SOA (Service-Oriented Architecture)

Explore SOA (service-oriented architecture), an important stage in the evolution of application development and integration.

Syllabus CSE 527 Model View Design Controller

Service Oriented Architecture

Objective

Define and understand service architecture concepts

Deploy web services orientation

Master Service oriented design

Course outcome

To learn fundamentals and techniques of SOA

Content

- Unit I Introduction
- UNIT II WEB SERVICES AND CONTEMPORARY SOA
- UNIT III SOA AND SERVICE-ORIENTATION
- UNIT IV BUILDING SOA (PLANNING AND ANALYSIS)
- UNIT V - SERVICE-ORIENTED DESIGN (9 hours)

CASES STUDIES

Case Study Background #1: Transit Line Systems, Inc.

- Transit Line Systems, Inc. (TLS) is a prominent corporation in the private transit sector.
- It employs more than 1,800 people and has offi ces in four cities. Although its primary line of business is providing private transit, it has a number of secondary business areas that include a maintenance and repair branch that outsources TLS service technicians
- to public transit sectors, and a tourism branch that partners with airlines and hotels. Of the 200 IT professionals who support TLS's automation solutions, approximately 50% are contractors who are hired on a per-project basis.
- TLS is a corporation that has undergone a great deal of change over the past decade. The identity and structure of the company has been altered numerous times, mostly because of corporate acquisitions and the subsequent integration processes. Its IT department has had to deal with a volatile business model and regular additions to its supported set of technologies and automation solutions.
- TLS's technical environment therefore is riddled with custom-developed applications and third-party products that were never intended to work together. The cost of business automation has skyrocketed, as the effort required to integrate these many systems has become increasingly complex and onerous. Not only has the maintenance of automation solutions become unreasonably expensive, but their complexity and lack of fl exibility have significantly slowed down the IT department's ability to respond to business change.

Case Study Background #1: Transit Line Systems, Inc.

Tired of having to continually invest in a non-functional technical environment, IT directors decide to adopt SOA as the standard architecture to be used for new applications. Web services are chosen as the primary technology-set to federate existing legacy systems. The driving motivation behind this decision is a desperate need to introduce enterprise-wide standardization and increase organizational agility.

Case Study Background #2: Midwest University Association

- Midwest University Association (MUA) is one of the oldest educational institutions west of the Mississippi in the continental U.S. It's rated among the top 10 leading universities in the engineering and research fields, and has six remote locations along with its main campus that employ more than 6,000 faculty and staff.
- Each program within the university has an independent IT staff and budget to support systems management. The remote campuses also have their own IT departments. Collaboration with external educational institutions is governed by an independent, central enterprise architecture group.
- There are various automated solutions for common processes, such as student enrollment, course cataloging, accounting, financials, as well as grading and reporting. The primary system for record keeping is an IBM mainframe that is reconciled every night with a batch feed from the individual remote locations. The different schools themselves employ a variety of technologies and platforms.

Case Study Background #2: Midwest University Association

- After a careful assessment of the existing infrastructure, it is decided to re-engineer several IT systems to a service-oriented architecture that will preserve legacy assets, simplify integration between various internal and external systems, and improve channel experience for both the students and staff. The enterprise architecture group at MUA has proposed a phased adoption of SOA via the use of REST services that can be leveraged across schools and from remote locations.

Others cases

Identify around you many reals situation or real systems of organizations that can be effectively federated with SOA.

Ideas behind Soa

- Advantages ?
- Concepts ?
- Technologies ?
- ...?

What is SOA, or service-oriented architecture?

- SOA, or service-oriented architecture, makes software components reusable and interoperable via service interfaces.

- Services use common interface standards and architectural pattern so they can be rapidly incorporated into new applications.
- This removes tasks from the application developer who previously had to know how to connect or provide interoperability with existing functions.

SAO ?

- Each service in an SOA embodies the code and data required to execute a discrete business function. Service interfaces provide loose coupling, meaning they can be called with little or no knowledge of how the service is implemented underneath. They can be written in Java, Microsoft.Net, Cobol or any other programming language.
- **e.g.** checking a customer's credit, calculating a monthly loan payment, or processing a mortgage application
- Service interfaces are frequently defined using Web Service Definition Language (WSDL)

SOA?

- SOA represents an important stage in the evolution of application development and integration over the last few decades.
- The services are exposed using standard network protocols (http/SOAP, RESTFULL, ...) to send requests to read or change data.
- Service governance controls the lifecycle for development and at the appropriate stage the services are published in a registry. These services can be built from scratch but are often created by exposing legacy systems as service interfaces.

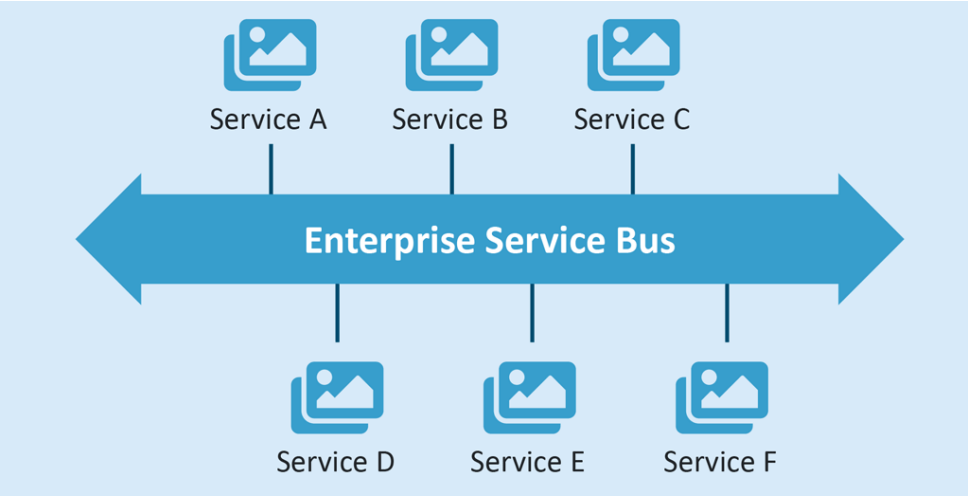
SOA?

- SOA emerged as a way of connecting an application to data or functionality housed in another system.
- Before SOA emerged, developers had to recreate complex point-to-point integration for each new development project. Exposing those functions through SOA services allowed the developer to simply reuse the existing capability and connect through the SOA ESB architecture (see below).
- Note that although SOA, and the more recent **microservices** architecture, share many words in common(i.e. "**service**" and "**architecture**"), they are only loosely related and, in fact, operate at different scopes, as discussed later in this article.

ESB?

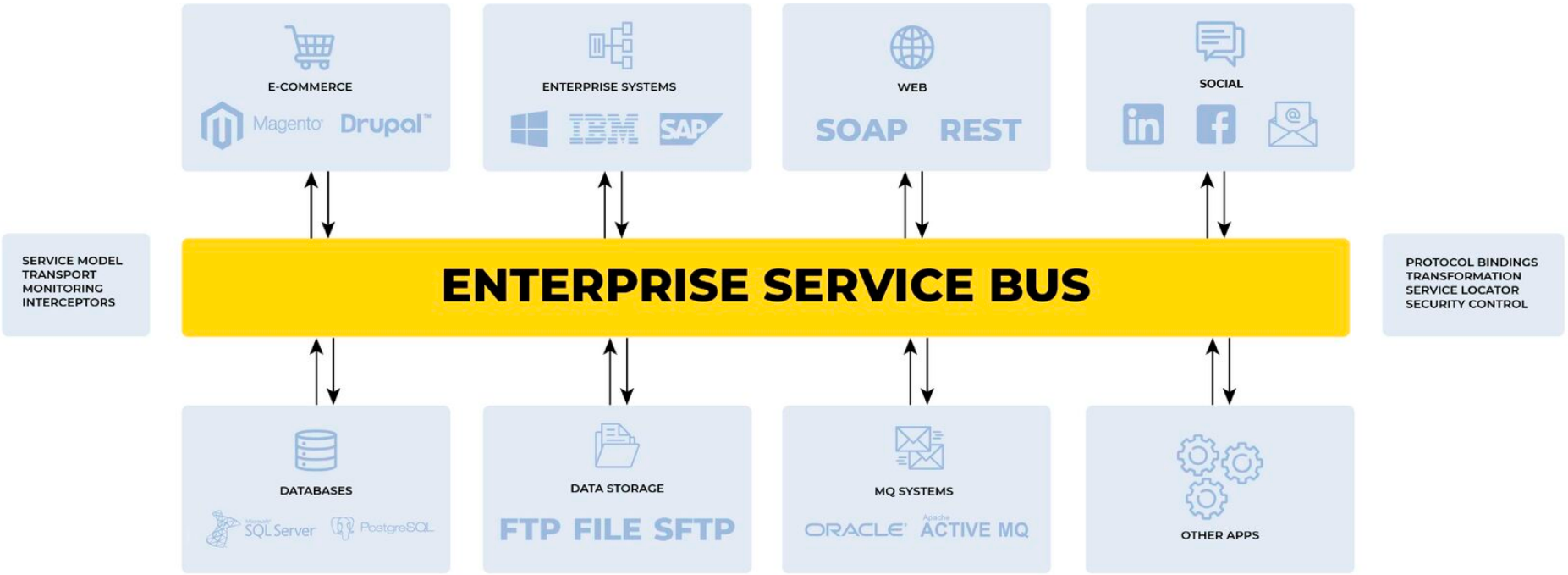
- An ESB, or enterprise service bus, is an architectural pattern whereby a centralized software component performs integrations between applications.
- It can make these integrations and transformations available as a service interface for reuse by new applications.
- The ESB pattern is typically implemented using a specially designed integration runtime and toolset.

ESB ILLustration?



iPaaS: an integration approach for the cloud era
or
ESB: an integration approach from the 90s

ESB ILLustration?



Benefits of SOA

Compared to the architectures that preceded it, SOA offered significant benefits to the enterprise:

- Greater business agility; faster time to market: Reusability is key.
- Ability to leverage legacy functionality in new markets
- Improved collaboration between business and IT

Benefits of SOA >> Greater business agility

The service oriented architectural approach supports scenarios for application integration, data integration and automation of business processes or workflows.

- The efficiency of assembling applications from reusable services - i.e. building blocks, rather than rewriting and reintegrating with every new development project - enables developers to build applications much more quickly.

>> Ability to leverage legacy functionality in new markets

- A well-crafted SOA enables developers to easily take functionality ‘locked’ in one computing platform or environment and extend it to new environments and markets.
- For example, many companies have used SOA to expose functionality from mainframe-based financial systems to new web applications,
- enabling their customers to serve themselves to processes and information previously accessible only through direct interaction with the company’s employees or business partners.

>> Improved collaboration between business and IT

- In an SOA, services can be defined in business terms (e.g., ‘generate insurance quote’ or ‘calculate capital equipment ROI’).
- This enables business analysts to work more effectively with developers on important insight
- such as the scope of a business process defined using services or the business implications of changing a process
- that can lead to a better result.

Advantages of SOA:

- **Service reusability:** In SOA, applications are made from existing services. Thus, services can be reused to make many applications.
- **Easy maintenance**: As services are independent of each other they can be updated and modified easily without affecting other services.
- **Platform independent**: SOA allows making a complex application by combining services picked from different sources, independent of the platform.
- **Availability:** SOA facilities are easily available to anyone on request.
- **Reliability:** SOA applications are more reliable because it is easy to debug small services rather than huge codes
- **Scalability:** Services can run on different servers within an environment, this increases scalability

Disadvantages of SOA:

- **High overhead:** A validation of input parameters of services is done whenever services interact this decreases performance as it increases load and response time.
- **High investment**: A huge initial investment is required for SOA.
- **Complex service management**: When services interact they exchange messages to tasks. the number of messages may go in millions. It becomes a cumbersome task to handle a large number of messages.

SOA Examples

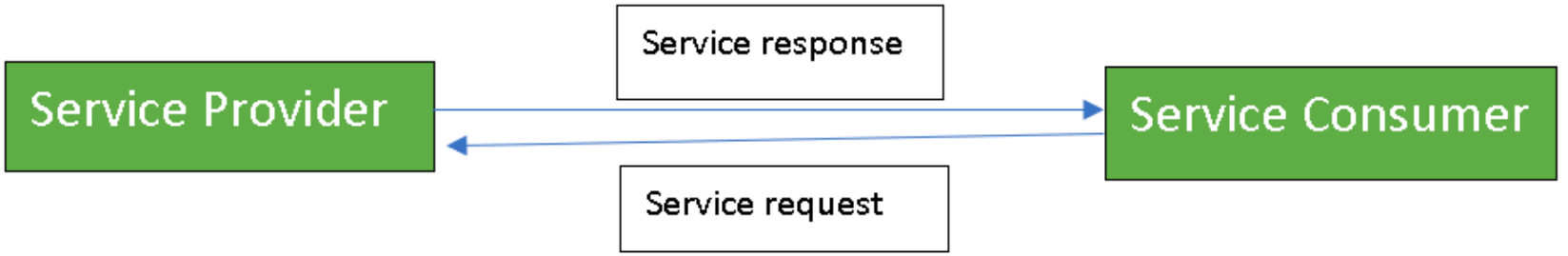
By 2010, SOA implementations were going full steam at leading companies in virtually every industry. For example:

- **Delaware Electric turned** to SOA to integrate systems that previously did not talk to each other, resulting in development efficiencies that helped the organization stay solvent during a five-year, state-mandated freeze on electric rates.
- **Cisco** adopted SOA to make sure its product ordering experience was consistent across all products and channels by exposing ordering processes as services that Cisco's divisions, acquisitions, and business partners could incorporate into their websites.
- **Independence Blue Cross** (IBC) of Philadelphia implemented an SOA to ensure that the different constituents dealing with patient data—IBC customer service agents, physicians' offices, IBC web site users—were working with the same data source (a 'single source of truth').
- SOA infrastructure is used by many armies and air forces to deploy situational awareness systems.
- SOA is used to improve healthcare delivery.
- Nowadays many apps are games and they use inbuilt functions to run. For example, an app might need GPS so it uses the inbuilt GPS functions of the device. This is SOA in mobile solutions.
- SOA helps maintain museums a virtualized storage pool for their information and content

