EAI

OVERVIEW OF ENTERPRISE APPLICATION INTEGRATION CONCEPTS AND ARCHITECTURES

Peter R. Egli
peteregli.net

Contents

1. EAI versus SOA versus ESB
2. EAI
3. SOA
4. ESB
5. N-tier enterprise architecture
6. WS-BPEL
7. WOA

1. EAI versus SOA versus ESB (1/3)

What is EAI (Enterprise Application Integration)?

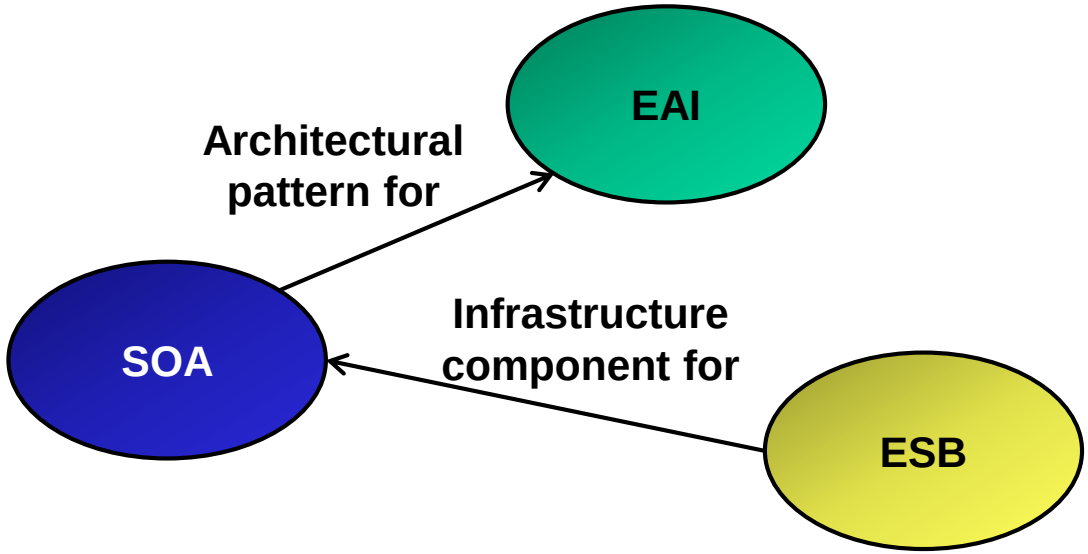
EAI aims at integrating different enterprise applications. Thus EAI is a goal for enterprise architectures.

What is SOA:

SOA is an architectural pattern that aims at concentrating common (business) functionality into distinct services and exposing these on endpoints. Thus SOA is a means or architectural pattern to achieve EAI.

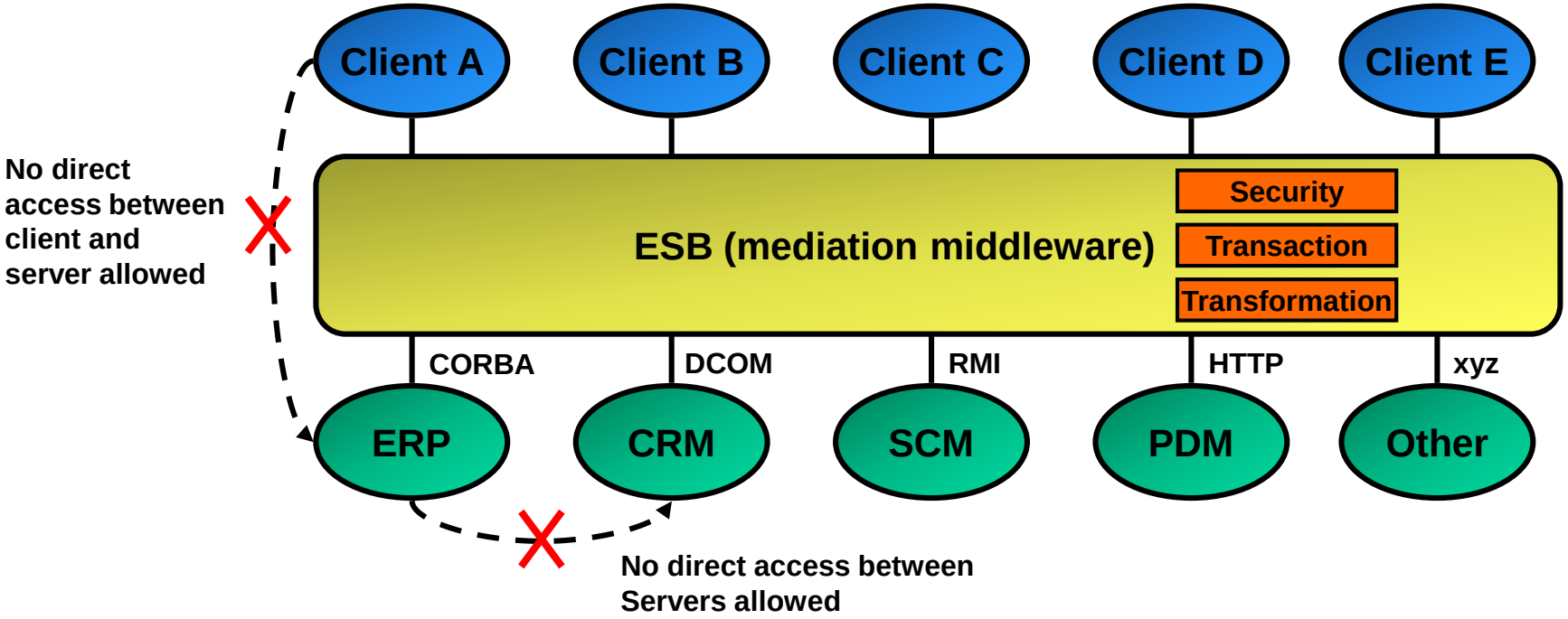
What is ESB?

