interaction behavior of the service in receipt of the message.
- The net result is standardized support for the use of unpredictable and highly flexible message exchanges, dynamically creatable and therefore adaptive and responsive to runtime conditions.

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

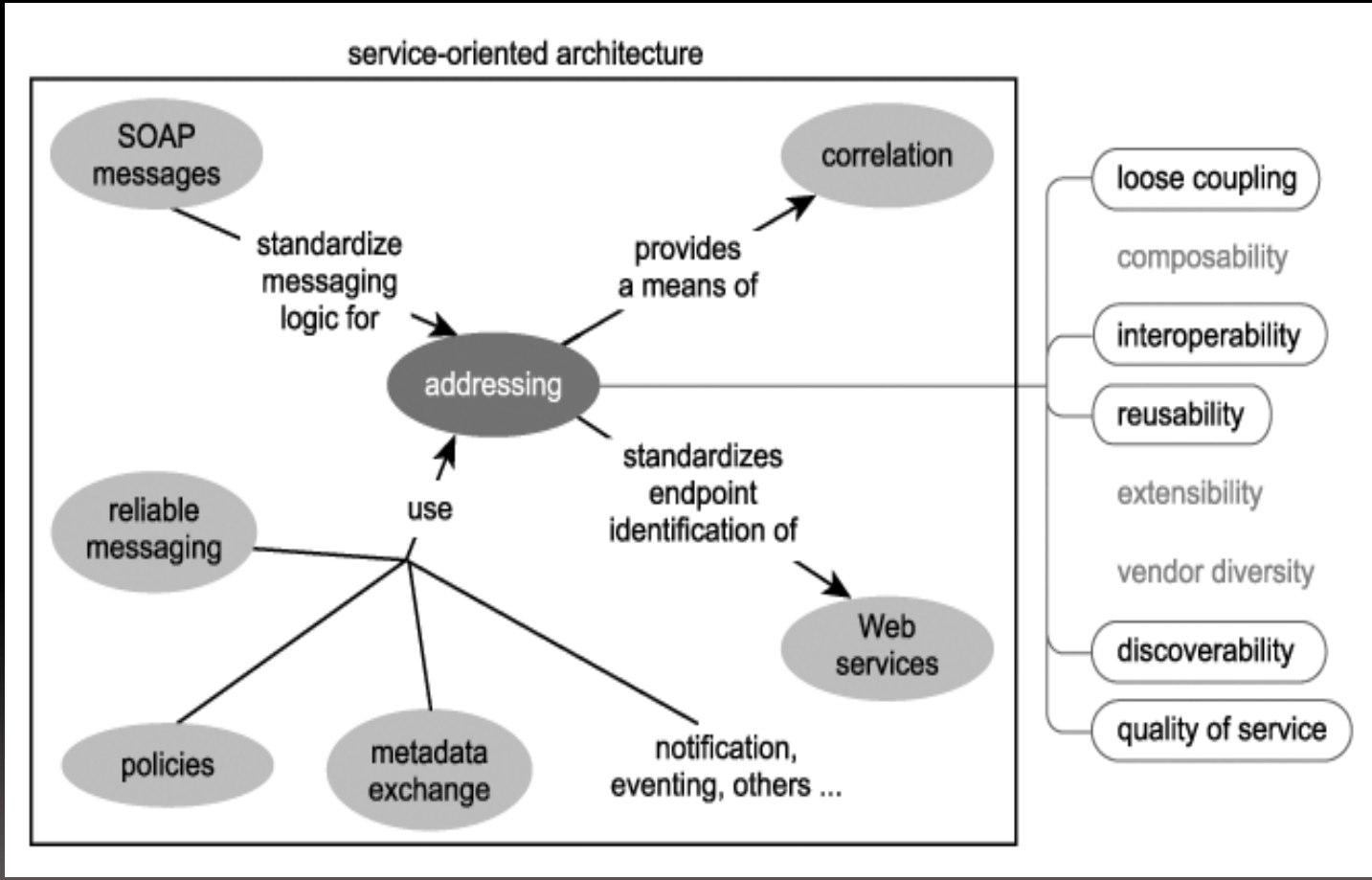
Addressing- Endpoint references

Addressing and SOA

- Addressing achieves an important low-level, transport standardization within SOA, further promoting open standards that establish a level of transport technology independence .
- The use of endpoint references and MI headers deepens the intelligence embedded into SOAP messages, increasing message-level autonomy.
- Empowering a message with the ability to self-direct its payload, as well as the ability to dictate how services receiving the message should behave, significantly increases the potential for Web services to be intrinsically interoperable.
- It places the task-specific logic into the message and promotes a highly reusable and generic service design standard that also facilitates the discovery of additional service metadata.

Addressing- Endpoint references

Addressing relating to other parts of SOA



Reliable messaging

- The benefits of a loosely coupled messaging framework come at the cost of a loss of control over the actual communications process.

- After a Web service transmits a message, it has no immediate way of knowing:
  - whether the message successfully arrived at its intended destination
  - whether the message failed to arrive and therefore requires a retransmission
  - whether a series of messages arrived in the sequence they were intended to

- Reliable messaging addresses these concerns by establishing a measure of quality assurance that can be applied to other activity management frameworks

Reliable messaging

WS-ReliableMessaging provides a framework capable of guaranteeing:

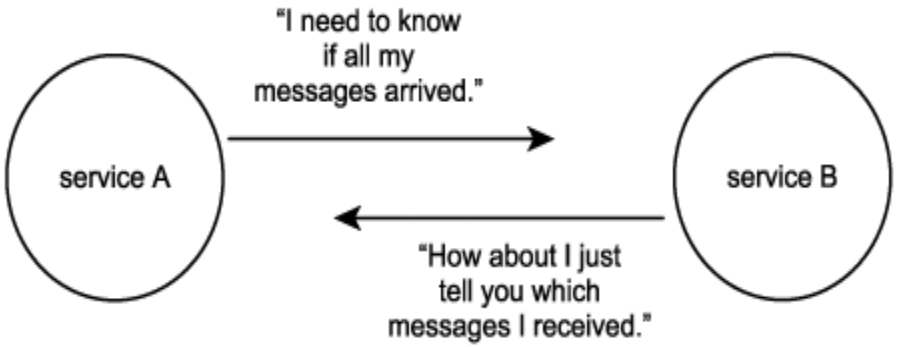
- that service providers will be notified of the success or failure of message transmissions
- that messages sent with specific sequence-related rules will arrive as intended (or generate a failure condition)

- Although the extensions introduced by reliable messaging govern aspects of service activities, the WS-ReliableMessaging specification is different from the activity management specifications .
- Reliable messaging does not employ a coordinator service to keep track of the state of an activity;
- instead, all reliability rules are implemented as SOAP headers within the messages themselves.

