Architecture of Enterprise Applications 19 Service-Oriented Architecture

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

http://reins.se.sjtu.edu.cn/~chenhp

e-mail: chen-hp@sjtu.edu.cn

Agenda

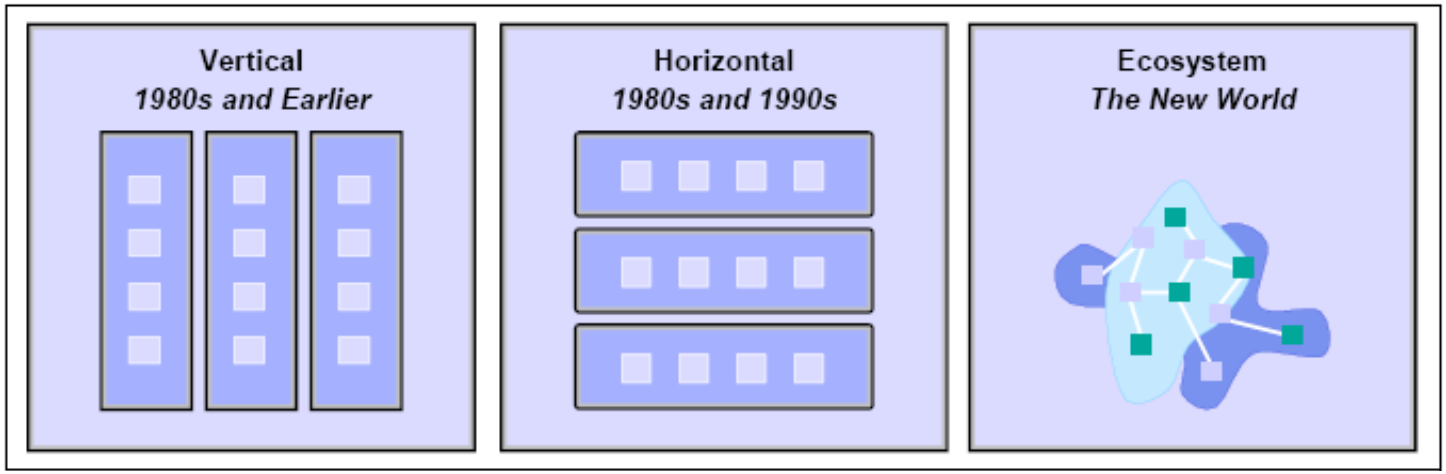
- SOA
  - Business drivers
  - SOA protocols stack
  - Service and component
- SOAD
  - Service-oriented analysis
  - Service-oriented design
- Model of SOAD
  - JBI
  - SCA/SDO
  - .NET
- ESB
  - ESB feature
  - MoM
  - ESB Core

The business drivers for a new approach

- While IT executives have been facing the challenge of
  - cutting costs
  - maximizing the utilization of existing technology
- At the same time they have to
  - continuously strive to serve customers better
  - be more competitive
  - be more responsive to the business's strategic priorities.
- There are two underlying themes behind all of these pressures: Heterogeneity and change

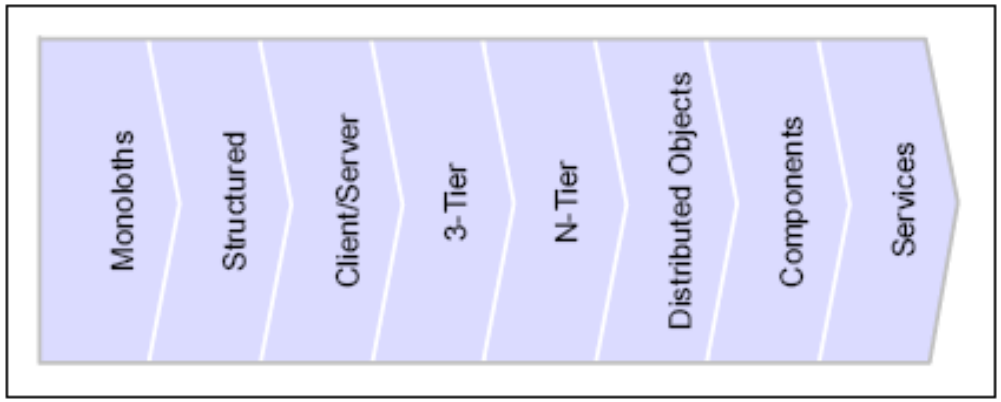
The business drivers for a new approach

- As a result, business organizations are evolving
  - from the vertical, isolated business divisions of the 1980's and earlier
  - to the horizontal business-process-focused structures of the 1980's and 1990's
  - towards the new ecosystem business paradigm. Business services now need to be componentized and distributed.
- There is a focus on the extended supply chain, enabling customer and partner access to business services.



The business drivers for a new approach

- Questions:
  - How do I make my IT environment more flexible and responsive to the ever changing business requirements?
  - How can we make those heterogeneous systems and applications communicate as seamlessly as possible?
  - How can we achieve the business objective without bankrupting the enterprise?
- Currently many IT executives and professionals alike believe that now we are getting really close to providing a satisfactory answer with service-oriented architecture.



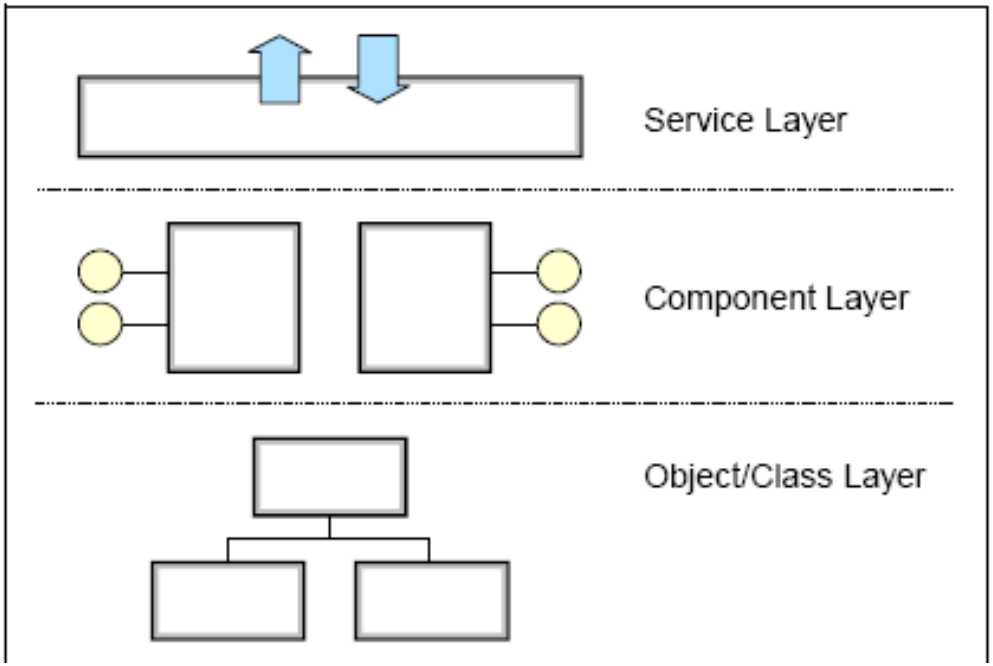
The business drivers for a new approach

- In order to alleviate the problems of heterogeneity, interoperability and ever changing requirements, such an architecture should provide a platform for building application services with the following characteristics:
  - Loosely coupled
  - Location transparent
  - Protocol independent

- Based on such a service-oriented architecture,
  - a service consumer does not even have to care about a particular service it is communicating with because the underlying infrastructure,
  - or service "bus", will make an appropriate choice on behalf of the consumer.

