DZone Refcardz

CONTENTS INCLUDE:

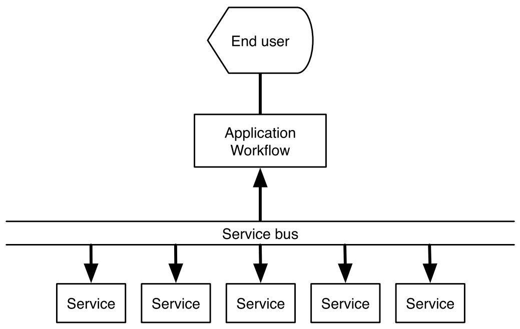
SOA Patterns

By Eugene Ciurana

ABOUT SOA PATTERNS

SOA patterns describe common architectures, implementations, and their areas of application to help in the planning, implementation, deployment, operation, and ongoing management and maintenance of complex systems.

SOA FUNDAMENTALS



Systems are described as services independent of the underlying technology.
Services are implemented through messaging.
A SOA involves service providers and service consumers.
Any participating system may act as either a provider or a consumer depending on the application's workflow.
Services and messages are stateless.
Services and consumers are often implemented in different programming languages, execute in different run-time environments, or both.
SOA involves the services themselves, a directory of available services in some form (service discovery), and public contracts for consumers to connect and use each service (service negotiation).

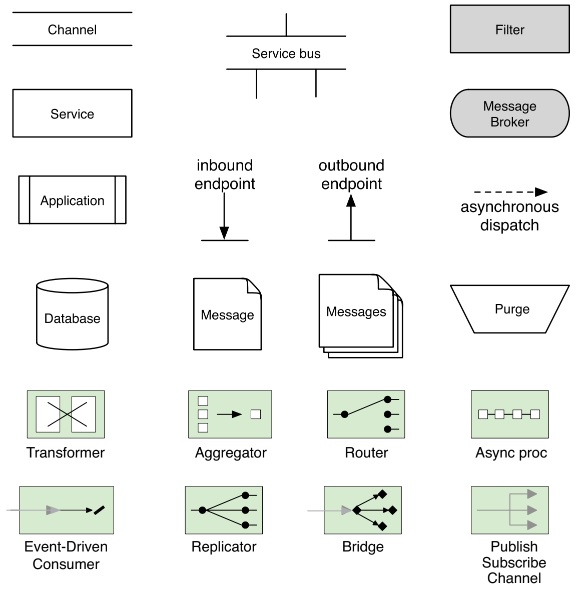
SOA differs from client/server architecture in that services are universally available and stateless, while client/server requires tight coupling among the implementation participants.

SOA implementation must provide consistent designs that leverage the target environment; design consistency is attained through the application of the eight SOA principles. Service must provide:

1.	Normalized service contract.
2.	Loose coupling between consumers and services, and between the services themselves.
3.	Abstraction from implementation details; the consumers only know the contract without worrying about implementation details.
4.	Ability to compose other services regardless of the complexity of the composition.
5.	Run-time environment autonomy.
6.	Statelessness.
7.	Reusability.
8.	Discoverability through meta data or public contract definitions.

These principles guide the SOA patterns described in the rest of this refcard.

PATTERN LANGUAGE



Each pattern includes a Pattern name, Icon, Summary, Problem, Solution, Application, Diagram, Results, and Examples.

The icon and diagram symbols were selected for their ease of whiteboard use and availability in most diagramming tools.

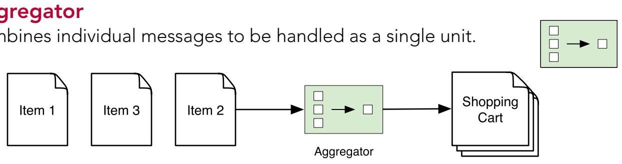
The patterns in this guide are classified into four major groups, and listed in alphabetical order within each group. A complete example appears at the end of this guide showing how to combine various patterns to describe a system.

BASIC SERVICE PATTERNS

These are the building blocks of more complex patterns.

