Service-Oriented Architecture

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

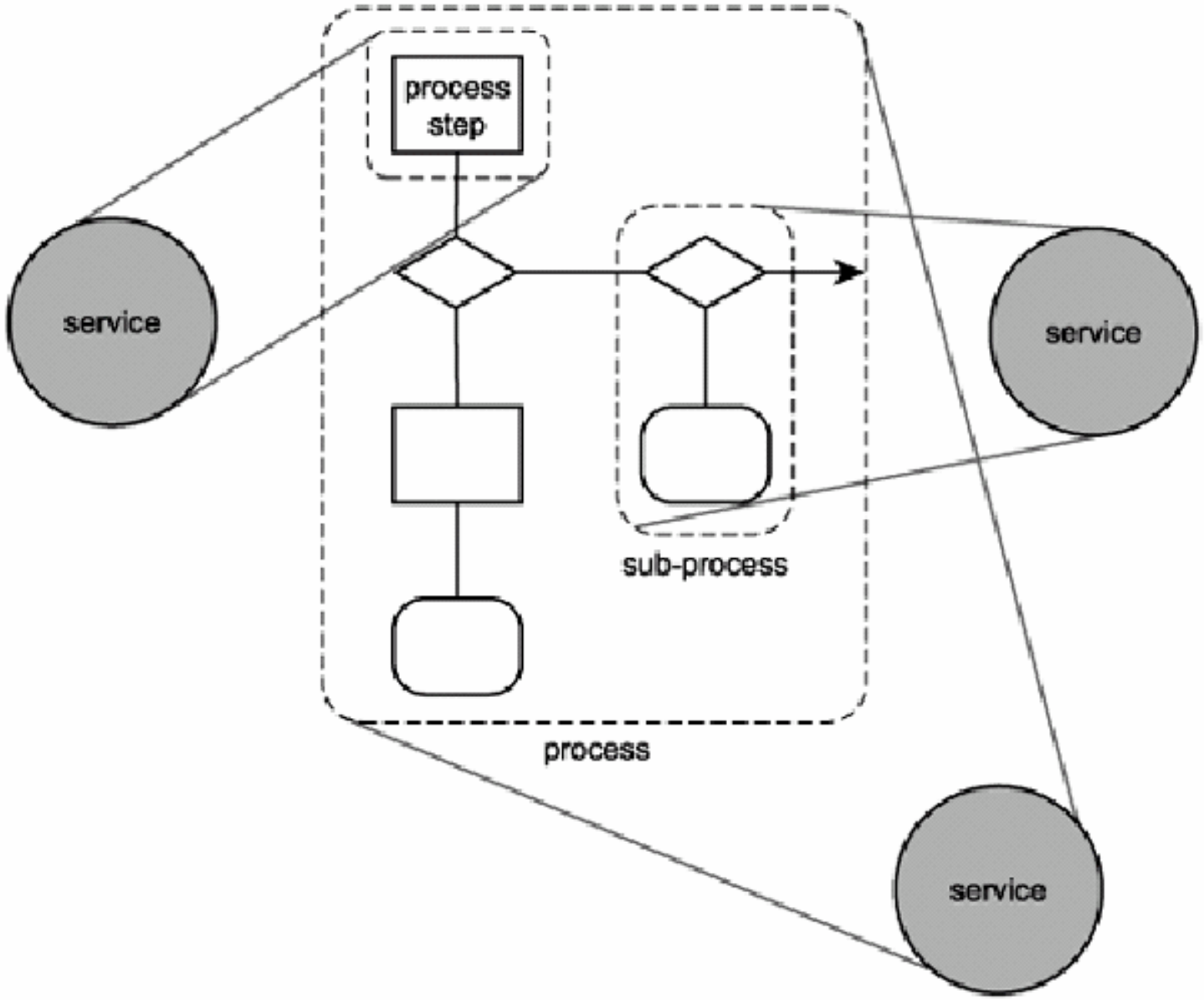
- “Because the term "service-oriented" has existed for some time, it has been used in different contexts and for different purposes. One constant through its existence has been that it represents a distinct approach for separating concerns. What this means is that logic required to solve a large problem 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.
- This approach transcends technology and automation solutions. It is an established and generic theory that can be used to address a variety of problems. What distinguishes the service-oriented approach to separating concerns is the manner in which it achieves separation.”

- “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.
- Note that 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.”

- Though we encourage independence within our business outlets, we must still ensure that they agree to adhere to certain baseline conventions for example, a common currency for the exchange of goods and services, a building code that requires signage to conform to certain parameters or perhaps a requirement that all employees speak the same language as the native consumers. These conventions standardize key aspects of each business for the benefit of the consumers without significantly imposing on the individual business's ability to exercise self-governance.
- Similarly, 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.

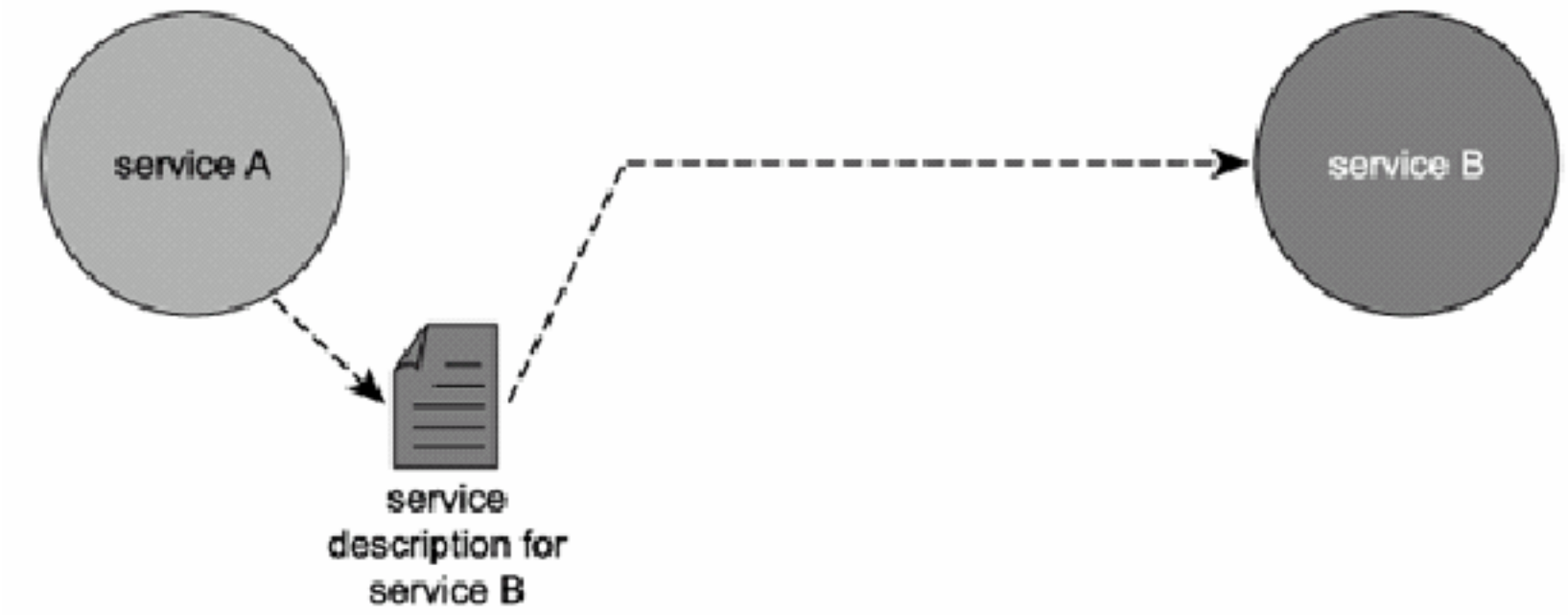
How services encapsulate logic

- 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.
- The concern addressed by a service can be small or large. Therefore, the size and scope of the logic represented by the service can vary. Further, service logic can encompass the logic provided by other services. In this case, one or more services are composed into a collective.


process step
service
service
service
sub-process
process

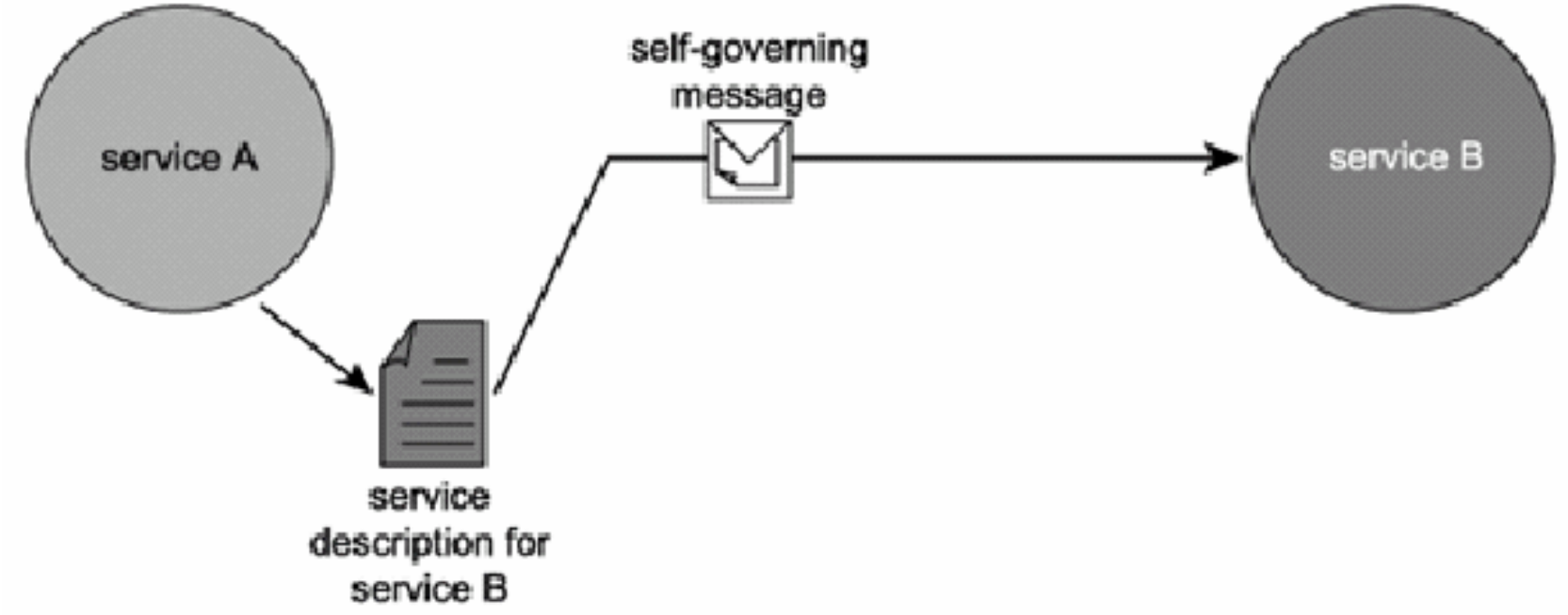
How services relate

- Within SOA, services can be used by other services or other programs. Regardless, the 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. The manner in which services use service descriptions results in a relationship classified as loosely coupled. For example, Next figure illustrates that service A is aware of service B because service A is in possession of service B's service description.
- For services to interact and accomplish something meaningful, they must exchange information. A communications framework capable of preserving their loosely coupled relationship is therefore required. One such framework is messaging.


service A
service B
service
description for
service B

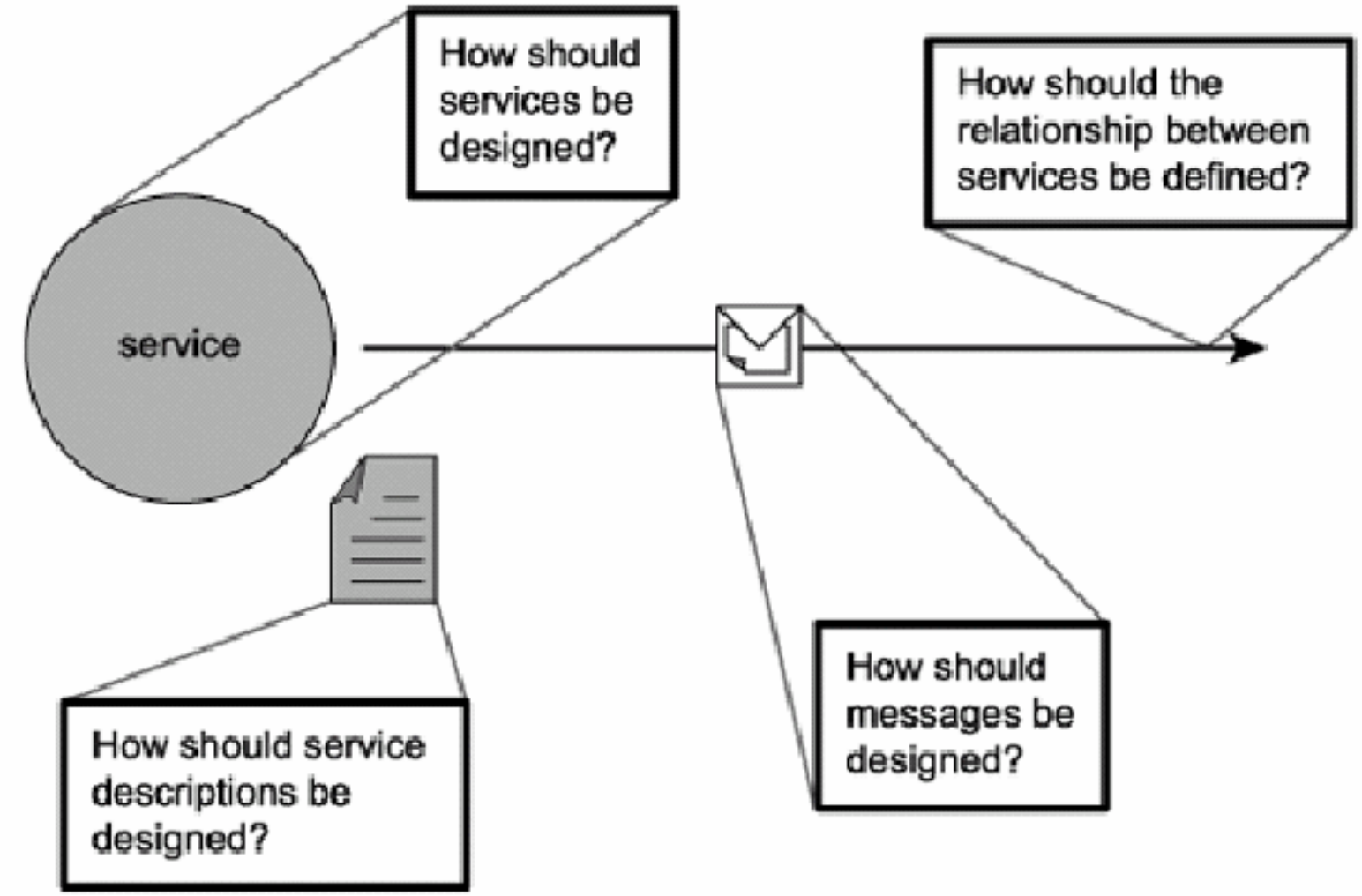
How services communicate

- After a service sends a message on its way, it loses control of what happens to the message thereafter.
- That is why we require messages to exist as "independent units of communication."
- This means that messages, like services, should be autonomous.
- To that effect, messages can be outfitted with enough intelligence to self-govern their parts of the processing logic.


service A
self-governing
message
service B
service
description for
service B

How services are designed

- Much like object-orientation, service-orientation has become a distinct design approach which introduces commonly accepted principles that govern the positioning and design of our architectural components.


How should services be designed?
How should the relationship between services be defined?
service
How should service descriptions be designed?
How should messages be designed?

How services are designed

- The application of service-orientation principles to processing logic results in standardized service-oriented processing logic. When a solution is comprised of units of service-oriented processing logic, it becomes what we refer to as a service-oriented solution.
- For the purpose of providing a preliminary introduction, let's highlight some of the key aspects of these **SOA principles** here:
  - **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.

How services are designed

- **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.
- **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.

How services are designed

- With a knowledge of the components that comprise our basic architecture and a set of design principles we can use to shape and standardize these components, all that is missing is an implementation platform that will allow us to pull these pieces together to build service-oriented automation solutions.
- The Web services technology set offers us such a platform. With a knowledge of the components that comprise our basic architecture and a set of design principles we can use to shape and standardize these components, all that is missing is an implementation platform that will allow us to pull these pieces together to build service-oriented automation solutions. The Web services technology set offers us such a platform.

How services are built

- As we mentioned earlier, the term "service-oriented" and various abstract SOA models existed before the arrival of Web services.
- However, no one technology advancement has been so suitable and successful in manifesting SOA than Web services.
- All major vendor platforms currently support the creation of service-oriented solutions, and most do so with the understanding that the SOA support provided is based on the use of Web services.
- Therefore, while we fully acknowledge that achieving SOA does not require Web services, the focus is on how SOA can and should be realized through the use of the Web services technology platform.

Primitive SOA

- The past few sections have described the individual ingredients for what we call primitive SOA.
- It is labeled as such because it represents a baseline technology architecture that is supported by current major vendor platforms.
- All forms of SOA we explore from here on are based on and extend this primitive model. Some of the extensions we discuss are attainable today through the application of advanced design techniques, while others rely on the availability of pre-defined Web services specifications and corresponding vendor support.
- *Refer: case study 01*

Summary of Key Points

- SOA and service-orientation are implementation-agnostic paradigms that can be realized with any suitable technology platform.
- Our primitive SOA model represents a mainstream variation of SOA based solely on Web services and common service-orientation principles.
- Throughout the discussions, any reference to the term "SOA" implies the primitive SOA model.

Common characteristics of 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 we refer to as contemporary SOA.
- Contemporary SOA builds upon the primitive SOA model by leveraging industry and technology advancements to further its original ideals. Though the required implementation technology can vary, contemporary SOAs have evolved to a point where they can be associated with a set of common characteristics.

Common characteristics of contemporary SOA

Specifically, we explore the following primary characteristics:

- 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.
- Contemporary SOA fosters intrinsic interoperability.
- Contemporary SOA promotes discovery.
- Contemporary SOA promotes federation.

Common characteristics of contemporary SOA

- Contemporary SOA promotes architectural composability.
- Contemporary SOA fosters inherent reusability.
- 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.

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

- Many argue that the manner in which SOA is used to qualify products, designs, and technologies elevates this term beyond one that simply relates to architecture.
- SOA, some believe, has become synonymous with an entire new world application computing platform.
- Because we positioned contemporary SOA as building upon and extending the primitive SOA model, we already have a starting point for our definition:
- *Contemporary SOA represents an architecture that promotes service-orientation through the use of Web services.*

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

- With SOA, however, the actual acronym has become a multi-purpose buzzword used frequently when discussing an application computing platform consisting of Web services technology and service-orientation principles.
- Because the acronym already represents the word "architecture" we are unfortunately subjected to statements that can be confusing.
- Perhaps the best way to view it is that if a product, design, or technology is prefixed with "SOA," it is something that was (directly or indirectly) created in support of an architecture based on service-orientation principles.

Contemporary SOA increases quality of service

- There is a definite need to bring SOA to a point where it can implement enterprise-level functionality as safely and reliably as the more established distributed architectures already do.
- This relates to common quality of service requirements, such as:
  - 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.

Contemporary SOA increases quality of service

- 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.

- Contemporary SOA is striving to fill the QoS gaps of the primitive SOA model. Many of the concepts and specifications will be discussed in “SOA and WS-* Extensions”, which provide features that directly address quality of service requirements.
- For lack of a better term, we'll refer to an SOA that fulfills specific quality of service requirements as "QoS-capable."

Contemporary SOA is fundamentally autonomous

- The service-orientation principle of autonomy requires that individual services be as independent and self-contained as possible with respect to the control they maintain over their underlying logic.
- This is further 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.
- SOA builds upon and expands this principle by promoting the concept of autonomy throughout solution environments and the enterprise. Applications comprised of autonomous services, for example, can themselves be viewed as composite, self-reliant services that exercise their own self-governance within service-oriented integration environments.

