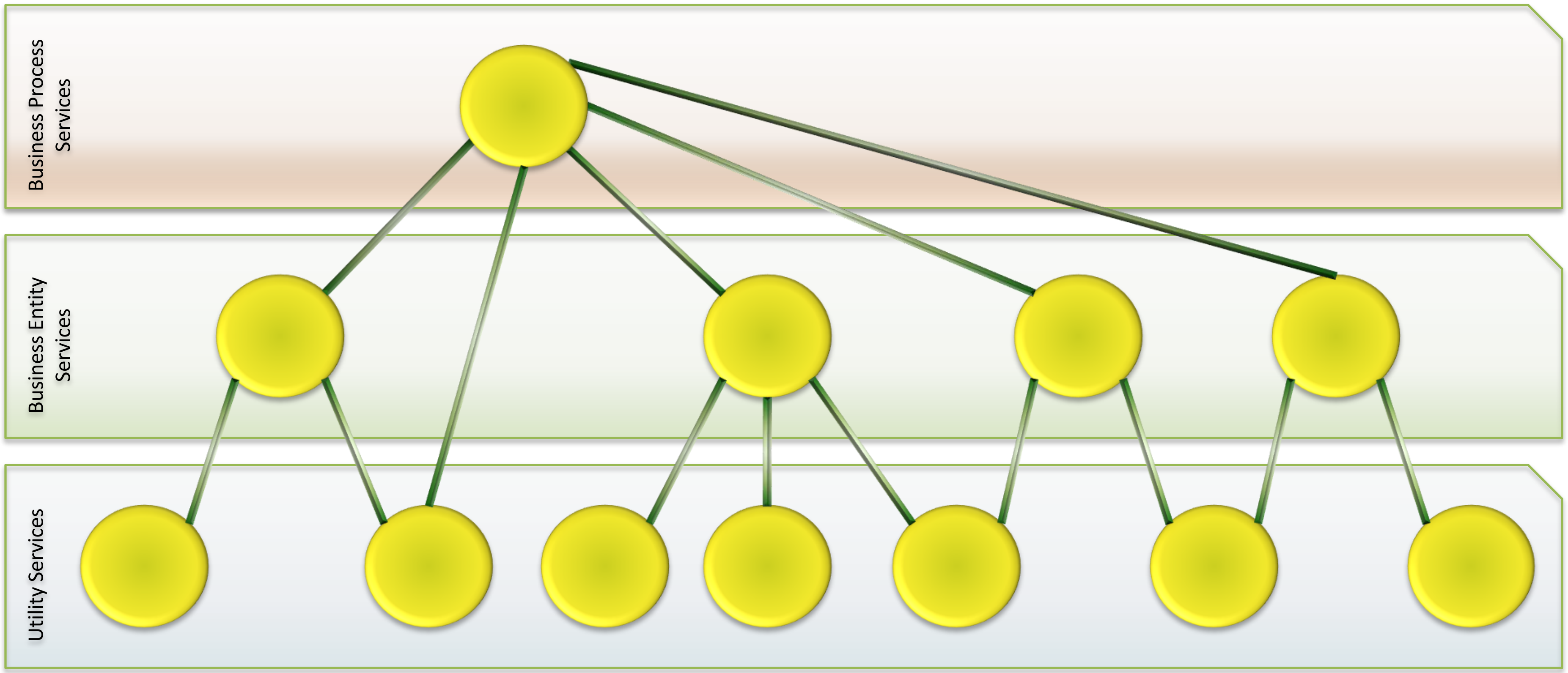
Service Design Principles

This extract is intended for quick reference purposes only. For complete treatment, refer to the source:

Erl, Thomas. SOA Principles of Service Design. New Jersey: Prentice Hall, 2007

Service Models

Services can be categorized based on the type of logic they encapsulate, the extent of reuse potential this logic has, and how this logic relates to existing domains within the enterprise.



While these layers tend to form a natural composition hierarchy, there are no hard-and-fast rules as to how services can be assembled.

Service Model Descriptions

Business Process Service

aka Task Services, Task-Centric Business Services, or Orchestrations

- Representative of a specific "parent" business task or process (e.g., BillPayment)
- Typically a controller responsible for composing more process-agnostic Business Entity Services
- Somewhat less reuse potential because of coupling to specific functional context
- May be implemented as an orchestration within an orchestration platform

Business Entity Service

aka Entity Services or Entity-Centric Business Services

- Representative of the organization's business entities (e.g., subscriber, payment, e-Bill)
- Highly reusable because it is agnostic to most parent business processes
- A single Business Entity Service can be leveraged to automate multiple parent business processes

Utility Service

aka Application Services, Infrastructure Services, or Technology Services

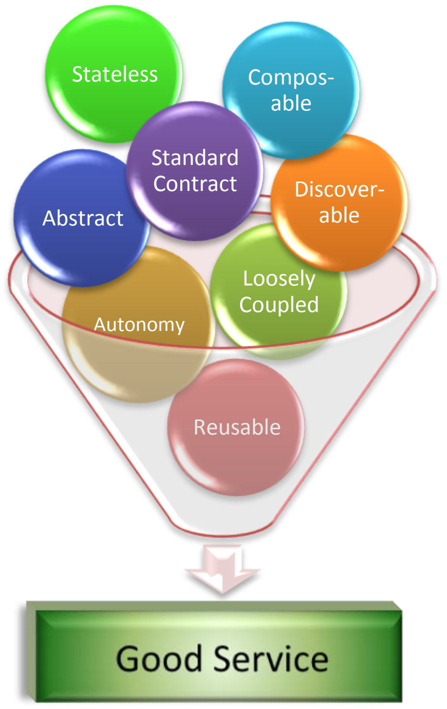
- Non-business-centric, providing distinct, technology-oriented services.
- Provides reusable, cross-cutting utility functionality such as event logging, notification, and exception handling.
- Application agnostic but focused on a specific processing context.

Service Design Principles

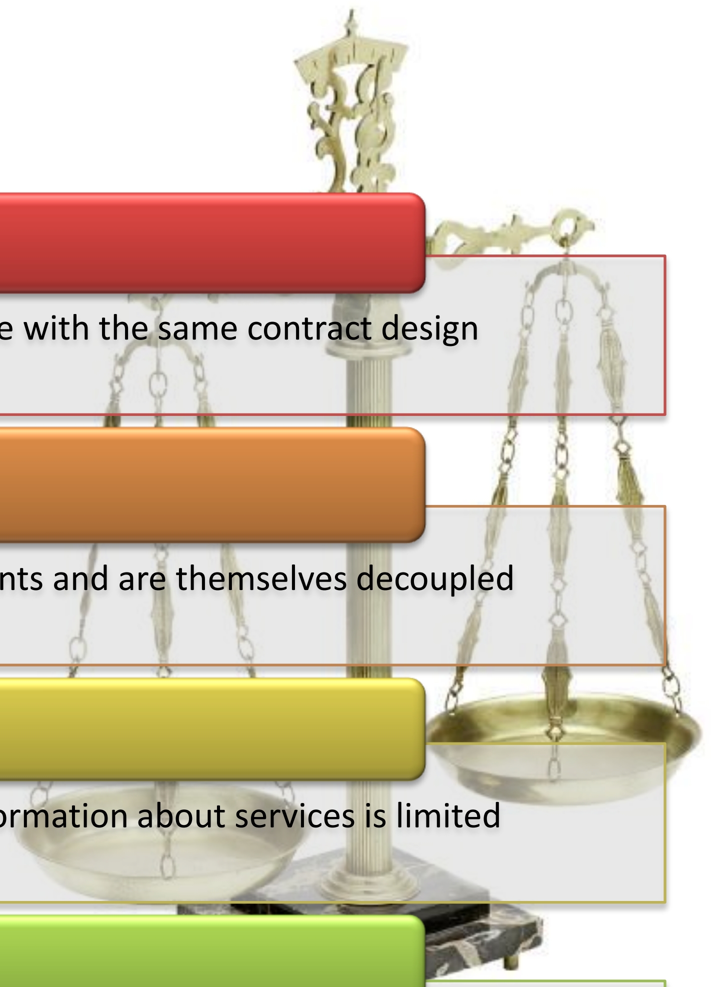
A Design Principle represents a highly recommended guideline for shaping solution logic in a certain way and with certain goals in mind.

Standard Service Design Principles

- Standardized Service Contract
- Service Loose Coupling
- Service Abstraction
- Service Reusability
- Service Autonomy
- Service Composability
- Service Statelessness
- Service Discoverability



Service Design Principles



Standardized Contract
Implement a standardized contract

- Services within the same service inventory are in compliance with the same contract design standards

Loose Coupling
Minimize dependencies

- Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment

Abstraction
Minimize the availability of meta information

- Service contracts only contain essential information and information about services is limited to what is published in service contracts

Reusability
Implement generic and reusable logic and contract

- Services contain and express agnostic logic and can be positioned as reusable enterprise resources

Service Design Principles

Autonomy
Implement independent functional boundary and runtime environment

- Services exercise a high level of control over their underlying runtime execution environment

Composability
Maximize composability

- Services are effective composition participants, regardless of the size and complexity of the composition

Statelessness
Implement adaptive and state management-free logic

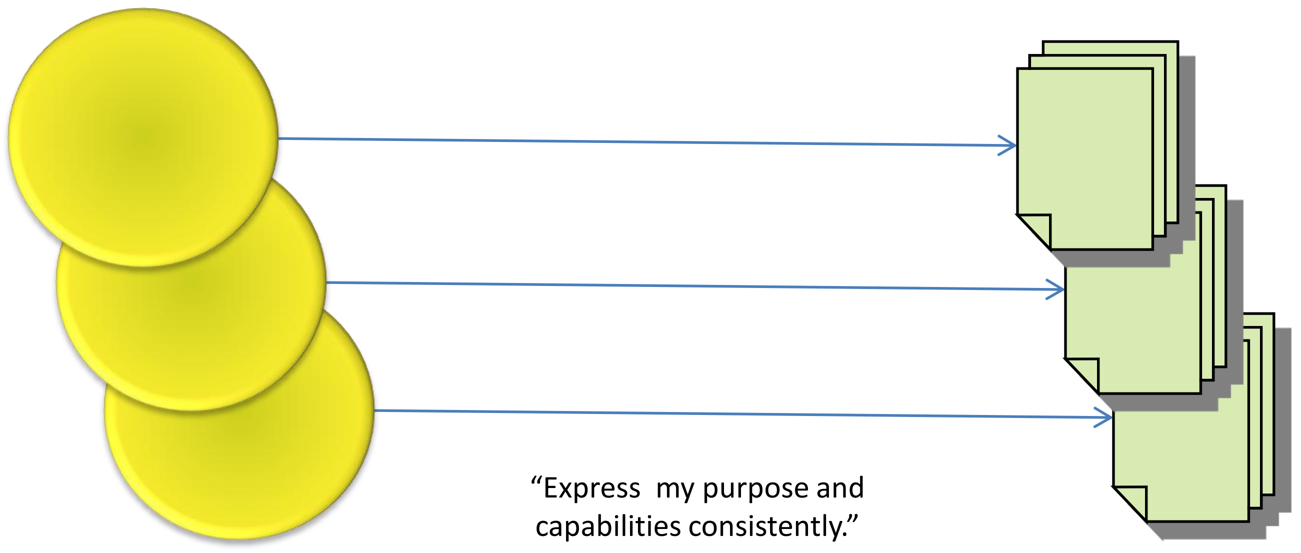
- Services minimize resource consumption by deferring the management of state information when necessary

Discoverability
Implement communicative meta information

- Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted

Service Role - Contracts

The fundamental role of this principle is to ensure consistent expression of the overall purpose and capabilities of the service.

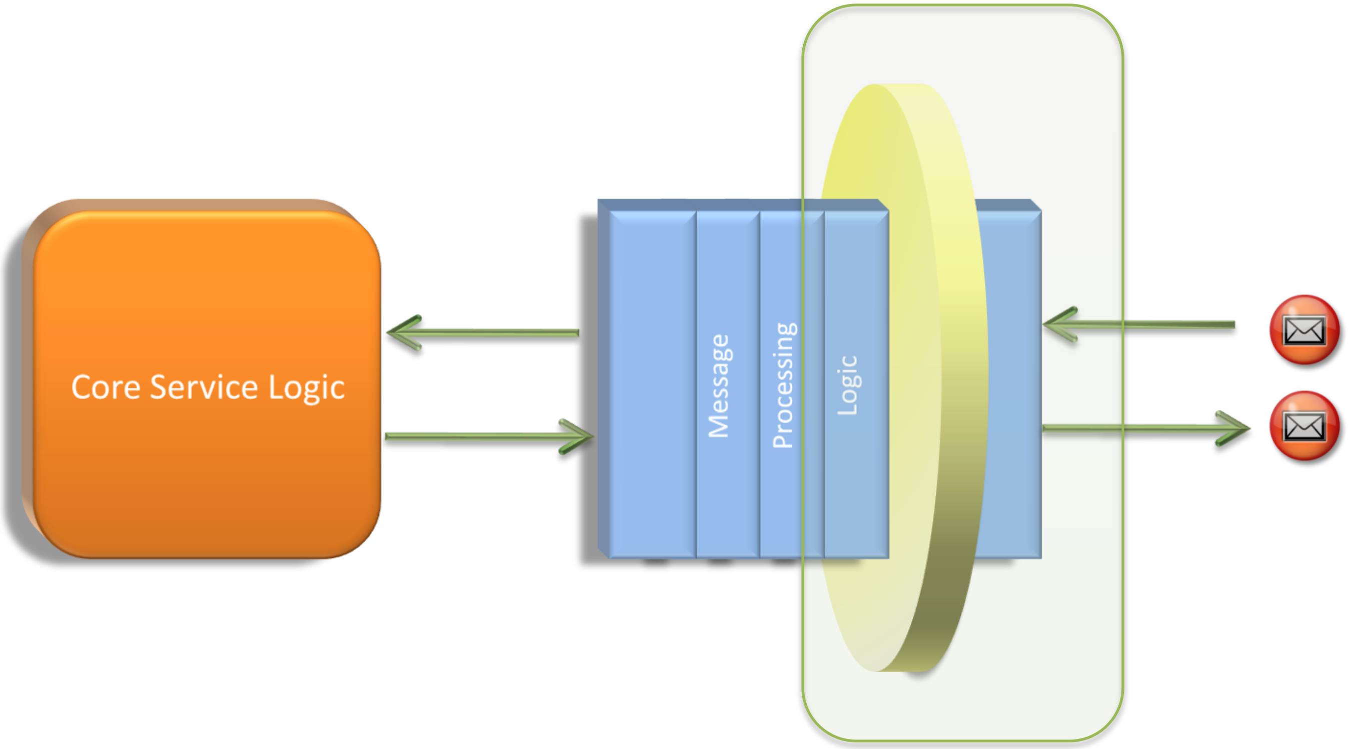


Service Profile - Contracts

Short Definition	Services share standardized contracts.
Long Definition	Services within the same service inventory are in compliance with the same contract design standards.
Goals	• To enable services with a meaningful level of natural interoperability within the boundary of a service inventory. This reduces the need for data transformation because consistent data models are used for information exchange. • To allow the purpose and capabilities of services to be more easily and intuitively understood. The consistency with which service functionality is expressed through service contracts increases interpretability and the overall predictability of service endpoints throughout a service inventory. *These goals are further supported by other service design principles.*
Design Characteristics	• A Service Contract (comprised of a technical interface or one or more service description documents) is provided with the Service. • The Service Contract is standardized through the application of design standards.