Aggregator

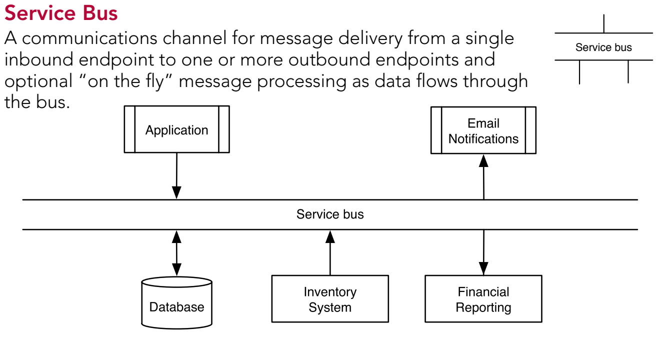
Combines individual messages to be handled as a single unit.



Problem	Stateless messages will not arrive at the service endpoint in a predetermined sequence. Messages may be processed by different services at different speeds and messages will arrive at an endpoint out of order. SOA systems guarantee message delivery but not delivery order.
Solution	Define an aggregator that receives a stream of data and groups related messages as a single entity for delivery to an endpoint for further processing. Aggregators are stateful intermediate processing units but deliver atomic payloads in a stateless manner.
Application	Group messages flowing through a service bus based on payload type or common attributes for further routing and processing.
Results	Flexibility in implementation because individual service providers can process data asynchronously without concern about state or sequence, delegating this to a workflow engine or to aggregators running in the SOA infrastructure.

Service Bus

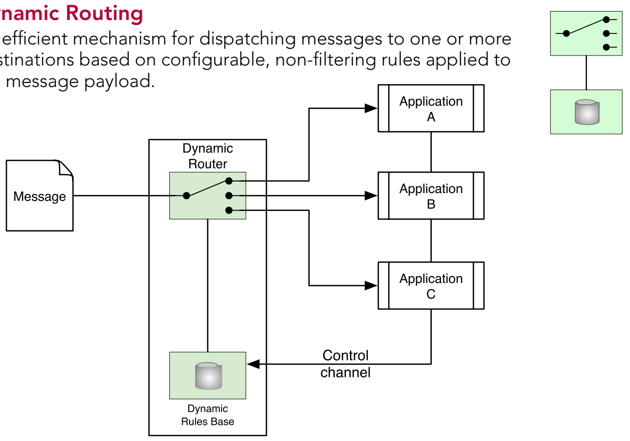
A communications channel for message delivery from a single inbound endpoint to one or more outbound endpoints and optional "on the fly" message processing as data flows through the bus.



Problem	Applications must communicate among them, some times using different protocols and technologies. Naïve implementations rely on point-to-point or hub-and-spoke, dedicated conduits that increase complexity, implementation time, and integration difficulty due to tight coupling between components.
Solution	Provide a data- or protocol-neutral conduit with abstract entry and exit points for interconnecting applications independently of their underlying technology.
Application	Heterogeneous system integration, legacy and new system interoperability, protocol abstraction.
Results	Message-Oriented Middleware (MOM): publish/subscribe queuing and enterprise service buses.

Dynamic Routing

An efficient mechanism for dispatching messages to one or more destinations based on configurable, non-filtering rules applied to the message payload.



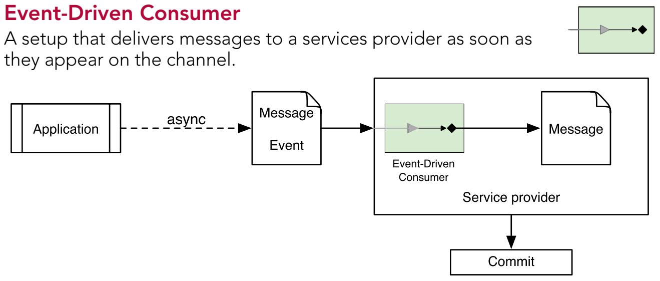
Problem	Routing messages through a distributed system based on filtering rules is inefficient because messages are sent to every destination's filter and router for inspection and rules resolution, whether the message could be processed or not.