Contemporary SOA is fundamentally autonomous

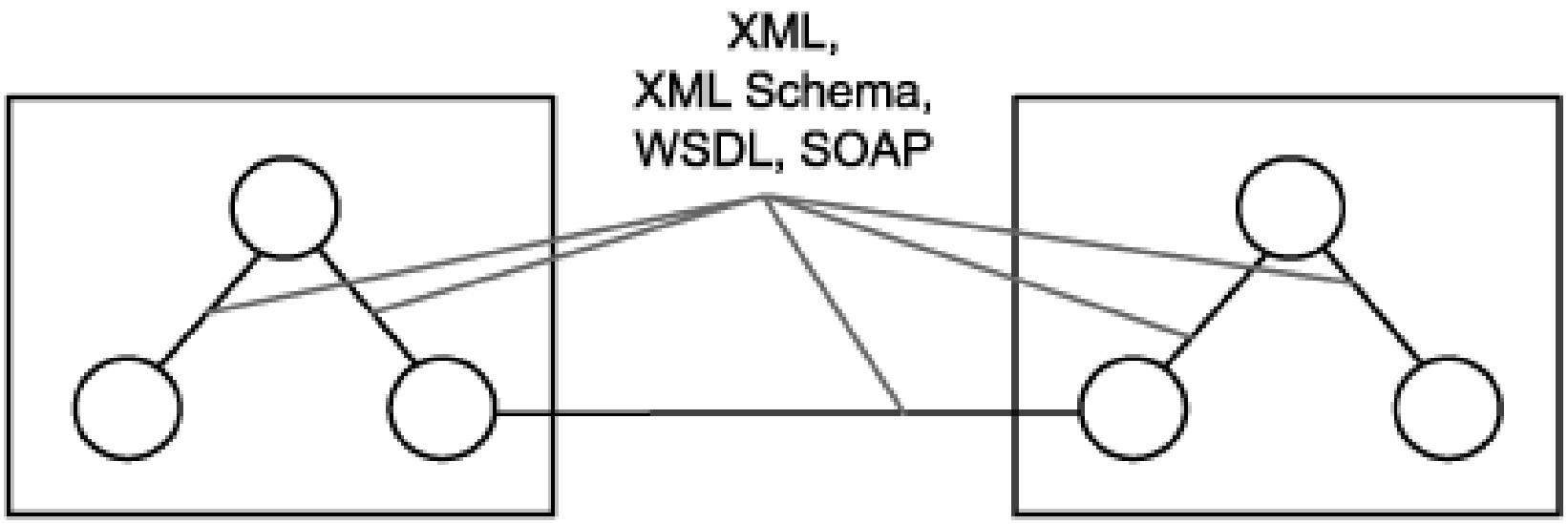
- Later we explain how by creating service abstraction layers, entire domains of solution logic can achieve control over their respective areas of governance.
- This establishes a level of autonomy that can cross solution boundaries.

Contemporary SOA is based on open standards

- Perhaps 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.
- Further, 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, 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.

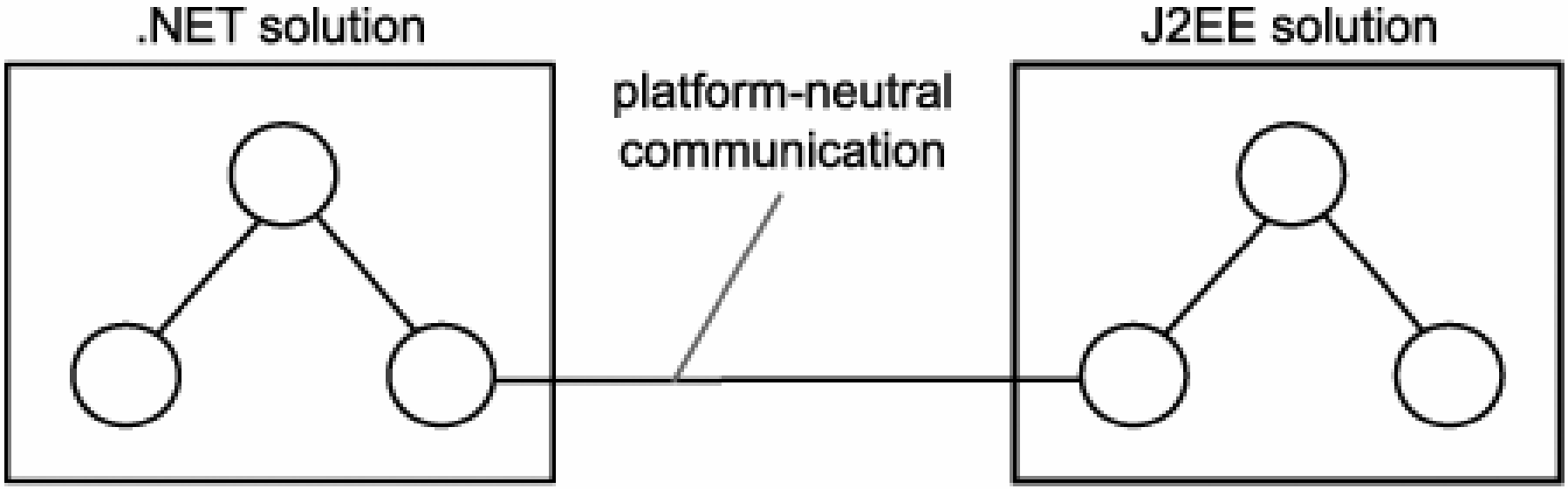
Contemporary SOA is based on open standards

- Contemporary SOAs fully leverage and reinforce this open, vendor-neutral communications framework in the next figure.
- 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.


XML,
XML Schema,
WSDL, SOAP

Contemporary SOA supports vendor diversity

- The open communications framework explained in the previous section not only 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.
- For example, regardless of how proprietary a development environment is, as long as it supports the creation of standard Web services, it can be used to create a non-proprietary service interface layer, opening up interoperability opportunities with other, service-capable applications.
- This, incidentally, has changed the face of integration architectures, which now can encapsulate legacy logic through service adapters, and leverage middleware advancements based on Web services.


.NET solution
platform-neutral
communication
J2EE solution

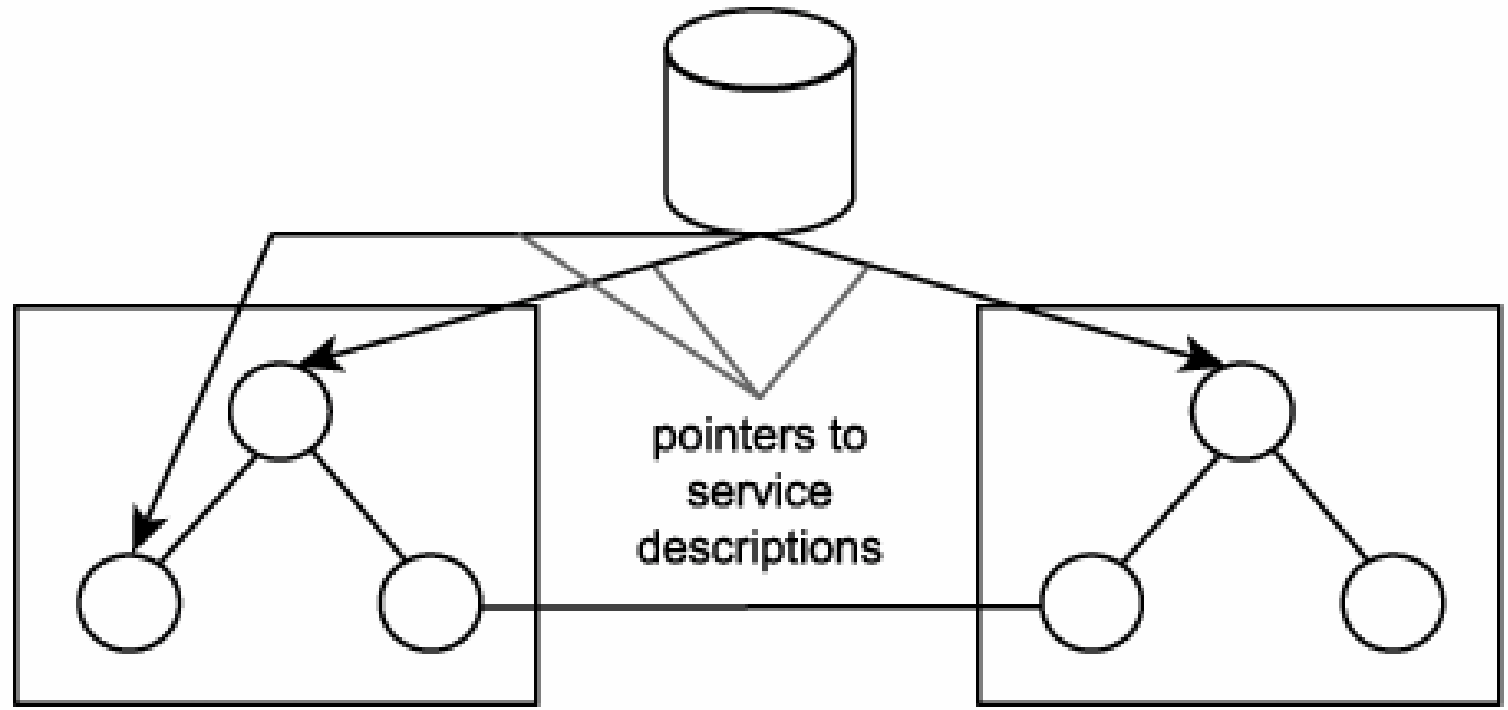
Contemporary SOA supports vendor diversity

- Organizations can certainly continue building solutions with existing development tools and server products. In fact, it may make sense to do so, only to continue leveraging the skill sets of in-house resources.
- However, the choice to explore the offerings of new vendors is always there. This option 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.

Contemporary SOA promotes discovery

- Even though the first generation of Web services standards included UDDI, few of the early implementations actually used service registries as part of their environments. This may have to do with the fact that not enough Web services were actually built to warrant a registry.
- However, another likely reason is that the concept of service discovery was simply not designed into the architecture. When utilized within traditional distributed architectures, Web services were more often employed to facilitate point-to-point solutions. Therefore, discovery was not a common concern.
- SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond. A serious SOA will likely rely on some form of service registry or directory to manage service descriptions

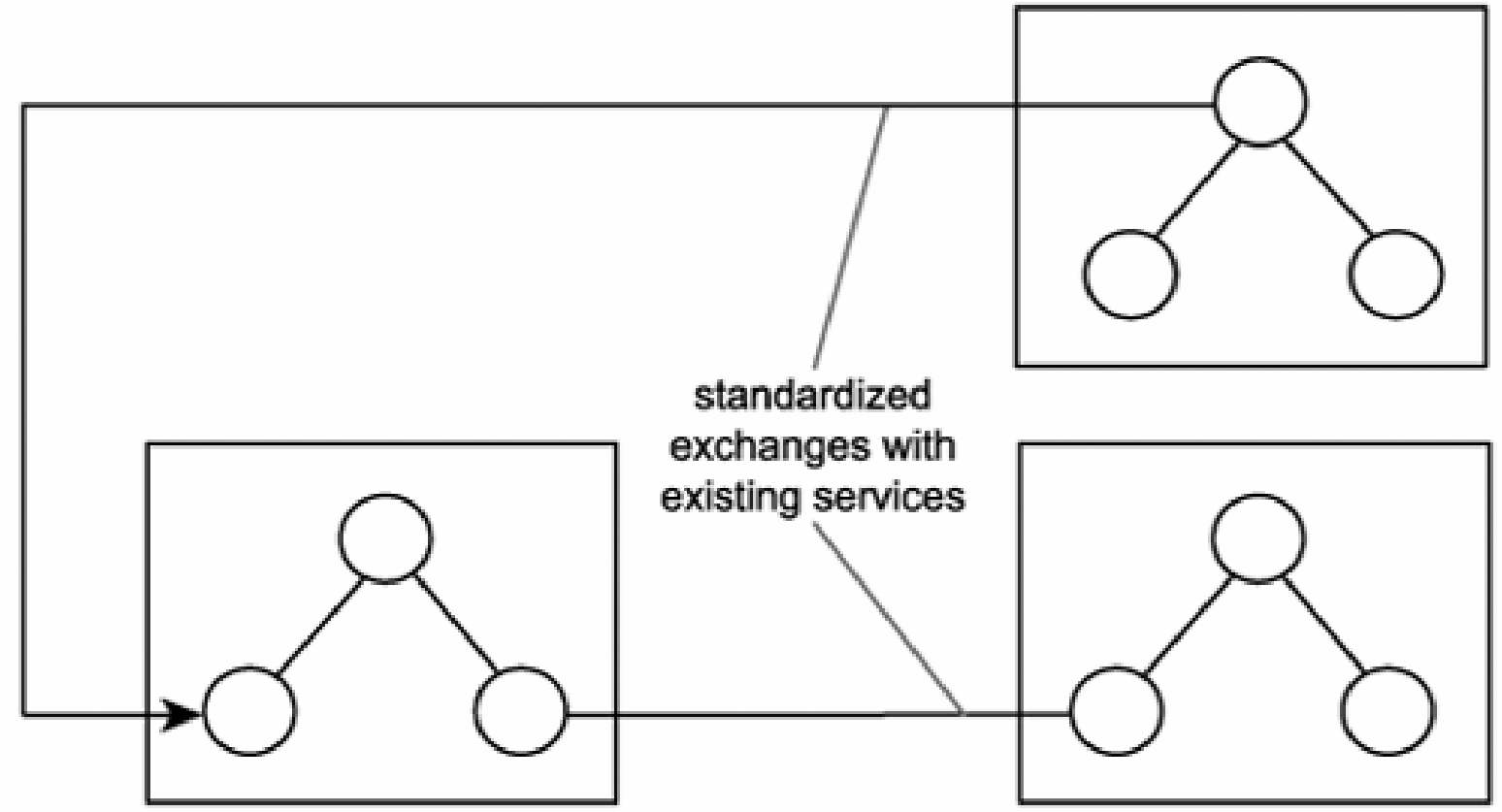
Registries enable a mechanism for the discovery of services



Contemporary SOA fosters intrinsic interoperability

- Further leveraging and supporting the required usage of open standards, a vendor diverse environment, and the availability of a discovery mechanism, is the concept of intrinsic interoperability.
- Regardless of whether an application actually has immediate integration requirements, design principles can be applied to outfit services with characteristics that naturally promote interoperability.
- When building an SOA application from the ground up, services with intrinsic interoperability become potential integration endpoints.
- When properly standardized, this leads to service-oriented integration architectures wherein solutions themselves achieve a level of intrinsic interoperability. Fostering this characteristic can significantly alleviate the cost and effort of fulfilling future cross-application integration requirements.

Intrinsically interoperable services enable unforeseen integration opportunities



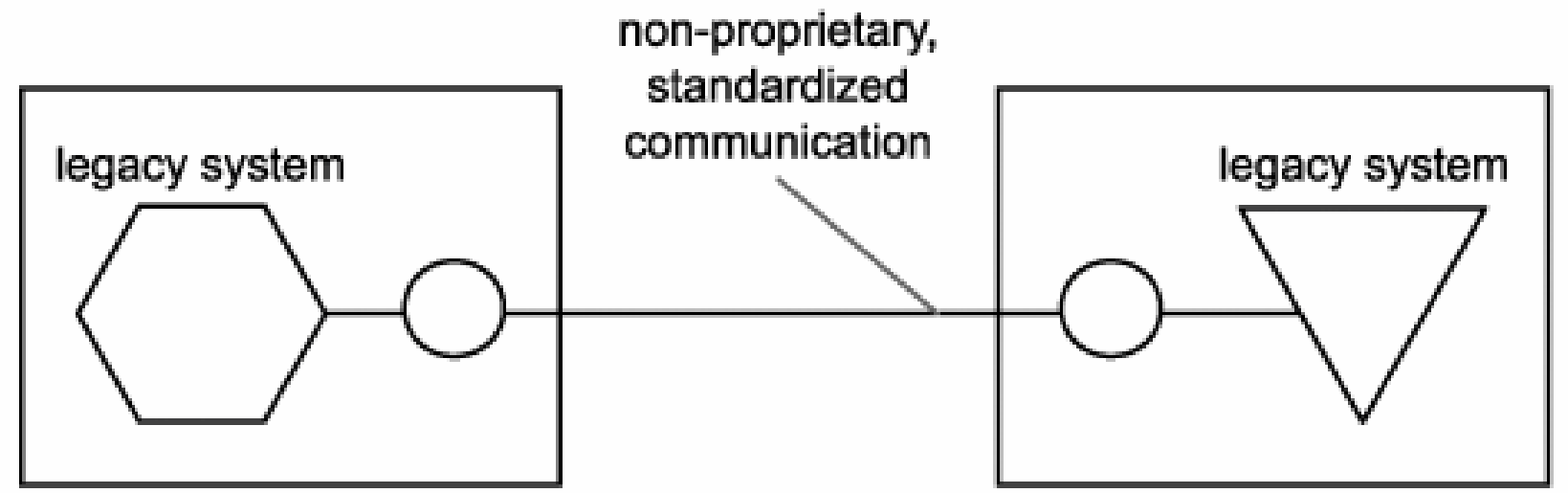
Contemporary SOA promotes federation

- Establishing SOA within an enterprise does not necessarily require that you replace what you already have. One of the most attractive aspects of this architecture is its ability to introduce unity across previously non-federated environments.
- While Web services enable federation, SOA promotes this cause 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 (also supported by an extensive adapter technology marketplace).

Contemporary SOA promotes federation

- Obviously, the incorporation of SOA with previous platforms can lead to a variety of hybrid solutions.
- However, the key aspect is that the communication channels achieved by this form of service-oriented integration are all uniform and standardized

Services enable standardized federation of disparate legacy systems



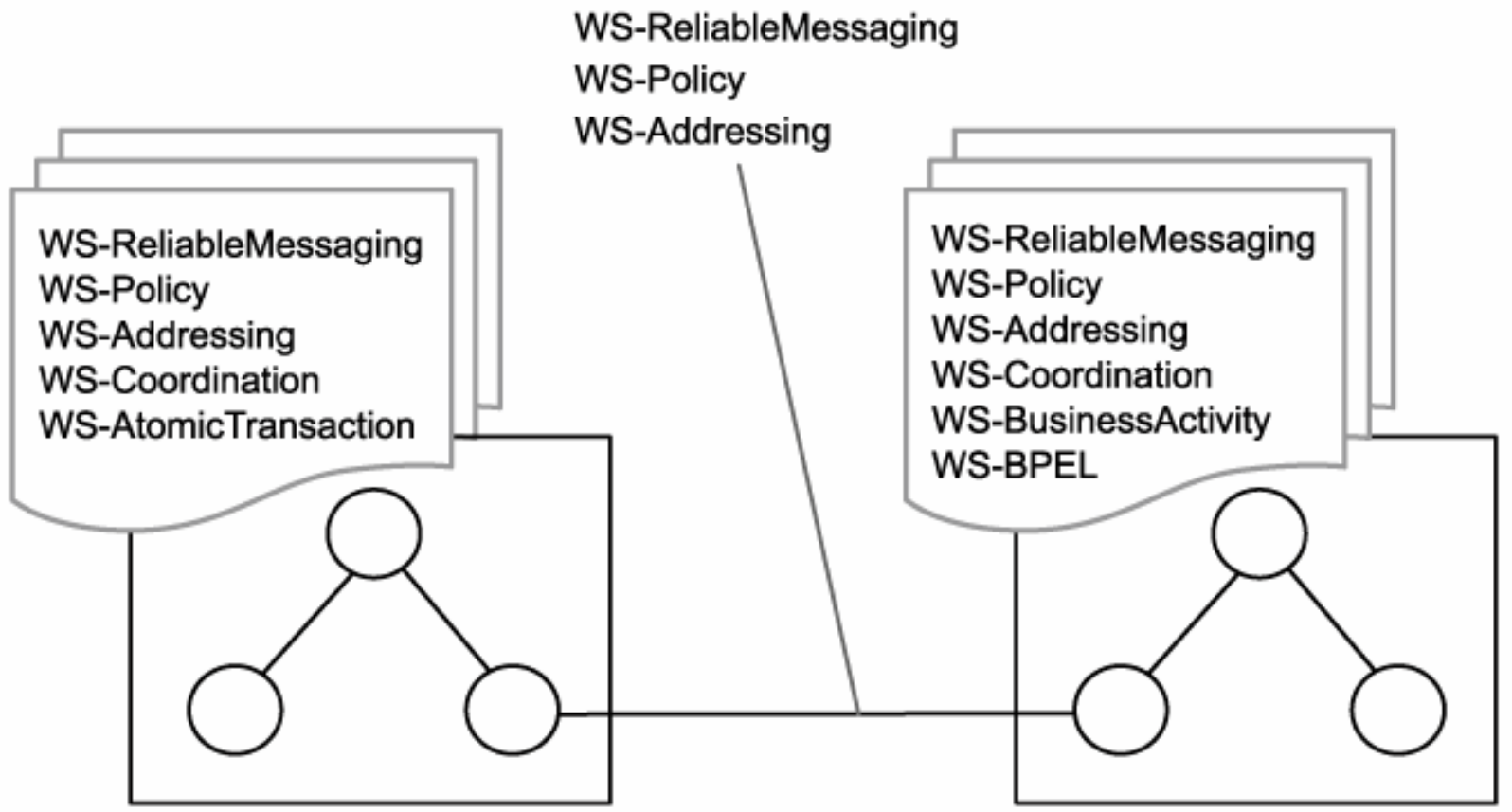
Contemporary SOA promotes architectural composability

- Composability is a deep-rooted characteristic of SOA that can be realized on different levels. For example, by fostering the development and evolution of composable services, SOA supports the automation of flexible and highly adaptive business processes. As previously mentioned, services exist as independent units of logic. A business process can therefore be broken down into a series of services, each responsible for executing a portion of the process.
- A broader example of 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.