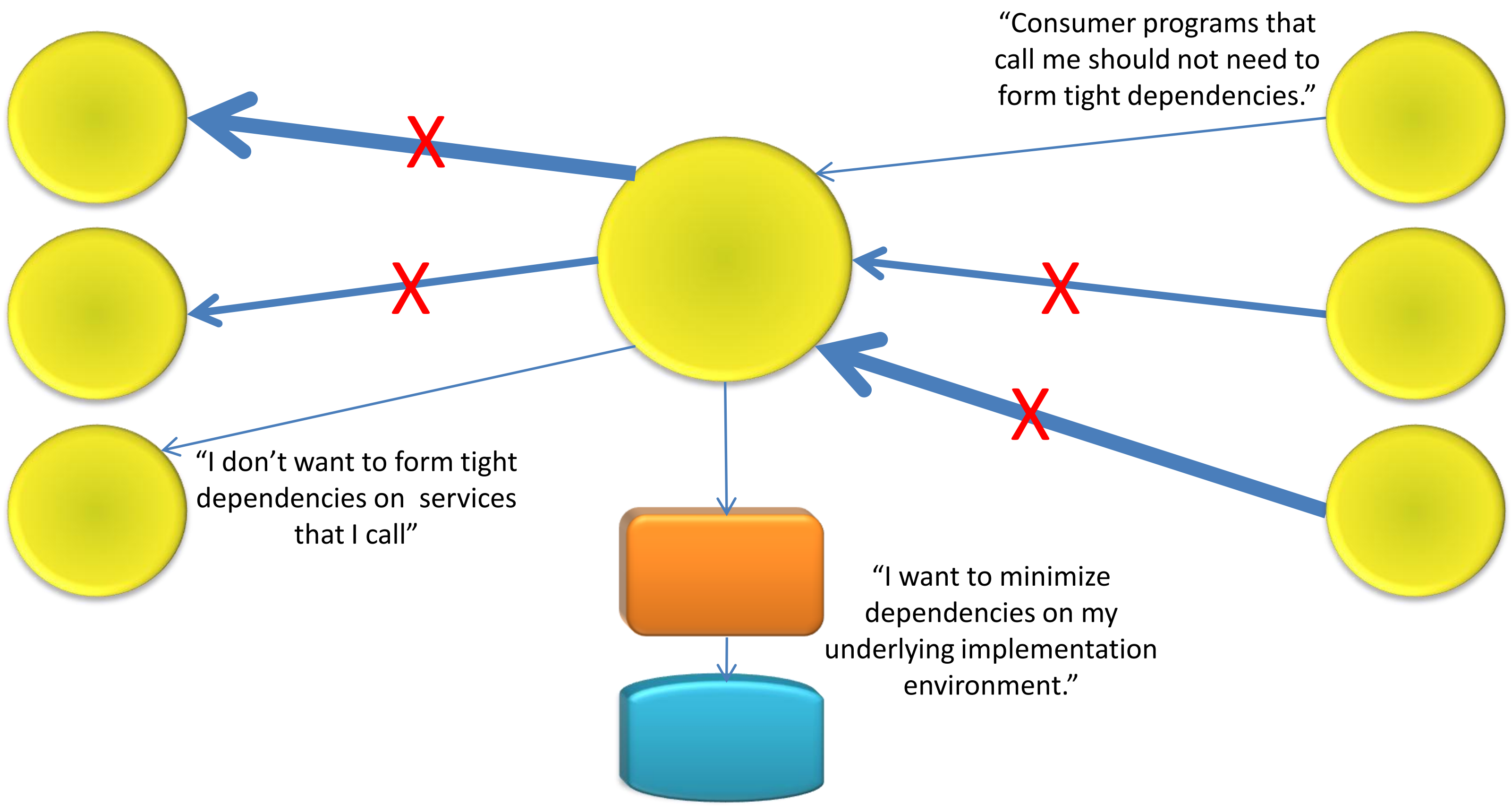
Service Region Influence - Contracts

The Contracts principle influences the Service Contract.



Service Role - Coupling

This principle emphasizes the reduction ("loosening") of coupling between the parts of a service-oriented solution, especially when compared to how applications have traditionally been designed. Specifically, loose coupling is advocated between a service contract and its consumers and between a service contract and its underlying implementation.

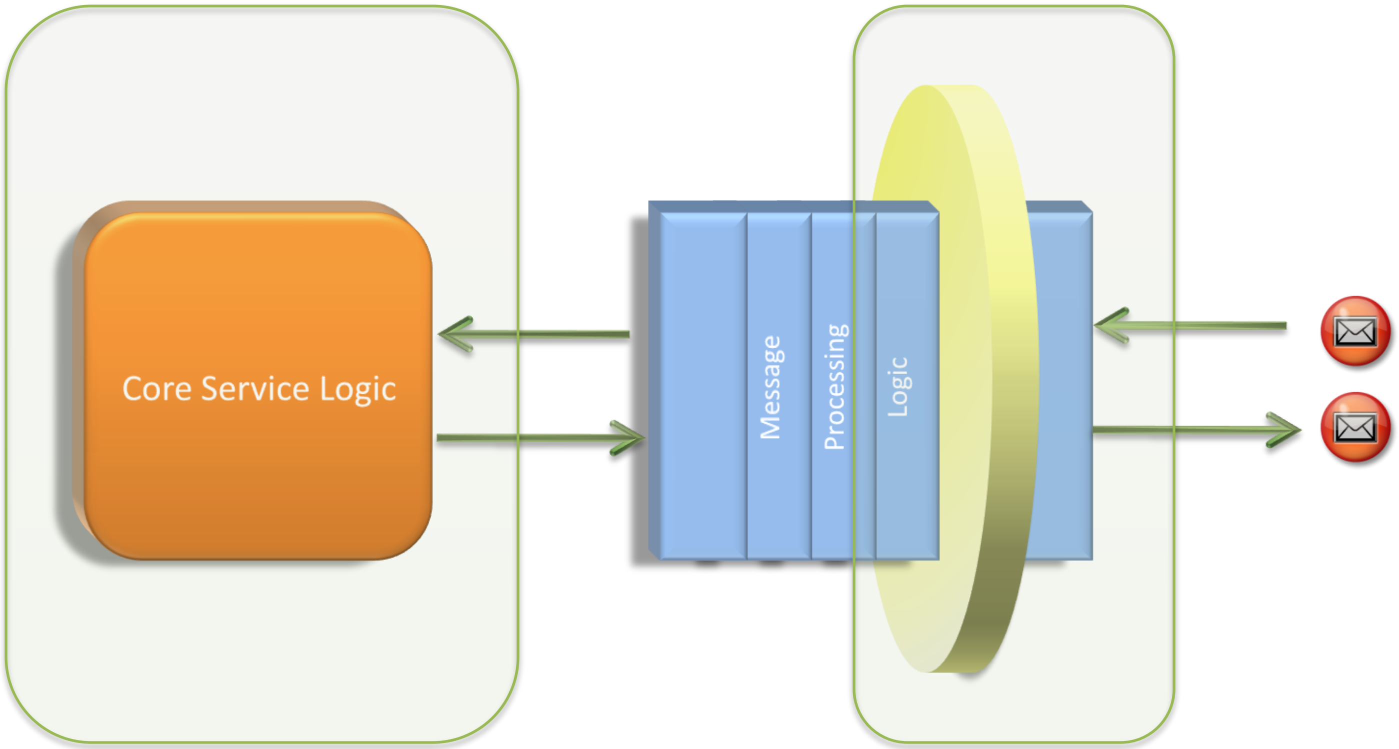

"Consumer programs that call me should not need to form tight dependencies."
"I don't want to form tight dependencies on services that I call"
"I want to minimize dependencies on my underlying implementation environment."
X
X
X
X

Service Profile - Coupling

Short Definition	Services are loosely coupled.
Long Definition	Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.
Goals	• By consistently fostering reduced coupling within and between services we are working toward a state where service contracts increase independence from their implementations and services are increasingly independent from each other. • This promotes an environment in which services and their consumers can be adaptively evolved over time with minimal impact on each other.
Design Characteristics	• The existence of a Service Contract that is ideally decoupled from technology and implementation details. • A functional service context that is not dependent on outside logic. • Minimal consumer coupling requirements.

Service Region Influence - Coupling

The Coupling principle primarily influences the Service Contract, and certain aspects of the principle influence the Service Logic.



Service Contract Coupling - Types

Service Contract Coupling

- Logic-to-Contract
- Contract-to-Logic
- Contract-to-Technology
- Contract-to-Implementation
- Contract-to-Functional

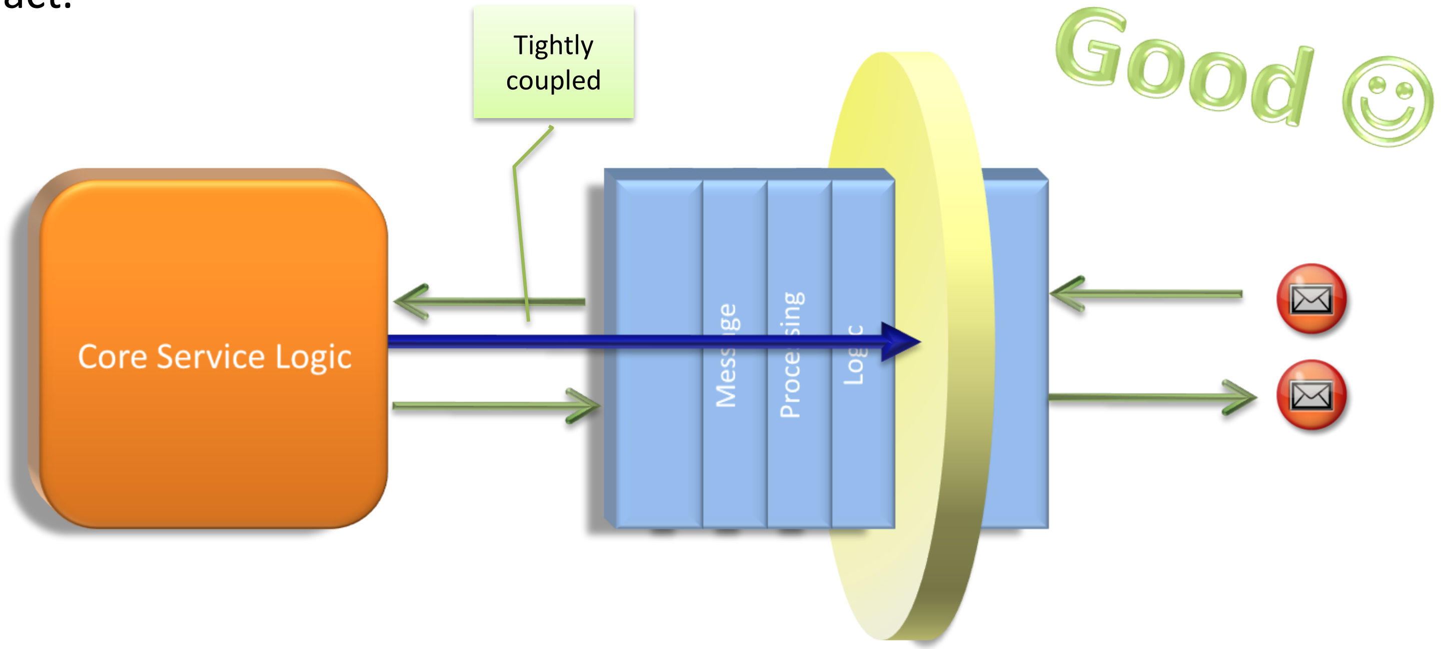
Service Consumer Coupling

- Consumer-to-Contract
- Consumer-to-Implementation

Service Contract Coupling Types

Logic-to-Contract Coupling

A Service created through the contract-first process will naturally result in the service logic forming a tightly coupled relationship on the service contract.