ESB is an infrastructure component to facilitate SOA (ESB = messaging backbone) and EAI.



1. EAI versus SOA versus ESB (2/3)

Traditional EAI architectures before SOA:



- Integration of applications (common middleware infrastructure through ESB).
- Inefficient for compound services (services calling other services have to pass through the central EAI middleware (ESB) with security checks and transformations for each call).
- Limited reuse of services due to hidden endpoints (classical C/S architecture).

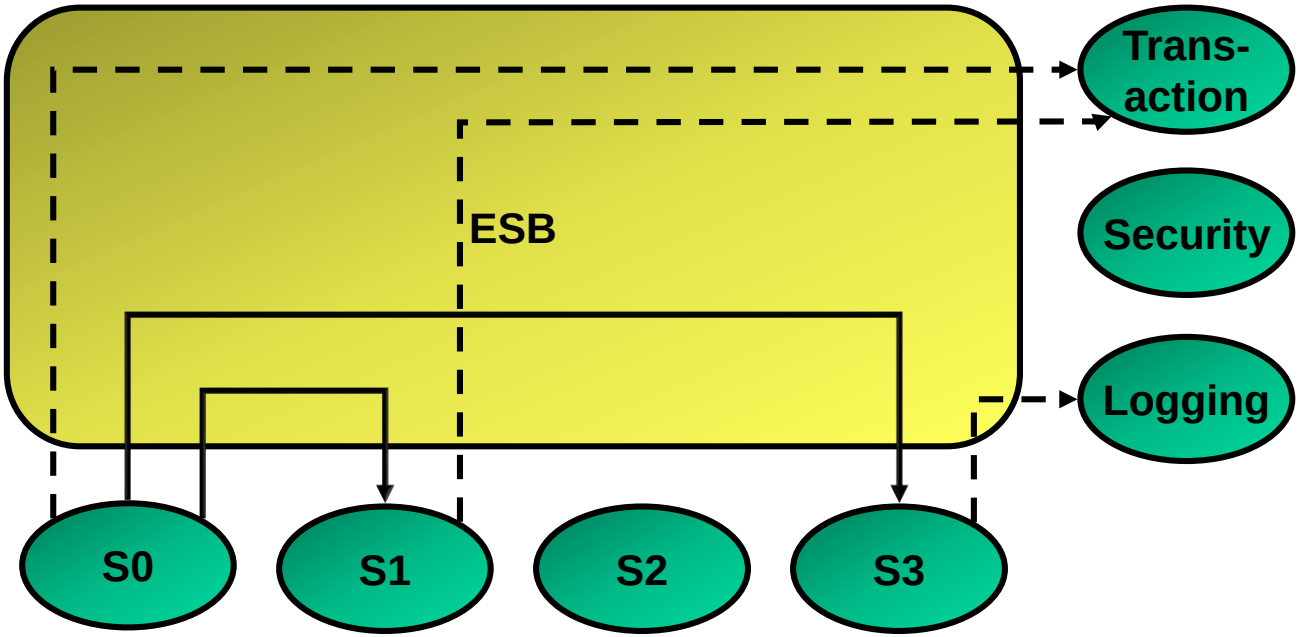
ERP: Enterprise Resource Planning
CRM: Customer Relationship Management
SCM: Supply Chain Management
PDM: Product Data Management

