Unit III

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

The roots of SOA (comparing SOA to past architectures)

- 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.

Type 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 the technical blueprint.
- Others include more detail, such as common data models, communication flow diagrams, application-wide security requirements, and aspects of infrastructure.
- It is not uncommon for an organization to have several application architectures. A single architecture document 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.

Enterprise architecture

- common for a master specification to be created, providing a high-level overview of all forms of heterogeneity that exist within an enterprise,
- enterprise architectures 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 For example, the goal of phasing out an outdated technology platform established in this specification.
- Finally, this document define the technology and policies behind enterprise-wide security measures. However, these often are isolated into a separate security architecture specification.

Service-oriented architecture

- service-oriented architecture spans both enterprise and application architecture domains.
- The benefit potential offered by SOA can be 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.

The following are the Characteristics of SOA:

Contemporary SOA is at the core of the service-oriented computing platform
SOA is used to qualify products, designs, and technologies
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 services.
Contemporary SOA increases quality of service

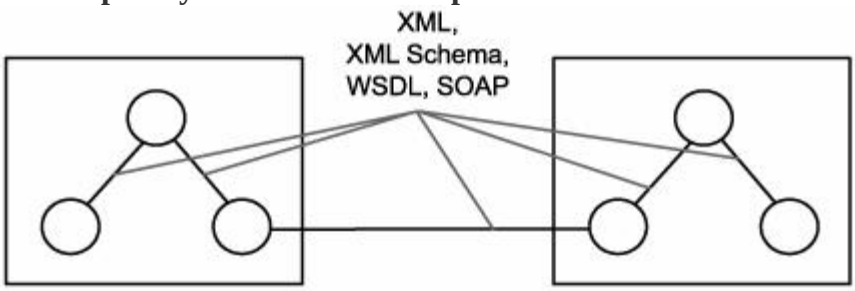
- The ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.
- Allowing tasks to be carried out reliably so that message delivery or notification of failed delivery can be guaranteed.

- 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 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.

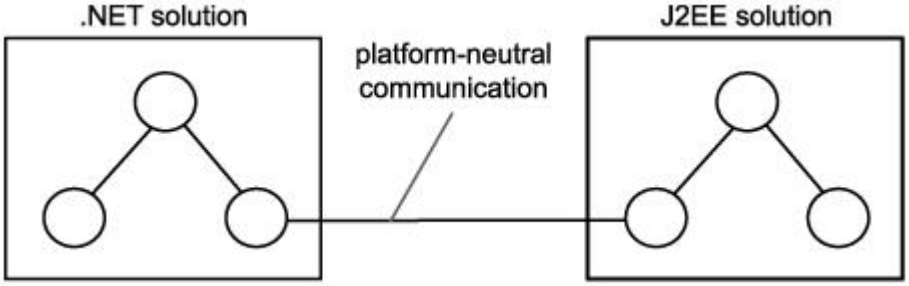
Contemporary SOA is based on open standards



Standard open technologies are used within and outside of solution boundaries.

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.

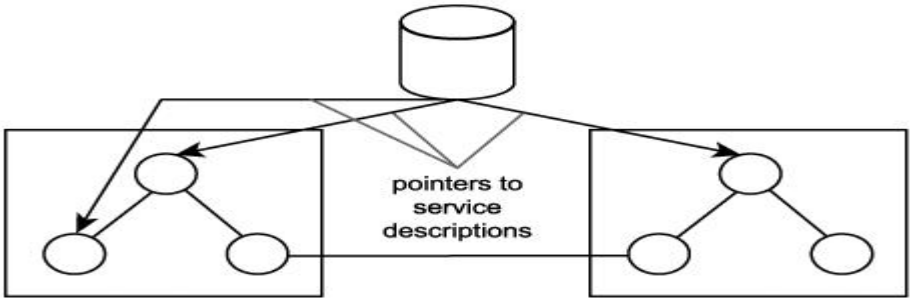
Contemporary SOA supports vendor diversity



Disparate technology platforms do not prevent service-oriented solutions from interoperating.

Organizations can certainly continue building solutions with existing development tools and server products. In fact, it may make sense to do so, only to continue leveraging the skill sets of in-house resources. However, the choice to explore the offerings of new vendors is always there. This option is made possible by the open technology provided by the Web services framework and is made more attainable through the standardization and principles introduced by SOA.

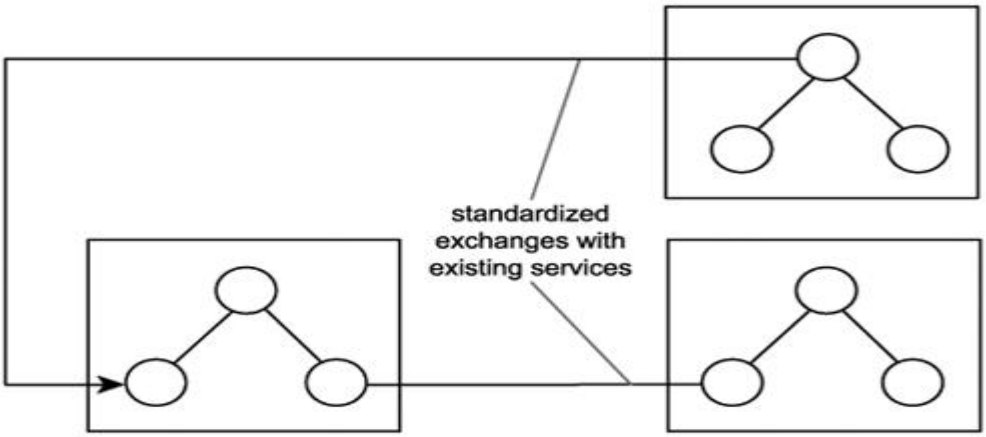
Contemporary SOA promotes discovery



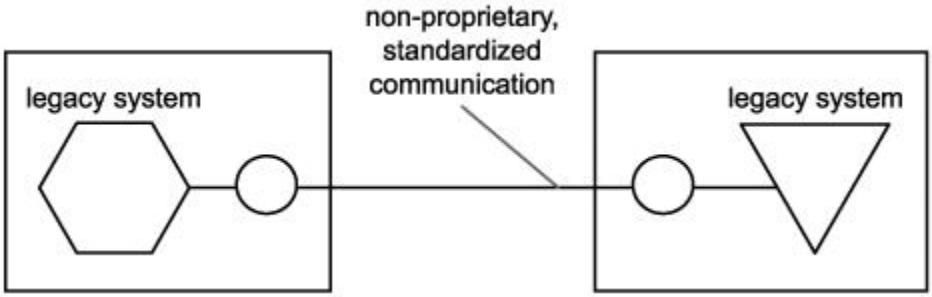
SOA supports and encourages the advertisement and discovery of services throughout the enterprise and beyond. A serious SOA will likely rely on some form of service registry or directory to manage service descriptions

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.



