UNIT-II

Roots of SOA – Characteristics of SOA - Comparing SOA to client-server and distributed internet architectures – Anatomy of SOA- How components in an SOA interrelate -Principles of service orientation

Roots of SOA (comparing SOA to past architectures)

What is architecture?

- IT departments started to recognize the need for a standardized definition of a baseline application that could act as a template for all others.
- This definition was abstract in nature, but specifically explained the technology, boundaries, rules, limitations, and design characteristics that apply to all solutions based on this template.
- This was the birth of the application architecture.

Types of Architecture:

- Application architecture
- Enterprise architecture
- Service-oriented architecture

Application Architecture

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

Enterprise Architecture

- In larger IT environments, the need to control and direct IT infrastructure is critical.
- When numerous, disparate application architectures co-exist and sometimes even integrate, the demands on the underlying hosting platforms can be complex and onerous.
- Therefore, it is common for a master specification to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise, as well as a definition of the supporting infrastructure.
- Continuing our previous analogy, an enterprise architecture specification is to an organization what an urban plan is to a city. Therefore, the relationship between an urban plan and the blueprint of a building are comparable to that of enterprise and application architecture specifications.
- Typically, changes to enterprise architectures directly affect application architectures, which is why architecture specifications often are maintained by the same group of individuals.

Further, enterprise architectures often contain a long-term vision of how the organization plans to evolve its technology and environments. For example, the goal of phasing out an outdated technology platform may be established in this specification.

Service-Oriented Architecture

- Service-oriented architecture spans both enterprise and application architecture domains.
- The benefit potential offered by SOA can only be truly realized when applied across multiple solution environments.
- This is where the investment in building reusable and interoperable services based on a vendor-neutral communications platform can fully be leveraged. This does not mean that the entire enterprise must become service-oriented.
- SOA belongs in those areas that have the most to gain from the features and characteristics it introduces.
- Note that the term "SOA" does not necessarily imply a particular architectural scope.
- An SOA can refer to 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.

Common Characteristics of 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.

Specifically, we explore the following primary characteristics:

1. Contemporary SOA is at the core of the service-oriented computing platform.
2. Contemporary SOA increases quality of service.
3. Contemporary SOA is fundamentally autonomous.
4. Contemporary SOA is based on open standards.
5. Contemporary SOA supports vendor diversity.
6. Contemporary SOA fosters intrinsic interoperability.
7. Contemporary SOA promotes discovery.
8. Contemporary SOA promotes federation.
9. Contemporary SOA promotes architectural composability.
10. Contemporary SOA fosters inherent reusability.
11. Contemporary SOA emphasizes extensibility.
12. Contemporary SOA supports a service-oriented business modeling paradigm.
13. Contemporary SOA implements layers of abstraction.
14. Contemporary SOA promotes loose coupling throughout the enterprise.
15. Contemporary SOA promotes organizational agility.
16. Contemporary SOA is a building block.
17. Contemporary SOA is an evolution.
18. Contemporary SOA is still maturing.
19. Contemporary SOA is an achievable ideal.

1. Contemporary SOA is at the core of the service-oriented computing platform An application computing platform consisting of Web services technology and service-orientation principles.

*Contemporary SOA represents an architecture that promotes service-orientation through the use of Web service*s.

2. Contemporary SOA increases quality of service

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.

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 we discuss in *Part*

3. 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.

4. 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 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.

5. 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.

6. 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.

7. 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.

8. 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).

9. 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.

10. 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.

11. 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.

12. Contemporary SOA supports a service-oriented business modeling paradigm

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 SOA

13. Contemporary SOA implements layers of abstraction

One of the characteristics that tend 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.

14. Contemporary SOA promotes loose coupling throughout the enterprise

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

15. Contemporary SOA promotes organizational agility

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.

16. Contemporary SOA is a building block

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.

17. 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.

18. 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

20. 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.

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 service*s.

*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 model*s.

*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 enterpris*e.

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

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.

Separating Concrete Characteristics

Looking back at the list of characteristics we just covered, we can actually split them into two group's 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.

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

o Contemporary SOA is at the core of the service-oriented computing platform. o Contemporary SOA is a building block.

o Contemporary SOA is an evolution.

o Contemporary SOA is still maturing.

o Contemporary SOA is an achievable ideal.

