

# Service Oriented Architecture

Version 9

Making Software Work Together™



ICNN

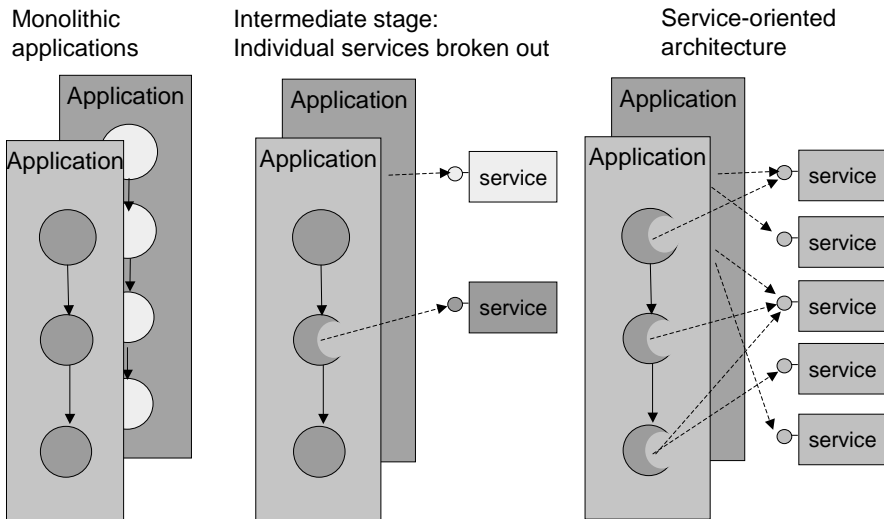
SOA-2

## Overview

*Ok, now we understand the Web Service technology, but how about Service Oriented Architectures?*

- A guiding analogy
- Terminology excursion
  - Service, interface, architecture model, architecture style, service-oriented architecture
- Benefits of service-oriented design
- Successful example of service-oriented design
- What does this mean to you?
  - How to prepare for SOA
- Summary

## From just using a Web Service to having a service-oriented architecture



A guiding example

## Bank Teller analogy for service-orientation

- Different types of tellers offer different services
  - Tellers specialized to perform only certain types of transactions (which are typically closely related)
  - Example partitioning:
    - Account Management (Opening and closing accounts)
    - Credits (inquiry about conditions, consulting, applying for mortgages)
    - Cash Register (Withdrawals, deposits, funds transfers)
    - Currency exchange (buy and sell foreign currencies)
- There may be several tellers offering the same set of services (for load balancing / failover)
- What happens behind the counter is not your business (bulletproof glass, iron bars)
- If you require a complex transaction, you may have to visit several tellers (customer as transaction coordinator)

## Terms and Definitions

**Service (Definition)**

- A **service** is a package of closely related standardized functions, which are called repeatedly in a similar fashion, and should therefore be implemented by a dedicated facility, which can be specialized to perform them.

Example: *Account Management*

- A service can be partitioned and have multiple **service functions**.

Example: *Open new account*

- The smallest subunits within service functions are called **service primitives**.

Example: *Generate next available account number*

## Terms and Definitions

**Characteristics to completely describe a service**

- Service requester (“client”)
  - *Who/which components use or need the service?*
  - For the service requester, the provided *service* is most important, and not how it is implemented (principle of information and implementation hiding)
- Service provider (“server”)
  - *Who/which components implement or provide the service?*
  - is responsible for hosting the service, and ensuring the promised QoS
  - may charge for service usage
- Qualities of Service (QoS)
  - *What are the parameters that allow to distinguish good service provisioning from bad?*
  - Examples: Reliable, predictable execution, cost, execution time, level of privacy, other guarantees

Terms and Definitions

## Interface (Definition)

- An *interface* constitutes the specification of a service, that is implemented by a certain component. The interface defines a contract, to which the component that implements it has to comply.
- Signature of interfaces **can be** described using formal languages
  - Web Services Description Language (WSDL)
  - OMG/ISO Interface Definition Language (IDL) (for CORBA)
  - UML Object Constraint Language (OCL)
  - But also: Java, C++ headers, . . .
- Type-safe interfaces sometimes introduce tight coupling
  - Web Services don't force you into type-safe interfaces
    - Different message types may be acceptable to a service

Terms and Definitions

## Software Architecture (Definition)

*Software architecture* encompasses the set of significant decisions about the organization of a software system

- selection of the *structural* elements from which the system is composed, and the interfaces to these
- *behavior* as specified in collaborations among those elements
- (de)*composition* of these structural and behavioral elements into a larger system
- architectural *style* that guides this organization

Mary Shaw  
(Carnegie Mellon Univ.)