Contemporary SOA promotes federation



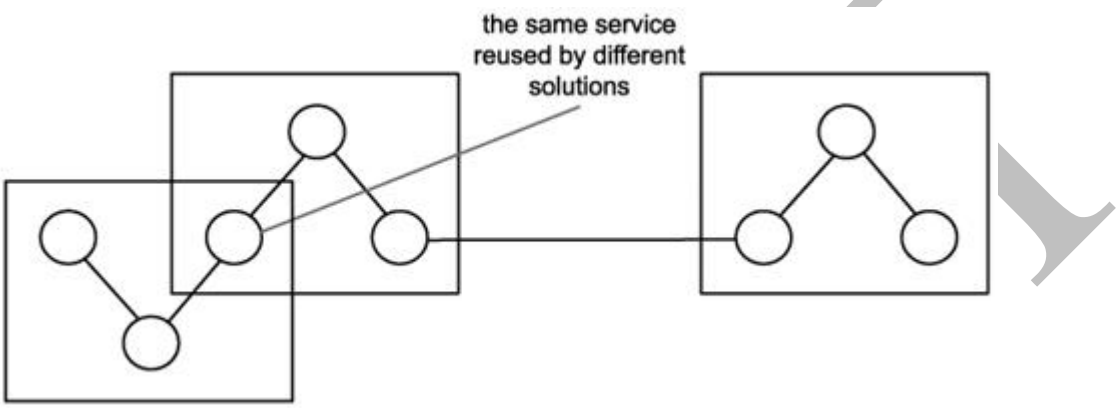
Establishing SOA within an enterprise does not necessarily require that you replace what you already have. One of the most attractive aspects of this architecture is its ability to introduce unity across previously non-federated environments. While Web services enable federation, SOA promotes this cause by establishing and standardizing the ability to encapsulate legacy and non-legacy application logic and by exposing it via a common, open,

and standardized communications framework (also supported by an extensive adapter technology marketplace).

Contemporary SOA promotes 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.

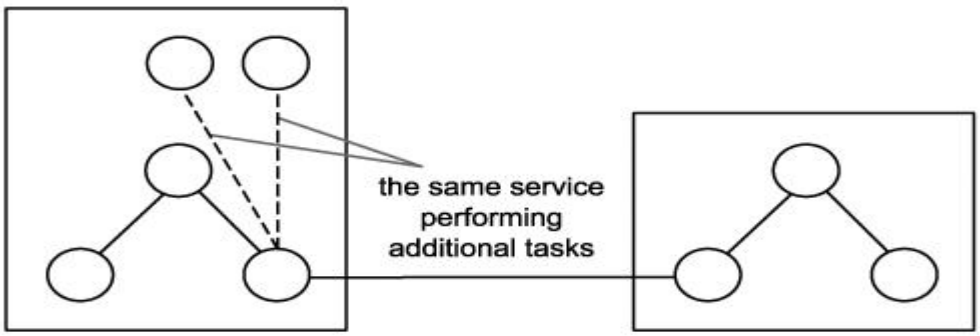
Contemporary SOA fosters inherent reusability



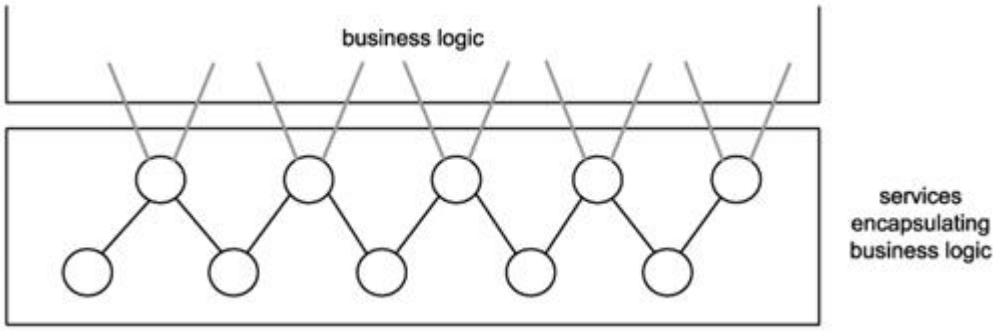
SOA establishes an environment that promotes reuse on many levels. For example, services designed according to service-orientation principles are encouraged to promote reuse, even if no immediate reuse requirements exist. Collections of services that form service compositions can themselves be reused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

Contemporary SOA emphasizes extensibility

Extensibility is also a characteristic that is promoted throughout SOA as a whole. Extending entire solutions can be accomplished by adding services or by merging with other service-oriented applications (which also, effectively, "adds services"). Because the loosely coupled relationship fostered among all services minimizes inter-service dependencies, extending logic can be achieved with significantly less impact.



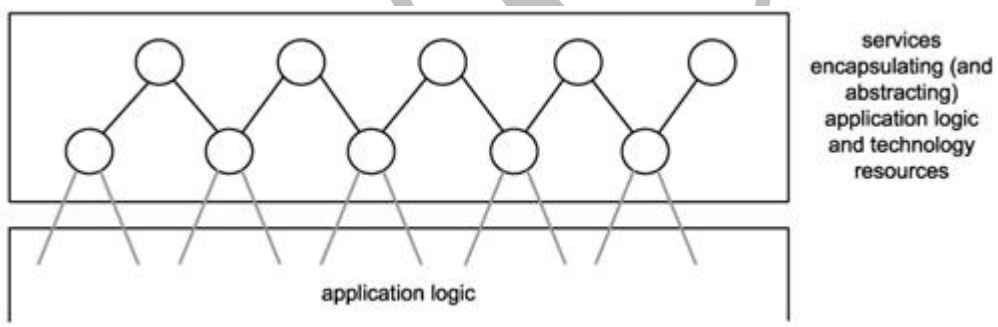
Contemporary SOA supports a service-oriented business modeling paradigm



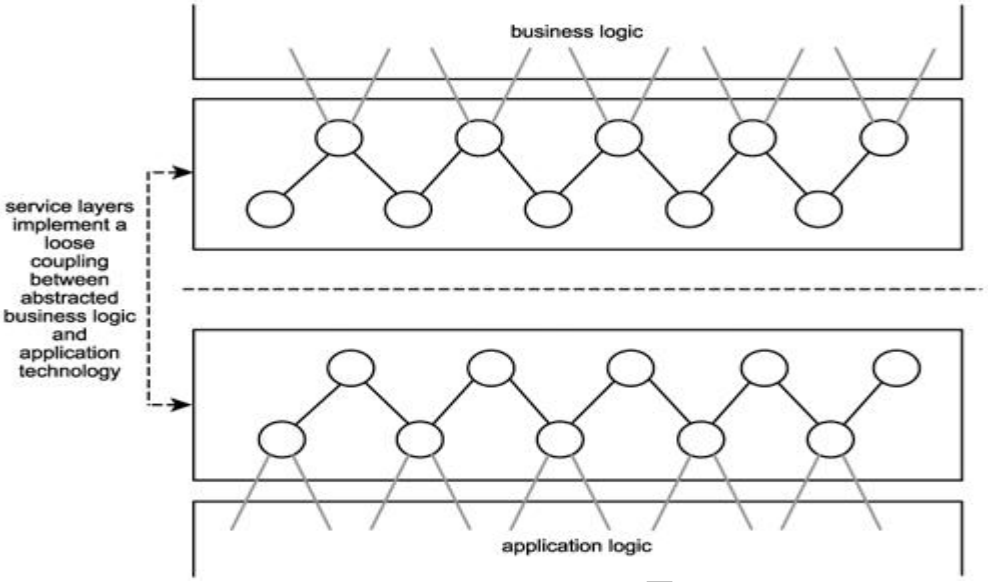
In other words, services can be designed to express business logic. BPM models, entity models, and other forms of business intelligence can be accurately represented through the coordinated composition of business-centric services. This is an area of SOA that is not yet widely accepted or understood.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

Contemporary SOA implements layers of abstraction

One of the characteristics that tends to evolve naturally through the application of service-oriented design principles is that of abstraction. Typical SOAs can introduce layers of abstraction by positioning services as the sole access points to a variety of resources and processing logic.