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

Reliable messaging

Reliable messaging provides a guaranteed notification of delivery success or failure.



In Plain English

In the last chapter's *Choreography* section we explained how our car wash had formed an alliance with the car wash located on the other side of the highway. Part of our arrangement was to share part-time workers during peak hours.

One of the workers that joined our team is named George. Though good at his job, George has a bad memory. When we request that workers from the other side walk over to help us out, we always are warned when one of those workers is George.

The walk from the other gas station is about one kilometer. Sometimes George forgets the way and gets lost. We therefore put a system in place where we agree to call the other company to tell them how many workers have arrived. If it's not equal to the number of workers they actually sent, it's usually because George has gone missing again.

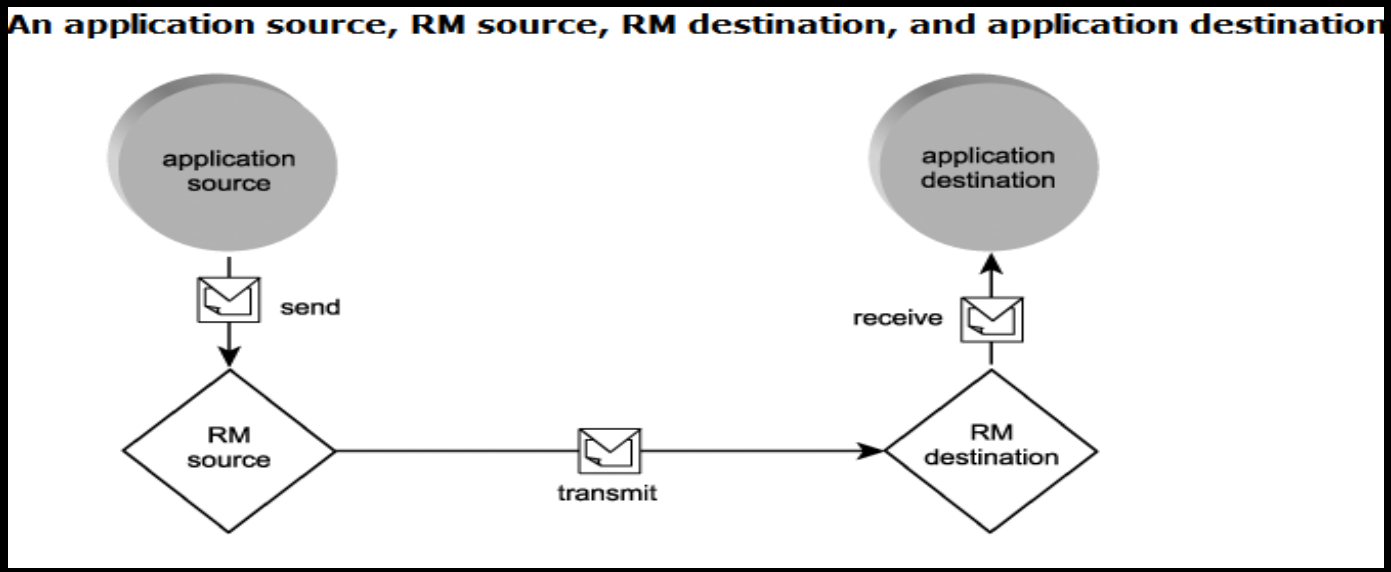
Our system of calling the other company to acknowledge the receipt of the workers and to report any missing workers builds an element of reliability into our resource sharing arrangement.

Reliable messaging

RM Source, RM Destination, Application Source, and Application Destination

- WS-ReliableMessaging makes a distinction between the parts of a solution that are responsible for initiating a message transmission and those that actually perform the transmission.
- It further assigns specific descriptions to the terms "send," "transmit," "receive," and "deliver," as they relate differently to these solution parts.
- These differentiations are necessary to abstract the reliable messaging framework from the overall SOA.
- An application source is the service or application logic that sends the message to the RM source, the physical processor or node that performs the actual wire transmission.
- RM destination represents the target processor or node that receives the message and subsequently delivers it to the application destination

Reliable messaging



Sequences

- A sequence establishes the order in which messages should be delivered.
- Each message that is part of a sequence is labeled with a message number that identifies the position of the message within the sequence.
- The final message in a sequence is further tagged with a last message identifier