Major roles within SOA

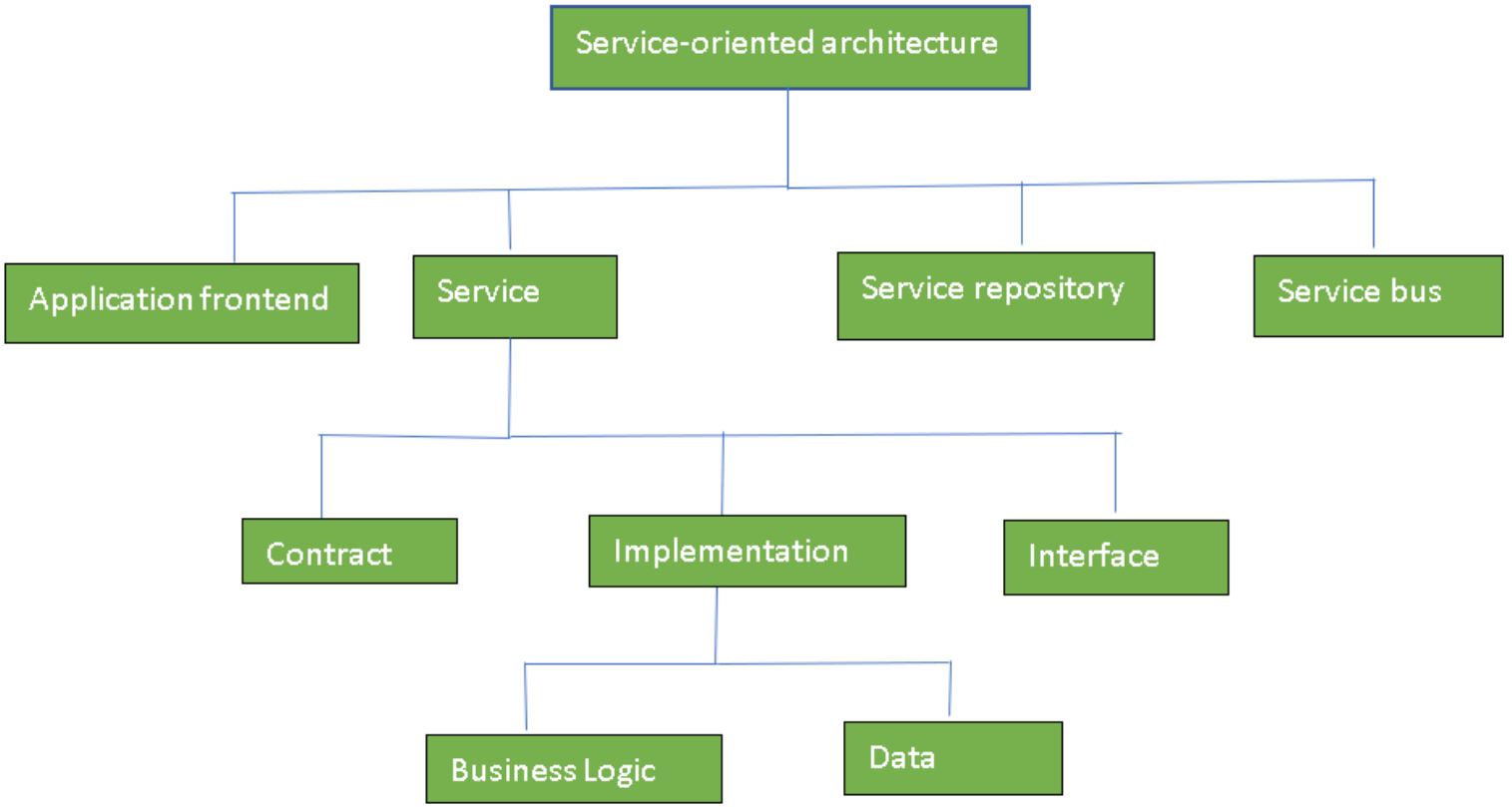
- **Service provider**: It is the maintainer of the service and the organization that makes available one or more services for others to use. To advertise services, the provider can publish them in a registry, together with a service contract that specifies the nature of the service, how to use it, the requirements for the service, and the fees charged.
- **Service consumer**: The service consumer can locate the service metadata in the registry and develop the required client components to bind and use the service.

Major roles within SOA



Services might aggregate information and data retrieved from other services or create workflows of services to satisfy the request of a given service consumer. This practice is known as **service orchestration** Another important interaction pattern is **service choreography**, which is the coordinated interaction of services without a single point of control.

Components of SOA:



Guiding Principles of SOA:

- **Standardized service contract**: Specified through one or more service description documents.
- **Loose coupling**: Services are designed as self-contained components, maintain relationships that minimize dependencies on other services.
- **Abstraction**: A service is completely defined by service contracts and description documents. They hide their logic, which is encapsulated within their implementation.
- **Reusability**: Designed as components, services can be reused more effectively, thus reducing development time and the associated costs.
- **Autonomy**: Services have control over the logic they encapsulate and, from a service consumer point of view, there is no need to know about their implementation.
- **Discoverability**: Services are defined by description documents that constitute supplemental metadata through which they can be effectively discovered. Service discovery provides an effective means for utilizing third-party resources.
- **Composability**: Using services as building blocks, sophisticated and complex operations can be implemented. Service orchestration and choreography provide a solid support for composing services and achieving business goals.

SOA vs. microservices

- **SOA is an integration architectural style and an enterprise-wide concept**. It enables existing applications to be exposed over loosely-coupled interfaces, each corresponding to a business function, that enables applications in one part of an extended enterprise to reuse functionality in other applications.
- **Microservices architecture is an application architectural style and an application-scoped concept.** It enables the internals of a single application to be broken up into small pieces that can be independently changed, scaled, and administered. It does not define how applications talk to one another—for that we are back to the enterprise scope of the service interfaces provided by SOA.
-

SOA vs. microservices

Monolithic

Startups,
Small Apps

Small resource
base

SOA

Enterprise apps with
complex operations

Microservices

Complex large-scale
systems

Multiple skilled teams

Serverless

Client-heavy apps

Fast-growing and
rapidly changing apps

High-latency
background tasks

Chap1: Fondemantals of SOA

Fundamental SOA

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.
- What distinguishes the service-oriented approach to separating concerns is the manner in which it achieves separation.
- Individual companies are service-oriented in that each provides a distinct service that can be used by multiple consumers.
- Collectively, these businesses comprise a business community.
- It makes sense for a business community not to be served by a single business outlet providing all services.
- By decomposing the community into specialized, individual outlets, we achieve an environment in which these outlets can be distributed.
- "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.

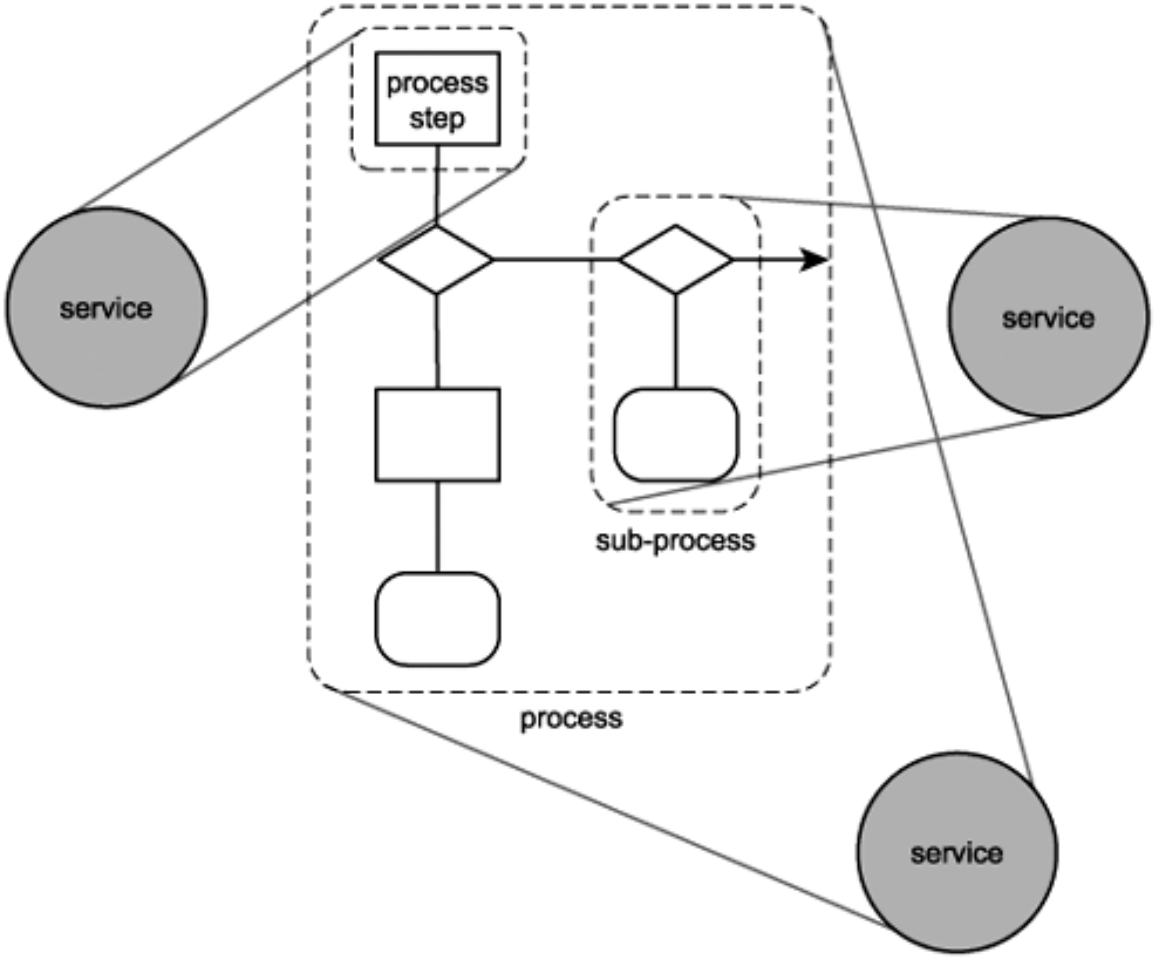
Fundamental SOA

- Distributing automation logic into separate units is nothing new.
- Even in a distributed business community, if we impose overbearing dependencies, we could inhibit the potential of individual businesses.
- By empowering businesses to self-govern their individual services, we allow them to evolve and grow relatively independent from each other.
- Though we encourage independence within our business outlets, we must still ensure that they agree to adhere to certain baseline conventionsfor 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.
- 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.
- 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.
- Therefore, the size and scope of the logic represented by the service can vary.

Fundamental SOA

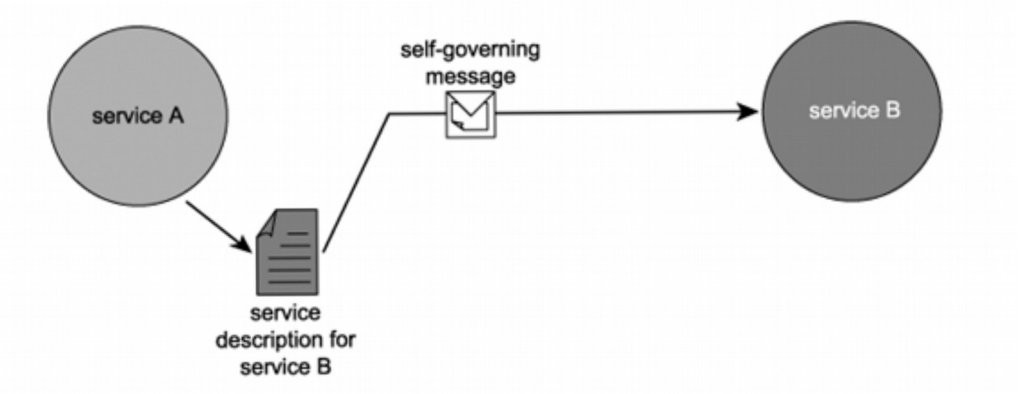
- For example, business automation solutions are typically an implementation of a business process.
- This process is comprised of logic that dictates the actions performed by the solution.
- As shown in Figure 3.
- 1, when building an automation solution consisting of services, each service can encapsulate 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.
- In the latter two cases, the larger scope represented by the services may encompass the logic encapsulated by other services.
- Services can encapsulate varying amounts of logic.

Fundamental SOA: Services can encapsulate varying amounts of logic.



How services relate ?

- Within SOA, services can be used by other services or other programs.
- For services to interact, they must be aware of each other.
- This awareness is achieved through the use of service descriptions. Figure bellow illustrates that service A is aware of service B because service A has B's service description.

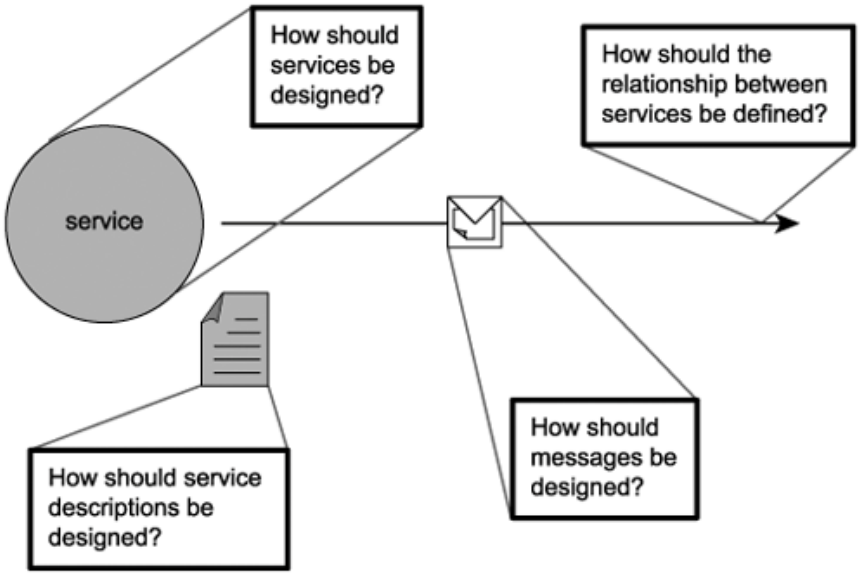


How services communicate

- 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.
- Services that provide service descriptions and communicate via messages form a basic architecture.
- What distinguishes ours is how its three core components (services, descriptions, and messages) are designed.