Contemporary SOA promotes architectural composability

- What provides this flexibility is the fact that second-generation Web services specifications are being designed specifically to leverage the SOAP messaging model. Individual specifications consist of modular extensions that provide one or more specific features.
- As the offering of WS-* extensions supported by a given vendor platform grows, the flexibility to compose allows you to continue building solutions that only implement the features you actually need.
- In other words, the WS-* platform allows for the creation of streamlined and optimized service-oriented architectures, applications, services, and even messages.
- With respect to our definition, let's represent this characteristic by describing the architecture as a whole as being composable. This represents both composable services, as well as the extensions that comprise individual SOA implementations.

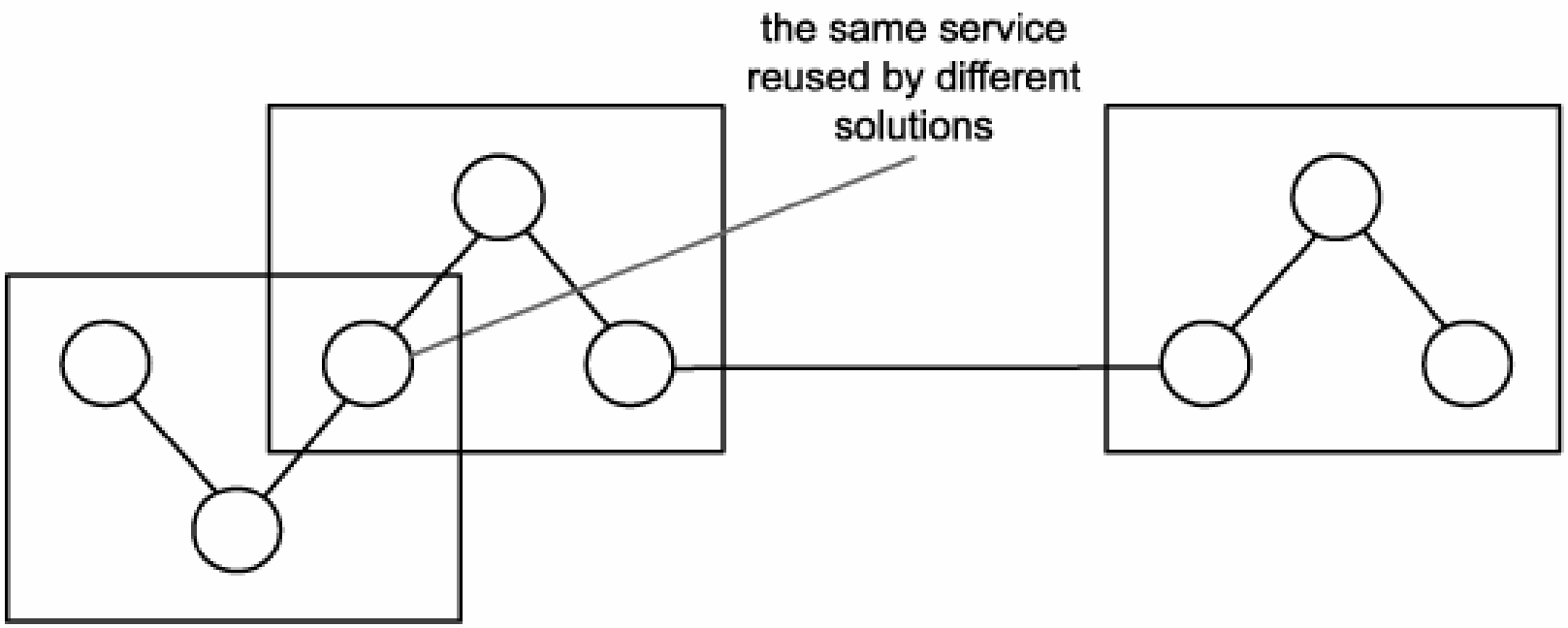
Different solutions can be composed of different extensions and can continue to interoperate as long as they support the common extensions required



Contemporary SOA fosters inherent reusability

- SOA establishes an environment that promotes reuse on many levels. For example, 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.
- The emphasis placed by SOA on the creation of services that are agnostic to both the business processes and the automation solutions that utilize them leads to an environment in which reuse is naturally realized as a side benefit to delivering services for a given project. Thus, inherent reuse can be fostered when building service-oriented solutions

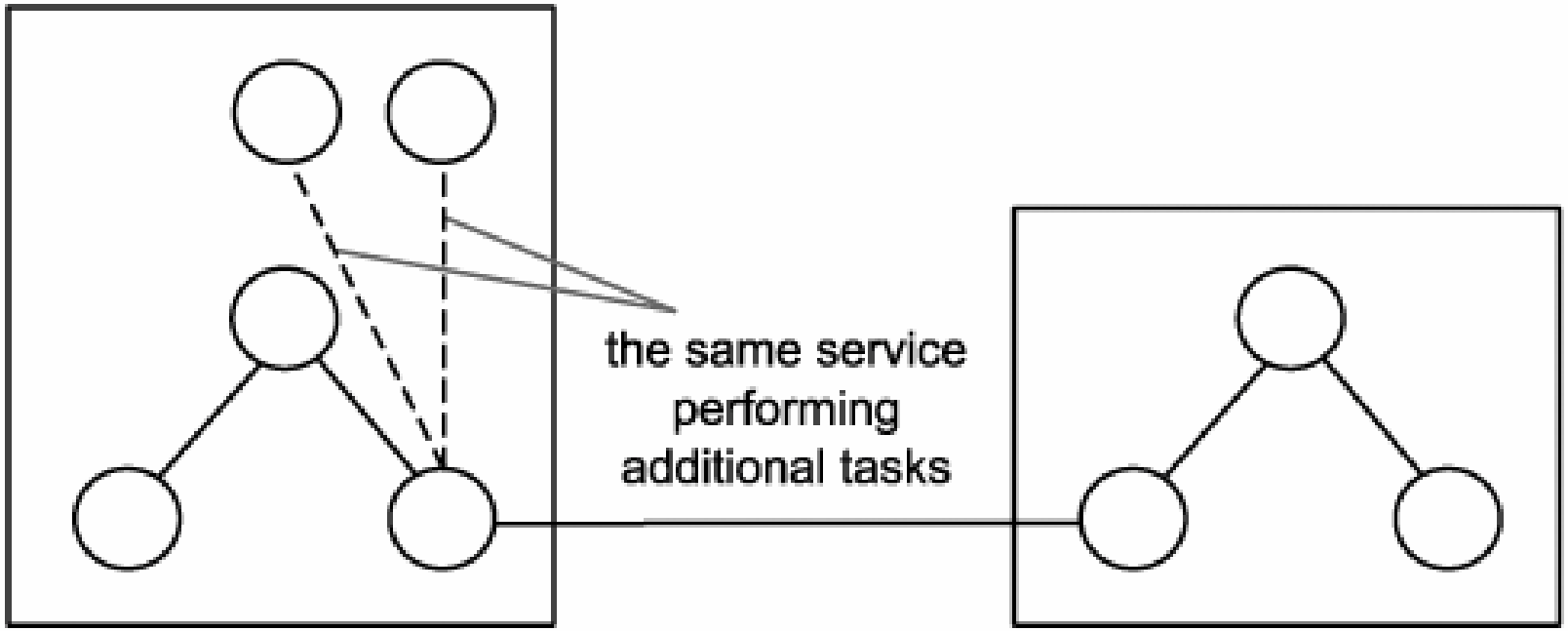
Inherent reuse accommodates unforeseen reuse opportunities



Contemporary SOA emphasizes extensibility

- When expressing encapsulated functionality through a service description, SOA encourages you to think beyond immediate, point-to-point communication requirements.
- 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.
- Extensibility is also a characteristic that is promoted throughout SOA as a whole. Extending entire solutions can be accomplished by adding services or by merging with other service-oriented applications (which also, effectively, "adds services"). Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.

Extensible services can expand functionality with minimal impact



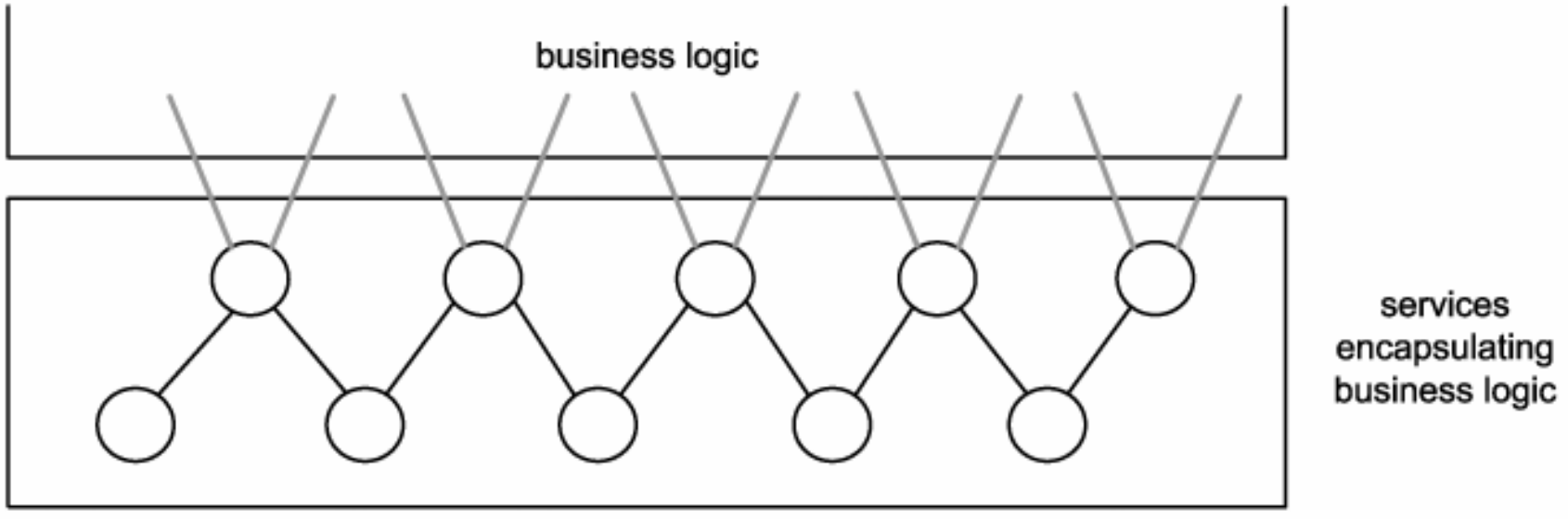
Contemporary SOA emphasizes extensibility

- Time to revisit our original definition to add a few adjectives that represent 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.*

Contemporary SOA supports a service-oriented business modeling paradigm

- In our description of a primitive SOA, we briefly explored how business processes can be represented and expressed through services. 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.

A collection (layer) of services encapsulating business process logic



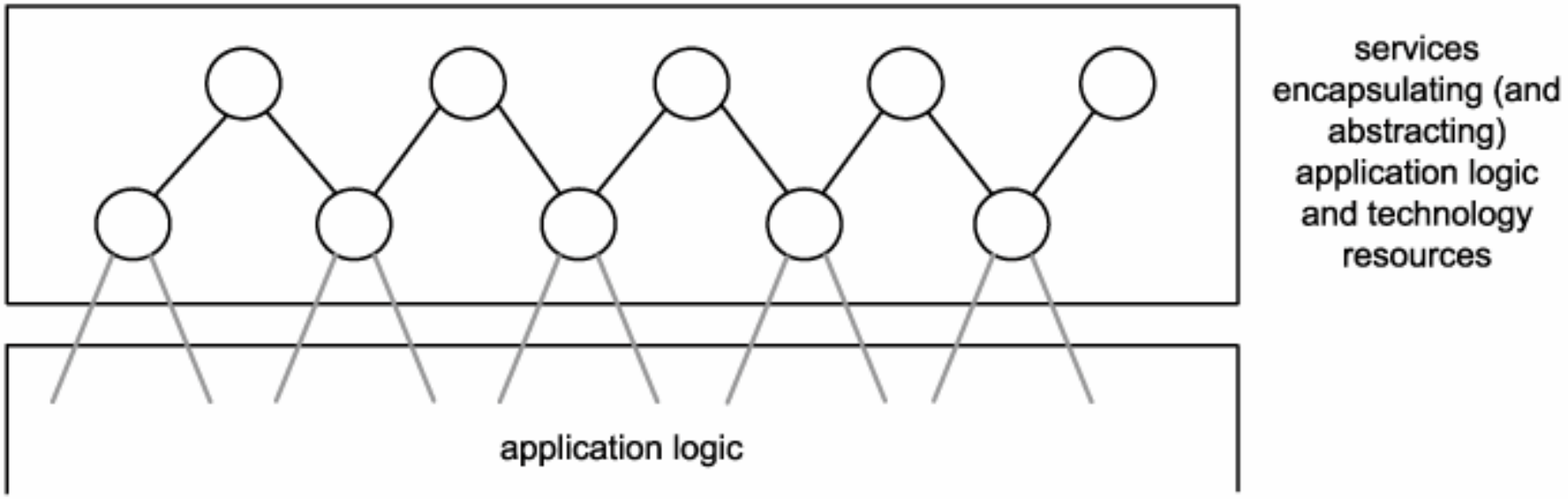
Contemporary SOA supports a service-oriented business modeling paradigm

- In other words, services can be designed to express business logic. BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services.
- This is an area of SOA that is not yet widely accepted or understood. We therefore spend a significant time in exploring the service-oriented business modeling paradigm.

Contemporary SOA implements layers of abstraction

- One of the characteristics that tends to evolve naturally through the application of service-oriented design principles is that 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.
- When applied through proper design, abstraction can be targeted at business and application logic. For example, 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.
- It is the mutual abstraction of business and technology that supports the service-oriented business modeling paradigm we discussed and further establishes the loosely coupled enterprise model explained in the following section.

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



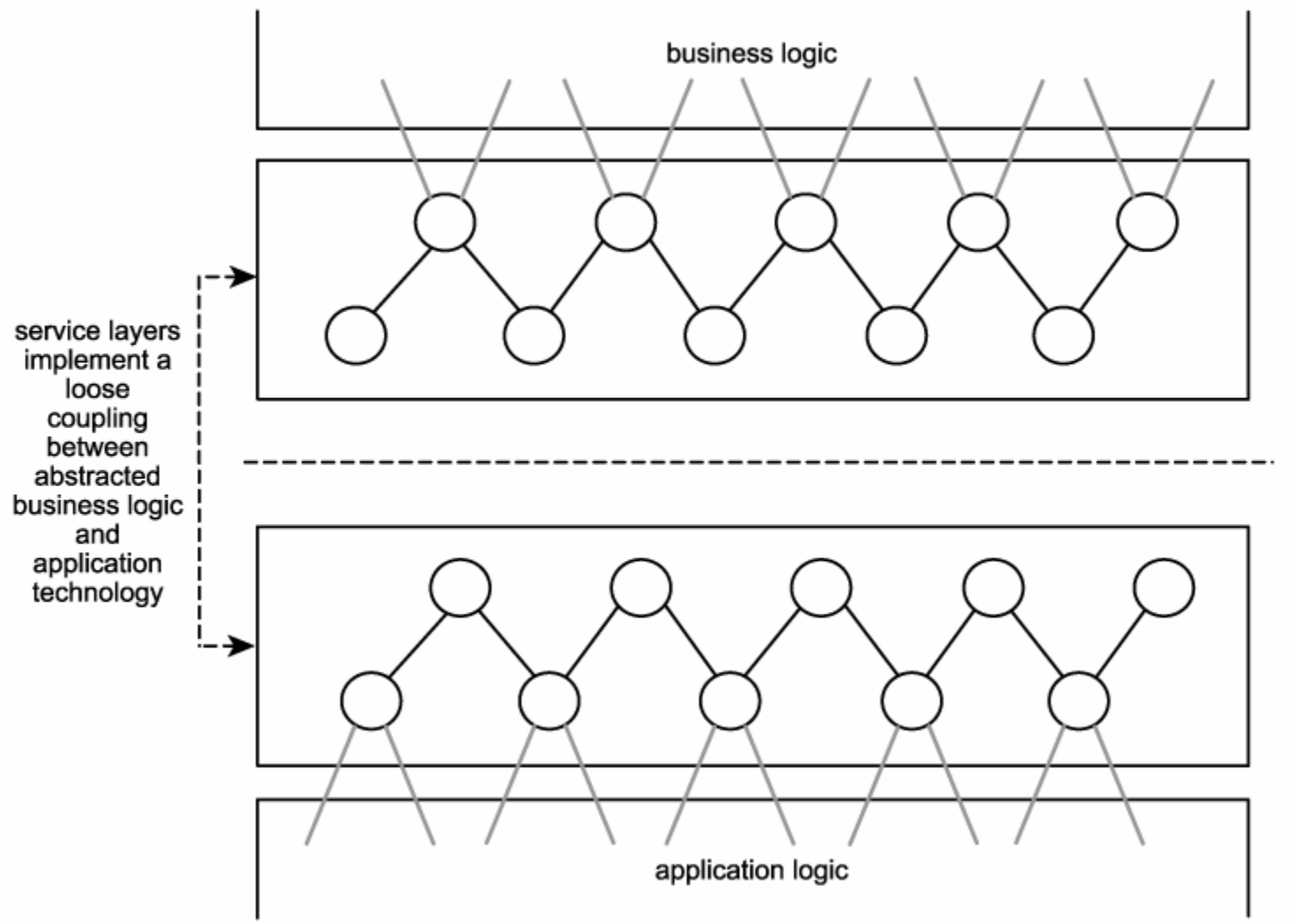
Contemporary SOA promotes loose coupling throughout the enterprise

- As we've established, a core benefit to building a technical architecture with loosely coupled services is the resulting independence of service logic. Services only require an awareness of each other, allowing them to evolve independently.
- Now, let's take a step back and look at the enterprise as a whole. Within an organization where service-orientation principles are applied to both business modeling and technical design, the concept of loose coupling is amplified.

Contemporary SOA promotes loose coupling throughout the enterprise

- 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 changea quality known as organizational agility.