CORBA: Common Object Request Broker Architecture
DCOM: Distributed COM
RMI: Remote Method Invocation

1. EAI versus SOA versus ESB (3/3)

The solution for EAI with SOA:

- **Common functionality is exposed as services.**
- **Endpoints (services) are exposed to be freely called by anyone.**
- **Services may call other services, clients may call services ("liberation" of endpoints).**
- **Services form a service grid with exposed application service endpoints and centralized infrastructure service endpoints.**



2. EAI (1/3)

What is EAI?

EAI is a business need or goal to integrate and couple diverse applications in an enterprise / organization. The benefits of EAI are:

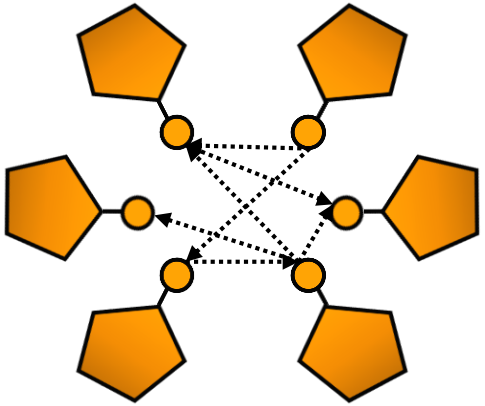
→ Share information between applications (basically connect the different DBs) and keep data consistent.

→ Potentially reduce the technology landscape, reduce heterogeneity (standard interfaces of services mandate the use of standards, applications have less freedom to choose from different DBs, OSs, middlewares etc.).

→ Faster and easier deployment of a new / updated application (interfaces for the integration are defined, middleware technologies are in place).

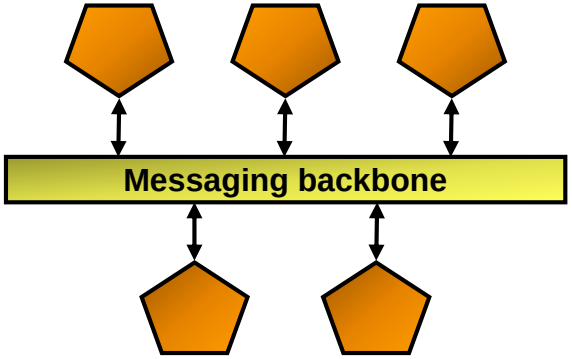
Traditional IT landscape:

n*(n-1)/2 application interface connections.



EAI architecture:

Central communication backbone. with standard interfaces.