Dynamic Routing, continued

Solution	Define a message router that includes both filtering rules and knowledge about the processing destination paths so that messages are delivered only to the processing endpoints that can act upon them. Unlike filters, message routers do not modify the message content and are only concerned with message destination.
Application	Message dispatching based on application-specific data elements such as customer attributes, message type, etc.
Results	Better overall message delivery and processing performance at the cost of increased delivery system complexity since the router must implement both knowledge of the destinations and heuristic, arbitrary rules. Excellent for decoupling applications by removing routing information from discrete systems.

Event-Driven Consumer

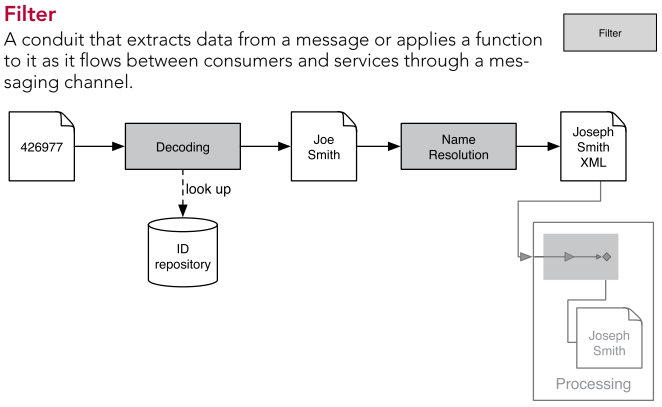
A setup that delivers messages to a services provider as soon as they appear on the channel.



Problem	Messaging systems based on blocking listeners or polling use unnecessary resources or idle for no good reason if the channel is starved. The message target blocks threads that the service could otherwise use for other tasks.
Solution	Implementation of a bus-based or application-specific callback mechanism that's invoked only if a message appears in its inbound channel. The messaging system may invoke the callback asynchronously or synchronously.
Application	Distributed systems with a varying set of consumers and service providers with varying degrees of CPU usage based on message payload; atomic transaction processing systems that require large scalability independent of the number of service consumers.
Results	Message processing is single-threaded scaling linearly with the number of dispatched messages. Threads consume messages as they become available and free up resources when done, to be reactivated when another message becomes available. Better run-time resource utilization.

Filter

A conduit that extracts data from a message or applies a function to it as it flows between consumers and services through a messaging channel.

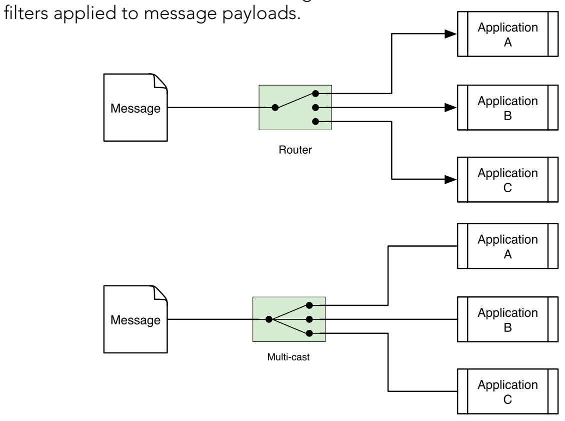


Problem	A need to implement flexible message processing between systems in a platform-independent manner and without introducing system dependencies or unnecessary coupling.
Solution	Implement conduits with a simple inbound/processing/outbound interface modeled after a function or pipe that facilitates composition of daisy-chained filters by allowing data transfers from the output of one filter to the input of the next. All filters, regardless of their internal structure, must share the same external interface to facilitate integration and recombination.
Application	Use of discrete functions on messages like encryption, data consolidation, redundancy elimination, data validation, etc. Filters split larger processing tasks into discrete, easy to manage units that can be recombined for use by multiple service providers.
Results	Filters eliminate data and dependencies by uniform defining a contract (inbound/outbound interface) that encourages reusability through composition. Filters are also interchangeable components that enable different workflow functionality without changing the filter itself.