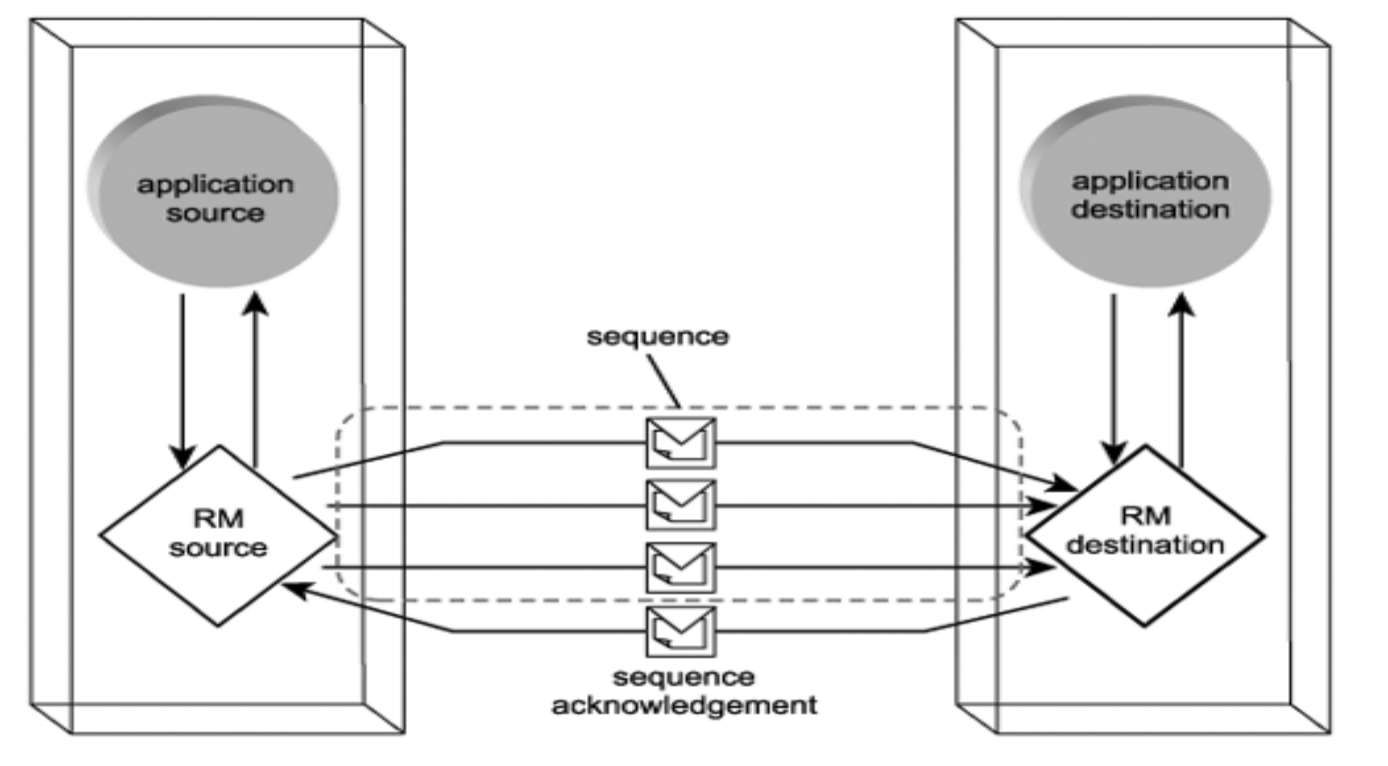
Reliable messaging

Acknowledgements

- A core part of the reliable messaging framework is a notification system used to communicate conditions from the RM destination to the RM source.
- Upon receipt of the message containing the last message identifier, the RM destination issues a sequence acknowledgement .
- The acknowledgement message indicates to the RM source which messages were received.
- It is up to the RM source to determine if the messages received are equal to the original messages transmitted.
- The RM source may retransmit any of the missing messages, depending on the delivery assurance used

Reliable messaging-Acknowledgements

A sequence acknowledgement sent by the RM destination after the successful delivery of a sequence of messages

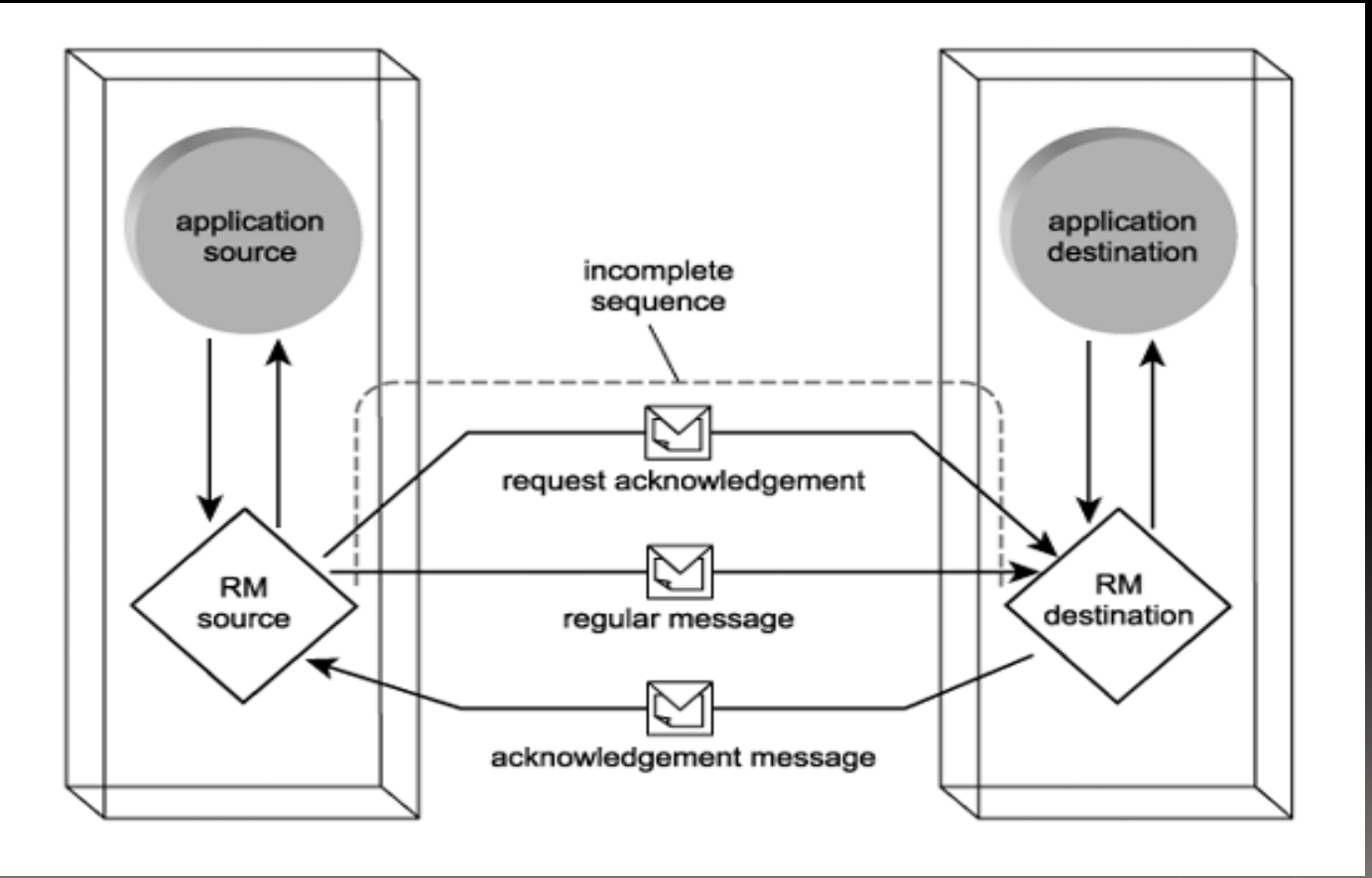


Reliable messaging-Acknowledgements

- An RM source does not need to wait until the RM destination receives the last message before receiving an acknowledgement.
- RM sources can request that additional acknowledgements be transmitted at any time by issuing request acknowledgements to RM destinations
- Additionally, RM destinations have the option of transmitting negative acknowledgements that immediately indicate to the RM source that a failure condition has occurred

