

WA1471 Service Oriented Architecture for Architects

Introduction to Service Oriented Analysis &
Design (SOAD)

EVALUATION ONLY



Objectives

- At the end of this chapter, participants will be able to:
 - Describe what Service Oriented Analysis and Design (SOAD) is about
 - Describe lessons learned from Object Oriented Analysis and Design (OOAD)
 - Identify two key shortcomings associated with applying OOAD to SOAD
 - Elaborate on the SOAD methodology

EVALUATION ONLY



Introduction to SOAD

- Service Oriented Analysis and Design (SOAD) is a formal methodology for:
 - Gathering requirements.
 - Analyzing the requirements to identify services and business processes.
 - Implementing the services and processes.
- Existing design methodologies such as Object Oriented Analysis and Design (OOAD) provide many best practices that never go out of style. Examples include abstraction, encapsulation, etc.
- Recent experience in developing SOA projects, however, has shown that OOAD alone has shortcomings when designing an SOA-based application. A formal SOA approach is needed.

SOAD is in its infancy at this point. This chapter will provide a coherent approach to analysis and design based on various publications and the author's experience.



Applying OOAD Principles

- Three well established Object Oriented Analysis and Design (OOAD) principles apply equally to Service Oriented Analysis and Design (SOAD):
 - Encapsulation (information hiding)
 - Inheritance (abstraction and hierarchy)
 - Polymorphism (type substitution)

EVALUATION ONLY



Encapsulation



- Encapsulation formally recognizes the need to hide the complex data structure and the business logic of an abstracted entity (class) from its user.
- That means a good class will only expose a set of easy-to-understand methods. (The internal implementation of these methods can be complex).
- A good class also hides data structure complexity.
 - User of the class should invoke methods to manipulate the internal state of an object.

Example of encapsulation

Consider a telephone directory class that employs an array data structure to store the phone numbers. If the search performance is terrible, the data structure can be changed to a HashMap for better performance without affecting the user of this abstraction.



Encapsulation in SOAD

- A good service will provide a high level of encapsulation.
 - It should provide a set of simple operations.
 - The input/output data structure for these operations also should be simple.
- A poorly designed service may have any one of these attributes:
 - Too many operations need to be called to achieve a task.
 - The input/output data structure is too complex.

Example: Acme Industries defines a **Supplier service** as a part of its supply chain. *Good encapsulation* might look like a single operation that accepts a well-defined purchase order data structure. *Poor encapsulation* might look like several calls needed to create an order, or perhaps distinct parameters for each aspect of the purchase order data.

A poorly designed service will have any one of these attributes:

- Too many operations need to be called to achieve a single business goal. For example, to place an order, one has to create the order first and then individually create the order line items. Then the billing and shipping addresses have to be separately added. A well designed service will provide a single operation to create the order.
- The input/output data structure is too complex with too many fields and deeply nested parent-child associations.



Inheritance

- Inheritance defines a relationship between general types and more specific types.
 - A *car* is a general type, a *BMW* is a more specific type
 - The idea is that the more specific types inherit qualities from the more general types (e.g. inheriting the concepts of *engine* or *tires*)
- Another way that the inheritance relationship is described is that it signifies an "is a" or an "is a kind of" relationship
 - A BMW is a kind of car
 - A car is *NOT* a kind of BMW
- In OOAD, inherited attributes are automatically copied into the specialized type
 - Data elements are copied automatically
 - Behavior (functions) are copied automatically

EVALUATION ONLY



Inheritance in SOAD

- The "is a" relationship is vitally important in SOAD.
 - Generalization allows different service implementations to be seen in a uniform manner by the consumer.
 - *For example, services offered by different suppliers of a business can be uniformly accessed by the business.*
- One service can even be an extension of another.
 - Simply have the specialized service call the more general service. In this way, the specialized service can take the results from the general service and add any additional details that are needed.


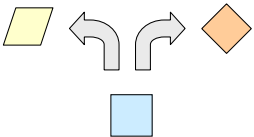
Example: *The PreferredSupplier service could call the base Supplier service to perform most tasks and then adjust the final result with a 10% discount.*

Does WSDL support inheritance?