## Terms and Definitions

**Architectural Style (Definition)**

- An *architectural style* defines a family of systems in terms of a pattern of structural organization [Garland/Shaw 96]
- An architectural style defines:
  - a *vocabulary of design elements* components (client, server, filter, layer, adapter...), and connector types (pipe, broadcast, queue,...)
  - a *set of configuration rules* (constraints) on how they can be combined
- Example styles in Software Architecture: Event-based, Repository-based, virtual machines, layered
- Good reference: Martin Fowler book on Enterprise Integration Architecture – introduces notation for architectural components of enterprise systems

## Terms and Definitions

**Service-Oriented Architecture (SOA)**

- The architectural style of an application is called ***service-oriented*** if it meets the following criteria:
  - It is not monolithic; common blocks of functionality are broken out of the applications and are instead provided by services
    - A significant part of the overall functionality is implemented by services, which exist otherwise independent of the application
  - Design elements for an SOA are:
    - Components: services (can be composites), consumers, providers
    - Connector type: (remote) service invocations
  - Configuration rules for an SOA are:
    - No strict layering (service implementations can use other services)
    - No centralized control entity
    - Services are designed for shared use, and for use that may not even have been anticipated at design time

## Terms and Definitions

**Service-Orientation**

- *Service-Orientation* is an organizational principle
  - A set of principles for building large systems
  - It is not tied to any particular technology
- **Examples:**
  - Common “horizontal” services:
    - Logging, authentication/single-sign-on, systems management, Directory lookup of services, event notification
  - “Vertical” services, specific to your business domain
    - Product feature search service, Address management, Order Status Tracking Service, Truck/trailer tracking service
- As in organizations, there is always more than one way to structure a large system
- The most important question: *How to decompose?*
  - What is the guiding abstraction mechanism?
  - Why would one favor one decomposition for another?

## Terms and Definitions

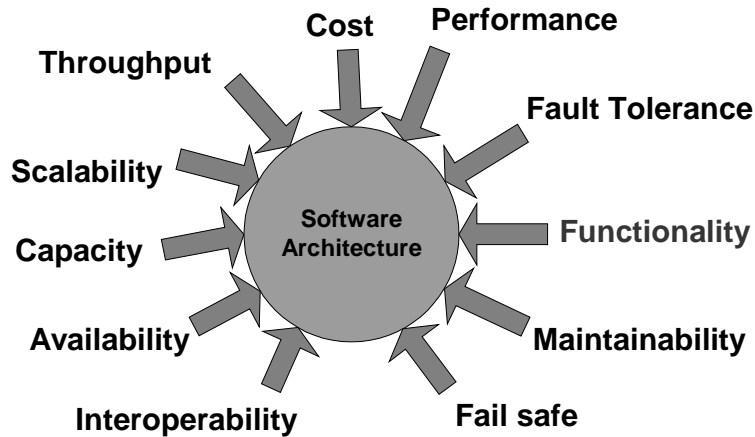
**Quality criteria for (software) architectures**

*Q: What makes one decomposition better than the other?*

*A: Depends on the priorities of your requirements!*

- Quality criteria used to measure the result:
  1. Derived from top rank use cases (*adequateness*)
  2. Balanced with existing assets: platform technology, frameworks, components
  3. Balanced with requirements (trade-offs, performance vs. security,...)
  4. Compliance with (domain-specific) industry standards and reference models (*interoperability, readiness for merging*)
  5. Designed to make the system more resilient to future changes (20 years?) (*maintainability*)
  6. Designed for substantial reuse, and with substantial reuse
  7. Intuitively understandable (people buy-in!) (*usability*)

## Forces and quality factors for software arch.



## Challenges and problems

- Technology heterogeneity of existing applications
- Many integration projects during the last three years
  - Have been tactical, not strategic (no enterprise focus)
  - Only solved problems for one project (e.g. for one channel)
  - Result: Stovepipe integrations
- Large organizations run literally hundreds of packaged applications and often even multiple ERP systems
  - These systems were designed as stand-alone and independent of each other
  - They overlap in functionality
  - They overlap in the data they manage
  - Integration does not remove any overlap
  - Consolidation would be better
- SOA helps to address these issues