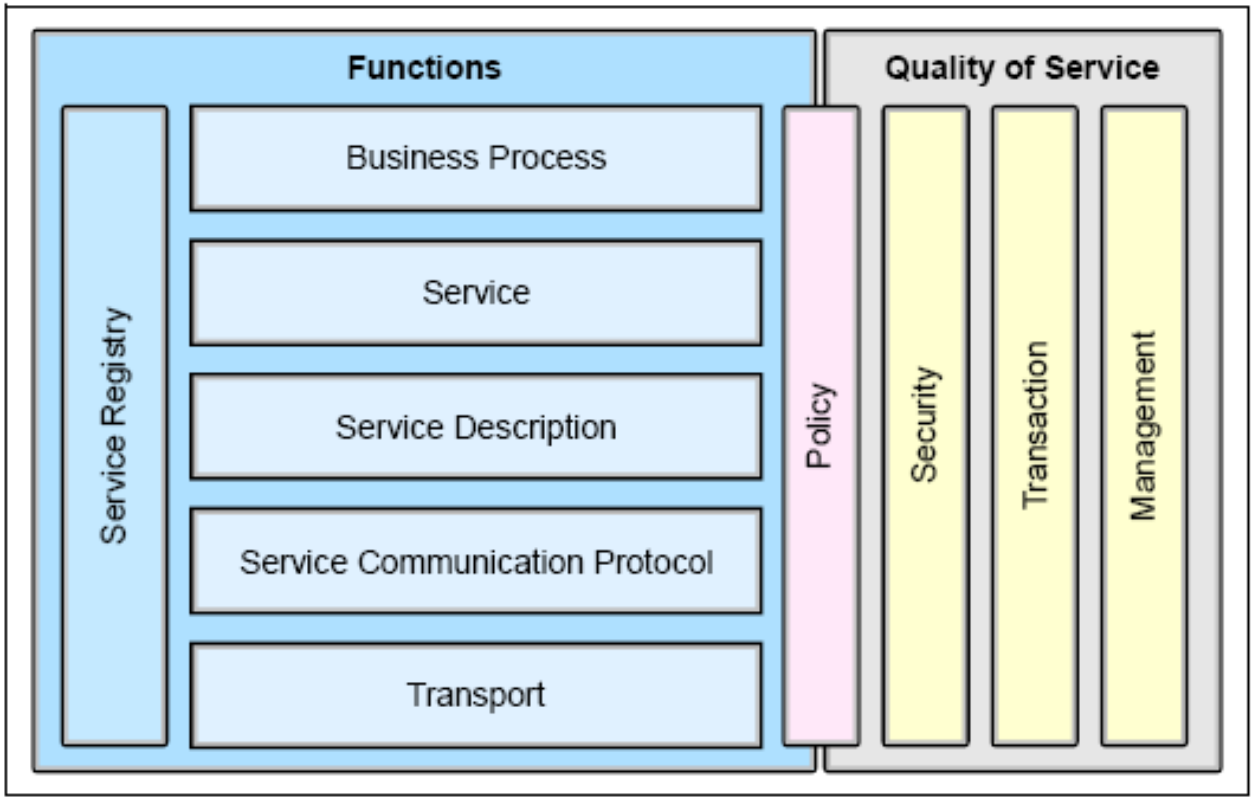
Layered application architectures

- Object-oriented technology and languages are great ways to implement components.
  - While components are the best way to implement services, though one has to understand that a good component-based application does not necessarily make an good service-oriented application.



A closer look at SOA

- Service-oriented architecture presents an approach for building distributed systems that deliver application functionality as services to either end-user applications or other services. It is comprised of elements that can be categorized into functional and quality of service.



A closer look at SOA

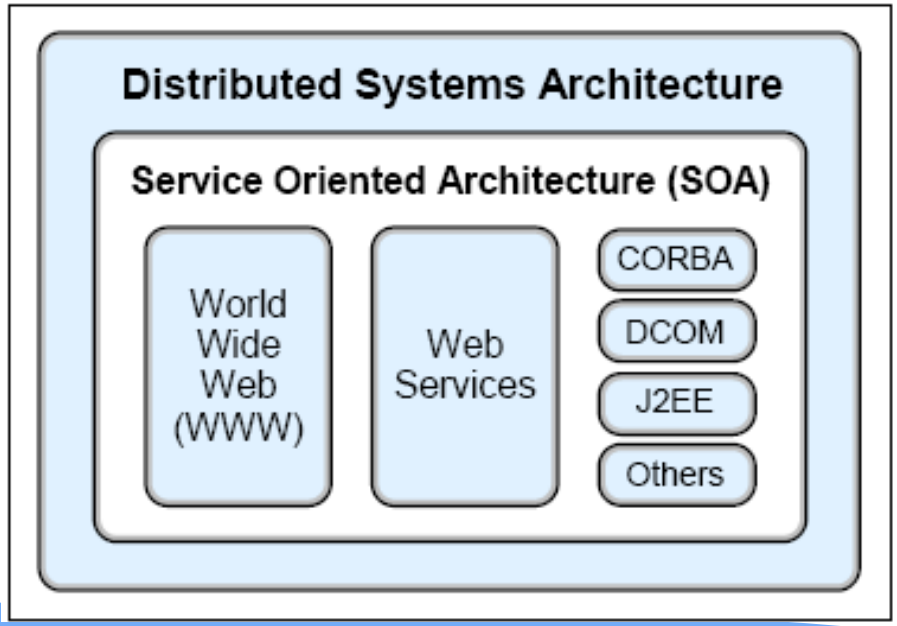
- Functional aspects include:
  - Transport
  - Service Communication Protocol
  - Service Description
  - Business Process
  - Service Registry

- Quality of service aspects include:
  - Policy
  - Security
  - Transaction
  - Management

SOA collaborations

- In addition to dynamic service discovery and definition of a service interface contract, a service-oriented architecture has the following characteristics:
  - Services are self-contained and modular.
  - Services support interoperability.
  - Services are loosely coupled.
  - Services are location-transparent.
  - Services are composite modules, comprised of components.
- These characteristics are also central to fulfilling the requirements for an e-business on demand™ operational environment.

- *Finally, service-oriented architecture is not a new notion.*

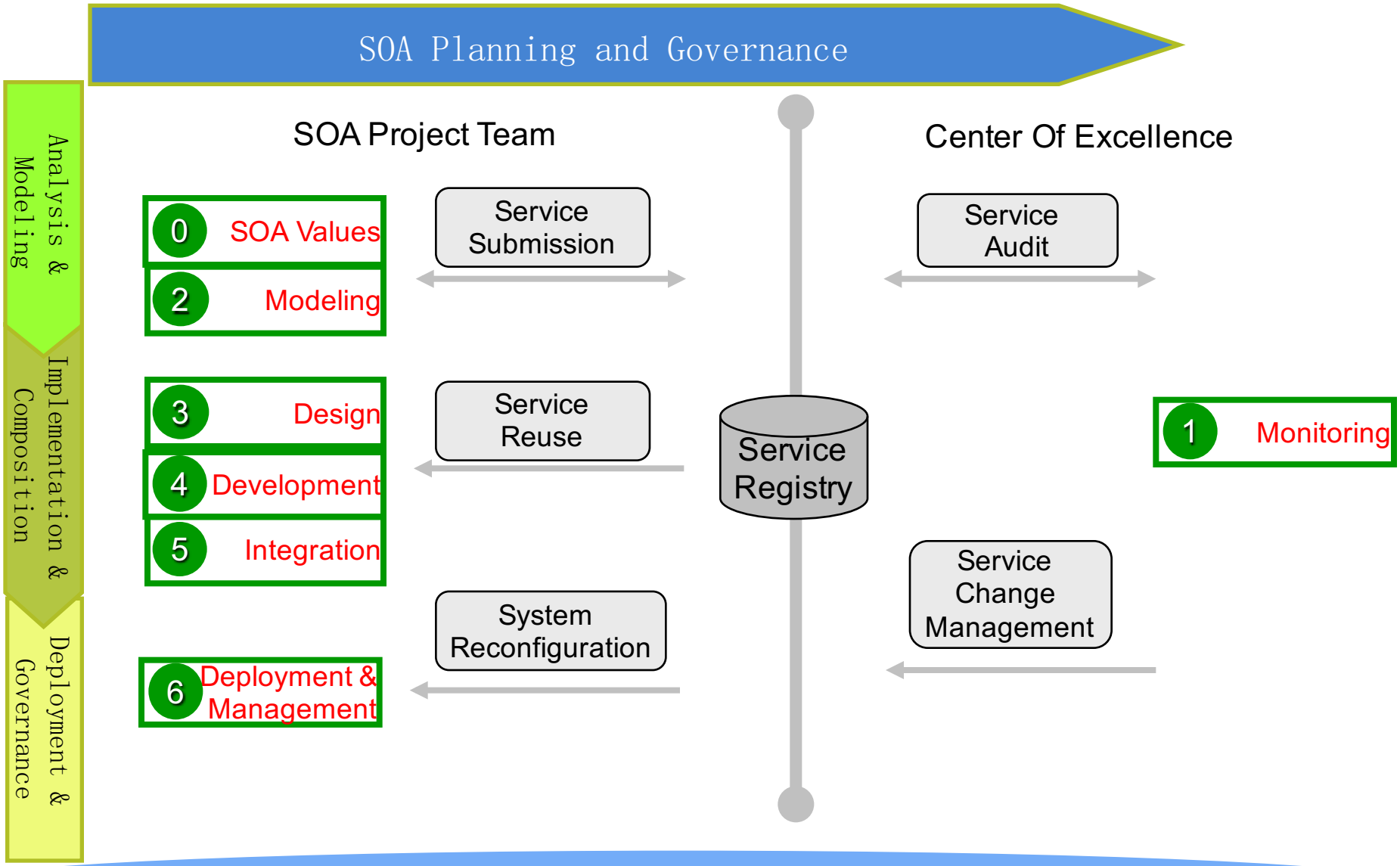


Services vs. Components

- A service is a coarse-grained processing unit that consumes and produces sets of objects passed-by-value.
  - It is not the same as an object in programming language terms. Instead, it is perhaps closer to the concept of a business transaction such as a CICS® or IMS™ transaction than to a remote CORBA object.