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



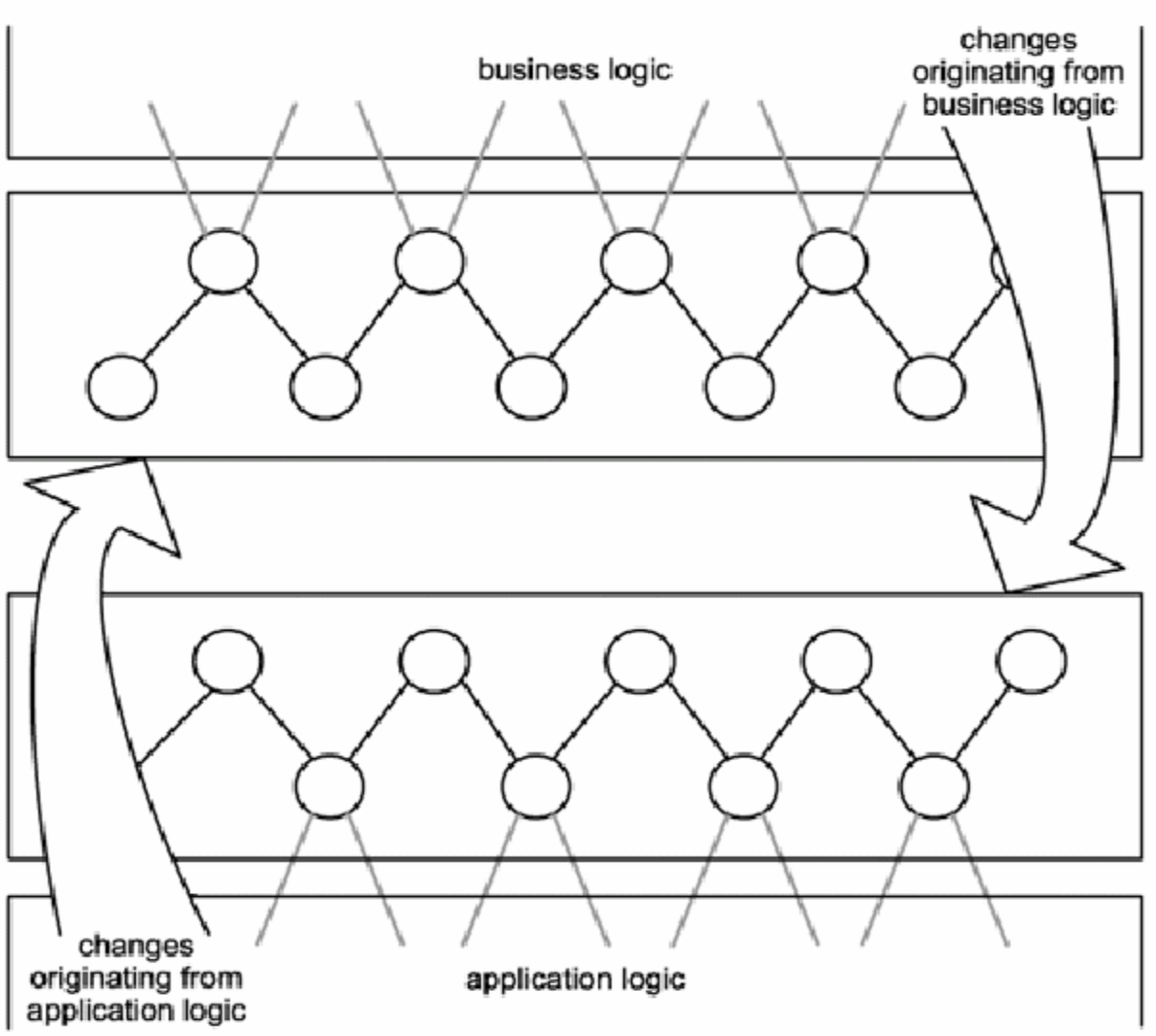
Contemporary SOA promotes organizational agility

- Whether the result of an internal reorganization, a corporate merger, a change in an organization's business scope, or the replacement of an established technology platform, 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.

Contemporary SOA promotes organizational agility

- 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

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



Contemporary SOA promotes organizational agility

- Other benefits realized through the standardization of SOA also contribute to minimizing dependencies and increasing overall responsiveness to change: notably, the intrinsic interoperability that can be built into services and the open communications framework established across integration architectures that enable interoperability between disparate platforms. Change imposed on any of these environments is more easily facilitated for the same reasons a loosely coupled state between services representing either ends of the communication channel.
- Organizational agility is perhaps the most significant benefit that can be realized with contemporary SOA.

Contemporary SOA is a building block

- A service-oriented application architecture will likely be one of several within an organization committed to SOA as the standard architectural platform. Organizations standardizing on SOA work toward an ideal 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 a part of this future-state environment, either as a standalone unit of business automation or as a service encapsulating some or all of the business automation logic. In responding to business model-level changes, SOAs can be augmented to change the nature of their automation, or they can be pulled into service-oriented integration architectures that require the participation of multiple applications.

Contemporary SOA is a building block

- What this all boils down to is that an individual service-oriented application can, in its entirety, be represented by and modeled as a single service. As mentioned earlier, there are no limits to the scope of service encapsulation. 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.
- This past set of characteristics has further broadened our definition. Let's append the definition with the following:
- *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.*

Contemporary SOA is an evolution

- SOA defines an architecture that is related to but still distinct from its predecessors. It differs from traditional client-server and distributed environments in that it is heavily influenced by the concepts and principles associated with service-orientation and Web services.
- It is similar to previous platforms in that it preserves the successful characteristics of its predecessors and builds upon them with distinct design patterns and a new technology set.
- For example, 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.

Contemporary SOA is still maturing

- While the characteristics described so far are fundamental to contemporary SOA, this point is obviously more of a subjective statement of where SOA is at the moment. Even though SOA is being positioned as the next standard application computing platform, this transition is not yet complete. Despite the fact that 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.
- Standards organizations and major software vendors have produced many specifications to address a variety of supplementary extensions. Additionally, the next generation of development tools and application servers promises to support a great deal of these new technologies. 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.

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. This will likely result in countless hybrid architectures, consisting mostly of distributed environments that are part legacy and part service-oriented.
- Further supporting this prediction is the evolving state of the technology set that is emerging to realize enterprise-level SOAs. As companies adopt SOA during this evolution, many will need to retrofit their environments (and their standards) to accommodate changes and innovations as SOA-related specifications, standards, and products continue to mature.

Defining SOA

- Now that we've finished covering characteristics, we can finalize our formal definition.
- *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.*

Defining SOA

- Though accurate, this definition of contemporary SOA is quite detailed. For practical purposes, let's provide a supplementary definition that can be applied to both primitive and contemporary SOA.
- *SOA is a form of technology architecture that adheres to the principles of service-orientation. When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise.*

Defining SOA: Separating concrete characteristics

- Looking back at the list of characteristics we just covered, we can actually split them into two groups characteristics that represent concrete qualities that can be realized as real extensions of SOA and those that can be categorized as commentary or observations.
- Collectively, these characteristics were useful for achieving our formal definition. From here on, though, we are more interested in exploring the concrete characteristics only.

Defining SOA: Separating concrete characteristics

- Let's therefore remove the following items from our original list:
- *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.*

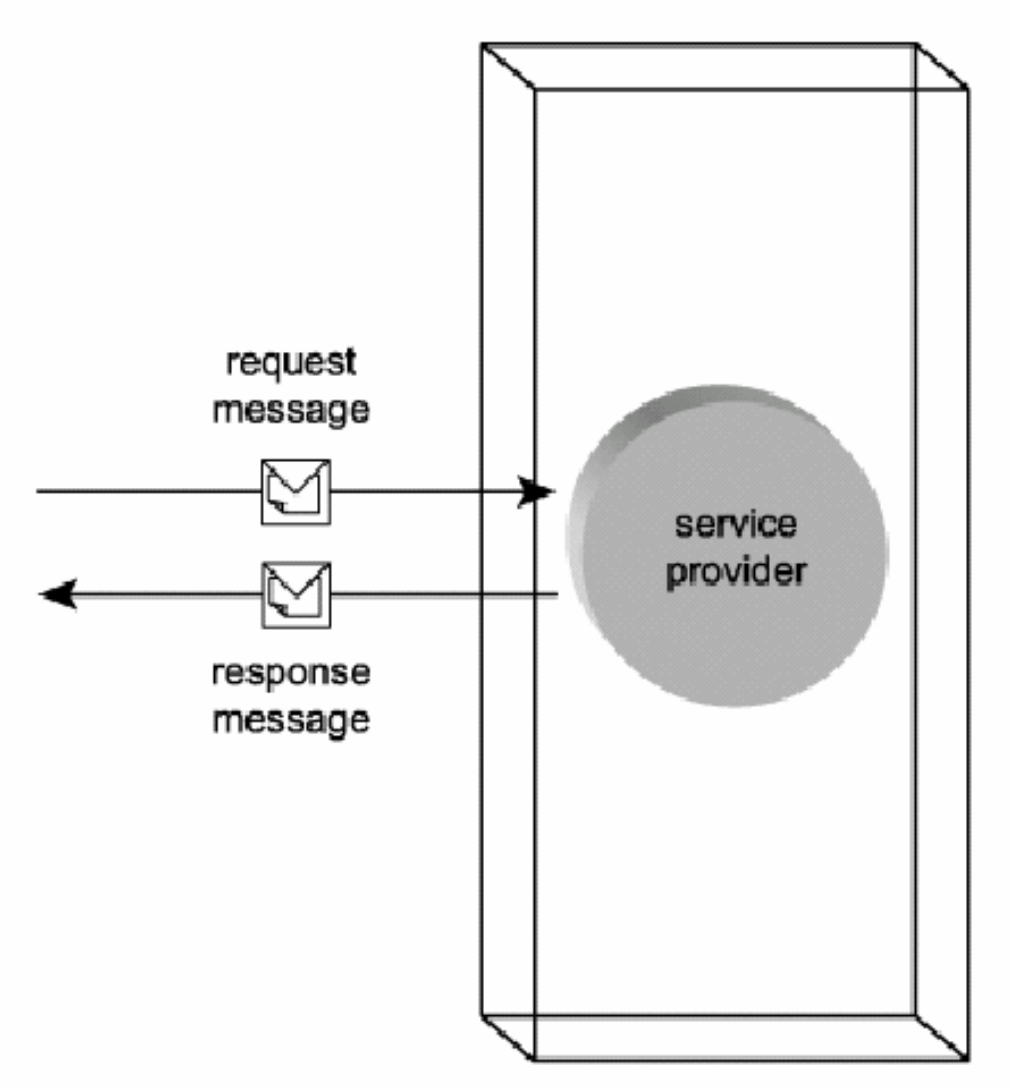
Defining SOA: Separating concrete characteristics

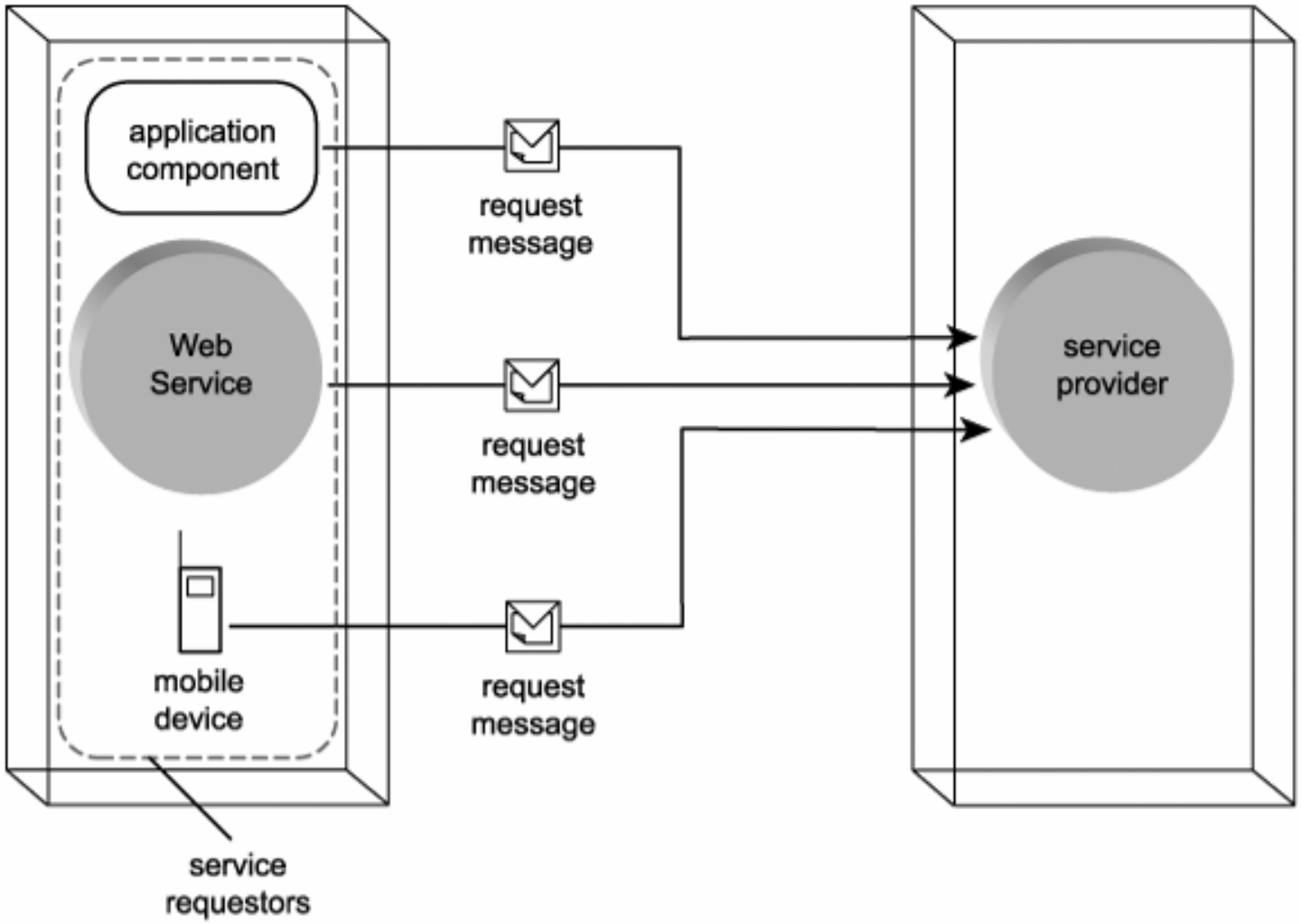
- By trimming these items, along with some superfluous wording, we end up with the following set of concrete characteristics.
- Contemporary SOA is generally:
  - *based on open standards*
  - *architecturally composable*
  - *capable of improving QoS*

Defining SOA: Separating concrete characteristics

- 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
- It is these characteristics that, when realized, provide tangible, measurable benefits.

Services (as Web services)


request
message
service
provider
response
message


application component
Web Service
mobile device
service requestors
request message
request message
request message
service provider

Case Study

- RailCo is one of many long-time vendors used by TLS. Historically, it was the primary air brake parts supplier TLS relied upon. Until recently, TLS had to order parts from RailCo via phone or fax. When a new air brake supplier surfaced, offering competitive prices and signing up with TLS's B2B solution, there was little need for TLS to continue exclusively with RailCo. In fact, TLS only contacted RailCo again when its new primary vendor could not supply a requested part.
- For RailCo to join its competitor as an online partner of TLS, it had to conform to rules and specifications defined by TLS. Specifically, TLS dictates that every supplier must allow TLS to programmatically interface with their inventory control system to submit purchase orders. Additionally, the supplier must be able to connect to TLS's external accounting interface to submit invoices and back-order information.

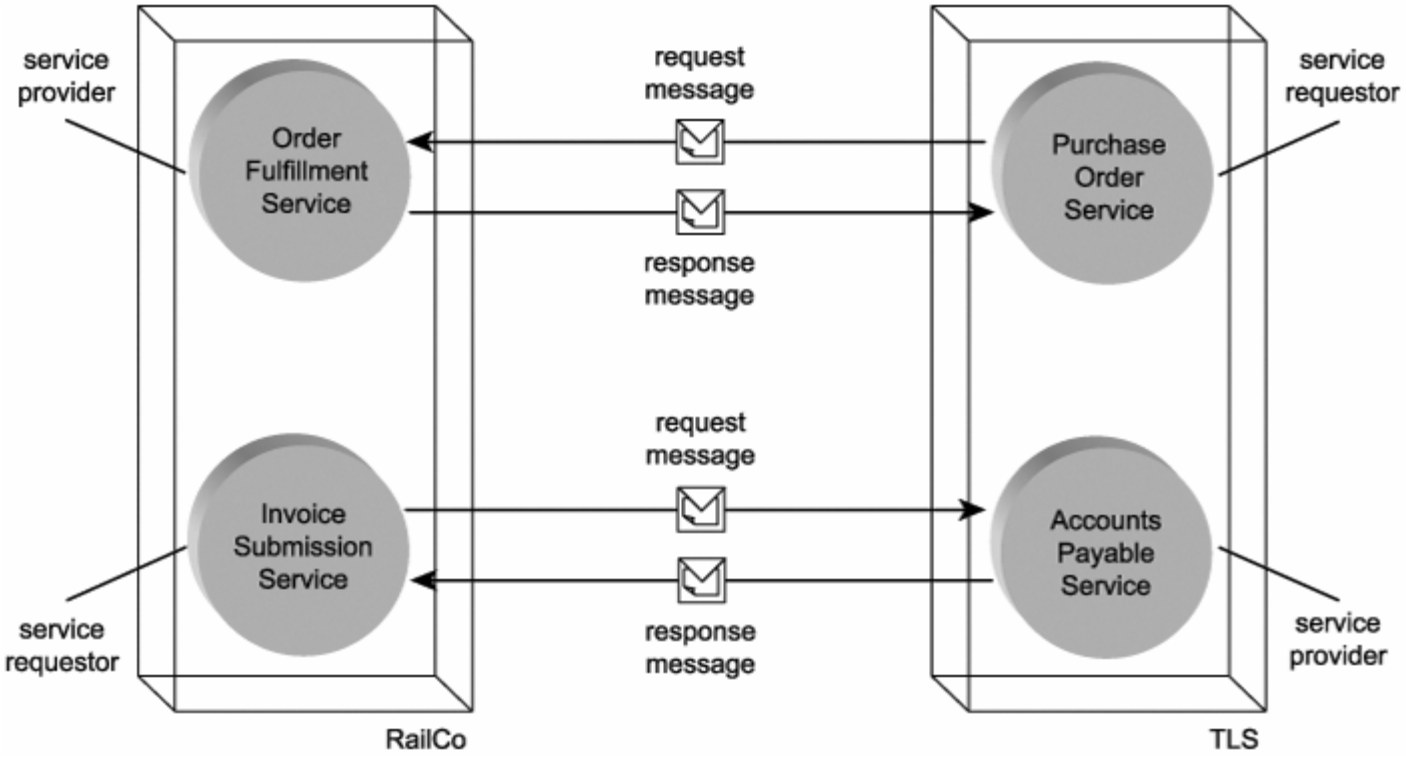
Case Study

- These policies forced RailCo to build an extension to their accounting system, capable of interacting with TLS's Web service-based B2B solution. After RailCo's application went online, the most common data exchange scenarios were as follows:
- TLS's Purchase Order Service submits electronic POs that are received by RailCo's Order Fulfillment Service.
- Upon shipping the order, RailCo's Invoice Submission Service sends an electronic invoice to TLS's Accounts Payable Service.
- Next figure illustrates these two message exchanges.

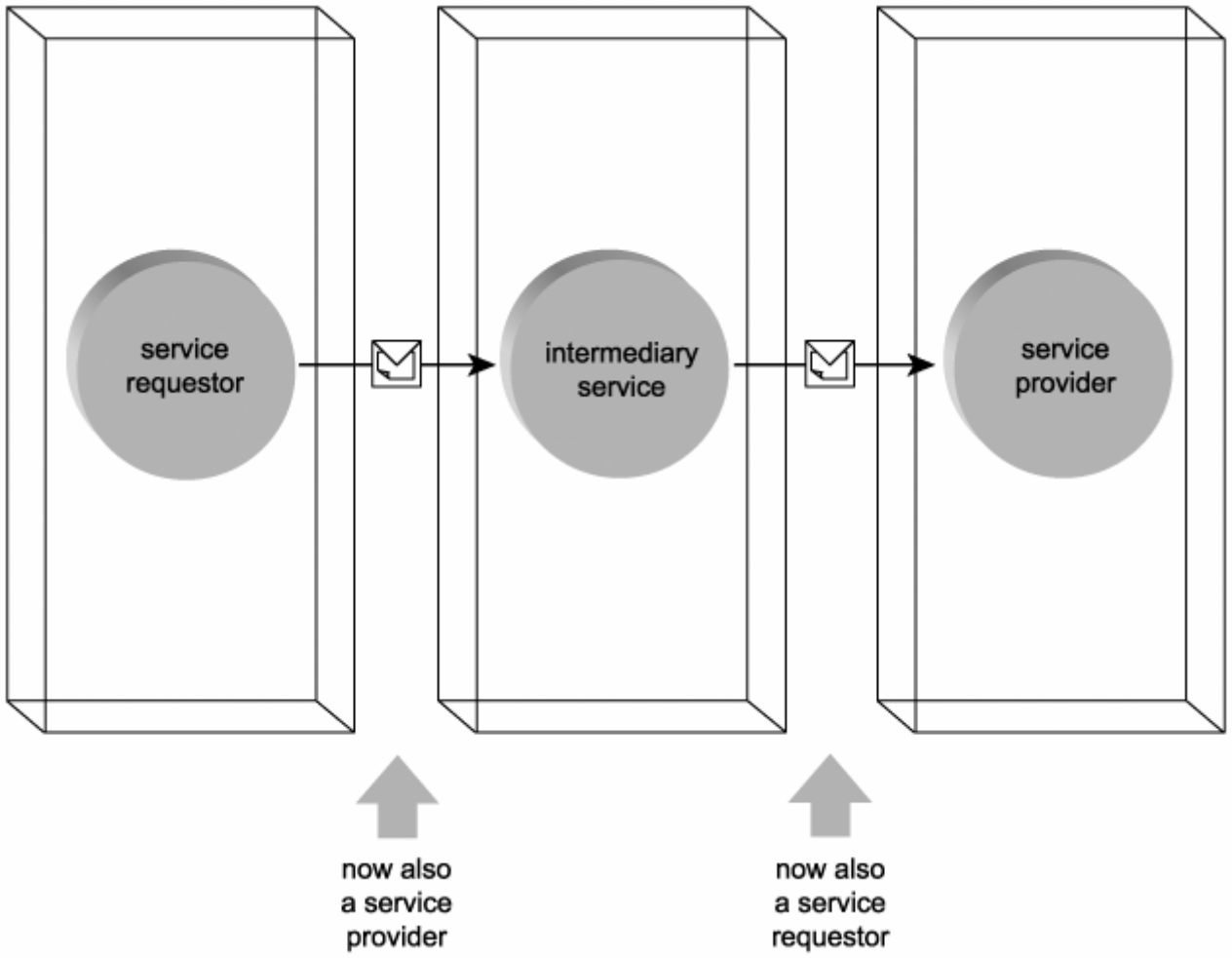
Case Study

- In the first scenario, TLS acts as the service requestor entity. Its Purchase Order Service was the service requestor (or service requestor agent) that initiated the interaction. Being the recipient of the order request, the Order Fulfillment Service is classified as the service provider (or service provider agent). As the owner of this Web service, RailCo is the service provider entity.
- The roles are reversed in the second scenario, where RailCo is the service requestor entity because its Invoice Submission Service acts as the service requestor. TLS's Accounts Payable Service receives the invoice message, making that Web service the service provider, and TLS the service provider entity.