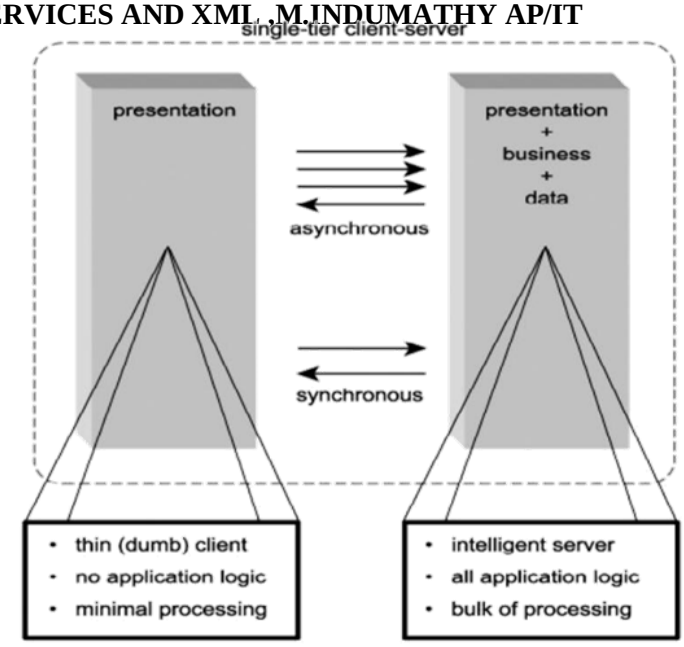
SOA vs. Client-Server Architecture

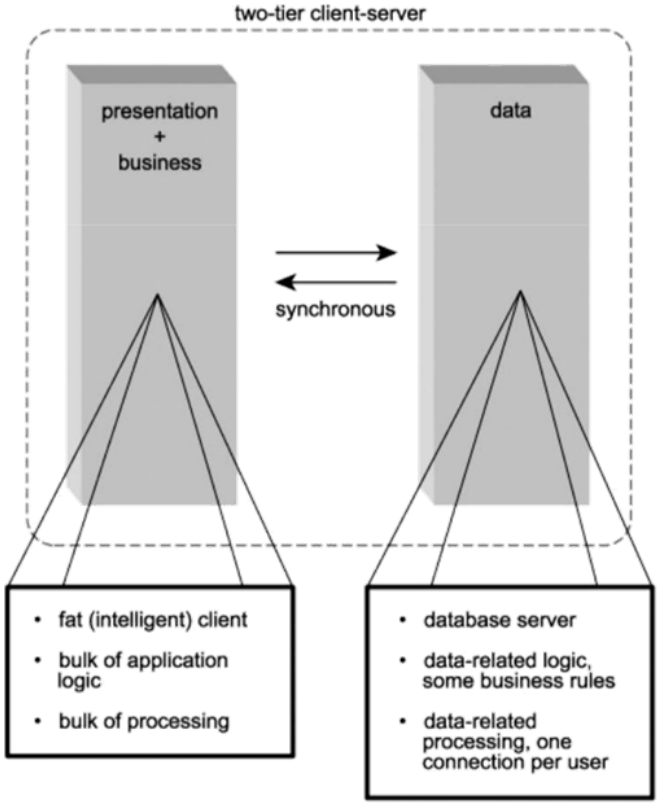
The original monolithic mainframe systems that empowered organizations to get seriously computerized often are considered the first inception of client-server architecture. These environments, in which bulky mainframe back-ends served thin clients, are considered an implementation of the single-tier client-server architecture Mainframe systems natively supported both synchronous and asynchronous communication. The latter approach was used primarily to allow the server to continuously receive characters from the terminal in response to individual key-strokes. Only upon certain conditions would the server actually respond.

While its legacy still remains, the reign of the mainframe as the foremost computing platform began to decline when a two-tier variation of the client-server design merged in the late 80s.

The common configuration of this architecture consisted of multiple fat clients, each with its own connection to a database on a central server. Client-side software performed the bulk of the processing, including all presentation-related and most data access logic. One or more servers facilitated these clients by hosting scalable RDBMSs.

Let's look at the primary characteristics of the two-tier client-server architecture individually and compare them to the corresponding parts of SOA.


single-tier client-server
presentation
presentation + business + data
asynchronous
synchronous
• thin (dumb) client
• no application logic
• minimal processing
• intelligent server
• all application logic
• bulk of processing


two-tier client-server
presentation + business
data
synchronous
• fat (intelligent) client
• bulk of application logic
• bulk of processing
• database server
• data-related logic, some business rules
• data-related processing, one connection per user

A two-tier client-server solution with a large user-base generally requires that each client establish its own database connection. Communication is predictably synchronous, and these connections are often persistent (meaning that they are generated upon user login and kept active until the user exits the application). Proprietary database connections are expensive, and the resource demands sometimes overwhelm database servers, imposing processing latency on all users.