WSDL syntax does not provide any direct way to extend a service from another service. The extended service must define all operations including the one inherited from the base service. To simplify the extension process, you import the base WSDL file from the extended WSDL file. This way, you don't have to redefine the data types and messages.

How is the "is a" relationship modeled in WSDL if there is no formal inheritance support?

First define the abstract nature of the service using the <portType> element. For example, we can define a <portType> called Supplier. We can define an invocation protocol specific binding (such as SOAP) for service using the <binding> tag. Finally, we specify an actual implementation of a service using the <service> element. The <service> element specifies the endpoint, binding and the abstract <portType>. We can define multiple <service> elements for different suppliers to the organization.



Polymorphism

- Polymorphism comes from two Greek words
 - *Poly* – meaning many
 - *Morphos* – meaning forms
- In OOAD, it means that the same operation can have many forms of behavior according to the type (class) of the object that performs the operation at runtime.
 - Essentially, a generalized type is able to behave differently depending upon what specialized type is substituted for it at runtime. They both support the same interface, but the behavior is different.
 - *For example, if you understand the interface involved in driving a car, then you could substitute a sports car for a minivan and get very different performance results.*

Polymorphism is generally implemented among classes that have a common superclass and that require dynamic or late binding because the method that is invoked may not be identified until runtime.



Polymorphism in SOAD

- At its core, polymorphism is about the ability to support a common interface and then determine the interface implementation at runtime
 - This is referred to as 'late binding' because the decision is made after compile/build time
 - At design time, the key is to craft an interface that is sufficiently abstract to support multiple specialized implementations
 - At run time, a means to support polymorphic calls must exist
- Service oriented systems support this in various ways
 - Middleware-based routing
 - Service endpoint addressing
 - Dynamic endpoint lookups

Example: All clients could code against the standard Supplier service. Business rules could then dictate when requests would be routed to the PreferredSupplier service and thus apply the discount.



Why OOAD Is Not Enough

- As we have just seen, most of the best practices of OOAD also apply in some way to SOAD.
- The problems with OOAD for SOA are twofold:
 - Level of granularity
 - Tight coupling

Level of granularity

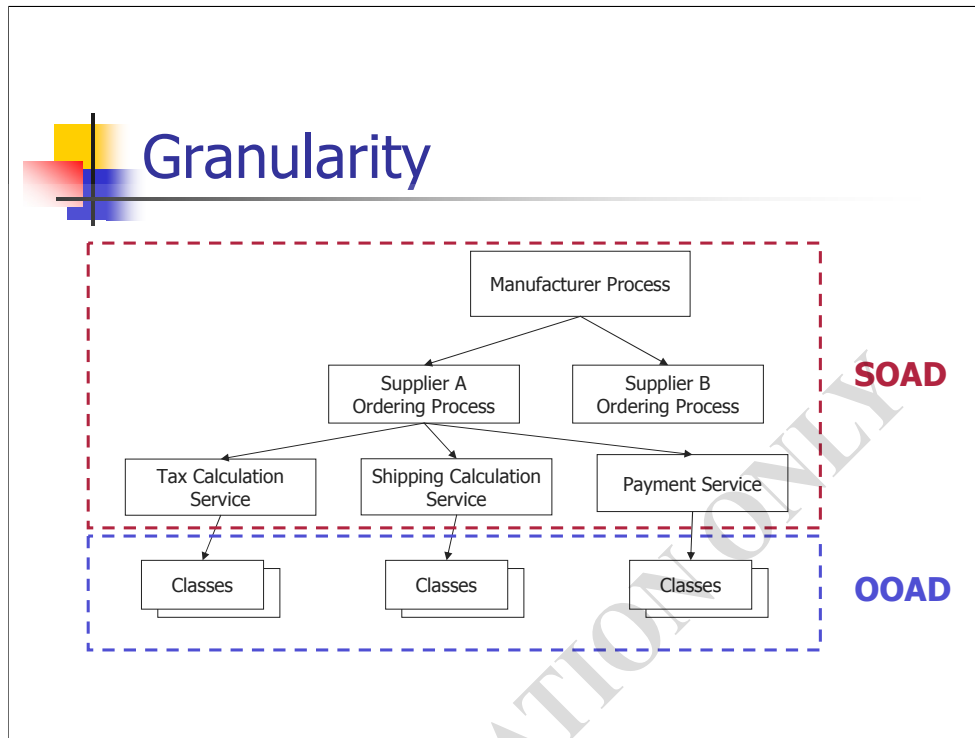
OOAD deals with much finer grained abstractions, such as classes.