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 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 principles here:Loose coupling Services, Service contract Services, 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
- Composability Collections of services can be coordinated and assembled to form composite services.
- Discoverability Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.
- 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.

•Case Study

- RailCo's accounting solution exists as a two-tier client-server application, where the bulk of application logic resides within an executable deployed on client workstations. The details of this application are described in the next chapter. For now, let's identify two of the primary tasks this application can perform:

- Enter Customer Purchase Order
- Create Customer Order
- The completion of each task involves a series of steps that constitute a business process. This process was originally modeled using standard workflow logic and then implemented as part of a packaged solution. Within the application, the process may or may not be separately represented by individual sets of programming routines. Regardless, it is compiled into a single executable that provides a fixed manner in which the process is automated.

•Case Study

Within a service-oriented business model, the logic behind each process would be partitioned into one or more services. If multiple services are used, the execution of the entire process would involve the composition of these services. In this case, each service may represent a sub-process or even a single step within the process that can be executed independently. For example, the Create Customer Order Process may consist of the following sub-processes:

- Retrieve Purchase Order Data
- Check Inventory Availability
- Generate Back Order
- Publish Customer Order
- As we know, a process in its entirety can be viewed and modeled as a service. Additionally, one or more processes can be combined to represent an even larger service. For example, the Create Customer Order and Generate Customer Invoice Processes may be combined to form a single Order Processing Process.

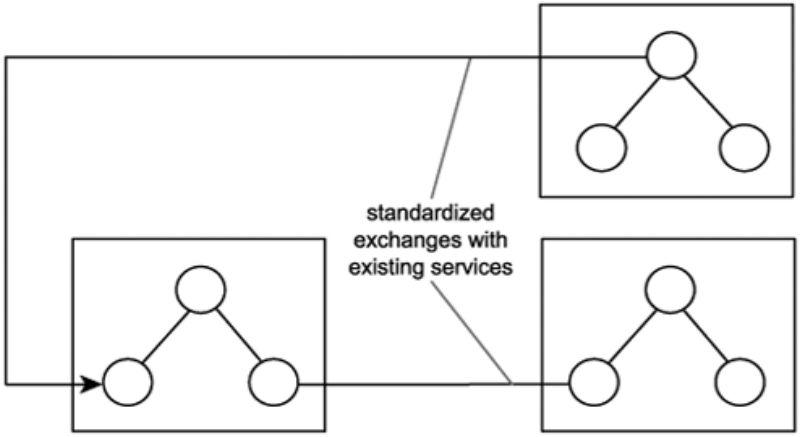
•Case Study

Finally, we would also expect these processes to be flexible so that they can incorporate processes or resources that exist elsewhere in the enterprise. For instance, we may decide to extend the Order Processing Process to include a sub-process that automatically retrieves the customer's current accounts payable mailing address. This sub-process may already exist as part of a separate Customer Contact Reporting Process.

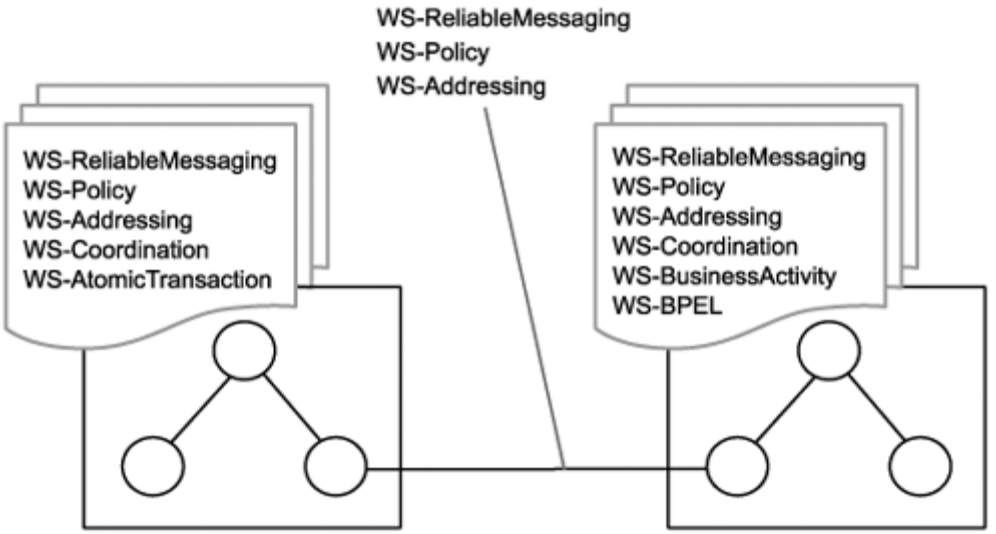
To implement such a model, we need a technical architecture capable of providing the following:

- The ability for business automation logic to be partitioned into units so as to properly represent services.
- The ability for these units of logic to be relatively independent of each other so as to support the requirement for them to participate in different compositions.
- The ability for these units of logic to communicate with each other in such a manner that their respective independence is preserved.
- The fundamental characteristics of service encapsulation, loose-coupling, and messaging, as realized through service-orientation principles and the Web services technology set, collectively fulfill these requirements through the implementation of a primitive SOA.

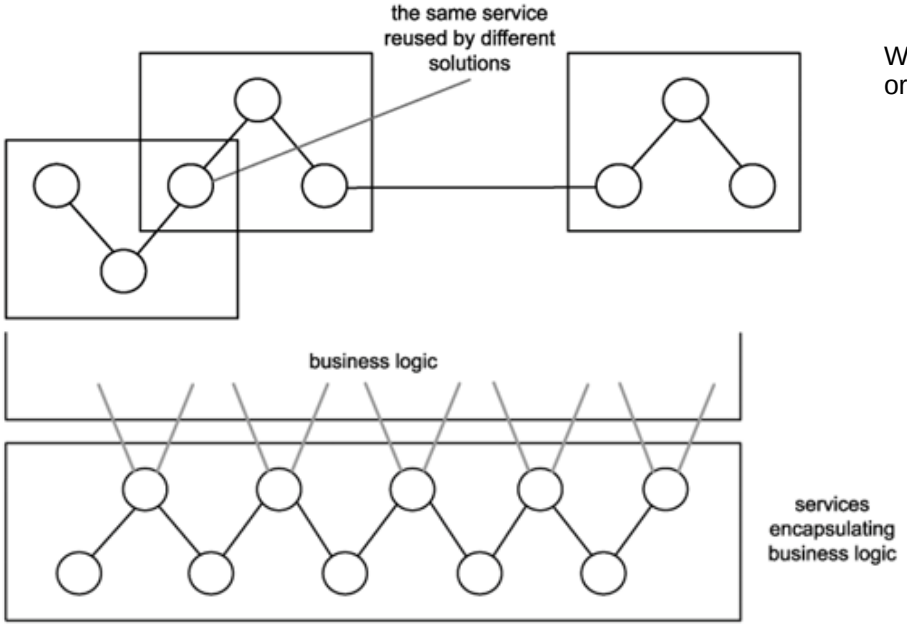
Comtemparay soa

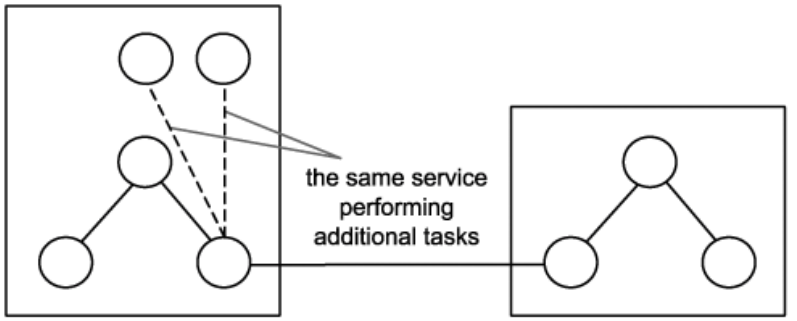


Contemporary SOA fosters intrinsic interoperability

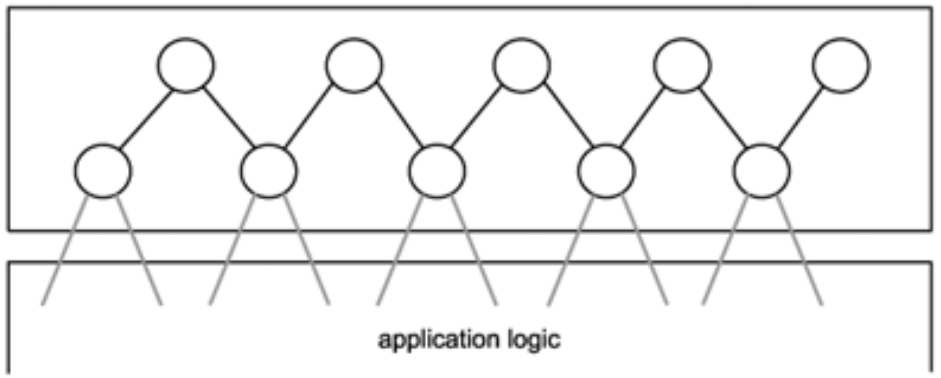


WS-* platform allows for the creation of streamlined and optimized service-oriented architectures, applications, services, and even messages.

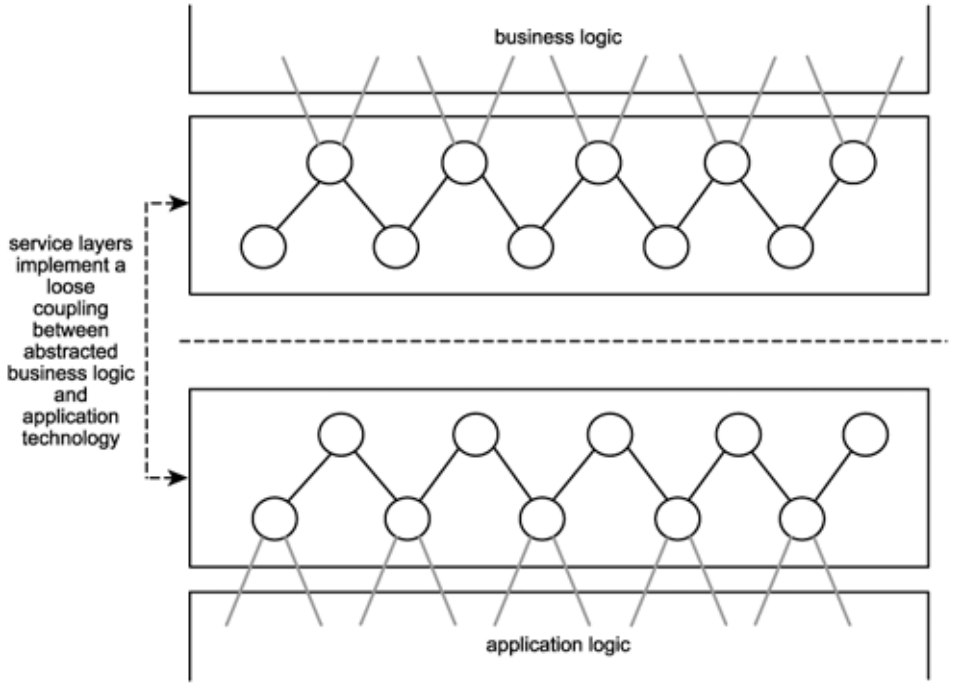




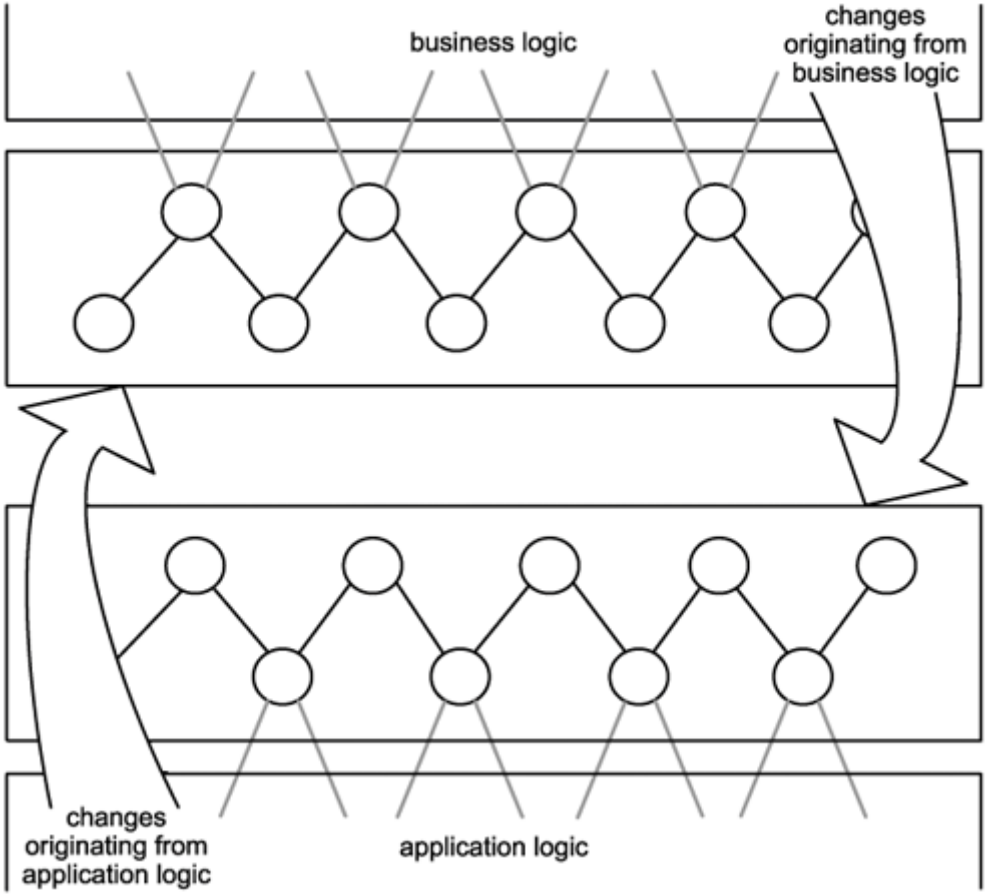
Comtemparay soa



services encapsulating (and abstracting) application logic and technology resources