Technology

The emergence of client-server applications promoted the use of 4GL programming languages, such as Visual Basic and PowerBuilder.

The technology set used by SOA actually has not changed as much as it has expanded. Newer versions of older programming languages, such as Visual Basic, still can be used to create Web services, and the use of relational databases still is commonplace.

The technology landscape of SOA, though, has become increasingly diverse. In addition to the standard set of Web technologies (HTML, CSS, HTTP, etc.) contemporary SOA brings with it the absolute requirement that XML data representation architecture be established, along with a SOAP messaging framework, and a service architecture comprised of the ever-expanding Web services platform.

Security

Besides the storage and management of data and the business rules embedded in stored procedures and triggers, the one other part of client-server architecture that frequently is centralized at the server level is security. Databases are sufficiently sophisticated to manage user accounts and groups and to assign these to individual parts of the physical data model.

Security also can be controlled within the client executable, especially when it relates to specific business rules that dictate the execution of application logic (such as limiting access to a part of a user-interface to select users). Additionally, operating system-level security can be incorporated to achieve a single sign-on, where application clearance is derived from the user's operating system login account information.

Administration

One of the main reasons the client-server era ended was the increasingly large maintenance costs associated with the distribution and maintenance of application logic across user workstations. Because each client housed the application code, each update to the application required a redistribution of the client software to all workstations. In larger environments, this resulted in a highly burdensome administration proces

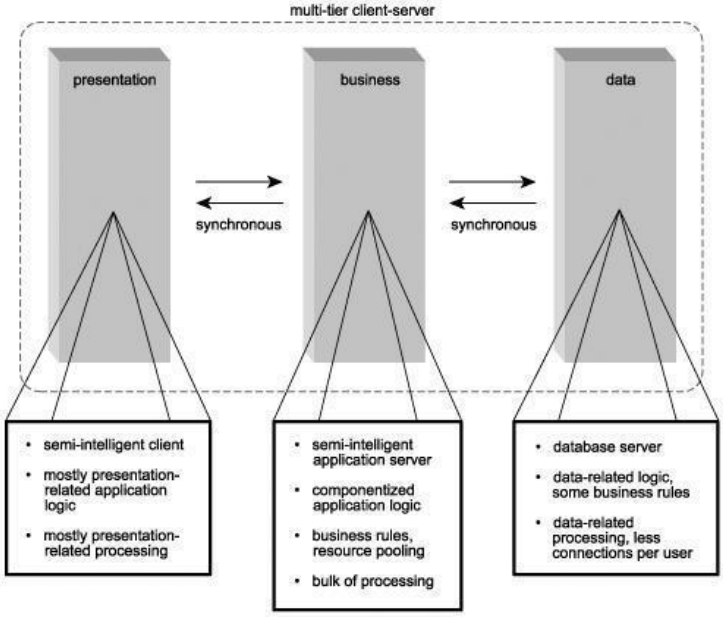
Maintenance issues spanned both client and server ends. Client workstations were subject to environment-specific problems because different workstations could have different software programs installed or may have been purchased from different hardware vendors. Further, there were increased server-side demands on databases, especially when a client-server application expanded to a larger user base.

Because service-oriented solutions can have a variety of requestors, they are not necessarily immune to client-side maintenance challenges. While their distributed back-end does accommodate scalability for application and database servers, new administration demands can be introduced.

SOA vs. Distributed Internet Architecture

In response to the costs and limitations associated with the two-tier client server architecture, the concept of building component-based applications hit the mainstream. Multi-tier client-server architectures surfaced, breaking up the monolithic client executable into components designed to varying extents of compliance with object-orientation.

Distributing application logic among multiple components (some residing on the client, others on the server) reduced deployment headaches by centralizing a greater amount of the logic on servers. Server-side components, now located on dedicated application servers, would then share and manage pools of database connections, alleviating the burden of concurrent usage on the database server.