Services swapping roles in different but related message exchanges



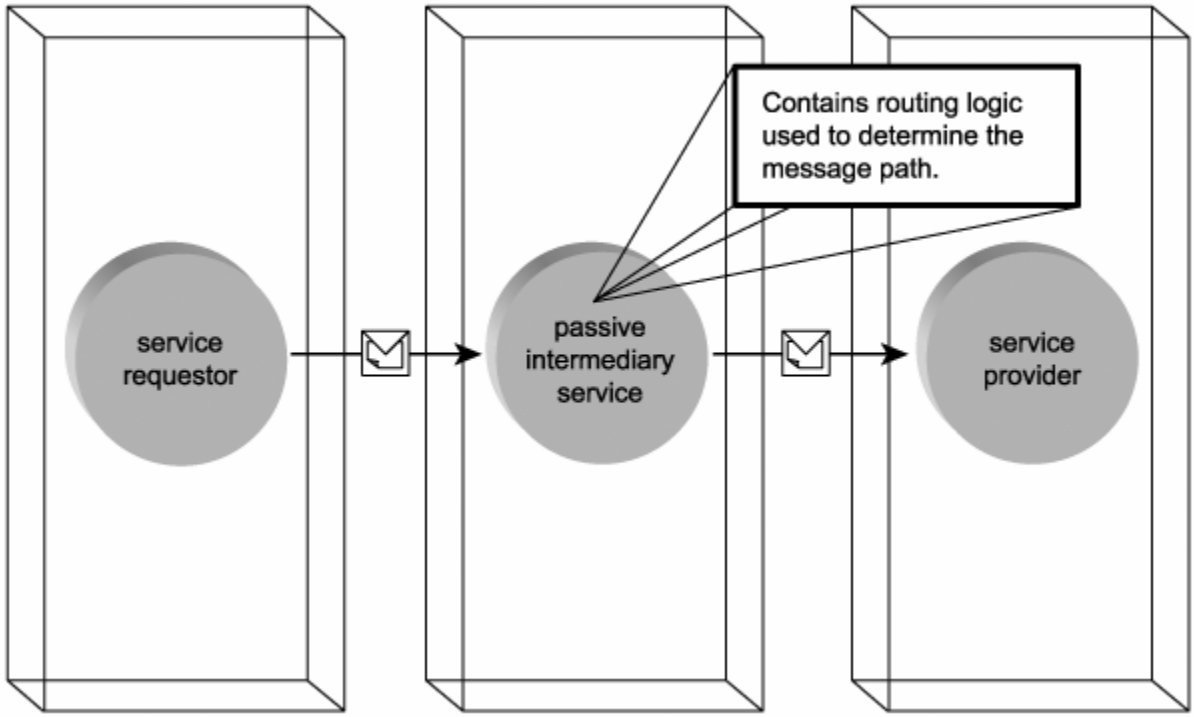
The intermediary service transitions through service provider and service requestor roles while processing a message



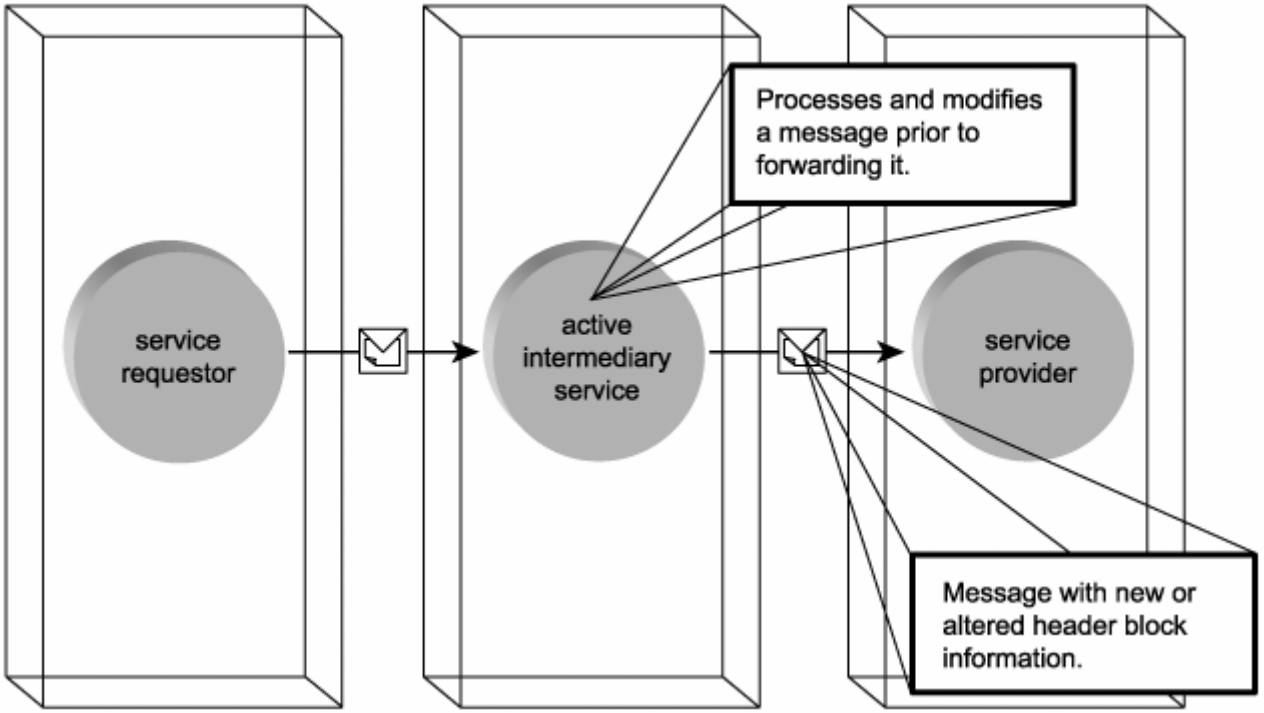
Case Study

- After shipping a TLS order, RailCo's Invoice Submission Service transmits a message containing an electronic invoice. The first TLS Web service to receive the message is a passive intermediary called the Load Balancing Service. Its purpose is to provide load balancing logic by checking the current processing statistics of available TLS servers. When the server with the lowest usage is identified, this passive intermediary routes the message accordingly.
- Upon receiving the message from the Invoice Submission Service requestor, the passive Load Balancing intermediary acts as the service provider. After it has determined where the message is to be forwarded to, it changes its role to service requestor to forward the invoice document to the destination Accounts Payable Service provider.
- Note: The Load Balancing Service (and the upcoming Internal Policy Service) is a form of intermediary that can be explicitly accessed as a Web service through a WSDL or it can act as a service agent. Service agents are intermediaries designed to intercept and process messages en route to their ultimate destinations and are explained later on. TLS opted to develop flexible intermediaries to fulfill requirements specific to their environments.

A passive intermediary service processing a message without altering its contents



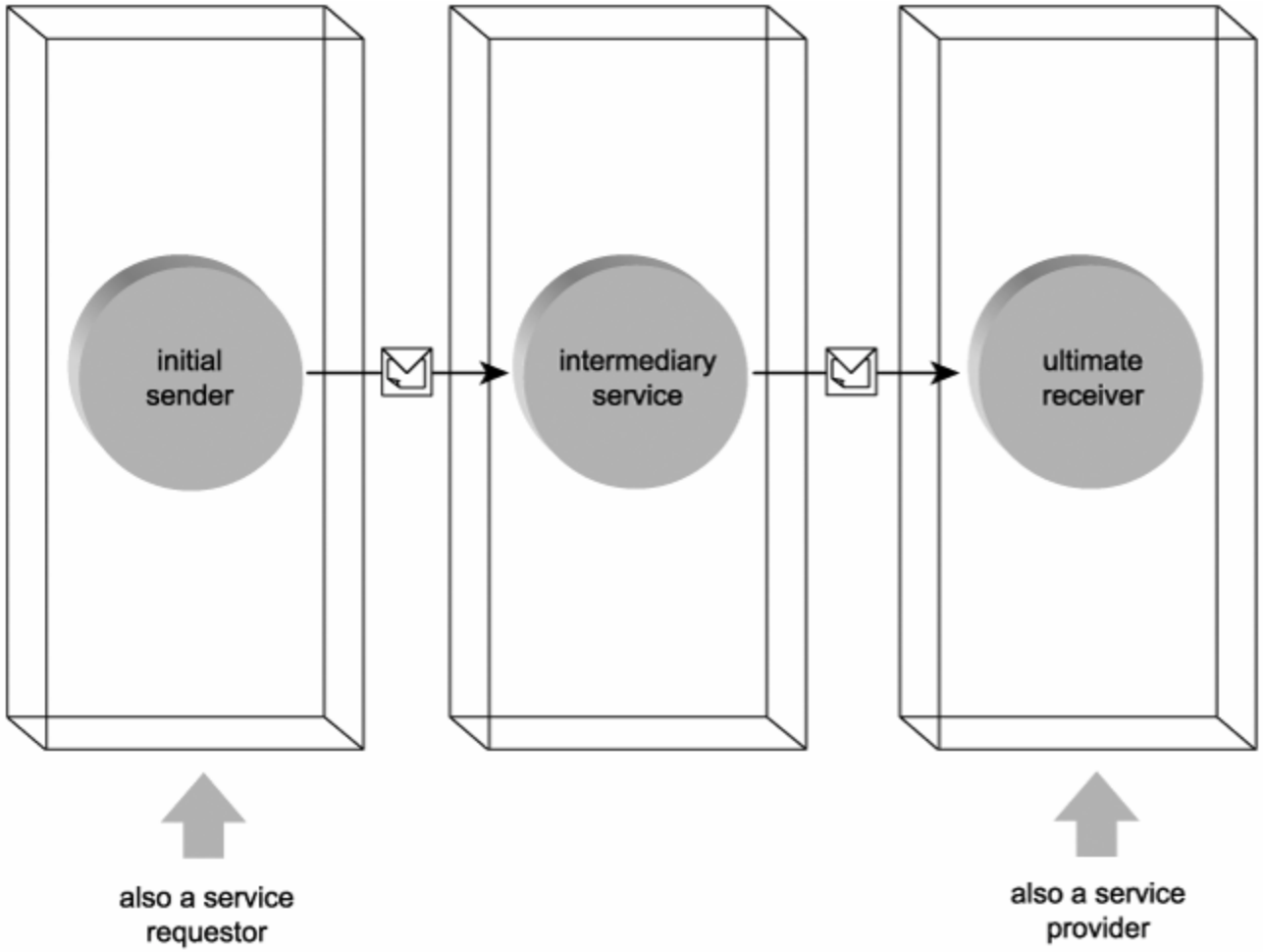
An active intermediary service



Case Study

- TLS employs a number of active intermediaries. The Internal Policy Service, for example, examines the message to determine whether it is subject to any internal policy restrictions.
- If it is, the active intermediary inserts a new header block containing one or more policy rules used by subsequent service providers. As with the passive intermediary example, the active intermediary transitions through service provider and service requestor roles before finally forwarding the message to the appropriate TLS service provider.

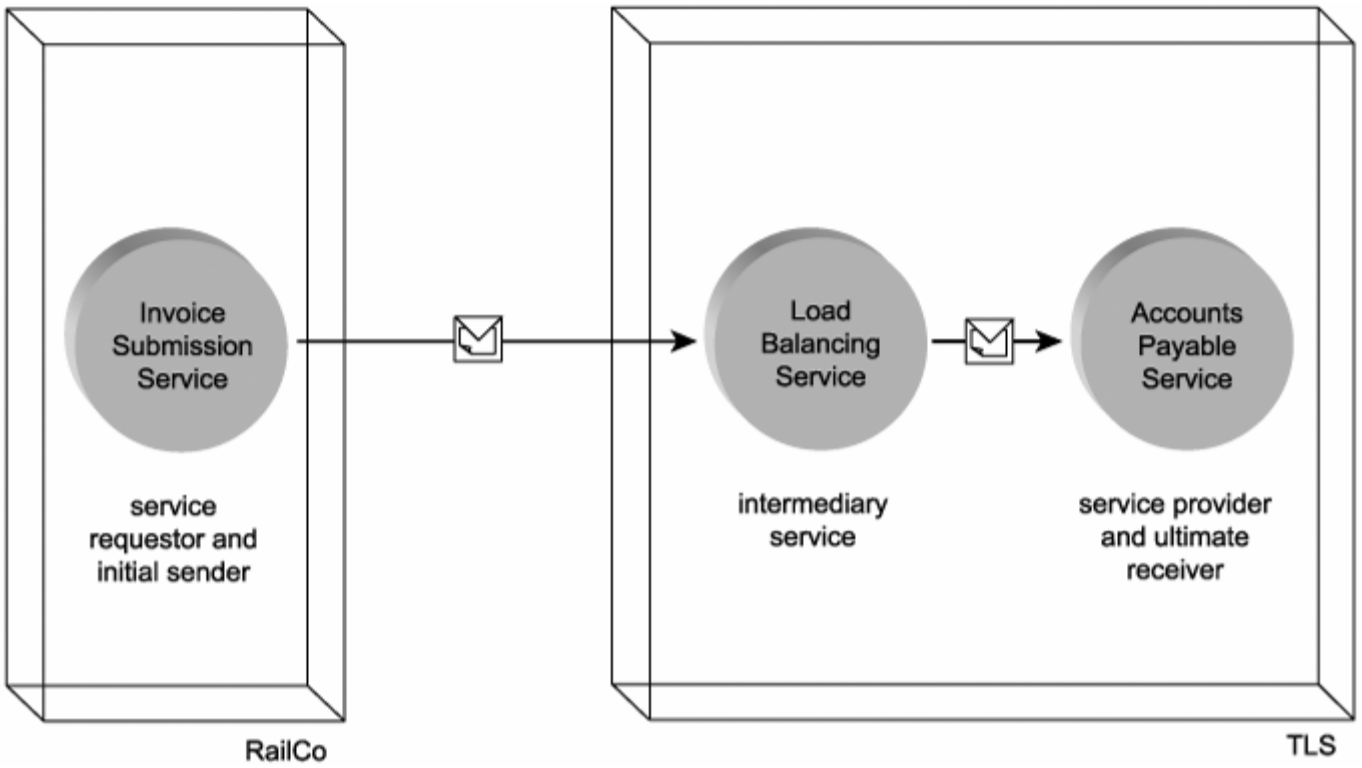
Web services acting as initial sender and ultimate receiver



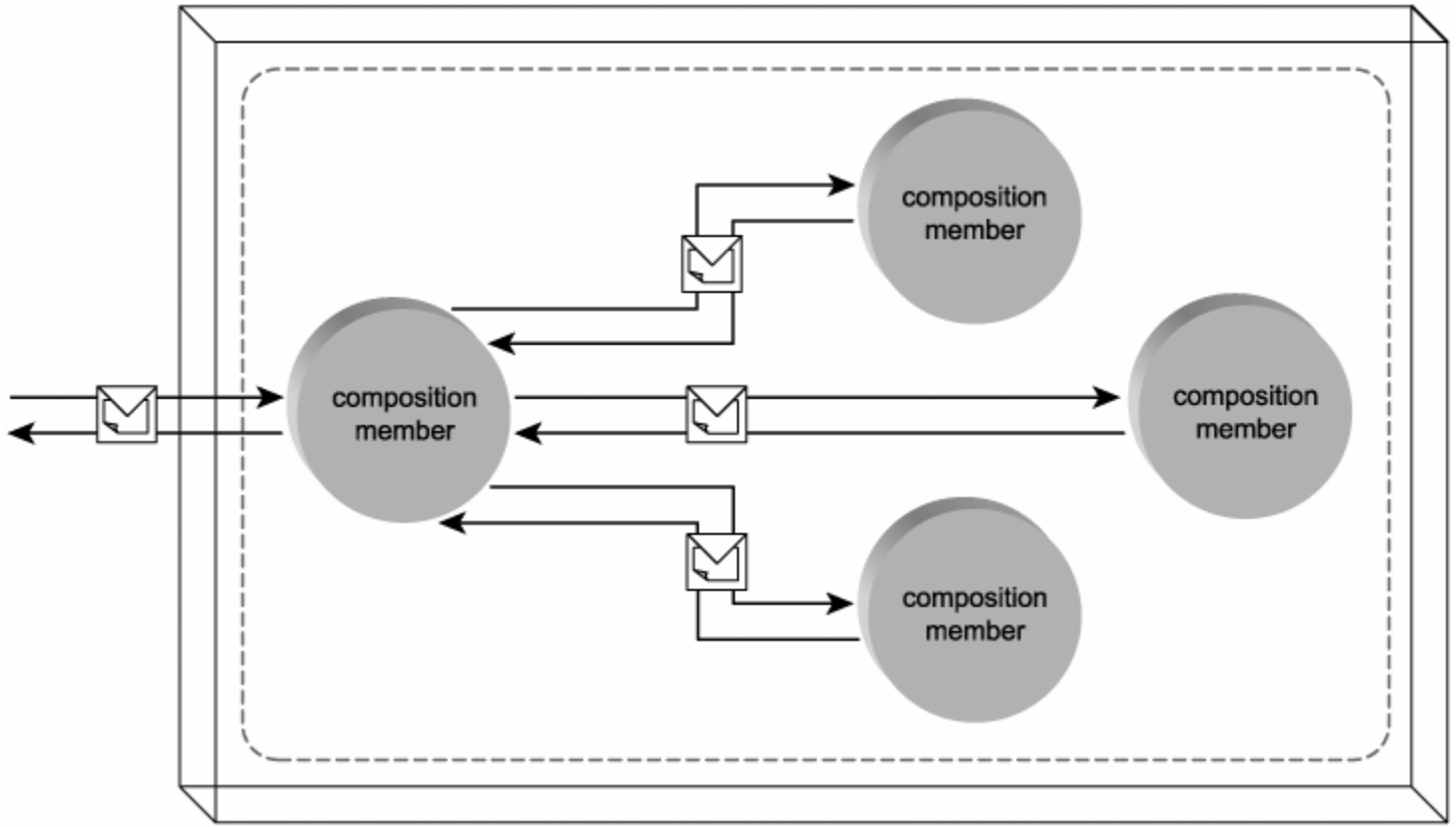
Case Study

- Expanding on the previous example that demonstrated the use of a passive intermediary, let's take a look at all the services involved in that message exchange. In this scenario, we had the RailCo Invoice Submission Service (acting as the service requestor) initiating the message transmission. By receiving the message, the Load Balancing intermediary acts as the service provider. Upon routing the message, the intermediary temporarily assumes the service requestor role and sends the message to the Accounts Payable Service, another service provider.
- These three physical services created four logical roles to complete two service requestor-to-service provider transmissions. There was, however, only one Web service that initiated the transmission. This was the Invoice Submission Service, and it is therefore considered the initial sender. Similarly, there was only one Web service that ended the overall activity, which makes the Accounts Payable Service the ultimate receiver.