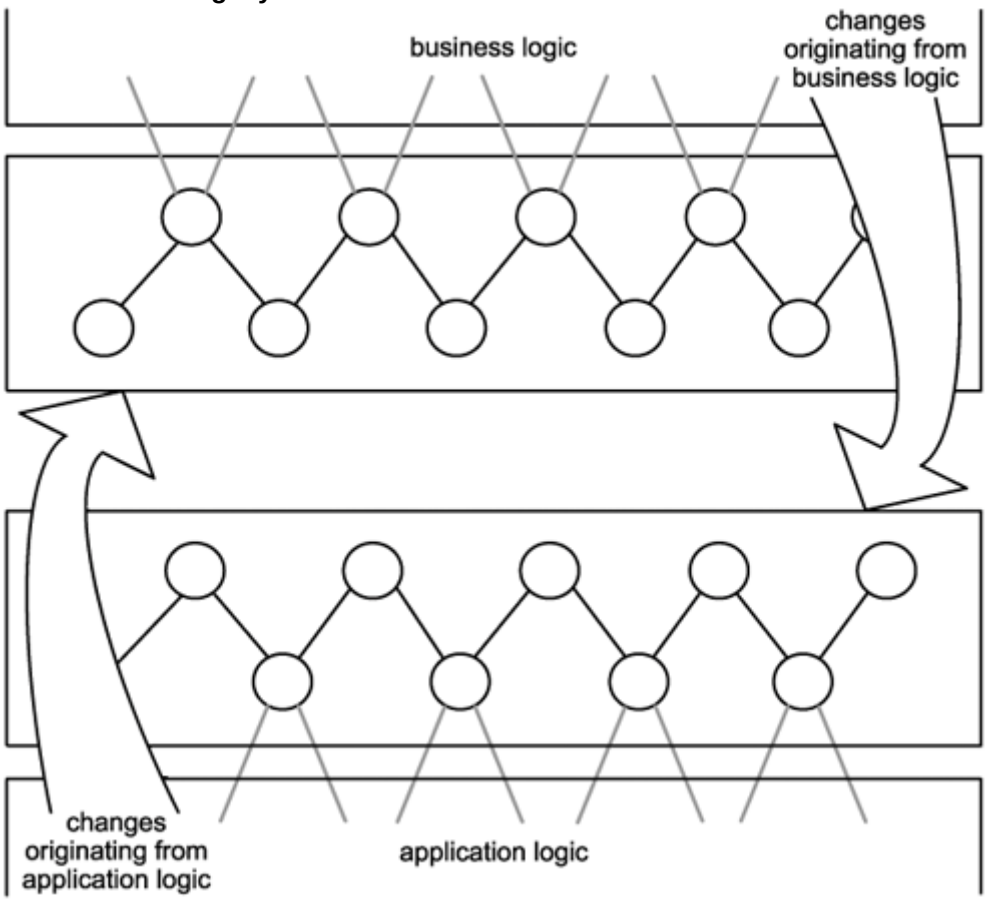
. Contemporary SOA promotes organizational agility



Comtemparay soa

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**

. Contemporary SOA promotes organizational agility

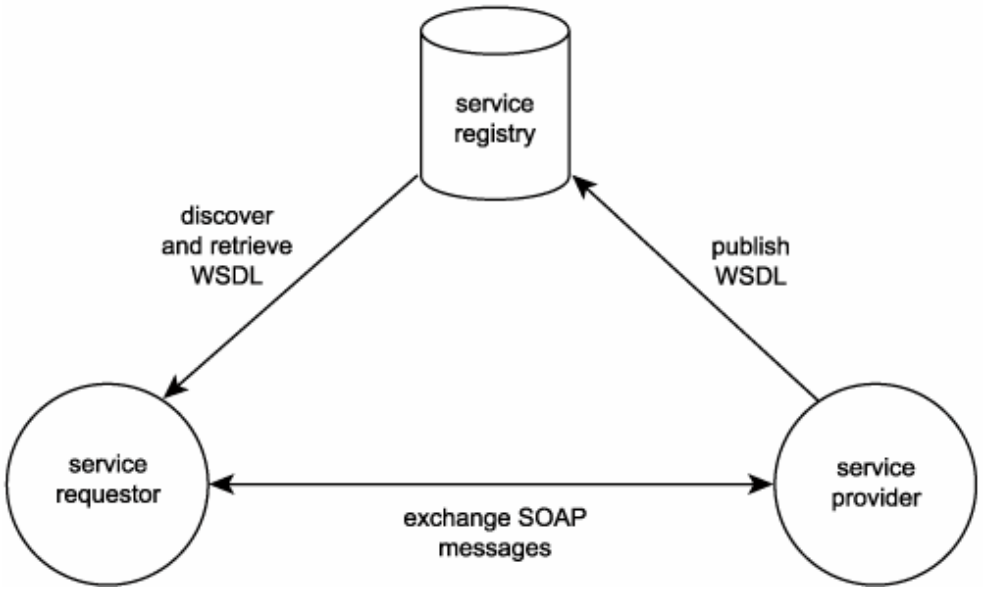


A loosely coupled relationship between business and application technology allows each end to more efficiently respond to changes in the other.

Common misperceptions about SOA

- Much of the confusion surrounding the meaning of SOA is caused by how this term has been used by the media and in marketing literature.
- The most common misperceptions relate to the use of Web services within distributed Internet architectures being mistaken as contemporary SOA.
- Some of the more dangerous assumptions about SOA are that service-oriented solutions are simple by nature, easy to build, and automatically interoperable.

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.

The core XML technology set has become a common part of distributed Internet architecture. It now also provides a foundation data representation and data management layer for SOA.

The first-generation Web services architecture grew out of the development of three key standards: WSDL, SOAP, and UDDI. While UDDI is still an optional discovery mechanism for most environments, WSDL and SOAP have become core technologies that build upon the XML layer to define the fundamental communications framework for SOA.

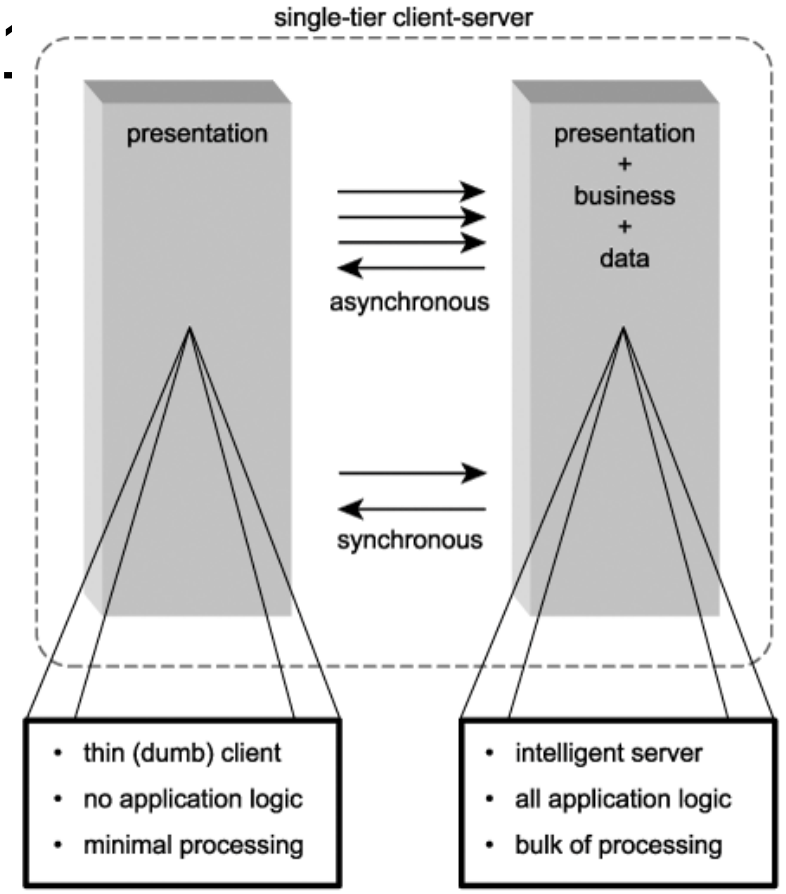
SOA fully leverages the roads paved by the XML and Web services initiatives. Its marriage of proven concepts with progressive technology has been fully embraced by the global IT community.

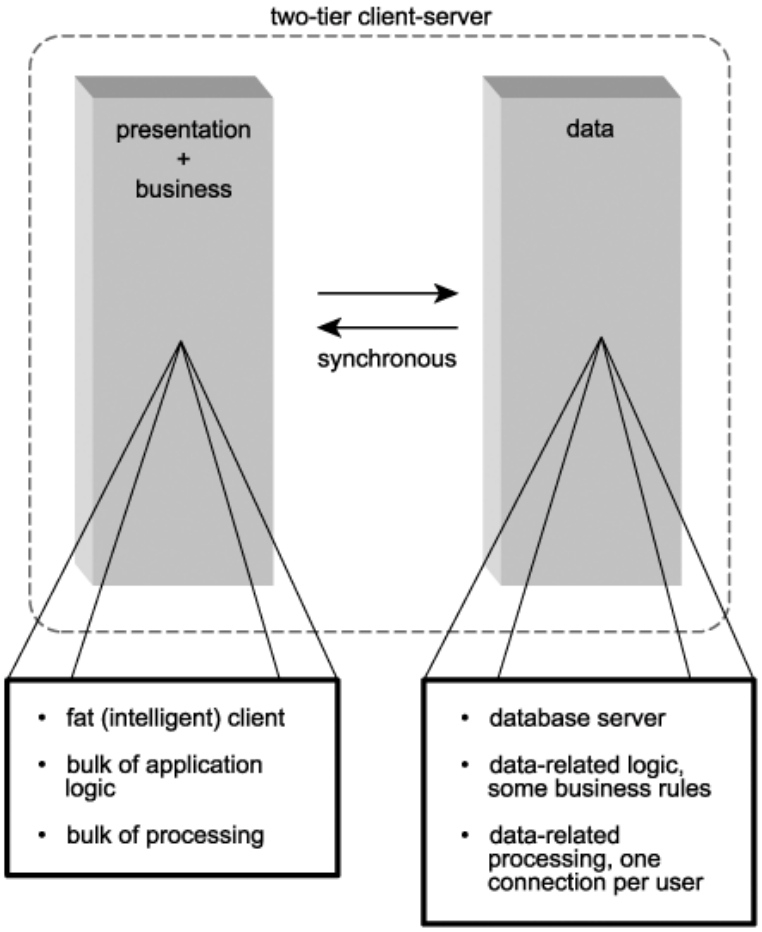
Though contemporary SOA has been shaped by the emergence and industry-wide acceptance of XML and Web services, the arrival of SOA introduces changes to the manner in which XML and Web services have been traditionally applied.

The continuing evolution of SOA (std. Org. and contrib. vendors)

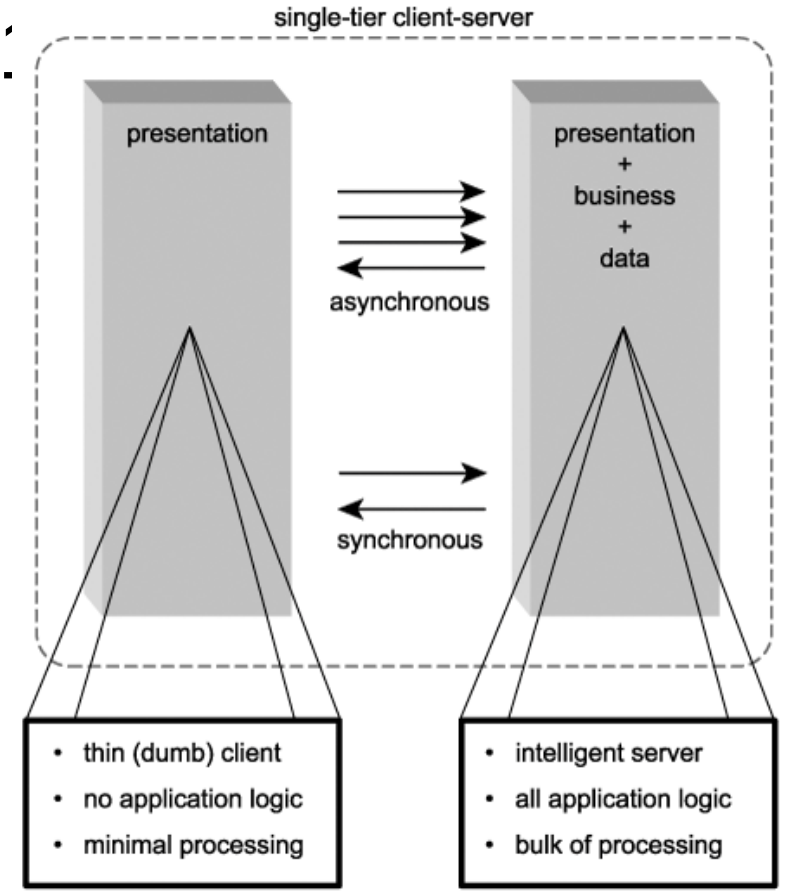
- The **W3C's** contributions to advancing the World Wide Web cannot be understated. In the SOA arena, its role has been primarily as a standards body responsible for specifications that provide core and generic functionality.
- **OASIS** evolved from an SGML standards organization to one focused almost exclusively on eBusiness specifications. Its overall goal is to support the creation of standards targeted at specific industries and to foster trade and commerce between eBusiness-enabled enterprises.
- As a group dedicated to the interoperability concerns of disparate platforms, the **WS-I** does not produce technology standards. Instead, this organization provides profile documents that establish a proven and tested collection of standards. Compliance to these profiles guarantees organizations that their environments support a level of industry-standard interoperability.
- Though standards organizations exist as separate entities, all are supported by and receive contributions from vendor representatives. Vendor contributions are motivated by a mixture of self-interest and common good.
- It is important to stay on top of the standards development landscape because it will allow you to plan a more educated migration toward SOA.