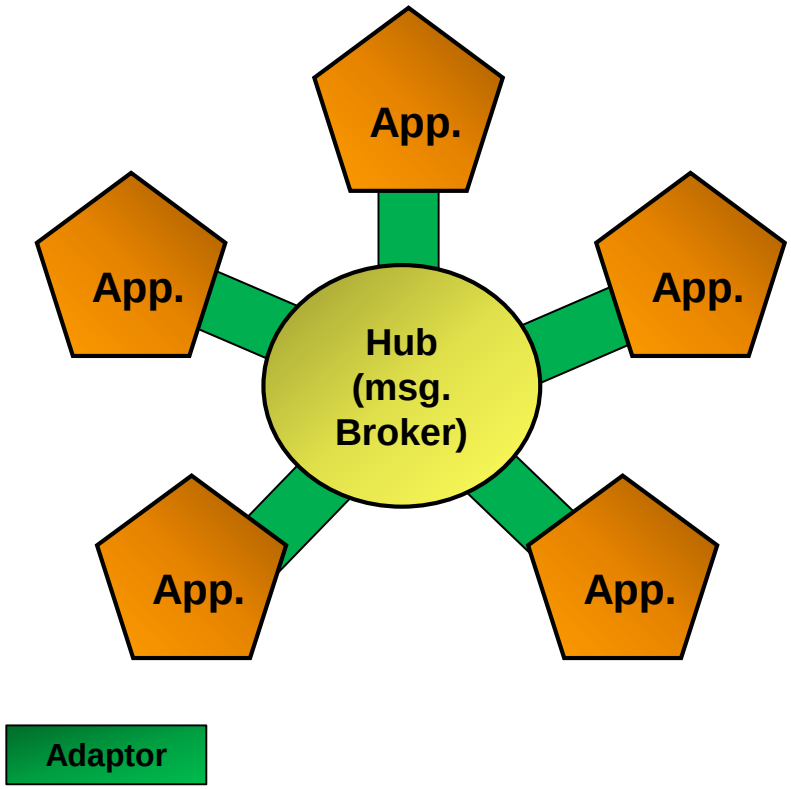


2. EAI (2/3)

Typical EAI topologies:

1. Hub and spoke:

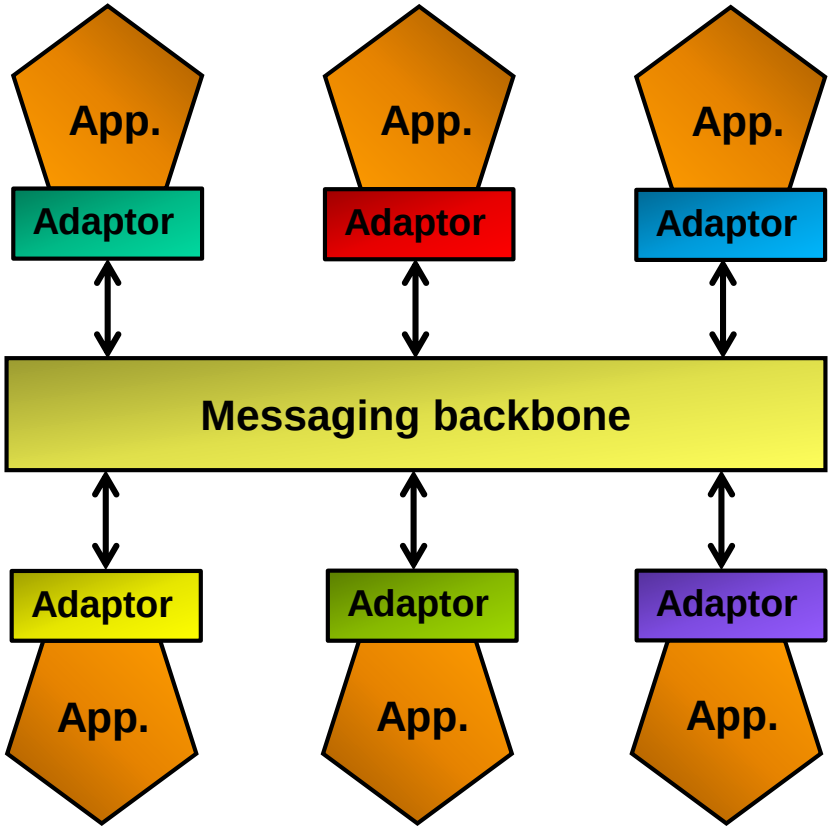
→ EAI system is at the center (hub), interaction with applications via adaptors (=spokes).



2. Bus:

→ EAI system is a bus.

→ Distributed message-oriented communication.



2. EAI (3/3)

EAI building blocks:

EAI can be accomplished in different ways. Most did not prove scalable (e.g. integration at DB level). Use of a centralized broker emerged as the best solution to the integration problem (scalability). This best practice has the following building blocks:

