Service Oriented Architecture: Principles and Practice

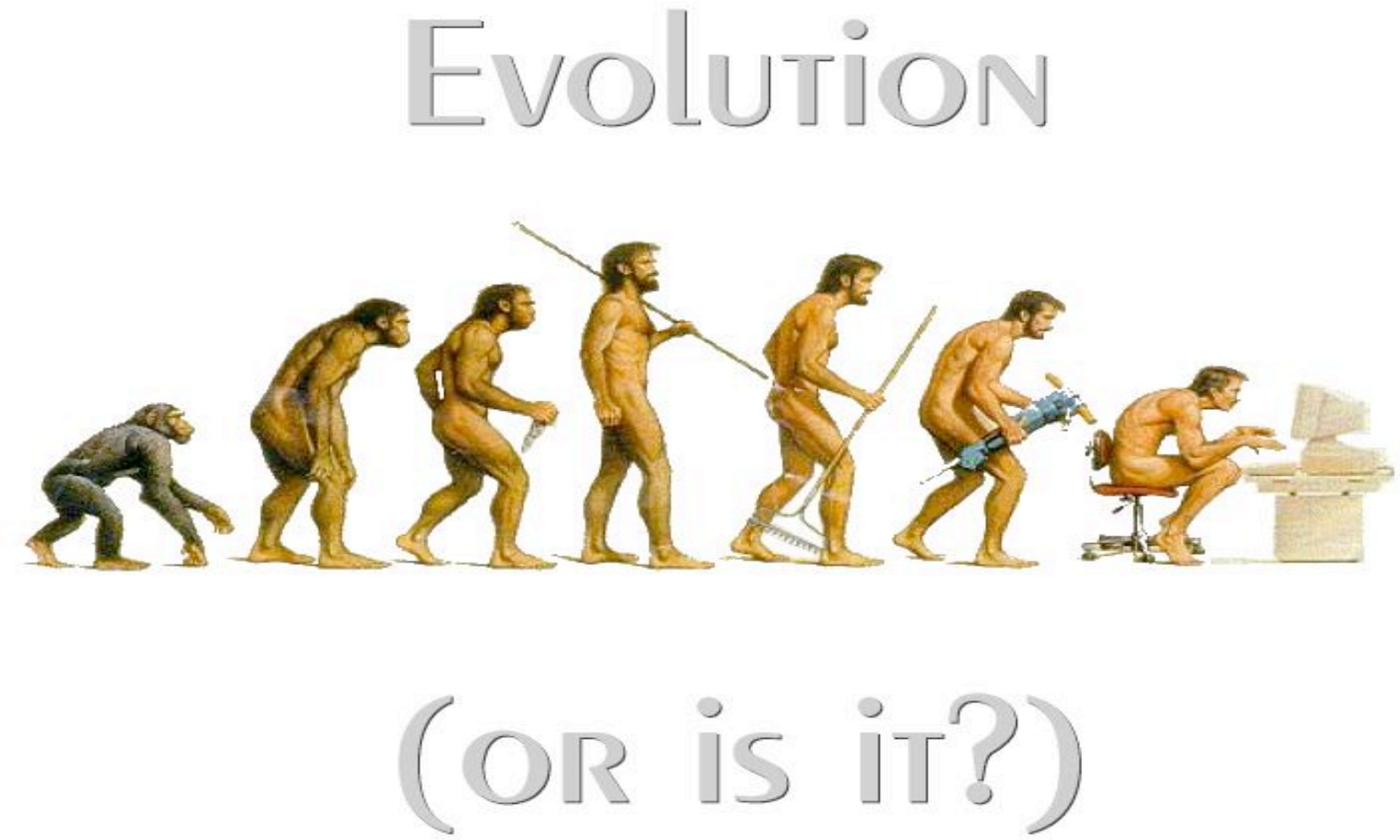
Dr Mark Little

Technical Development Manager, Red Hat

Overview

- **Evolution of distributed systems**
- **SOA principles**
  - Relationship to Web Services
  - The Enterprise Service Bus
- **Implementation experience**
- **Advanced concepts**
  - Session management
  - Transactions
- **Standards**

Origins



Remote Procedure Call

- **First used in Unix back in the 1970's to aid distributed development**
  - Try to make distribution opaque
  - Leverage a well known pattern: local procedure calls
- *Integrated Systems Architecture (ISA), Open Network Computing (ONC)*, the *Open Software Foundation Distributed Computing Environment (OSF/DCE)*, and the *Object Management Group Common Object Request Broker Architecture (CORBA)*

Principles of stub generation

- **A distributed application consists of:**
  - The calling process (client)
  - Remote server process
    - Executes the requested operation
- **The physical components are:**
  - The client
  - The client stub
  - The transport
  - The server stub
  - The server

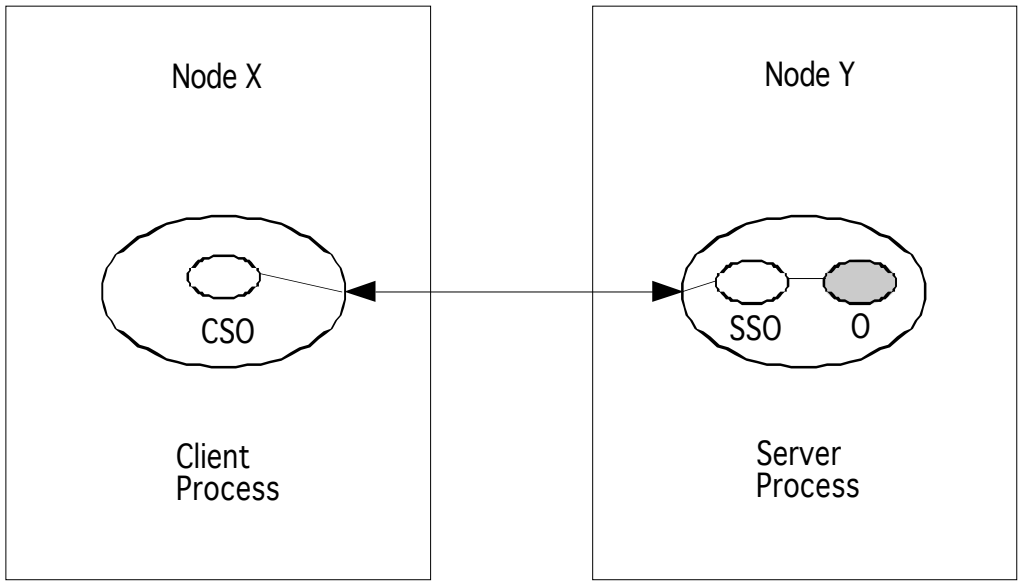
The Stub Generator

- The client and server are designed and implemented as if the application was to execute in a traditional centralized environment
- The client and server stubs hide the underlying distribution
- The production of the stubs can be automated by the use of a *Stub Generator*
  - **This parses a description of the interface between the client and the server**
  - ***Interface Definition Language***

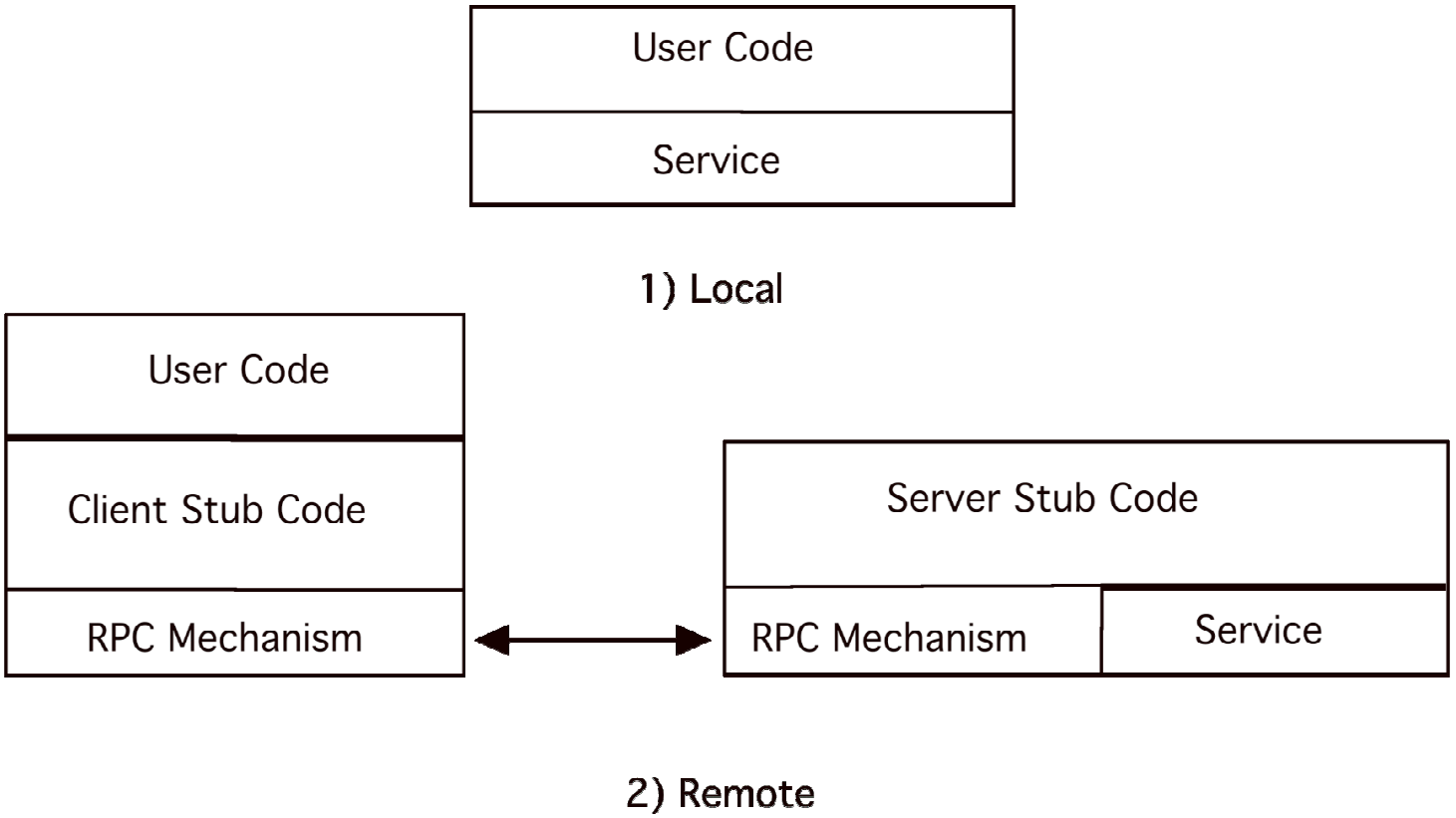
IDL example

```
module CosConcurrencyControl
{
    enum lock_mode { read, write, upgrade, intention_read, intention_write };

    exception LockNotHeld {};

    interface LockCoordinator
    {
  void  drop_locks  ();
    };

    interface LockSet
    {
  void  lock  (in  lock_mode  mode);
  boolean  try_lock  (in  lock_mode  mode);

  void unlock  (in  lock_mode  mode)  raises  (LockNotHeld);

  void  change_mode  (in  lock_mode  held_mode,
          in  lock_mode  new_mode)  raises  (LockNotHeld);

  LockCoordinator  get_coordinator  (in  CosTransactions::Coordinator  which);
    };


    interface TransactionalLockSet
    {
  void lock (in CosTransactions::Coordinator current, in lock_mode mode);
  boolean try_lock (in CosTransactions::Coordinator current, in lock_mode
mode);

  void unlock (in CosTransactions::Coordinator current, in lock_mode mode)
raises (LockNotHeld);

  void change_mode (in CosTransactions::Coordinator current,
            in  lock_mode  held_mode,
          in  lock_mode  new_mode)  raises  (LockNotHeld);

  LockCoordinator  get_coordinator  (in  CosTransactions::Coordinator  which);
    };

    interface LockSetFactory
    {
  LockSet  create  ();
  LockSet  create_related  (in  LockSet  which);

  TransactionalLockSet  create_transactional  ();
  TransactionalLockSet  create_transactional_related  (in  TransactionalLockSet
which);
    };
};
```

Stub example



Local versus remote



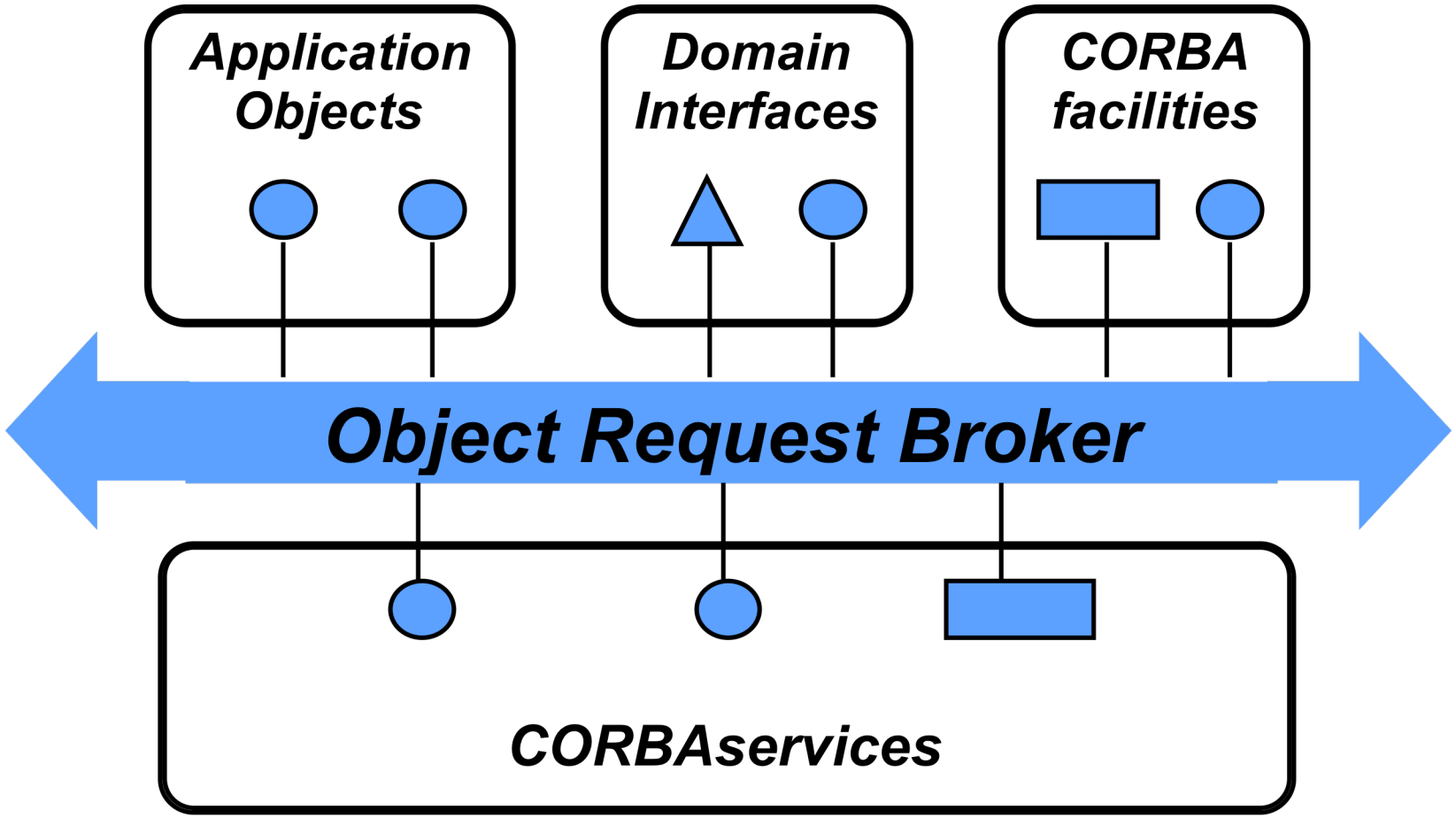
Stub generation problems

- **Machine heterogeneity**
  - Byte ordering, floating point representation
- **Parameter passing semantics and types**
  - Call by reference, call by value
  - Restrict the types that can be passed
- **Self referential structures**
  - Linked lists, circular data structures
- **Failures**
  - Independent failure modes of client and service

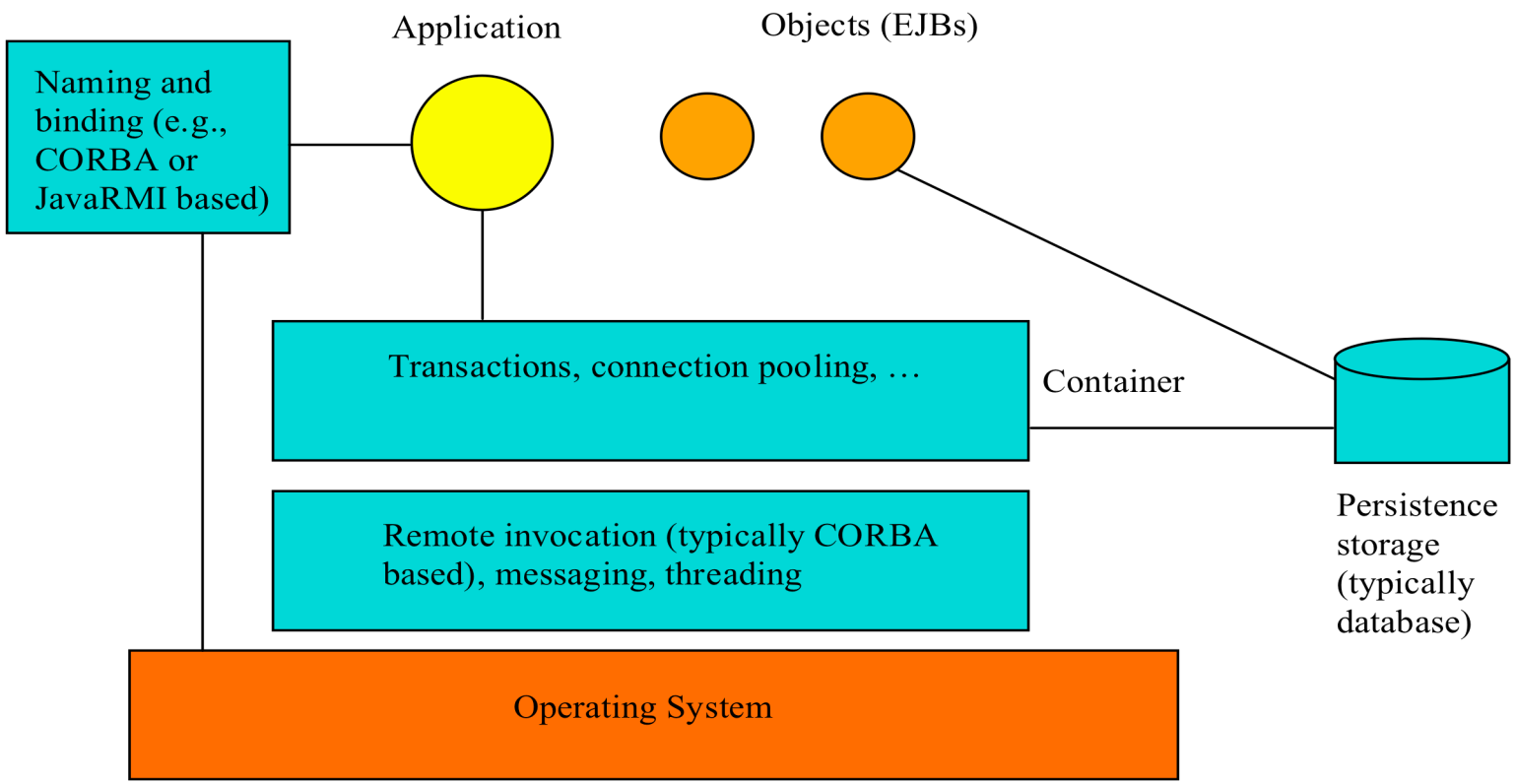
Distribution opacity or transparency?