- o Multiple client-server architectures have appeared
- o Client-server DB connections have been replaced with Remote Procedure Call connections (RPC) using CORBA or DCOM
- o Middleware application servers and transaction monitors require significant attention
- o Distributed Internet architecture introduced the Web server as a new physical tier and HTTP replaced RPC protocols

o Distributed Internet application put all the application logic on the server side o Even client-side scripts are downloaded from the server

o Entire solution is centralized o Emphasis is on:

o How application logic is partitioned o Where partitioned units reside

o How units of logic should interact

o Difference lies in the principles used to determine the three primary design considerations o Traditional systems create components that reside on one or more application servers

o Components have varying degrees of functional granularity

o Components on the same server communicate via proprietary APIs. o RPC protocols are used across servers via proxy stubs

o Actual references to other physical components can be embedded in programming code (tight coupling)

o SOAs also rely on components o Services encapsulate components

o Services expose specific sets of functionality

o Functionality can originate from legacy systems or other sources

o Functionality is wrapped in services

o Functionality is exposed via open, standardized interface, irrespective of technology providing the solution

o Services exchange information via SOAP messages. SOAP supports RPC-style and document-style messages

o Most applications rely on document-style o Messages are structured to be self-sufficient

o Messages contain Meta information, processing instructions, policy rules o SOA fosters reuse on a deep level by promoting solution-agnostic services

Comparing SOA to Client-Server and Distributed Internet Architectures Client-Server Application Technology

o The technology set for client-server applications included 4GLs like VB and PowerBuilder, RDBMSs

o The SOA technology set has expanded to include Web technologies (HTML, CSS, HTTP, etc)

o SOA requires the use of XML data representation architecture along with a SOAP messaging framewor

Client-Server Application Security

o Centralized at the Server level

o Databases manage user accounts and groups and also controlled within the client executable o Security for SOA is much more complex

o Security complexity is directly related to the degree of security measures required

o Multiple technologies are required, many in WS-Security framework

Client-Server Application Administration

o Significant maintenance costs associated with client-server o Each client housed application code

o Each update required redistribution

o Client stations were subject to environment-specific problems o Increased server-side demands on databases

o SOA solutions are not immune to client-side maintenance challenges

o Distributed back-end supports scalability, but new admin demands are introduced

o Management of server resources and service interfaces may require new admin tools and even a private registry

o Commitment to services and their maintenance may require cultural change in an organization

Logic Components of the Web Services Framework

Web services contain one or more operations.

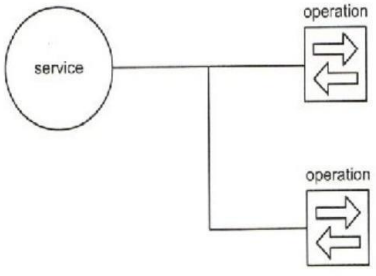


Figure 8.4
A Web service sporting two operations.

Each operation governs the process of a specific function the web service is capable of performing.

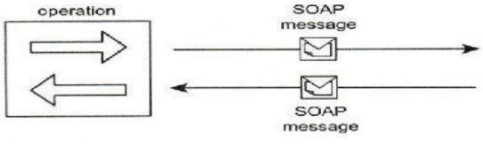


Figure 8.5
An operation processing outgoing and incoming SOAP messages.

Web services form an activity though which they can collectively automate a task.

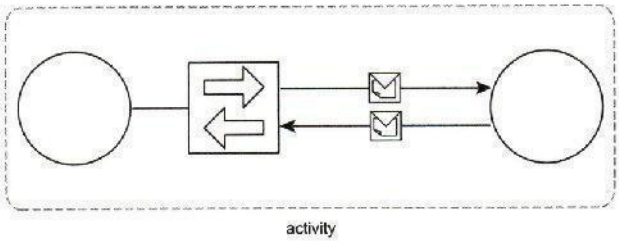


Figure 8.6
A basic communications scenario between Web services.

Logic Components of Automation Logic

- o Fundamental parts of the framework
  - o SOAP messages
  - o Web service operations o Web services
  - o Activities

The purpose of these views is to express the process, services and operations. o Is also provides a flexible means of partitioning and modularizing the logic. o These are the most basic concepts that underlie service-orientation.

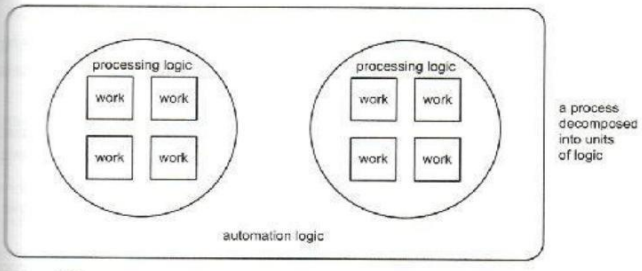


Figure 8.7
A primitive view of how SOA modularizes automation logic into units.