Load Balancing Service acting as an intermediary between the initial sender and the ultimate receiver



A service composition consisting of four members



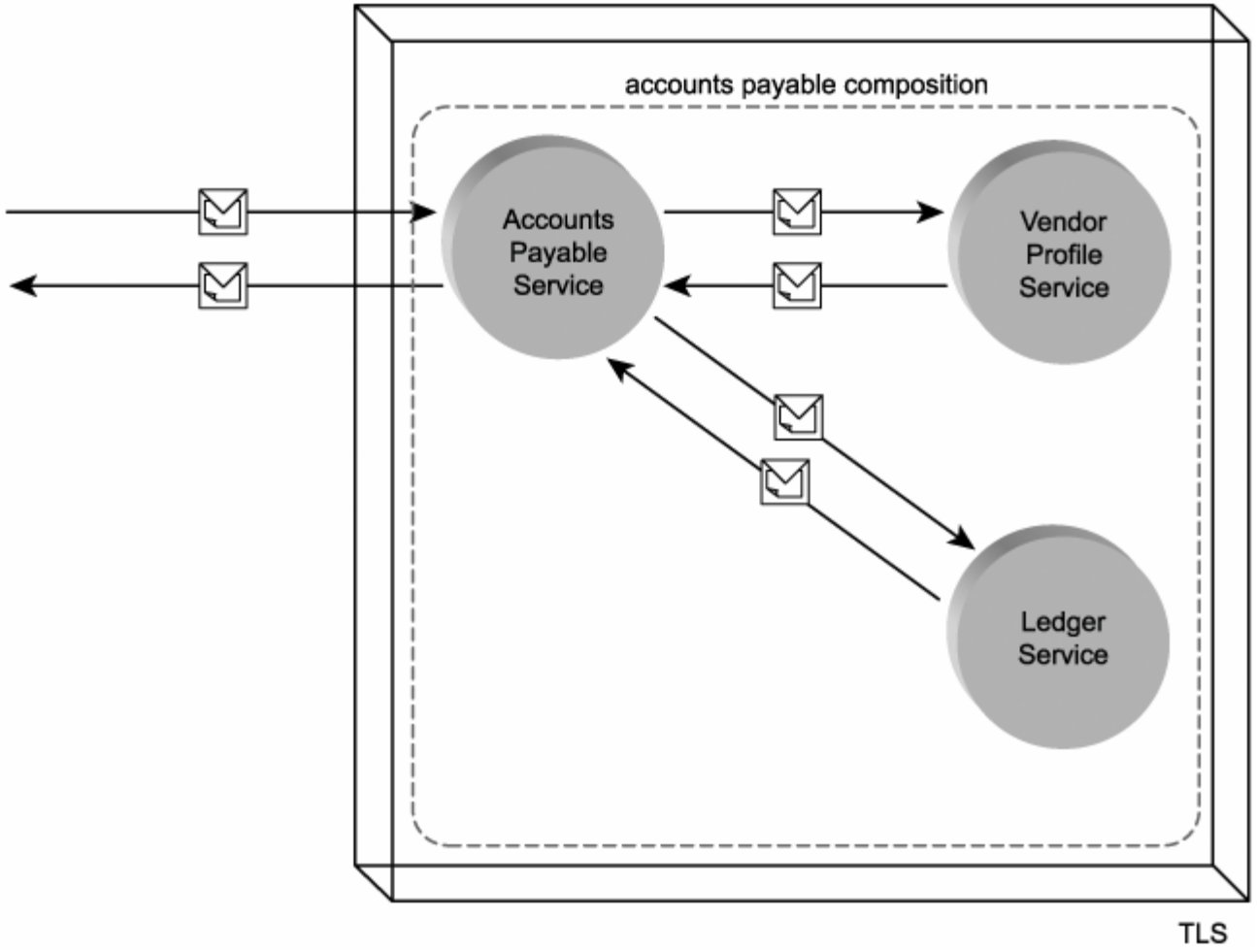
Case Study

- When the TLS Accounts Payable Service receives an invoice, it invokes a series of additional services to fully process the invoice contents:

1. It first uses the Vendor Profile Service to validate the invoice header data and link the invoice document to a vendor account.
2. Next, the Accounts Payable Service extracts taxes and shipping fees and directly logs all amounts into the appropriate A/P accounts.
3. Finally, the Accounts Payable Service passes the Ledger Service the invoice total, which it uses to update the General Ledger.

- In this scenario our service composition consists of three composition members, spearheaded by the Accounts Payable Service.

The Accounts Payable Service enlisting other services in a service composition



Service models

- The roles we've 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.

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

Utility service model

- Any generic Web service or service agent designed for potential reuse can be classified as a utility service. The key to achieving this classification is that 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

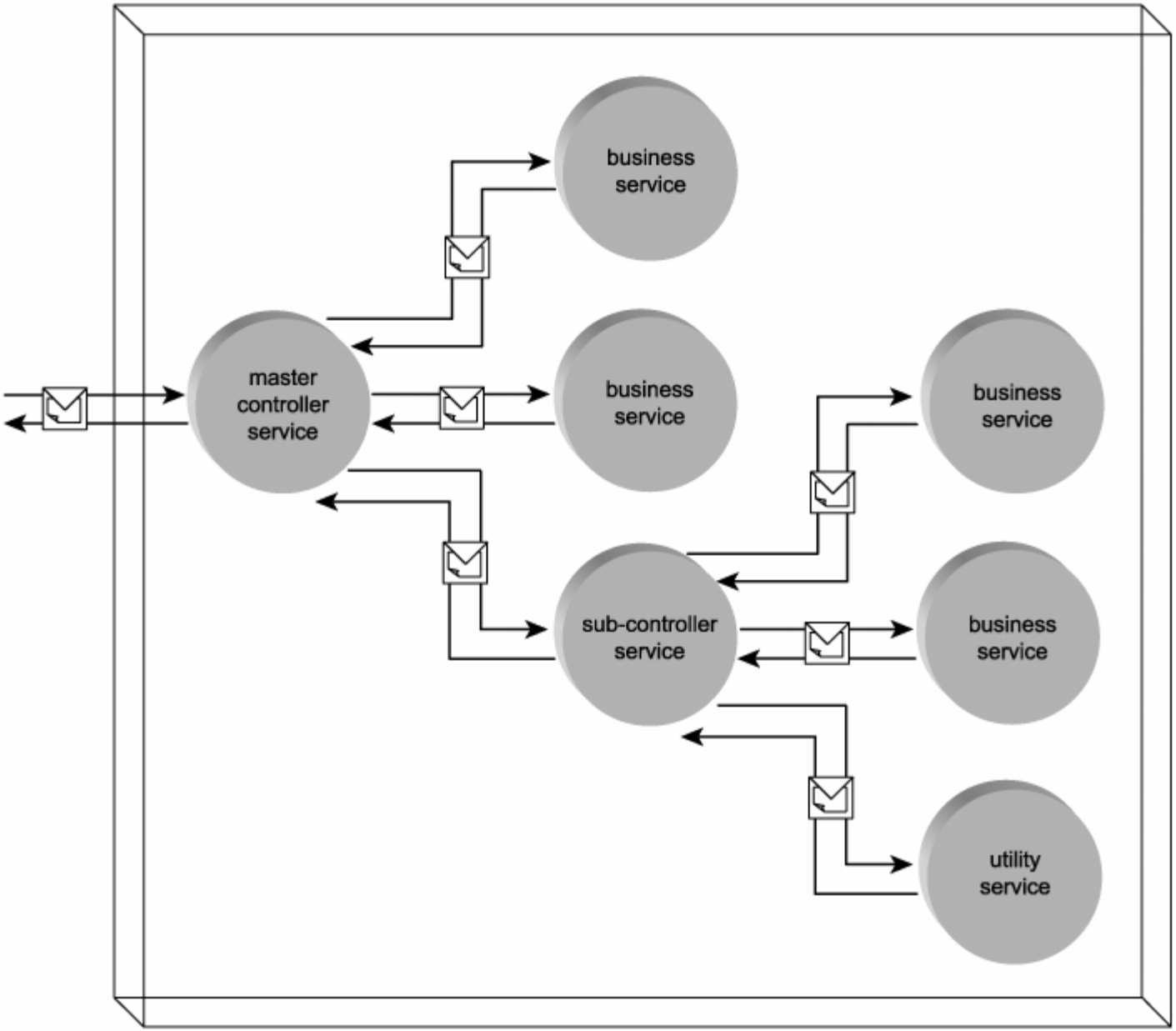
Case Study

- In the examples we've gone through so far, we've described eight Web services. Six of these are business services, while the other two are utility services, as follows:
  - Accounts Payable Service = business service
  - Internal Policy Service = utility service
  - Invoice Submission Service = business service
  - Ledger Service = business service
  - Load Balancing Service = utility service
  - Order Fulfillment Service = business service
  - Purchase Order Service = business service
  - Vendor Profile Service = business service
- The Load Balancing and Internal Policy Services are classified as utility services because they provide generic functionality that can be reused by different types of applications.
- The application logic of the remaining services is specific to a given business task or solution, which makes them business-centric services.

Controller service model

- Service compositions are comprised of a set of independent services that each contribute to the execution of the overall business task. The assembly and coordination of these services is often a task in itself and one that can be assigned as the primary function of a dedicated service or as the secondary function of a service that is fully capable of executing a business task independently. The controller service fulfills this role, acting as the parent service to service composition members.
  - Controller services are used within SOAs as follows:
  - to support and implement the principle of composability
  - to leverage reuse opportunities
  - to support autonomy in other services
- Note that controller services themselves can become subordinate service composition members. In this case the composition coordinated by a controller is, in its entirety, composed into a larger composition. In this situation there may be a master controller service that acts as the parent to the entire service composition, as well as a sub-controller, responsible for coordinating a portion of the composition.

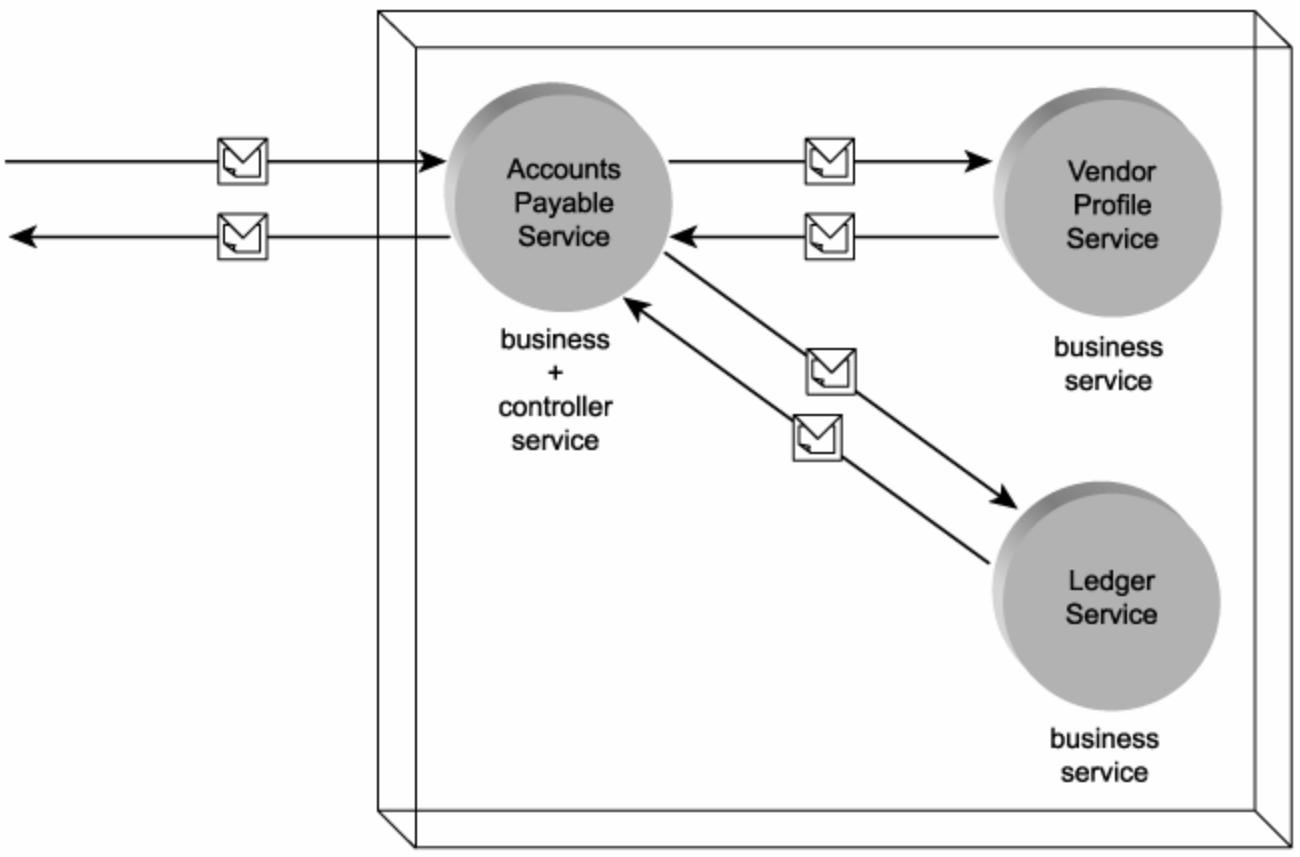
A service composition consisting of a master controller, a sub-controller, four business services, and one utility service



Case Study

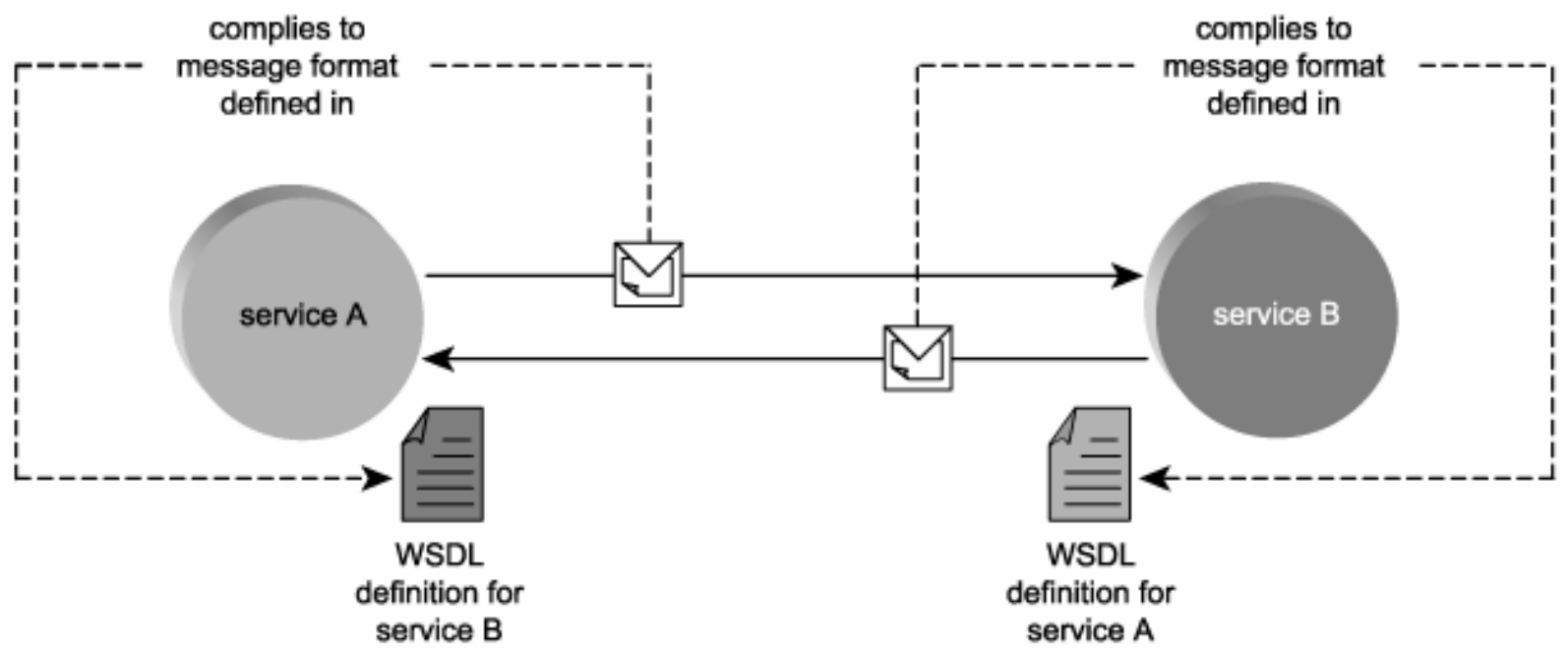
- In our previous example we demonstrated how the Accounts Payable Service initiated and coordinated a service composition consisting of two additional composition members. That would classify the Accounts Payable Service as a controller service.
- The fact that we already labeled this service as a business service does not conflict with this new classification; a single service can be classified as more than one service model

The Accounts Payable Service acting as a business and controller service, composing two other business services



Service descriptions (with WSDL)

WSDL definitions enable loose coupling between services



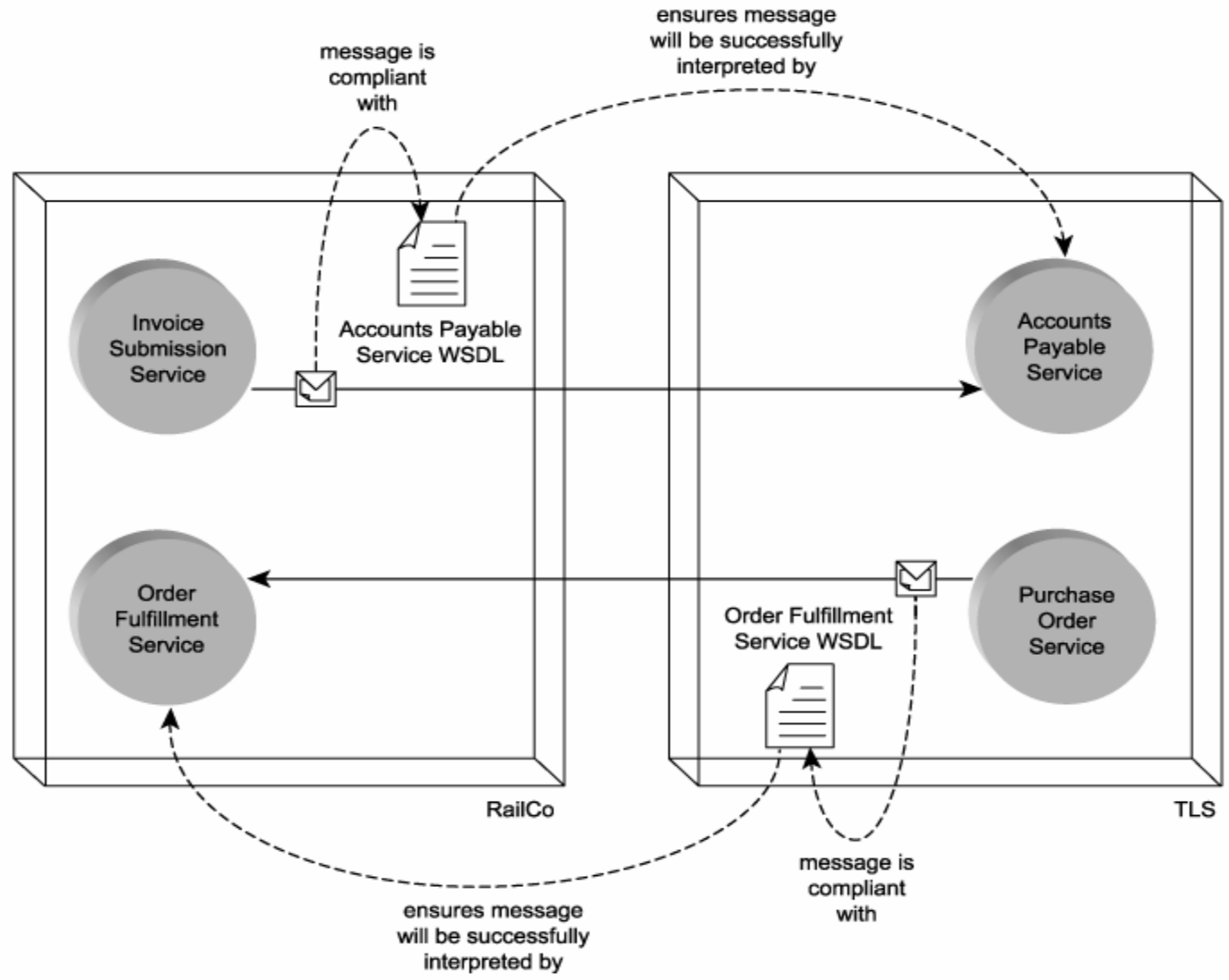
Case Study

- For RailCo to design its B2B Web services in full compliance with the TLS services, RailCo acquires the WSDL service description published by TLS for their Accounts Payable Service. This definition file then is used by developers to build the Invoice Submission Service so that it can process SOAP messages in accordance with the service interface requirements defined in the TLS service descriptions.
- Further, RailCo provides TLS with a copy of the WSDL definition for the RailCo Order Fulfillment Service. TLS registers this service description and adds it to the list of vendor endpoints that will receive electronic purchase orders.