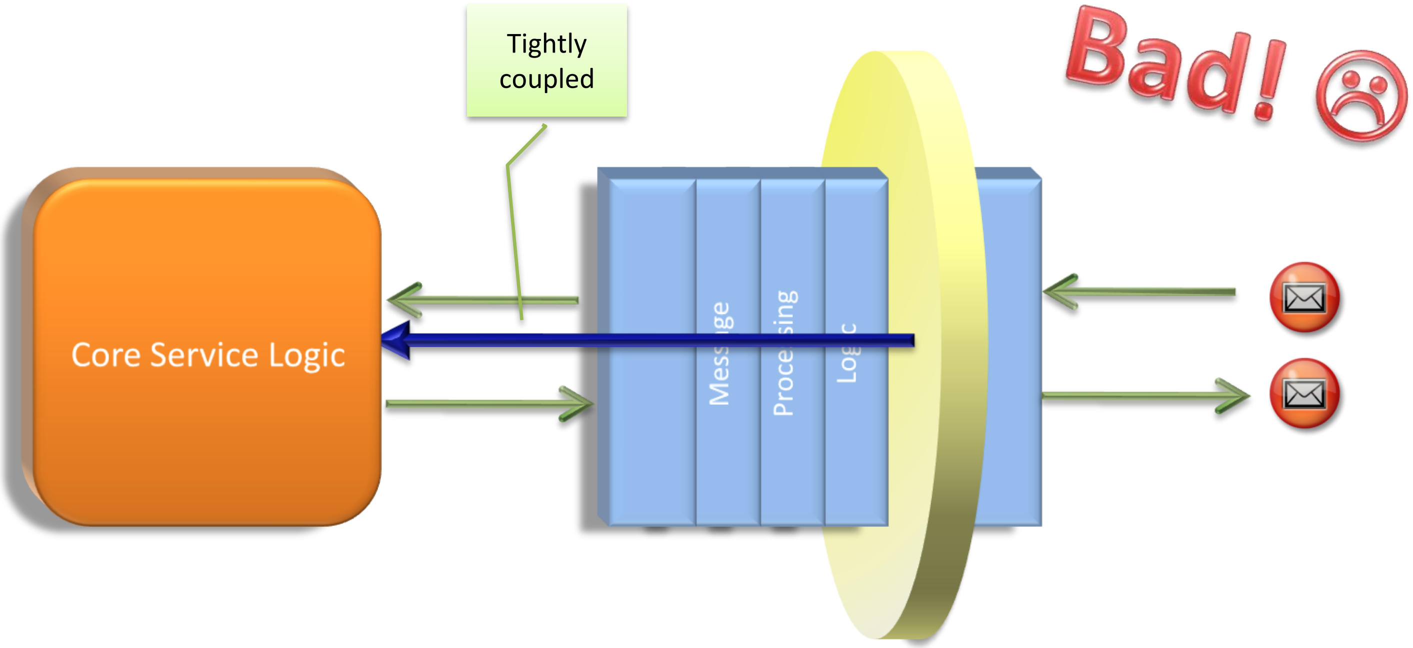


The Service Contract is not coupled to the logic at all, allowing the service logic to be replaced in the future without affecting service consumers that have formed dependencies on the contract.

Service Contract Coupling Types

Contract-to-Logic Coupling

Allowing the Service Contract to be determined by the Service Logic is an established anti-pattern that shortens the lifespan of the Service Contract and inhibits the long-term evolution of the Service.

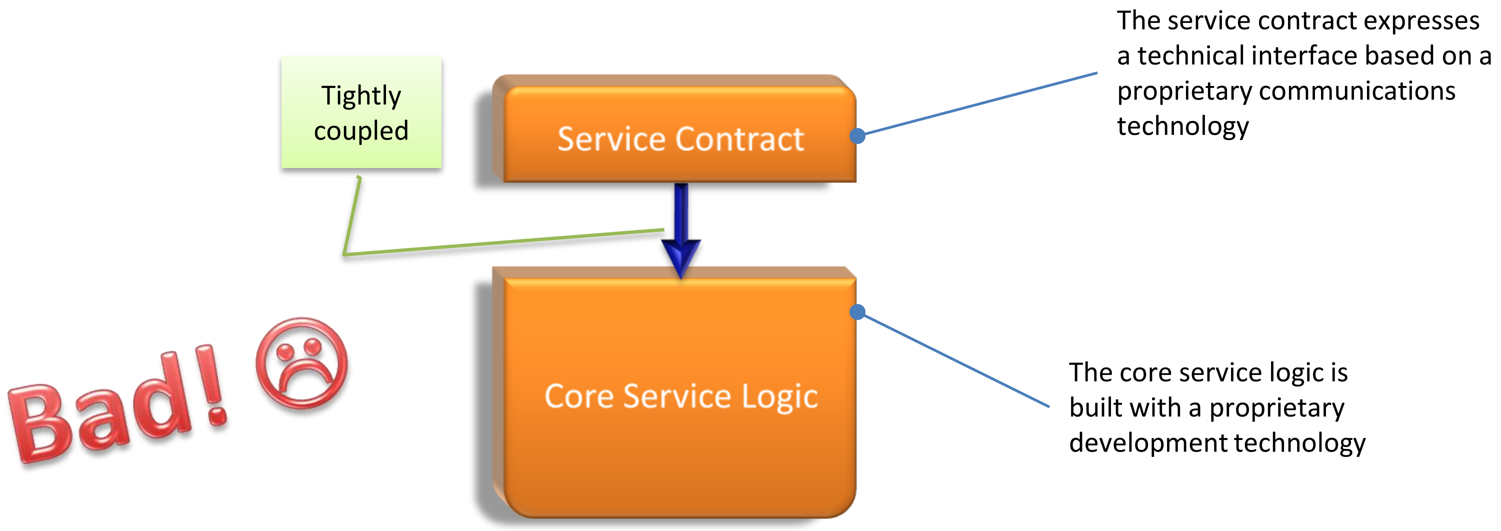


Common examples of Contract-to-Logic examples are the auto-generation of WSDL definitions using component interfaces as the basis for the contract design, as well as the auto-generation of XML schemas from database tables and other parts of physical data models.

Service Contract Coupling Types

Contract-to-Technology Coupling

A Service developed as a proprietary component can require that the service contract exist as a proprietary extension of the service. This couples the contract to the implementation technology which, in turn, imposes the requirement that all service consumers support the same non-standard communications protocol.

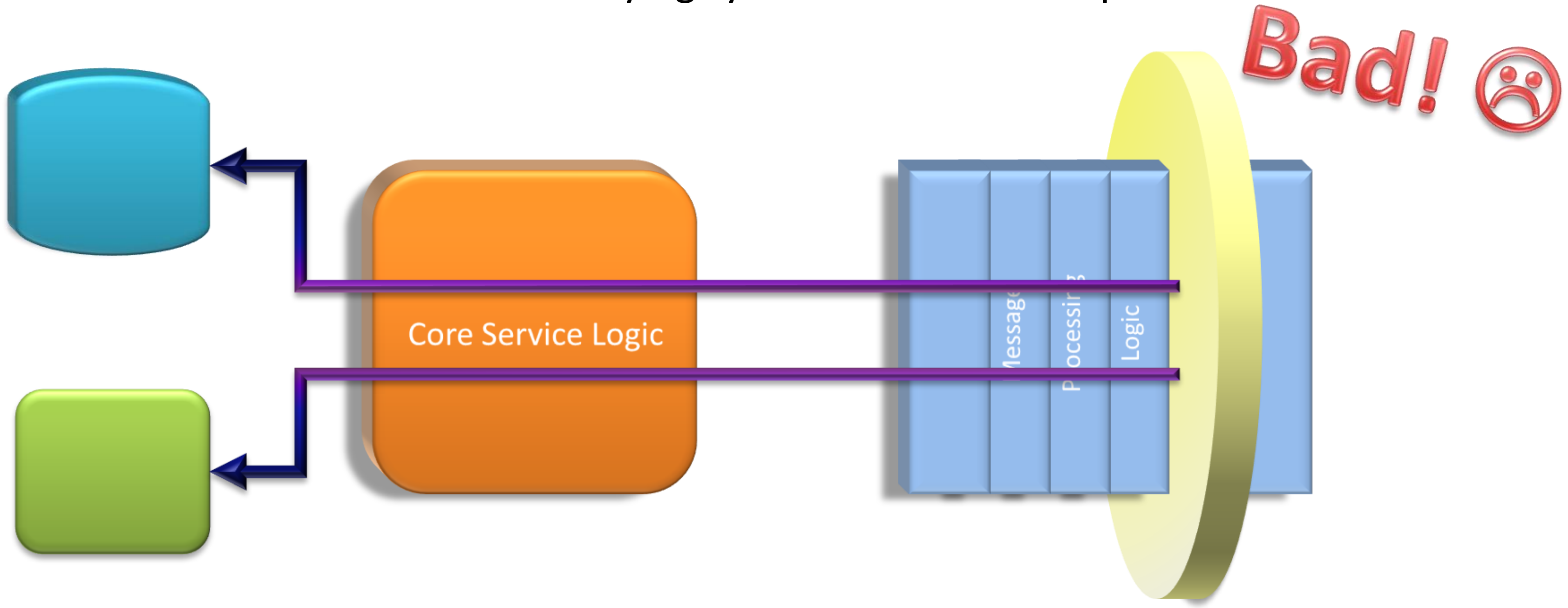

Tightly coupled
Service Contract
The service contract expresses a technical interface based on a proprietary communications technology
Bad!
Core Service Logic
The core service logic is built with a proprietary development technology

Making the technical service contract dependent on proprietary technology limits the potential consumers to those who are capable of supporting the technology.

Service Contract Coupling Types

Contract-to-Implementation Coupling

It's important to prevent characteristics of the implementation environment from becoming embedded within the Contract content. Legacy system dependencies and database schemas for underlying systems are common pitfalls.



Contracts must not be tied to implementation characteristics, but it is relatively normal (and appropriate) for some forms of service logic to be bound and connected to certain implementation technologies and products beyond the core service logic. This enables the logic to effectively access and interact with these resources.

Service Contract Coupling Types

Contract-to-Functional Coupling

Logic encapsulated by a Service should not be designed specifically in support of a body of functionality that exists outside of the Service boundary.

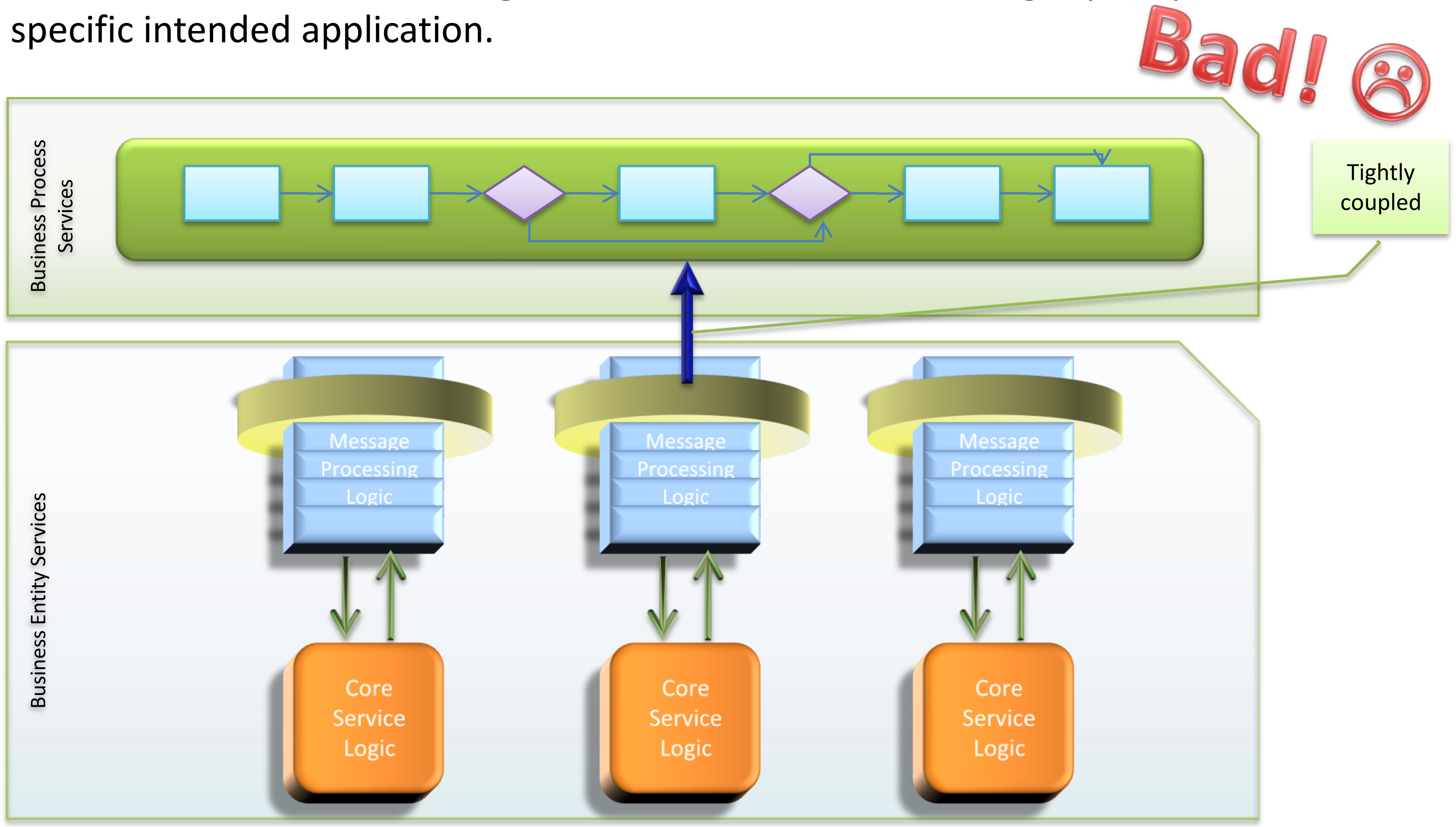
- Parent Process Coupling
  - The Service Logic and Contract can become tightly coupled to a parent process if the service is designed to specifically support a particular business process.
- Service-to-Consumer Coupling
  - The Service Logic and Contract can become tightly coupled and exhibit consumer-specific functional coupling if the service is designed to support a single consumer.

