SERVICE-ORIENTED COMPUTING SERIES FROM THOMAS ERL

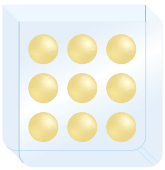
"This obligatory almanac of SOA design patterns will become the foundation upon which many organizations will build successful SOA solutions."
—Stephen Bennett, Director, Technology Business Unit, Oracle

SOA Design Patterns

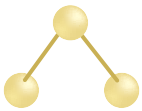
Thomas Erl

Foreword by Grady Booch

With contributions by David Chappell, Jason Hogg, Anish Karmarkar, Mark Little, David Orchard, Thomas Rischbeck, Satadru Roy, Arnaud Simon, Clemens Utschig, Dennis Wisnosky, and others.



service inventory

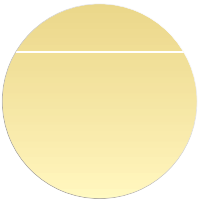


service composition

service (labeled)

services

service layer



service (chorded circle notation)

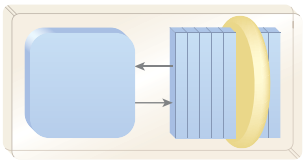
component or program

malicious component or program

decoupled service contract

service agent

firewall



Web service with service contract

component with service contract

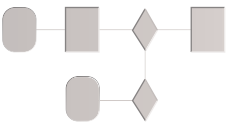
WSDL definition

XML Schema definition

WS-Policy definition

general machine processable document

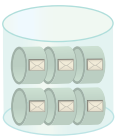
human readable document



business process logic

message

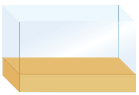
security element or locked resource



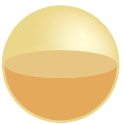
message queue

repository or registry

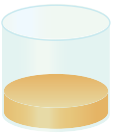
actively processing



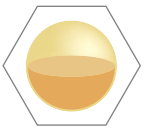
state data in memory



service with state data (stateful service)



repository with state data



grid service

zone or region

conflict symbol

transition arrow

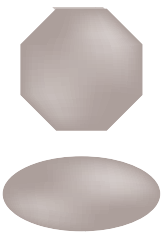
human

client workstation

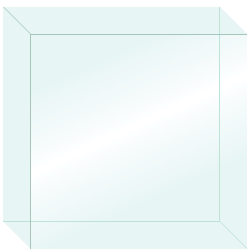
user interface

mobile device

product or system

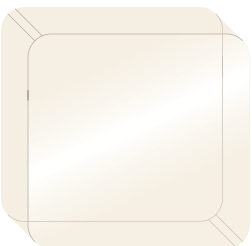


symbols used in conceptual relationship diagrams



general physical boundary

service inventory boundary



service boundary

CASE STUDY EXAMPLE

When finalizing the service contracts described in the Schema Centralization (200) case study example, Cutit architects incorporate a policy that requires that all messages transmitted to any Web service comply to the SOAP 1.2 standard.

The initial approach was to add this policy to each individual Web service contract, as shown in Figure 8.15.

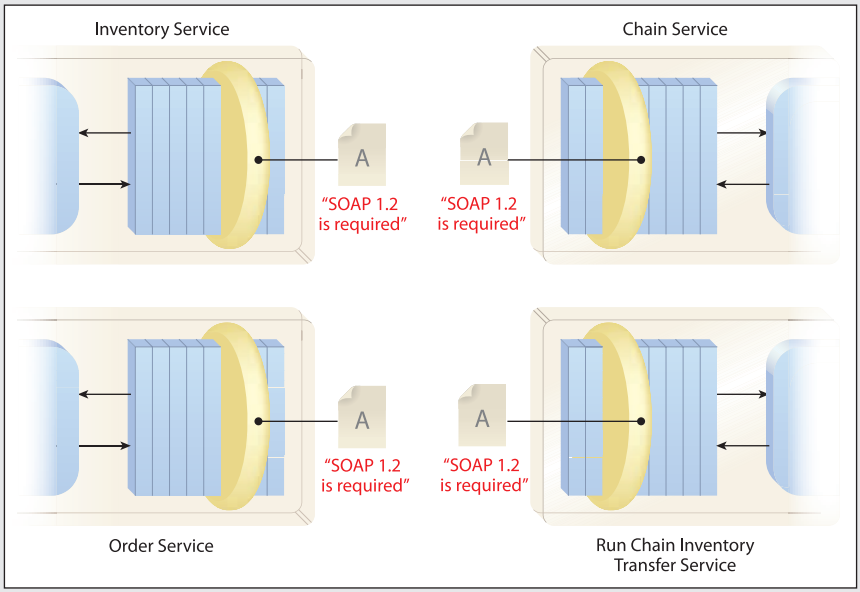


Figure 8.15
The same policy (A) is added redundantly across all Web service contracts.