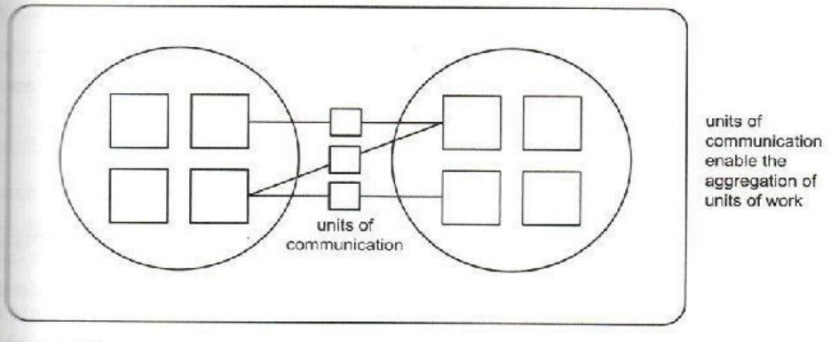


Figure 8.8
A primitive view of how units of communication enable interaction between units of logic.

Components of an SOA

Messages = units of communication

Operations = units of work

Services = units of processing logic

Processes = units of automation logic

Message

A message represents the data required to complete some or all parts of a unit of work.

Operation

An operation represents the logic required to process messages in order to complete a unit of work.

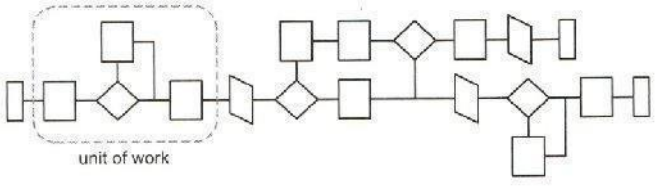


Figure 8.9
The scope of an operation within a process.

Service

A service represents a logically grouped set of operations capable of performing related units of work.

Processes

- A process contains the business rules that determine which service operations are used to complete a unit of automation.
- A process represents a large piece of work that requires the completion of smaller units of work.

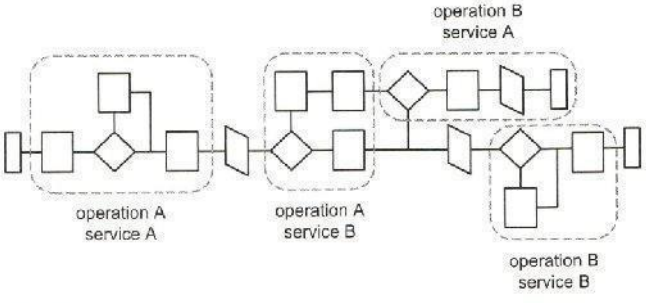


Figure 8.10
Operations belonging to different services representing various parts of process logic.

How Components in an SOA Inter-Relate

Having established the core characteristics of our SOA components, let's now look at how these components are required to relate to each other:

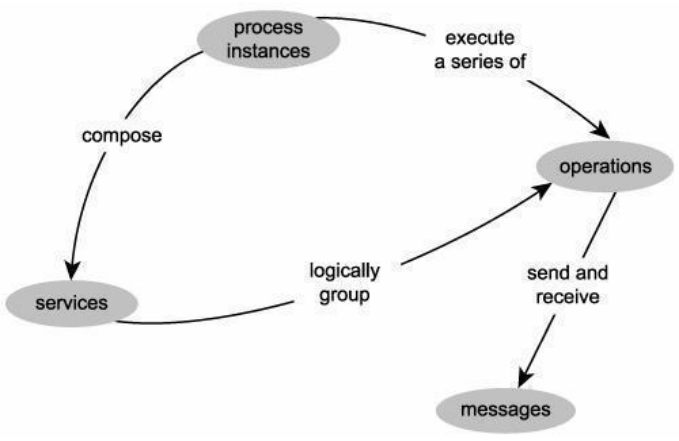


Figure. How the components of a service-oriented architecture relate.

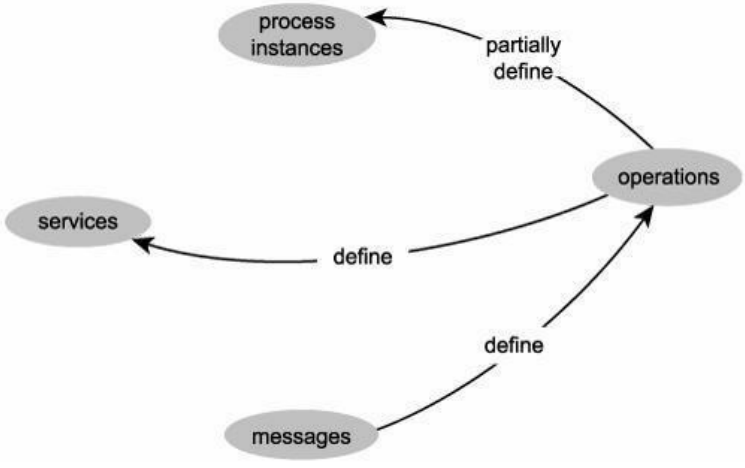


Figure. How the components of a service-oriented architecture define each other.