- **Prior to RPC, distribution was explicit**
  - UDP/IP was the transport
  - TCP/IP was yet to really take off
- **Distribution was hidden by RPC**
  - Distributed object systems are just a logical extension
- **Makes it easier for many developers in the short term**
  - But as scale of systems increase, different failure models make it harder to achieve
- **Distribution transparency is back in vogue**

CORBA middleware



J2EE


Application
Objects (EJBs)
Naming and binding (e.g., CORBA or JavaRMI based)
Transactions, connection pooling, …
Container
Remote invocation (typically CORBA based), messaging, threading
Persistence storage (typically database)
Operating System

What is SOA?

- *An SOA is a specific type of distributed system in which the agents are "services" (http://www.w3.org/TR/2003/WD-ws-arch-20030808/#id2617708*
- Adopting SOA is essential to delivering the business agility and IT flexibility promised by Web Services.
- But SOA is not a technology and does not come in a shrink-wrapped box
  - **It takes a different development methodology**
  - **It's not about exposing individual objects on the "bus"**

Services

- **Services represent building blocks for applications**
  - Allowing developers to organize their capabilities in ways that are natural to the application and the environment in which they operate.
- **A Service provides information as well as behaviour and it does not expose implementation (back-end) choices to the user.**
  - Furthermore a service presents a relatively simple interface to other services/users.

Tightly coupled

- Client and server technologies based on RPC
  - **Hide distribution**
  - **Make remote service invocation look the same as local component invocation**
- Unfortunately this *tightly coupled* applications
  - **Changes to the IDL require re-generation of stubs**
    - **And dissemination of new code**
    - **Or errors will occur during interactions**
  - **Such applications can be brittle**
    - Hard to control the infrastructure as needed
    - No quiescent period

Loosely coupled

- SOA is an architectural style to achieve *loose coupling*
  - **A service is a unit of work done by a service provider to achieve desired end results for a consumer.**
- SOA is deliberately not prescriptive about what happens behind service endpoints
  - **We are only concerned with the transfer of structured data between parties**
- SOA turns business functions into services that can be reused and accessed through standard interfaces.
  - **Should be accessible through different applications over a variety of channels.**

But …

- **There are degrees of coupling and you should choose the level that is right for you**
- **At the one extreme**
  - Defining specific service interfaces, akin to IDL
    - Easier to reason about the service
    - Limits the amount of freedom in changing the implementation
- **At the other extreme**
  - Single operation (e.g., doWork)
    - More flexibility is changing the implementation
      - Well, almost …
    - More difficult to determine service functionality a priori
      - Need more service metadata

Uniform interface versus specific

- **The same requirements are present throughout the stack**
  - Split differently between the infrastructure and the “application”
- **Uniform allows for generic infrastructural support**
  - Caching, extremely loose coupling
    - Web
  - Can push more requirements on to the “developer”
- **Specific allows for more limited generic support**
  - Targeted caching, application semantics
  - Impacts less on the “developer” but may cost in terms of coupling

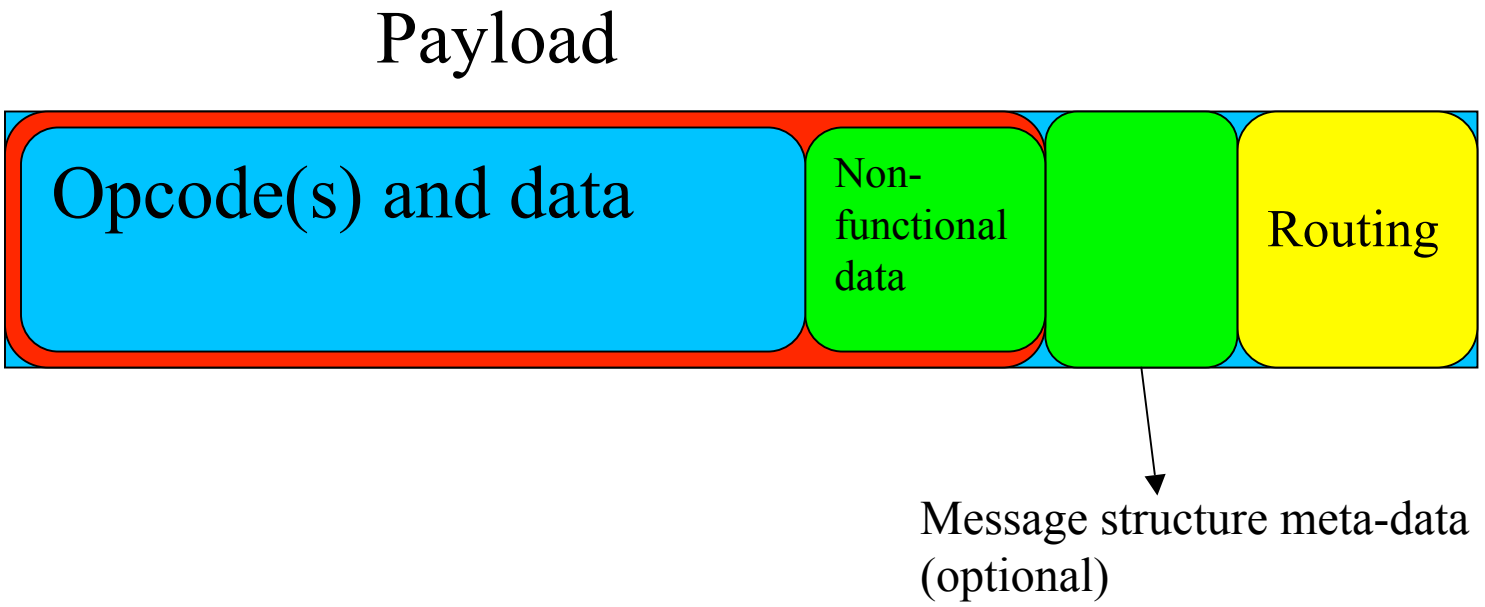
Data Loose Coupling

- SOA says nothing directly about data or transport transformation
  - **The mismatch between data representations will limit loose coupling between the Service provider and consumers**
- What is needed is an effective solution to decouple the provider and consumer data and protocol representations
  - **The consumer and provider must be isolated from knowledge of each other's data formats.**
- The service should be concerned with the semantic meaning of data and not how that data is represented or structured
  - **The SOA approach to providing this loose coupling is by separating the translation requirement into a separate service that can act as an intermediary between providers and consumers, hiding any differences in data structure and allowing the parties involved in the interactions to function un-changed**

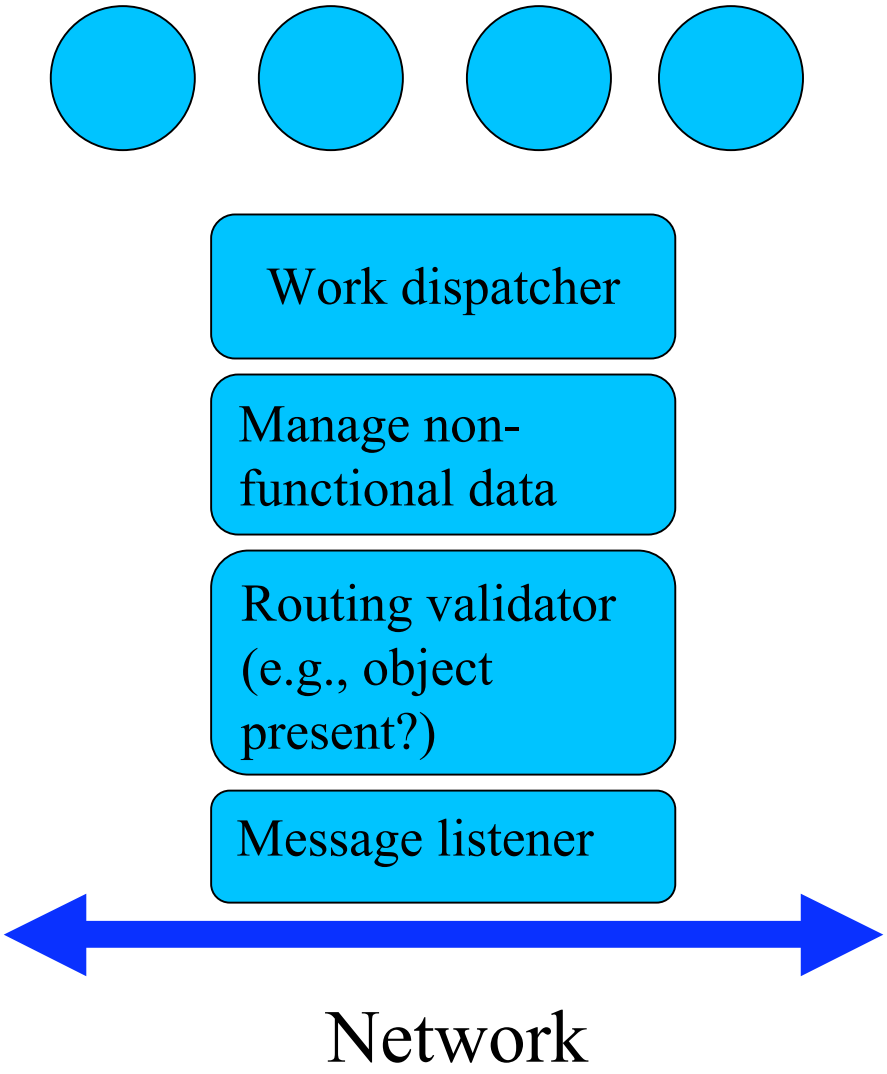
The Service Contract

- **Defines what work/operations/methods the service can accept**
  - May be an amalgamation of back-end implementation details
- **Explicitly part of what an IDL offers**
  - Checked and enforced by the corresponding stubs

Example Message Structure



Service stack example



Conclusions on contracts

- **The Service Contract needs to be defined somewhere**
- **By tying implementation to the contract, changes are enforced by the stubs**
  - Simple for developers
  - More complex for deployers
- **Or contract can be enforced within the service implementation**
  - More complex for developers
  - Simpler for deployers

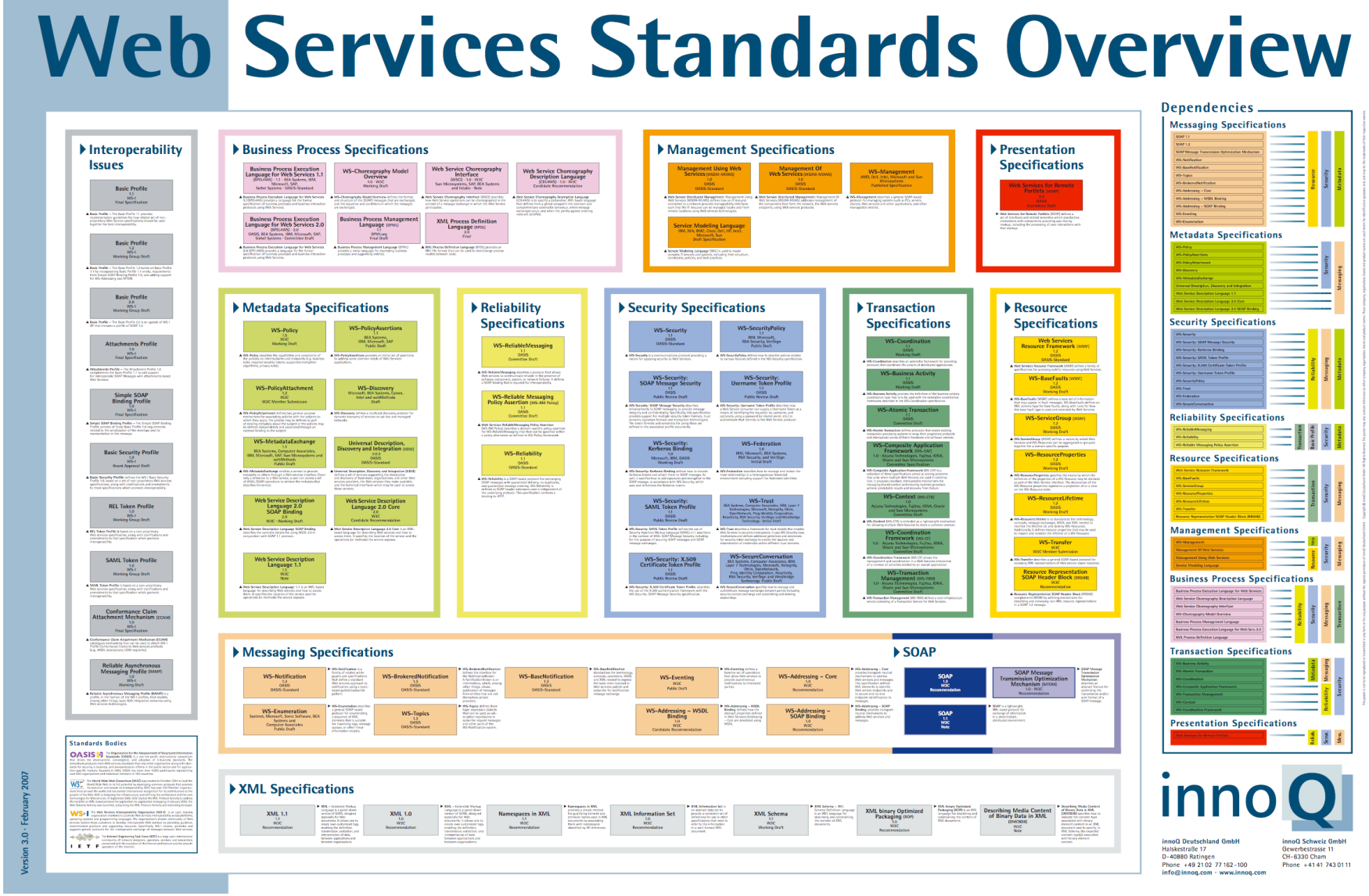
Achieving loose coupling

- **SOA employs two architectural constraints**
  - A small set of simple and ubiquitous interfaces to all participating software agents. Only generic semantics are encoded at the interfaces. The interfaces should be universally available for all providers and consumers.
  - Descriptive messages constrained by an extensible schema delivered through the interfaces. No, or only minimal, system behavior is prescribed by messages. A schema limits the vocabulary and structure of messages. An extensible schema allows new versions of services to be introduced without breaking existing services

What about Web Services?

- **Popular integration approach**
  - XML
  - HTTP
  - Pretty much universal acceptance (see bullets above!)
- **Not specific to SOA**
  - Web Services began life as CORBA-over-HTTP
  - XML-RPC
- **Web Services+SOA gives benefits**
  - Loose coupling
  - Interoperability
  - Enterprise capabilities, e.g., security and transactions

Relationship to WS-*



WS-*

- **WS-Eventing**
  - Provides a protocol that allows web services to subscribe to/register for event notification messages. Sink=receiver/consumer Source=sender/producer
- **WS-Security**
  - Provides SOAP header extensions for client authentication (username/password, x509), message integrity (XML Signature) and message confidentiality (XML Encryption).
- **WS-Addressing**
  - Provides SOAP header extensions to specify:
    Message Destination
    Source Endpoint
    Reply Endpoint
    Fault Endpoint
    Action
    Unique Message ID
    Relationship to other messages
    Parameters

WS-*

- **WS-Context**
  - Provides SOAP header extensions that provide scope to a series of interactions/operations (similar to a HTTP cookie)
- **WS-Coordination**
  - The procedure to establish a coordination context between a set of participants of an activity
- **WS-AtomicTransaction**
  - 2PC – two phase commit semantics
- **WS-BusinessActivity**
  - Long running business activity made up of multiple atomic transactions. Compensation semantics

WS-*

- **WS-Policy**
  - A standard XML schema for identifying:
    - Digital signature required
    - Encryption required
    - Security token to be used (x509, user/pass)
    - Expiration of the message
  - Allows for automatic enforcement of the above
- **MTOM**
  - Message Transmission Optimization Mechanism
  - Provides an encoding mechanism for including binary attachments directly in the SOAP envelope. Similar in concept to MIME and email attachments.

