Service Oriented Architecture

Numan Sheikh
Assistant Professor
GIFT University
What is a Service?

- Lego block. Building Block
- Has standard based interface
- Loosely coupled
- Platform independent
- Composable
- Discoverable
- Vertical slicing
What is a Service?

- A service is a reusable component that can be used as a building block to form larger, more complex business-application functionality.

- A service provides a **discrete business function** that **operates on data**.

- A service may be as simple as **get me some person data**, or as complex as **process a disbursement**.

What is a Service?

- Its job is to ensure that the business functionality is applied **consistently**, returns **predictable results**, and operates within the quality of service required.

- From a theory point of view, it really doesn’t matter how a service is implemented.
What is a Service?

• **Stateless**

  – The service does not maintain state between invocations. It takes the parameters provided, performs the defined function, and returns the expected result. If a transaction is involved, the transaction is committed and the data is saved to the database.

What is a Service?

• Only the service knows how it is implemented

• A requestor only knows
  – what the service is and
  – how to request it

• The consumer of the service is required to provide only the stated data on the interface definition, and to expect only the specified results on the interface definition.

• The service is capable of handling all processing (including exception processing).
Service Oriented Architecture

DEFINITION

Service Oriented Architecture

• There are three concepts that can be used by SOA
  – Architectural Concept
  – Business Solution Styles
  – Supporting Infrastructure/Environment
What is SOA?

• SOA is **Legoware**.
• Made of Lego Blocks

What is SOA?

• SOA is an architecture that a software system is divide into high flexible and re-usable component unit called service and compose with interconnection of these services.

• Applications built using an SOA style deliver functionality as services that can be used or reused when building applications or integrating within the enterprise or trading partners.

• The general concept of service in SOA is that service is a major unit of software component with business sense. These services have low dependency and each can be implemented without platform dependency while the interfaces that the service provides must be standardized.
SOA is NOT

– Technical standard

– A technology

It is

– Just architecture blue print

Architecture

• Architecture implies a consistent and coherent design approach. Essential principles include:

  – **Consistency**: The same challenges should be addressed in a uniform way.

  – **Reliability**: The structures created must be fit to purpose and meet the demands for which they are designed.

  – **Extensibility**: A design must provide a framework that can be expanded in ways both foreseen and unforeseen.

  – **Scalability**: The implementation must be capable of being scaled to accommodate increasing load by adding hardware to the solution.
Service Oriented Architecture: Architectural Concept

• **Stateless**

• **Loose coupling** between a
  – **service** (a stateless, self-contained business function) and its
  – **provider** (the physical assets that perform the function)

Service Oriented Architecture: Architectural Concept

• Only the service knows how it is implemented

• A requestor only knows
  – what the service is and
  – how to request it

• The consumer of the service is required to provide only the stated data on the interface definition, and to expect only the specified results on the interface definition.

• The service is capable of handling all processing (including exception processing).
Service Oriented Architecture: Business Solution Styles

• Composite Application Development
  – User interaction drives the request for one or more services
  – Requests are usually synchronous in nature
  – Delivered via portal

• Flow Development
  – Business Process-Driven Architecture
    • Workflow: a series of activities/functions completes a business transaction or process
    • Typically long-running/asynchronous
  – Event-Driven
    • An inside or outside event triggers other interested applications

Service Oriented Architecture: Infrastructure / Environment

• An SOA environment allows services to be defined, developed and used by other services

• These services can be assembled into solutions through business rules, interaction mechanisms and user interfaces

• Enables service discovery, policy definition and enforcement, quality of service, transaction management, audit and usage metering
Service Oriented Architecture

WHY?

Current Environment

"That reminds me – I have to sort out the business services on the corporate network."
Application Centric

Business functionality is duplicated in each application that requires it.

Current Environment

Narrow Consumers
Limited Business Processes

Integration
Architecture

Overlapped resources
Overlapped providers

EAI ‘leverage’ application silos with the drawback of data and function redundancy.

Service Centric

SOA structures the business and its systems as a set of capabilities that are offered as Services, organized into a Service Architecture.