- A service consists of a collection of components that work in concert to deliver the business function that the service represents.
  - Thus, in comparison, components are finer-grained than services.
  - In addition, while a service maps to a business function, a component typically maps to business entities and the business rules that operate on them.

Service-Centered System Development

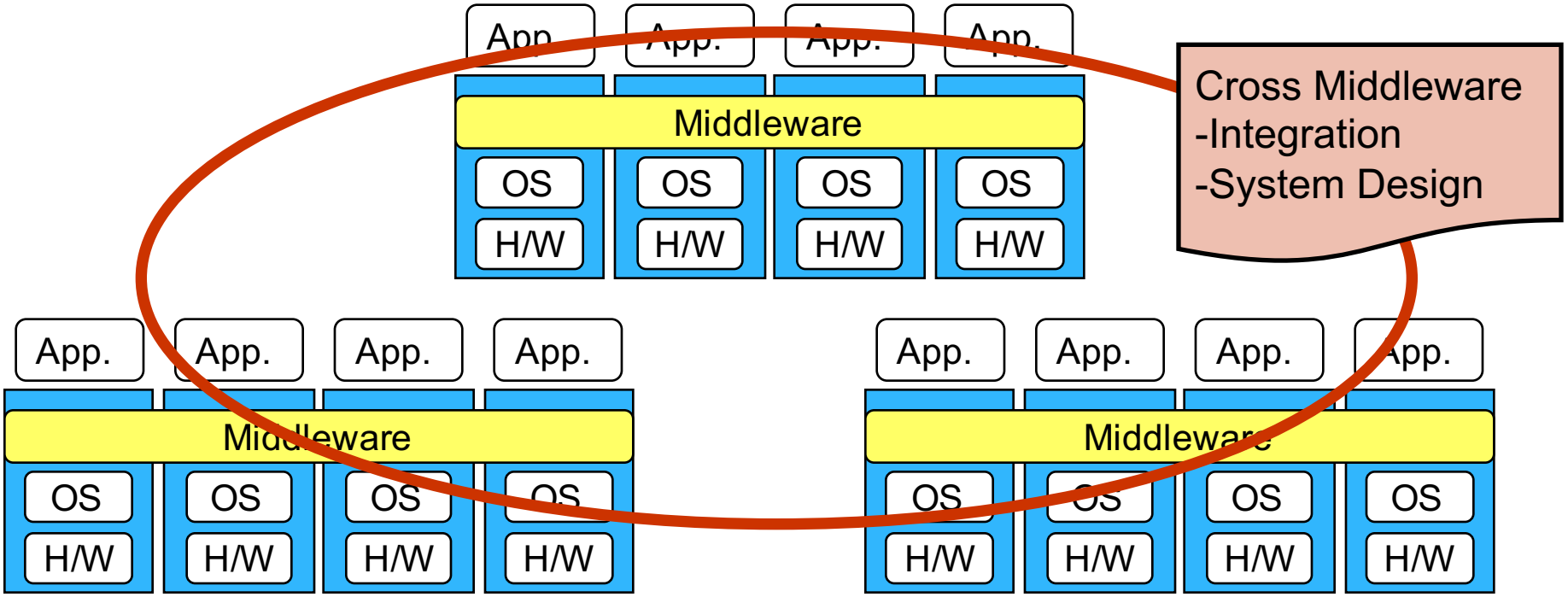


Modeling Motivations

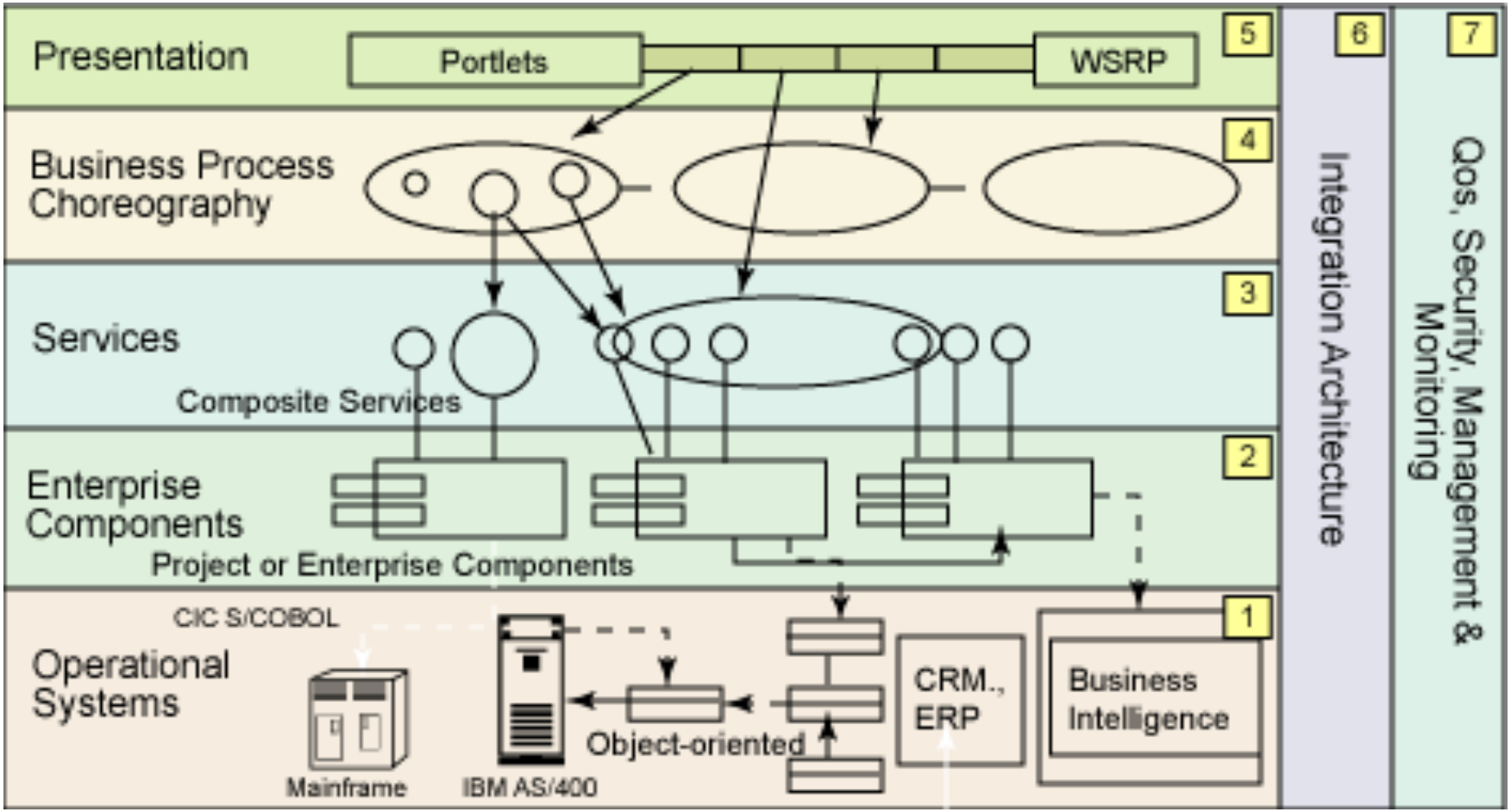
- Varieties of Platforms
  - Varieties of Hardware Architecture
    - Pentium, PowerPC, PA-RISC, Sparc, 370, …
  - Variety of Networks
    - Ethernet, ATM, IP, SS7, Applealk, USB, Firewire, …
  - Variety of Programming Languages
    - C/C++. Java, VB, C#,…
  - Variety of Operating Systems
    - Unix, Windows, NT/XP. Mainframe, Mobile, …
  - Variety of Middlewares
    - JAVA/CORBA, COM+/.NET, Web Services, ….