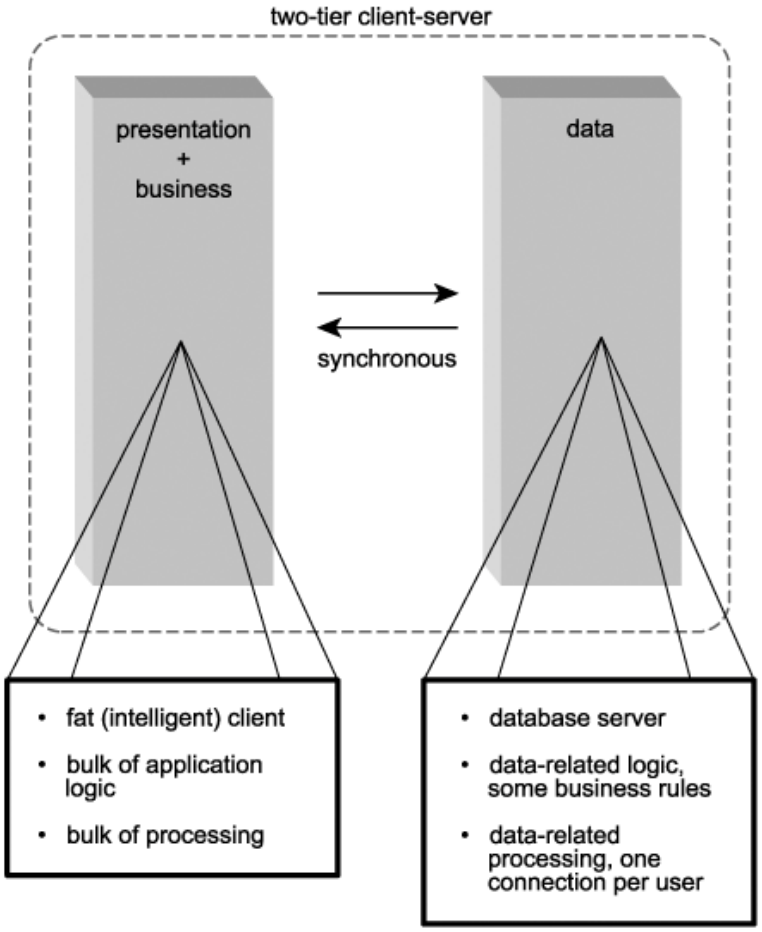
The roots of SOA (comparing SOA to past architectures)





The roots of SOA (comparing SOA to past architectures)





Case Study 1

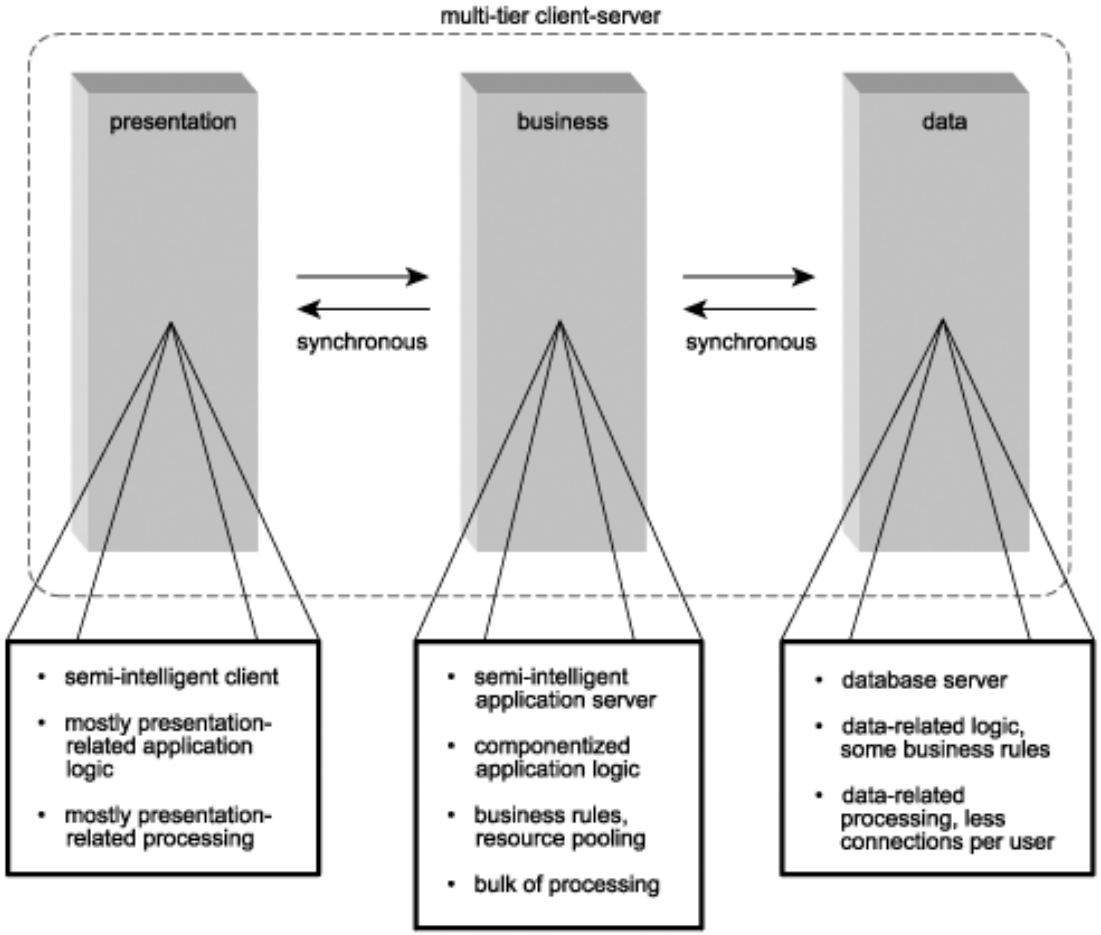
RailCo's accounting system is a classic two-tier client-server application. Its GUI front-end consists of a single executable designed for deployment on old Windows workstations. It provides user-interfaces for looking up, editing, and adding various accounting records. It also offers a financial reporting facility that can produce a fixed amount of statements with detailed or summarized accounting data.

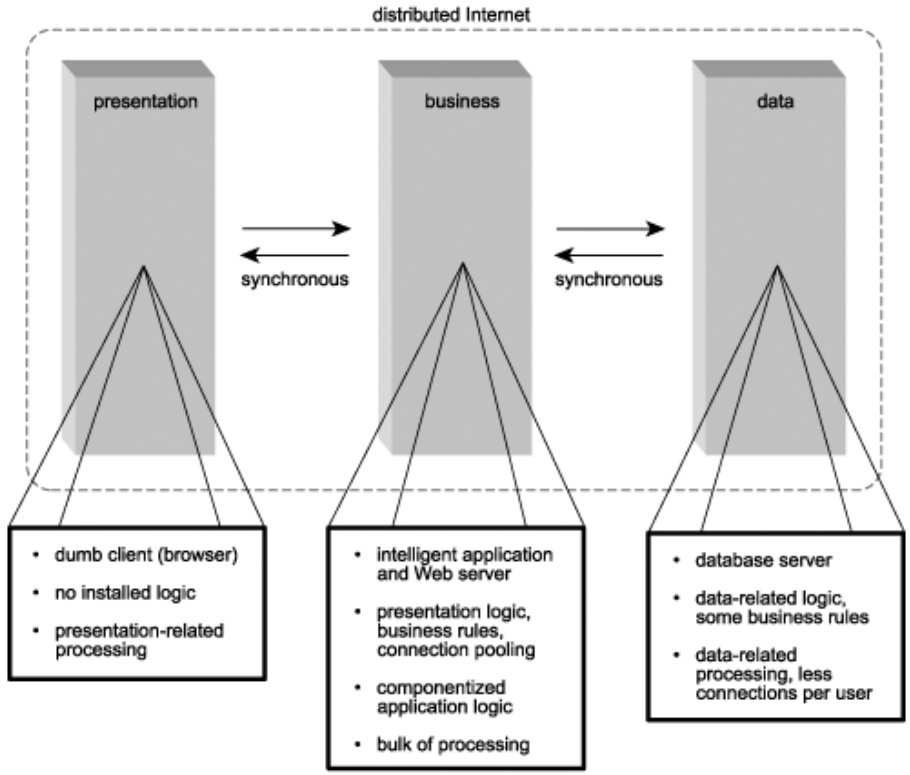
- Considering it's only ever had two to three users, there have never really been performance problems on the database end. The now outdated RDBMS that has been in place for the past decade has been reliable and has required little attention.
- However, problems with this application have surfaced:
- Operating system upgrades have introduced erratic behavior on some screens, resulting in unexplainable error messages. It is uncertain if these are caused by other programs that have been installed on the workstations.
- The workstations themselves have been rarely upgraded and have not kept pace with the hardware demands of recent software upgrades. After the accounting system launches, there is little more the user can do with the computer. As a result, employee productivity has been affected somewhat.
- Following a new records management policy and some billing procedure changes, a modification to the overall billing process was imposed on the accounting personnel. Because the accounting system was not designed to accommodate this change, employees are required to supplement the automated billing process by manually filling out supplementary forms.
- Fundamentally, this accounting system has been getting the job done. However, the actual accounting tasks performed by the users have become increasingly convoluted and inefficient. This is due to the questionable stability of the workstation environments and also because the system itself is not easily adaptable to changes in the processes it automates.

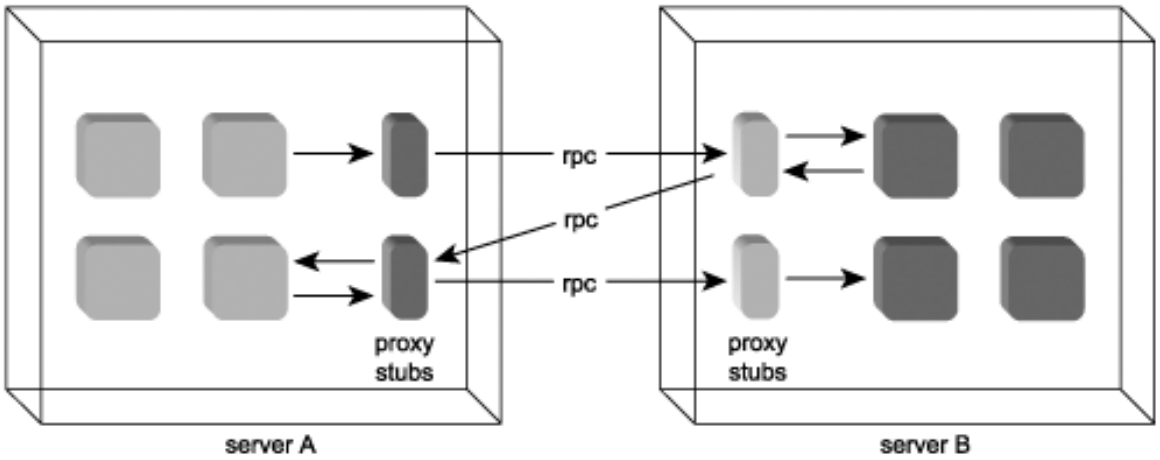
Case Study 1(solution)

SOA can address issues such as these, as follows:

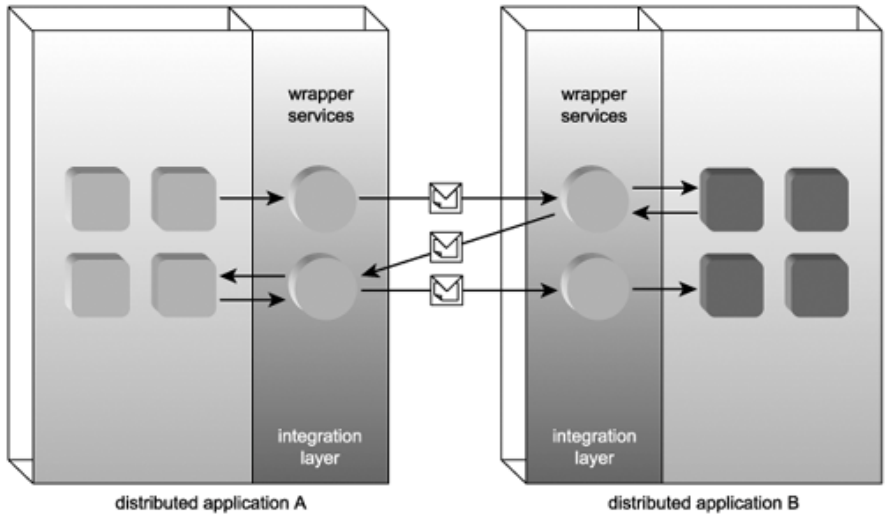
- Service-oriented solutions eliminate dependencies on user workstation environments by delegating all processing to the server-side (as regular distributed Internet applications already have been doing).
- SOA establishes an adaptable and extensible architectural model that allows solutions to be enhanced with minimal impact. Services can encapsulate existing legacy logic providing a standardized API that can plug into larger integrated solutions. Further, when building custom service-oriented applications, extensibility can be built into the solution environment, supporting future enhancements, also with minimal impact.







Components rely on proxy stubs for remote communication.



Web services as component wrappers

Case study 2

The TLS accounting system consists of a large, distributed component-based solution. Some 50 odd components host and execute various parts of the application logic. For performance and security reasons, some components have been deployed on separate application servers.

- Overall, the execution of a typical accounting task will involve four to five physical layers consisting of:
- A Web server hosting server-side scripts that relay HTTP requests to components on application servers and then relay responses from those components back to the browser clients.
- An application server hosting a controller component that generates a transaction context and manages more specialized components.
- A possible second application server hosting two or more business components that enforce specific business rules and perform various functions related to a particular business context. This server also may host one or more data components that encapsulate the data access logic required to interact with application repositories.
- A database server hosting a complete RDBMS environment.
- This enterprise solution has undergone many changes and enhancements over the past few years. Some of the primary issues that have arisen include:
- Initially, many components were custom developed to alter or extend existing functionality. Each redevelopment project has become increasingly expensive. This trend is being blamed on the overhead associated with the amount of testing and redeployment effort required to ensure that all pre-existing dependencies are not affected by any modification to a component's functionality.
- Because state management was never standardized, a design disparity has emerged. Some components manage state information by caching data in memory, while others use application server-deployed databases. This became an issue when XML was first introduced as a standard data format. Permanent state management designs already had a relational storage format in place that was incompatible with the required XML document structures.