Task Services typically limit their functional scope and context of a particular business process (as opposed to the agnostic context of a Business Service) and therefore exhibit an intentional or *targeted* functional coupling.

Service Contract Coupling Types

Contract-to-Functional Coupling

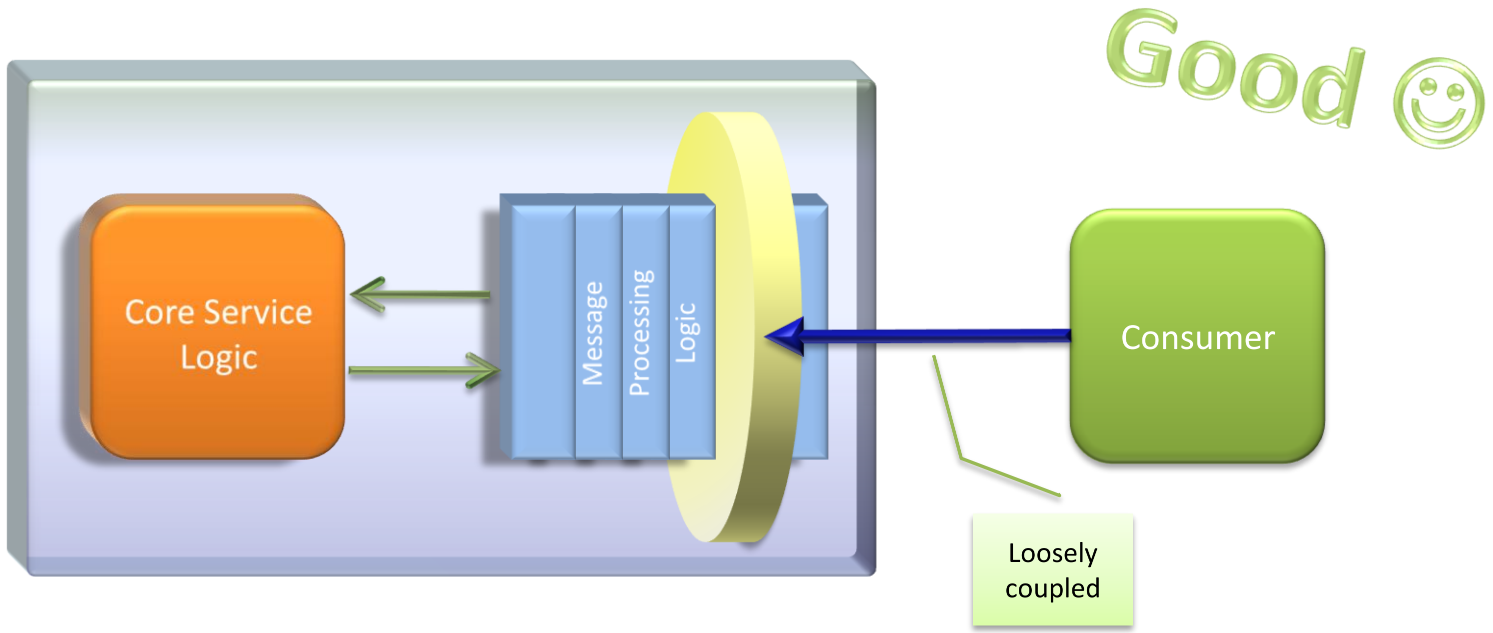
Service Contracts should be agnostic of context rather than tightly coupled to a specific intended application.



Service Consumer Coupling Types

Consumer-to-Contract Coupling

Consumer-to-Contract coupling is a desirable form of coupling because it achieves the greatest amount of independence between the consumer and the service.

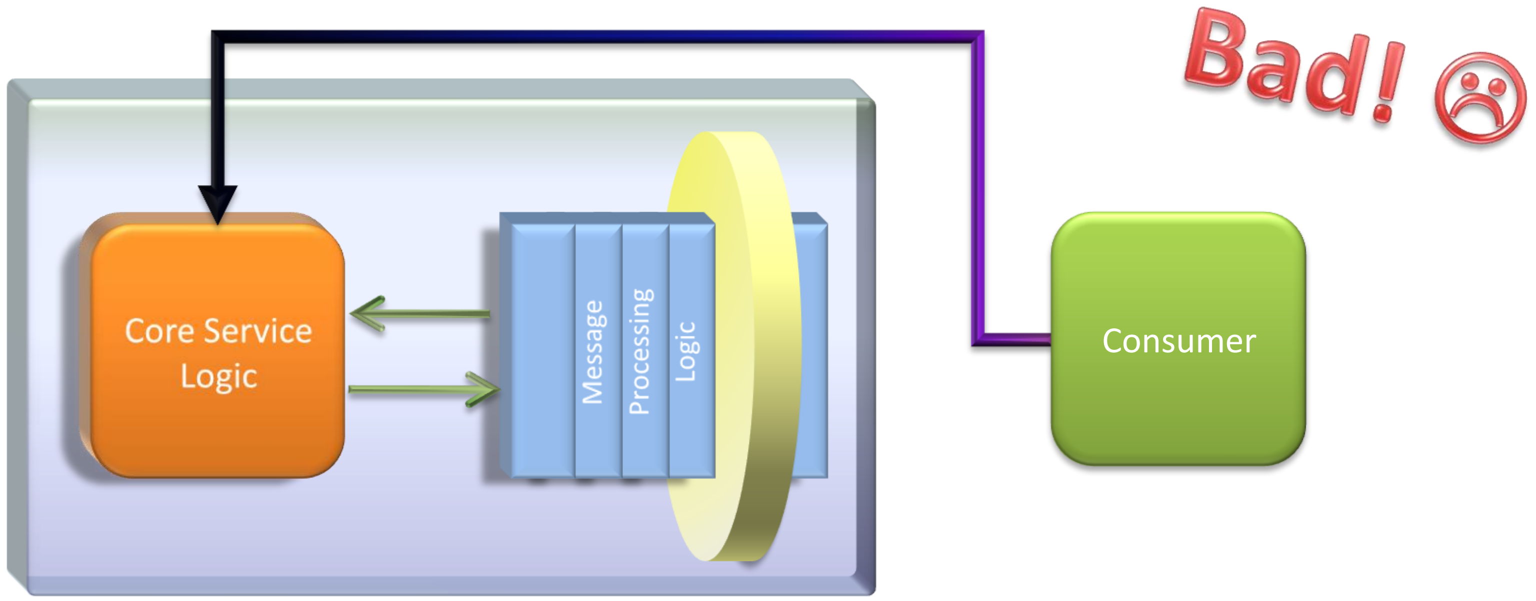


Though this is a desirable form of coupling, Service Contracts are designed to minimize the degree of coupling experienced by the Consumer. Likewise, Consumers are designed to avoid any non-essential coupling to the Service Contract.

Service Consumer Coupling Types

Consumer-to-Implementation Coupling

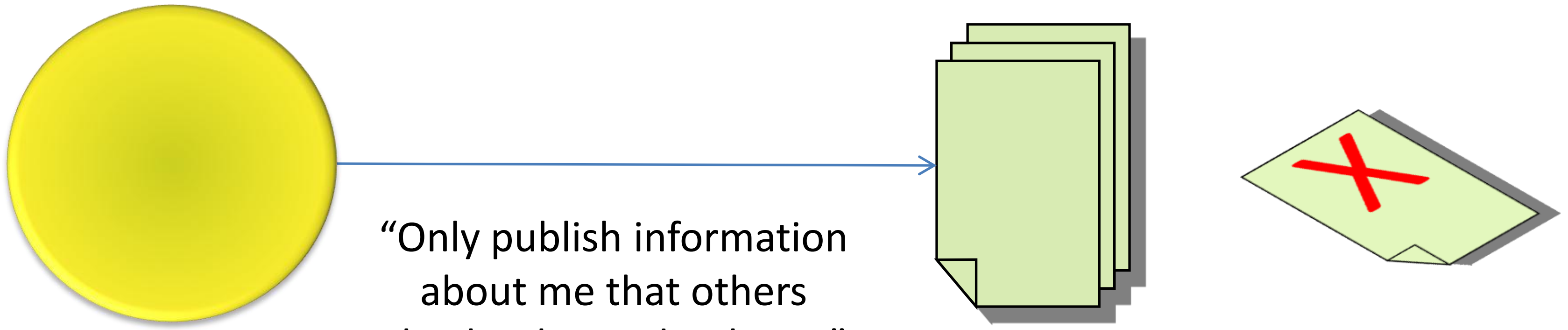
When consumers bypass the Service Contract and exploit other entry points to the desired functionality, the future of both the Service and the Consumer are inhibited.



With services orientation, the Service Contract is a first-class citizen and must be the ONLY means of access to the functional responsibilities of the Service. This was not true in many past integration architectures.

Service Role - Abstraction

The Abstraction principle helps to avoid the proliferation of unnecessary service information, meta or otherwise.


"Only publish information about me that others absolutely need to know"

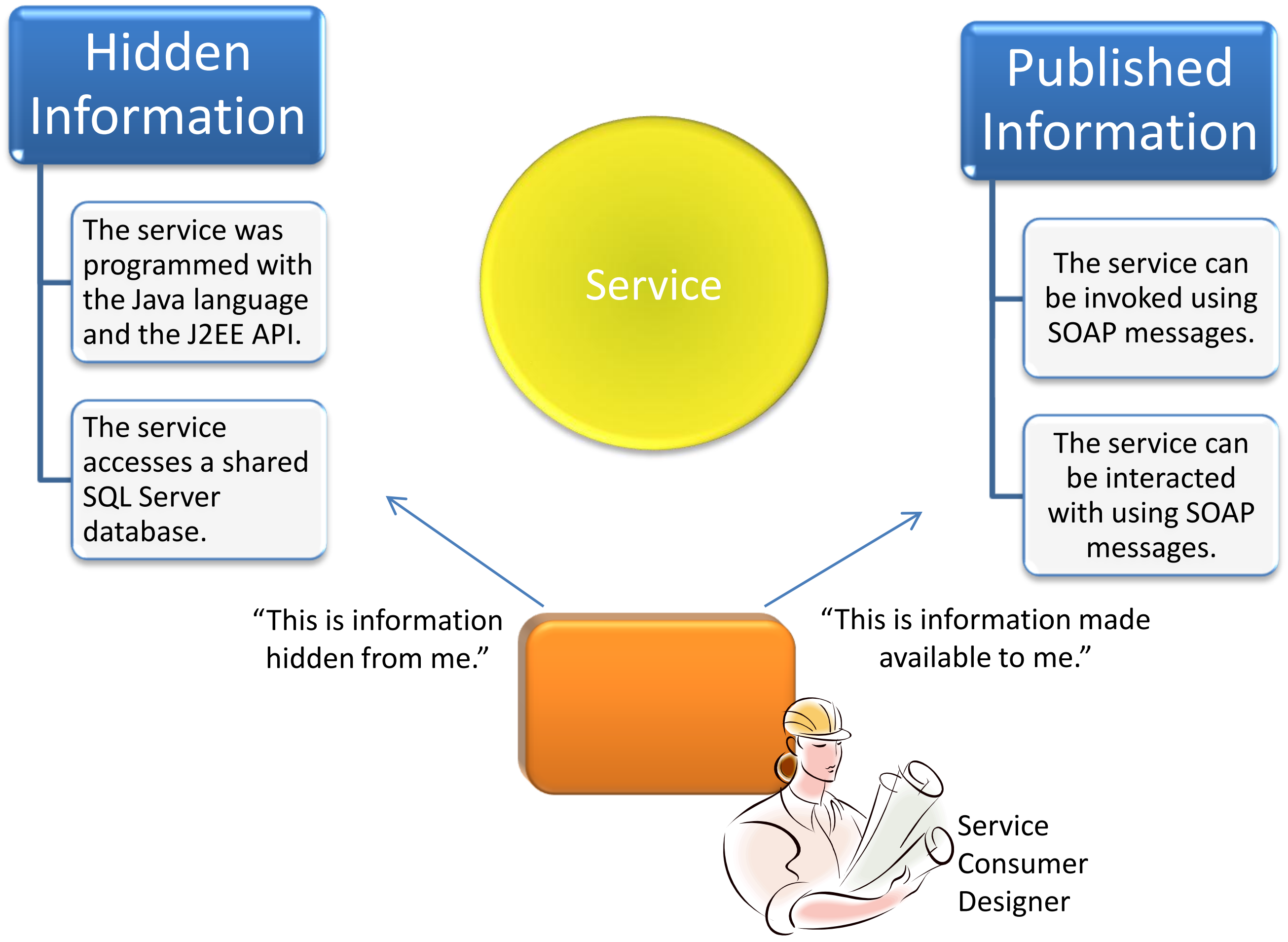
Service Profile - Abstraction

Short Definition	Non-essential service information is abstracted.
Long Definition	Service contracts only contain essential information and information about services is limited to what is published in the service contracts.
Goals	Many of the other principles emphasize the need to publish more information in the service contract. The primary role of this principle is to keep the quantity and detail of contract content concise and balanced and prevent unnecessary access to additional service details.
Design Characteristics	• Services consistently abstract specific information about technology, logic, and function away from the outside world (the world outside of the service boundary). • Services have contracts that concisely define interaction requirements and constraints and other required service meta details. • Outside of what is documented in the Service Contract, information about a service is controlled or altogether hidden within a particular environment.