Fortunately …

- **SOA is technology agnostic**
- **WS-* offers the potential for interoperable SOA**
- **But it is just as easy to develop closely-coupled applications in WS-***
- **Most vendor WS-* tools are direct mappings of distributed object tools**
  - SOA != distributed objects with angle brackets
- **A SOA infrastructure should support and encourage SOA principles**
  - Sometimes it is easier said than done

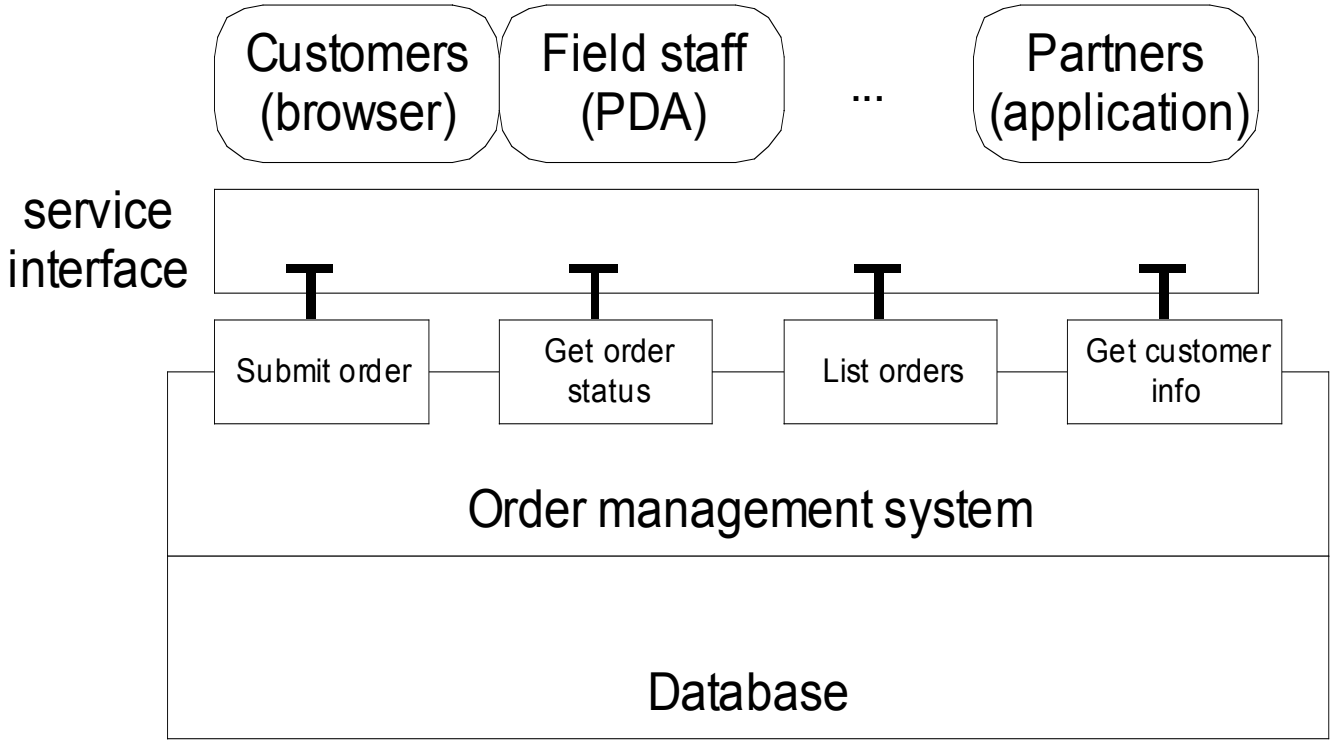
SOA components

- The key components of a Service Oriented Architecture are
  - **The messages that are exchanged**
  - **The agents that act as service requesters and service providers**
  - **The shared transport mechanisms that allow the flow of messages**
- A description of a service that exists within an SOA is essentially just a description of the message exchange pattern between itself and its users

SOA Concepts

- **SOA breaks the 3-tier approach**
  - Inserts a new interface layer to de-couple business logic and back-end implementation choices from presentation layer
- **Turns business functions into services**
  - Can be reused and access through standard interfaces
- **Services form the service layer**
  - Accessed through different applications
  - Over a variety of channels

Example SOA application



Advantages

- **Interoperability**
  - **“The Web Services Interoperability Organization is an open industry effort chartered to promote Web Services interoperability across platforms, applications, and programming languages. The organization brings together a diverse community of Web services leaders to respond to customer needs by providing guidance, recommended practices, and supporting resources for developing interoperable Web services”**
- Efficiency
  - Components within architecture can aid in becoming reusable assets in order to avoid reinventing the wheel
- Standardisation
  - **HTTP, XML**

More advantages

- Business analysts focus on higher order responsibilities in the development lifecycle
- Separating functionality into component-based services that can be tackled by multiple teams enables parallel development
- Quality assurance and unit testing become more efficient; errors can be detected earlier in the development lifecycle
- Development teams can deviate from initial requirements without incurring additional risk
- Functional decomposition of services and their underlying components with respect to the business process helps preserve the flexibility, future maintainability and eases integration efforts
- Security rules are implemented at the service level and can solve many security considerations within the enterprise

Architecture and implementation

Component triad

- **Provider**
  - A provider is an entity that makes a Service available for use by one or more Requestors, optionally facilitating this by publishing details of the Service through a Broker
- **Requestor**
  - A requestor is an entity that uses (consumes) a Service. It may discover the availability and details of this Service via. a Broker or by other means
- **Broker**
  - A broker is an entity that provides directory style registration and lookup service to Providers and potential Requestors

SOAP

- *Simple Object Access Protocol*
  - **"a lightweight protocol for exchange of information in decentralized, distributed environment." SOAP is the key to the *binding operation* between the service requestor and provider**
- Envelope
  - **Encapsulates the message that is exchanged**
    - **Within the envelope there exists a *header* and *body***
      - **header is optional and enables a SOAP message to have additional functionality**
      - **body is not optional and contains the actual message data**
- Encoding Rules
  - **"Defines a serialization mechanism that can be used to exchange instances of application-defined data types." (www.w3.org)**
  - **Built around XML Schema Structures and XML Schema Data Types**

WSDL

- **Provides a way for services to describe themselves**
  - **Important for the *find operation* between the service requestor and discovery agency**
- WSDL document provides
  - The functionality provided by the service
  - Information needed to access the service. Some of this information includes encoding and transport information
  - Location information

WSDL document

- **Contains**
  - Data Types: contains information about the data types needed to access the service
  - Message: an abstract description of the data being accessed or requested
  - Operation: describes what a service can do and is comprised of messages
  - Port Type: used to map a set of operations to one or more endpoints
  - Binding: this allows you to specify a concrete protocol such as HTTP and a data format such as SOAP to bind a port type
  - Port: a combination of a binding and a physical network address
  - Service: a collection of related ports

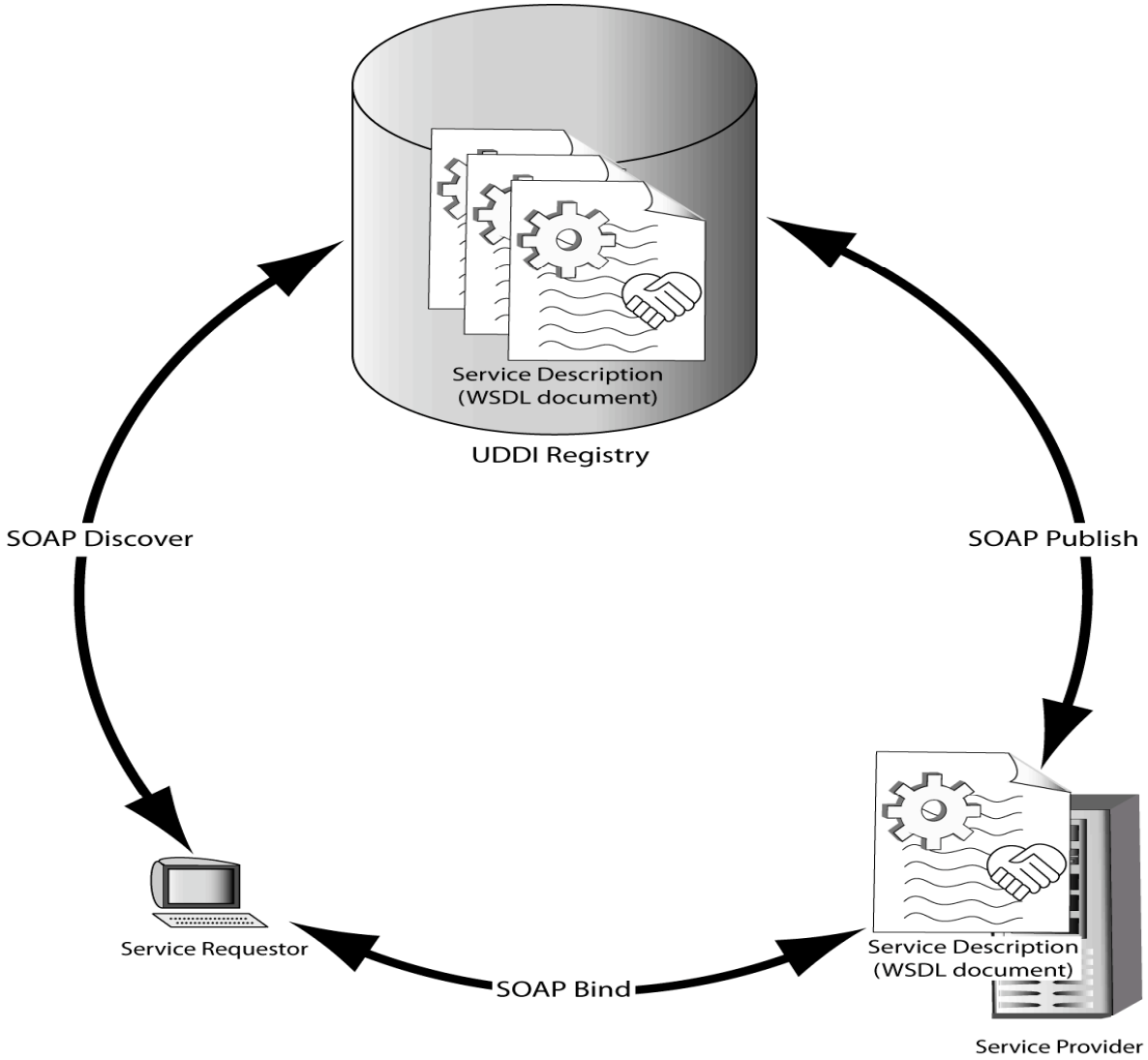
Message Exchange Patterns

- Synchronous request-response
- Synchronous one-way
- Pub-Sub
- Broadcast/multicast

UDDI

- Universal Distribution, Discover and Interoperability (UDDI) registry
  - Directory service for Web Services
- Enables service discovery through queries to the UDDI registry at design time or at run time
- Also allows providers to publish descriptions of their services to the registry
- Typically contains a URL that locates the WSDL document for the web services and contact information for the service provider
  - White pages: contain general information such as the name, address and other contact information about the company providing the service
  - Yellow pages: categorize businesses based on the industry their services cater to
  - Green pages: provide information that will enable a client to bind to the service that is being provided

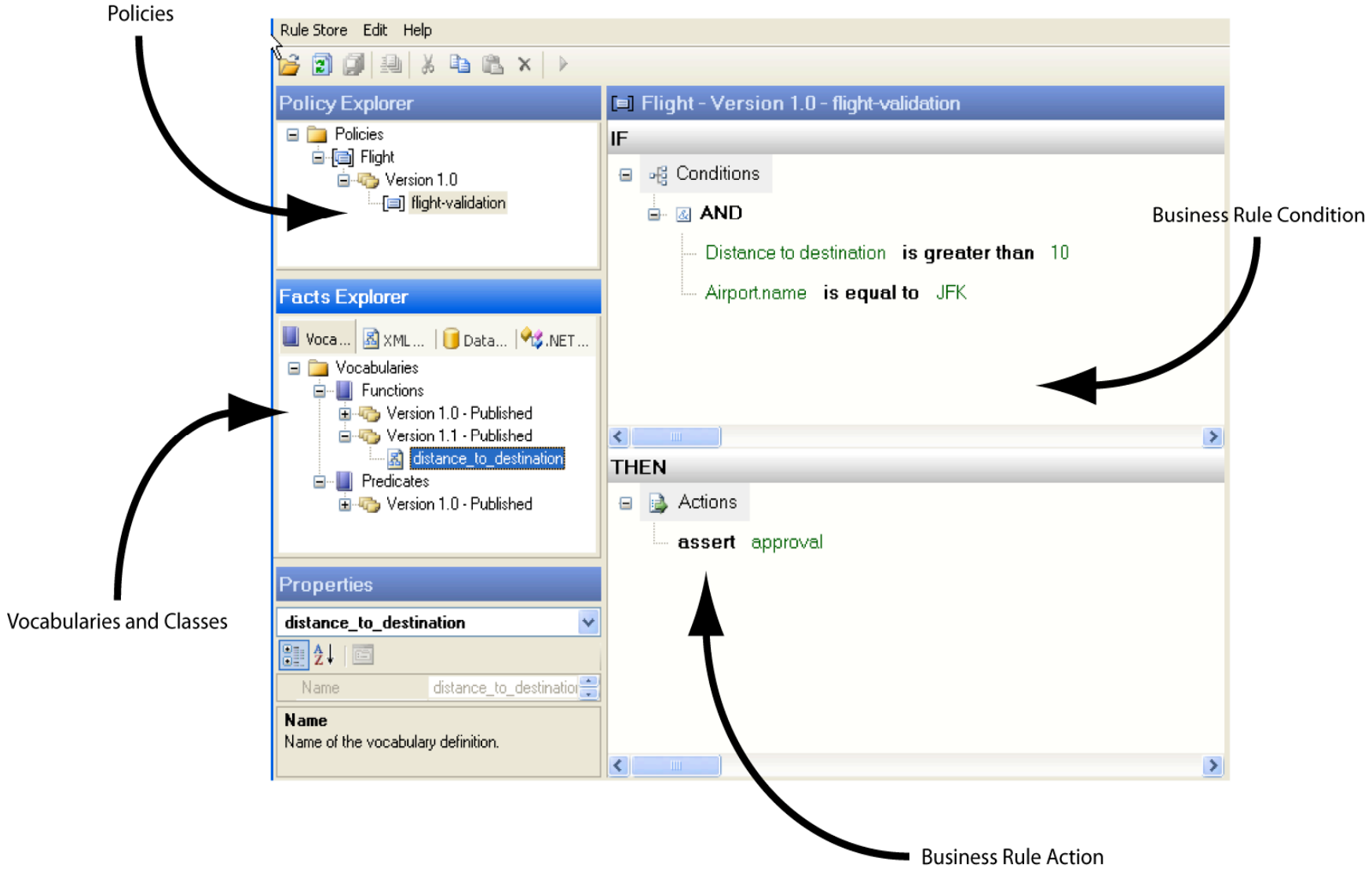
Web Services implementation



Repository

- **Service metadata, which is important for contract definitions**
  - Functional and non-functional aspects
    - Transactional, secure, QoS, …
    - Policies
  - MEPs
    - One-way
    - Request-response
  - Message structure
    - Where data resides
  - Governance
- **Service binaries**
- **Business rules**
- **Workflow tasks or process control information**

The BRMS


Policies
Rule Store Edit Help
Policy Explorer
Policies
Flight
Version 1.0
flight-validation
Facts Explorer
Voca... XML... Data... .NET...
Vocabularies
Functions
Version 1.0 - Published
Version 1.1 - Published
distance_to_destination
Predicates
Version 1.0 - Published
Properties
distance_to_destination
Name
distance_to_destinatio
Name
Name of the vocabulary definition.
Flight - Version 1.0 - flight-validation
IF
Conditions
AND
Distance to destination is greater than 10
Airport.name is equal to JFK
Business Rule Condition
THEN
Actions
assert approval
Vocabularies and Classes
Business Rule Action

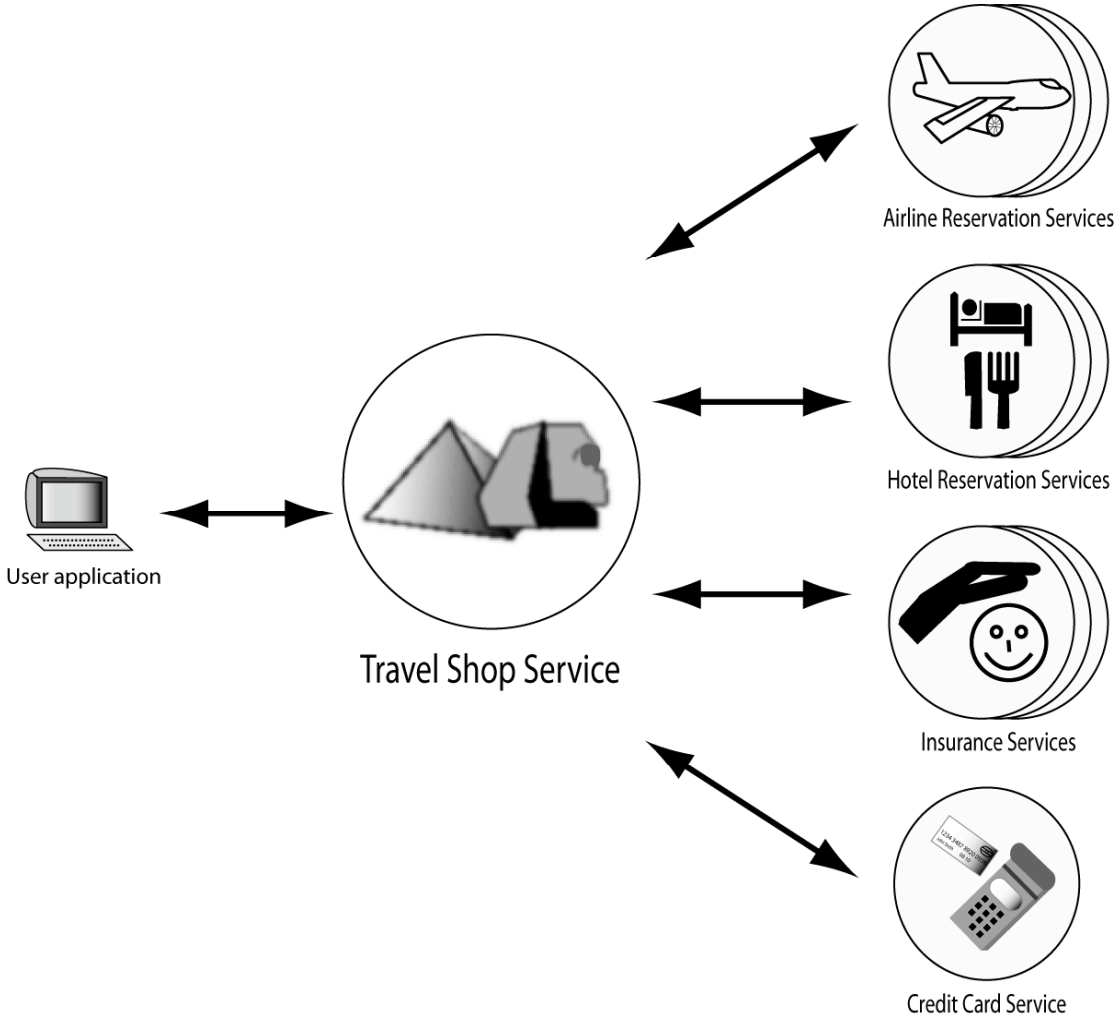
Governance

- **Monitoring and managing distributed systems is complex**
  - No concept of “now”
  - Failures, network partitions etc.
- **SOA is more difficult**
  - No control over infrastructure
  - No notion of trust
  - Indeterminate delays
- **Governance is critically important**
  - What services are running?
  - What are their contracts?
  - What are SLAs?
    - Are they being violated?