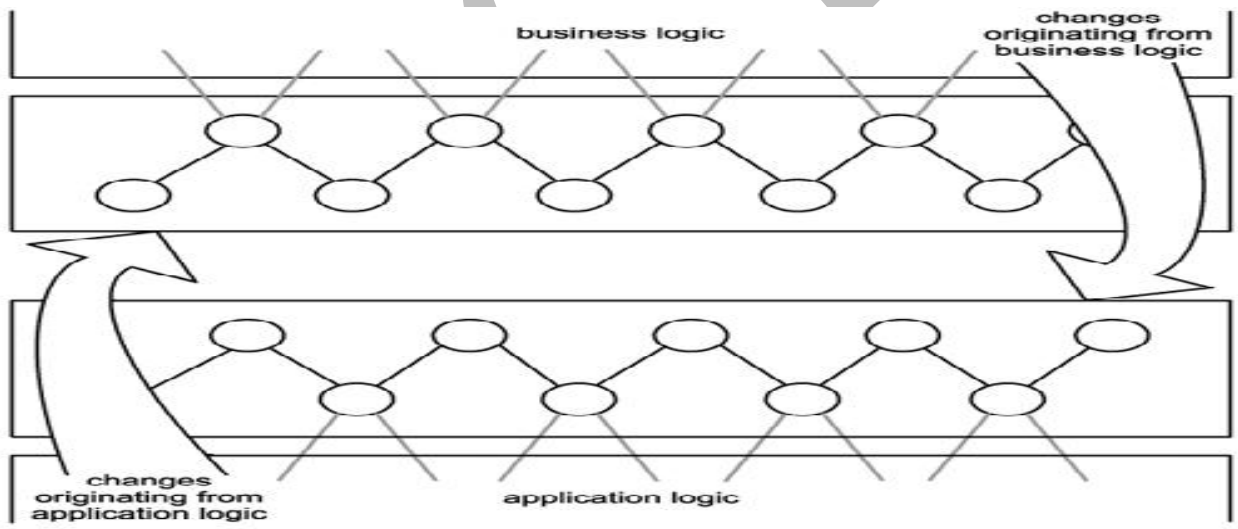


Contemporary SOA promotes loose coupling throughout the enterprise



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.

Contemporary SOA promotes organizational agility



Whether the result of an internal reorganization, a corporate merger, a change in an organization's business scope, or the replacement of an established technology platform, an organization's ability to accommodate change determines the efficiency with which it can respond to unplanned events.

Change in an organization's business logic can impact the application technology that automates it. Change in an organization's application technology infrastructure can impact the business logic automated by this technology. The more dependencies that exist between these two parts of an enterprise, the greater the extent to which change imposes disruption and expense.

Contemporary SOA is a building block

A service-oriented application architecture will likely be one of several within an organization committed to SOA as the standard architectural platform. Organizations standardizing on SOA work toward an ideal known as the service-oriented enterprise (SOE), where all business processes are composed of and exist as services, both logically and physically

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.

Contemporary SOA is still maturing

While the characteristics described so far are fundamental to contemporary SOA, this point is obviously more of a subjective statement of where SOA is at the moment. Even though SOA is being positioned as the next standard application computing platform, this transition is not yet complete. Despite the fact that Web services are being used to implement a great deal of application functionality, the support for a number of features necessary for enterprise-level computing is not yet fully available.

Contemporary SOA is an achievable ideal

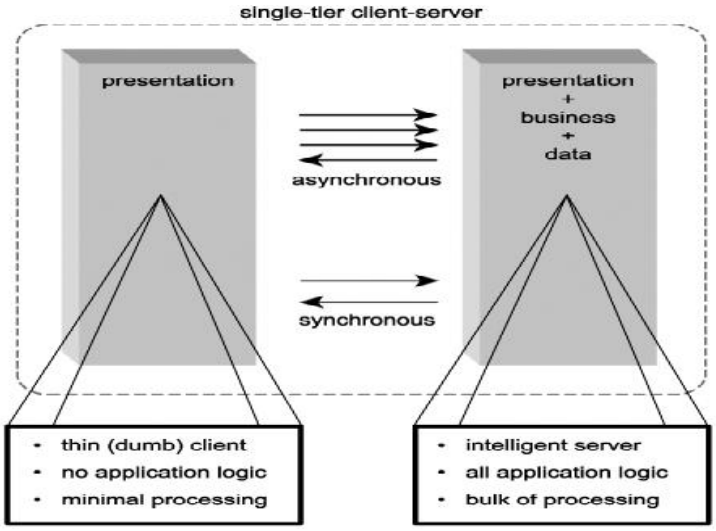
A standardized enterprise-wide adoption of SOA is a state to which many organizations would like to fast-forward. The reality is that the process of transitioning to this state demands an enormous amount of effort, discipline, and, depending on the size of the organization, a good amount of time. Every 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.

Lecture Notes:

SOA vs. client-server architecture

- Any environment in which one piece of software requests or receives information from another can be referred to as "client-server."
- Every variation of application architecture that ever existed (including SOA) has an element of client-server interaction in it
- The original monolithic mainframe systems that empowered organizations to get seriously computerized often are considered the first inception of client-server architecture.

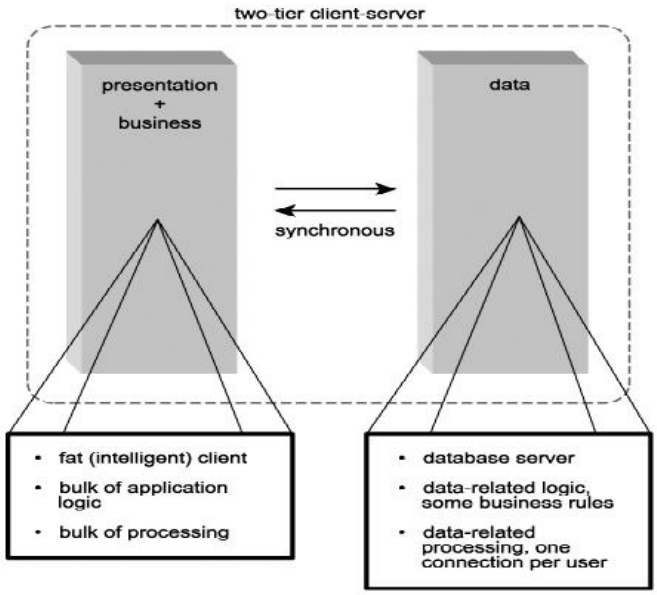
- These environments, in which bulky mainframe back-ends served thin clients, are considered an implementation of the single-tier client-server architecture



single tier client server architecture

Two-tier of the client-server

- The foremost computing platform began to decline when a two-tier variation of the client-server design
- This new approach introduced the concept of delegating logic and processing duties onto individual workstations, resulting in the birth of the fat client. Further supported by the innovation of the graphical user-interface (GUI), two-tier client-server was considered
- 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.