Tight coupling

Classes are tightly coupled with each other. In many programming languages, it is difficult to change the implementation of one class without having to recompile all related classes (related using the hierarchy principle) and the application. The Factory pattern is often suggested to introduce loose coupling in OOAD. EJB goes a long way towards providing loose coupling.

More on tight coupling

Java allows modification of a class without having to recompile related classes or the entire application. But, it still does not solve the problem of tight coupling. Let us say that you have purchased Java accounting software. You want to change the implementation code of the TaxCalculator class. How can you do that without having access to the source code? Use of factory pattern solves this problem.



Business processes are the most coarse-grained artifacts in SOAD. A process can involve other processes as shown in the diagram above. A process is made up of various services. A service is implemented using traditional OOAD techniques. For example, it involves fine-grained classes.



The Need for Loose Coupling

- Loose coupling allows a service to be accessed using any arbitrary document format, regardless of syntax, provided that the physical business data is present within (or can be determined from) the document contents.
- Some incompatibility issues addressed by loose coupling are:
 - *Filtering*: Reject input that does not meet certain requirements e.g. parameter bounds.
 - *Renaming*: Fix clashes in nomenclature e.g. 'first_name' and 'FirstName'.
 - *Completion*: Fill out missing fields using sensible defaults.
 - And more...
- The service bus infrastructure allows us to map data.
- Loose coupling can also cover protocol mismatch. The service bus can perform needed protocol translation.