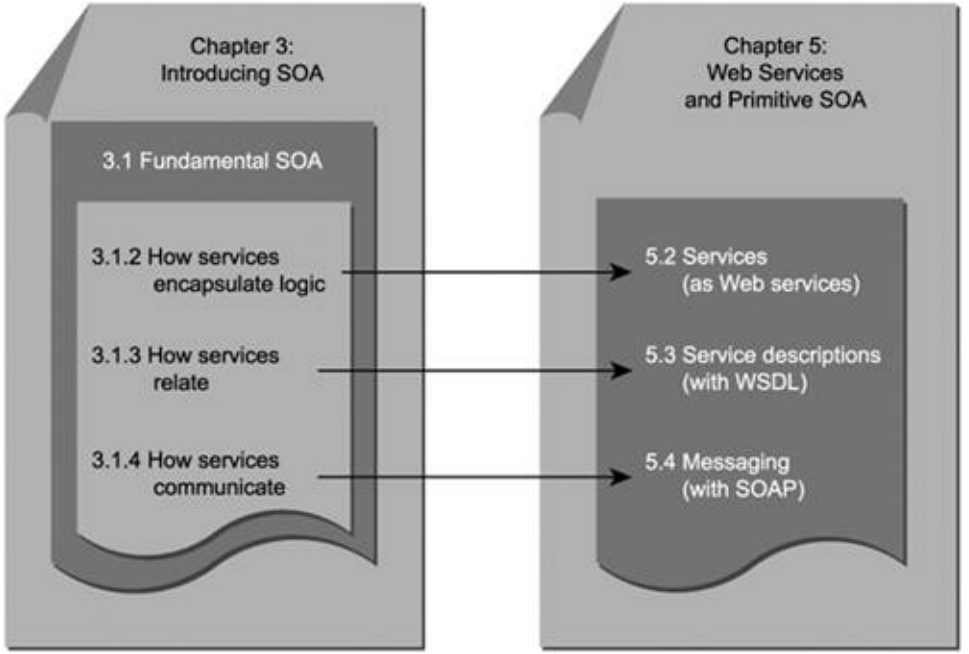
Case study 2 (sol.)

- SOA establishes a loosely coupled relationship between units of processing logic encapsulated as services. This allows the logic within each service boundary to be updated and evolved independently of existing service requestors, as long as the original service contract is preserved.
- SOA promotes the standardization of XML data representation throughout solution environments. Further, service statelessness is emphasized by deferring state management to the message level. This maximizes reuse, availability, and scalability of service logic but also provides a standardized state management approach.

The roots of SOA (comparing SOA to past architectures)

- SOA is a radical departure from client-server architecture. Current SOAs still employ some of the technologies originally used to build client-server applications. Though more sophisticated, SOAs introduce complexity that sharply contrasts the simplicity of a two-tier client-server architecture.
- Distributed Internet architecture has much in common with SOA, including a significant amount of its technology. However, SOA has distinct characteristics relating to both technology and its underlying design principles. For example, SOA introduces processing and security requirements that differ from distributed Internet architecture, and SOA administration is typically more complex due to its reliance on messaging-based communication.
- Traditional architectures have and can continue to use Web services within their own design paradigms. It's important to not confuse these architectures with SOA. Non-SOA use of Web services is typically found within distributed Internet architectures, where Web services are employed to mirror RPC-style communication.

Web Services and Primitive SOA



Ex sevice role: as the initiator, relayer, or the recipient of a message

Web services can be labeled using temporary and permanent classifications.

Temporary classifications relate to roles assumed by a service at runtime. For example, an intermediary service can transition through different roles while processing a message.

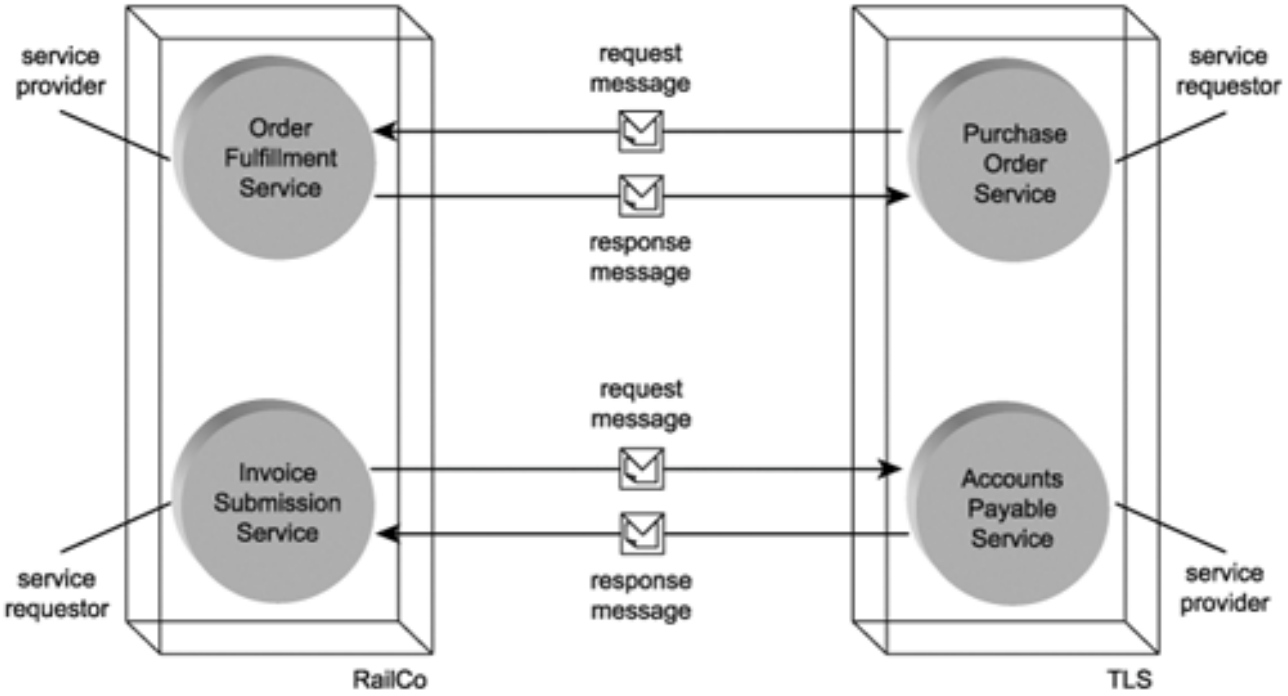
Service models refer to permanent classifications that represent the logic housed by the service, as well as its role within the overall solution. A service can belong to more than one service model.

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.
- 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.

Case study

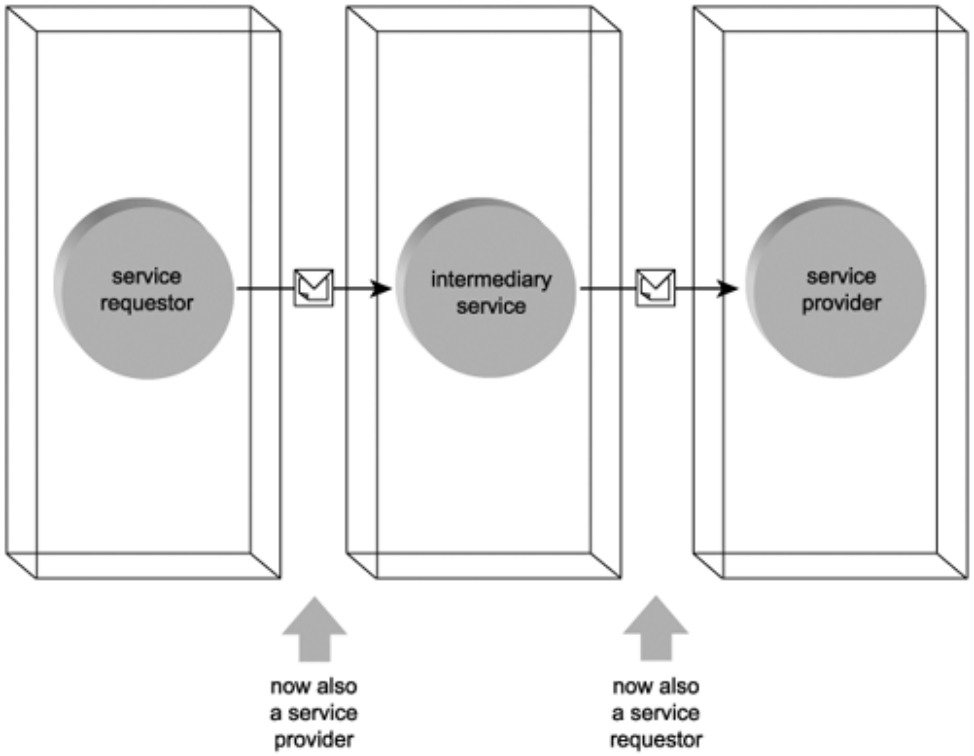
- **TLS and RailCo services swapping roles in different but related message exchanges.**



- RailCo's Order Fulfillment Service was the recipient of an order request. As the owner of this Web service, RailCo is the service provider entity.
- The roles are reversed in the second scenario, where RailCo acts as the service requestor (or service provider agent).

Intermediarie services

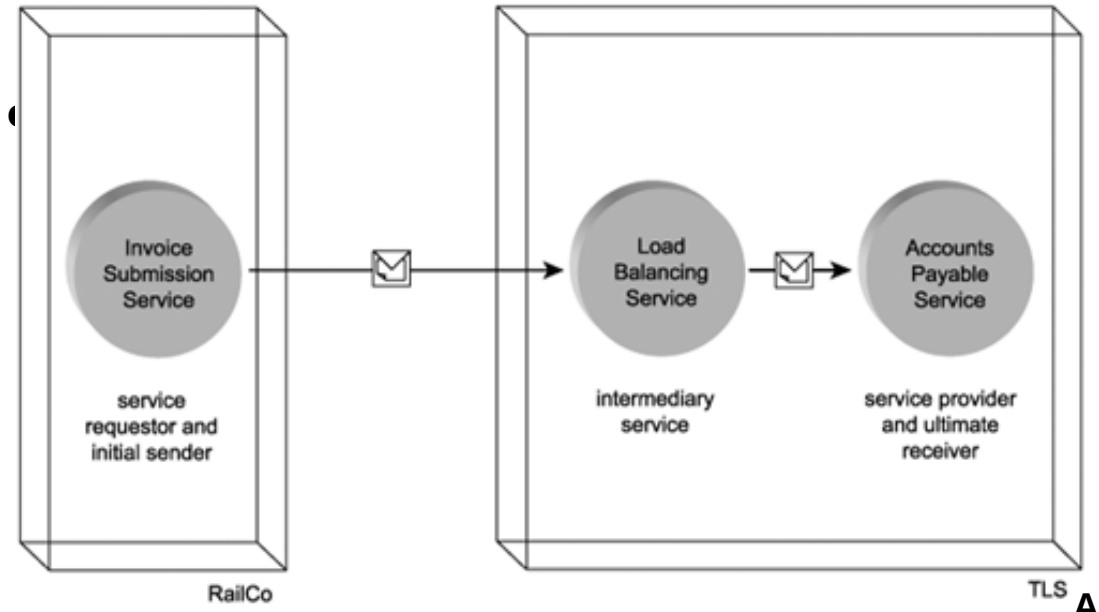
case



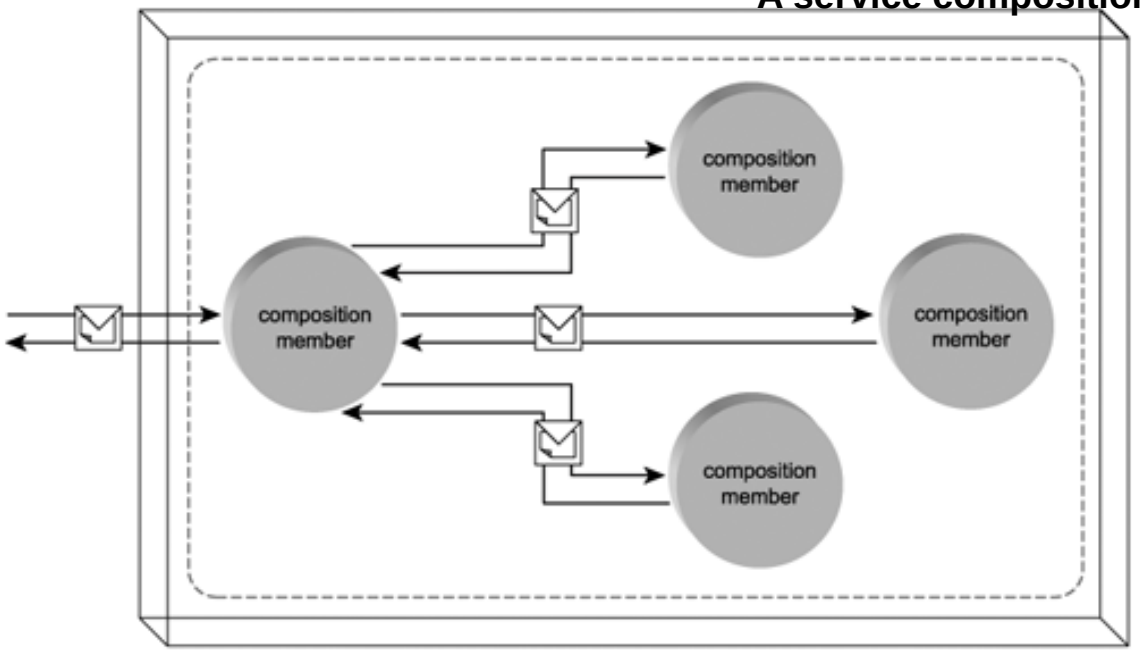
Case Study

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

Composites services

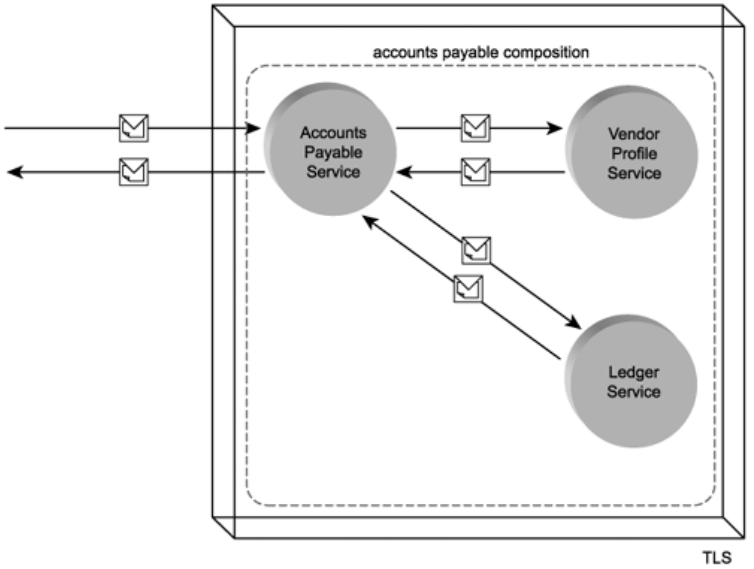


A service composition consisting of four members.