Modeling Motivations

- Integration challenges
  - Integration across middleware
  - System design across middleware



The SOA Layers

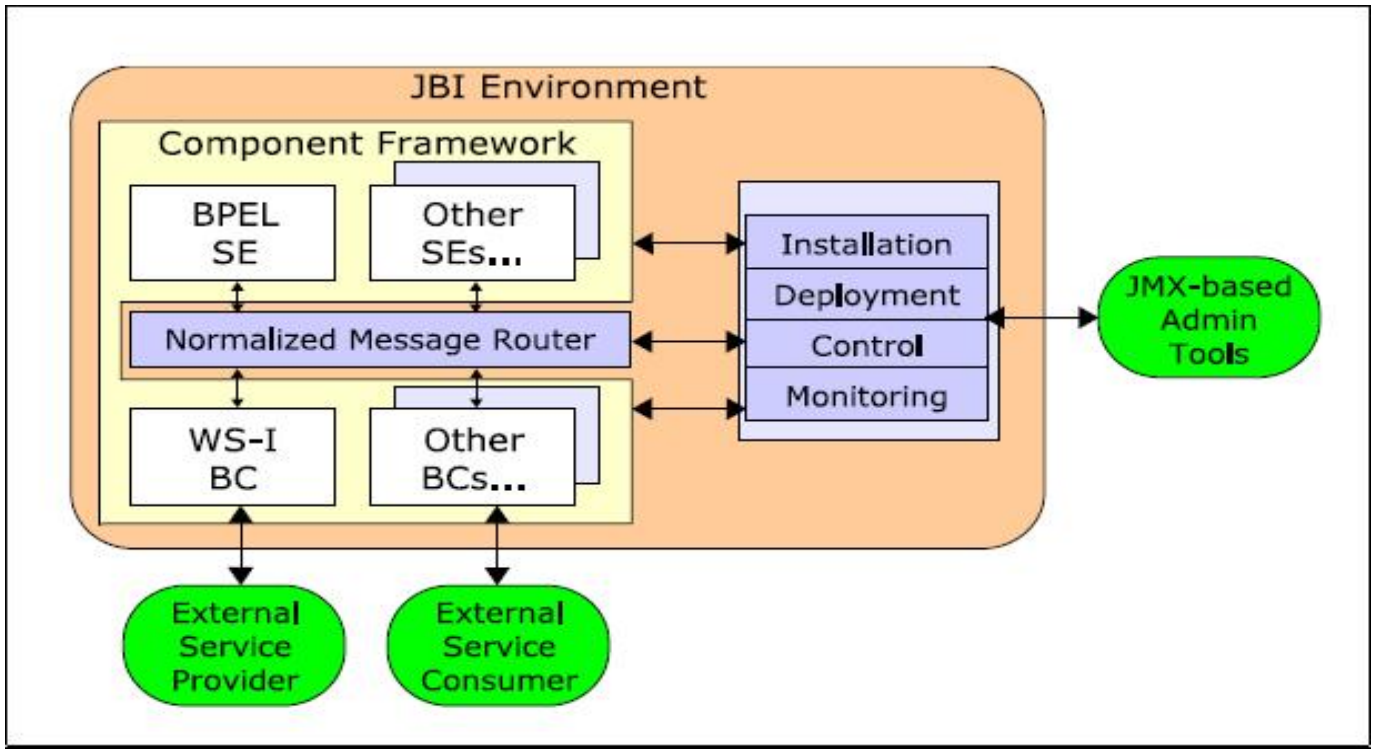


Service-Oriented Modeling & Analysis: Roles & Activities

Customer View	Service Identification	Service Categorization	Service Exposure Decisions	Choreography Or Composition	Quality of service
Provider View	Component Identification	Component Specification	Service realization	Service Management	Standards implementation
	Service Allocation to Components	Layering the Component	Technical Prototyping	Product selection	Architectural Decisions (state, flow, Dependencies)