Service Lifecycle

- **Services go through four phases:**
  - Model
  - Assemble
  - Deploy
  - Manage
- **Lifecycle management concentrates on the development and deployment of services**
  - Is affected by its relationship with other services
- **Governance brings access control, policies etc. into the way in which services are used within a business process**

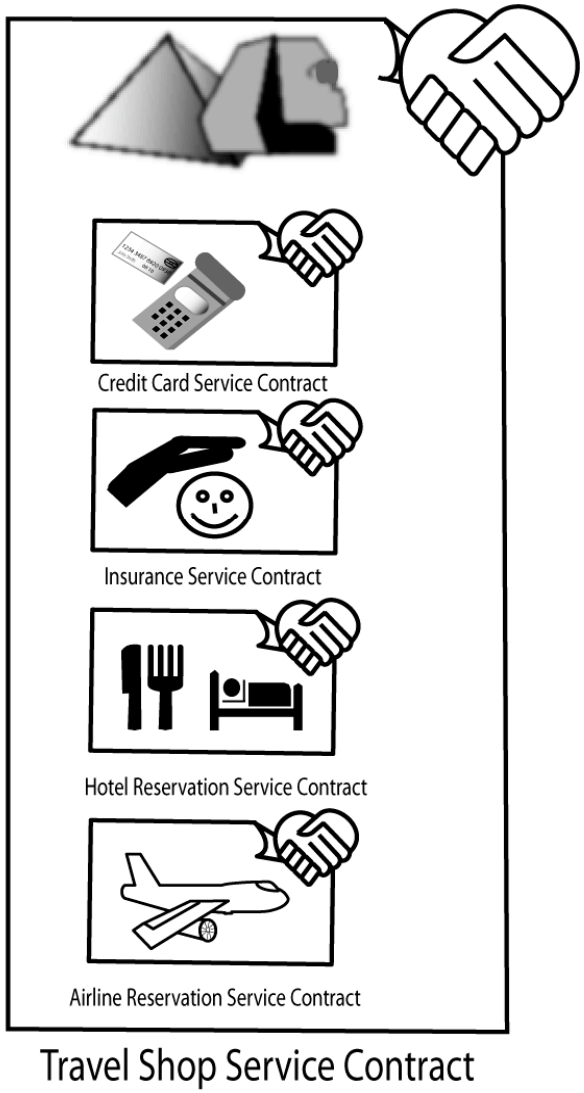
Composite service



Contracts, policies and SLAs

- “Is this service really offering what I want?”)
- “Is this service really doing what it said it would?”
- Composition of services has an affect
- What is a contract?
  - **The service interface**
  - **The messages it can accept, their formats**
  - **A legal contract entered into when using the service**
- The difference between a policy and a contract is that the latter is an agreed policy between service and user

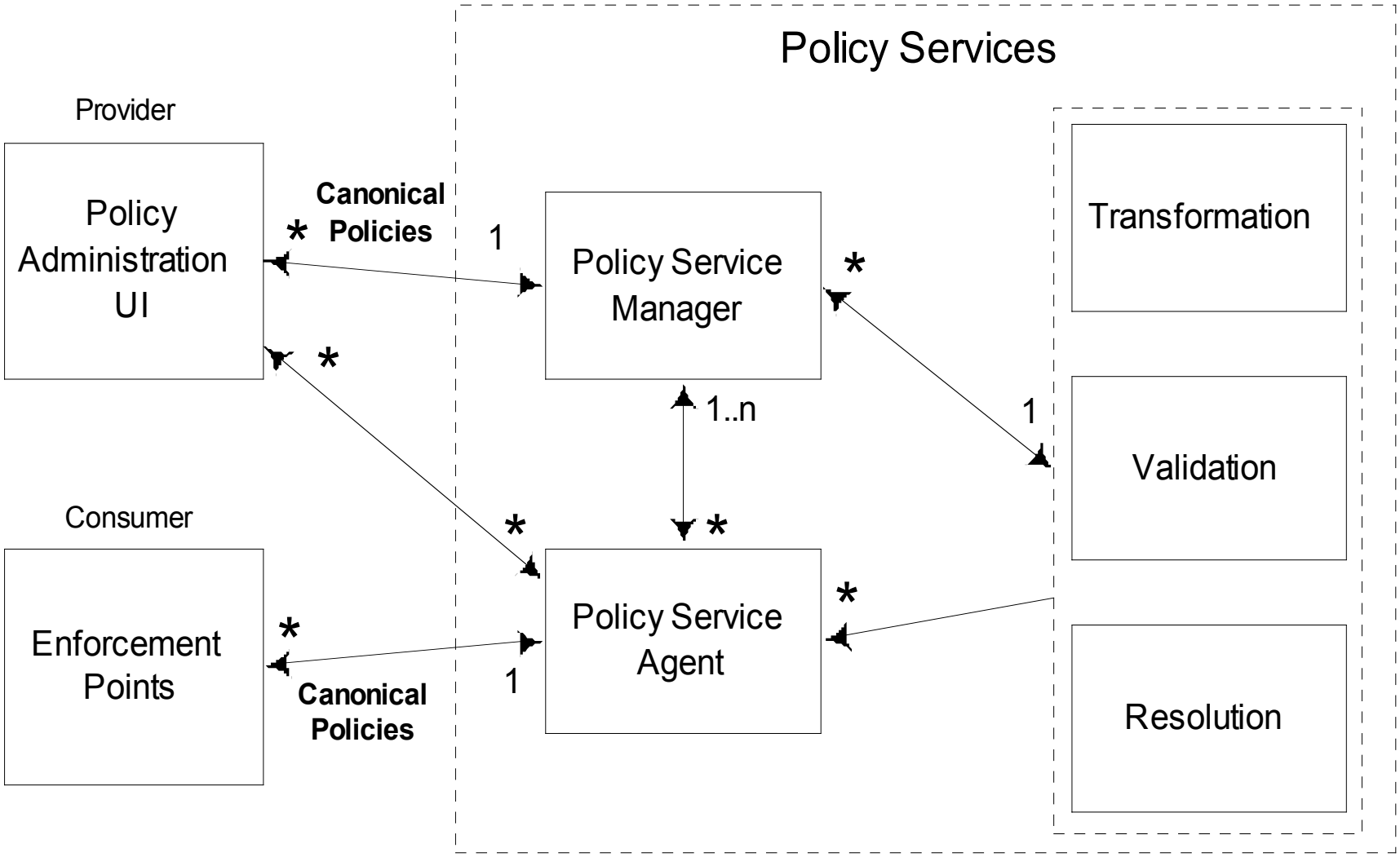
Composite SLA



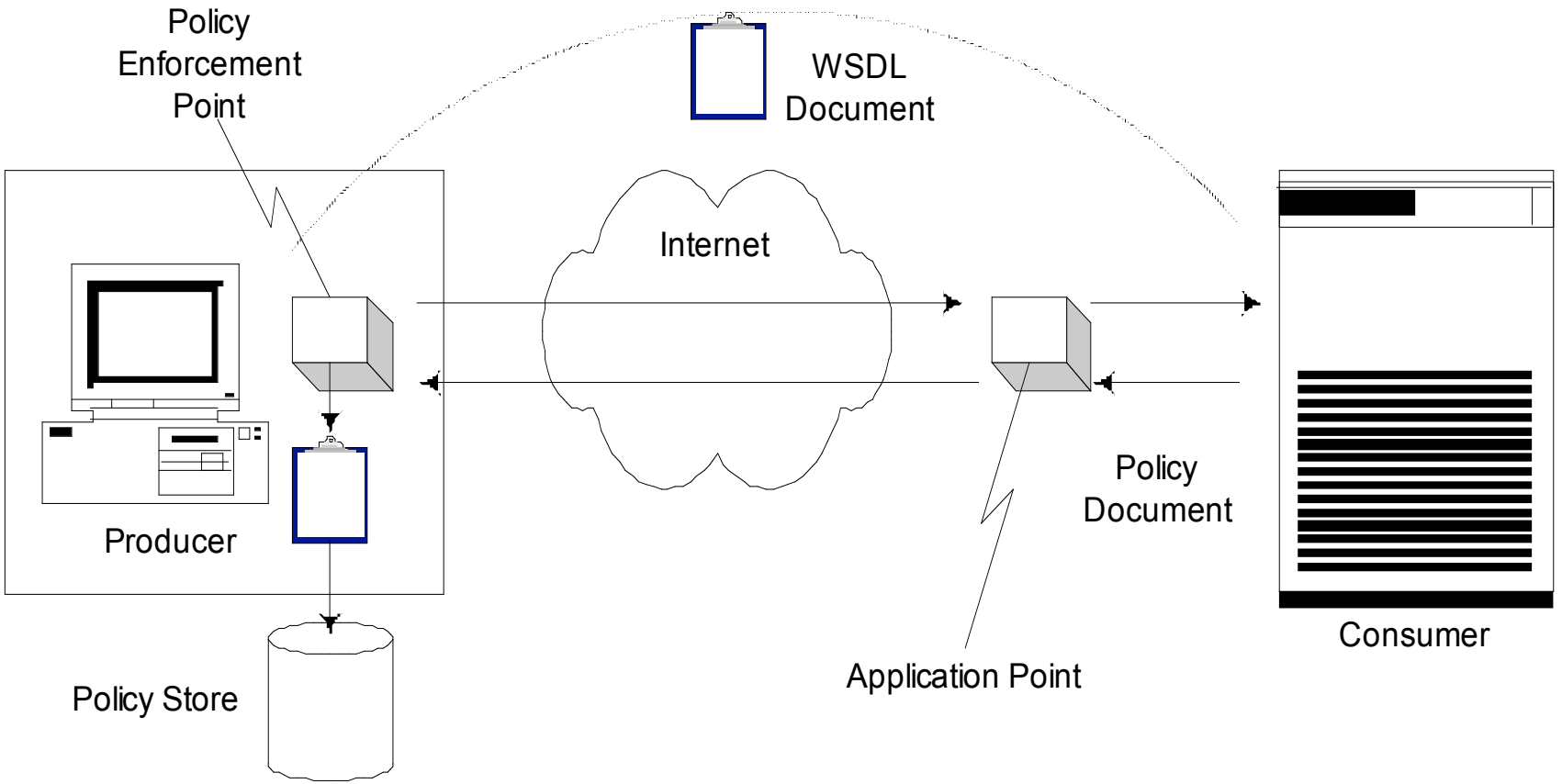
Policies

- **No policy support**
  - The need for policies must be defined outside of the ESB and communicated using ad hoc techniques
- **Definition of policies**
  - Capture and creation of policies at design-time (typically via a graphical interface) and run-time (usually through an intermediary such as a registry)
- **Management of policies**
  - The policies of services to be viewed (either directly by contacting the running service, or indirectly via an intermediary) and updated
- **Enforcement**
  - Policies are verified and enforced by the ESB.
- **Storage**
  - A library of policy types can be built up and shared between services and developers

Policy Management



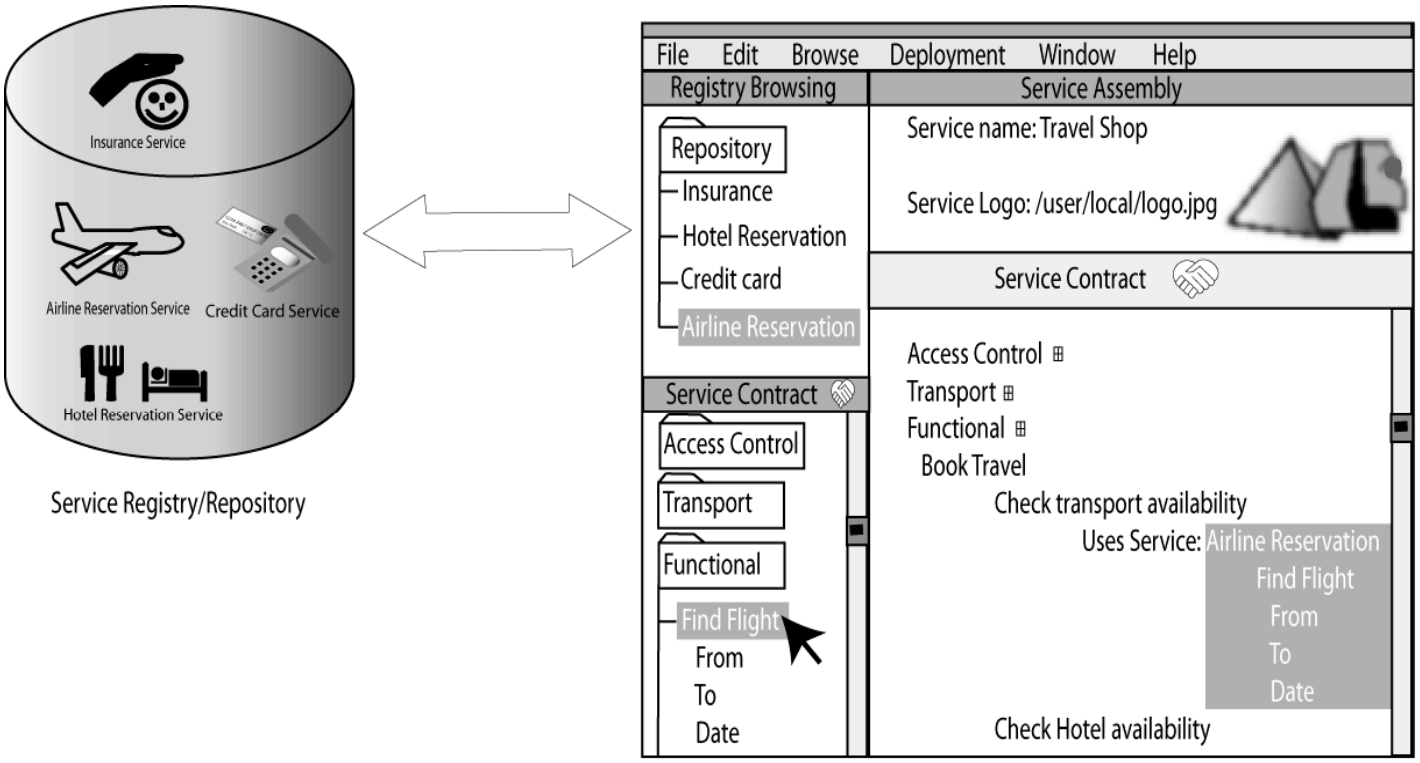
Policy Enforcement



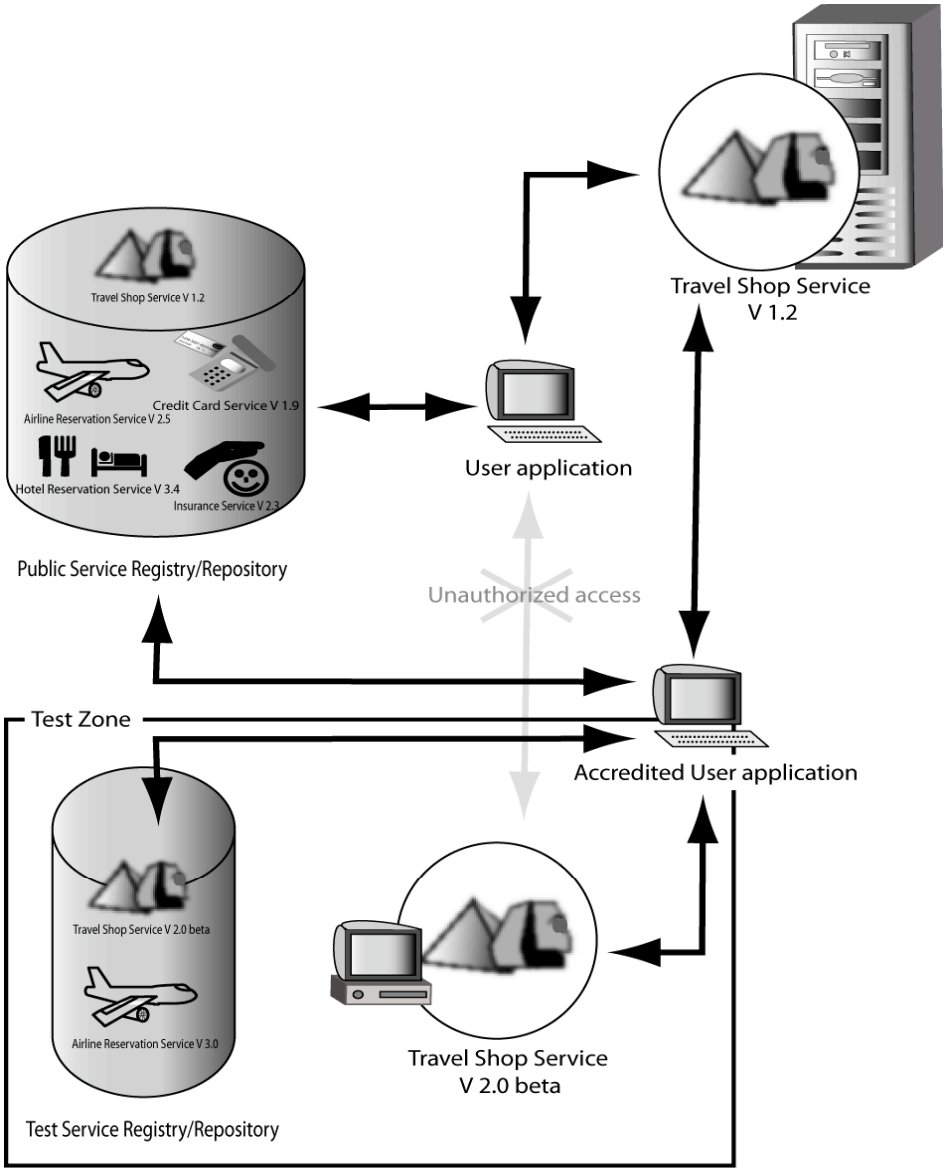
Other meta-data

- Policies that describe configuration/description information for non-functional capabilities of the service, such as those defined by the WS-Security or WS-TX policies, for configuring low-level security and transactional aspects of the service.
- Policies that are markers for compliance or compatibility with certain standards or specifications, such as support for WS-Addressing or compliance with the WS-I basic profiles.
- Policies that represent constraints that must be fulfilled, such as SLAs or contractual obligations.