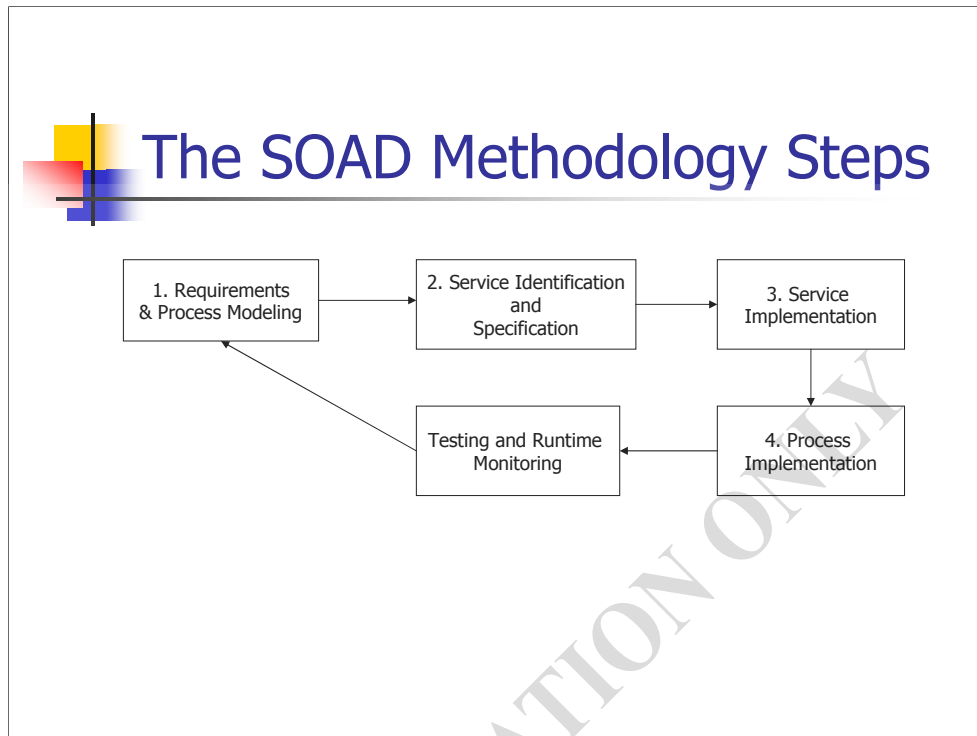
EVALUATION ONLY



The SOAD Methodology

- OOAD methodology is covered by the Rational Unified Process (RUP). A similar approach can be used with SOAD.
- Like RUP, SOAD also needs an approach that emphasizes iterative development. The following steps are performed sequentially—and repeated as many times as needed.
 1. Requirements Gathering and Process Modeling
 2. Service Identification and Specification
 3. Service Implementation
 4. Process Implementation
- Using an iterative development approach minimizes risk. Fundamental design flaws surface early on.

EVALUATION ONLY





Stage 1: Requirements Gathering & Process Modeling

- The initial emphasis in SOAD is the business process. (In OOAD, the focus is on use cases.)
- The first step is to describe the business process in a plain text document. It is a good practice to follow the use case format, which is well established and lends itself to the capturing of flows.
 - Someone with good domain knowledge, such as a business analyst or a manager, will create this initial document.
 - The business process document captures the interaction between businesses or different software systems in a business.

By business process we usually mean the automated business processes, or business processes that involve tasks that are automatically performed by software systems. It is acceptable that a small number of tasks are performed manually, such as entering data from one system to another system or placing an order with a supplier by sending a fax. Such manual tasks increase the cost of running a business. If a process involves many manual tasks, it may not be well suited for an SOA-based solution. SOA is, after all, interaction between different software systems.



Stage 1: Requirements Gathering & Process Modeling

- Next, create a business process diagram.
 - You are encouraged to use a tool that can save the process diagram as a BPEL document.
 - BPEL is an industry standard, but no standard exists at this point for graphic notations of various activities carried out by a process. Each vendor (such as IBM and BEA) uses different notations.
- Run simulations to estimate the time and cost of running the process.
- The following deliverables are produced by the end of this stage:
 - The Business Process document
 - Business Process diagram
 - A BPEL document for the Business Process diagram

Depending on the tool you use, it will either save the process diagram natively as a BPEL file or allow you to export the diagram as a BPEL document. The advantage of using BPEL is that it is an industry standard. You can send the file to others who can open it using a different tool.



Stage 2: Service Identification

- In stage 2, business process activities are mapped to service operations and key service artifacts are produced.
- This stage has two primary steps:
 - In the first half of this stage, business analysts identify logical services in support of the process and create or modify service contracts.
 - In the second half of this stage, architects or designers specify service interfaces (WSDL) in support of the service contracts identified by the analysts.
- Deliverable artifacts from this stage include:
 - New/updated *service contracts*
 - New/updated *service interfaces* (WSDL)

A process itself should contain minimal business logic. All substantial business logic, such as calculating taxes and shipping, should be performed by a service. The business process should invoke various operations of these services.

Some of the services will be built using a bottom-up approach—by wrapping existing applications. Re-use through bottom-up development is a primary goal at this stage.



Stage 3: Service Implementation

- Select a specific service from the previous step.
- Bottom-Up (used when functionality already exists)
 - Model data mapping of input and output data
 - Document protocol translation if required
 - Design a simple wrapper Web service if needed
- Top-Down (used when functionality must be developed)
 - Develop use cases describing how the operations will be implemented
 - Class identification
 - Deliverables: Class diagram and Object interaction diagram
- Implement the service and begin to unit test it.
- Deliverables from this stage include:
 - Service implementations
 - Potentially supporting UML diagrams (Class and/or Object)

Top-Down Development

A service can be implemented using the standard OOAD techniques, which means first developing a use case for each operation supported by the service, and next identifying classes. Produce class diagrams and object interaction diagrams.

Keep in mind that the design and business logic of a service are of no concern to the business process.