Basic Service Patterns, continued

Router

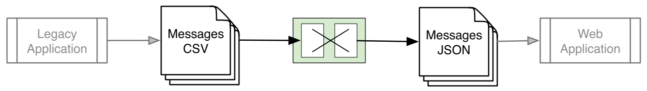
A general mechanism for dispatching messages to one or more destinations based on configurable rules or filters applied to message payloads.



Problem	An application must connect with one or more application endpoints without coupling itself with any of them.
Solution	Use a conduit that allows configurable delivery rules based on the message payload, data filters, or content type. Routing may be sequential (endpoints receive the payload one after another) or in parallel (all endpoints receive the payload at virtually the same time).
Application	Content delivery in service buses, message dispatching, message proxies, enterprise integration applications, and other systems where messages must be delivered to endpoints following a sequence of applying a rule set.
Results	The router abstraction is in use in all modern SOA systems in some forms, whether available in queuing or bus-based systems out of the box, or implemented in custom-made applications and message delivery systems because they provide an elegant and simple mechanism for system-independent message delivery.

Translator or Transformer

A mechanism for converting a message payload from one representation to another as it flows through the messaging system.



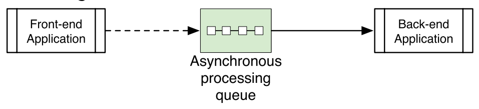
Problem	Heterogeneous systems integration (legacy, in-house, and vendor-provided) may use different message representation for input or output.
Solution	Provide a system-independent mechanism for altering the message payload and metadata (envelope) prior to delivery to an application endpoint.
Application	Message translation at the application endpoint because these translations are system- or protocol-dependent, unlike filters which are generic.
Results	Translators are one of the most effective message transformation mechanisms because they allow application developers and integrators to insulate, implement, test, and maintain these system components without modifying existing application workflow or business logic.

ARCHITECTURAL PATTERNS

Architectural patterns reflect solutions specific to common design issues in the definition of service-oriented system implementations.

Asynchronous Processing

A mechanism for queuing messages between one or more endpoints to decouple processing time and resources for each stage.



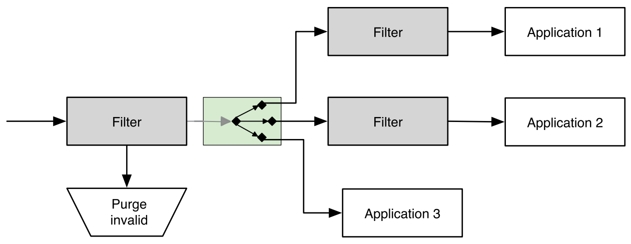
Problem	Synchronous processing may result in poor server performance and reduced reliability.

Architectural Patterns, continued

Solution	Consumers exchange messages with the services through a processing queue that decouples front-end (message capture) from the back-end (processing); messages arrive into the queue at a rate different from that of processing.
Application	Any application that requires independent scalability of the front- and back-end functionality such as mainframe data consolidation (back-end) of e-commerce order fulfillment (front-end, middleware).
Results	Processing queues are well-understood and scale horizontally or vertically, depending on the application requirements. Plenty of open-source and commercial implementations, and several reference implementations and APIs are available.

Bridge

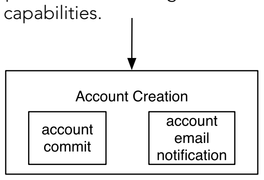
A mechanism for connecting two or more applications over a common data path, each using a different protocol and in which messages may require processing or analysis as they flow between endpoints for routing.



Problem	Application endpoints may reside in different parts of the enterprise network, use different protoocls, or may require processing based on specific message attributes.
Solution	Define a bridge between applications that provides a mechanism for routing messages, filtering them, and transform them.
Application	SOA proxies between application endpoints on the cloud and application endpoints in the middleware or back-end; ESB processing.
Results	Good for extending applications by focusing development only on intermediate processing between system and using existing systems as-they-are. Bridging allows easy integration of legacy and SOA systems.