## Benefits of service-oriented design (1)

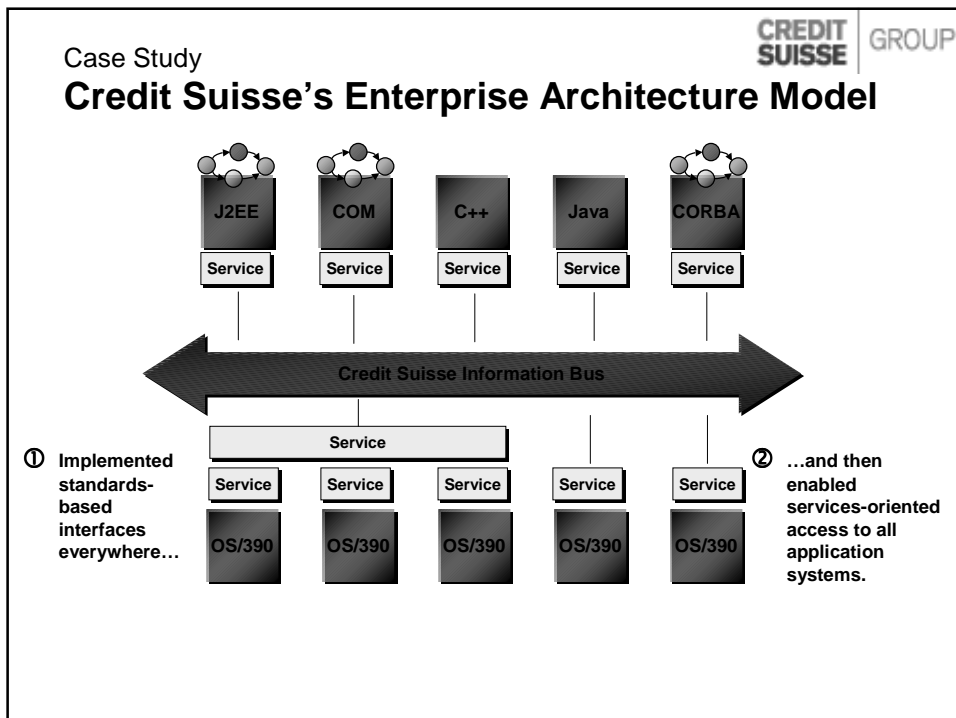
- More flexibility (“business agility”)
  - Assumption: business process logic and business rules are no longer buried inside applications
  - Result:
    - Since they are now explicit, processes can be changed easier
    - Existing services can be used in different contexts
    - Shorter time-to market for changed processes
- Reduced cost of operation through consolidation
  - Assumption: Redundant functionality is eliminated
  - Result:
    - Fewer servers
    - Fewer licenses
    - Fewer assets to manage
    - Lower maintenance cost

## Benefits of service-oriented design (2)

- Higher quality
  - Eliminating redundancy will reduce inconsistent data and inconsistent behavior
  - More transparency
  - Improved system architecture – easier to understand
- Reduced risk, cost and complexity for development
  - Clean architecture ⇒ reduced cost and risk
  - Increased developer productivity through reuse
    - Projects can leverage existing services
  - “Black box” reuse instead of copy&paste reuse
- Lessen the dependencies on vendors
  - Service implementations can be replaced as long as interfaces stay the same
  - Services can be relocated from one platform to another
  - Services can even be outsourced to an external provider

## Benefits of service-oriented design (3)

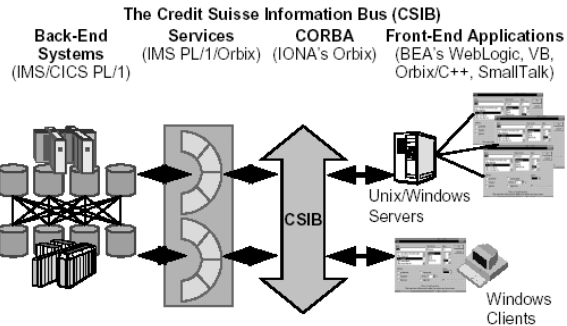
- Good service design (partitioning) will outlive your middleware or implementation technology
  - All you have to do is to put a wrapper around it, if required
  - Many mainframe systems today provide many useful services that should be made available to applications elsewhere in the enterprise
- Commoditizing more and more parts of the IT infrastructure
  - Off-the-shelf infrastructure components are moving up the layers and coming closer to the application!
  - Due to existing industry standards and available products, developers stop building this stuff themselves:
    - 1990: DBMS, TP Monitors
    - 1992: Networking stacks
    - 1995: CORBA, RPC Middleware, Reliable Messaging
    - 1998: Naming Service, Publish and Subscribe, Event Notification
    - 2000: Various J2EE Services



Case Study

# Credit Suisse's Enterprise Architecture Model

- 500 business service
- 50-plus front-end composite applications
- 100,000 users
- 7 million service invocations per week



"There is no need for us to dismantle our existing infrastructure to deploy on-line business opportunities. IONA enables us to leverage multiple technology generations in a flexible system, which ensures shorter development cycles and investment protection."

*Stephan Murer, Credit Suisse*

## What does all this mean for you? (1)

- Your role: Developer or Software Architect
  - Project-focused ("get this done")
  - Success metric: Project / deliverable completed in time
  - Goal: Minimize time and cost of development
  - Granularity of re-use: class libraries, source code, classes, frameworks
- What you need to do
  - be familiar with Web Services standards and technologies
  - get your feet wet
  - be aware of existing services in your enterprise
    - try to (re)use them as much as possible

## What does all this mean for you? (2)

- Your role: Enterprise Architect or CIO
  - Enterprise- or infrastructure-focused
  - Success metric: Resourcefulness
  - Goals: Minimize TCO across projects, reduce complexity, maximize “business agility”
  - Granularity of re-use: services, infrastructure
- What you need to do
  - understand Web Services technologies
  - define the roadmap, act as catalyst
  - define the business architecture
  - work for the application teams and encourage them to use the services of “your” infrastructure

## Summary

- Precise terminology helps
  - to understand and handle the problem
  - to identify and debunk unbaked concepts
- Service-orientation is an architectural style
- Web services are a good technology to build service oriented architectures
  - Simple, easy to understand
  - Inexpensive technology, no major investment needed
  - Can start small and proceed incrementally