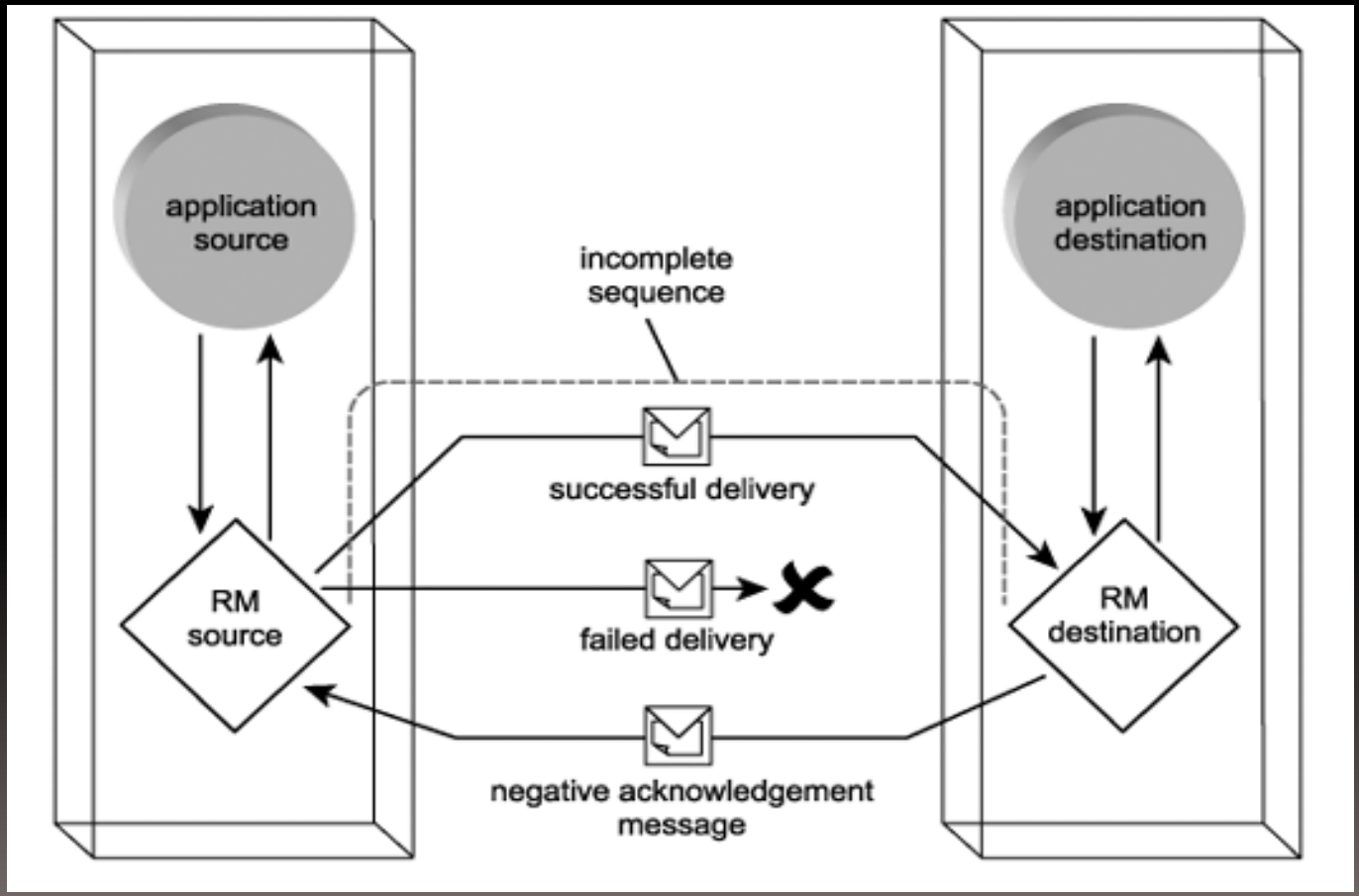
Reliable messaging-Acknowledgements

A request acknowledgement sent by the RM source to the RM destination, indicating that the RM source would like to receive an acknowledgement message before the sequence completes



Reliable messaging-Acknowledgements

A negative acknowledgement sent by the RM destination to the RM source, indicating a failed delivery prior to the completion of the sequence



Reliable messaging-Delivery assurances

- The nature of a sequence is determined by a set of reliability rules known as delivery assurances.
- Delivery assurances are predefined message delivery patterns that establish a set of reliability policies.

The following delivery assurances are supported:

The AtMostOnce delivery assurance

- promises the delivery of one or zero messages. If more than one of the same message is delivered, an error condition occurs