An operation sends and receives messages to perform work.

An operation is therefore mostly defined by the messages it processes. A service groups a collection of related operations.

A service is therefore mostly defined by the operations that comprise it.

A process instance can compose services.

A process instance is not necessarily defined by its services because it may only require a subset of the functionality offered by the services.

A process instance invokes a unique series of operations to complete its automation.

Every process instance is therefore partially defined by the service operations it uses.

Common Principles of Service- Orientation

o Services are reusable

o Services share a formal contract o Services are loosely coupled

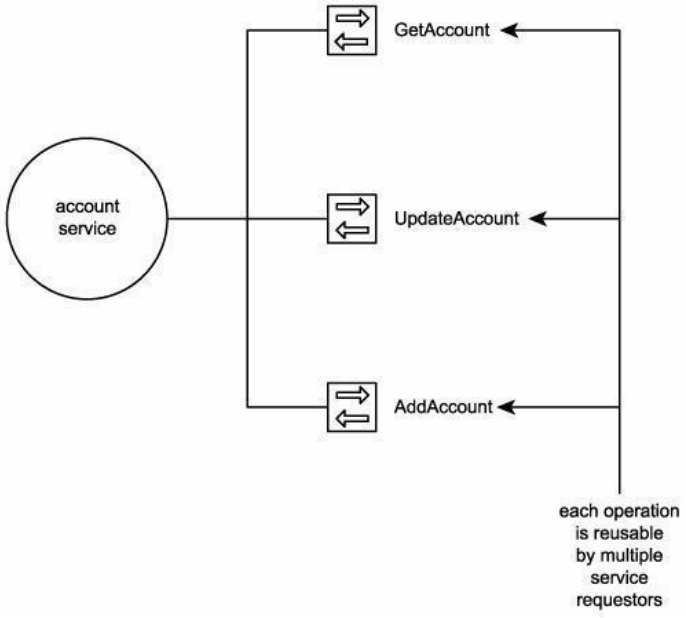
o Services abstract underlying logic o Services are composable

o Services are autonomous o Services are stateless

o Services are discoverable

Services Are Reusable

- o Regardless of weather immediate reuse opportunities exist, services are designed to support potential reuse.
- o Service-oriented encourages reuse in all services.
- o By applying design standards that require reuse accommodate future requirements with less development effort.



Case Study

RailCo created the Invoice Submission Service which contains two operations o SubmitInvoice o GetTLSMetadata

SubmitInvoice - Allows RailCo send electronic invoices to TLS Account Payable Service

GetTLSMetadata – checks periodically for changes to TLS Account Payable Service

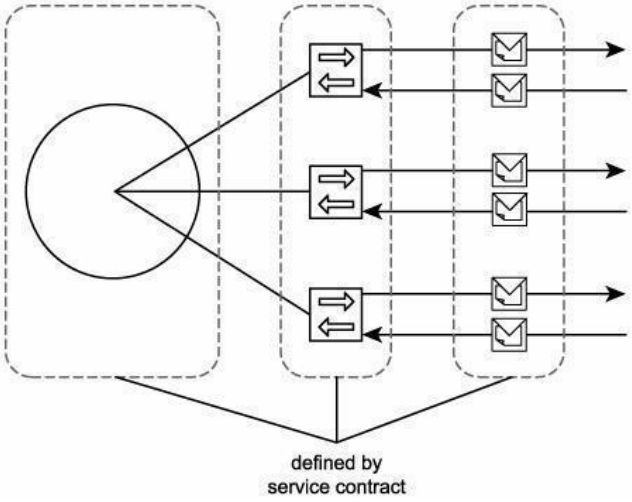
Services Share a Formal Contract

For services to interact, they need not share anything but formal contract that describe each service and define the terms of information exchange.

Service contracts provide a formal definition of:

- o The service endpoint o Each service operation
- o Every input and output message supported by each operation
- o Rules and characteristics of the service and its operations Service contacts

define almost all of the primary parts of an SOA.