e.g

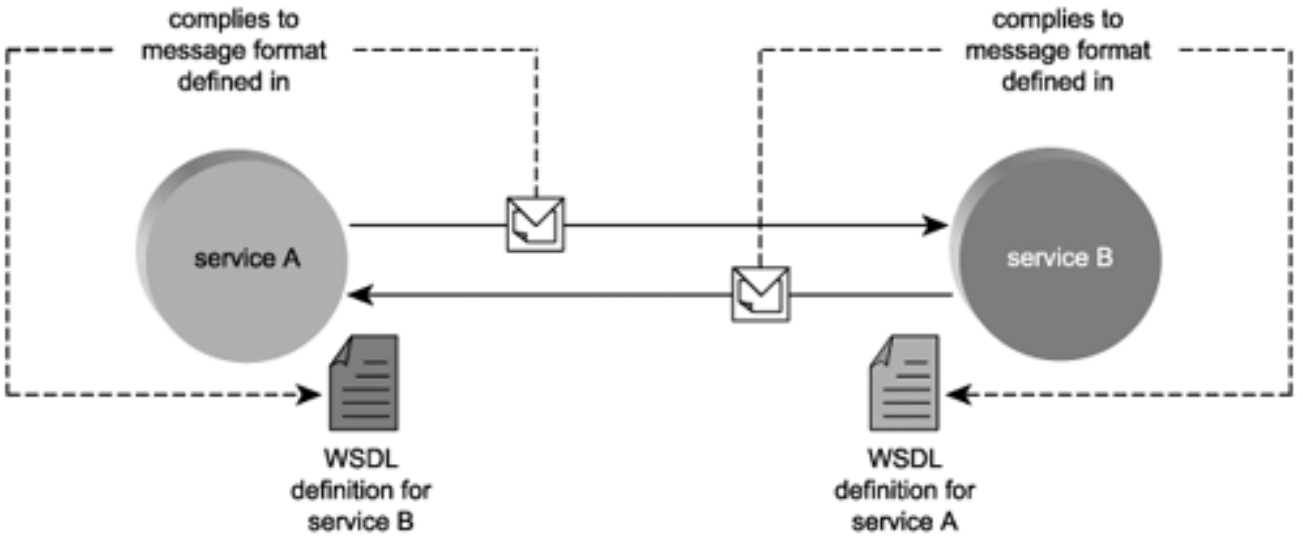
- When the TLS Accounts Payable Service receives an invoice, it invokes a series of additional services to fully process the invoice contents:
- It first uses the Vendor Profile Service to validate the invoice header data and link the invoice document to a vendor account.
- Next, the Accounts Payable Service extracts taxes and shipping fees and directly logs all amounts into the appropriate A/P accounts.
- 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



Service descriptions (with WSDL)

•WSDL definitions enable loose coupling between Services

- synthax

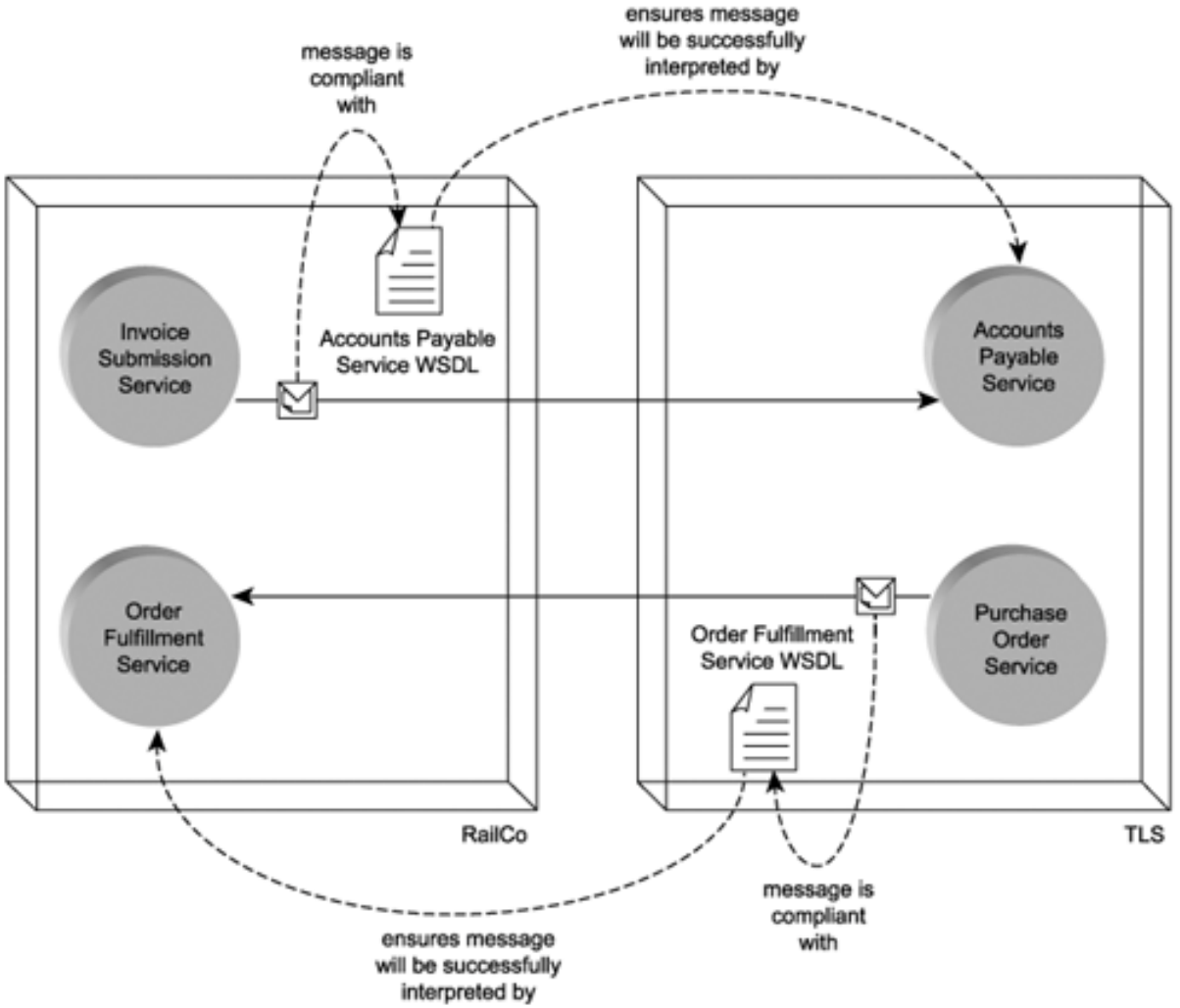


For RailCo to design its B2B Web services in full compliance with the TLS services, RailCo acquires the WSDL service description published by TLS for their Accounts Payable Service. This definition file then is used by developers to build the Invoice Submission Service so that it can process SOAP messages 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.

•WSDL definitions enable loose coupling between Services

- fig



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.

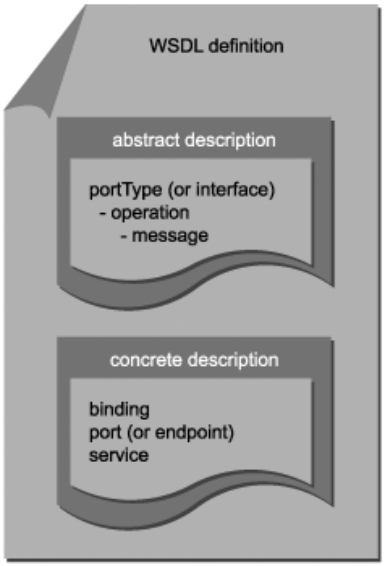
The Invoice Submission Service was built as a service requestor that issues messages compliant with the Accounts Payable WSDL. The Order Fulfillment Service was designed as a service provider according to published specifications by TLS. This guarantees TLS that its Purchase Order Service (acting as 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.

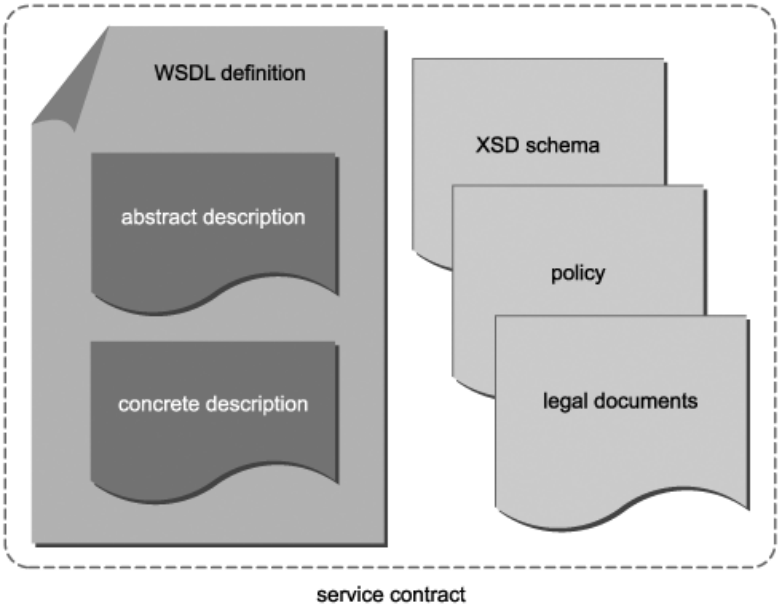
Service endpoints and service descriptions

A **WSDL** describes the point of contact for a service provider, also known as the service endpoint or just endpoint.

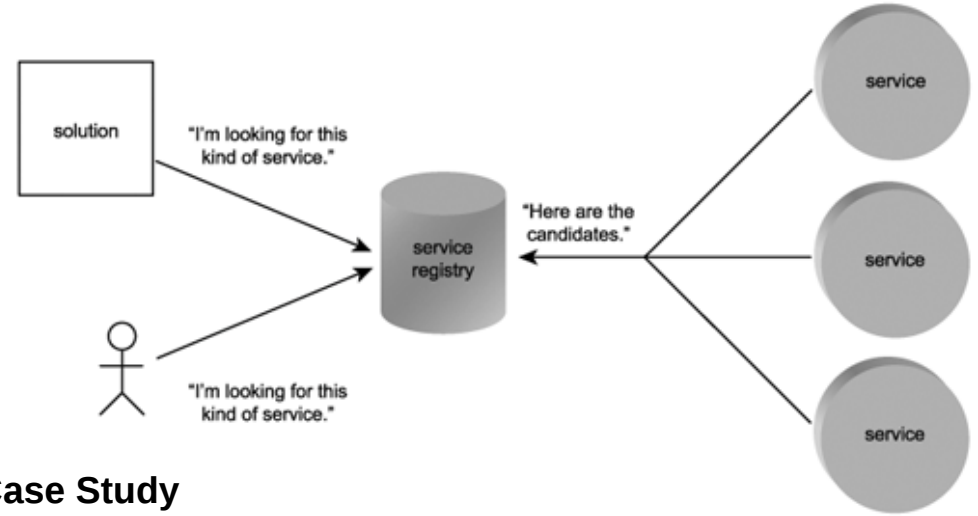
It provides a formal definition of the endpoint interface (so that requestors wishing to communicate with the service provider know exactly how to structure request messages) and also establishes the physical location (address) of the service.

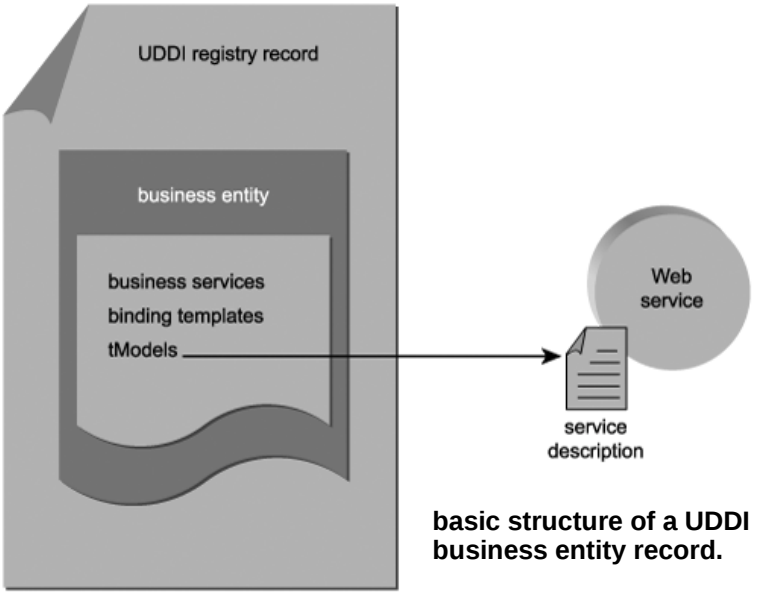
A service description is a set of conditions that must be met and accepted by a potential service requestor to enable successful communication. A service contract can refer to additional documents or agreements not expressed by service descriptions. For example, a Service Level Agreement agreed upon by the respective owners of a service provider and its requestor can be considered part of an overall service contract.