Two tier client server Architecture

Application logic

- Client-server environments place application logic into the client software. that controls the user experience, as well as the back-end resources.
- One exception is the distribution of business rules. A popular trend was to embed and maintain business rules relating to data within stored procedures and triggers on the database. This abstracted a set of business logic from the client and simplified data access programming.
- The presentation layer within service-oriented solutions can vary. Any piece of software capable of exchanging SOAP messages according to required service contracts can be classified as a service requestor.
- Within the server environment, options exist as to where application logic can reside and how it can be distributed. These options do not preclude the use of database triggers or stored procedures.
- However, service-oriented design dictating the partitioning of processing logic into autonomous units. This facilitates specific design qualities, such as service statelessness and interoperability, as well as future composability and reusability.

Application processing

- client-server application logic resides in the client component, the client workstation is responsible for the bulk of the processing. The 80/20 ratio often is used as a rule of thumb, with the database server typically performing twenty percent of the work.
- 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.
- Additionally, the clients are assigned the majority of processing responsibilities, they too often demand significant resources. Client-side executables are fully stateful and consume a steady chunk of PC memory. User workstations therefore often are required to run client programs exclusively so that all available resources can be offered to the application.
- Processing in SOA is highly distributed. Each service has an explicit functional boundary and related resource requirements. In modeling a technical service-oriented architecture, have many choices as to how can position and deploy services.
- Enterprise solutions consist of multiple servers, each hosting sets of Web services and supporting middleware. There is, therefore, no fixed processing ratio for SOAs. Services can be distributed as required,
- Communication between service and requestor can be synchronous or asynchronous. This flexibility allows processing to be further streamlined, This promotes the stateless and autonomous nature of services.

Technology

- **4GL programming languages,**
  - **Visual Basic and PowerBuilder.**
    - Windows operating system by providing the ability to create aesthetically rich and more interactive user-interfaces.
  - **On the back-end, major database vendors,**

- such as Oracle, Informix, IBM, Sybase, and Microsoft, provided robust RDBMSs that could manage multiple connections, while providing flexible data storage and data management features.

- **The technology set used by SOA**
  - Visual Basic, still can be used to create Web services, and the use of relational databases still is commonplace.
  - **Web technologies**
    - **(**HTML, CSS, HTTP, XML,J2EE,.NETetc.)
      - SOA brings with it the absolute requirement that an XML data representation architecture be established, along with a SOAP messaging framework, and a service architecture comprised of the ever-expanding Web services platform.

Security

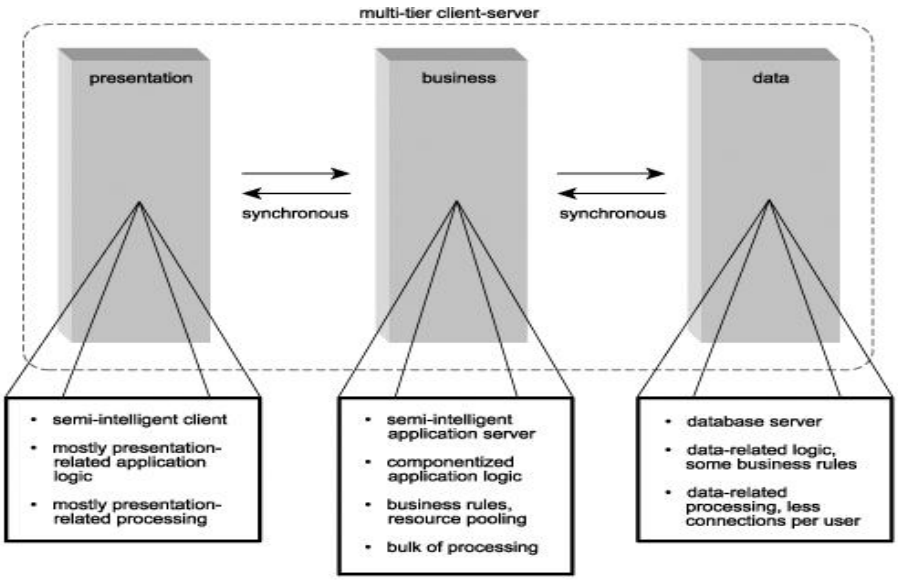
- The one part of client-server architecture that frequently is centralized at the server level is security. Databases are sufficiently sophisticated to manage user accounts and groups and to assign these to individual parts of the physical data model.
- Security can be controlled within the client executable, 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,
- The advantages of SOA, most architects envy the simplicity of client-server security. Corporate data is protected via a single point of authentication, establishing a single connection between client and server.
- In the distributed world of SOA, this is not possible. Security becomes a significant complexity directly relational to the degree of security measures required. Multiple technologies are typically involved, many of which comprise the WS-Security framework

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

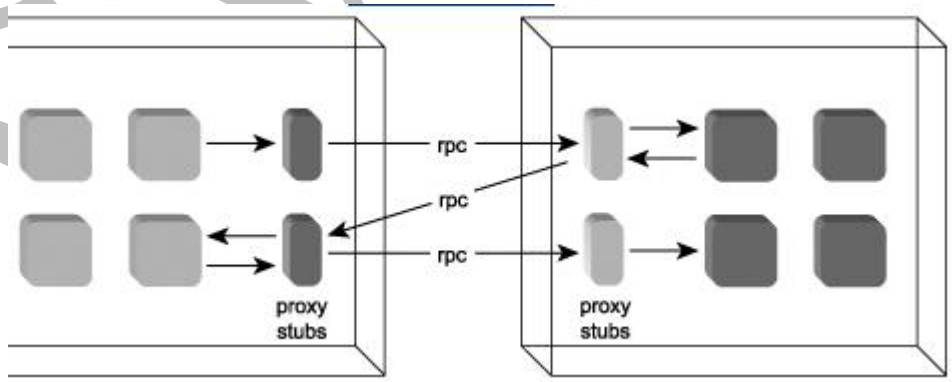
SOA vs. distributed Internet architecture

- Distributed architecture could be designed as SOAs.
- There are distributed environments in existence that may have been heavily influenced by service-oriented principles
- Multi-tier client-server architectures
  - Multi-tier client-server architectures breaking up the monolithic client executable into components designed.
  - 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, located on dedicated application servers, would then share and manage pools of database connections.A single connection could easily facilitate multiple users.



Multi tier client server Architecture

- Replacing client-server database connections was the client-server remote procedure call (RPC) connection.
- RPC technologies such as CORBA and DCOM allowed for remote communication between components residing on client workstations and servers.
- Issues similar to the client-server architecture problems involving resources and persistent connections emerged. Adding to this was an increased maintenance effort resulting from the introduction of the middleware layer. **For example**, application servers and transaction monitors required significant attention in larger environments.