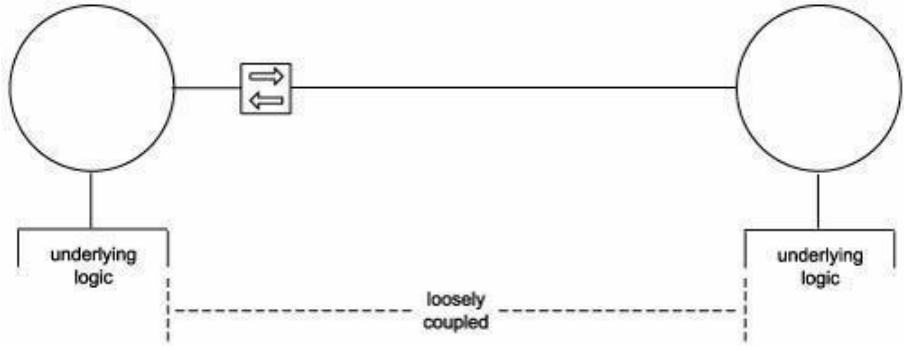


Case Study

RailCo and TLS agreed to a service contract that allow the two companies to communicate TLS defined the definition of the associated service description documents TLS ensures a standardized level of conformance that applies to each of its online vendors TLS can change the service at any time

Services Are Loosely Coupled

Services must be designed to interact without the need for tight, cross-service dependencies.

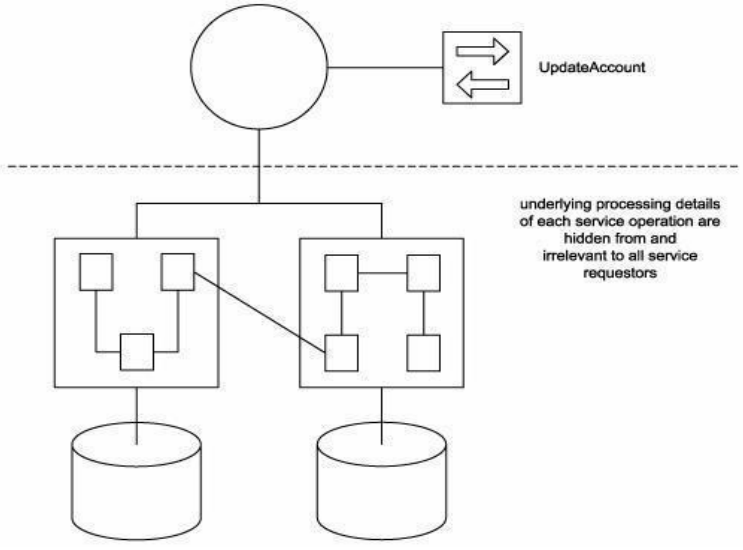


Case Stud

TLS services are designed to communicate with multiple online vendors make it loosely coupled RailCo service are designed to communicate only with TLS B2B system so is considered less loosely coupled

Services Abstract Underlying Logic

The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service requestors.



Case Study

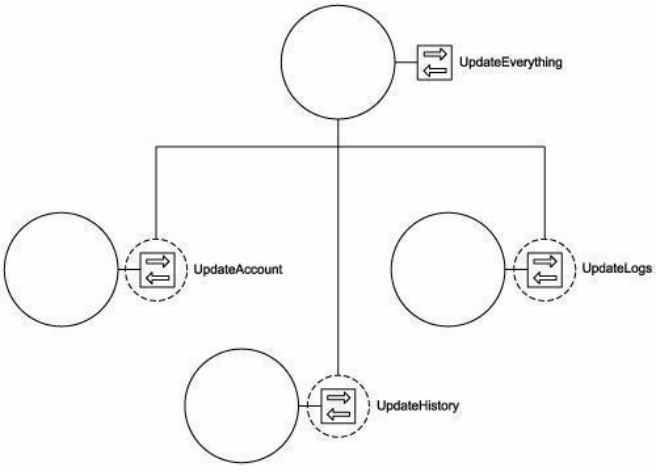
RailCo Web services hide the underlying legacy code need to produce the invoices that are needed to sent to TLS

TLS Web services hide the underlying services that process the invoices from multiple online vendors

Neither service requestor require any knowledge of what processes are working on the other's service providers

Services Are Composable

Services may be composing other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.