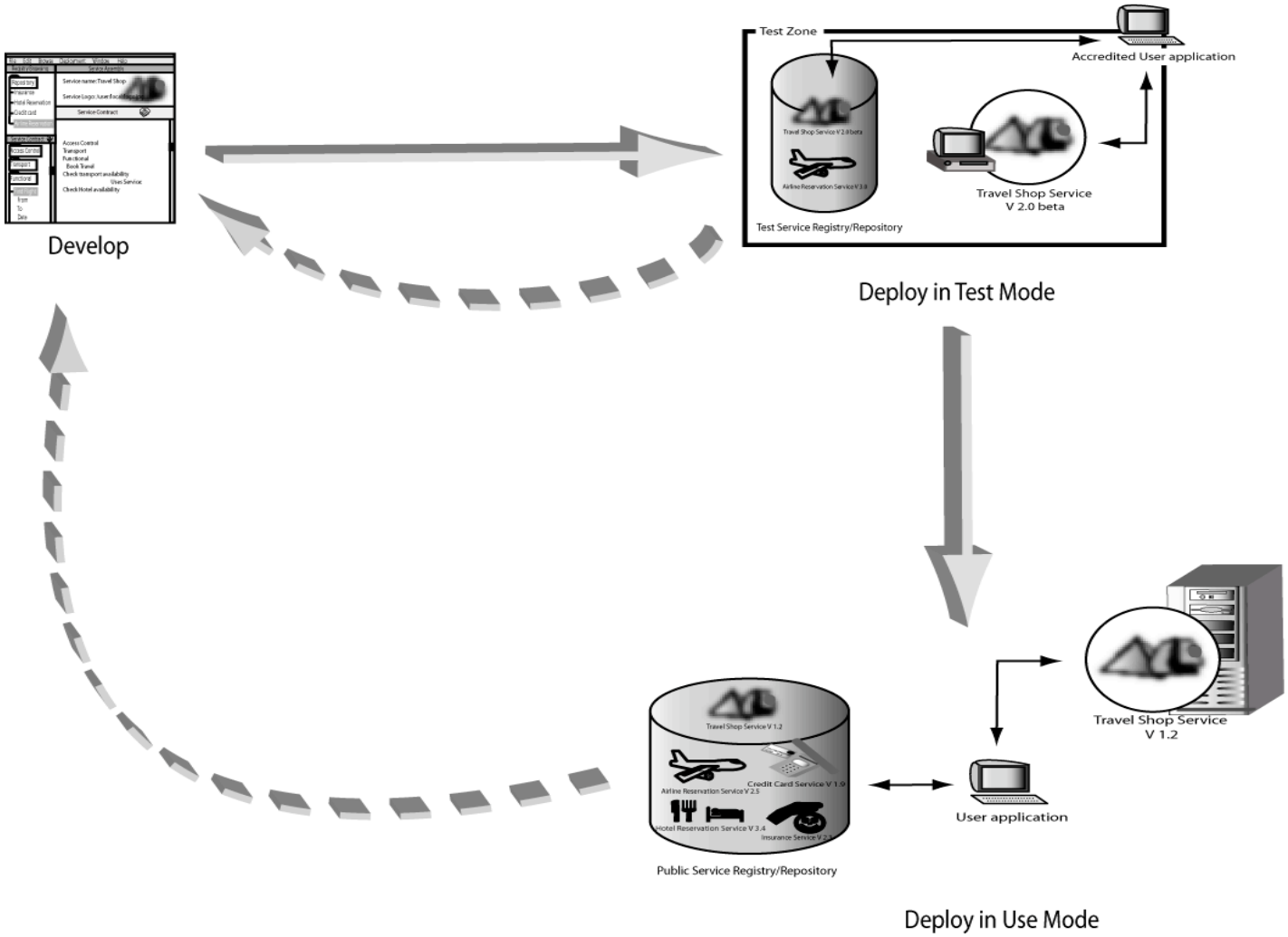
Design-time service discovery



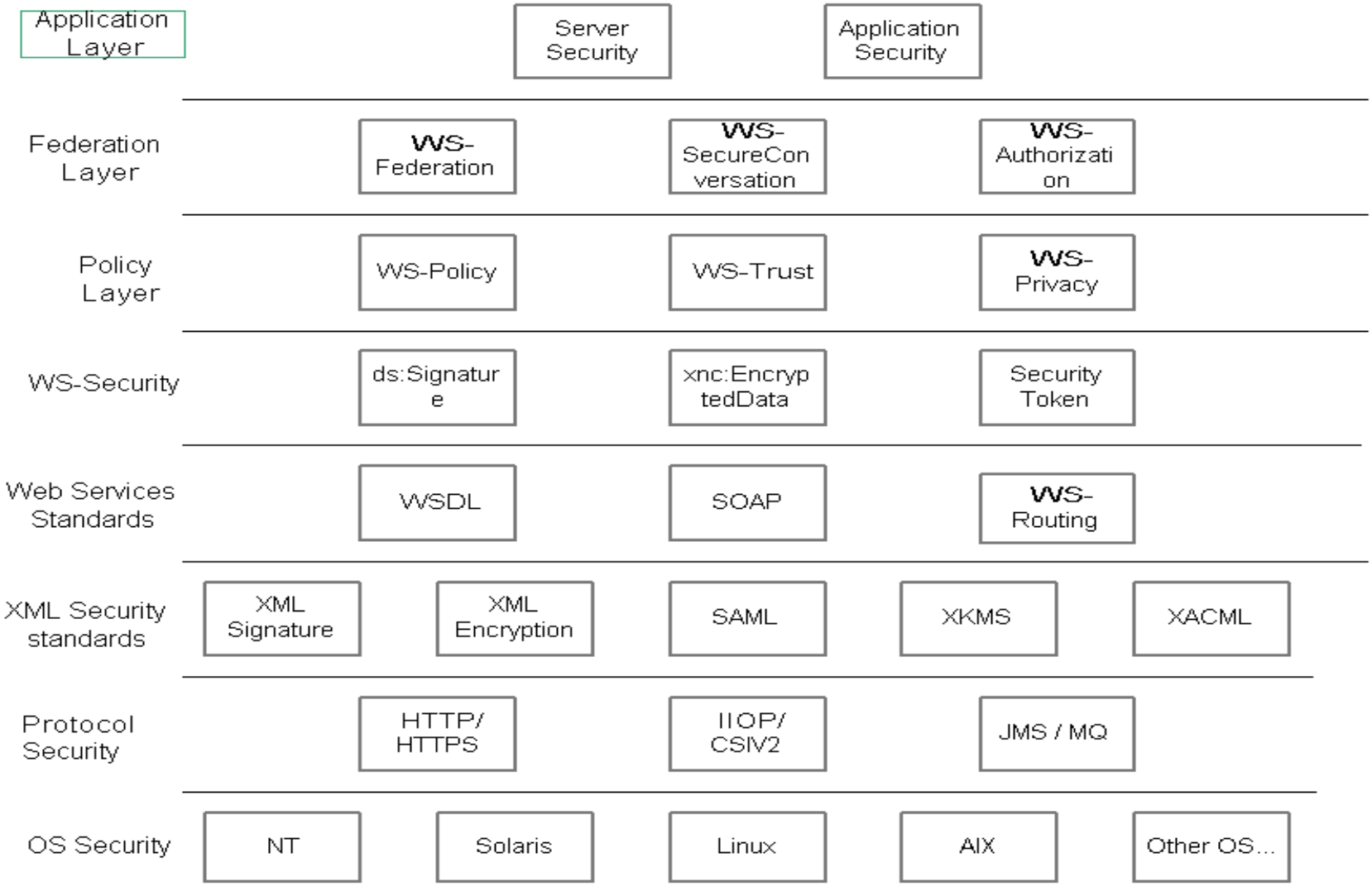
Service testing



Service deployment



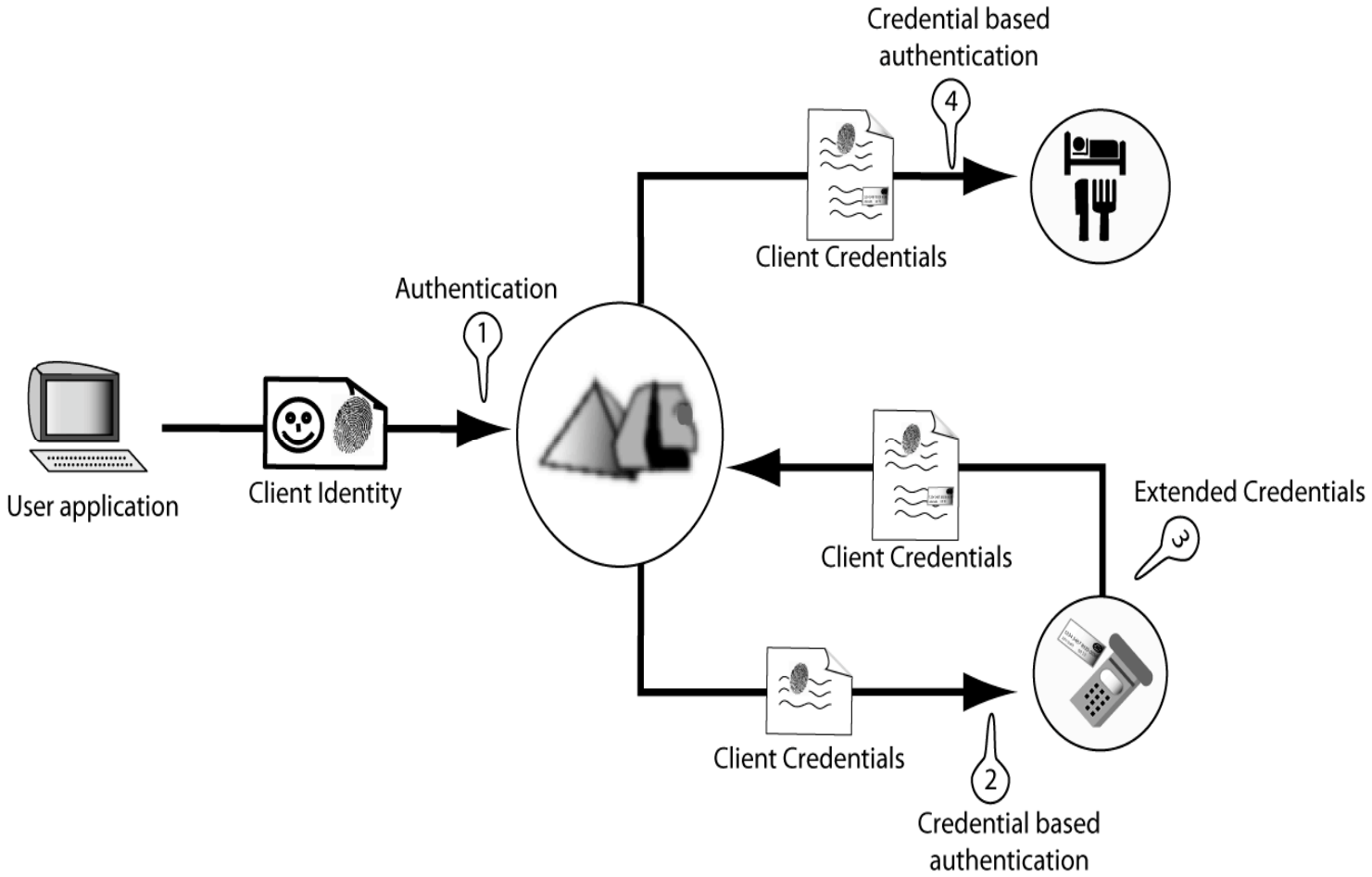
Security


Application Layer
Server Security
Application Security
Federation Layer
WS-Federation
WS-SecureConversation
WS-Authorization
Policy Layer
WS-Policy
WS-Trust
WS-Privacy
WS-Security
ds:Signature
xnc:EncryptedData
Security Token
Web Services Standards
WSDL
SOAP
WS-Routing
XML Security standards
XML Signature
XML Encryption
SAML
XKMS
XACML
Protocol Security
HTTP/ HTTPS
IIOP/ CSIV2
JMS / MQ
OS Security
NT
Solaris
Linux
AIX
Other OS...

Identity within SOA

- Must have some means by which a user (human or process) can establish its identity (obtain a credential) and then pass this to a target service in a format it understands
  - **Standards based formats are very important**
    - **WS-Security**
- It is common to have composite services forming a hierarchy
  - **The SOA must ensure that every intermediary can authenticate the requesting client (which could be a service) before passing credentials to the next service**
  - **As the credential information flows, it may be augmented or completely changed by each intermediate service: identity management must be federated hierarchically in order for it to scale and match the business domain**

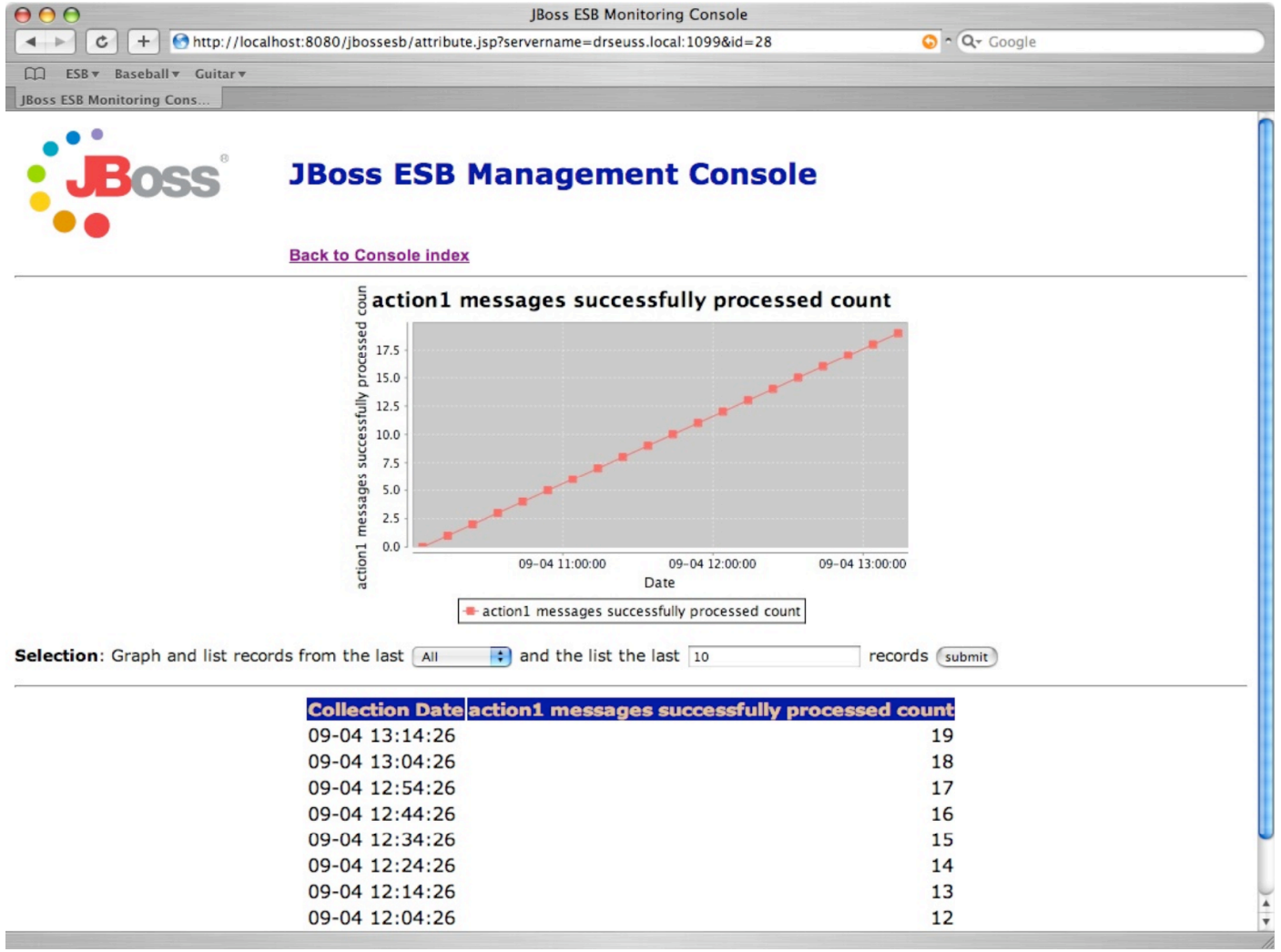
Identity management



Business Activity Monitoring

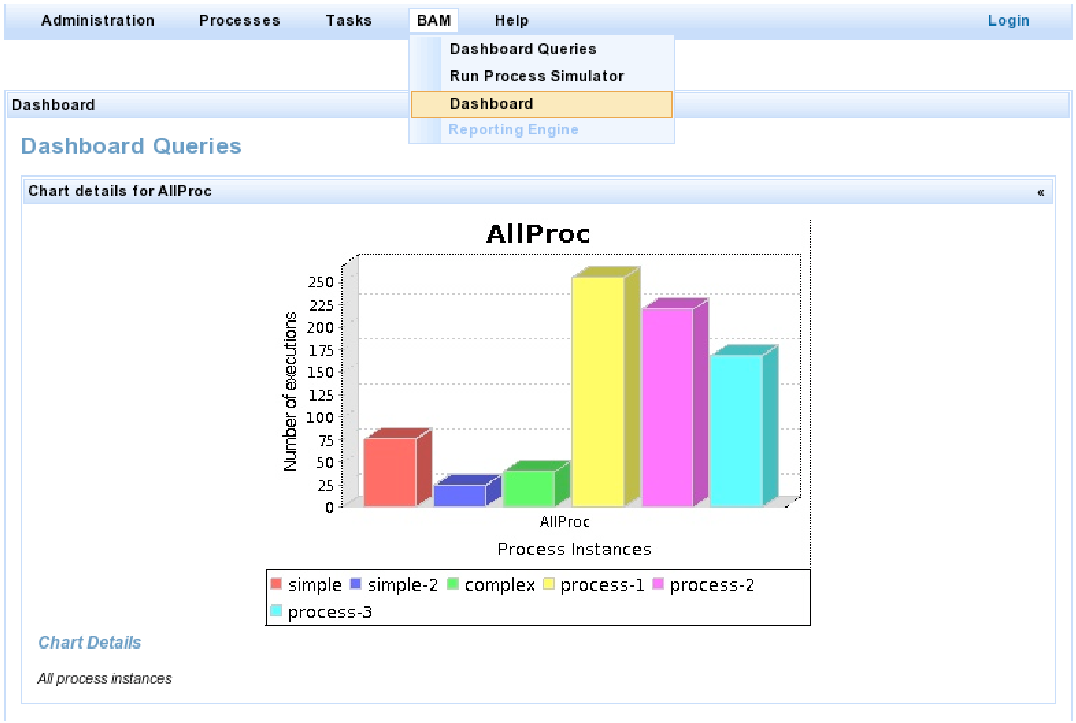
- Real-time access to critical business performance metrics
  - **Helps to improve the efficiency and effectiveness of business processes**
- Real-time process/service monitoring is a common capability supported in many distributed infrastructures
  - **BAM differs in that it draws information from multiple sources to enable a broader and richer view of business activities**
  - **BAM also encompasses business intelligence as well as network and systems management**
  - **BAM is often weighted toward the business side of the enterprise**
    - **As such, there has recently been a movement for good BAM implementations to be closely related to the governance infrastructures**