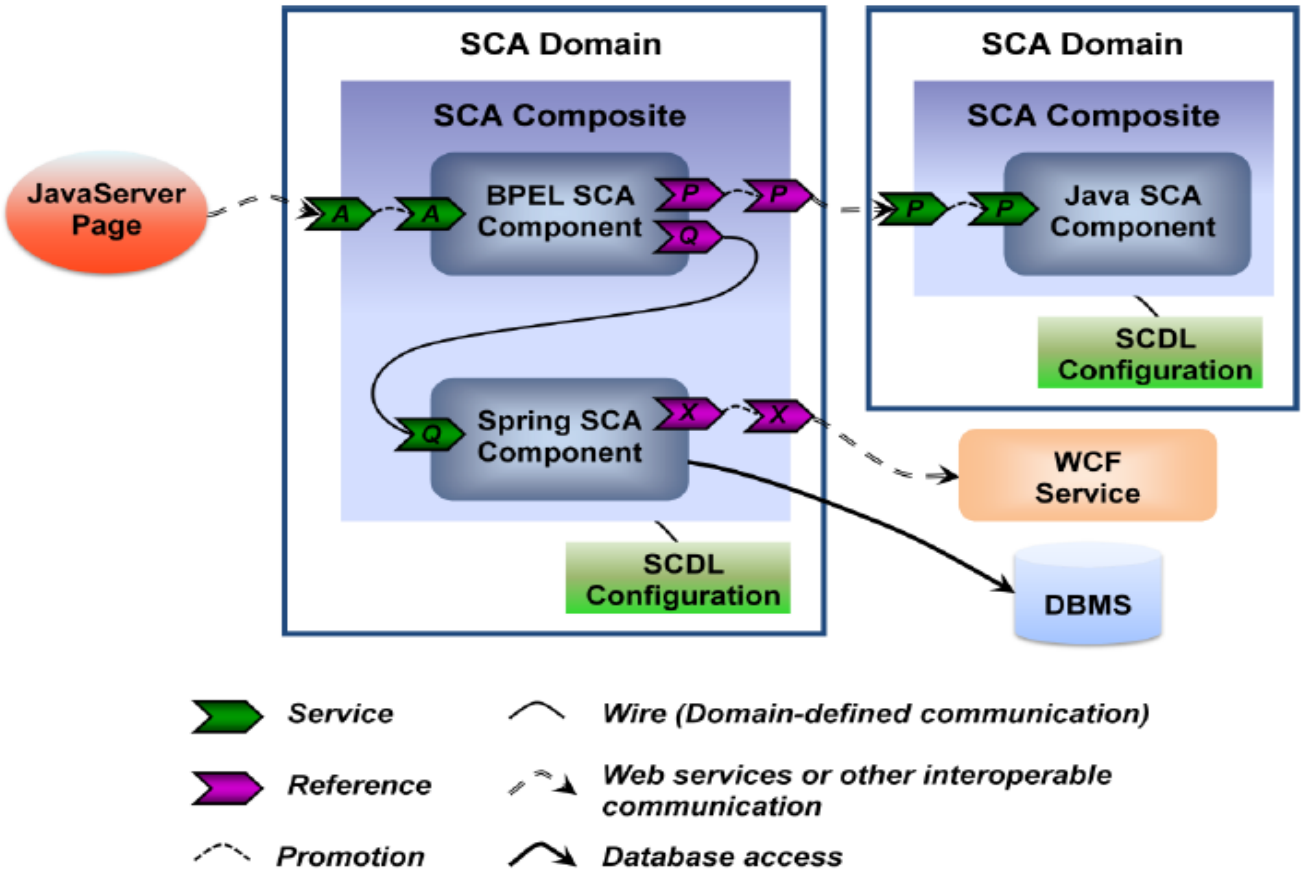
Models of SOAD

- **JBI（Java Business Integration）JSR 208**



Models of SOAD

- **SCA(Service Component Architecture)/SDO(Service Data Object)**



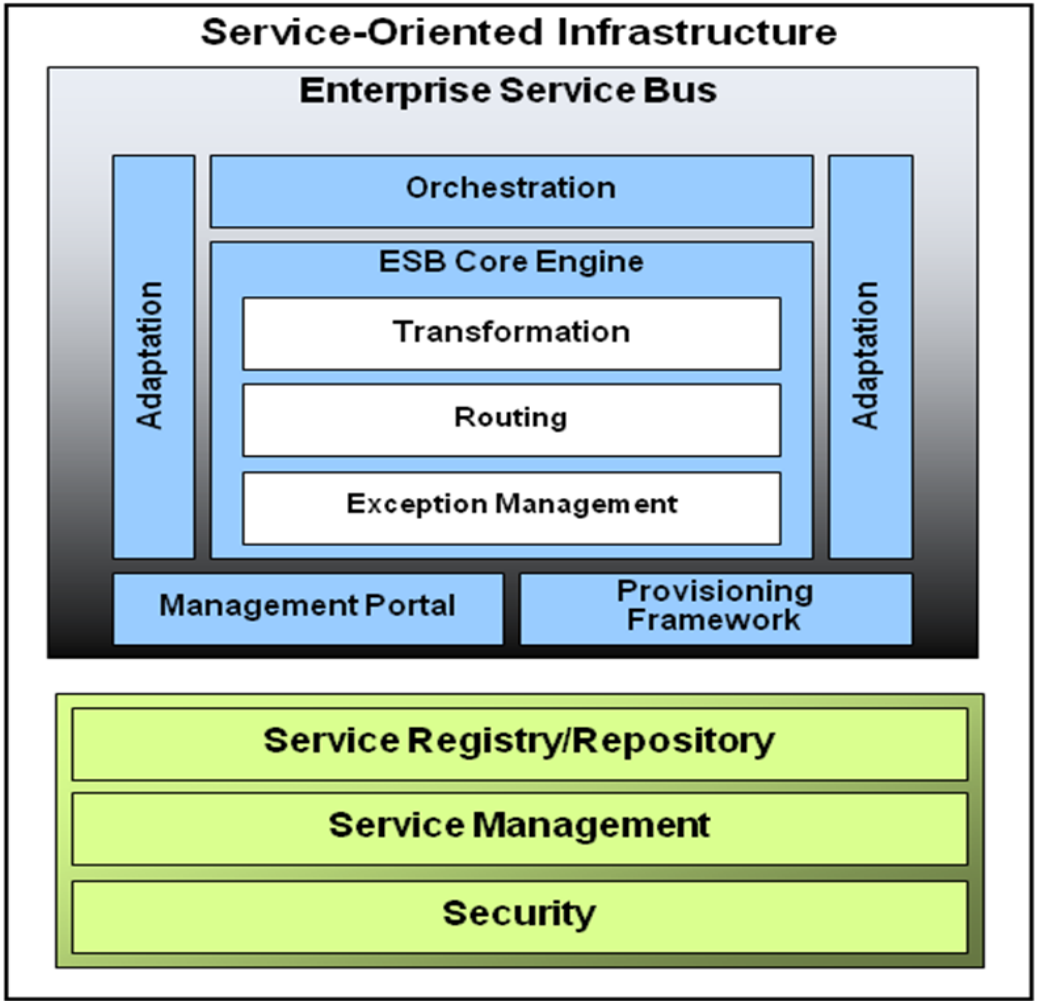
Models of SOAD

- **Windows Server2003 + .NET Framework3.0 + BizTalk Server2006 R2**

- **BizTalk Server**
  - **Message routing**
  - **Message validation**
  - **Message transformation)**
  - **Centralized exception management**
  - **Extensible adapter framework**
  - **Service orchestration**
  - **Business rules engine**
  - **Business activity monitoring**

Models of SOAD

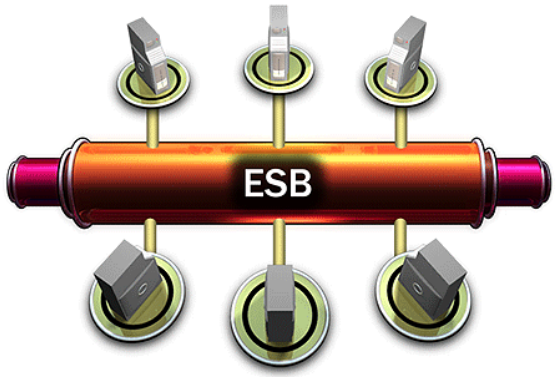
- **Windows Server + .NET Framework + BizTalk Server**



Necessity is the mother of invention

- ESB does not come of nowhere; many catalysts helped it develop and evolve. Lessons were learned from past technology approaches that extend back more than a decade.

- ESB is not merely an academic experience; it was born out of necessary, based on real requirements arising from difficult integration problems that couldn't be solved by any preexisting integration technology.

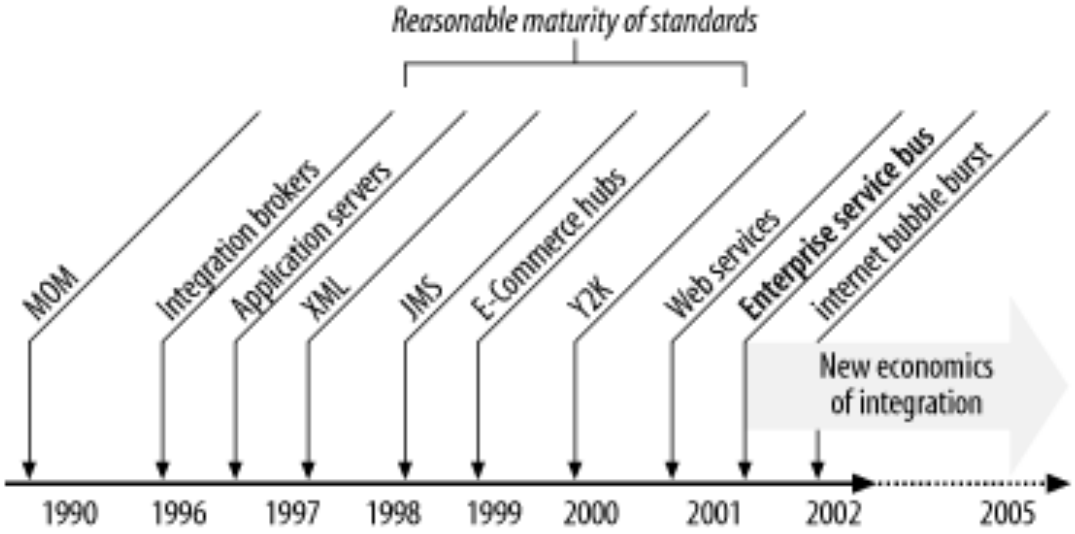


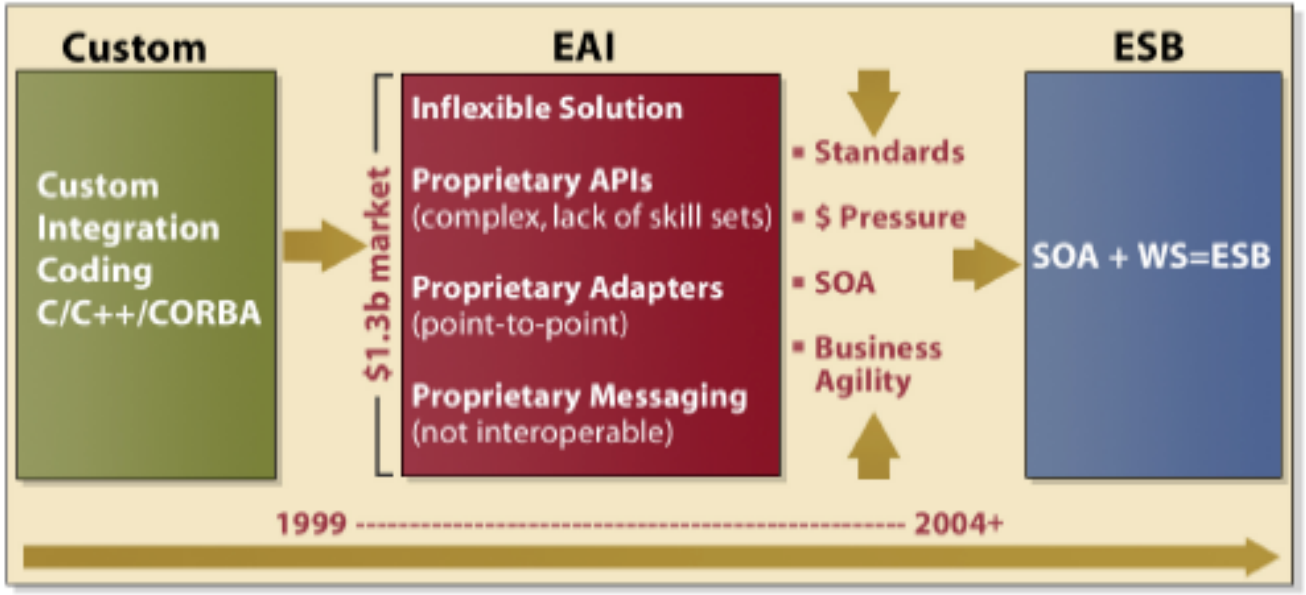
The Necessity of ESB

- Enterprise that wish to implement SOA need a more sophisticated, manageable infrastructure that can support high volumes of individual interactions.
- Such infrastructure should support more established integration styles
  - Message-oriented
  - Event-driven
  - Legacy integration
- Such infrastructure should support enterprise-level quality of service.