Service Profile - Abstraction

Technology Information Abstraction

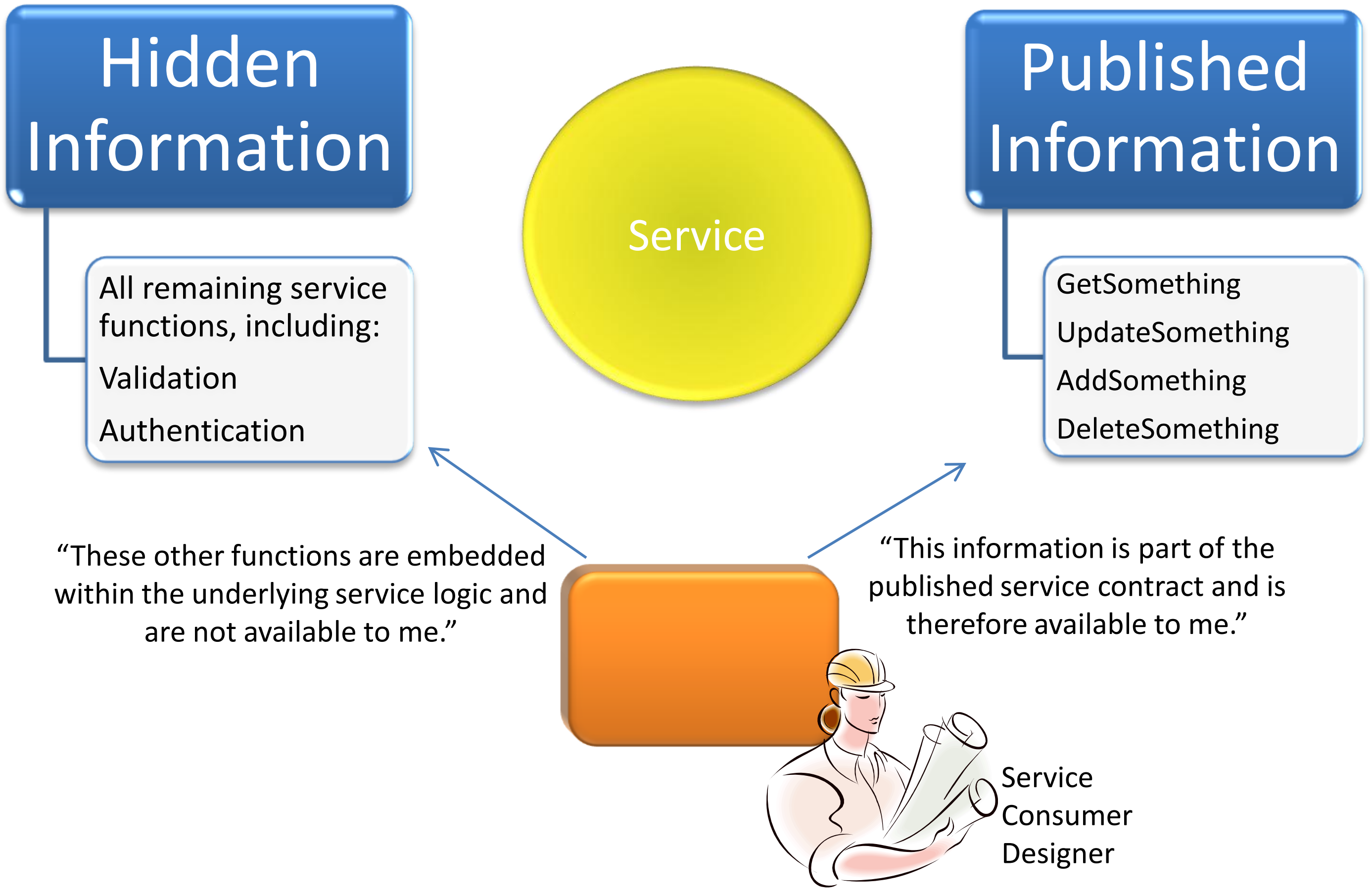
Services only reveal information to each other relevant to their runtime invocation and interaction requirements.


Hidden Information
The service was programmed with the Java language and the J2EE API.
The service accesses a shared SQL Server database.
Service
Published Information
The service can be invoked using SOAP messages.
The service can be interacted with using SOAP messages.
"This is information hidden from me."
"This is information made available to me."
Service Consumer Designer

Service Profile - Abstraction

Functional Abstraction

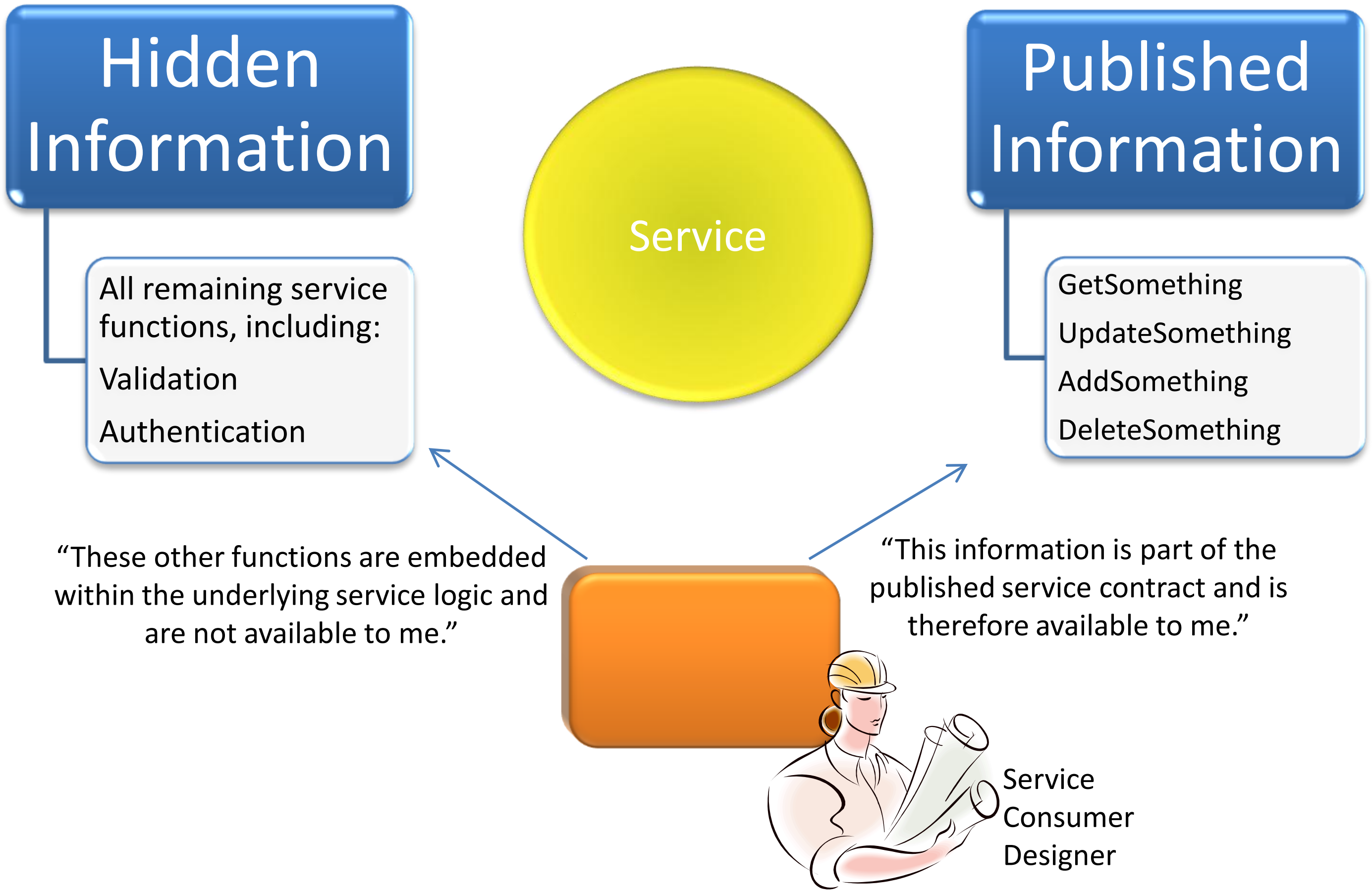
Only certain functions are exposed through the Service Contract. This limits the extent to which a consumer program can be built to programmatically interface and interact with the service.


Hidden Information
All remaining service functions, including:
Validation
Authentication
Service
Published Information
GetSomething
UpdateSomething
AddSomething
DeleteSomething
"These other functions are embedded within the underlying service logic and are not available to me."
"This information is part of the published service contract and is therefore available to me."
Service Consumer Designer

Service Profile - Abstraction

Programmatic Logic Abstraction

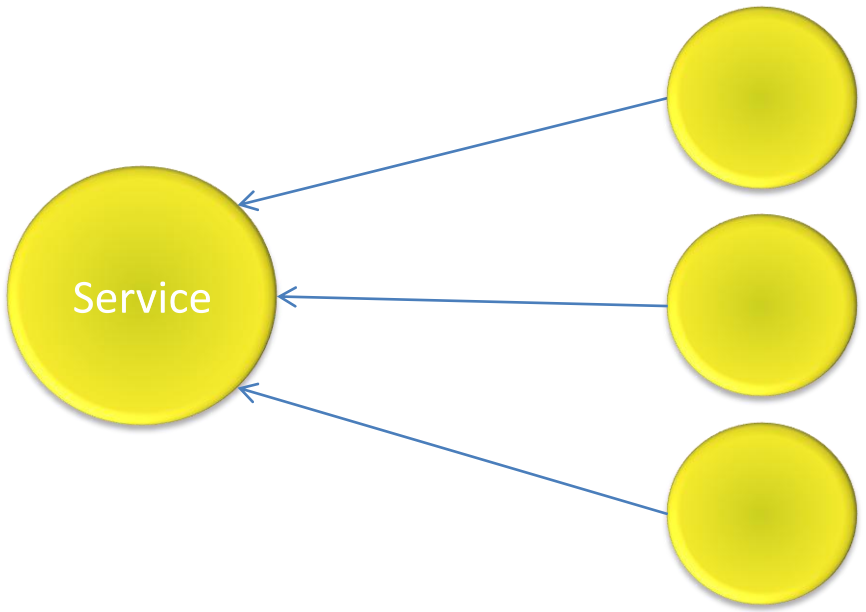
sss


Hidden Information
All remaining service functions, including:
Validation
Authentication
Service
Published Information
GetSomething
UpdateSomething
AddSomething
DeleteSomething
"These other functions are embedded within the underlying service logic and are not available to me."
"This information is part of the published service contract and is therefore available to me."
Service Consumer Designer

Service Role - Reusability

The Reusability principle strives to get the most possible value out of each piece of software.

“Make my capabilities useful for more than one purpose.”

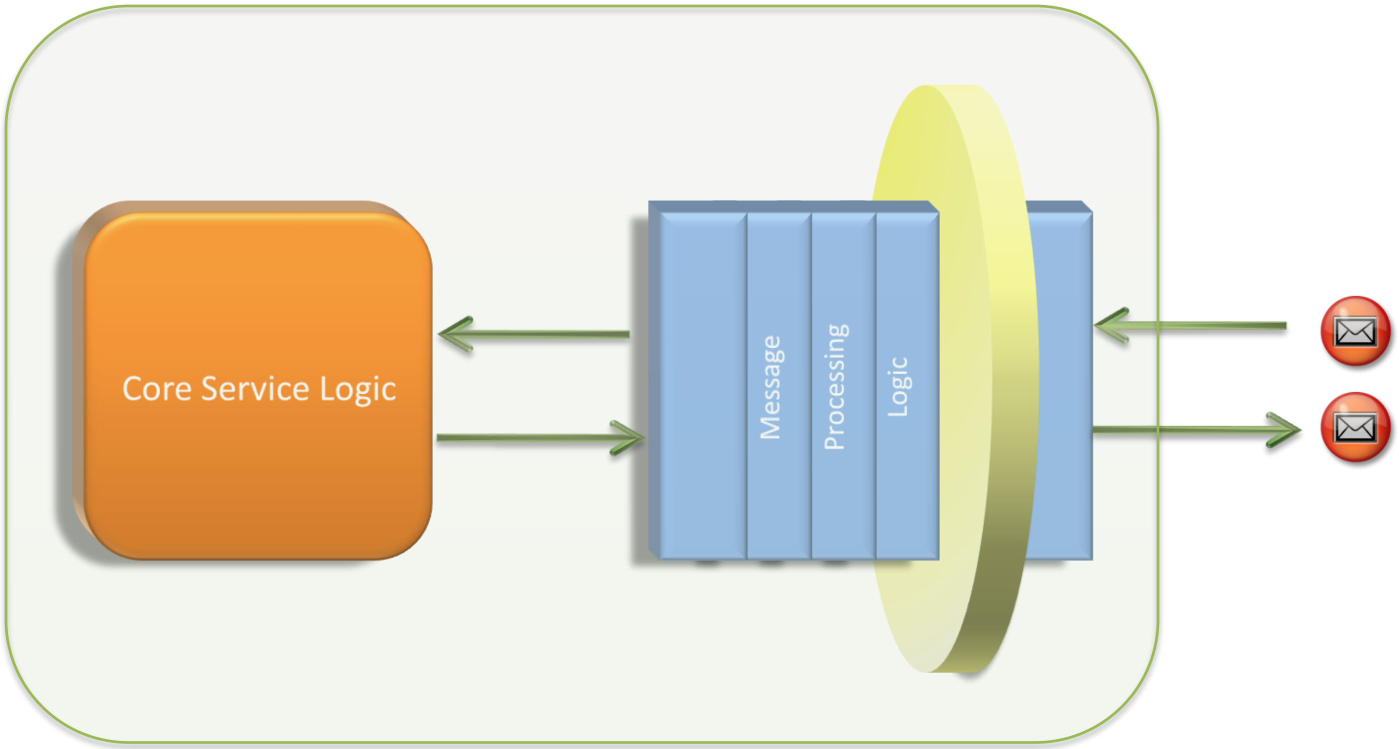


Service Profile - Reusability

Short Definition	Services are reusable.
Long Definition	Services contain and express agnostic logic and can be positioned as reusable enterprise resources.
Goals	The goals behind Service Reusability are tied directly to some of the most strategic objectives of service-oriented computing: • To allow for service logic to be repeatedly leveraged over time so as to achieve an increasingly high return on the initial investment of delivering the service. • To increase business agility on an organizational level by enabling the rapid fulfillment of future business automation requirements through wide-scale service composition. • To enable the creation of service inventories with a high percentage of agnostic services.
Design Characteristics	• *The service is defined by an agnostic functional context* – the logic encapsulated by the service is associated with a context that is sufficiently agnostic to any one usage scenario so as to be considered reusable. • *The service logic is highly generic* – the logic encapsulated by the service is sufficiently generic, allowing it to facilitate numerous usage scenarios by different types of service consumers. • *The service has a generic and extensible contract* – the service contract is flexible enough to process a range of input and output messages.