Goal

Multiple Service Consumers
Multiple Business Processes

Shared Services

Multiple Discrete Resources
Multiple Service Providers

Service virtualizes how that capability is performed, and where and by whom the resources are provided, enabling multiple providers and consumers to participate together in shared business activities.

source: TietoEnator AB, Kurts Bildner
### Before SOA – After SOA

<table>
<thead>
<tr>
<th>Before SOA</th>
<th>After SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Situated · Closed · Monolithic · Brittle</td>
<td>Shared · Collaborative · Interoperable · Integrated</td>
</tr>
</tbody>
</table>

#### Application Dependent Business Functions

- **Before SOA**
  - Order Processing
  - Check Customer Status
  - Determine Product Availability
  - Verify Customer Credit
  - Order Status

- **After SOA**
  - Composite Application
  - Order Processing
  - Account Management
  - Composed Business Process

#### Reusable Business Services

- **Before SOA**
  - Reusable Service
  - Reusable Service

- **After SOA**
  - Reusable Service
  - Reusable Service

#### Why SOA?

**To enable Flexible, Federated Business Processes**

- Enabling a virtual federation of participants to collaborate in an end-to-end business process
- Enabling alternative implementations
- Enabling reuse of Services
- Enabling virtualization of business resources
- Enabling aggregation from multiple providers

### Source

- IBM
- TietoEnator AB, Kurts Bilder
Why SOA? To enable Business Process Optimization and the Real Time Enterprise (RTE)

Seamless End to End Process

BPM Expressed in terms of Services Provided/Consumed

SOA Patterns: Single, Multi-Channel Service for consistency

Service to Customers

Smart Clients
Stores POS
Mobile
3rd Party Agents
Portal

Service from Multiple Suppliers

Source: TietoEnator AB, Kurts Bilder

Why SOA?
Enable Structural Improvement

ERP X Process Z Partner A Process Y

Standardizing capabilities Information Consistency Policy Consistency

e.g. Single Customer Details Service

Reducing impact of change Consolidation/Selection process Encapsulating implementation complexity

Consolidation/Selection process

ERP Z CRM System 2 CRM System 1 Product System

e.g. Multiple Sources of Customer Details

Policy Rationalization and Evolution Resource Virtualization
Why Loose Coupling?

- A loosely-coupled architecture is seen as helping to reduce overall complexity and dependencies.

SOA Defined

- SOA is a software architecture model
  - in which business functionality are logically grouped and encapsulated into
    - self contained,
    - distinct and reusable units called services that
      - represent a high level business concept
      - can be distributed over a network
      - can be reused to create new business applications
      - contain contract with specification of the purpose, functionality, interfaces (coarse grained), constraints, usage
    » ... of the business functionality
What is Service Architecture?

- A collection of services
- classified into types
- arranged into layers
- Governed by architectural patterns and policies

Big (outer) vs. Little (inner) SOA
SOA is an evolutionary step for architecture

SOA is an evolutionary step in reusability and communication
SOA is an evolutionary step in distributed communications

Service Architecture Organized by Layers

Reasons for Layering

1. Flexible composition.
2. Reuse.
3. Functional standardization in lower levels
4. Customization in higher layers
5. Separation of Concerns.
6. Policies may vary by Layer

Example Layers

- Presentation & workflow
- Composed Services
- Basic Services
- Underlying API

according to: TietoEnator AB, Kurts Bilder
Major service types

- **Basic Services:**
  - Data-centric and logic-centric services
  - Encapsulate data behavior and data model and ensures data consistency (only on one backend).
  - Basic services are stateless services with high degree of reusability.
  - Represent fundamental SOA maturity level and usually are build on top existing legacy API (underlying services)

- **Composed Services:**
  - Expose harmonized access to inconsistent basic services technology (gateways, adapters, façades, and functionality-adding services).
  - Encapsulate business specific workflows or orchestrated services.
SOA Principles

• Standardized Service Contracts
• Loose Coupling
• Abstraction
• Reusability
• Autonomy
• Statelessness
• Discoverability
• Composability

Standardized Service Contracts

• “Services within the same service inventory are in compliance with the same contract design standards.”
• Services use service contract to
  – Express their purpose
  – Express their capabilities
• Use formal, standardized service contracts
• Focus on the areas of
  – Functional expression
  – Data representation
  – Policy

Source: Thomas Erl
Loose Coupling

- “Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environment.”
- Create specific types of relationships within and outside of service boundaries with a constant emphasis on reducing (“loosening”) dependencies between
  - Service contract
  - Service implementation
  - Service consumers

Source: Thomas Erl

Abstraction

- “Service contracts only contain essential information and information about services is limited to what is published in service contracts”
- Avoid the proliferation of unnecessary service information, meta-data.
- Hide as much of the underlying details of a service as possible.
  - Enables and preserves the loosely coupled relationships
  - Plays a significant role in the positioning and design of service compositions

Source: Thomas Erl
Reusability

• “Services contain and express agnostic logic and can be positioned as reusable enterprise resources.”