ESB is emerging as the unifying concept for such infrastructure

The Evolution of ESB





ESB Minimum Capabilities

Category	Capabilities	Reasons
Communications	► Routing ► Addressing ► At least one messaging style (request / response, pub/sub) ► At least one transport protocol that is or can be made widely available	Provide location transparency and support service substitution
Integration	► Several integration styles or adapters ► Protocol transformation	Support integration in heterogeneous environments and support service substitution
Service interaction	► Service interface definition ► Service messaging model ► Substitution of service implementation	Support SOA principles, separating application code from specific service protocols and implementations
Management	► Administration capability	A point of control over service addressing and naming

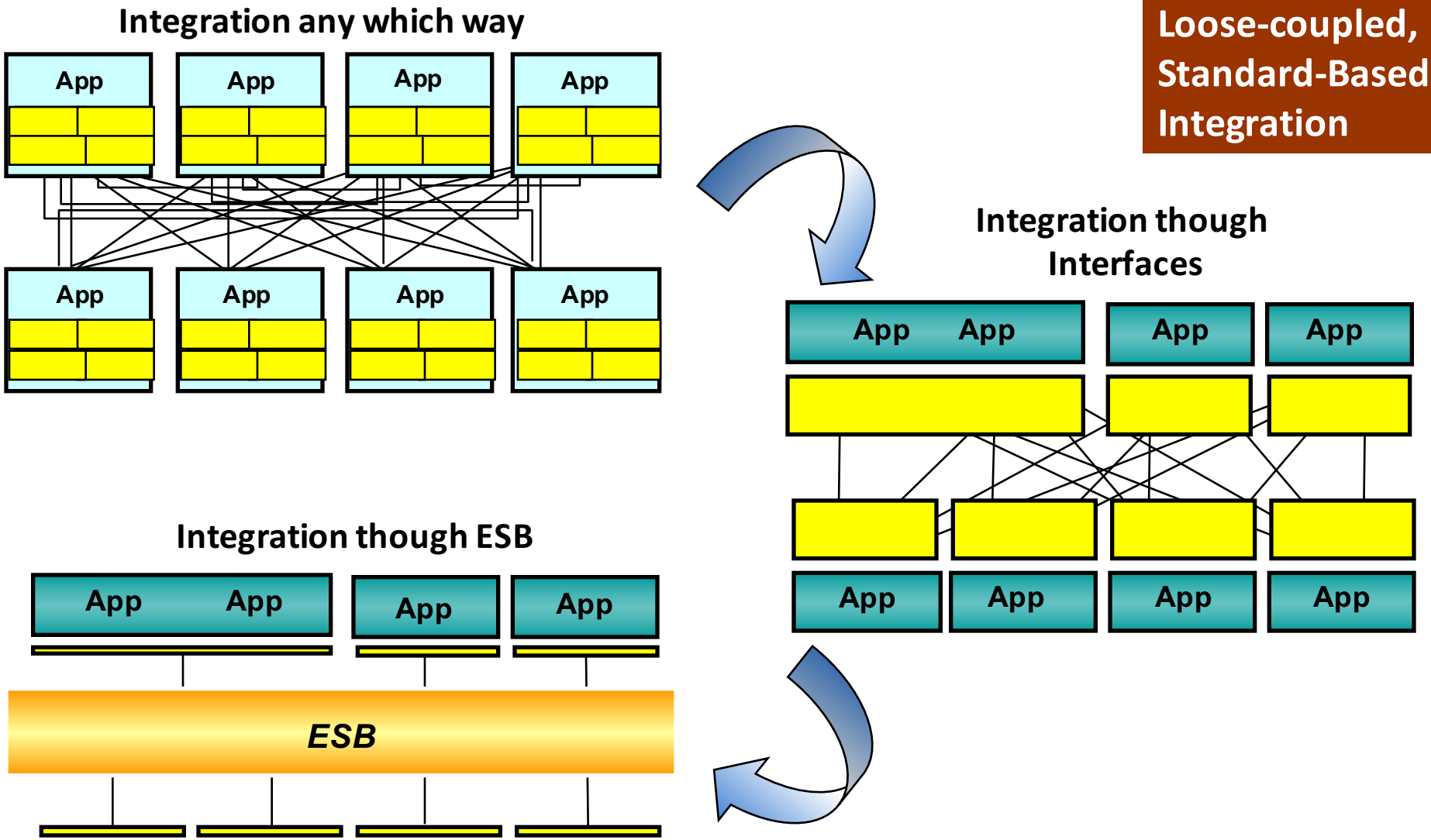
-- "Getting Started with Websphere ESB",
IBM Red Book (SG24-7212-00), 2006

ESB Extended Capabilities

Communication	Service interaction
► Routing ► Addressing ► Protocols and standards (HTTP, HTTPS) ► Publish/subscribe ► Response/request ► Fire and forget, events ► Synchronous and asynchronous messaging	► Service interface definition (WSDL) ► Substitution of service implementation ► Service messaging models required for communication and integration (SOAP, XML, or proprietary Enterprise Application Integration models) ► Service directory and discovery
Integration	**Quality of service**
► Database ► Existing and application adapters ► Connectivity to enterprise application integration middleware ► Service mapping ► Protocol transformation ► Data enrichment ► Application server environments (J2EE and .Net) ► Language interfaces for service invocation (Java, C/C++/C#)	► Transactions (atomic transactions, compensation, WS-Transaction) ► Various assured delivery paradigms (WS-ReliableMessaging or support for Enterprise Application Integration middleware)
Security	**Service level**
► Authentication ► Authorization ► Non-repudiation ► Confidentiality ► Security standards (Kerberos, WS-Security)	► Performance (response time, throughput, and capacity) ► Availability ► Other continuous measures that might form the basis of contracts or agreements

The ESB Story

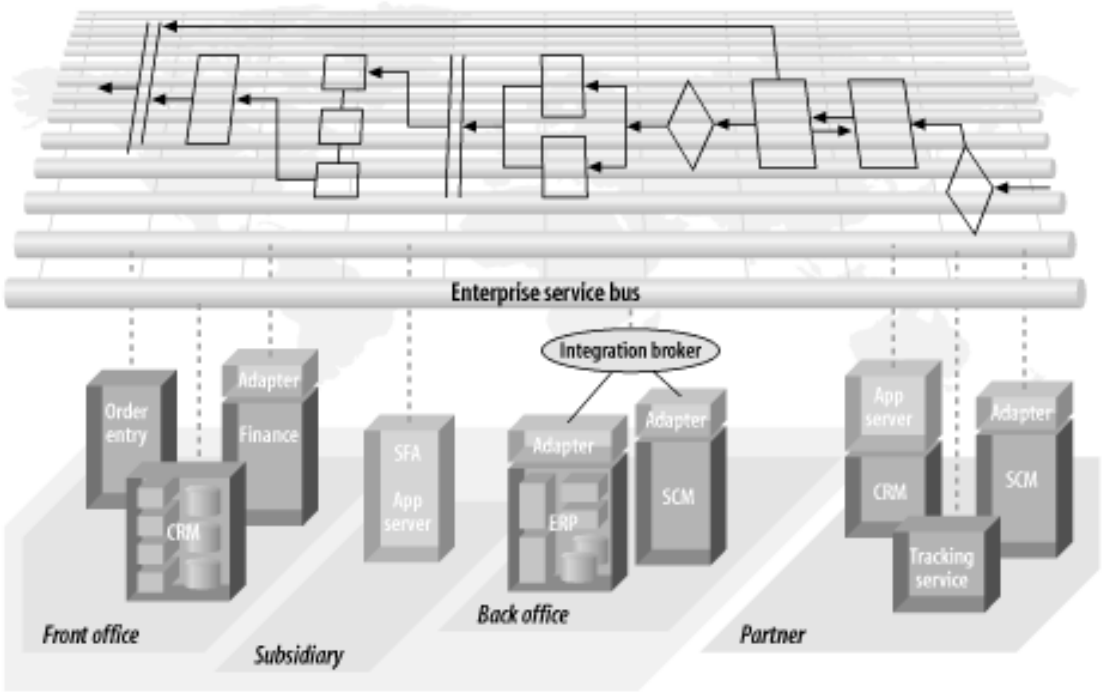
Loose-coupled, Standard-Based Integration