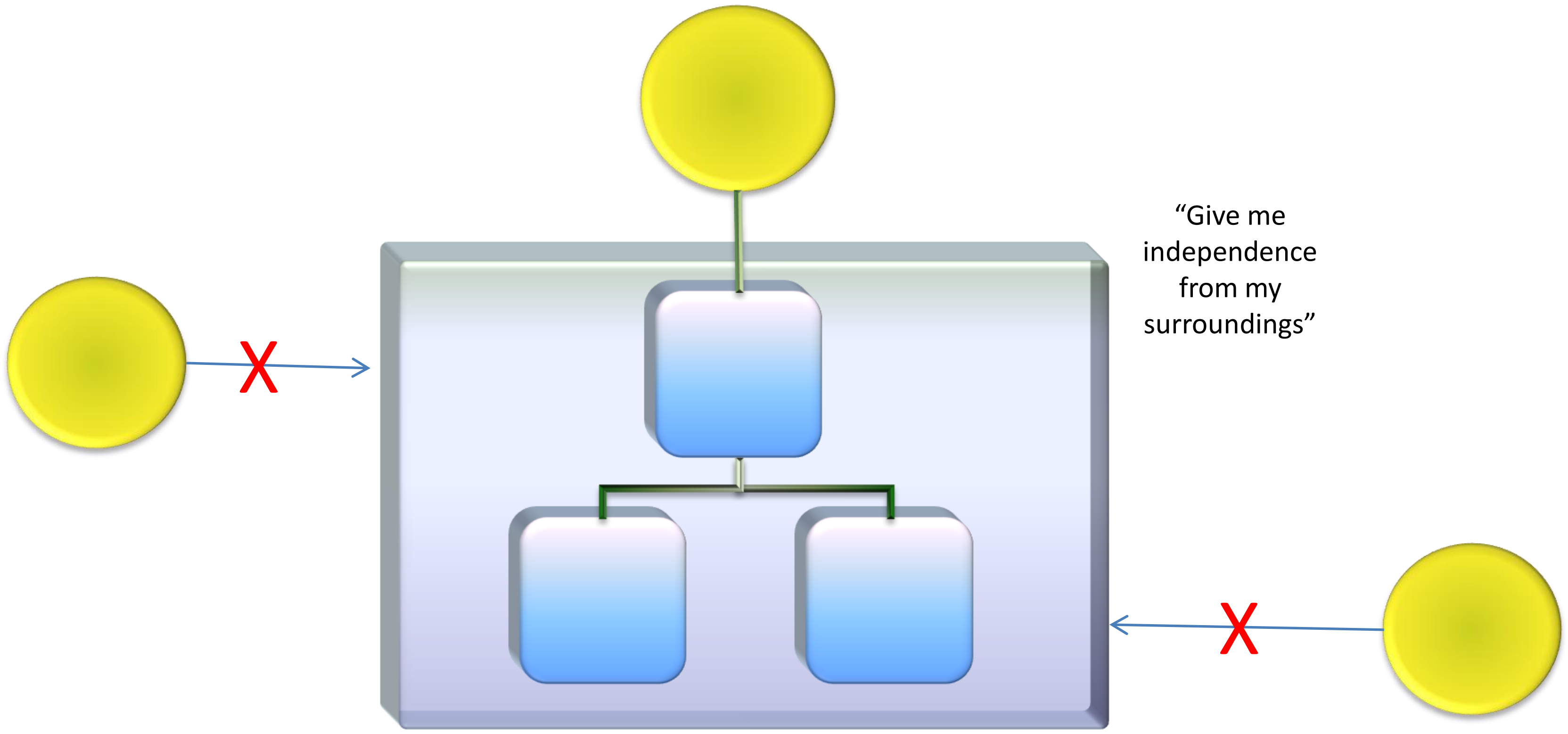
Service Region Influence - Reusability

The Reusability principle can affect all parts of a Service. The Contract design, the Message Processing Logic, and the underlying Core Service Logic can all be shaped by a service's reusability requirements.



Service Role - Autonomy

The Autonomy principle encourages the independence of a service implementation.



"Give me independence from my surroundings"

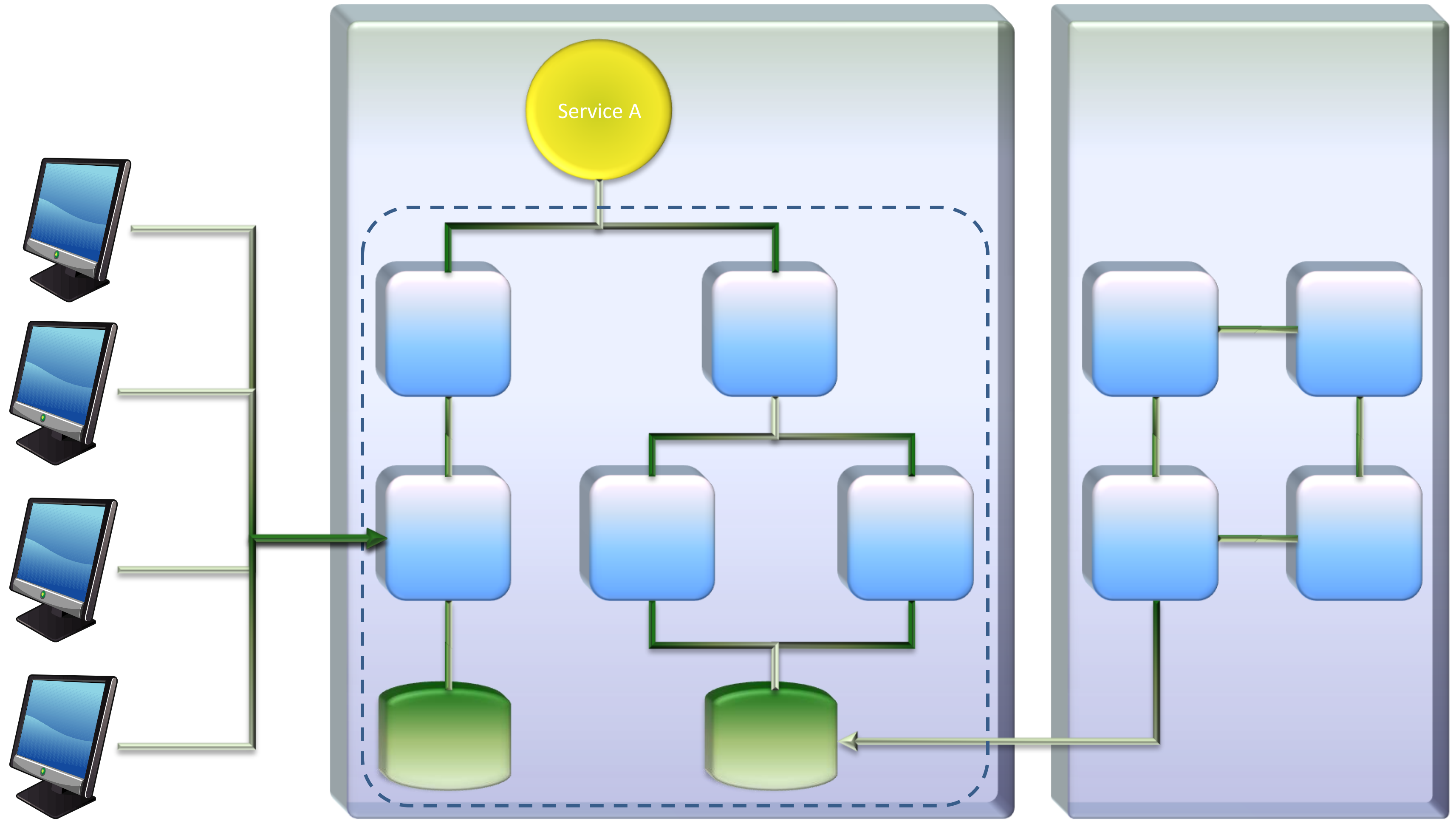
Service Profile - Autonomy

Short Definition	Services are autonomous.
Long Definition	Services exercise a high level of control over their underlying runtime execution environment.
Goals	• To increase a service's runtime reliability, performance, and predictability, especially when being reused and composed. • To increase the amount of control a service has over its runtime environment. By pursuing autonomous design and runtime environments, we are essentially aiming to increase post-implementation control over the service and the service's control over its own execution environment.
Design Characteristics	• Services have a contract that expresses a well-defined functional boundary that should not overlap with other services. • Services are deployed in an environment over which they exercise a great deal (and preferably an exclusive level) of control. • Service instances are hosted by an environment that accommodates high concurrency for scalability purposes.

Service Autonomy Types

Shared Autonomy

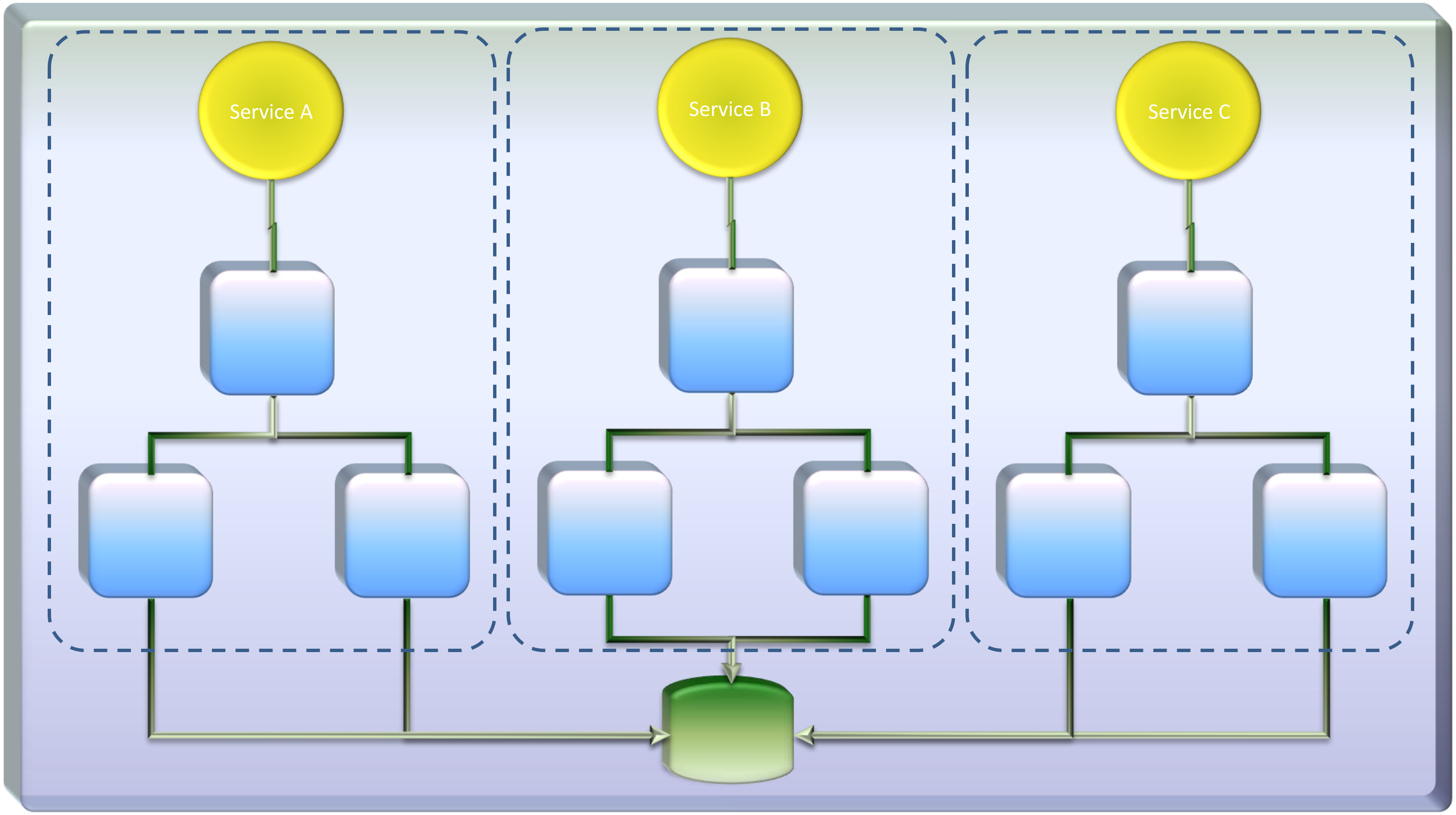
Service A encapsulates a legacy application with an existing user-base and a point-to-point integration channel.