Service description locations centralized in a registry



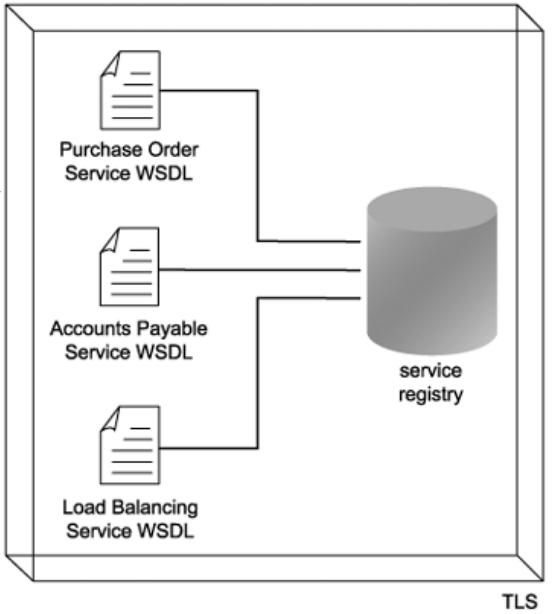


basic structure of a UDDI business entity record.

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 (Figure 5.20). 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).



Messaging (with SOAP)

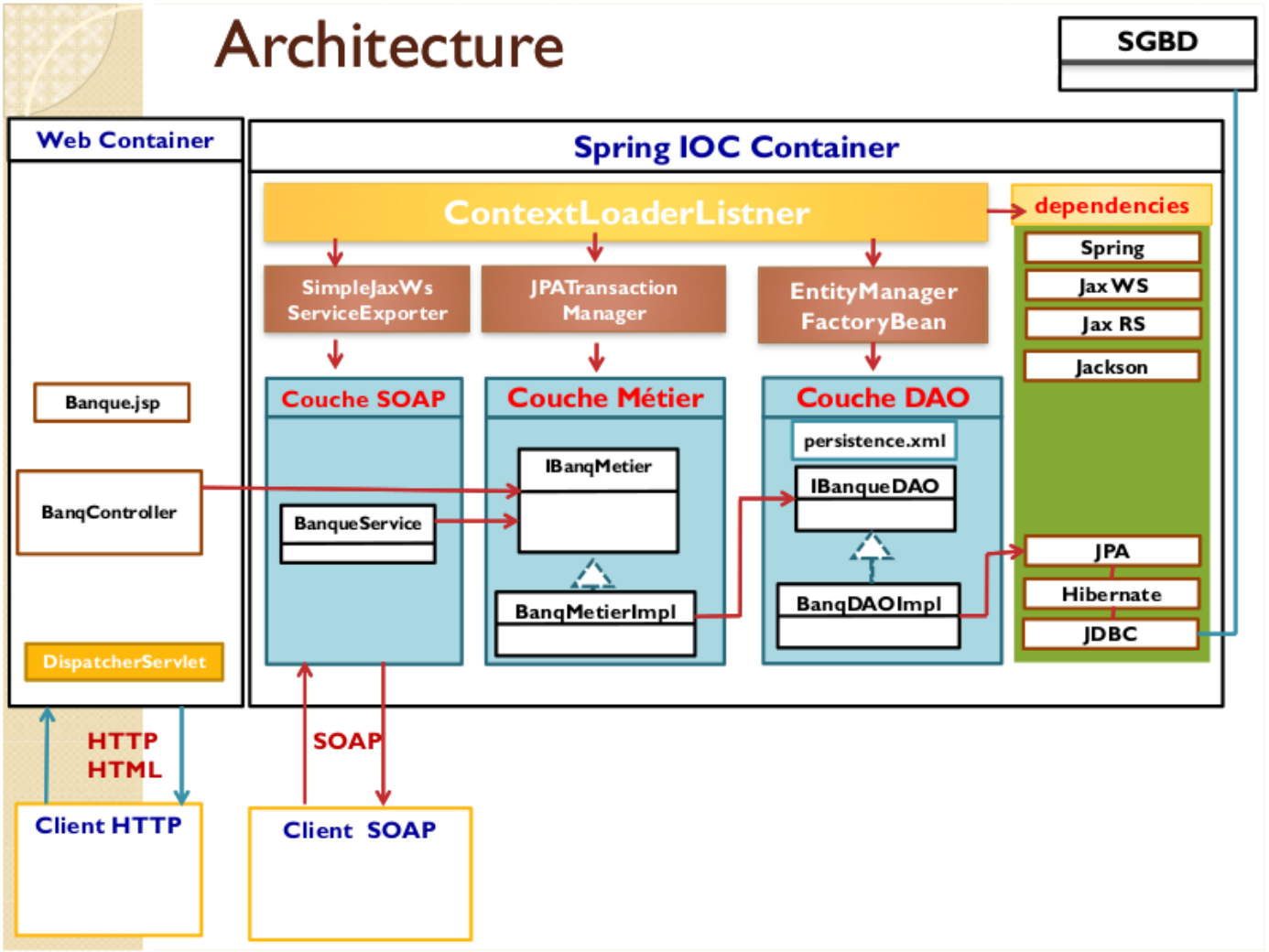
Web Service in a Spring based J2EE web application

We want to create a J2EE web application based on Spring which allows to deploy a web service which allows to:

- Add an account
- Consult an account
- View all accounts
- Make a deposit of an amount in an account
- Withdraw an amount from an account
- Transfer an amount from one account to another
- Delete an account
- Each account is defined by the code, the balance and the creation date
- Accounts will be stored in a MySQL database

Arch.

- Architecture

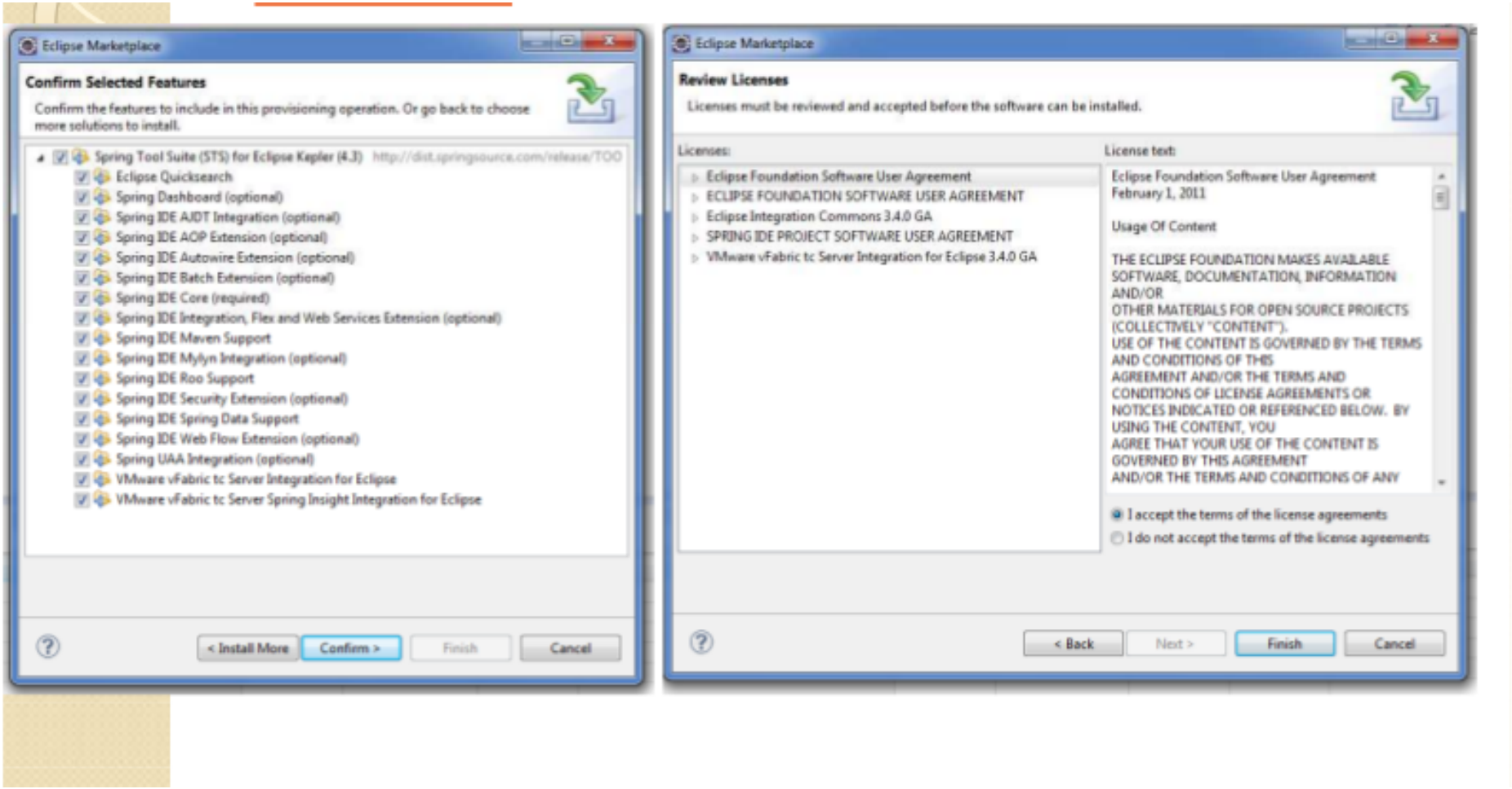


Prerequisities

- Netbeans 8.2 IDE
- Java JDK 7/8
- Glassfish Server
- Once you have the above items installed, proceed to the next step.

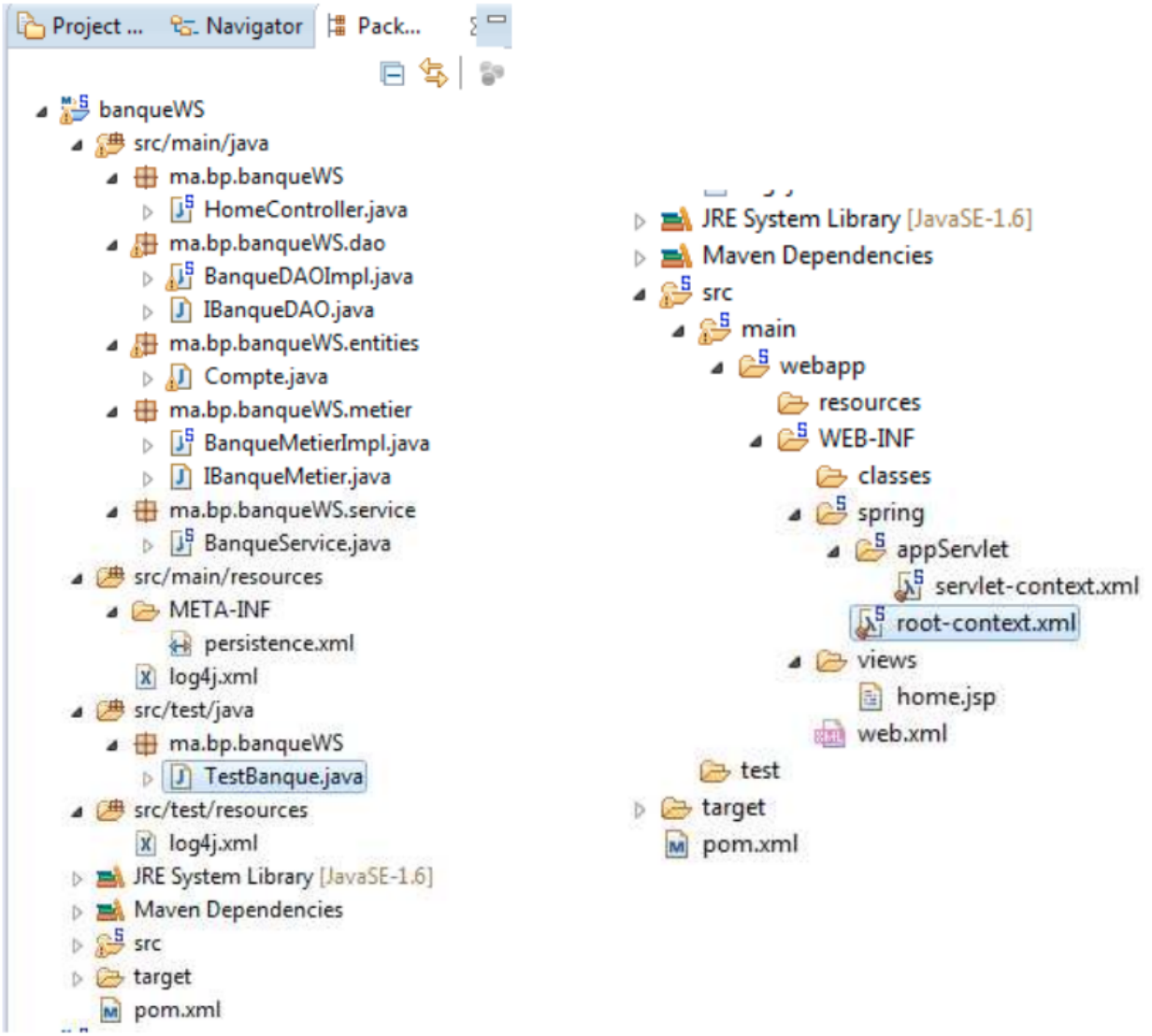
Installing the plugin: springtools for eclipse

- Steps



Structure project

- Steps



Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/applicationContext.xml</param-value>
    </context-param>
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <servlet>
        <servlet-name>dispatcher</servlet-name>
        <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
        <load-on-startup>2</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>dispatcher</servlet-name>
        <url-pattern>*.htm</url-pattern>
    </servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>redirect.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

root-context.xml

- /WEB/WEB-INF/spring/root- context.xml
- This file is read by ContextLoaderListener, when the server is started. This is a file in which application context will be built
- ContextLoaderListener represents Spring IOC it is therefore a file for the injection of dependencies At the moment it is empty

Web services in an application J2EE web based on EJB3 and wilfly

- In a server-based J2EE web application J2EE application
- WebSphere
- jboss
- Jonah
- Glassfish
- No configuration is needed to deploy the web service. Just create a web project based on the server of application
- Create the web service
- Start the server and deploy the web project
- Each application server has its own JAX WS implementation: CXF, AXIS, etc...

Web services in an application J2EE web based on EJB3 and wilfly

- structure