Bottom-Up

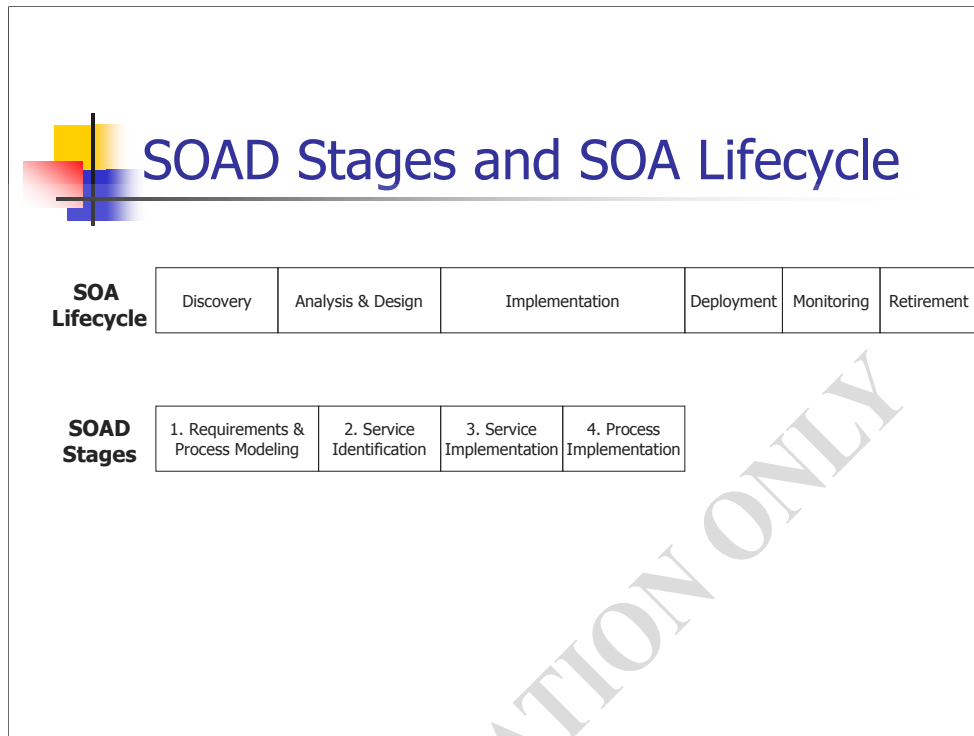
An application may already provide the functionalities offered by a service. All you have to do here is build a wrapper Web service. In this case, you will need to work on data mapping. That is mapping XML data to whatever the existing application can work with.



Stage 4: Process Implementation

- Finally, developers complete the process implementation:
 - Each activity is implemented as a service operation invocation. In the BPEL diagram, specify the necessary linkage information (such as operation name and service end point information).
 - For services that only have a data-oriented interface, take necessary action to export data out of the process. (For example, publish a message in a JMS queue.)
 - Provide data mapping when the invoked services expect data in a different format than what the process works with.
 - Execute the process in a unit testing environment.
- Deliverables from this stage include:
 - Completed BPEL and deployment module
 - Updated process documentation

EVALUATION ONLY



The SOAD stages are really a further sub-division of the SOA lifecycle phases.

- 1) SOAD Stage #1 Requirements gathering and Process Modeling span the Discovery phase and the Analysis and Design phase of SOA lifecycle. In this stage, use cases are developed and the business processes are modeled.
- 2) SOAD Stage #2 Service Identification falls entirely within the Analysis and Design phase of SOA.
- 3) The Implementation phase of SOA lifecycle is broken up in two stages of SOAD. Stage #3 deals with service implementation. Stage #4 deals with the business process implementation.



Summary

- The best practices of OOAD continue to apply in SOAD:
 - Abstraction
 - Encapsulation
 - Modularity
 - Hierarchy
- The SOAD Process consists of four stages:
 - Process Modeling
 - Service Identification and Specification
 - Service Design and Implementation
 - Process Implementation

EVALUATION ONLY