Case Study

o TLS accounts Payable Service is composed on three services o Accounts

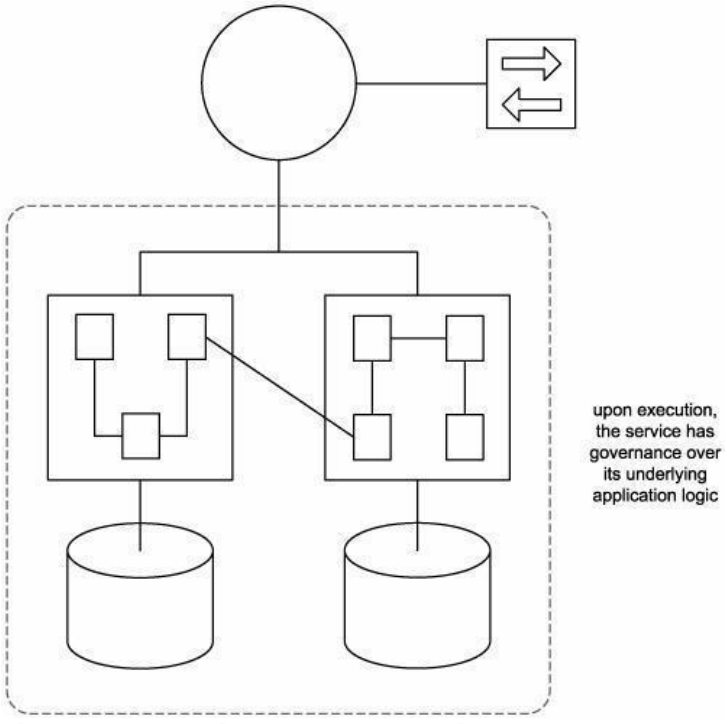
Payable Service

- Vendor Profile Service

o Ledger Service

Services Are Autonomous

The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.



Note that autonomy does not necessarily grant a service exclusive ownership of the logic it encapsulates. It only guarantees that at the time of execution, the service has control over whatever logic it represents. We therefore can make a distinction between two types of autonomy.

Service-level autonomy Service boundaries are distinct from each other, but the service may share underlying resources. For example, a wrapper service that encapsulates a legacy environment that also is used independently from the service has service-level autonomy. It governs the legacy system but also shares resources with other legacy clients.

Pure autonomy The underlying logic is under complete control and ownership of the service. This is typically the case when the underlying logic is built from the ground up in support of the service.

Case Study

Given the distinct tasks they perform, the following three RailCo services all are autonomous:

Invoice Submission Service Order Fulfillment

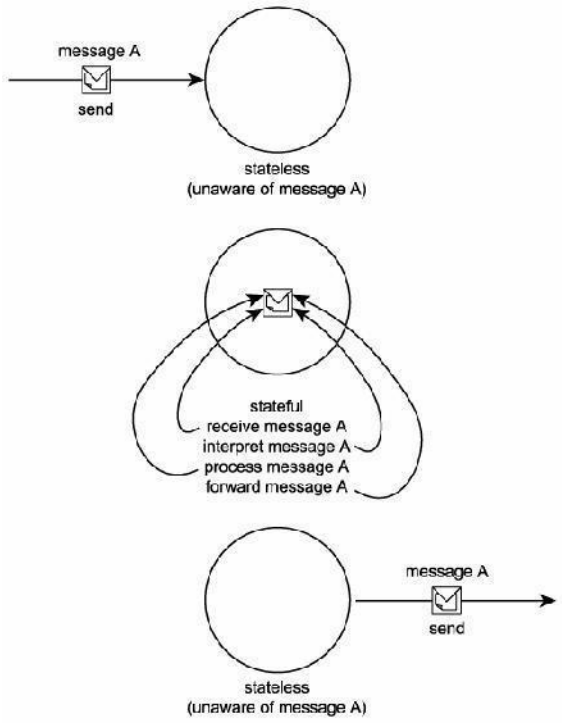
Service TLS Subscription Service

Each represents a specific boundary of application logic that does not overlap with the boundary of any other services.

Autonomy in RailCo's services was achieved inadvertently. No conscious effort was made to avoid application overlap, as the services were delivered to simply meet specific connectivity requirements.

Services Are Stateless

Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.



Statelessness is a preferred condition for services and one that promotes reusability and scalability. For a service to retain as little state as possible, its individual operations need to be designed with stateless processing considerations.

Case Study

As with loose coupling, statelessness is a quality that can be measured in degrees. The RailCo Order Fulfillment Service is required to perform extra runtime parsing and processing of various standard SOAP header blocks to successfully receive a purchase order document submitted by the TLS Purchase Order Service. This processing ties up the Order Fulfillment Service longer than, say, the

Invoice Submission Service, which simply forwards a predefined SOAP message to the TLS Accounting Service.

Services are discoverable

Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

Case Study

RailCo provides no means of discovery for its services, either internally or to the outside world. Though outfitted with its own WSDL definition and fully capable of acting as a service provider, the Invoice Submission Service is primarily utilized as a service requestor and currently expects no communication outside of the TLS Accounts Payable Service