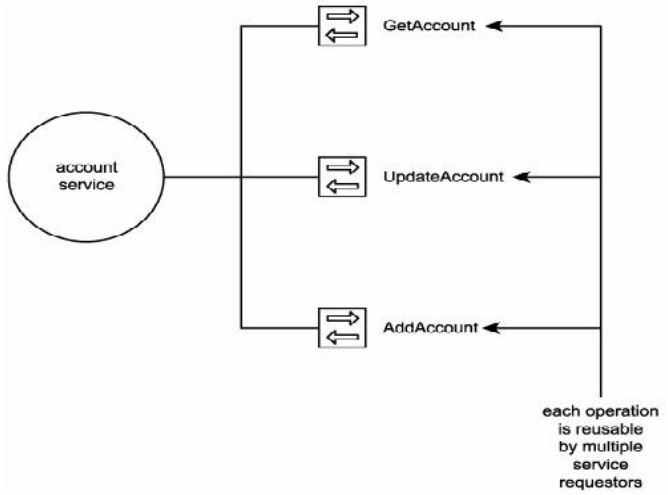
proxy stubs for remote communication

Common principles of service-orientation

- A common set of principles most associated with service-orientation.
  - **Services are reusable:** services are designed to support potential reuse.

- **Services share a formal contract** For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.
- **Services are loosely coupled Services** must be designed to interact without the need for tight, cross-service dependencies.
- **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.
- **Services are compos able** Services may compose other services. This promotes reusability and the creation of abstraction layers.
- **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.
- **Services are stateless** Services should not be required to manage state information.
- **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.

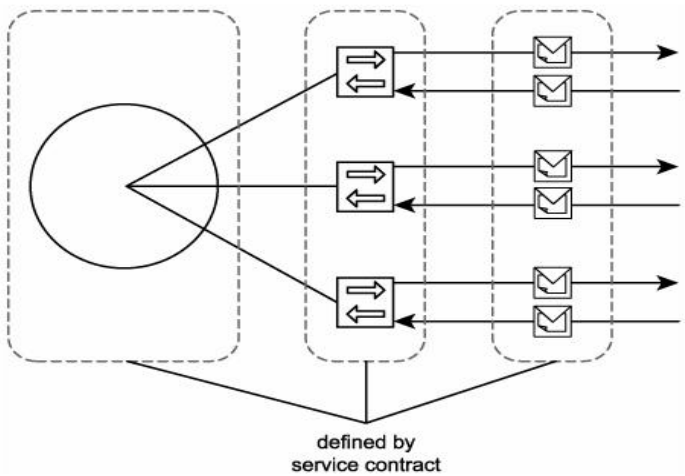
Services are reusable



Reusable service exposes reusable operations

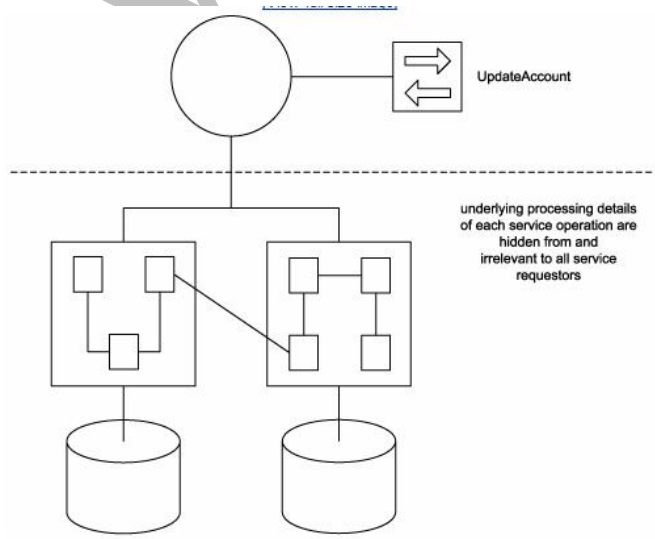
Services share a formal contract

- Service contracts provide a formal definition of:
  - the service endpoint
  - each service operation
  - every input and output message supported by each operation
  - rules and characteristics of the service and its operations



Service contract formally define service, operation, message component

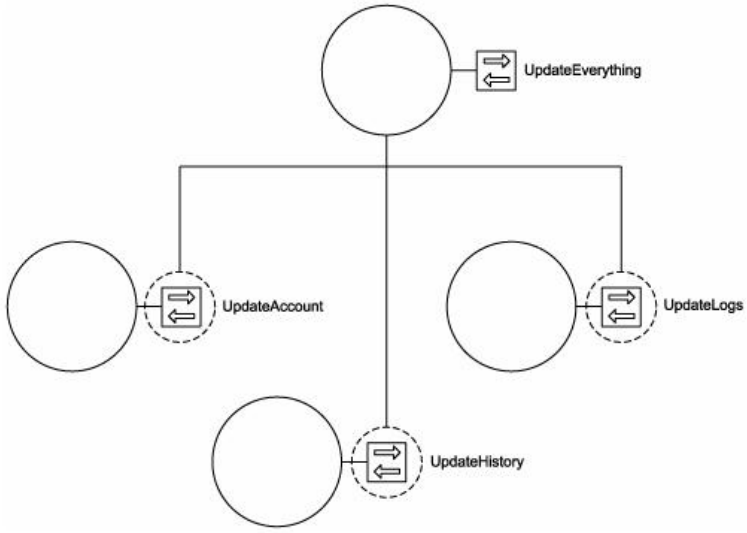
Services abstract underlying logic



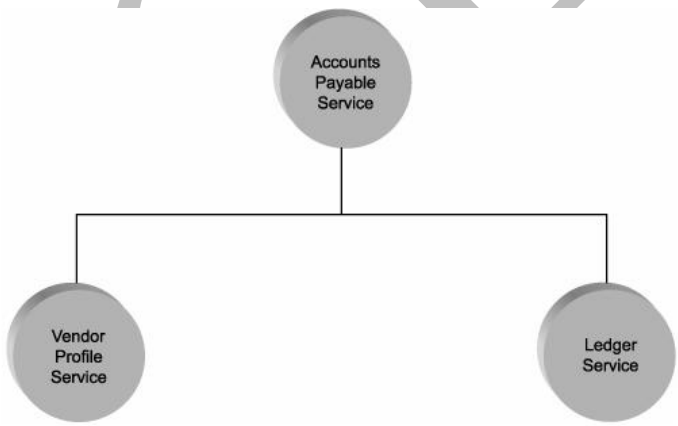
Service operations abstract the underlying details of the functionality they expose

Services are composable

- A service can represent any range of logic from any types of sources, including other services