After the architecture design specification was reviewed, concerns were raised about the redundancy introduced by adding identical policies across multiple contracts. Should the policies ever need to be augmented or removed, it would require a significant governance effort, especially if this approach was taken with all of the services in the Cutit inventory.

Subsequent to some research, a Cutit architect discovers that the middleware product they were considering would allow them to leverage the ability to centralize a policy so that it could be shared across multiple Web service contracts. A prototype is assembled with the architecture illustrated in Figure 8.16, demonstrating a single policy being dynamically applied to multiple Web service contracts.

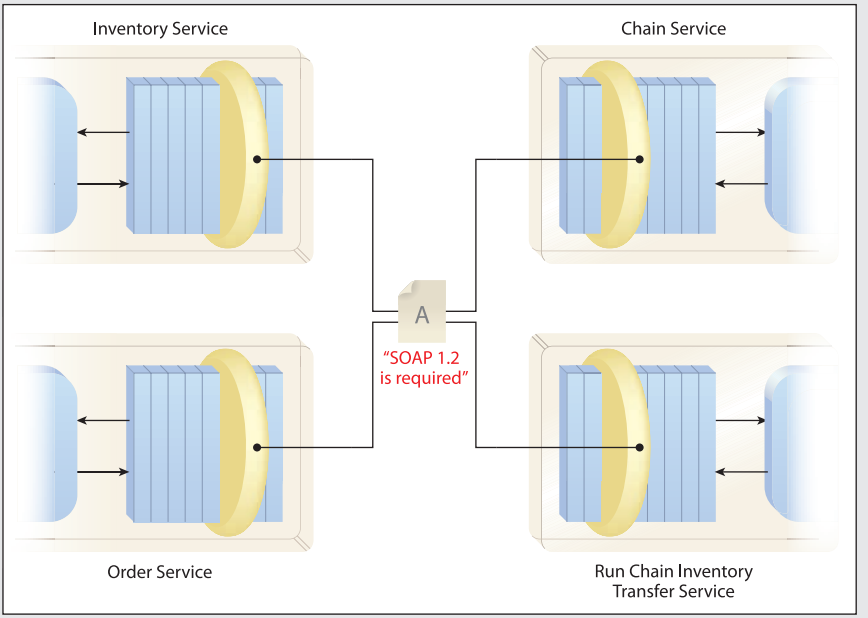


Figure 8.16
A single global policy (A) is established, thereby replacing the redundant policy definitions entirely.

Assuming Policy A in the preceding diagram resides in globalPolicies.xml, the `portType` element in the WSDL definition for the Order service might contain a `wsp:PolicyURIs` attribute, as follows:

```
<definitions targetNamespace=
  "http://cutitsaws.com/contract/order"
  xmlns:tns="http://cutitsaws.com/contract/order"
  ...>
  ...
  <portType name="ptOrder"
    wsp:PolicyURIs="pol:globalPolicies.xml">
    <operation name="SubmitOrder">
      <input message="tns:msgSubmitOrderRequest"/>
      <output message="tns:msgsubmitOrderResponse"/>
    </operation>
    ...
  </portType>
...
</definitions>
```

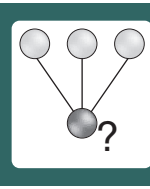
Example 8.2
The Order service contract with an external reference to a global policy definition that is shared by other services.

NOTE

Besides the use of the `wsp:PolicyURIs` attribute, there are several other ways to externally reference and attach policies to WSDL definitions, as explored in Chapter 16 of *Web Service Contract Design and Versioning for SOA.*

Rules Centralization

How can business rules be abstracted and centrally governed?