BAM



Collection Date	action1 messages successfully processed count
09-04 13:14:26	19
09-04 13:04:26	18
09-04 12:54:26	17
09-04 12:44:26	16
09-04 12:34:26	15
09-04 12:24:26	14
09-04 12:14:26	13
09-04 12:04:26	12

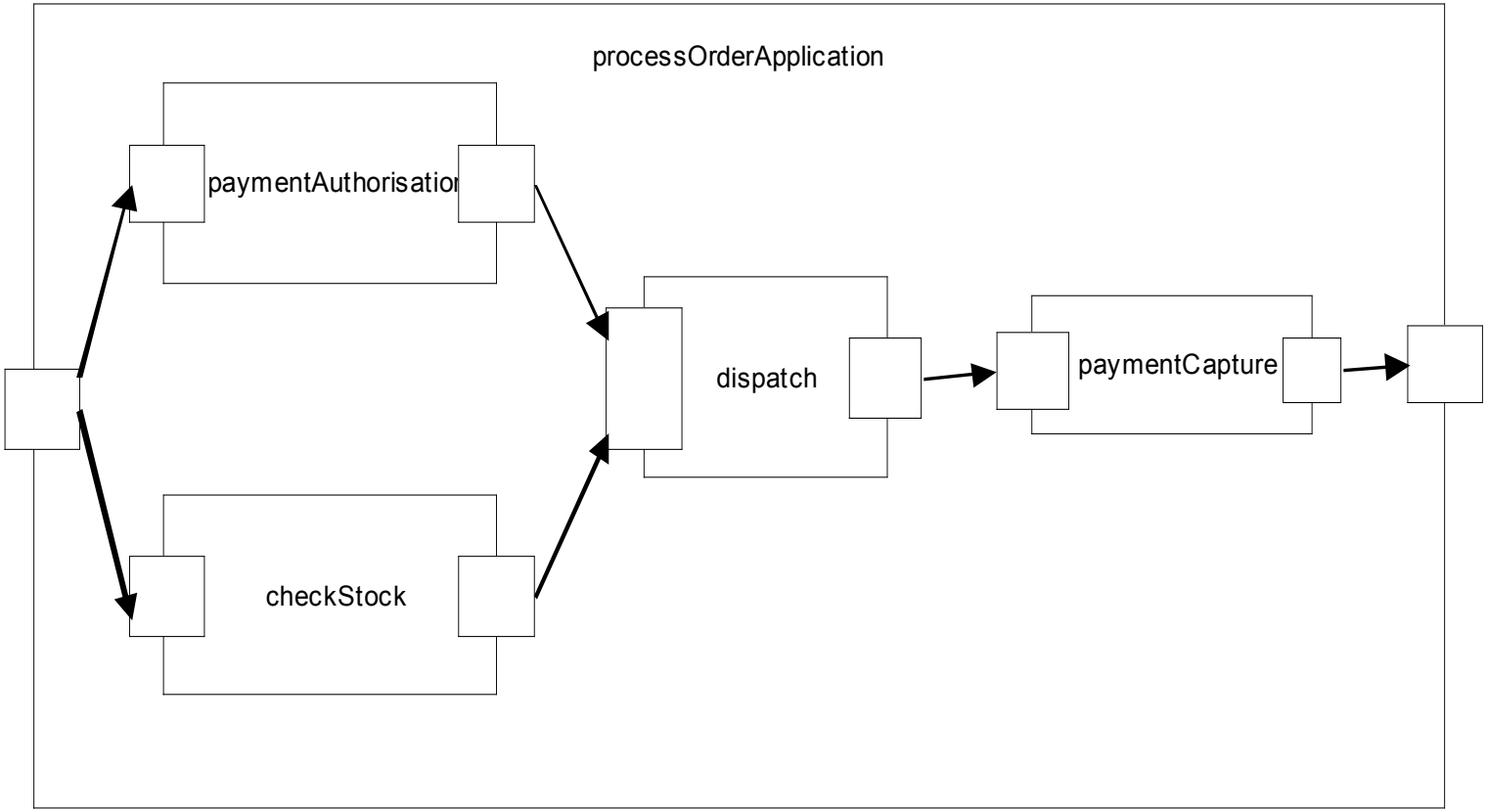
More BAM


Administration
Processes
Tasks
BAM
Help
Login
Dashboard Queries
Run Process Simulator
Dashboard
Reporting Engine
Dashboard
Dashboard Queries
Chart details for AllProc
AllProc
Number of executions
250
225
200
175
150
125
100
75
50
25
0
AllProc
Process Instances
simple
simple-2
complex
process-1
process-2
process-3
Chart Details
All process instances

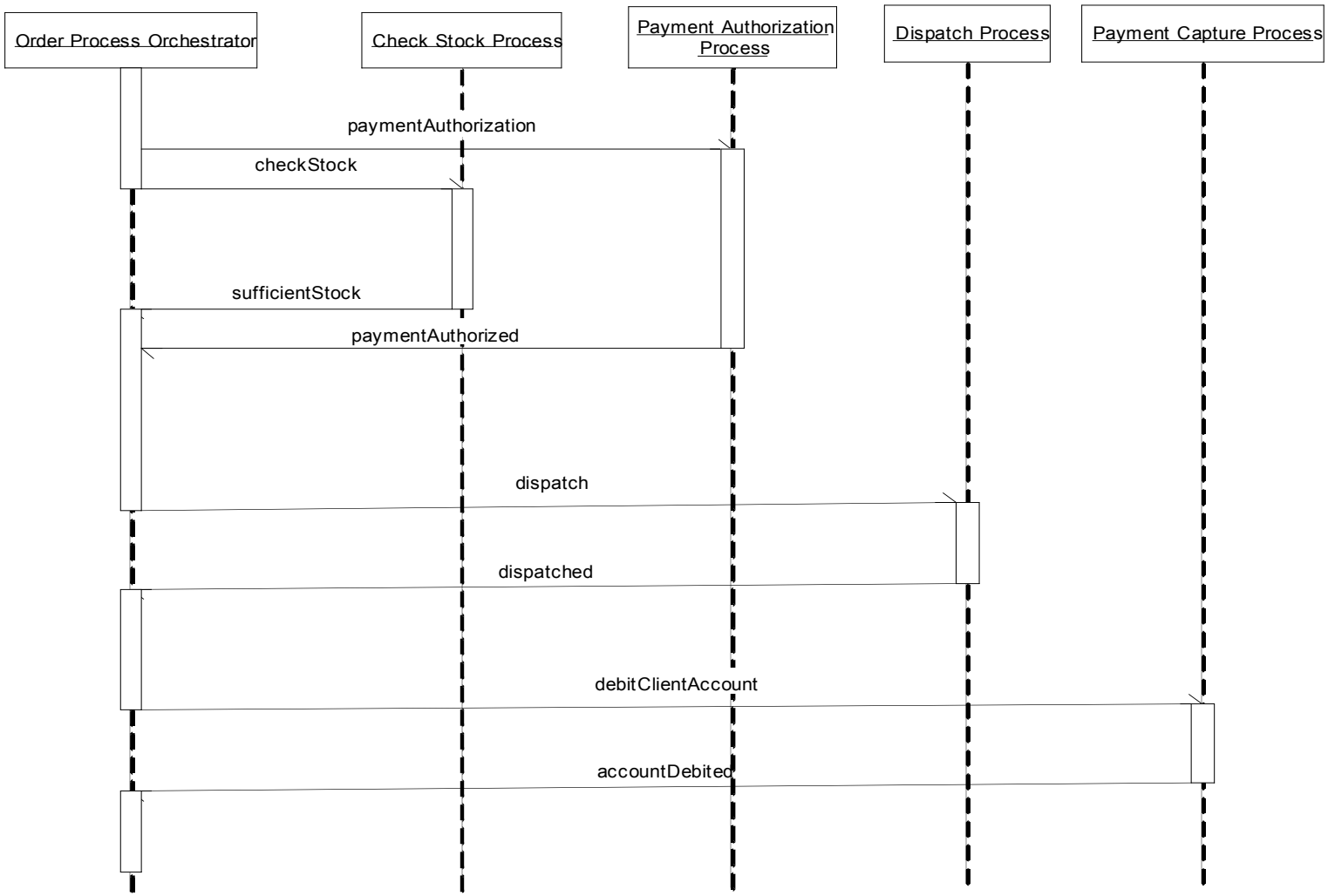
Service orchestration

- **Orchestration (e.g., BPM or workflow) is important in many distributed environments**
  - More so as the scale and complexity increases
- **Need to have intra service task orchestration**
  - Control the transition of the state of a service as it executes tasks
- **Need to have inter service orchestration**
  - Control the invocations of services as messages flow through the infrastructure