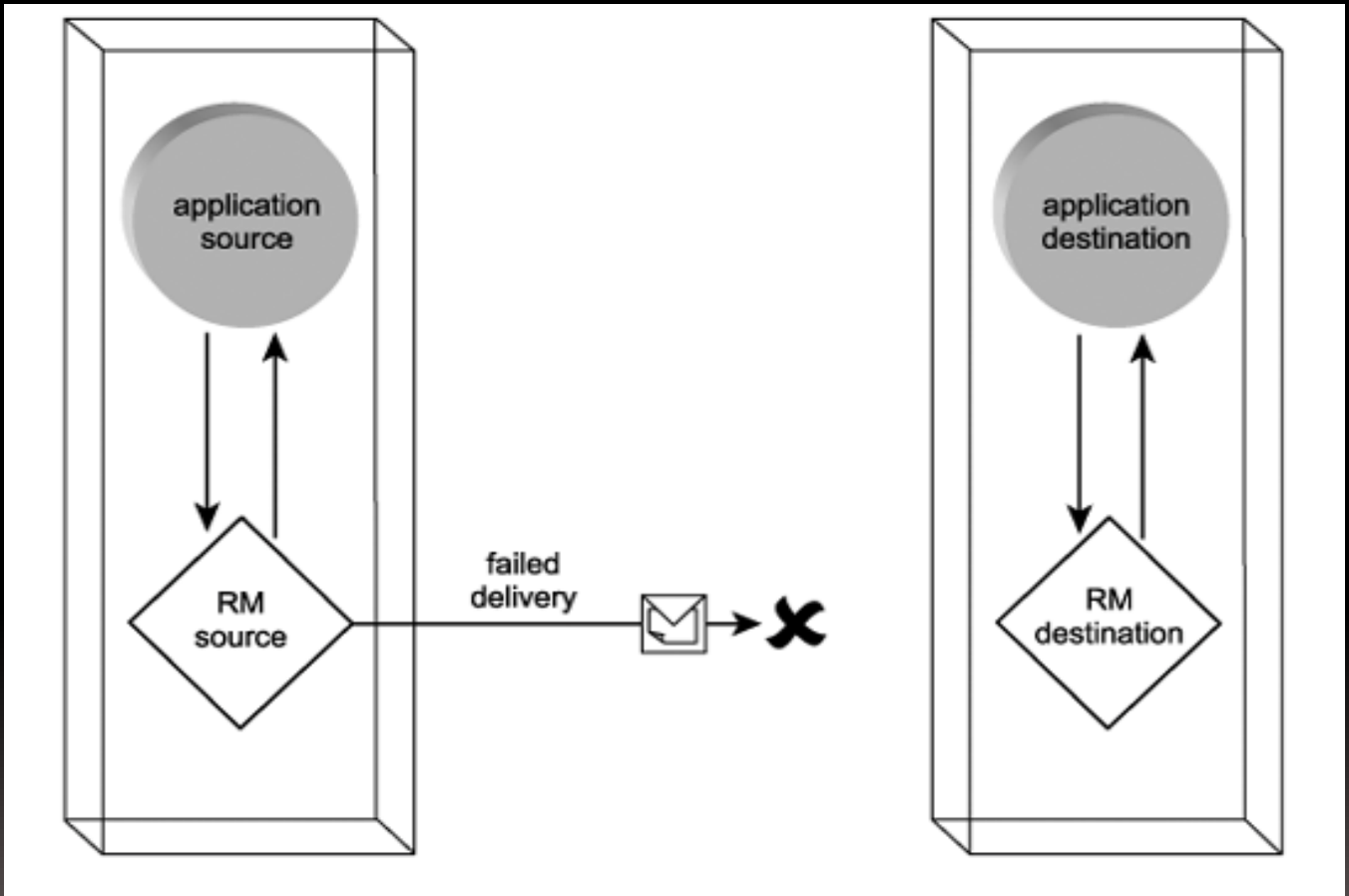
The ExactlyOnce delivery assurance

- guarantees that a message only will be delivered once. An error is raised if zero or duplicate messages are delivered

The InOrder delivery assurance

- is used to ensure that messages are delivered in a specific sequence .The delivery of messages out of sequence triggers an error. This delivery assurance can be combined with any of the previously described assurances.

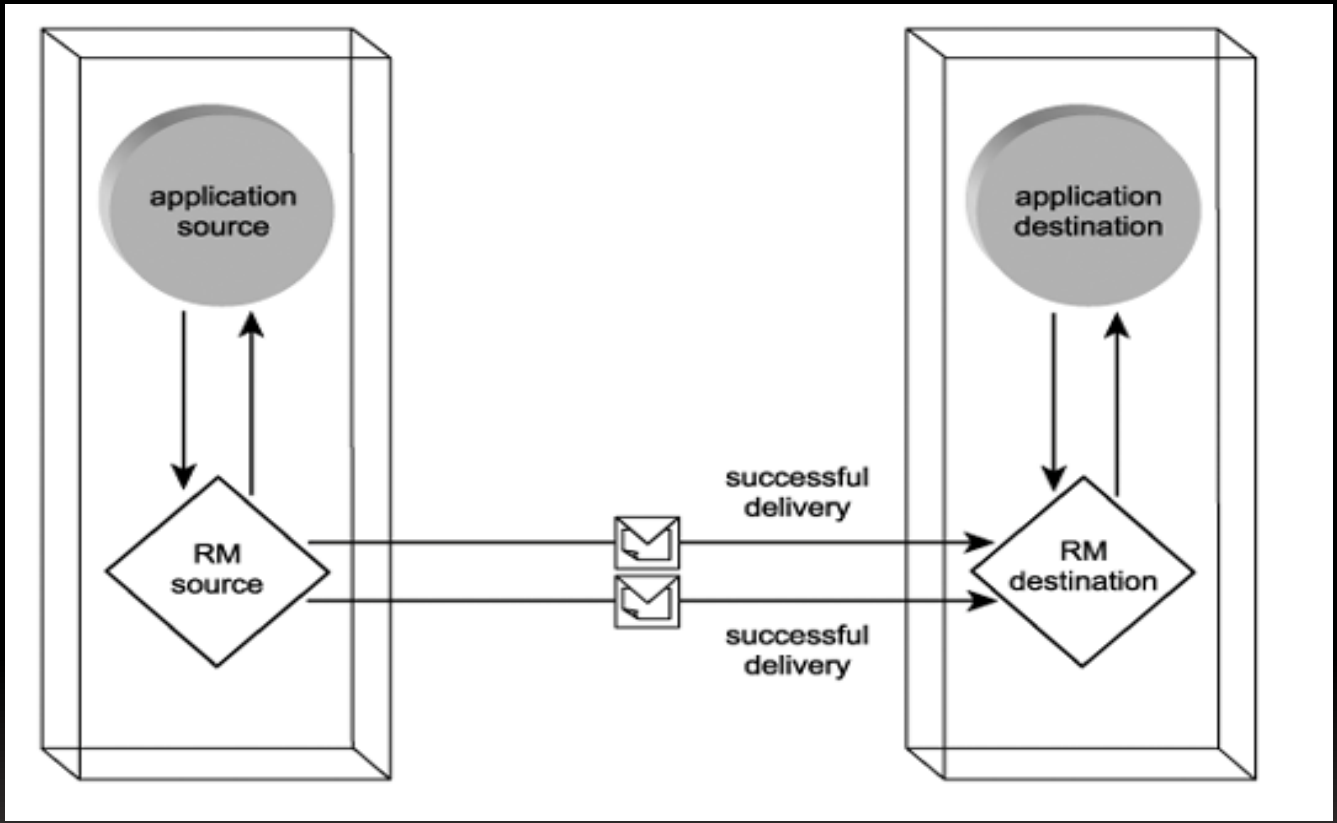
The AtMostOnce delivery assurance



The AtMostOnce delivery assurance promises the delivery of one or zero messages.