Problem	The same business rules may apply across different business services, leading to redundancy and governance challenges.
Solution	The storage and management of business rules are positioned within a dedicated architectural extension from where they can be centrally accessed and maintained.
Application	The use of a business rules management system or engine is employed and accessed via system agents or a dedicated service.
Impacts	Services are subjected to increased performance overhead, risk, and architectural dependency.
Principles	Service Reusability
Architecture	Inventory

Table 8.4
Profile summary for the Rules Centralization pattern.

Problem

The workflow logic within any given business process is driven by and structured around rules specific to how the logic must be carried out, as per the policies, regulations, and preferences of the organization. Individual business service capabilities frequently must carry out their encapsulated logic in accordance with these rules.

It is not uncommon for the same rule to be applied to different scenarios involving different business entities. This results in a need to incorporate one rule within multiple bodies of service logic. As an organization changes over time, so do certain business rules. This can lead to modifications within individual entity business services as well as business process logic encapsulated by task services or otherwise (including the occasional utility service). Having to revisit multiple services each time a business rule changes can be counter-productive.

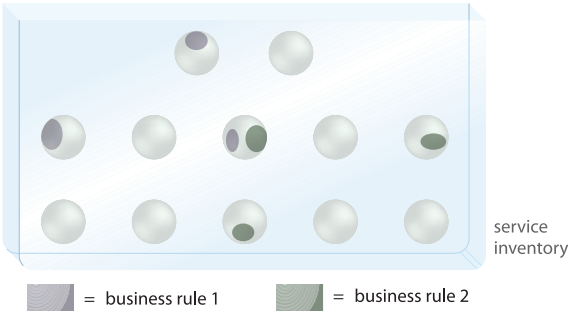


Figure 8.17

Just two business rules can find their way into several different business services and, in this case, even a utility service. A global change to either rule will therefore impact multiple services.

Solution

Business rules can be physically abstracted into a dedicated part of the architecture under the management of specialized rules engines and platforms. This centralizes access to business rule logic and avoids redundancy. It further centralizes the governance of business rules so that they can be modified and evolved from a single location.

Application

Different business rules management systems exist, each introducing a relatively proprietary runtime and administration platform. A central service can be established to provide an official access point for the creation, modification, retrieval, and application of business rules.

Modern runtime platforms also offer native rules repositories and processing logic that is made accessible via a set of system service agents and APIs. This allows any service to interface with business rules-related logic without having to compose a separate service.

NOTE

Centralized rule services are most often classified as members of the utility service layer because they provide generic processing functionality that leverages technology resources and because their functional context is not derived from any organization-specific business models. Even though rule data is business-centric, to the rules service it is just data that it is required to manage and dispense.

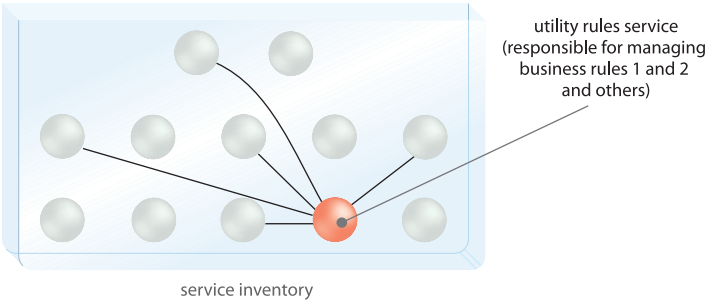


Figure 8.18
All business rules are encapsulated by a single rules service accessed at runtime by other services that need to retrieve or apply business rule logic. (Service agents are also commonly used to provide native access to abstracted rules, as explained shortly.)

Impacts

Because this pattern is applied across an entire service inventory, it can impact an architecture in several ways:

- While it achieves the centralization of business rules data within an inventory, Rules Centralization also ends up *decentralizing* business logic associated with business services. For example, business rules related to the processing of invoices would normally be encapsulated by an Invoice entity service. However, this pattern would move those business rules into a separate location.
- The performance requirements of affected services are increased due to the need for business rules to be retrieved or applied at runtime. Caching mechanisms can alleviate this impact to an extent (usually when rules are temporarily stored as state information for a particular service composition).
- If existing runtime platform features cannot be leveraged to establish centralized rules management, this pattern generally results in the introduction of a separate business rules management product. This extension can increase the size, complexity, and overall operational cost of a technology architecture and must furthermore be sufficiently reliable to consistently accommodate service usage patterns. A rules management system prone to runtime failure can paralyze an entire service inventory.