• Reusable services have the following characteristics:
  – Defined by an agnostic functional context
  – Logic is highly generic
  – Has a generic and extensible contract
  – Can be accessed concurrently

Source: Thomas Erl

Autonomy

• "Services exercise a high level of control over their underlying runtime execution environment."

• Represents the ability of a service to carry out its logic independently of outside influences

• To achieve this, services

• Primary benefits
  – Increased reliability
  – Behavioral predictability

Source: Thomas Erl
Statelessness

- "Services minimize resource consumption by deferring the management of state information when necessary."
- Incorporate state management deferral extensions within a service design
- Goals
  - Increase service scalability
  - Support design of agnostic logic and improve service reuse

Source: Thomas Erl

Discoverability

- "Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted."
- Service contracts contain appropriate meta data for discovery which also communicates purpose and capabilities to humans
- Store meta data in a service registry or profile documents

Source: Thomas Erl
Composability

• "Services are effective composition participants, regardless of the size and complexity of the composition."
• Ensures services are able to participate in multiple compositions to solve multiple larger problems
• Related to Reusability principle
• Service execution should efficient in that individual processing should be highly tuned
• Flexible service contracts to allow different types of data exchange requirements for similar functions

Source: Thomas Erl

SOA – Best Practices

• Encapsulate what Varies
  – A service that can change over time, or that could be ‘ripped and replaced,’ should have an interface defined that remains constant

• Program to Interfaces, not Implementations
  – Services should provide interfaces that other services can call, rather than make direct calls to underlying component APIs. For example, it is better to make calls to a JMS implementation rather than direct MOM API requests.

• Depend on Abstractions, not Concrete Services
  – Create a StockQuote interface that includes generic fields required for a stock quote. Don’t allow your service to make direct calls to an external vendor like XMethods.
SOA – Best Practices

• **Strive for Loosely-Coupled Designs Between Services that Interact**
  – Today’s enterprise service provided by SAP could be provided by Oracle tomorrow.
  – Define what is needed for billing information in an interface, and maintain the interface, even if the implementation changes behind it.

• **Services should be Open for Extension, but Closed for Modification**
  – Look at adding functionality to a service through external means, rather than constantly changing the inner code.

SOA – Best Practices

• **Favor Composition over Inheritance**
  – Instead of changing an existing service, create a façade interface that calls multiple low-level services

• **Principle of Least Knowledge (Talk Only to Your Immediate Friends)**
  – When you design a service, limit the number of low-level services it interacts with, and the type of data sent to these services.
  – Limit the coupling of overall systems.
  – When dealing with sensitive information, pass only the data the other system needs, don’t just send all of the state fields if the other system won’t use them.
Applying SOA - Governance

- Governance is a program that makes sure people do what is ‘right’

- In conjunction with software, governance controls the development and operation of software

  - Goal: Establish SOA organization governance (SOA Board) that governs SOA efforts and breaks down capabilities into non-overlapping services

Applying SOA - Governance

- **Policies**
  - Codification of laws, regulations, corporate guidelines and best practices
  - Must address all stages of the service lifecycle (technology selection, design, development practices, configuration management, release management, runtime management, etc.)

- **Processes**
  - Enforce policies
  - System-driven processes (code check-in, code builds, unit tests)
  - Human-driven process (requests, design reviews, code reviews, threat assessment, test case review, release engineering, service registration, etc.)

- **Metrics**
  - Measurements of service reuse, compliance with policy, etc.
  - Organization
  - Governance program should be run by SOA Board, which should have cross-functional representatives
Applying SOA – Governance

Applying SOA - Challenges

• Service Orientation
• Reuse
• Sharing of Responsibilities
• Increased complexity!

Business functionality has to be made available as services. Service contracts must be fixed.

Implemented services must be designed with reuse in mind. This creates some overhead.

Potential service users must be involved in the design process and will have influence on the service design.
Applying SOA – Renovation Roadmap

Conclusion and Summary

• If done correct, SOA is “not just another architectural fad”

• SOA seeks to bridge the gap between business and technology promoting business agility (its all about managing change)

• SOA
  – Is complex
  – Requires governance
  – Requires executive management buy-in
  – Requires commitment with resources (people and $$)