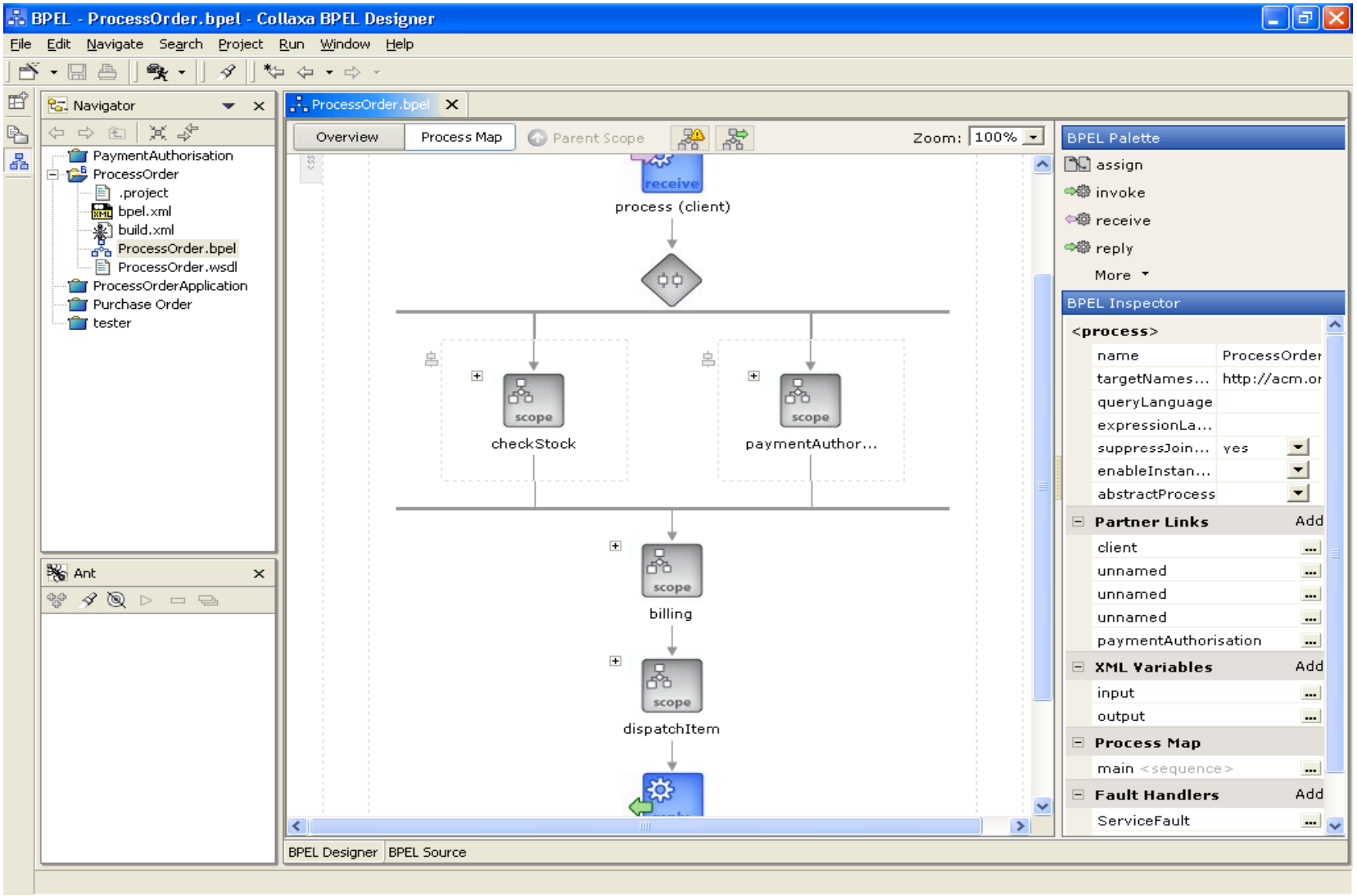
Process Order Example



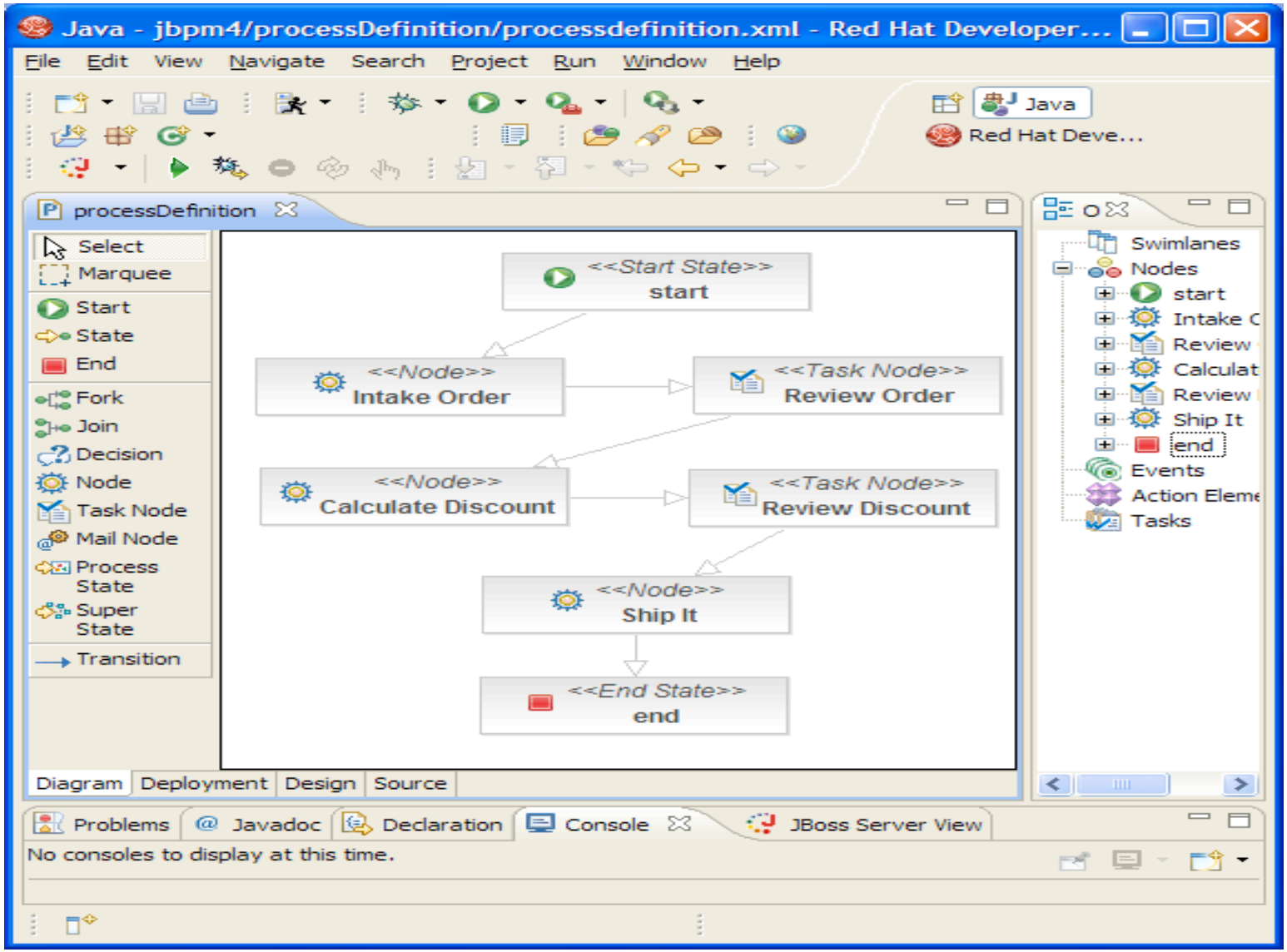
Orchestrating message flows



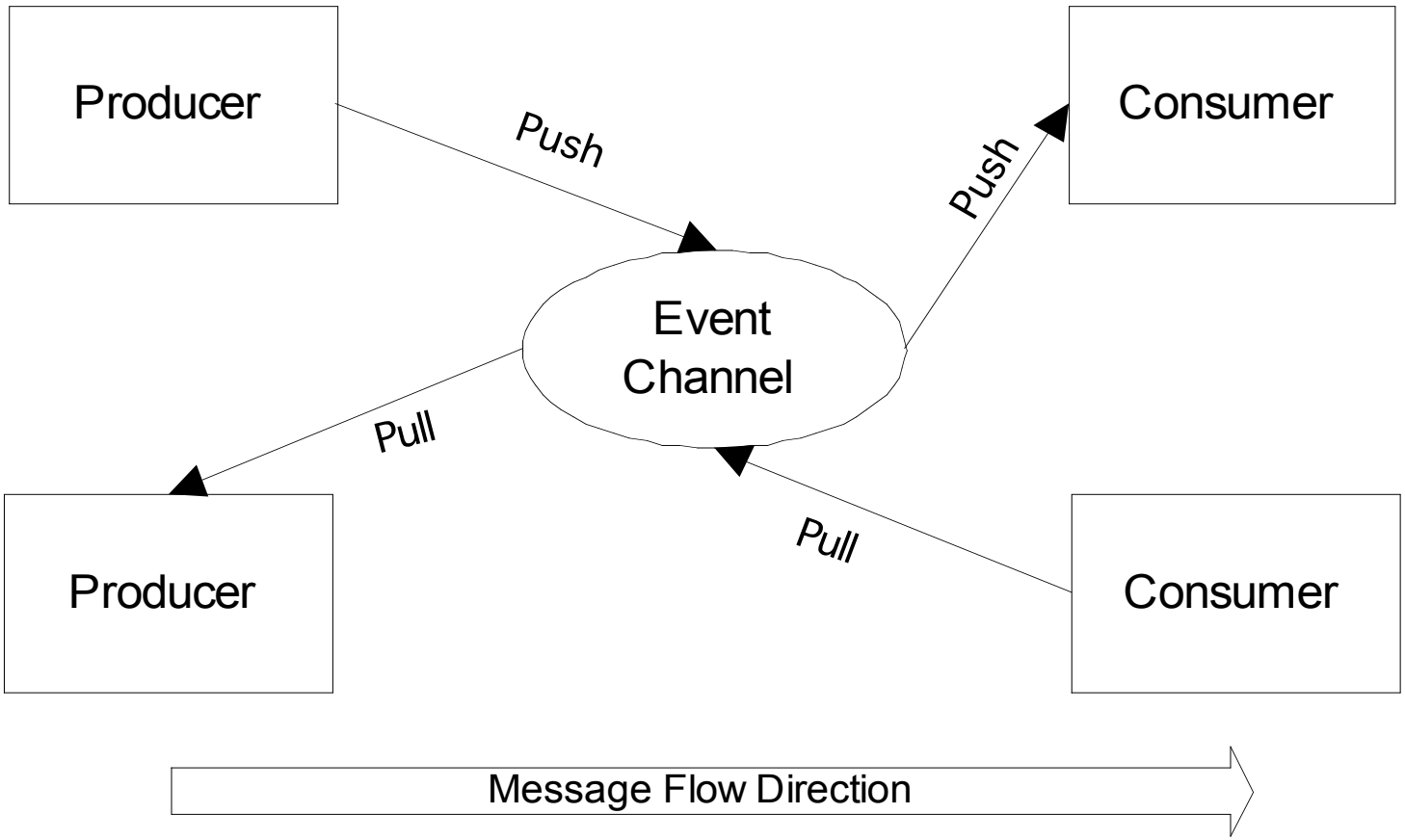
WS-BPEL example



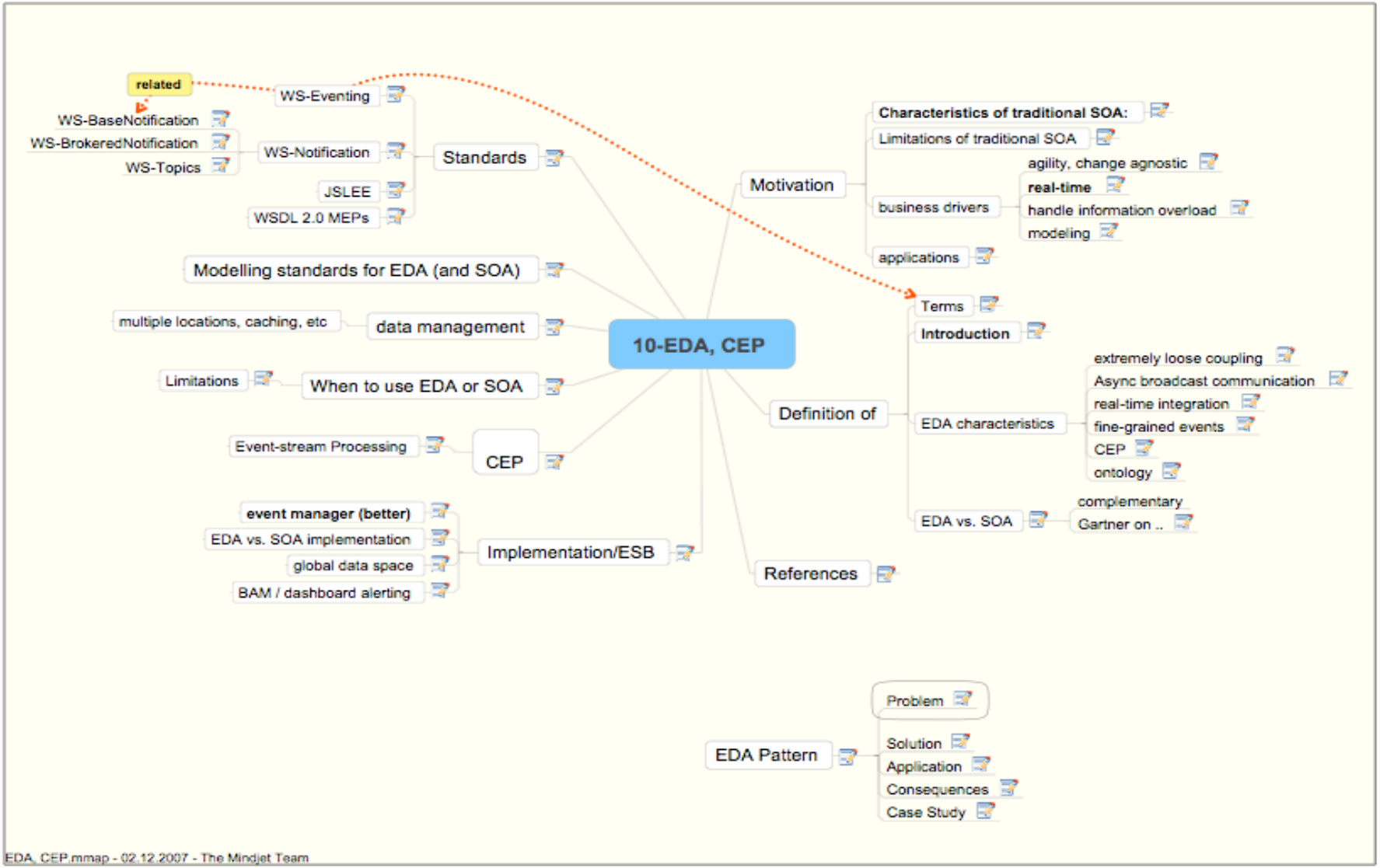
jBPM example



Event Driven Architecture



EDA in a Nutshell



EDA vs SOA?

- **Everything can be mapped to an event**
  - Mouse click, arrival of message, send of message
- **Reasoning about (distributed) systems in terms of events may be more natural**
- **EDA can be a methodology on SOA**
- **EDA can be an architecture on SOA**

The Enterprise Service Bus

- **Started life as JMS++**
- **Rapid adoption as SOI**
  - Messaging infrastructure
  - Process orchestration, typically via WS-BPEL
  - Protocol translation
  - Adapters
  - Change management (hot deployment, versioning, lifecycle management)
  - Fault tolerance
  - Security
  - Governance

SOA Infrastructure

The JBoss SOA Platform

- **A Service Oriented Infrastructure**
  - Based on JBossESB, Drools, JBossWS, JBossTS, JBoss Messaging and jBPM
  - Can run stand-alone or be deployed into JBossAS
- **JBossESB acts as the glue**
  - Supported protocols and capabilities make it more of an Internet Service Bus
  - Currently uses the “doWork” service definition approach
- **Encourages an incremental approach to SOA**
  - You don’t need to be a domain expert to benefit from it
  - Build up your knowledge in step with your requirements

Relationship to JBossESB

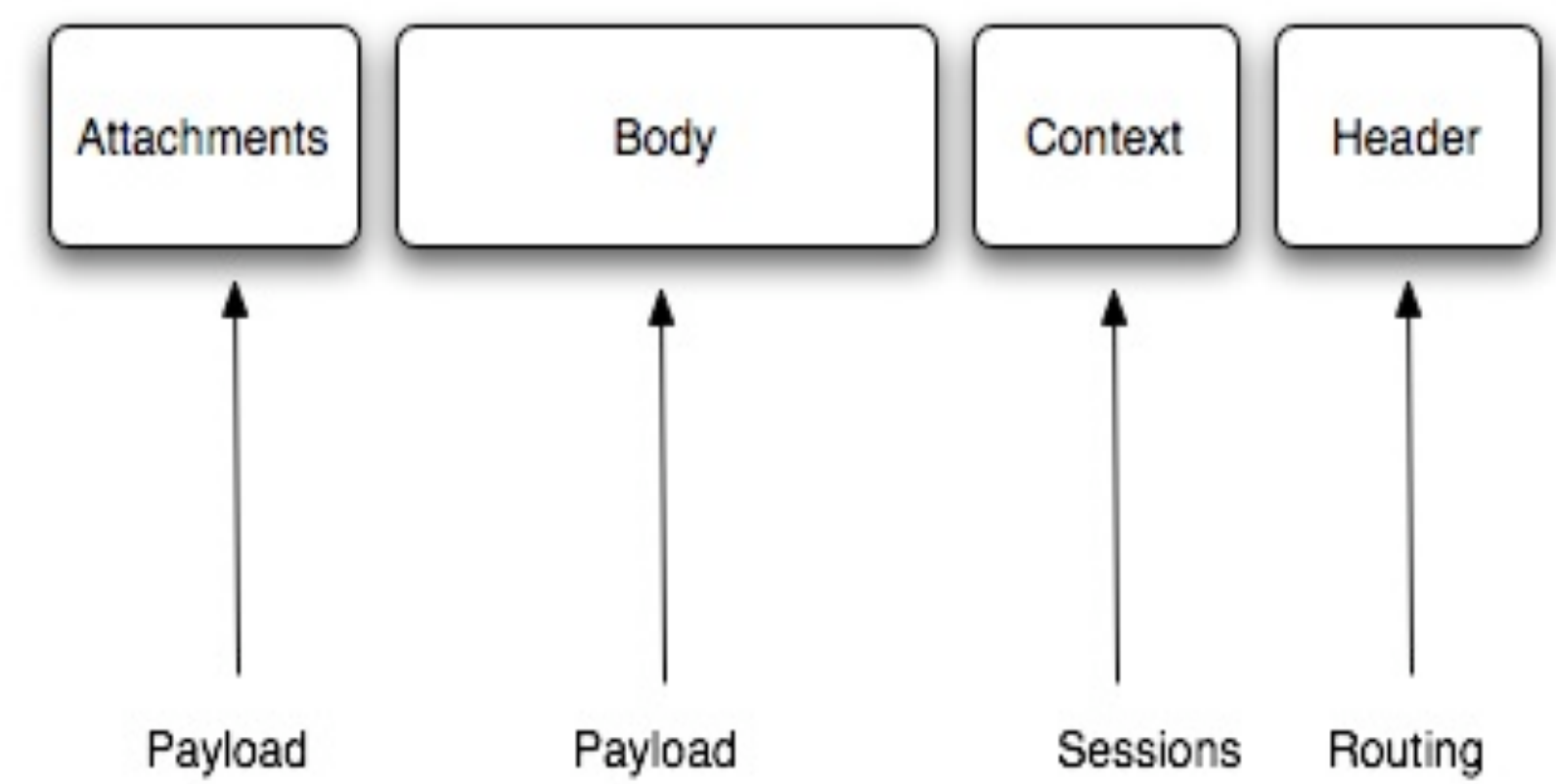
- **Messages and services are key to architecture**
- **Inherently asynchronous**
  - Correlated one-way messages for RPC
- **Support for Web Services**
- **Support for task management**
- **Adapters**
  - JCA
  - Gateways
- **Flexible architecture**
  - Multi-implementation approach

Where does it fit?

Services and messages

- **Within the SOA-P everything is a service**
- **All services are interacted with via messages**
  - Messages are part of the contract between client and service
- **Messages do not imply specific implementations of carrier-protocol**
- **Services do not need to be bound to specific implementations of carrier-protocol**
  - Email, S-FTP, JMS, File, etc.
  - More can be added as required

The Message envelope



Message implementations

- **On-the-wire representation may be tailored for environment**
  - E.g., binary versus text
    - Though binary breaks loose coupling
- **Only the structure of the Message is mandated**
- **Two wire-formats provided**
  - Java Serialized
    - Exposes implementation choice
  - XML
- **Others can be added statically or dynamically**

Message delivery in the SOA-P

- **Addressed via WS-Addressing Endpoint References**
  - Transport agnostic
- **Supports request-response as well as one-way MEP**
- **Mandatory to define the recipient address**
- **Optional**
  - Reply address
  - Message relationship information
  - Fault address

Gateway Services

- **Need to allow legacy services to plug-in to the bus**
- **Need to allow legacy clients to plug-in to the bus**
- **Neither have concept of Message or EPR**
- **Must bridge from ESB-aware to ESB-unaware domains**
  - Gateways perform this role
- **This allows the bus to be extended across the enterprise without perturbing existing infrastructure**

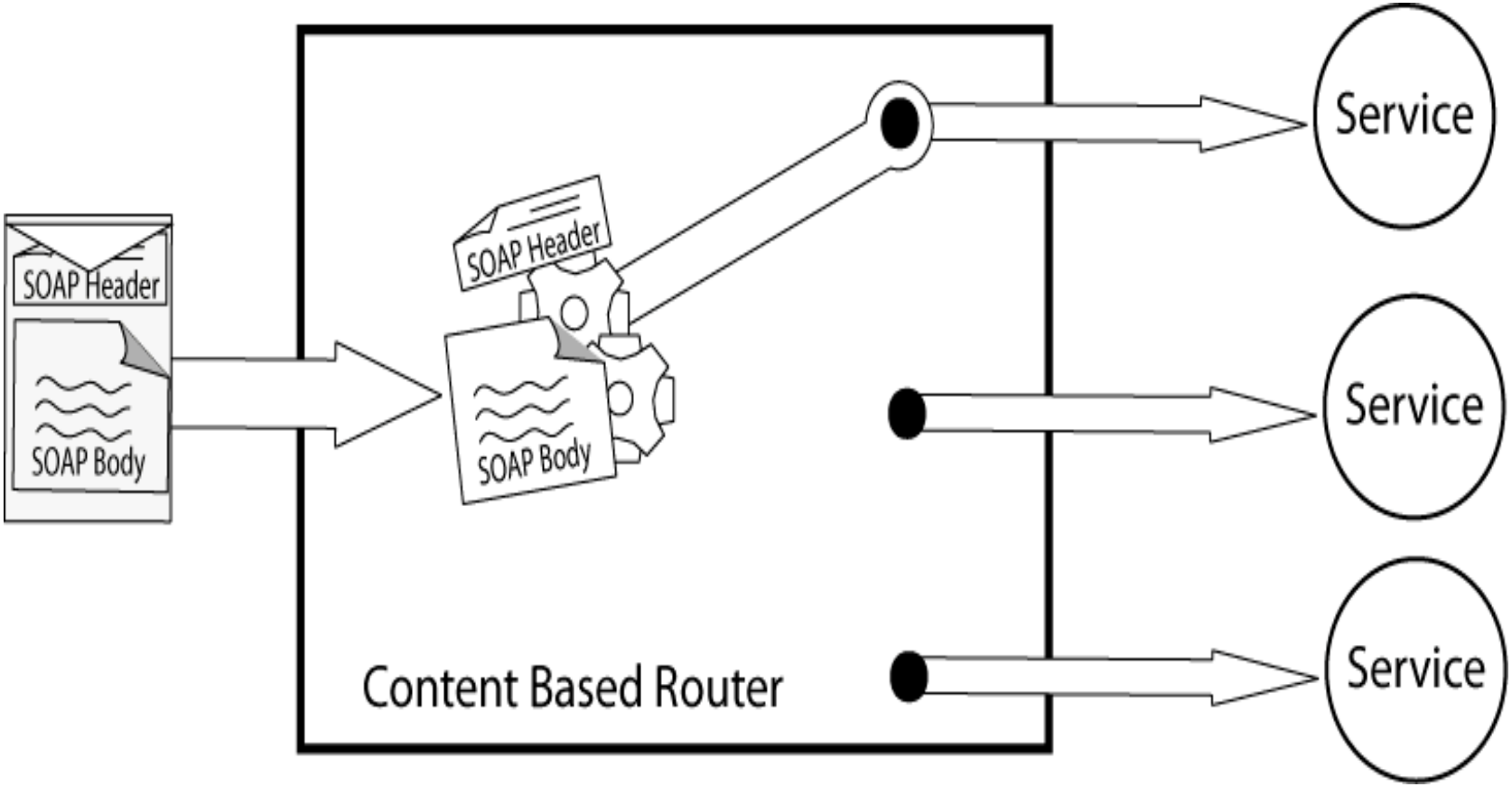
Service registration

- **Services are identified by Service Name but addressed by EPR**
  - Can be clustered for high availability and load balancing
- **Registry associates <Service Name, EPRs>**
- **Service may be available on more than one EPR**
  - E.g., different qualities of service
- **Services are expected to store EPR when activated**
- **Senders look up EPR(s) using Service Name**
  - May select on other criteria