Cross-Service Operation

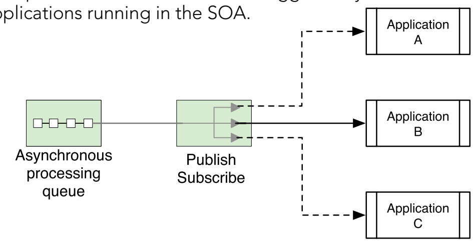
A mechanism for coordinating multiple run-time activities which together comprise a service with guaranteed completion or roll-back capabilities.



Problem	Two or more services, possibly running across multiple systems, must complete successfully; if one or more fail all the services associated with it and the application response must roll-back to their previous state for maximum application integrity.
Solution	Granular services may be wrapped in another service that provides integrity checks and ensures successful completion or graceful degradation, if any, if the granular services fails.
Application	Transactional systems.
Results	May require a transaction processor (commercial, potential vendor lock-in) wrapper to collaborate with the rest of the SOA infrastructure; consumes more resources to preserve original state for each granular service in case roll-back is necessary.

Event-Driven Dispatching

A mechanism for routing messages to consumers in response to specific events in the SOA or triggered by specific applications running in the SOA.

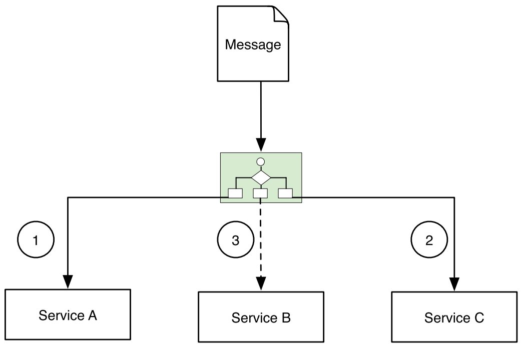


Architectural Patterns, continued

Problem	Consumers must process messages as they become available in a system but polling for such messages is inefficient.
Solution	The consumers are implemented as reentrant, blocking applications that subscribe to a coummuniations channel. The consumers remain dormant until an event or message awakens them; the SOA dispatches the message or event in response to system or application states.
Application	Publish/subscribe systems to support asynchronous processing applications.
Results	Event-driven dispatching is hard to implement in cross-service operations. This pattern is better applied to granular services, or to treat a cross-service operation as a black box by ignoring the intermediate steps involved in the operation.

Process Aggregation

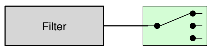
A method of combining two or more non-sequential, inter-dependent processing steps.

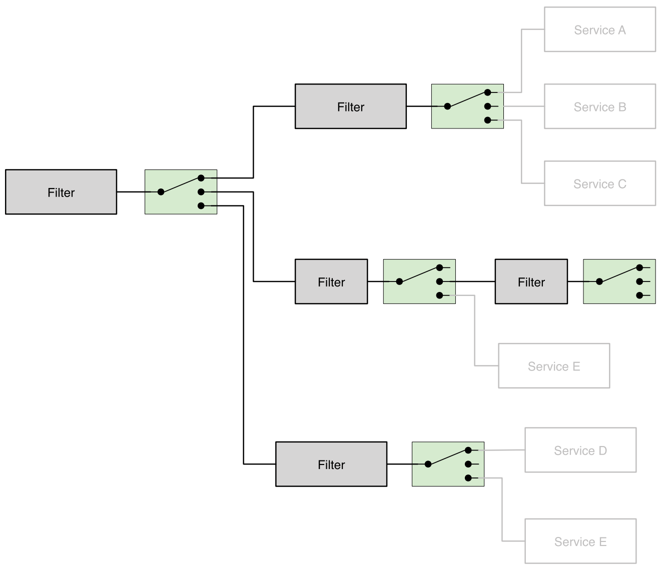


Problem	Multiple services may be required to complete an operation but not all are known at design time, the sequence may vary depending on changing business rules, and it's not necessary to successfully complete all granular processing successfully (i.e. it requires no transactional capability).
Solution	A processing service executes the granular service calls, maintains internal state, determines processing steps, and provides synchronous or asynchronous service responses to the consumers.
Application	Systems that require multiple processes running in parallel but are not transactional, or have a mix of transactional and non-transactional components.
Results	Process aggregation provides integration flexibility but it's hard to manage. Break it down into smaller application clusters (cross-service operations or aggregations) down functional lines, synchronicity requirements, or any other criteria.