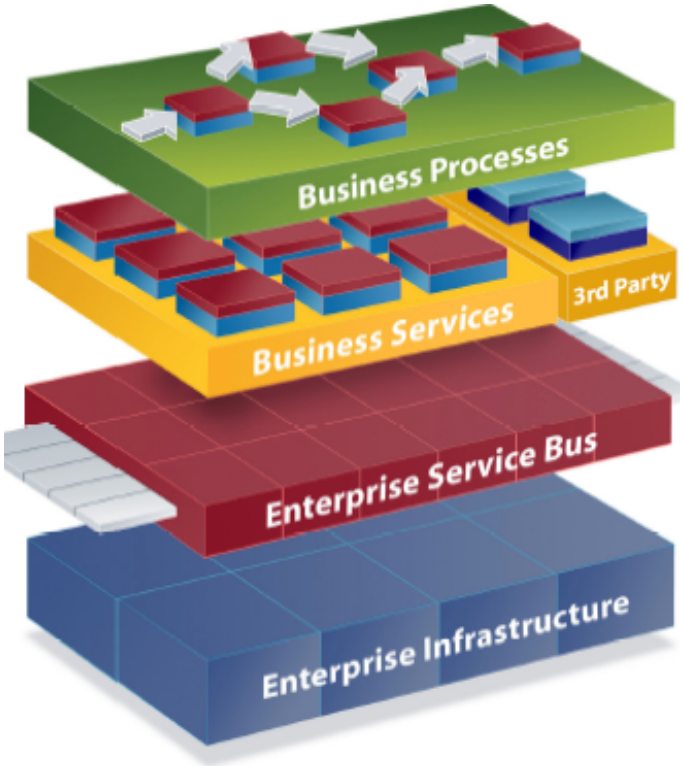


The ESB Story

Highly Distributed Integration And Selective Deployment

Orchestration and process flow spanning highly distributed deployment topologies across physical and logical boundaries





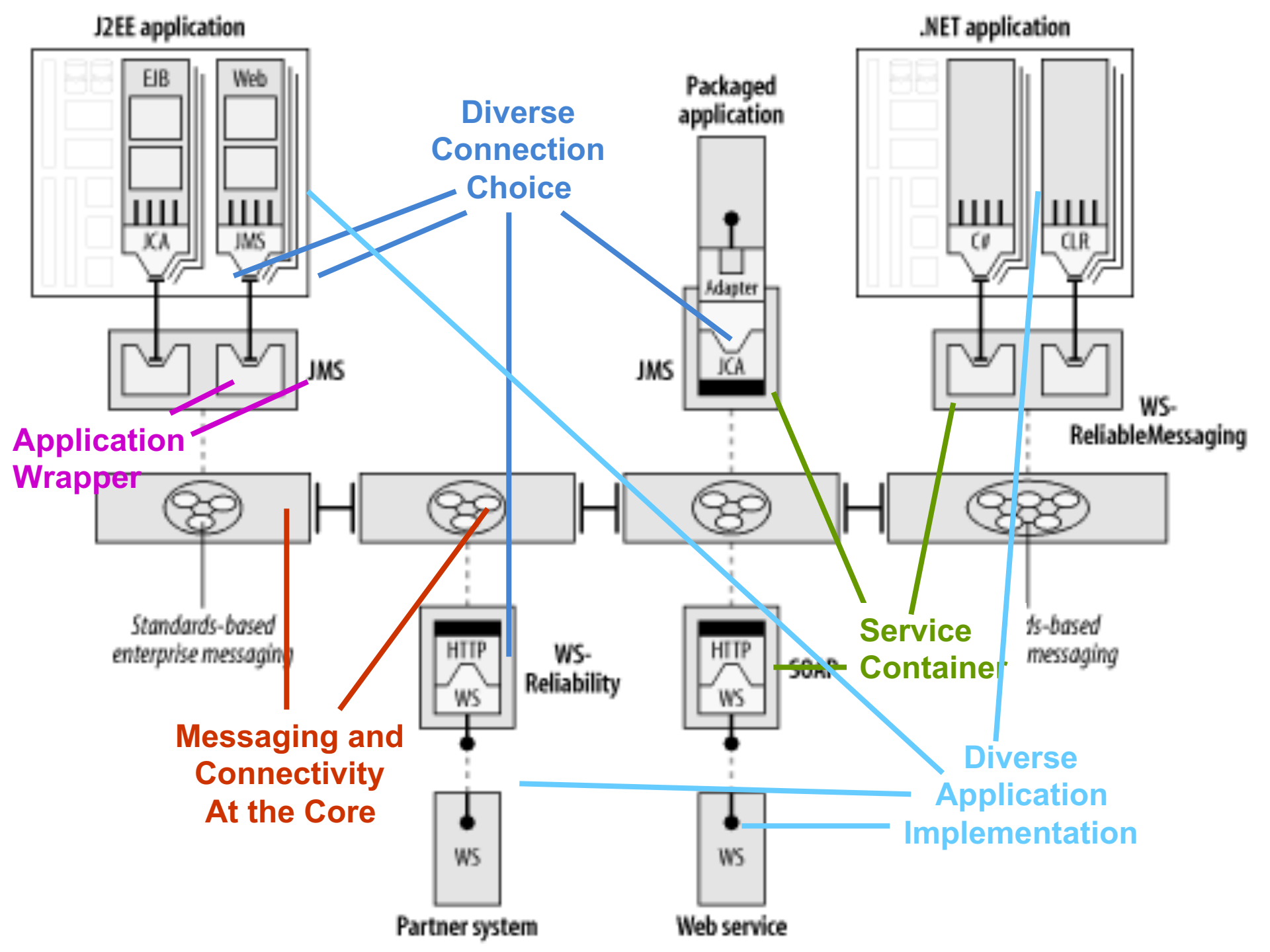
Extensibility through layered services

ESB Characteristics

- Pervasiveness
- Highly distributed, event-driven SOA
- Selective deployment of integration components
- Security and reliability
- Orchestration and process flow
- Autonomous yet federated managed environment
- Incremental adoption. An ESB can be used for small project
- XML support
- Real-time insight

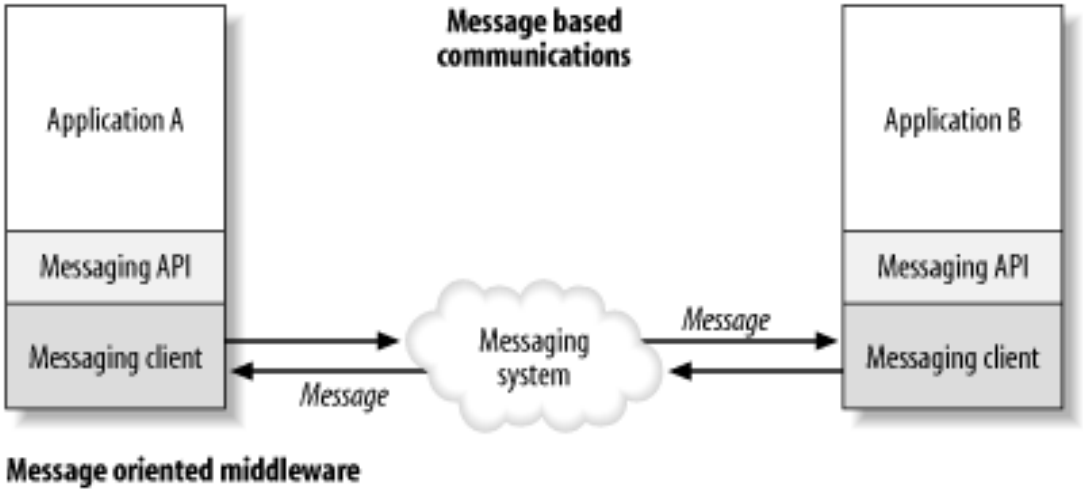
ESB Key Components

- Message Oriented Middleware
  - Robust, reliable transport
  - Efficient movement of data across abstract data channels
  - End-to-end reliability
- Service Container and Abstract Endpoints
  - Endpoints
    - Logical abstraction, representing remote services in various implementations
  - Container
    - The physical manifestation of the endpoints
    - Distributed and lightweight
- Intelligent routing
  - Message routing based on content and context
  - Message routing based on business process rules
  - Business process orchestration based on a rules language such BPEL4WS