Service Autonomy Types

Service Logic Autonomy (partially isolated services)

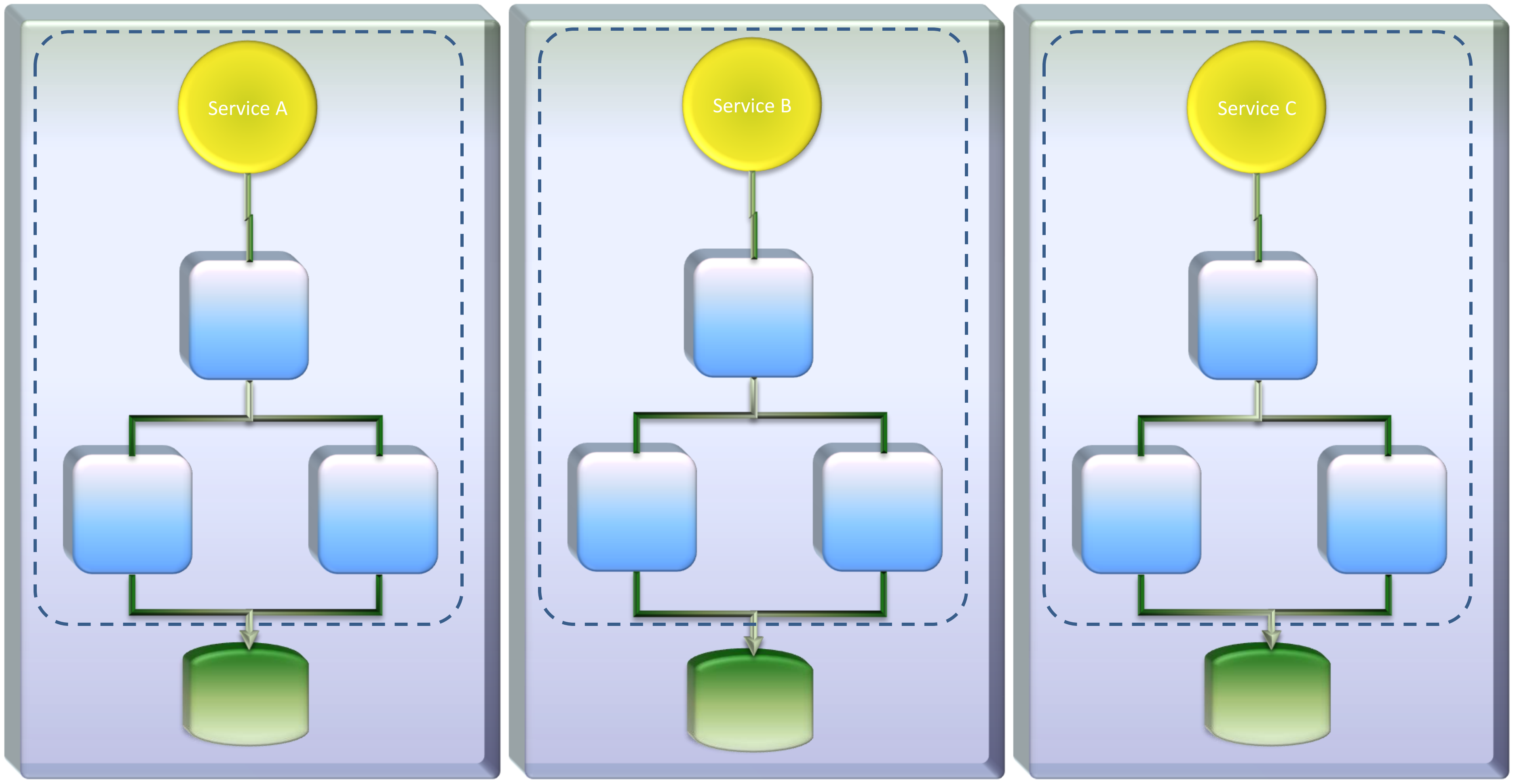
Service A, B, and C are each implemented with dedicated components, but all three services share the same database.



Service Autonomy Types

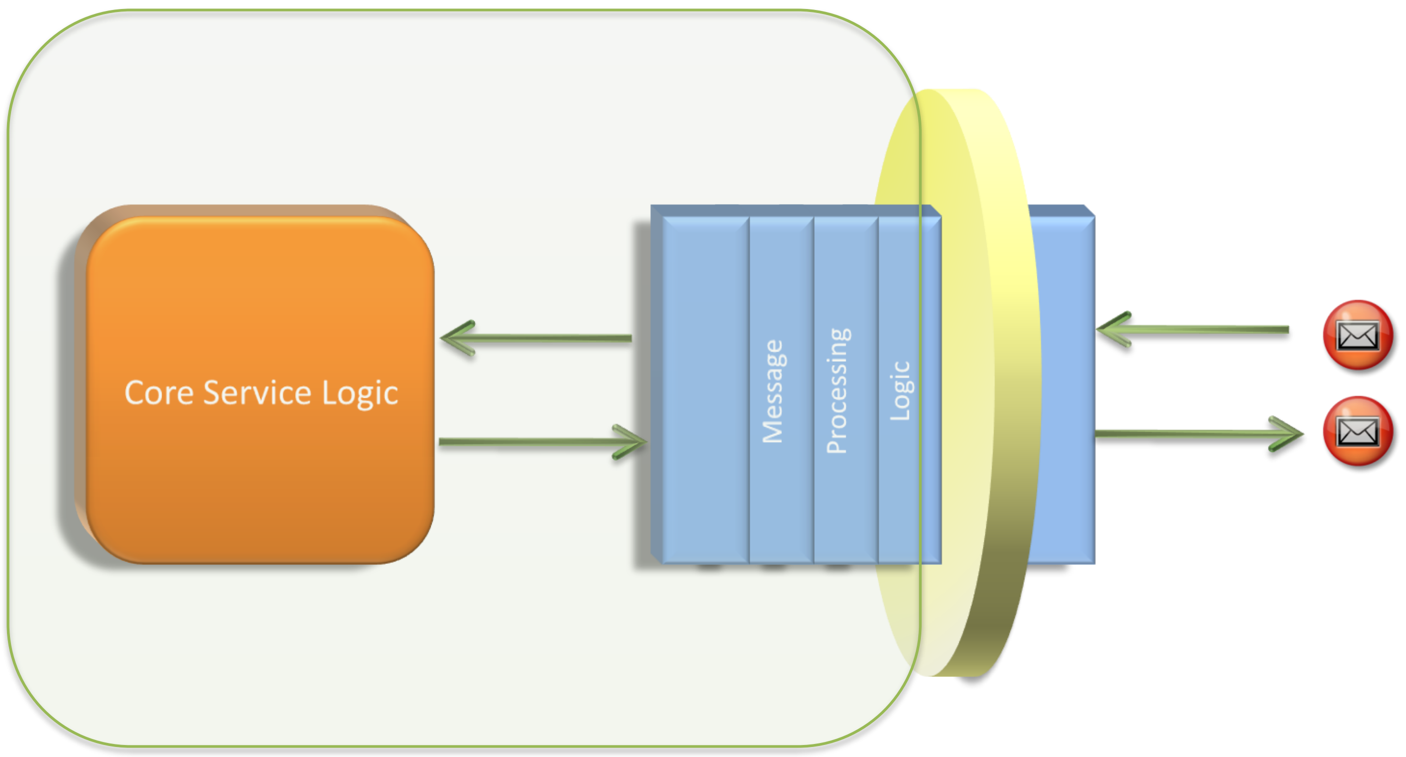
Pure Autonomy (isolated services)

Service A, B, and C are each implemented with dedicated components and isolated database and runtime environments.



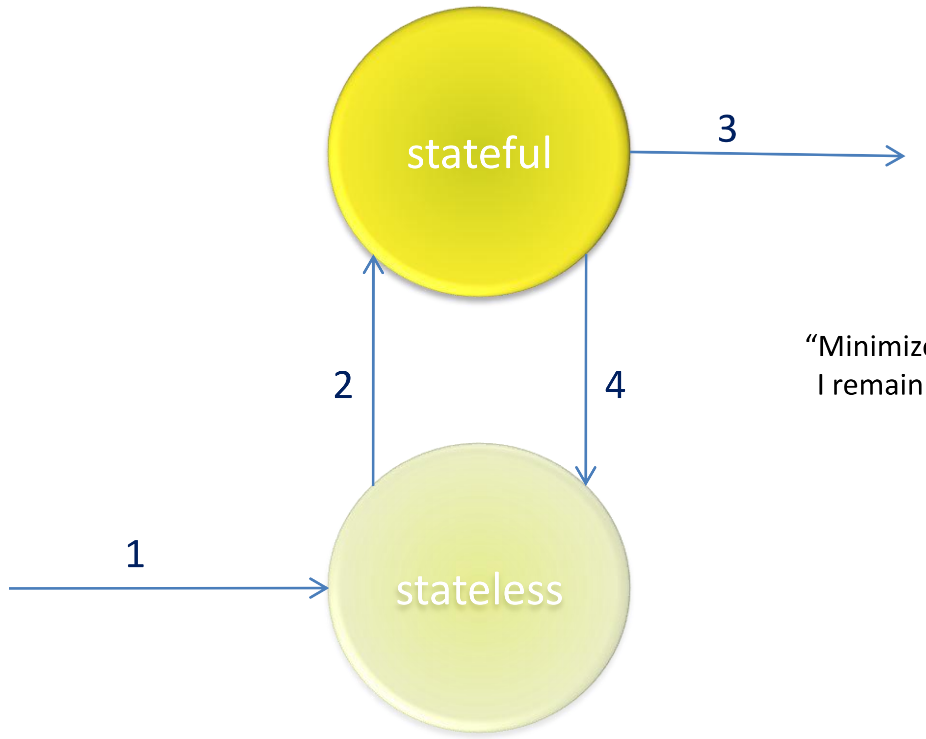
Service Region Influence - Autonomy

The Autonomy principle is almost exclusively focused on the service implementation, with an emphasis on the Core Service Logic. In some situations, the Service Contract may also be affected.



Service Role - Statelessness

The Statelessness principle encourages the incorporation of state management deferral extensions within the Service Design so as to keep services in a stateless condition wherever appropriate.



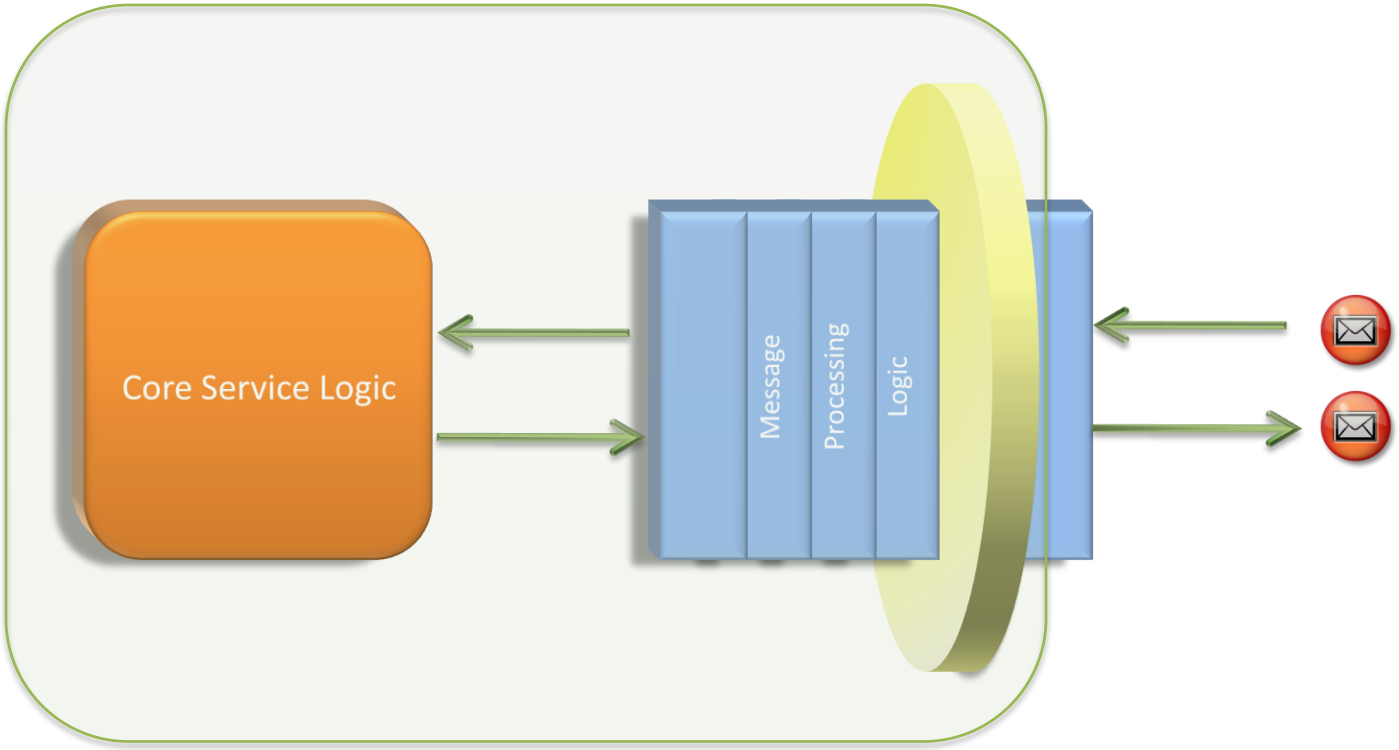
"Minimize the time I remain stateful"

Service Profile - Statelessness

Short Definition	Services minimize statefulness.
Long Definition	Services minimize resource consumption by deferring the management of state information when necessary.
Goals	• To increase service scalability. • To support the design of agnostic service logic and improve the potential for service reuse.
Design Characteristics	What makes this somewhat of a unique principle is the fact that it is promoting a condition of the service that is temporary in nature. Depending on the service model and state deferral approach used, different types of design characteristics can be implemented. Some examples include: • Highly business process-agnostic logic so that the service is not designed to retain state information for any specific parent business process. • Less constrained service contracts so as to allow for the receipt and transmission of a wider range of state data at runtime. • Increased amounts of interpretative programming routines capable of parsing a range of state information delivered by messages and responding to a range of corresponding action requests.