Routing and Filtering

A formal mechanism for routing messages to application endpoints between endpoints.





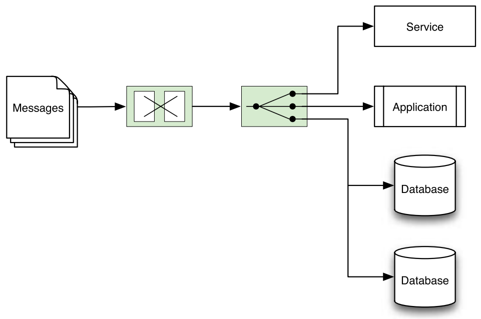
Problem	Messages must be dispatched to various applications based on their payload, attributes, protocol, or all of these.

Architectural Patterns, continued

Solution	Provide a formal mechanism for routing messages by recursive definition of filter, one or more routers, filters, routers, and so on.
Application	Rules-based processing, workflow, event-driven dispatchers.
Results	The recursive nature of the definitions simplifies management. Naïve implementaters some times define filters or routers without formalizing their order, resulting in unintentional application coupling or resource exhaustion due to excessive use of filters or routers, respectively.

Replicator

Messages or payloads must be replicated across multiple endpoints with identical configurations.



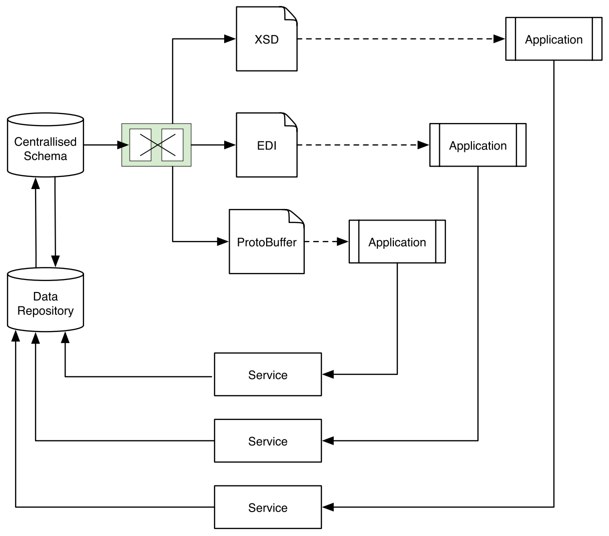
Problem	Decoupled, horizontally scalable services get stuck in a bottleneck caused by shared access to a common message pool or data source.
Solution	Message or data replication features are implemented as part of the SOA message flow so that independent applications or endpoints may consume them in parallel.
Application	Read-only data resources or messages flowing through the SOA to increase throughput.
Results	Additional cost, complexity management if replicators are allowed to proliferate unchecked. Excellent way of providing scalability when the replicators are confined to specific problem domain service paths.

COMPOUND PATTERNS

Compound patterns aggregate the basic patterns to define a cohesive representation of a system. Patterns are never used in isolation, nor are they a goal by themselves. A subsystem may be built around two or more patterns. This section shows how the basic patterns defined earlier in this refcard can be combined into more sophisticated system descriptions.

Centralized Schema

Defines a method of sharing schemas across application boundaries to avoid redundant data representation and service definition.