Case Study

- Note that because it is TLS that defines the terms of message exchange with other parties, RailCo developed both of its services to meet TLS's requirements.
- The Invoice Submission Service was built as a service requestor that issues messages compliant with the Accounts Payable WSDL. The Order Fulfillment Service was designed as a service provider according to published specifications by TLS. This guarantees TLS that its Purchase Order Service (acting as a service requestor) can continue to issue messages in its current format and that all recipient endpoints will be able to receive and understand them.

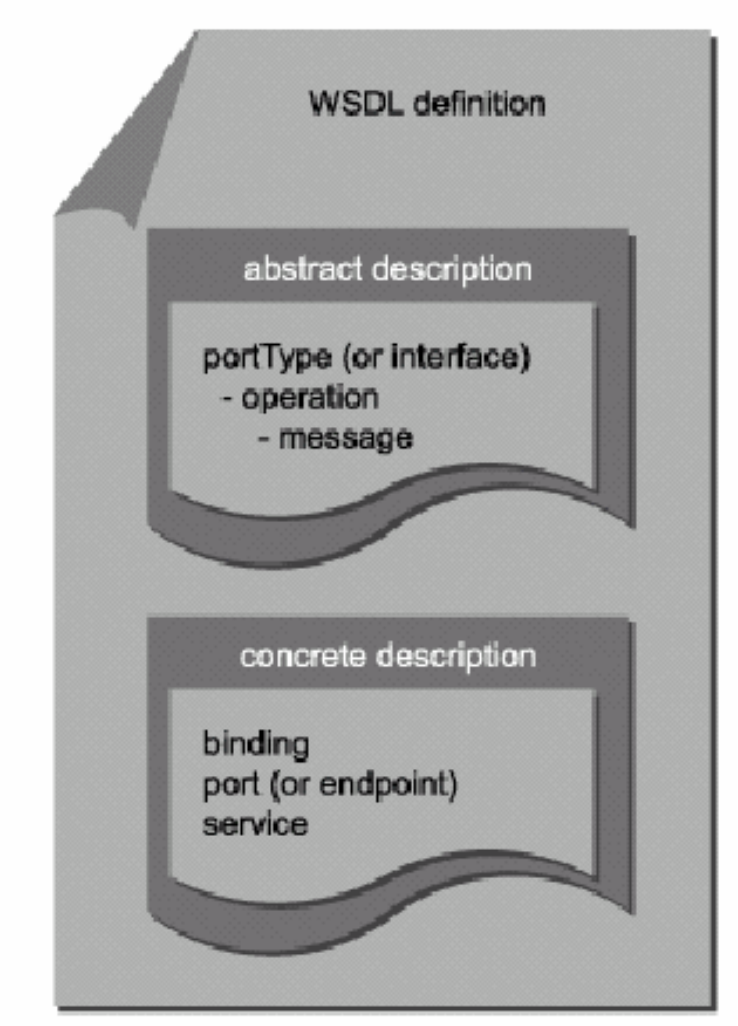
Each service requestor is using the WSDL of a service provider to ensure that messages sent will be understood and accepted



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. I
- t 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

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



portType, operation, and message

- 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. Much 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.
- Note that the transmission sequence of these messages can be governed by a predetermined message exchange pattern that also is associated with the operation.
- Note:The term "portType" is being renamed to "interface" in version 2.0 of the WSDL specification.

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

- A WSDL description's binding describes the requirements for a service to establish physical connections or for connections to be established with the service. In other words, a 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. 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.

Concrete description

- Note:
- The term "port" is being renamed "endpoint" in version 2.0 of the WSDL specification. The WSDL endpoint should not be confused with the general term "endpoint" used to reference the point of contact for a Web service. Though related, the term "endpoint" is used in a much broader sense than the WSDL endpoint, which refers to a language element that only represents the physical address of the service.

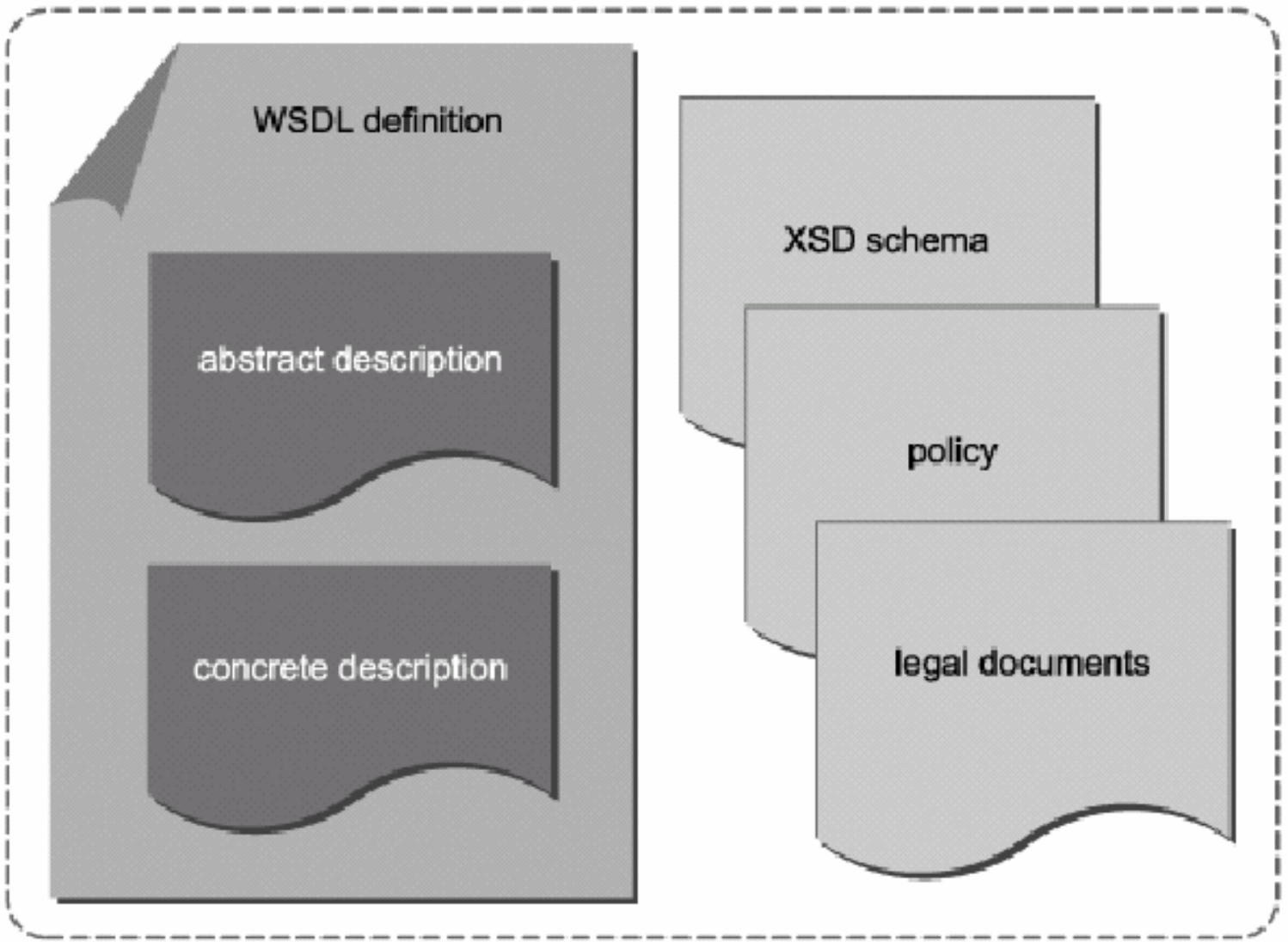
Case Study

- The TLS Accounts Payable Service was created to receive invoices submitted by numerous vendors. Its associated service description therefore has a simple abstract description consisting of one interface definition that contains a single operation called SubmitInvoice.
- Specified within the operation is one input and one output message. The input message is responsible for accepting the invoice document from a vendor service requestor (such as the RailCo Invoice Submission Service). The output message is used by the Accounts Payable Service to send a message of acknowledgement indicating that the submitted invoice document has been successfully received and that its contents are valid. The concrete part of this service description simply binds the operation to the SOAP protocol and provides a location address for the Accounts Payable Service.

Metadata and service contracts

- 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.
- So now we have up to three separate documents that each describe an aspect of a service:
  - WSDL definition
  - XSD schema
  - 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 contract a set of conditions that must be met and accepted by a potential service requestor to enable successful communication.
- Note that a service contract can refer to additional documents or agreements not expressed by service descriptions. For example, a Service Level Agreement (SLA) agreed upon by the respective owners of a service provider and its requestor can be considered part of an overall service contract.

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



Semantic descriptions

- Most of the metadata currently provided by services focuses on expressing technical information related to data representation and processing requirements. However, these service description documents generally do not prove useful in explaining details about a service's behavioral characteristics. In fact, the most challenging part of providing a complete description of a Web service is in communicating its semantic qualities.
- Examples of service semantics include:
  - how a service behaves under certain conditions
  - how a service will respond to a specific condition
  - what specific tasks the service is most suited for

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 usually of greater importance 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.
- Although service policies can be designed to express preferences and assertions that communicate aspects of service behavior, efforts are currently underway (primarily by the W3C) to continually extend the semantic information provided by service description documents. For the time being, we must focus on the service description capabilities offered to us through WSDL definitions, XSD schemas, and policies.

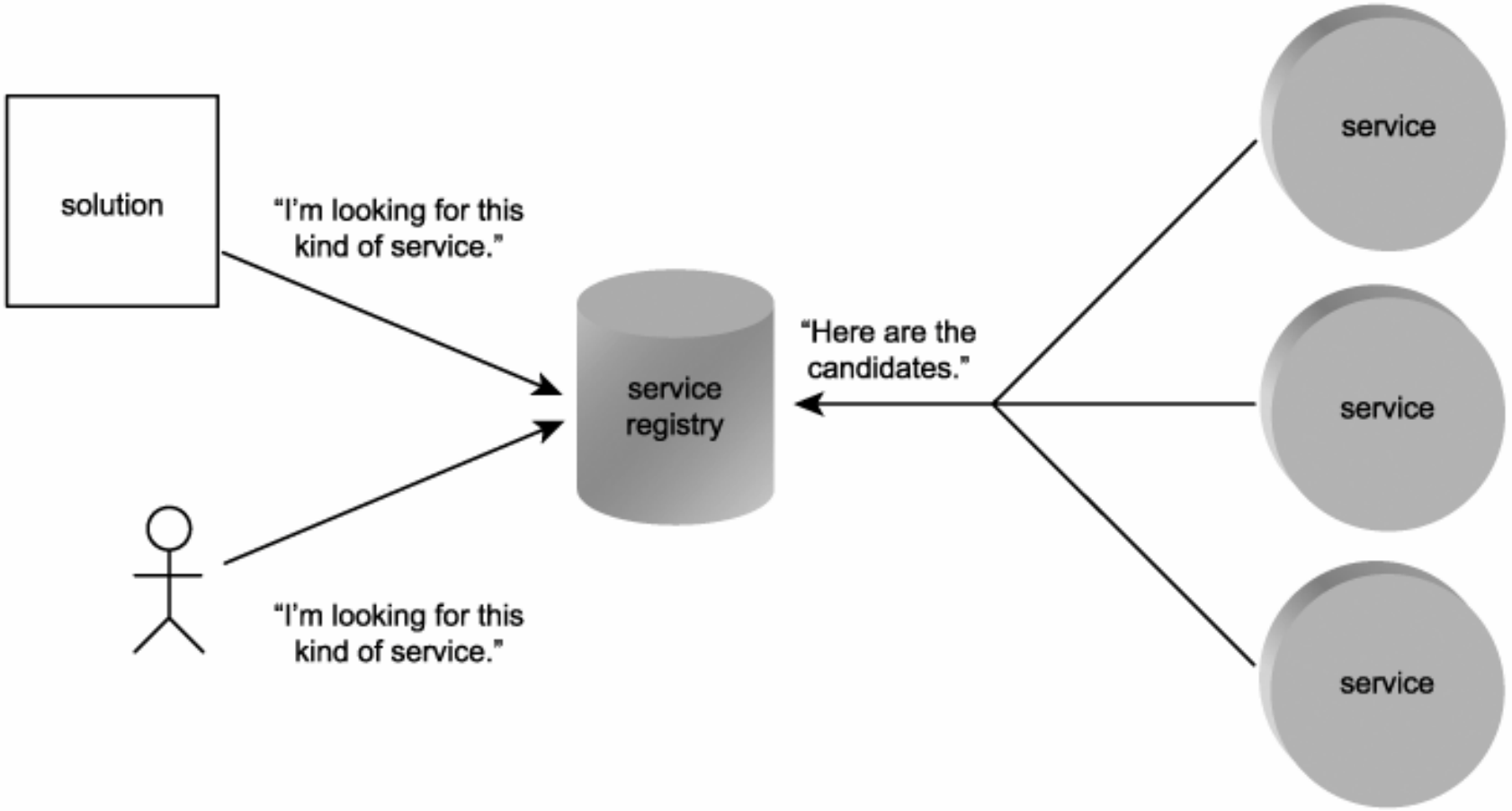
Semantic descriptions

- As we've established, the sole 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
- When the initial set of Web services standards emerged, this eventuality was taken into account. This is why UDDI formed part of the first generation of Web services standards. Though not yet commonly implemented, UDDI provides us with a registry model worth describing.

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.

Service description locations centralized in a registry



Private and public registries

- 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.
- Following are descriptions of the primary parts that comprise UDDI registry records.

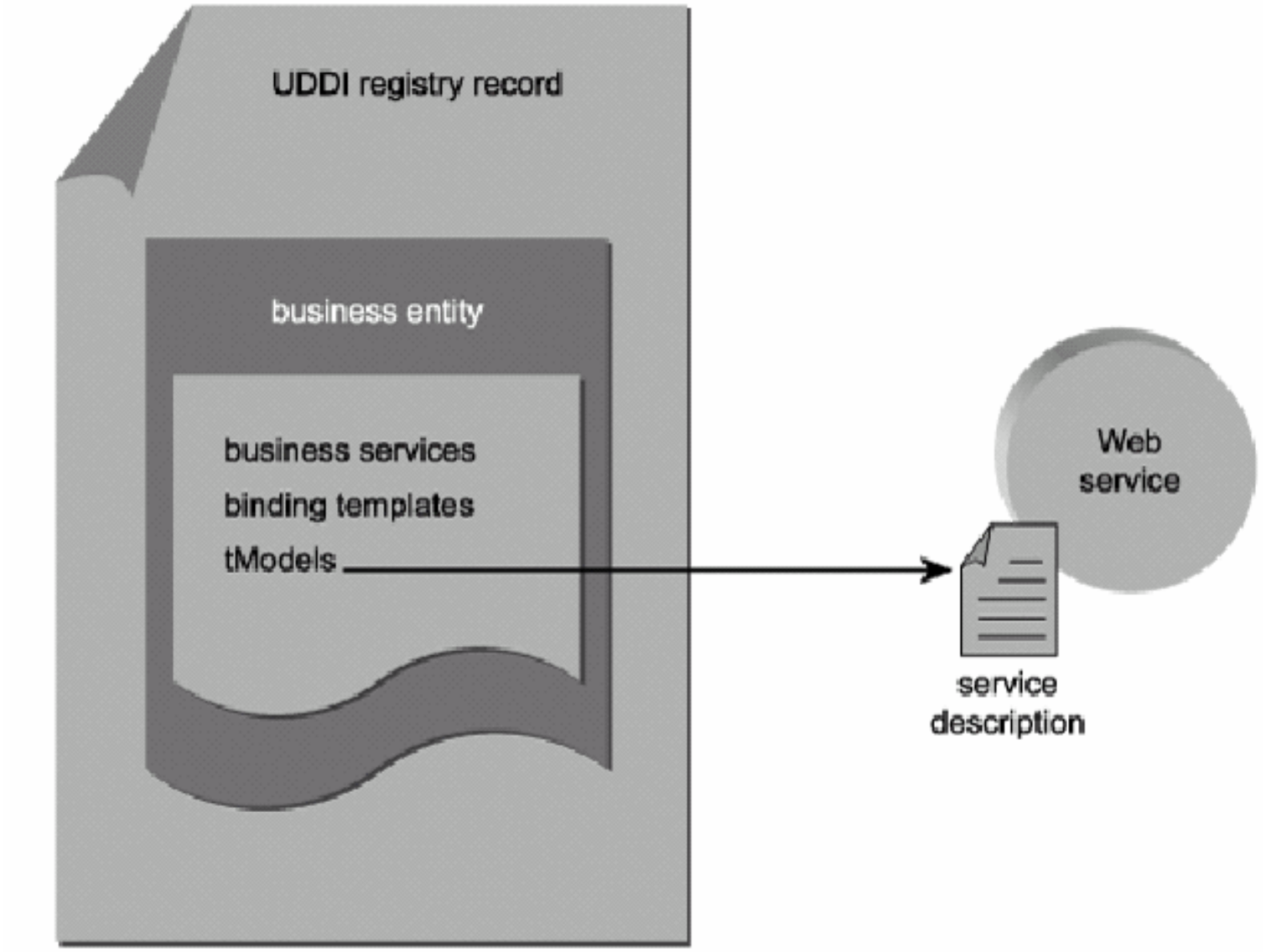
Business entities and business services

- 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.

Binding templates and tModels

- You might recall that 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

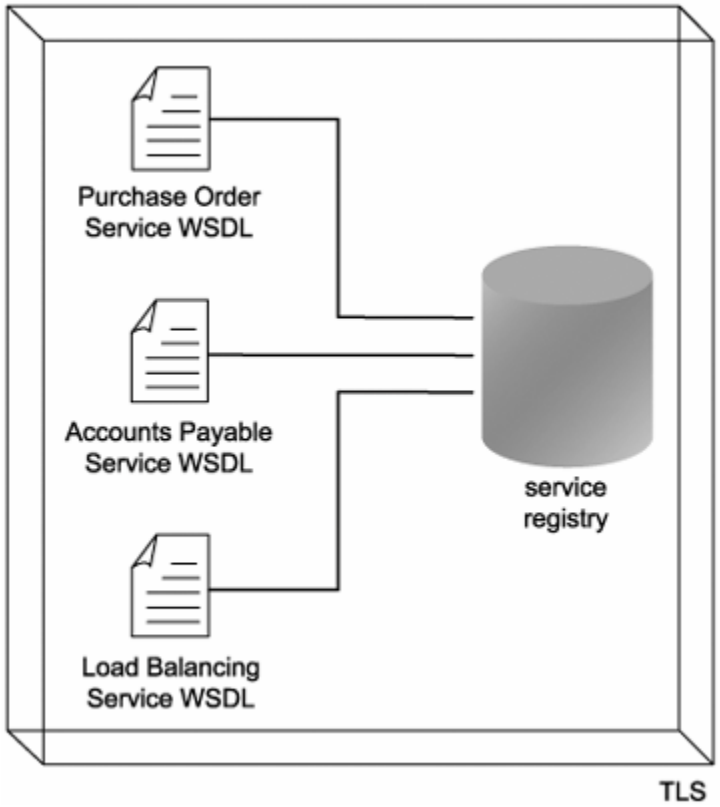
The basic structure of a UDDI business entity record



Case Study

- At any given time there are several concurrent development and integration projects underway at TLS. Almost every project results in the creation of new services. Some are developed as part of service-oriented solutions, while others originate from legacy adapters and ancillary services appended to older distributed systems. The net result is a constantly growing pool of unmanaged services.
- After a year-end review of past development initiatives, it was discovered that several project teams had inadvertently built Web services with very similar functionality. To avoid a recurrence of redundant effort, a private registry was created. Project teams responsible for any currently active service descriptions were required to register their services in the registry (and this registration process became part of the standard development lifecycle from there on).

The TLS service registry containing pointers to current TLS WSDL definitions.



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. What this means is that through 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.

Header blocks

- 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.

Header blocks

- Examples of the types of features a message can be outfitted with using header blocks include:
- processing instructions that may be executed by service intermediaries or the ultimate receiver
- routing or workflow information associated with the message
- security measures implemented in the message
- reliability rules related to the delivery of the message
- context and transaction management information
- correlation information (typically an identifier used to associate a request message with a response message)

Header blocks

- These and many other features are available, and the selection is continually growing. Because header blocks can be based on the use of different supplementary extensions, 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.
- Note: Processing instructions provided in SOAP header blocks are different from the processing instructions natively supported by the XML language.