If more than one of the same message is delivered, an error condition occurs

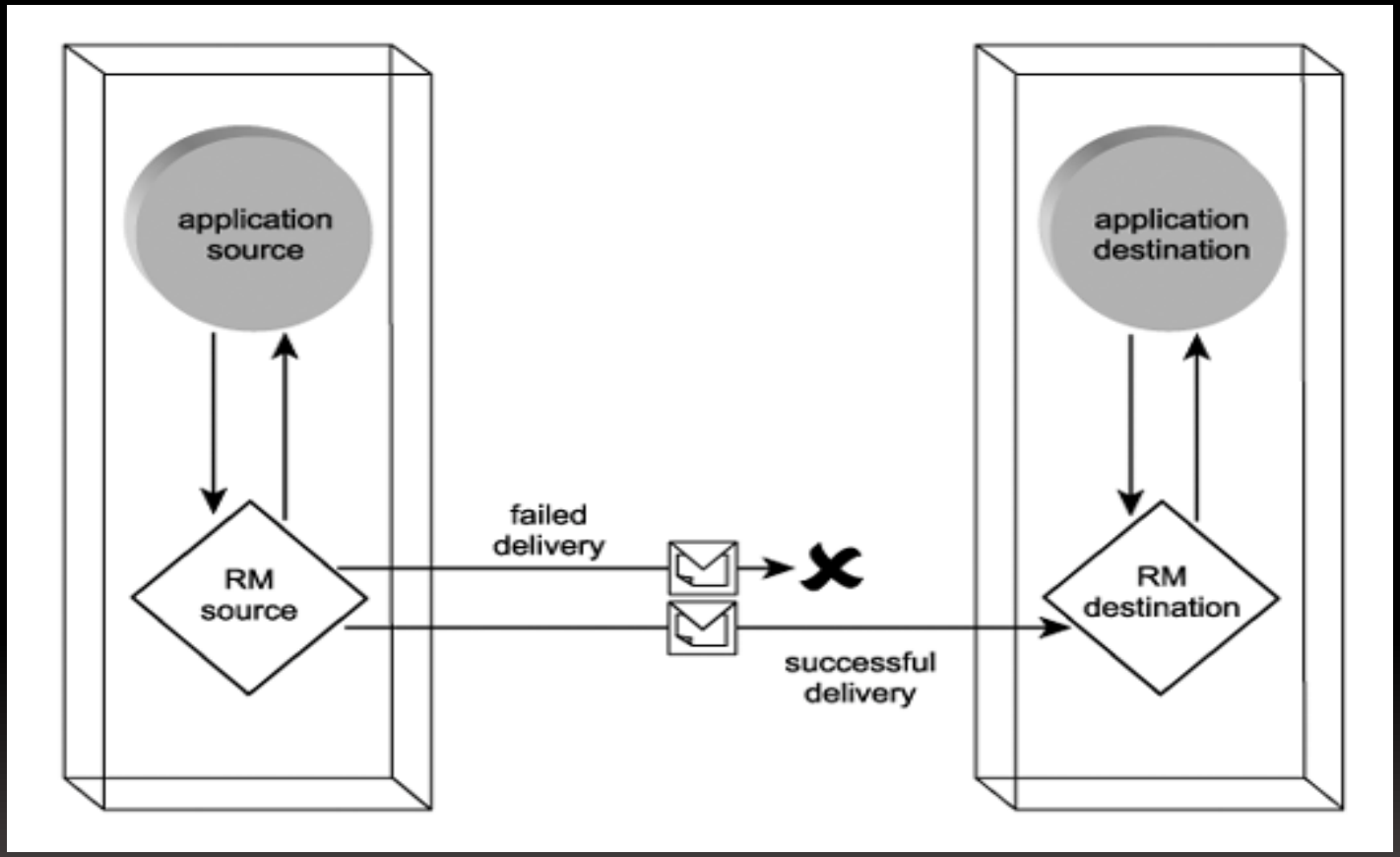
The AtLeastOnce delivery assurance



The AtLeastOnce delivery assurance allows a message to be delivered once or several times.

The delivery of zero messages creates an error condition

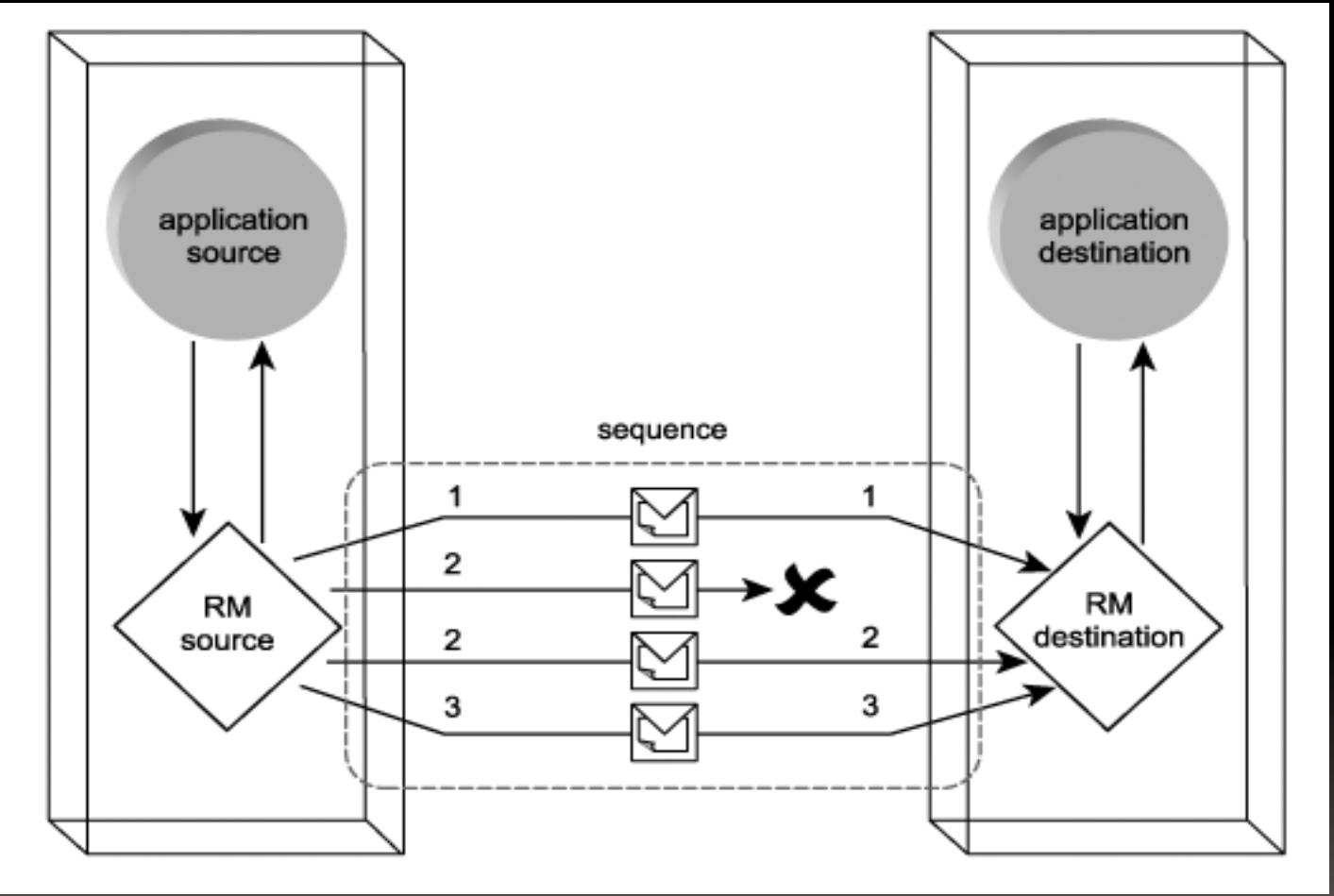
The ExactlyOnce delivery assurance



The ExactlyOnce delivery assurance guarantees that a message only will be delivered once.