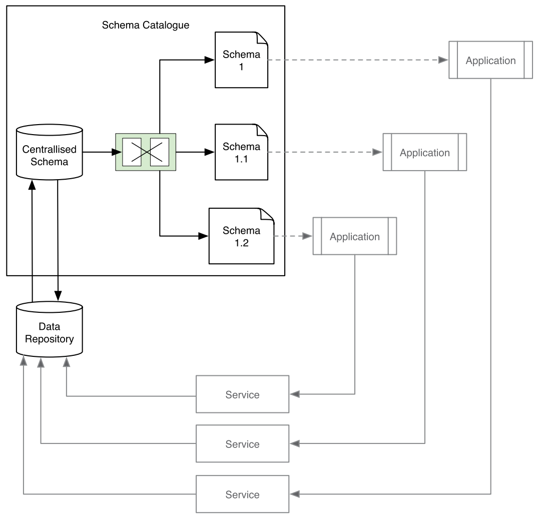


Compound Patterns, continued

Problem	Similar data sets must be processed by services or applications with different capabilities, resulting in unwieldy service contracts or data schemas.
Solution	Define rich data schemas as entities that are separate from the service contracts and from the physical manifestation of the data as it flows through the system.
Application	Any contract-first web services, regardless of implementation technology (JMS, SOAP, other) in which more than one system will transmit, transform, process, or store data.
Results	Easy to implement if the developers make a conscious decision to separate the schema from the services where it's used. A good centralized schema implementation can generate different format definitions that, although incompatible with one another all have a 1:1 mapping to the data model.

Concurrent Contracts

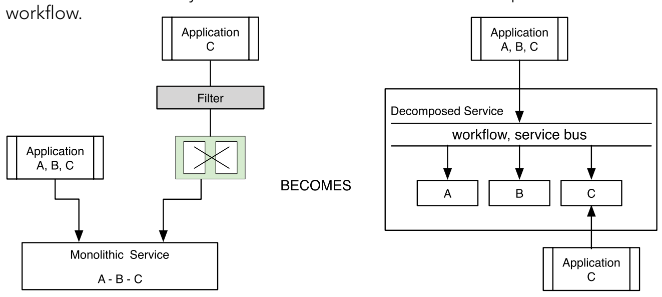
Method for allowing multiple consumers with different abstractions or implementations to simultaneously consume the same service.



Problem	The service contract may not be suitable for all the services potential consumers.
Solution	Multiple contracts may exist for the same service, each with a different level of abstraction than the others in the same group, to fit corresponding service level agreements or to accommodate legacy systems.
Application	Problem domains where various consumers need must process different subsets of the same data, like a customer master or a stock tracking system.
Results	Easy to implement if it's based on a centralized schema and it uses automated transformers or rule-based systems for generating each application contract; it may become unwieldy if the application contracts are manually generated or managed instead of handled by the centralized schema or an automated catalogue.

Decomponse Capability

A way of designing services to reduce the impact of functional deconstruction if it becomes necessary due to bloat or evolution of business processes and workflow.



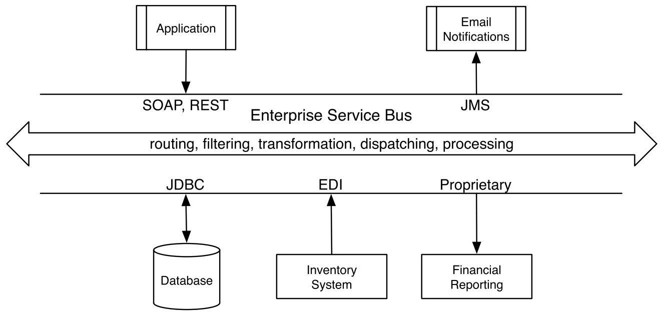
Problem	A service may need decomposition without altering its core functionality, including the service's contract itself.
Solution	Maintain physical separation of the data schemas from the services definition, combining them only for generating specific service implementations, so that data and services may change independently of one another. Define evolutionary service changes in terms of the existing services and basic patterns like filtering, routing, and transformations.

Compound Patterns, continued

Application	Evolution of large, mission-critical systems which provide additional functionality as business requirements are implemented. Any application where incremental features built into a service result in bloat or performance bottlenecks.
Results	Capability decomposition almost always results in the definition of a new service topology that supports the original functionality for legacy or older consumers while providing new functionality or additional features as needed. Decomposition should be transparent to the consumers but lead to modular service design and implementation.