Case Study

- Invoices sent via SOAP messages to TLS are required to contain a number of standard header blocks for them to be accepted and processed by the TLS Accounts Payable Service.
- Specifically, the required header blocks include:
- A correlation identifier that conforms to a standard format and is further extended with a value derived from the date and time of the message transmission. The correlation identifier therefore relates the original submission to the eventual response.
- Organization-level security credentials used for authentication purposes. Each vendor has a security account with the TLS B2B system, and the assigned credentials are required with every message transmission.
- The Accounts Payable Service expects these pieces of meta information, and the gathered rules and instructions shape its subsequent processing of the message contents.

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.

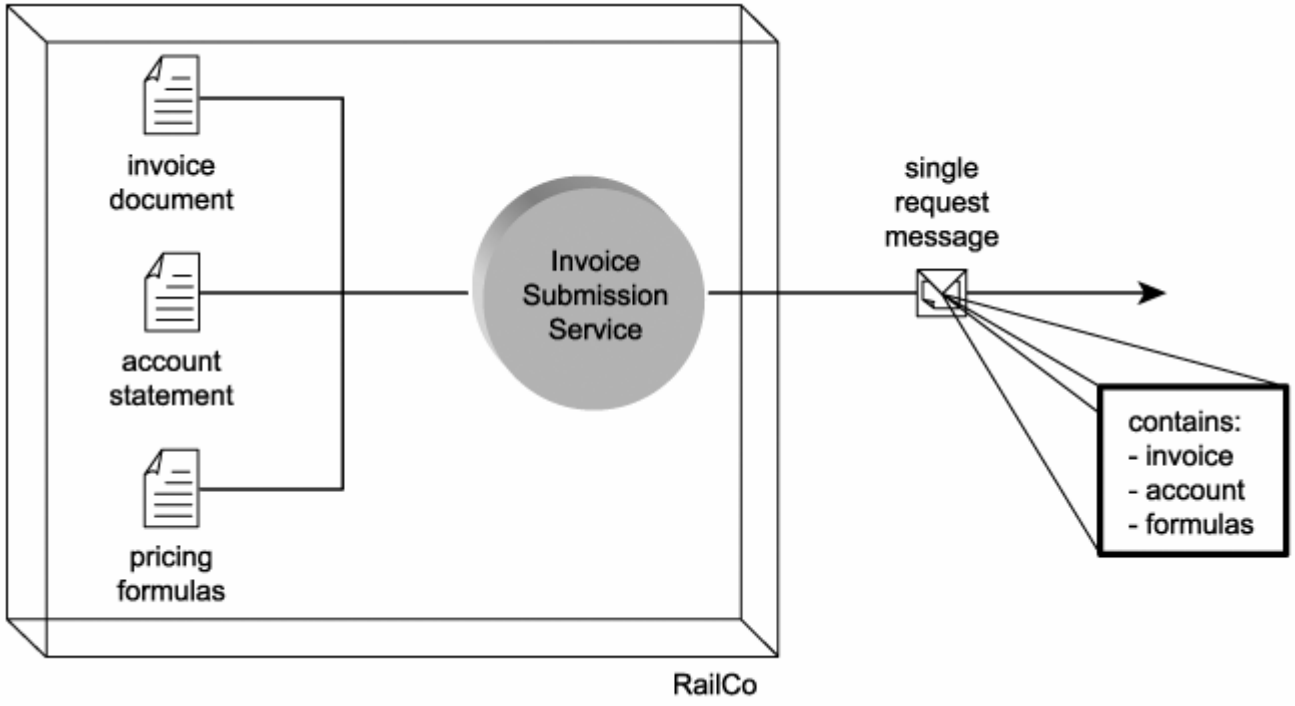
Message styles

- Note:
- Don't confuse document-style SOAP messages with document-centric XML documents.
- The latter term generally refers to published documents represented by XML and is used to distinguish these types of XML documents from those that contain application data (which are typically referred to as data-centric XML documents).

Case Study

- Traditionally, the submission of an invoice involved a number of interactions between RailCo and its customer, including:
- The generation and mailing of the invoice document.
- The generation and mailing of an account statement, showing all currently outstanding amounts owed by the customer.
- The generation and mailing of a quantity discount reminder, explaining RailCo's volume pricing policy, and showing how close the customer is to reaching a quantity discount based on parts ordered to date.
- When forced to submit invoices electronically to TLS via the Invoice Submission Service, all three of these documents needed to be included in the same message. As a result, a single document-style message used by RailCo is capable of providing an invoice, an account statement, and volume discount pricing formulas

Invoice Submission Service packaging the contents of three documents into one SOAP message



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.

Case Study

- TLS accounting policy requires that all issued purchase orders in excess of $100,000 require the signature of a senior manager. Further, these 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. To accommodate this requirement, the Purchase Order Service was designed with an alternative operation.
- The accounting system currently used by TLS offers the ability to scan any accounting-related documents. The scanned images are archived on a separate server and linked to the corresponding accounting records via the archive image path. When PO cost totals exceed the $100,000 limit, a custom-developed extension to the accounting system invokes the alternative Purchase Order Service operation and passes it a copy of the signed PO document image. The service, in turn, generates a SOAP message in which the PO document image exists as a SOAP attachment.

Faults

- Finally, 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.

Case Study

- The before mentioned SOAP message containing a SOAP attachment is also outfitted with a fault area housing exception information relating specifically to the attached data. Should the recipient of the SOAP message be unable to properly process the attachment or should the attachment encounter delivery problems, standard fault codes and descriptions are used to generate a response message that is returned to TLS.

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.

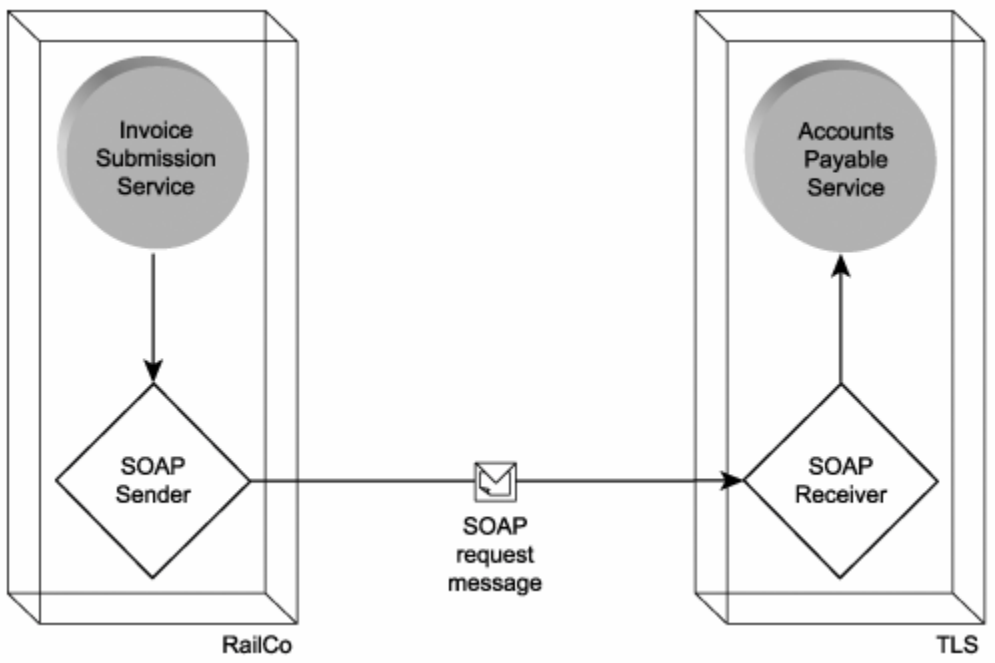
Node types

- As with the services that use them, 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). You'll notice that these names are very similar to the Web service roles we discussed at the beginning. The SOAP specification has a different use for the term "role" and instead refers to these SOAP types or labels as concepts.
  - SOAP sender SOAP node that transmits a message
  - SOAP receiver SOAP node that receives a message
  - SOAP intermediary 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

Case Study

- When the RailCo Invoice Submission Service sends a SOAP message containing an invoice, the underlying SOAP server software (representing the initial SOAP sender node) executes the transmission of the SOAP message via HTTP.
- Prior to the TLS Accounts Payable Service actually receiving the invoice message, the TLS SOAP server or listener (representing the ultimate SOAP receiver node) receives the message first

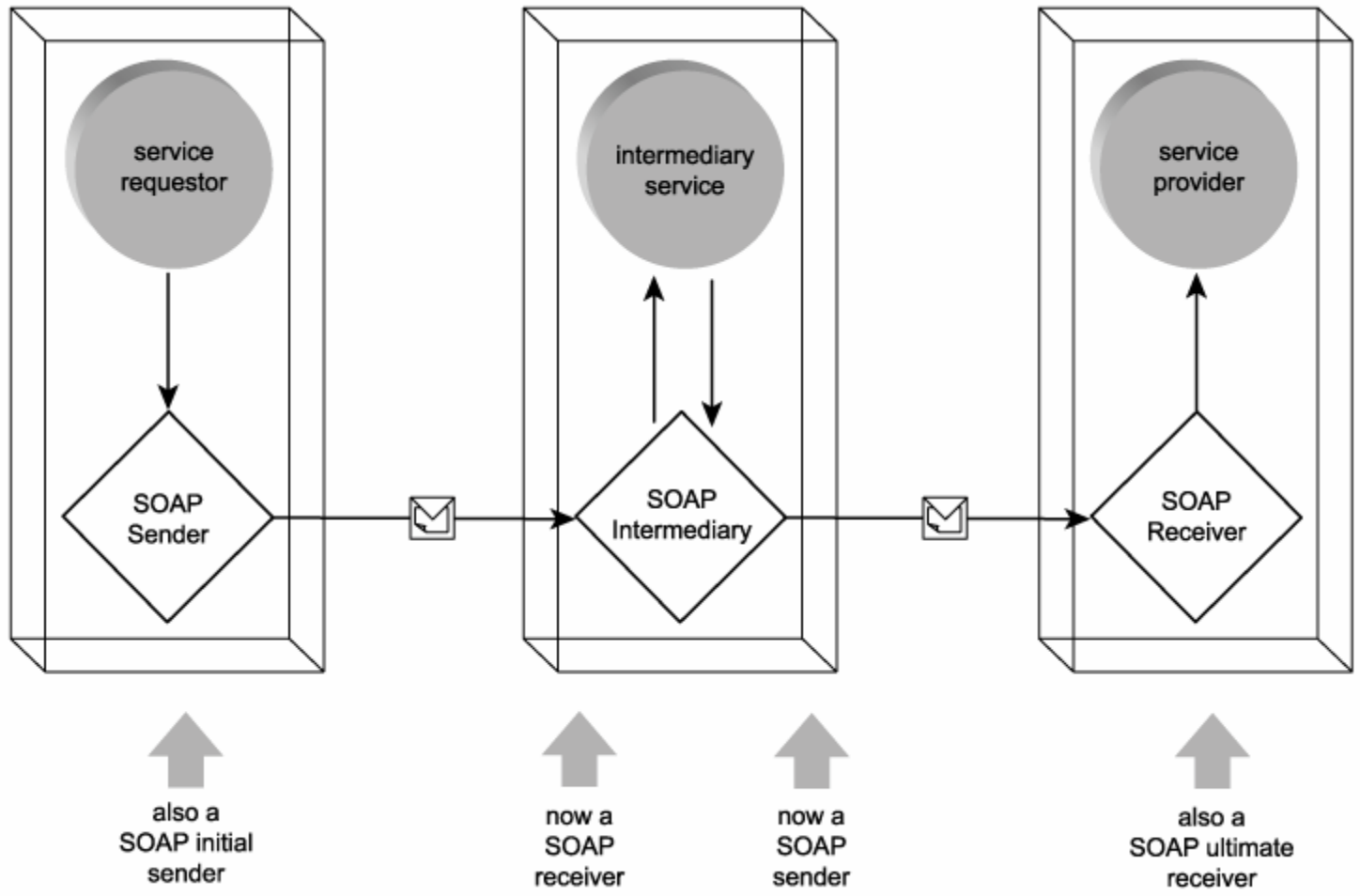
The positioning of SOAP nodes within a message transmission



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.

Different types of SOAP nodes involved with processing a message



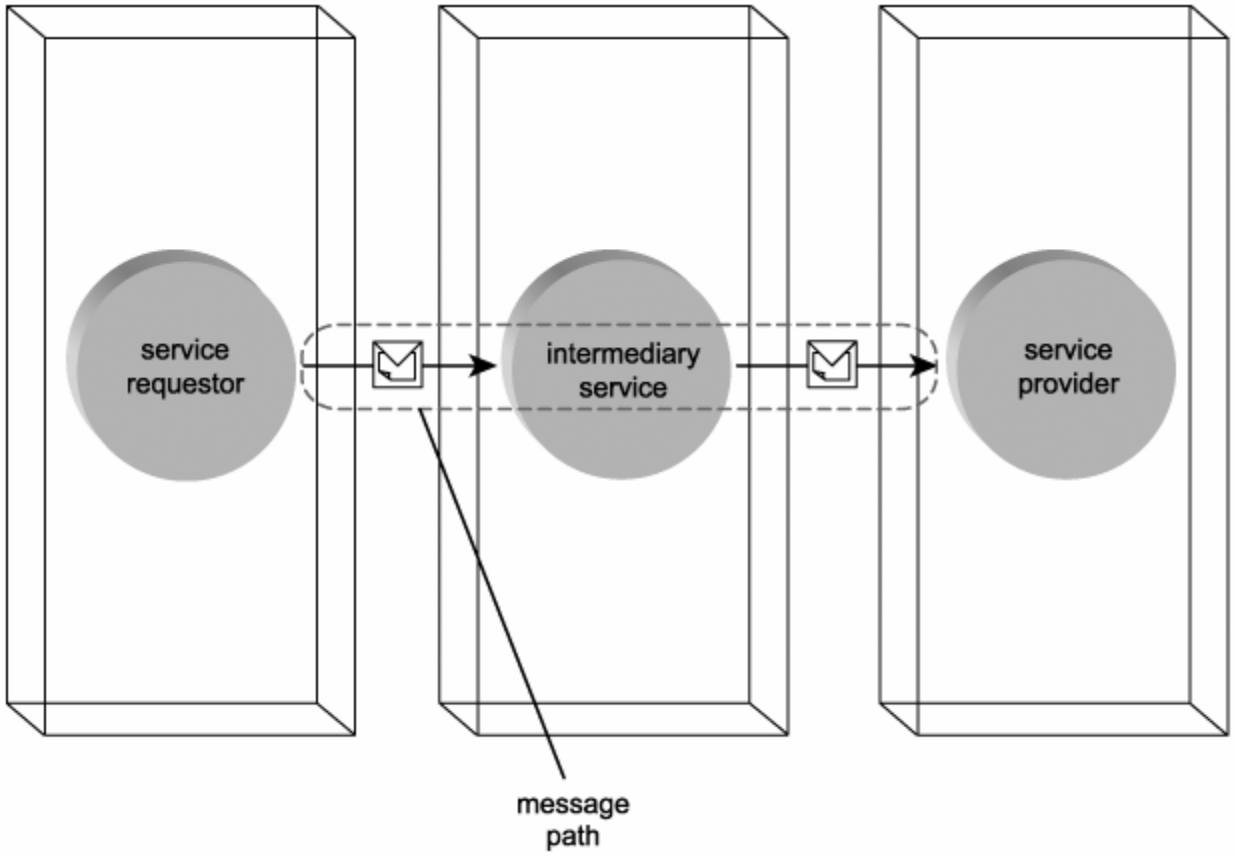
SOAP intermediaries

- 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.
- 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.

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. Therefore, 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.

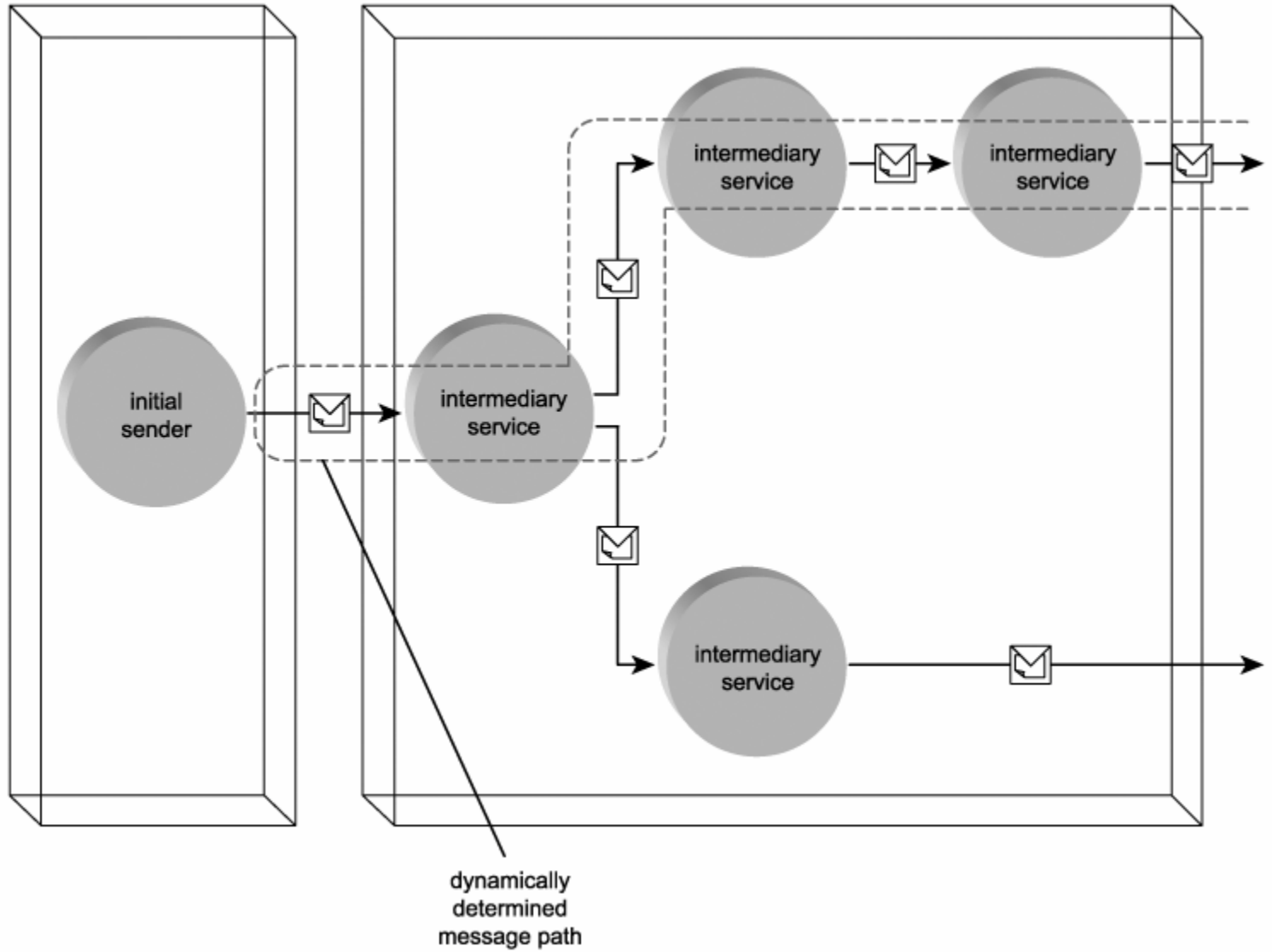
A message path consisting of three Web services



Message paths

- Note also that 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.

A message path determined at runtime



Message paths

- When used within the context of SOAP nodes, this term is qualified and therefore 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

Case Study

- Revisiting our invoice submission scenario one last time, we can establish both logical and physical views of the path along which the invoice SOAP message travels.
- From a logical perspective, the message path is always the same. The RailCo Invoice Submission Service requestor acts as the initial sender and is therefore the starting point of the path. The first service provider the message encounters is the TLS Load Balancing intermediary. This service then becomes the next service requestor and forwards the message to the Accounts Payable Service provider. As the last service provider along the path, this Web service becomes the ultimate receiver. This establishes a logical message path consisting of three services.
- The corresponding SOAP message path is not as predictable. Because the Load Balancing Service will only decide which physical server to route a message to when it actually receives and processes the message, the ultimate SOAP receiver is not determined until runtime.

Summary of Key Points

- The SOAP messaging framework fulfills the need for SOA's reliance on "independent units of communication," by supporting the creation of intelligence-heavy, document-style, and highly extensible messages.
- SOAP messaging establishes a standard message structure that includes an extensible header section used by numerous WS-* extensions to implement enterprise-level features.
- The SOAP node view of the Web services framework abstracts the physical communications framework, which consists of a series of SOAP servers.