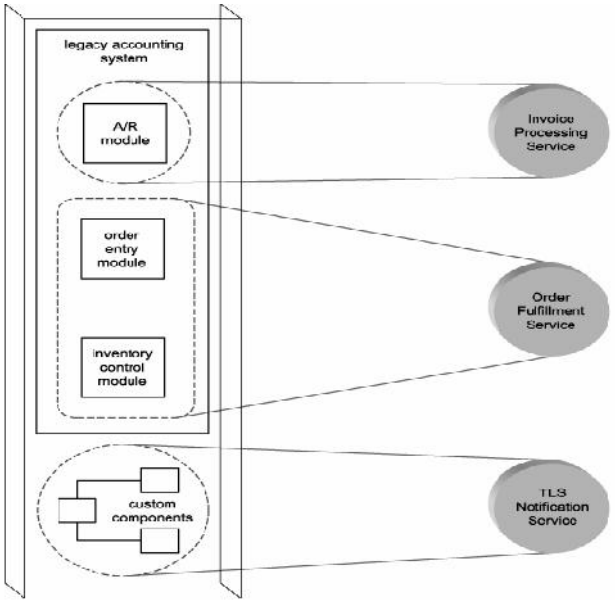
UpdateEverything operation encapsulating a service composition



TLS Accounts payable Service composition

Services are autonomous

- Autonomy requires that the range of logic exposed by a service exist within an explicit boundary. This allows the service to execute self-governance of all its processing.
- It also eliminates dependencies on other services

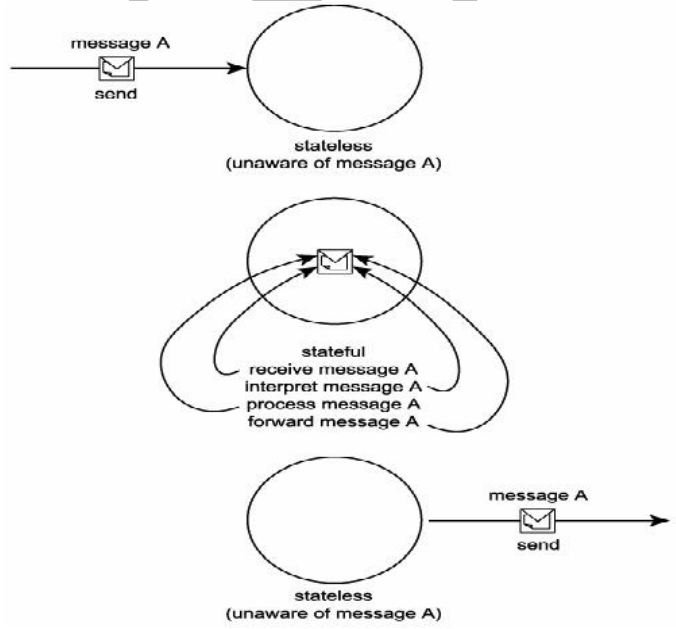


Service autonomous

Services are stateless

- Services should minimize the amount of state information they manage and the duration for which they hold it.

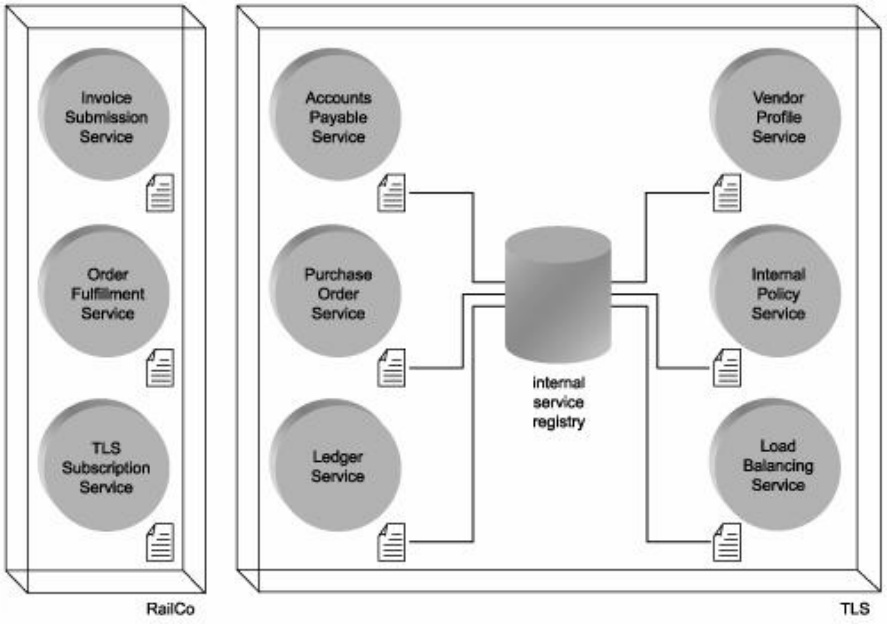
- State information is data-specific to a current activity



Stateless services

Services are discoverable

– Discovery helps avoid the implement redundant logic. Because each operation provides a potentially reusable piece of processing logic, metadata attached to a service needs to sufficiently.



RailCo's services are not discoverable, but TLS's inventory of services are stored in an internal registry

7 **Lecture Notes:**

Service layer abstraction

What logic should be represented by services?

Services can be modeled to represent either or both types of logic, as long as the principles of s orientation can be applied.

However, to achieve enterprise-wide loose coupling (the first of our four outstanding SOA characte physically separate layers of services are, in fact, required. When individual collections of s represent corporate business logic and technology-specific application logic, each domain of the en is freed of direct dependencies on the other.

This allows the automated representation of business process logic to evolve independently fr technology-level application logic responsible for its execution. In other words, this establishes a coupled relationship between business and application logic.

How should services relate to existing application logic?

Much of this depends on whether existing legacy application logic needs to be exposed via serv whether new logic is being developed in support of services. Existing systems can impose any nur constraints, limitations, and environmental requirements that need to be taken into consideration service design.

Applying a service layer on top of legacy application environments may even require that some s orientation principles be compromised. This is less likely when building solutions from the ground u service layers in mind, as this affords a level of control with which service-orientation can be incorporated into application logic.

Either way, services designed specifically to represent application logic should exist in a separat We'll therefore simply refer to this group of services as belonging to the *application service layer*.

How can services best represent business logic?

Business logic is defined within an organization's business models and business processes. When m services to represent business logic, it is most important to ensure that the service representation logic is in alignment with existing business models.

It is also useful to separately categorize services that are designed in this manner. Therefore, we'll services that have been modeled to represent business logic as belonging to the *business service la* adding a business service layer, we also implement the second of our four SOA characteristics, w

support for service-oriented business modeling.

How can services be built and positioned to promote agility?

The key to building an agile SOA is in minimizing the dependencies each service has within i processing logic. Services that contain business rules are required to enforce and act upon these r runtime. This limits the service's ability to be utilized outside of environments that require these Similarly, controller services that are embedded with the logic required to compose other servic develop dependencies on the composition structure.

Introducing a parent controller layer on top of more specialized service layers would allow us to esta centralized location for business rules and composition logic related to the sequence in which servi executed. Orchestration is designed specifically for this purpose. It introduces the concept of a p service, capable of composing other services to complete a business process according to pre workflow logic. Process services establish what we refer to as the *orchestration service layer*.

While the addition of an orchestration service layer significantly increases organizational agility (r three on our list of SOA characteristics), it is not alone in realizing this quality. All forms of org service abstraction contribute to establishing an agile enterprise, which means that the creation of s application, business, and orchestration layers collectively fulfill this characteristic.