An error is raised if zero or duplicate messages are delivered

The InOrder delivery assurance



The InOrder delivery assurance is used to ensure that messages are delivered in a specific sequence . The delivery of messages out of sequence triggers an error

WEB SERVICES AND CONTEMPORARY SOA
(PART II: ADVANCED MESSAGING, METADATA, AND SECURITY)

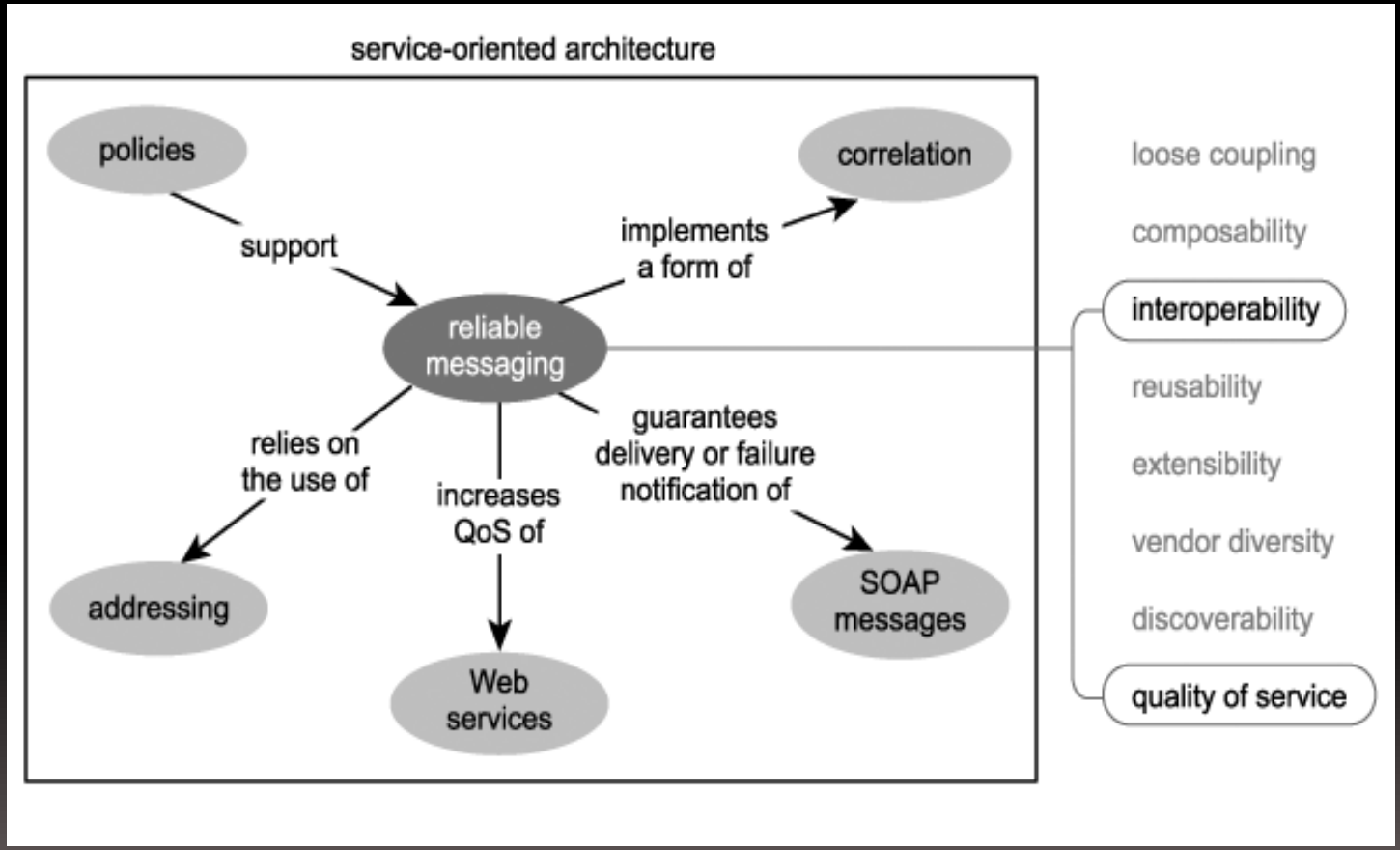
Reliable messaging

Reliable messaging and SOA

- Reliable messaging brings to service-oriented solutions a tangible quality of service .
- It introduces a flexible system that guarantees the delivery of message sequences supported by comprehensive fault reporting.
- This elevates the robustness of SOAP messaging implementations and eliminates the reliability concerns most often associated with any messaging frameworks.

Reliable messaging

Reliable messaging relating to other parts of SOA



Correlation

- One of the fundamental requirements for exchanging information via Web services is the ability to persist context and state across the delivery of multiple messages.