J2EE application
EJB
Web
JCA
JMS
.NET application
C#
CLR
Packaged application
Adapter
JCA
JMS
JMS
WS-ReliableMessaging
Diverse Connection Choice
Application Wrapper
Standards-based enterprise messaging
HTTP
WS
WS-Reliability
HTTP
WS
SOAP
Service Container
ds-based messaging
Messaging and Connectivity At the Core
Diverse Application Implementation
WS
WS
Partner system
Web service

Message Oriented Middleware (MOM)

- Virtual channels that an ESB uses to route messages
- Self-contained units of information (messages)
- Asynchronous communication
- Applications are abstractly decoupled
- Messaging system supports the management of connection points between multiple messaging clients, and of multiple channels of communication between connection points.
  - Message server
  - Message broker



Apache ServiceMix

- Apache ServiceMix is
  - a flexible, open-source integration container that
    - unifies the features and functionality of Apache ActiveMQ, Camel, CXF, and Karaf into a powerful runtime platform you can use to build your own integrations solutions.
  - It provides a complete, enterprise ready ESB exclusively powered by OSGi.

Apache ServiceMix

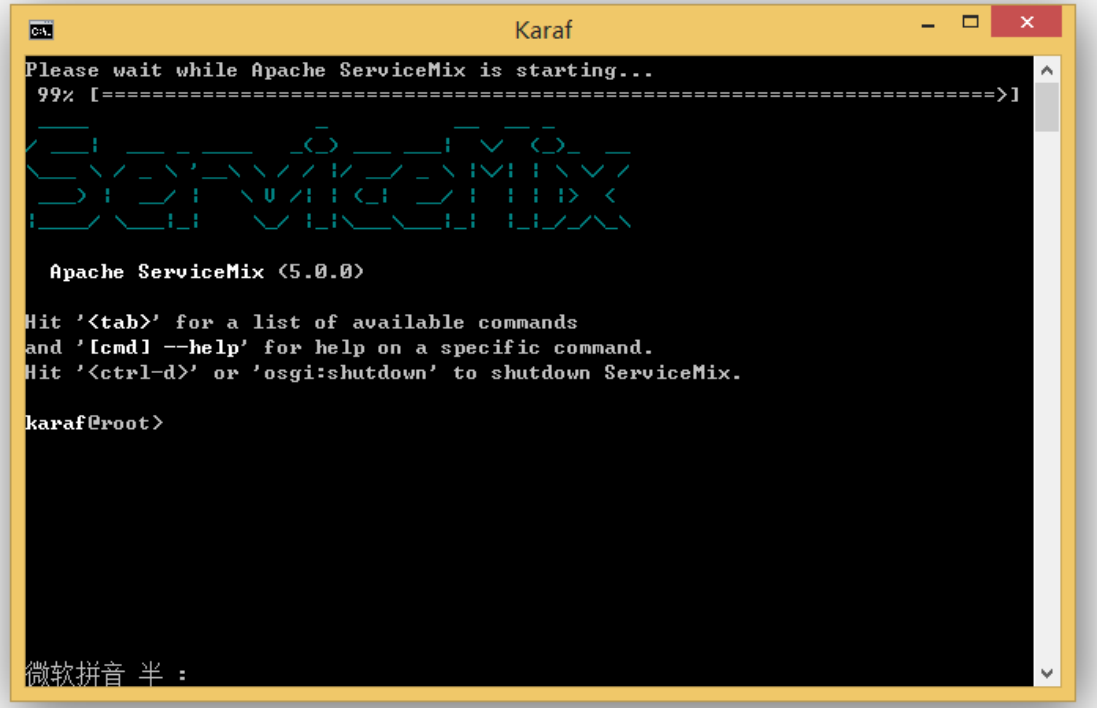
- The main features are:
  - reliable messaging with Apache ActiveMQ
  - messaging, routing and Enterprise Integration Patterns with Apache Camel
  - WS-* and RESTful web services with Apache CXF
  - OSGi-based server runtime powered by Apache Karaf

- Through additional installable features, ServiceMix also supports:
  - BPM engine via Activiti
  - full JPA support via Apache OpenJPA
  - XA transaction management via JTA via Apache Aries
  - legacy support for the JBI standard (deprecated after the ServiceMix 3.x series) through the Apache ServiceMix NMR that includes a rich Event, Messaging and Audit API

- Applications for ServiceMix can be built using:
  - OSGi Blueprint
  - OSGi Declarative Services
  - Spring DM (legacy)

Apache ServiceMix

- Startup ServiceMix:
  - > cd SERVICEMIX_HOME
  - > .\bin\servicemix



- **A simple scenario**
  - In this simple scenario, we're going to move files from an input directory called camel/input to an output directory called camel/output.
  - To ensure we can keep track of which files get moved, we'll also write a message to the log file whenever we move a file.

- **Creating the route**
  - One of the most simple ways to deploy a new route on ServiceMix, is by defining the route in a Blueprint XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<blueprint xmlns="http://www.osgi.org/xmlns/blueprint/v1.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="
            http://www.osgi.org/xmlns/blueprint/v1.0.0
            http://www.osgi.org/xmlns/blueprint/v1.0.0/blueprint.xsd">
    <camelContext xmlns="http://camel.apache.org/schema/blueprint">
        <route>
            <from uri="file:camel/input"/>
            <log message="Moving ${file:name} to the output directory"/>
            <to uri="file:camel/output"/>
        </route>
    </camelContext>
</blueprint>
```

- **Deploying the route**
  - In order to deploy and start the route, just copy the XML file you created into ServiceMix' deploy directory.
  - The file will get picked up and deployed by ServiceMix.
  - You will see a camel/input folder appear in your ServiceMix installation directory and any files you copy into that directory will get moved into the camel/output directory.

- **Another scenario**
  - In this scenario, we also want to move files between directories. Instead of logging the move directly, we are going to send an event JMS message onto a queue.
  - Afterwards, we will create a second Camel route to receive the events and log them.

- First,
  - we have to install an optional feature to allow Camel send and receive JMS messages.
  - We'll talk about optional features a bit more in the next page of this quickstart guide, but for now, just run the following command in the console.
  - karaf@root> features:install camel-jms

- **Moving files and sending event messages**

```
<?xml version="1.0" encoding="UTF-8"?>
  <blueprint>
    <camelContext>
      <route>
        <from uri="file:activemq/input"/>
        <to uri="file:activemq/output"/>
        <setBody>
          <simple>
            FileMovedEvent(file: ${file:name},
            timestamp: ${date:now:hh:MM:ss.SSS})
          </simple>
        </setBody>
        <to uri="activemq://events" />
      </route>
    </camelContext>
    <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
      <property name="brokerURL" value="tcp://localhost:61616"/>
      <property name="userName" value="smx"/>
      <property name="password" value="smx"/>
    </bean>
</blueprint>
```

- **Receiving event messages**

```
<?xml version="1.0" encoding="UTF-8"?>
 <blueprint>
   <camelContext>
     <route>
       <from uri="activemq://events"/>
       <to uri="log:events"/>
     </route>
   </camelContext>
   <bean id="activemq" class="org.apache.activemq.camel.component.ActiveMQComponent">
     <property name="brokerURL" value="tcp://localhost:61616"/>
     <property name="userName" value="smx"/>
     <property name="password" value="smx"/>
   </bean>
</blueprint>
```

Project

- Requirement
  - Try ServiceMix. You can submit a separate project to demonstrate SOA application.
  - The SOA application your submitted could be a Document Approval System, in which the approval service can be substitute by other implementations without modification of any source code since the services are decoupled with each other.

References

- Patterns Service Oriented Architecture and Web Services
  - IBM Redbook
- Courseware of Prof. Xiaoying Bai
  - Department of Computer Science and Technology, Tsinghua University
- ServiceMix
  - http://servicemix.apache.org/

Thank You!