Abstraction is the key

Though we addressed each of the preceding questions individually, the one common element to al answers also happens to be the last of our four outstanding SOA characteristics: layers of abstraction

We have established how, by leveraging the concept of composition, we can build specialized la services. Each layer can abstract a specific aspect of our solution, addressing one of the issu identified. This alleviates us from having to build services that accommodate business, applicati agility considerations all at once.

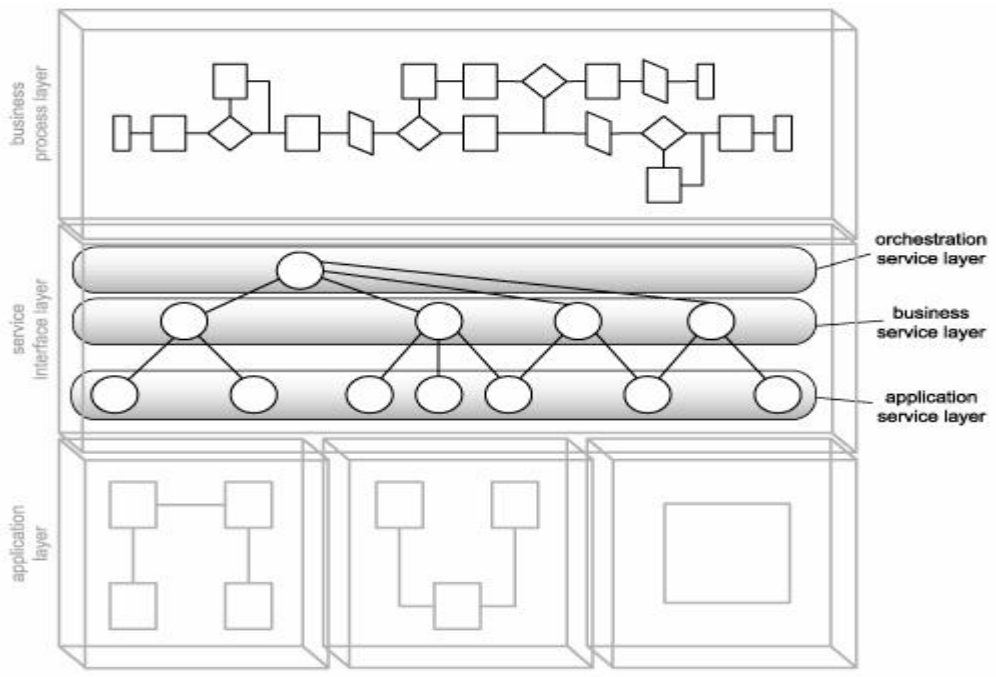
The three layers of abstraction we identified for SOA are:

- the application service layer

- the business service layer
- the orchestration service layer

Each of these layers is introduced individually in the following sections.

The three primary service layers.



Application service layer

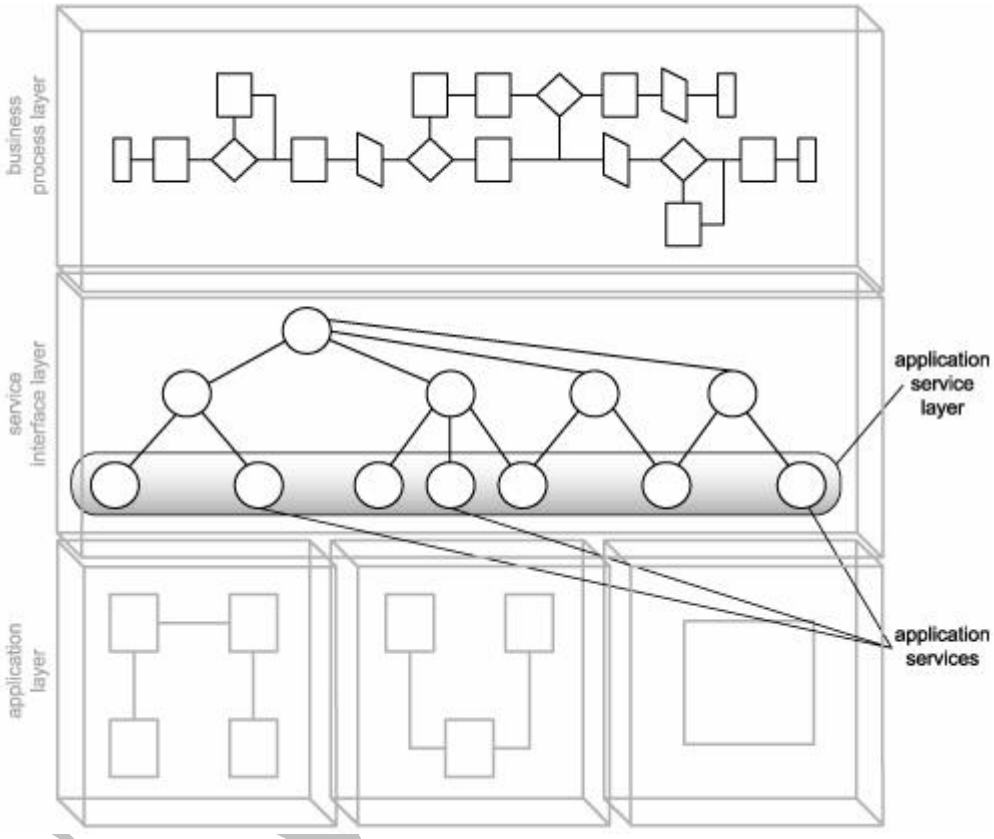
The application service layer establishes the ground level foundation that exists to express techn
specific functionality. Services that reside within this layer can be referred to simply as *application s*
. Their purpose is to provide reusable functions related to processing data within new or legacy app
environments.

Application services commonly have the following characteristics:

- they expose functionality within a specific processing context
- they draw upon available resources within a given platform
- they are solution-agnostic
- they are generic and reusable
- they can be used to achieve point-to-point integration with other application services
- they are often inconsistent in terms of the interface granularity they expose
- they may consist of a mixture of custom-developed services and third-party services that hav

purchased or leased

The application service layer.



Typical examples of service models implemented as application services include the following:

- utility service
- wrapper service

When a separate business service layer exists, there is a strong motivation to turn all application s into generic utility services. This way they are implemented in a solution-agnostic manner, pr reusable operations that can be composed by business services to fulfill business-centric pro requirements.