1. Centralized broker:
→ Handles security, access and communication.
→ ESB

2. Data model:
→ Common data model based on a standard data structure. XML has become the de-facto standard.

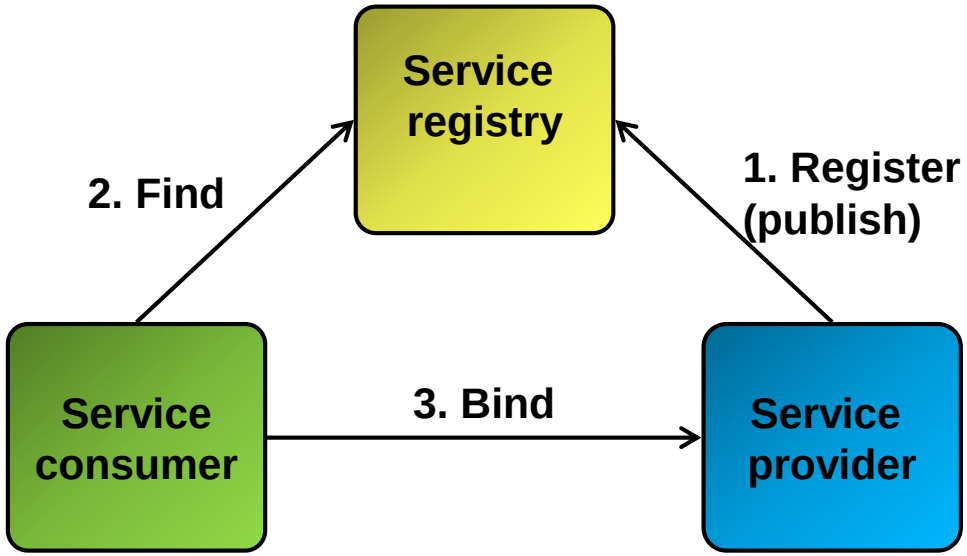
3. Adaptor / connector:
→ Adaptors / connectors connect applications to the central broker.

4. System model:
→ Defines the interface including API and data flow to a component that connects to the central broker. Allows other applications to interact with this component in a standardized way.

3. SOA (1/3)

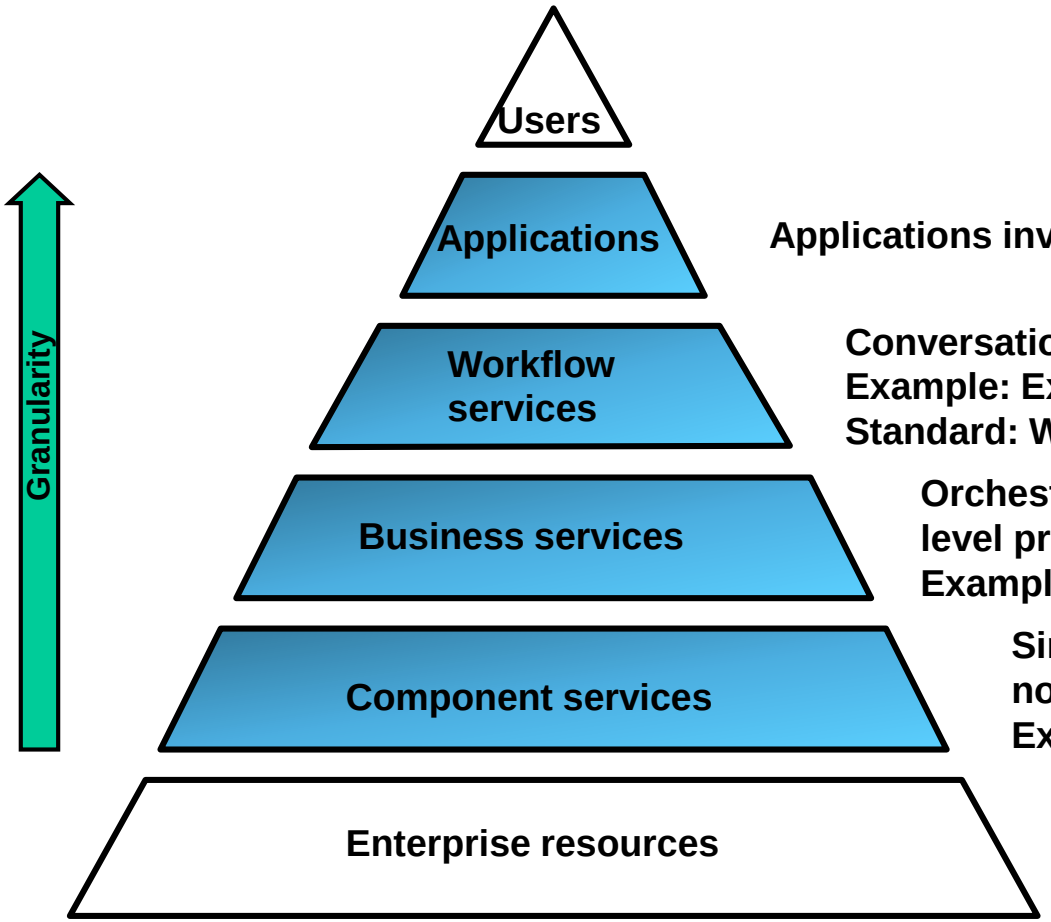
SOA aims to extract common (service) functionality from different applications and expose it on a service endpoint.
In the basic SOA pattern, service consumer, provider and registry are separated into different entities.
The service registry helps decoupling service consumer and provider so that the consumer does not need to know the location of the provider.