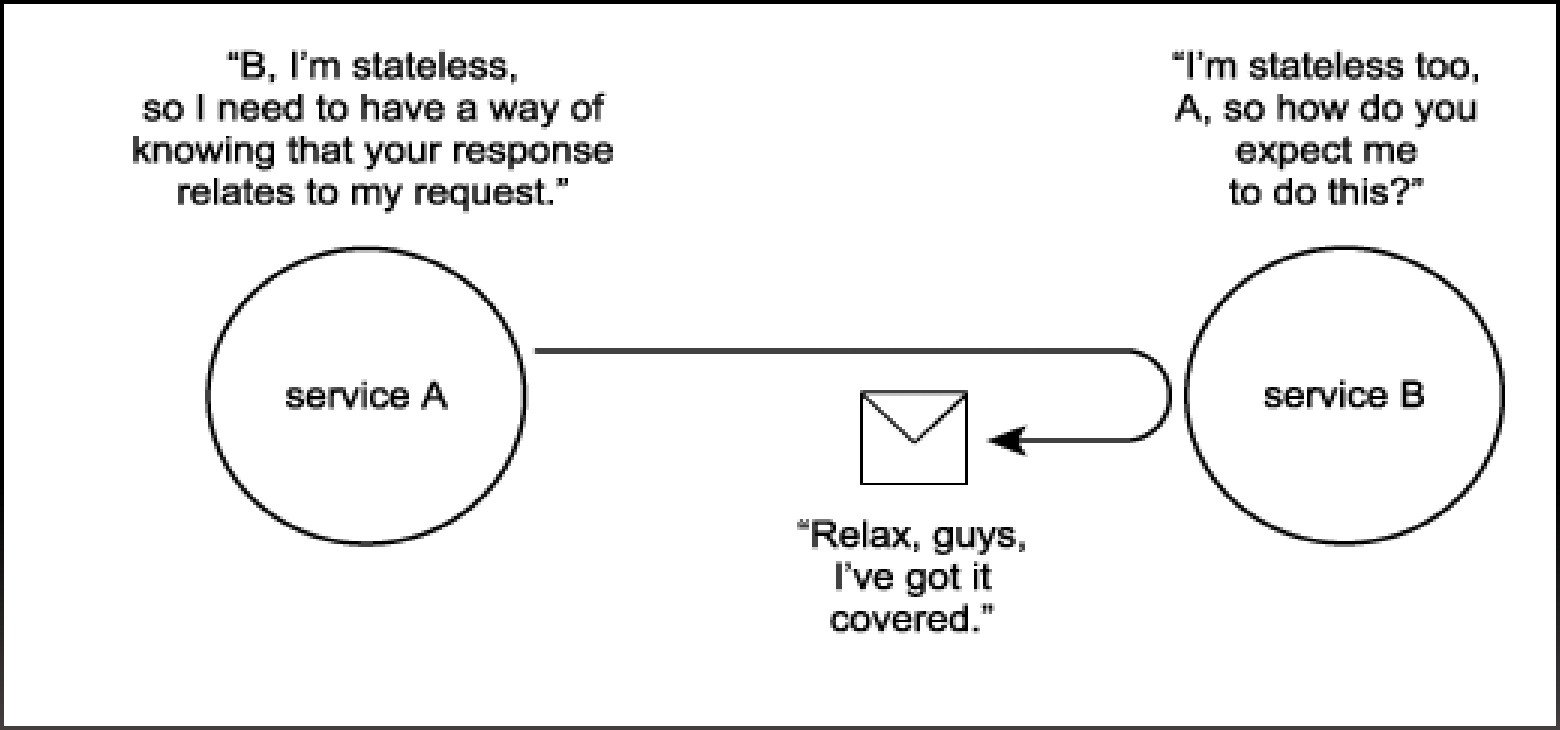
- Because a service-oriented communications framework is inherently loosely coupled, there is no intrinsic mechanism for associating messages exchanged under a common context or as part of a common activity.

- Even the execution of a simple request-response message exchange pattern provides no built-in means of automatically associating the response message with the original request.

- Correlation addresses this issue by requiring that related messages contain some common value that services can identify to establish their relationship with each other or with the overall task they are participating in .

Correlation

Correlation places the association of message exchanges into the hands of the message itself



Correlation

In Plain English

- To encourage repeat business, we introduce a promotion where, after ten visits to our car wash, your eleventh visit is free.
- We implement this promotion through the use of a punch card.
- Every time a customer drives in, we punch the driver's card.
- This card associates the current visit with all of the previous visits.
- Essentially, the punch card provides us with a form of correlation.
- Without it we would have a very hard time remembering which customers had visited us before.

Correlation

Correlation in abstract

- The technology that enabled tightly bound communication between components, databases, and legacy applications typically established an active connection that persisted for the duration of a given business activity (or longer).
- Because the connection remained active, context was inherently present, and correlation between individual transmissions of data was intrinsically managed by the technology protocol itself.
- When one stateless service sends a message to another, it loses control of the message and preserves no context of the activity in which the message is participating.
- It is up to the message to introduce the concept of correlation to provide services with the ability to associate a message with others.
- This is achieved by embedding a value in the message that is propagated to all related messages.

Correlation

Correlation in abstract

- When a service processes a message and locates this value, it can establish a form of context, in that it can be used to associate this message with others.
- The nature of the context can vary.
- For example, a message could be part of a simple exchange activity, an atomic transaction, or a long running orchestration

Correlation in MEPs and activities

- Because they are generic and non-business-specific in nature, MEPs and activities have no predefined notion of correlation.
- They are simple, conceptual building blocks incorporated and assembled by either custom-developed solutions that employ custom correlation identifiers and related processing logic or by specifications that impose proprietary forms of correlation

Correlation

Correlation in coordination

- The context management framework provided by WS-Coordination establishes a sophisticated mechanism for propagating identifiers and context information between services.
- A separate activation service is responsible for creating new activities and subsequently generating and distributing corresponding context data. S
- ervices can forward this information to others that can use it to register for participation in the activity.

- While context management uses a correlation-type identifier to uniquely represent a specific activity context, it goes well beyond correlation features to provide a comprehensive context management framework that can be leveraged through activity protocols, such as those supplied by the WS-AtomicTransaction and WS-BusinessActivity extensions.

Correlation

Correlation in orchestration

- WS-BPEL orchestrations need to concern themselves with the correlation of messages between process and partner services.
- This involves the added complexity of representing specific process instances within the correlation data.
- Further complicating this scenario is the fact that a single message may participate in multiple contexts, each identified by a separate correlation value.
- To facilitate these requirements, the WS-BPEL specification defines specific syntax that allows for the creation of extensible correlation sets.
- These message properties can be dynamically added, deleted, and altered to reflect a wide variety of message exchange scenarios and environments.

Correlation in addressing

WS-Addressing's message id and relationship MI headers provide inherent correlation abilities, which can be leveraged by many composition and messaging extensions.

Correlation

Correlation in reliable messaging

- Every message that participates in a WS-ReliableMessaging sequence carries sequence information with it.
- This data consists of a sequence identifier that represents the series of messages required to follow the messaging rules of the sequence, along with a message identifier that identifies how the current message fits into the overall sequence.
- As a whole, this information can be considered correlation-related.
- However, its primary purpose is to support the enforcement of reliability rules.

Correlation and SOA

- Correlation is a key contributor to preserving service autonomy and statelessness.