Enterprise Service Bus

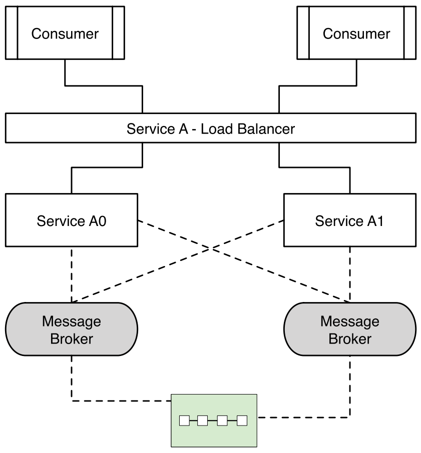
A communications channel for message delivery from a single inbound endpoint to one or more outbound endpoints and provides protocol handling, message filtering, transformation, and routing, and optional "on the fly" message processing.



Problem	Applications must communicate among them, some times using different protocols and technologies. Naïve and legacy implementations rely on point-to-point, dedicated conduits that increase complexity, implementation time, and integration difficulty due to tight coupling between components.
Solution	Provide a data- or protocol-neutral conduit with abstract entry and exit points for interconnecting applications independently of their underlying technology.
Application	Enterprise integration, heterogeneous system integration, legacy and new system interoperability, protocol abstraction.
Results	The emergence of a family of products that implement this concept under the guise of Message-Oriented Middleware (MOM): publish/subscribe queuing and enterprise service buses.

Fault-Tolerant Service Provider

Mechanism for deploying a service platform to achieve near-zero downtime in case one of the services providers or the platform itself have a catastrophic failure.

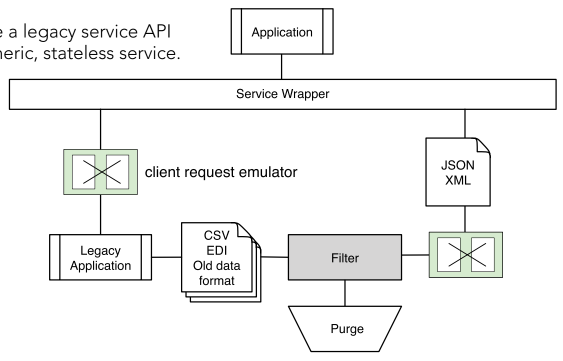


Problem	Mission-critical applications are the main candidates for SOA implementations and must provide appropriate fault-tolerance and recovery in case of catastrophic failure.
Solution	Provide redundant service containers and message brokers complemented by network-level load balancing and routing; ensure that application services are stateless and re-entrant when possible.
Application	Services in fast growing, high-availability environments with near-zero downtime service level agreements.
Results	Easy to implement for stateless services. This pattern may be used for providing both scalability and fault-tolerance.

Compound Patterns, continued

Wrapper

Encapsulate a legacy service API inside a generic, stateless service.



Compound Patterns, continued

Problem	Legacy systems may offer limited service capabilities, or their only interface with other applications may be through file data exchanges or legacy APIs.
Solution	Wrap the interoperation mechanisms within a service façade that operates with the legacy system as if it were a legacy consumer, and exposes a normalized SOA interface to new consumers.
Application	Integration with legacy mainframe or client/server systems to expose their capability to new services and consumers.
Results	Many legacy client/server applications are tightly coupled and even a wrapper may not be enough to expose their capabilities as a service. Extensive rewrites may be required or a service may offer only read-only capabilities. If reimplementation is necessary, then implement as a stateless service and draw a migration plan to phase out the existing legacy service or system.

ABOUT THE AUTHOR

Eugene Ciurana

Eugene Ciurana is an open-source evangelist who specializes in the design and implementation of mission-critical, high-availability large scale systems. As Director of Systems Infrastructure for LeapFrog Enterprises, he and his team designed and built a 100% SOA-based system that enables millions of Internet-ready educational hand held products and services. As chief liaison between Walmart.com Global and the ISD Technology Council, he led the official adoption of Linux and other open-source technologies at Walmart Stores Information Systems Division.

Publications

- *Developing with the Google App Engine*
- *Best Of Breed: Building High Quality Systems, Within Budget, On Time, and Without Nonsense*
- *The Tesla Testament: A Thriller*

Web site

http://eugeneciurana.com

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-33-2
ISBN-10: 1-934238-33-3
50795
9 781934 238332
$7.95

Version 1.0