The service registry is an optional entity. In smaller deployments running a service registry may be 'overkill'.



3. SOA (2/3)

Services can be exposed at different levels / granularity:
Finding the right granularity is crucial for a successful SOA.
A layering as follows may help in defining / decomposing the service landscape.



Applications invoke workflow services.

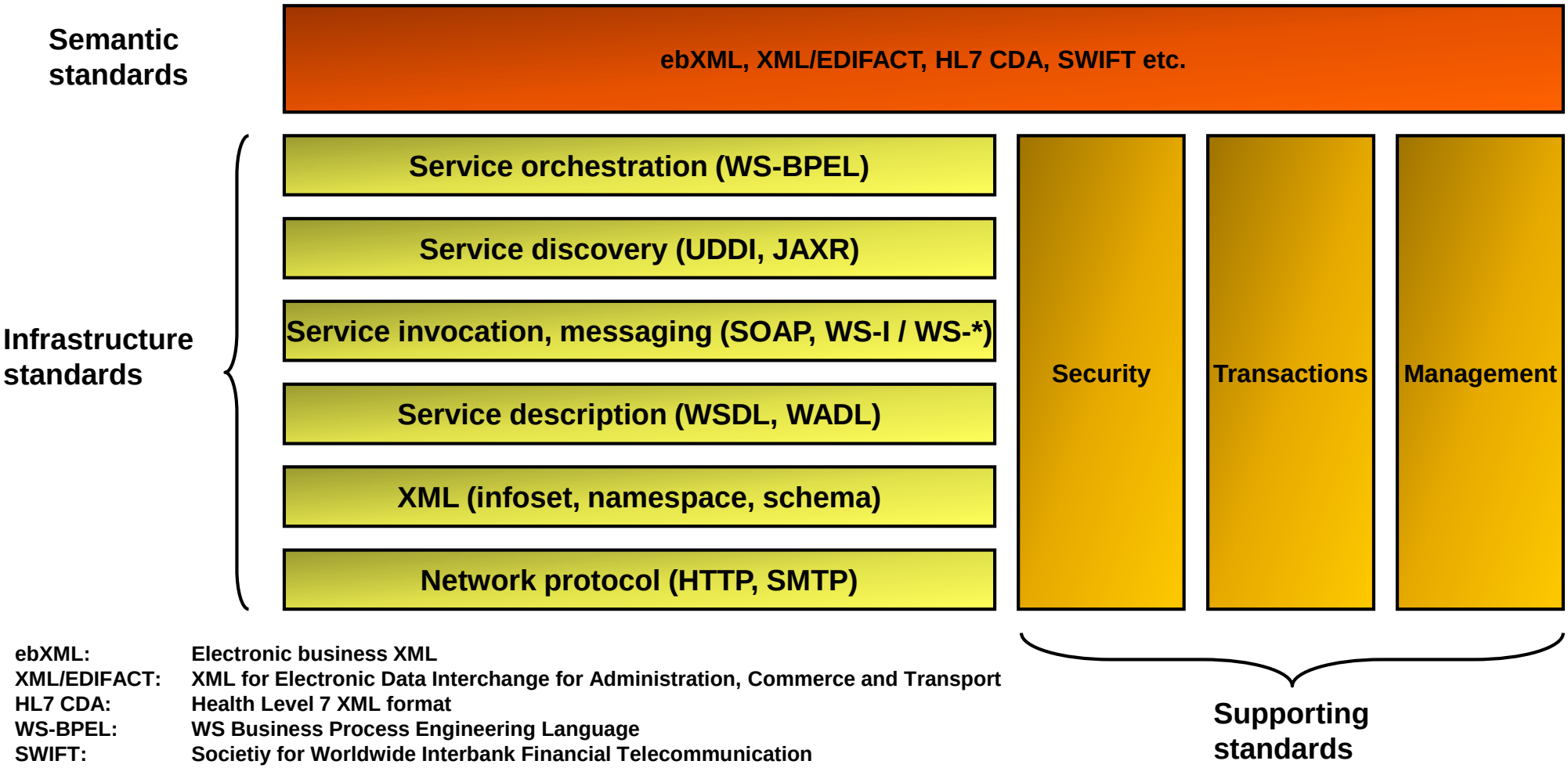
Conversational workflow services have a state.
Example: Expense processing service.
Standard: WS-BPEL

Orchestrates component services into a single business level process.
Example: Submit an expense report.