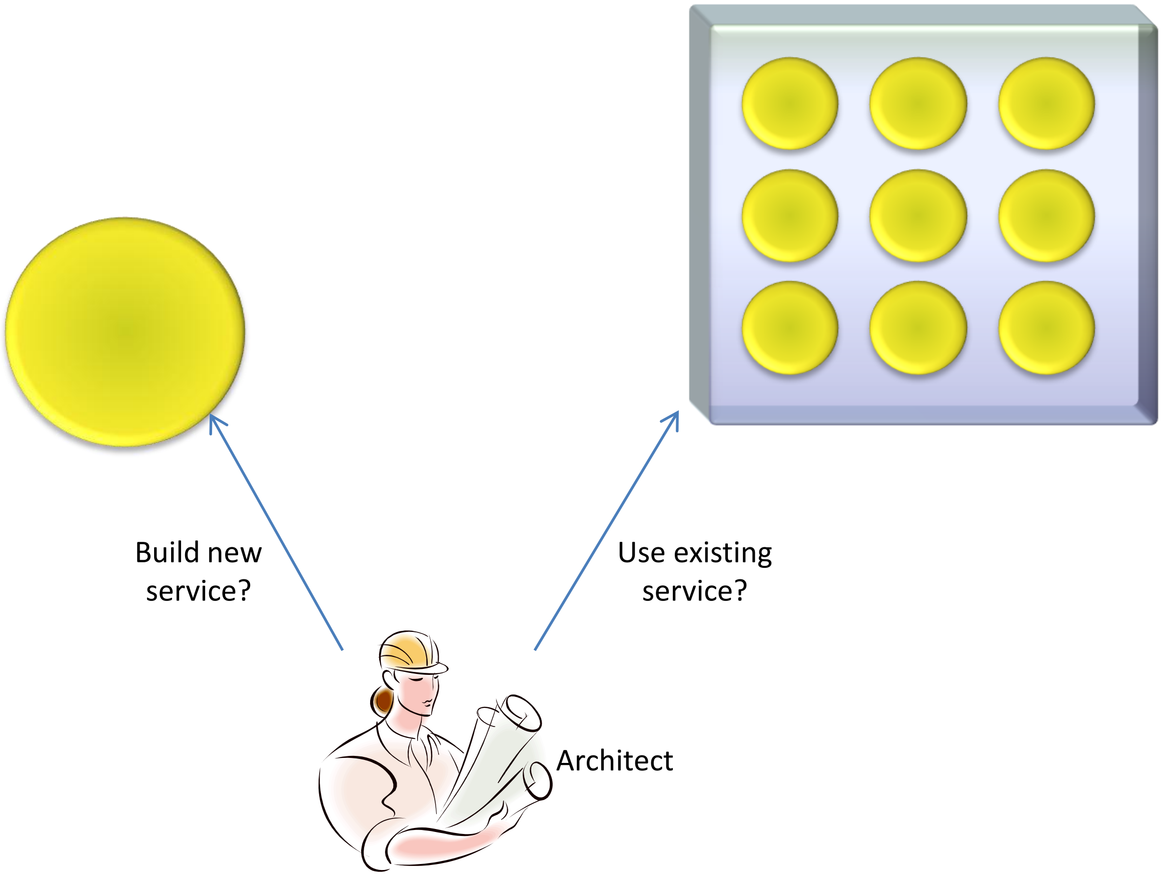
Service Region Influence - Statelessness

The Statelessness principle affects the Service Contract but can also directly influence how the Core Service Logic is designed, right down to the individual programming routines and even the core algorithms that lie beneath each service capability.



Service Role - Discoverability

Discovery helps us determine whether the automation requirements we need to fulfill already exist within a service inventory.

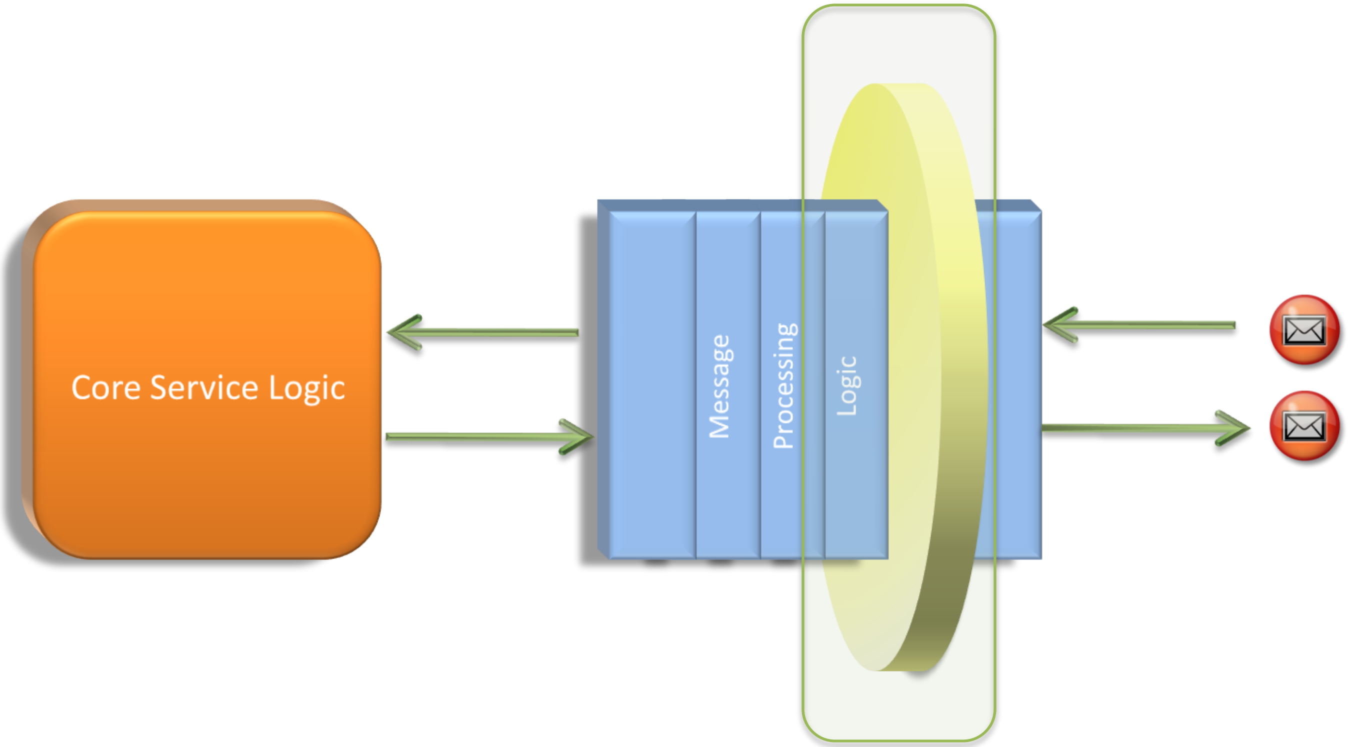


Service Profile - Discoverability

Short Definition	Services are discoverable.
Long Definition	Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted.
Goals	• Services are positioned as highly discoverable resources within the enterprise. • The purpose and capabilities of each service are clearly expressed so that they can be interpreted by humans and software programs. Achieving these goals requires foresight and a solid understanding of the nature of the service itself. Depending on the type of service model being designed, realizing this principle may require both business and technical expertise.
Design Characteristics	• Service contracts are equipped with appropriate meta data that will be correctly referenced when discovery queries are issued. • Service contracts are further outfitted with additional meta information that clearly communicates their purpose and capabilities to humans. • If a service registry exists, registry records are populated with the same attention to meta information as just described. • If a service registry does not exist, service profile documents are authored to supplement the service contract and to form the basis for future registry records.

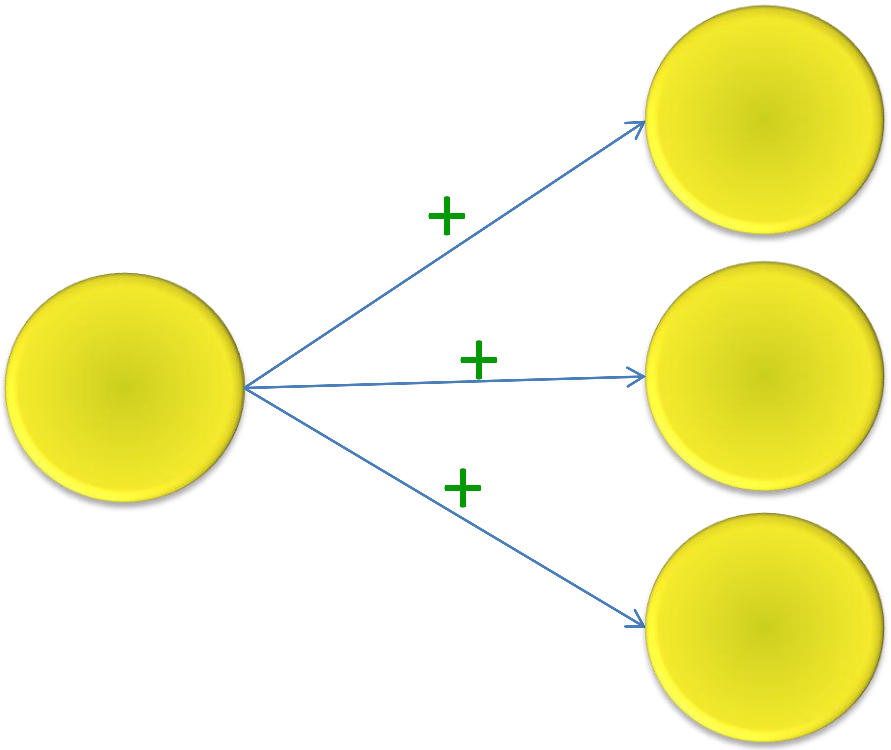
Service Region Influence - Discoverability

The Discoverability principle is focused solely on the Service Contract documents.



Service Role - Composability

Composability introduces design considerations that ensure that services are able to participate in multiple compositions to solve multiple larger problems.



Allow my capabilities to be repeatedly combined with those of other services.

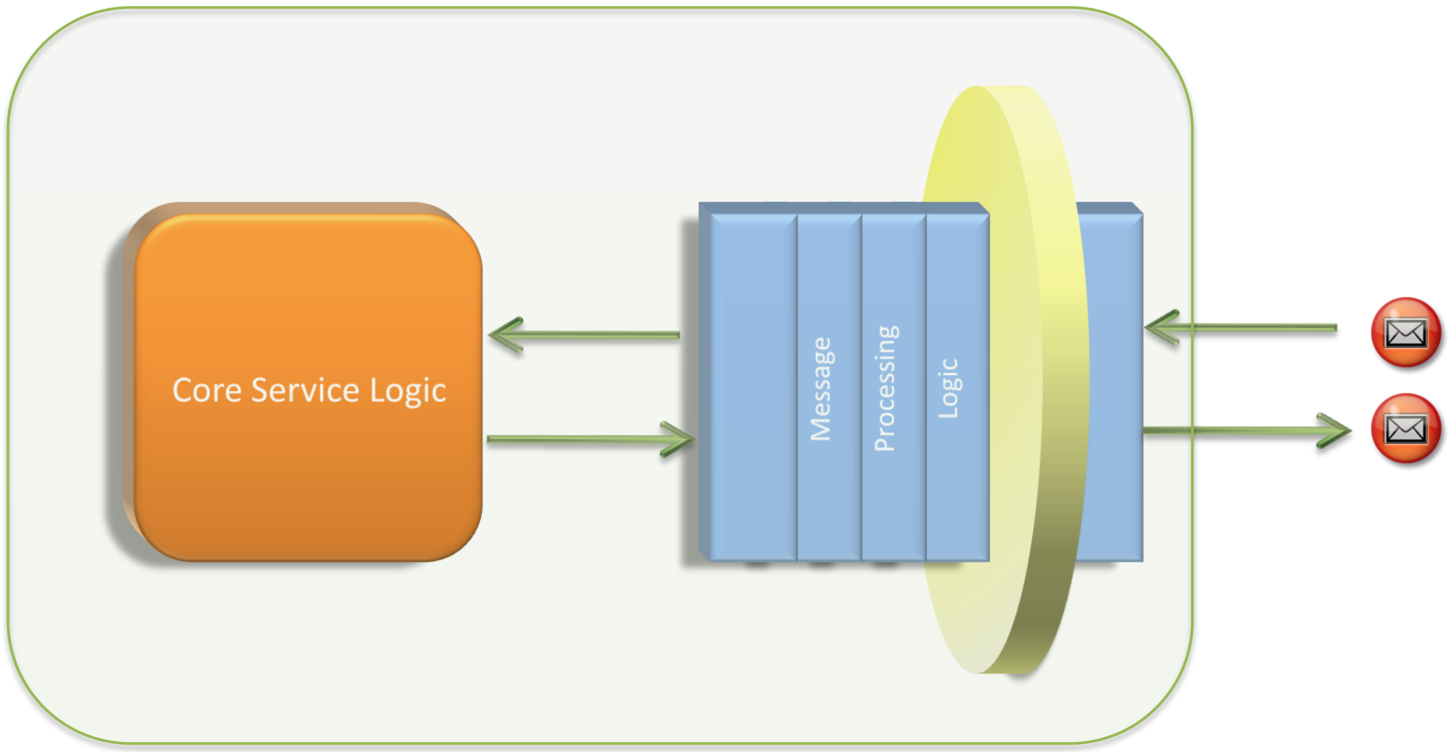
Assembling capabilities from different sources to solve a larger problem is the foundation of distributed computing.

Service Profile - Composability

Short Definition	Services are composable.
Long Definition	Services are effective composition participants, regardless of the size and complexity of the composition.
Goals	Over and beyond simply attaining reuse, service composition provides the medium through which we can achieve what is often classified as the ultimate goal of service-oriented computing. By establishing an enterprise comprised of solution logic represented by an inventory of highly reusable services, we provide the means for a large extent of future business automation requirements to be fulfilled through composition of existing services.
Design Characteristics	In addition to the Service Reusability considerations, the following characteristics are emphasized by this principle: • The service needs to possess a highly efficient execution environment. More so than being able to manage concurrency, the efficiency with which composition members perform their individual processing should be highly tuned. • The service contract needs to be flexible so that it can facilitate different types of data exchange requirements for similar functions. This typically relates to the ability of the contract to exchange the same type of data at different levels of granularity.

Service Region Influence - Composability

The Composability principle can influence all parts of a service primarily because composition builds on reuse and other design characteristics established by supporting principles.


Core Service Logic
Message
Processing
Logic