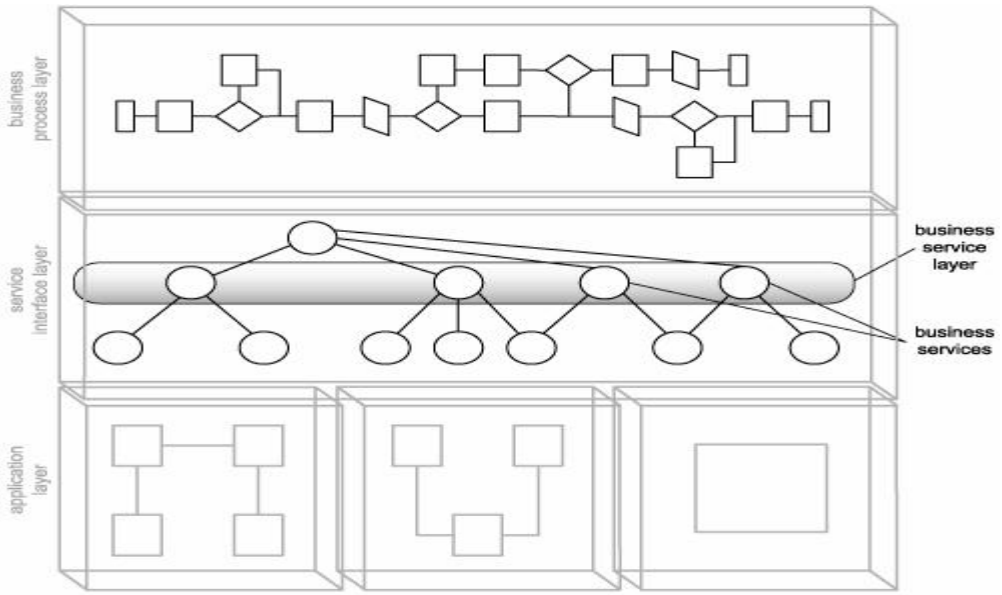
Alternatively, if business logic does not reside in a separate layer, application services may be requ implement service models more associated with the business service layer. For example, a application service also can be classified as a business service if it interacts directly with applicatio

and contains embedded business rules.

Services that contain both application and business logic can be referred to as *hybrid application s* or just *hybrid services*. This service model is commonly found within traditional distributed archite It is not a recommended design when building service abstraction layers. Because it is so common, t it is discussed and referenced throughout this book.

Finally, an application service also can compose other, smaller-grained application services (such as services) into a unit of coarse-grained application logic. Aggregating application services is fre done to accommodate integration requirements. Application services that exist solely to enable inte between systems often are referred to as *application integration services* or simply *integration se* Integration services often are implemented as controllers.

Because they are common residents of the application service layer, now is a good time to introd *wrapper service* model. Wrapper services most often are utilized for integration purposes. They co services that encapsulate ("wrap") some or all parts of a legacy environment to expose legacy functi to service requestors. The most frequent form of wrapper service is a service adapter provided by vendors. This type of out-of-the-box Web service simply establishes a vendor-defined service interfa expresses an underlying API to legacy logic.



Business services are the lifeblood of contemporary SOA. They are responsible for expressing business logic through service-orientation and bring the representation of corporate business models into the Web services arena.

Application services can fall into different types of service model categories because they simply represent a group of services that express technology-specific functionality. Therefore, an application service can be a utility service, a wrapper service, or something else.

Business services, on the other hand, are always an implementation of the business service model. The sole purpose of business services intended for a separate business service layer is to represent business logic in the purest form possible. This does not, however, prevent them from implementing other service models. For example, a business service also can be classified as a controller service and a utility service.

In fact, when application logic is abstracted into a separate application service layer, it is more than likely that business services will act as controllers to compose available application services to execute their business logic.

Business service layer abstraction leads to the creation of two further business service models:

- *Task-centric business service* A service that encapsulates business logic specific to a task or

business process. This type of service generally is required when business process logic is not centralized as part of an orchestration layer. Task-centric business services have limited reuse potential.

- *Entity-centric business service* A service that encapsulates a specific business entity (such as an invoice or timesheet). Entity-centric services are useful for creating highly reusable and business process-agnostic services that are composed by an orchestration layer or by a service layer consisting of task-centric business services (or both).

When a separate application service layer exists, these two types of business services can be positioned to compose application services to carry out their business logic.

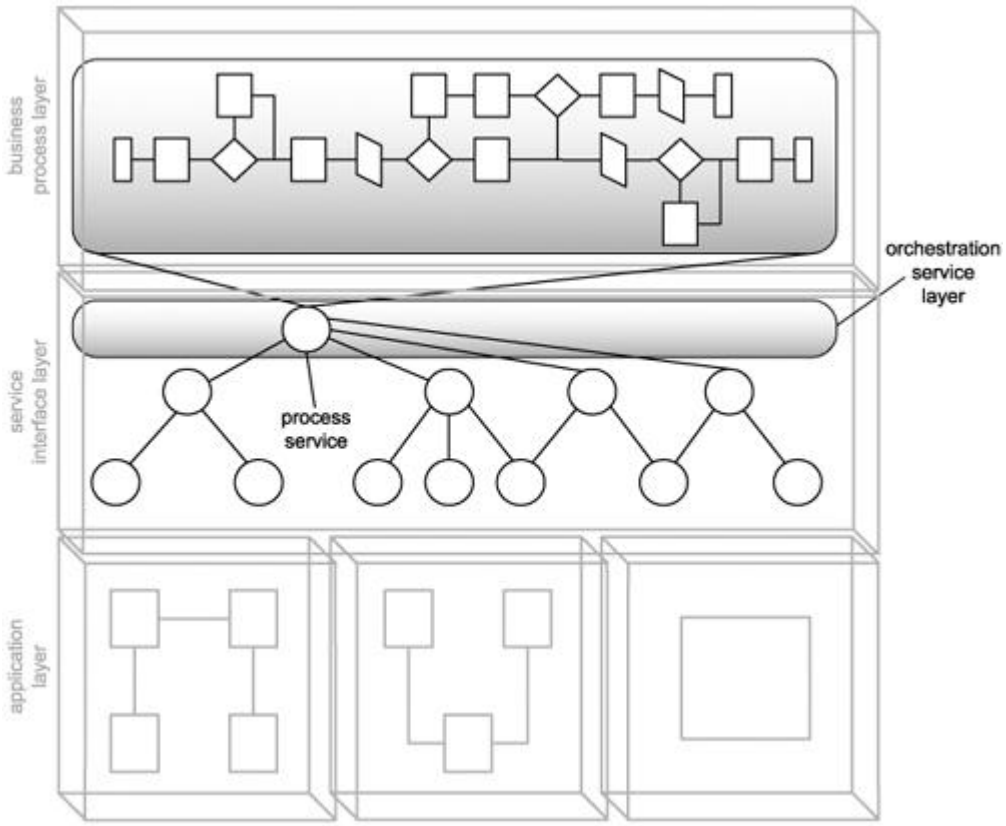
Orchestration service layer

Orchestration is more valuable to us than a standard business process, as it allows us to directly link process logic to service interaction within our workflow logic. This combines business process modeling with service-oriented modeling and design. And, because orchestration languages (such as WS-BPEL) realize workflow management through a process service model, orchestration brings the business process into the service layer, positioning it as a master composition controller.

The orchestration service layer introduces a parent level of abstraction that alleviates the need for other services to manage interaction details required to ensure that service operations are executed in a specific sequence. Within the orchestration service layer, *process services* compose other services that provide specific sets of functions, independent of the business rules and scenario-specific logic required to execute a process instance.

All process services are also controller services by their very nature, as they are required to compose other services to execute business process logic. Process services also have the potential of becoming utility services to an extent, if a process, in its entirety, should be considered reusable. In this case, a process service that enables orchestration can itself be orchestrated.

The orchestration service layer.


business process layer
orchestration service layer
service interface layer
process service
application layer