Simple atomic action on a resource; action does not depend on any other service / component.
Example: Simple access to a DB table.

3. SOA (3/3)

Standards are crucial for SOAs.
These standards may be layered as follows:



ebXML:	Electronic business XML
XML/EDIFACT:	XML for Electronic Data Interchange for Administration, Commerce and Transport
HL7 CDA:	Health Level 7 XML format
WS-BPEL:	WS Business Process Engineering Language
SWIFT:	Societiy for Worldwide Interbank Financial Telecommunication

4. ESB

Enterprise Service Bus is an infrastructure to facilitate SOA.
ESB is basically a messaging backbone / broker which provides the following functions:

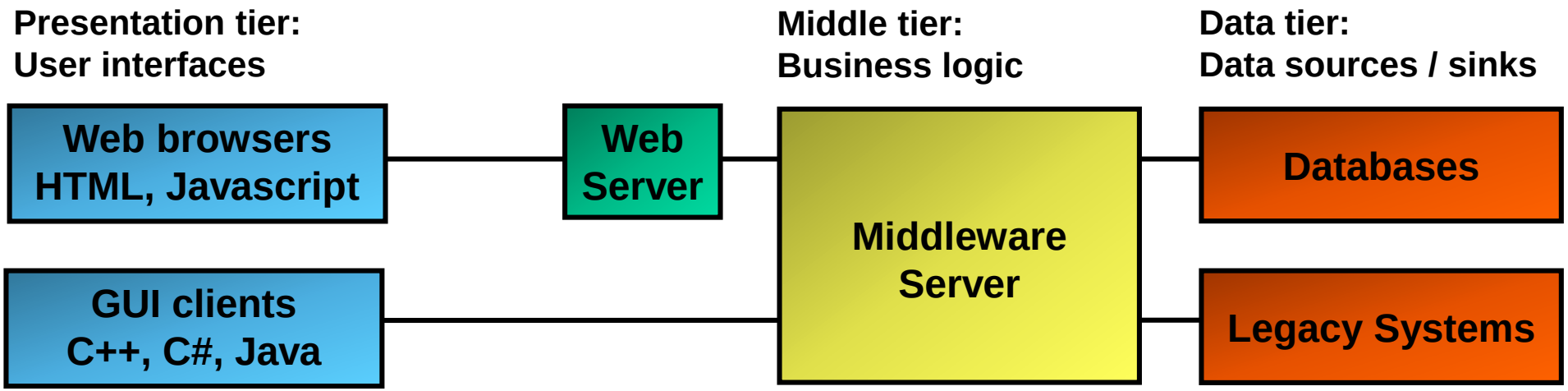
Category	Functions
Invocation	Support for synchronous and asynchronous transport protocols, service mapping (locating and binding).
Routing	Addressability, static/deterministic routing, content-based routing, rules-based routing, policy-based routing.
Mediation	Adapters, protocol transformation, service mapping.
Messaging	Message-processing, message transformation and message enhancement.
Process choreography	Implementation of complex business processes.
Service orchestration	Coordination of multiple implementation services exposed as a single, aggregate service.
Complex event processing	Event-interpretation, correlation, pattern-matching.
Other quality of service	Security (encryption and signing), reliable delivery, transaction management.
Management	Monitoring, audit, logging, metering, admin console, BAM.

Source: http://en.wikipedia.org/wiki/Enterprise_service_bus

Examples:
Mule
IBM WebSphere ESB
Microsoft BizTalk server

5. N-tier enterprise architecture

"Best practice" for "horizontal" decomposition of an application: 3-tier.
→ Separation of concerns (user interface, business logic and data handling) improves maintainability and extensibility.



6. WS-BPEL (1/2)

What is WS-BPEL (Web Services Business Process Execution Language)?

WS-BPEL is a language to define business processes based on web services.
BPEL binds (web) services together to form larger complex business services.
Thus BPEL is kind of a business programming language.

BPEL provides:

a. Control flow (branch, loop, parallel).
b. Asynchronous correlation.
c. Transaction support.

For writing business programs, the following components are necessary:

1. Programming logic (provided by BPEL).
2. Data types (provided by the XSD of a web service).
3. Input / output (provided by WSDL that defines the web service messages).

WS-BPEL versus BPEL4WS:

BPEL4WS: Original standards by BEA, IBM, MS, SAP and Siebel.
WS-BPEL: Successor to BPEL4WS defined by OASIS (name to comply with WS-* scheme).

WS-BPEL and BPMN (Business Process Modelling Notation):

BPMN defines the (graphical) notation for business process elements while WS-BPEL defines an XML-based business process description language.

6. WS-BPEL (2/2)

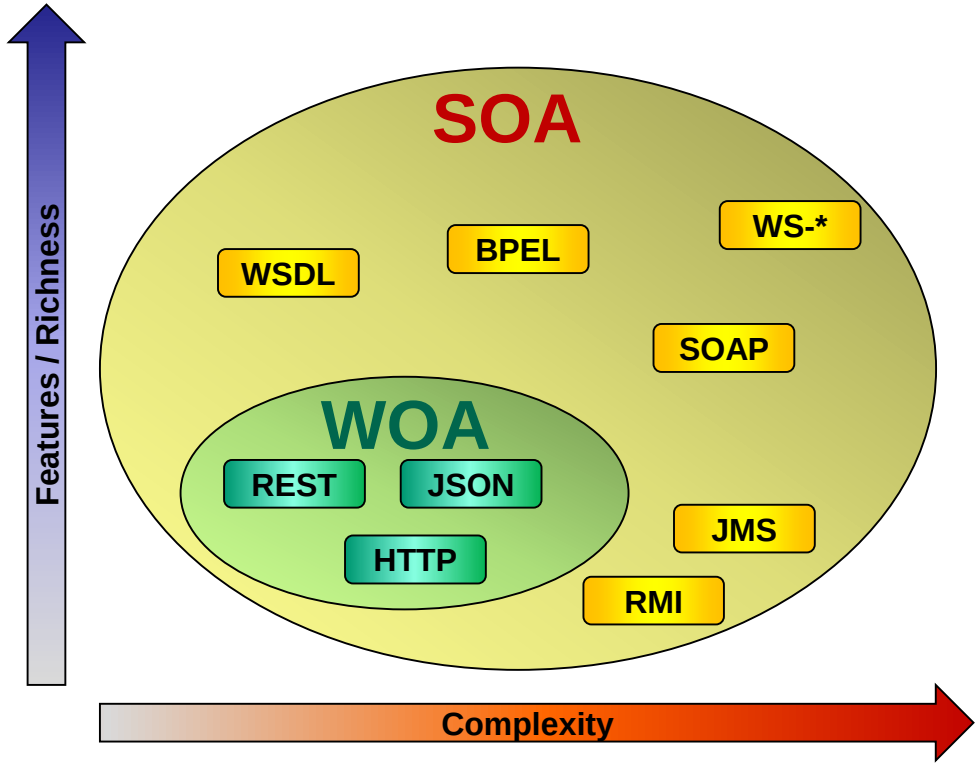
BPEL hello world example:

```
<?xml version="1.0" encoding="UTF-8"?>
<process
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:print="http://www.eclipse.org/tptp/choreography/2004/engine/Print"

  <!--Hello World - my first ever BPEL program -->
    <import importType="http://schemas.xmlsoap.org/wsdl/"
      location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
      namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />
    <partnerLinks>
      <partnerLink name="printService" partnerLinkType="print:printLink" partnerRole="printService"/>
    </partnerLinks>

    <variables>
      <variable name="hello_world" messageType="print:PrintMessage" />
      </variables>
      <assign>
        <copy>
          <from><literal>Hello World</literal></from>
          <to>.value</to>
        </copy>
      </assign>

    <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />

</process>
```

Source: http://www.eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.html

7. WOA

Web Oriented Architecture is a concept that extends or simplifies SOA through the use of REST and POX (Plain Old XML).
WOA / REST is simply another (simpler?!) approach to SOA.



Source: http://www.zdnet.com/blog/hinchcliffe/the-soa-with-reach-web-oriented-architecture/27

POX: Plain Old XML (like POJO, but with XML)
JSON: Javascript Object Notation (more compact alternative to XML)
BPEL: Business Process Execution Language