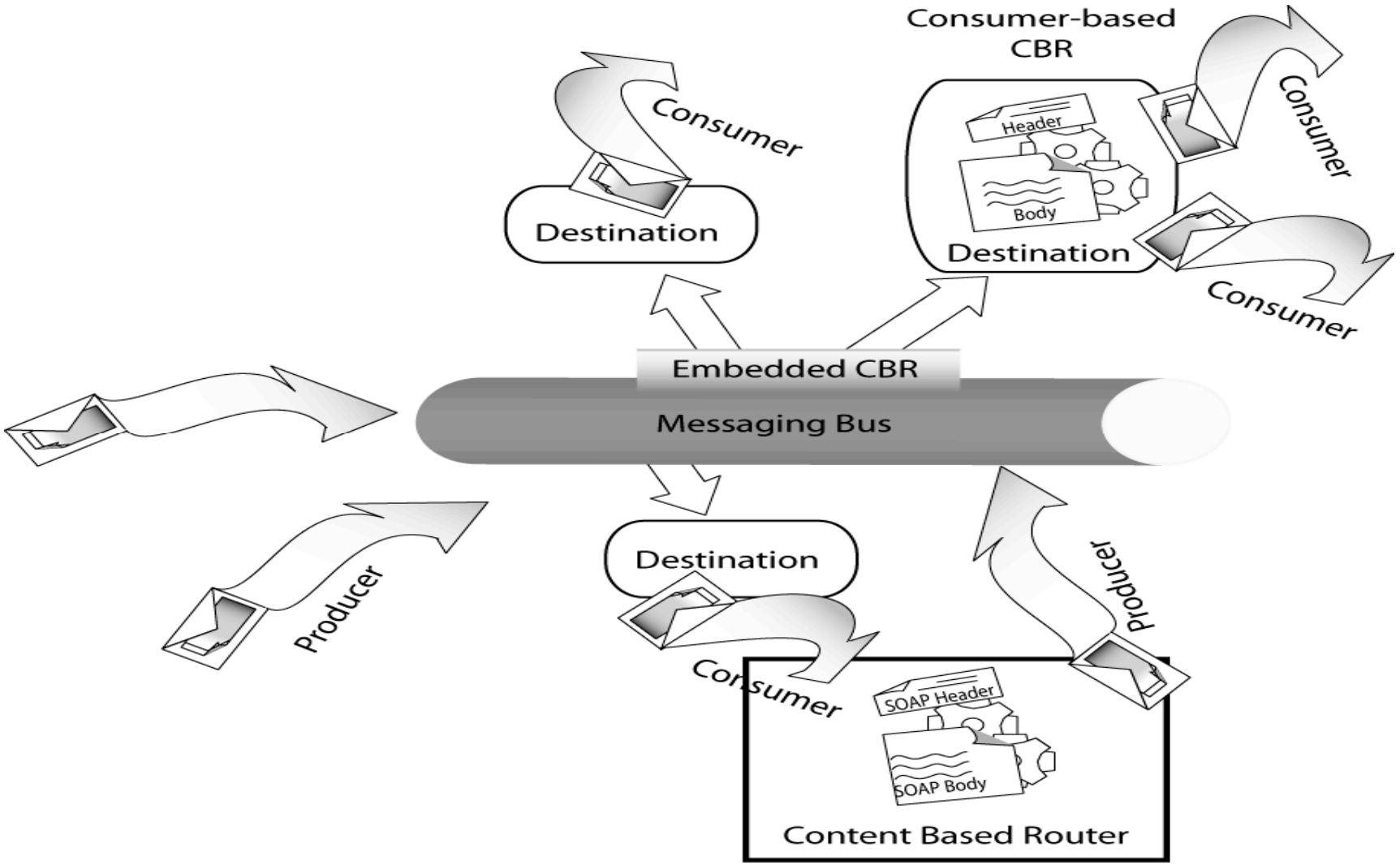
Content based routing

- **Intermediary services can redirect messages based on content**
  - Hiding federating service implementations
  - Business logic choices
  - Fault tolerance
- **Not a requirement for SOA**
  - But does help loose coupling and legacy integration
- **SOA-P has a CBR Service**
  - Supports JBoss Rules and XPath expressions

Web Service example



SOA Platform example



JBoss Rules

```
rule "Routing Rule - Serialized based message"

        when
                Message( type == MessageType.JAVA_SERIALIZED)
        then
                System.out.println("Serialized");
                destinationServices.add("test_category:Serialized_ServiceDestination");

end

rule "Routing Rule - XML based message"

        when
                Message( type == MessageType.JBOSS_XML)
        then
                System.out.println("JBoss_XML");
                destinationServices.add("test_category:JBOSS_XMLDestination");
end
```

Message transformation

- **Different services may communicate in different vocabularies**
  - Particularly with dynamic service registration/updates
- **Data may need to be restructured based on recipient, time of day, etc.**
- **Several ways to do transformation**
- **Transformation Service**
  - Smooks
  - XSLT
  - Others can be plugged in

Message store

- **Messages can be durable recorded**
- **Useful for audit trail, debugging, replay etc.**
  - Sometimes mandated by local laws
- **Separate service**
- **Flexible implementations possible**
  - Service API does not impose implementation restrictions
  - Out-of-the-box uses JDBC

Advanced Concepts

Sessions and SOA

- **A session concept common to distributed systems**
  - CORBA IOR
  - Transactions
  - HTTP Servers
  - MOM message groups

Http Sessions

- **At the heart of the Web**
- **HTTP provides Cookies (RFC 2965)**
  - Many applications use for conversational state
    - Opaque, owned by server
- **Browser is not coupled to web site**
  - Simple
  - Flexible

Web Services

- **Business functions as modeled as networked services**
  - Explicit failure modes
- **Coarse grained, orthogonal business functions**
  - Repurposable services
- **Focus on the exchange of self-describing (XML) business documents**
  - Higher level abstractions
- **Leverage the ubiquitous protocols of the Web**
  - Easy interconnectivity

Web Services and sessions

- **Two models to consider**
  - WS-Addressing
  - WS-Context (generally, context)

WS-Addressing

- **Addressing in SOAP header**
  - destination, reply to, fault to, message id, relates to
- **Endpoint References (EPR)**
  - Includes endpoint URL
  - Session information
- **Need to look at EPRs to understand session model**

WS-A session model

EndpointReference plus ReferenceParameters

```
<wsa:EndpointReference xmlns:wsa=".." xmlns:ex="..">
   <wsa:Address>http://www.ex.com/s1</wsa:Address>
   <wsa:ReferenceParameters>
      <ex:ObjectKey>0xAF87</ex:ObjectKey>
   </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

WS-A session model

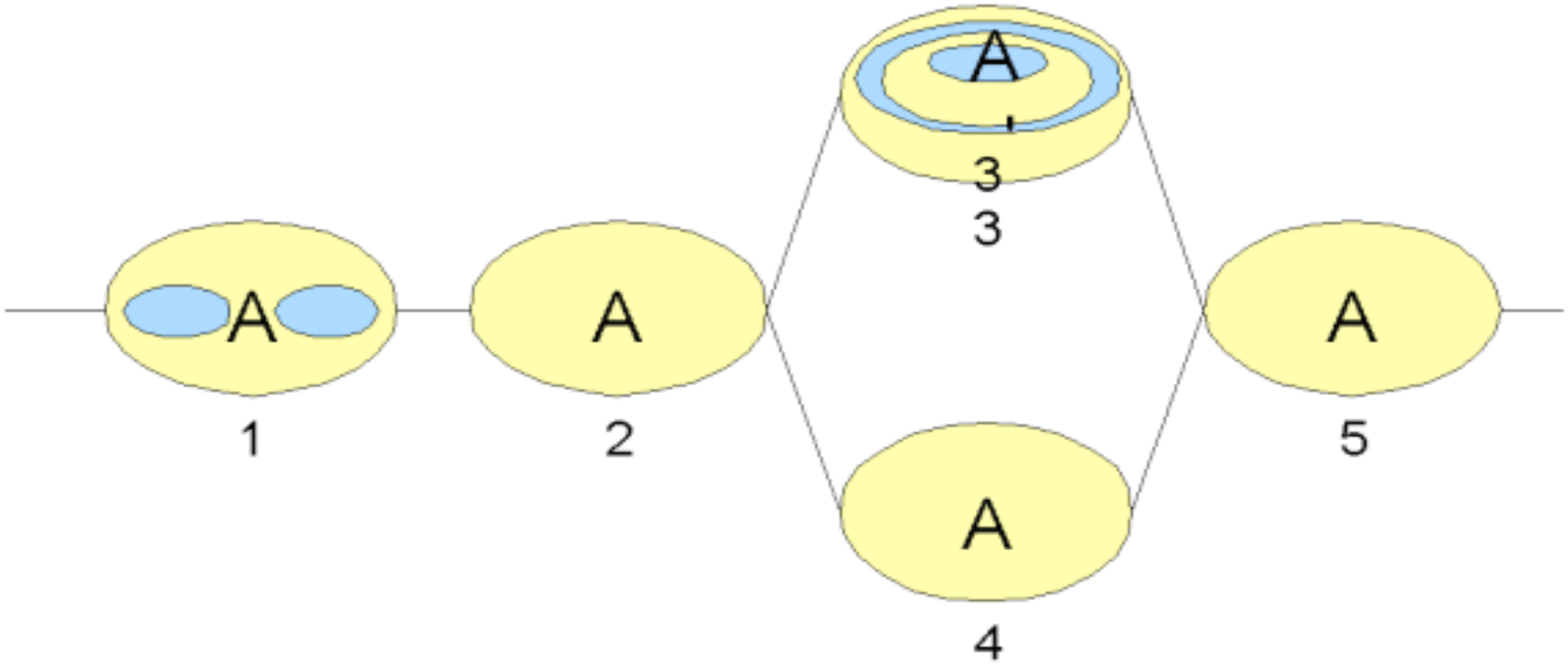
SOAP Binding

```
<S:Envelope xmlns:ex="... ">
   <S:Header>
      <wsa:To>http://www.ex.com/s1</wsa:To>
      <ex:ObjectKey>0xAF87</ex:ObjectKey>
   </S:Header>
....
</S:Envelope>
```

WS-Context

- **Scoping for arbitrary units of work**
  - Created, made to run and then terminated
  - Boundary of a context
- **Can do anything in an activity**
  - use the default context however services want
- **Default context is essentially a UID (URI)**
  - Just for correlation

Activity example



Context specifics

- **Context is a first-class element**
  - URI which may represent a web resource
- **Basic context contains**
  - Unique activity id for each activity
  - Timeout period (lifetime of activity)
- **May be augmented:**
  - Dynamically as remote invocations progress
  - Before application invocation occurs

Propagation

- **Context information propagated in SOAP header**
  - Minimally defined by a URI
    - In which case this is a service reference
- **Context definer specifies whether it must be understood by receiver**
  - MustPropagate

Context session model

```
<soap:Envelope>
  <soap:Header = mustUnderstand="1">
    <context
   xmlns="http://www.webservicestransactions.org/schemas/wsctx/2003/03"
       timeout="100"/”>
      <context-identifier>
   http://www.webservicestransactions.org/wsctx/abcdef:012345
       </context-identifier>
         <type>    http://www.webservicestransactions.org/wsctx/context/type1
       </type>
     </context> . . . .
```

Examples of context

- **WS-ReliableMessaging**
- **WS-AtomicTransaction**
- **WS-BusinessActivity**
- **WS-Enumeration**
- **WS-BPEL**
- **WS-SecureConversations**
- **SOAPConversations**
- **Apache Axis state management**

- **Toward a single model?**
  - Continuous reinvention bad
  - Important to get security right

Implications for SOA

- **Loosely coupled systems.**
  - WS-Addressing tightly couples the session to the reference.
    - Scalability, Fault tolerance
  - WS-Context allows a service client to bind the relationship to the service dynamically and temporarily
- **Do not expose implementation details**
  - WS-Addressing encourages object modeling
- **WSDL itself is agnostic about session models**
  - Nothing about the semantic of the session model contained in WS-Addressing can be expressed in WSDL.

Fault tolerance

- **Machines and software fail**
  - Fundamental universal law (entropy increases)
  - Things get better with each generation, but still statistically significant
- **Failures of centralized systems difficult to handle**
- **Failures of distributed systems are much more difficult**

Fault tolerance techniques

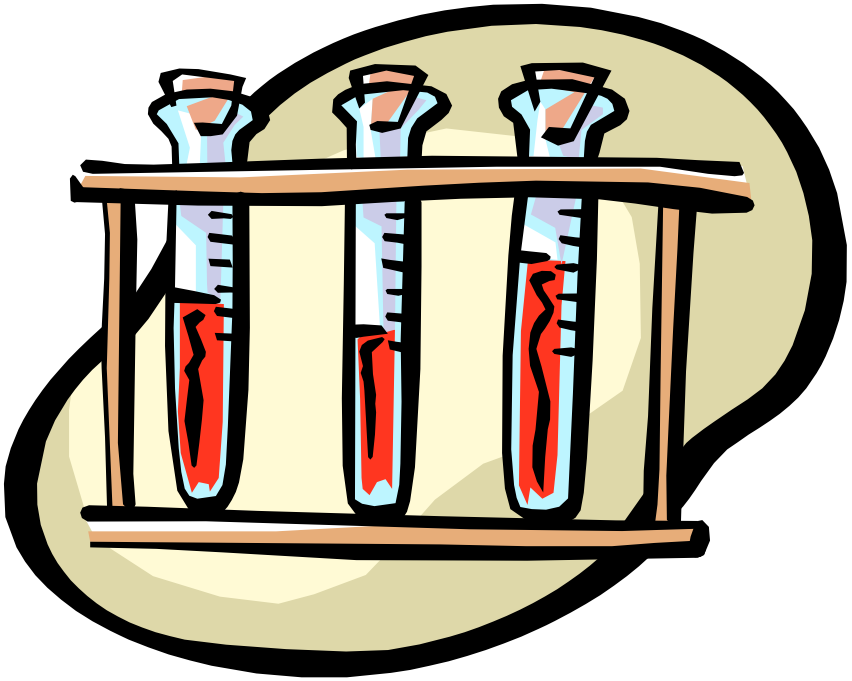
- **Replication of resources**
  - Increase availability
    - Probability is that a critical number of resources remain operational
    - “Guarantee” forward progress
  - Tolerate programmer errors by heterogeneous implementations
- **Spheres of control**
  - “Guarantee” no partial completion of work in the presence of failures

What is a transaction?

- **Mechanistic aid to achieving correctness**
- **Provides an "all-or-nothing" property to work that is conducted within its scope**
  - Even in the presence of failures
- **Ensures that shared resources are protected from multiple users**
- **"Guarantees" the notion of shared global consensus**
  - Different parties in different locales have the same view of the transaction outcome

ACID Properties

- Atomicity
- Consistency
- Isolation
- Durability



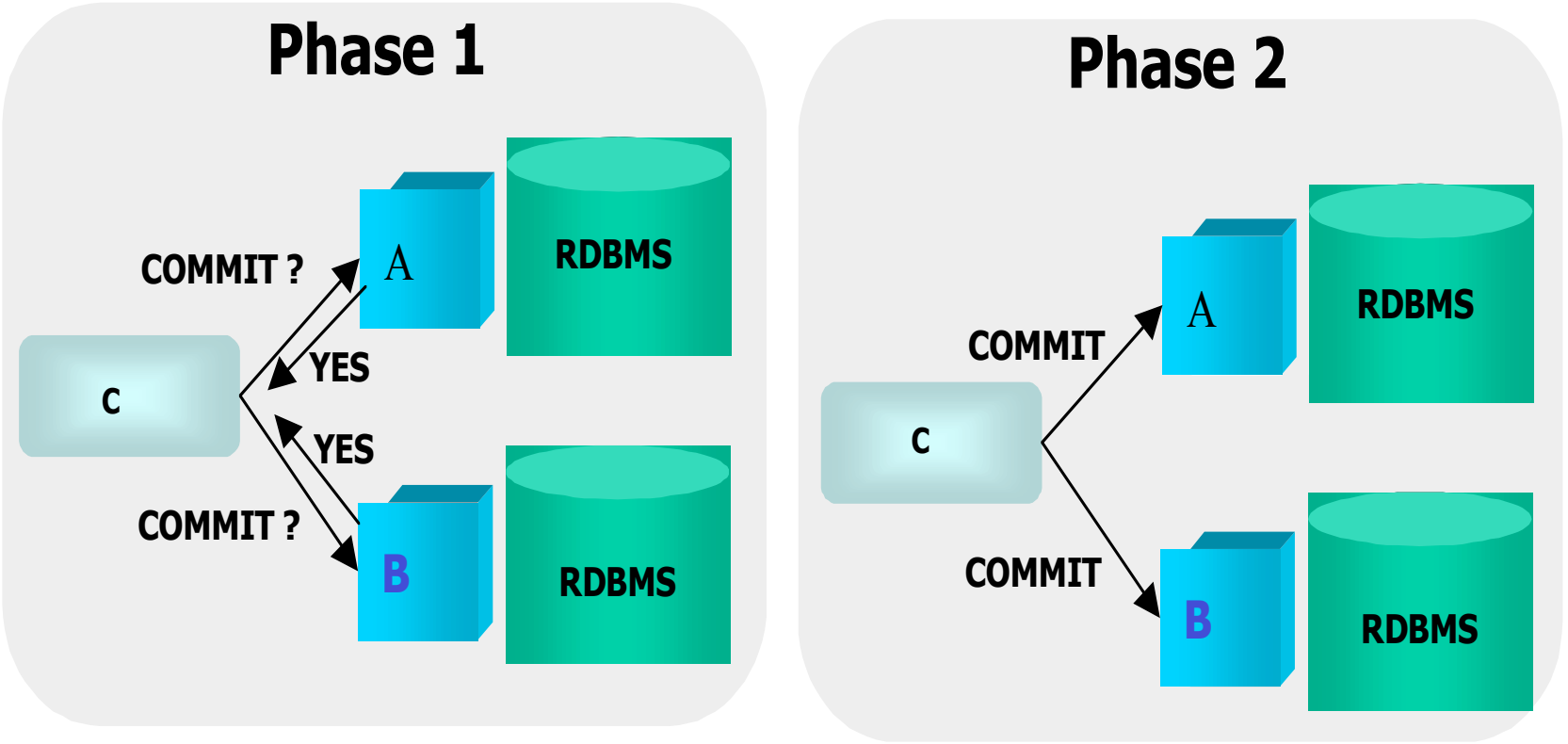
Atomicity

- **Within the scope of a transaction**
  - all changes occur together OR no changes occur
- **Atomicity is the responsibility of the Transaction Manager**
- **For example - a money transfer**
  - debit removes funds
  - credit add funds
  - no funds are lost!

Two-phase commit

- **Required when there are more than one resource managers (RM) in a transaction**
- **Managed by the transaction manager (TM)**
- **Uses a familiar, standard technique:**
  - marriage ceremony - Do you? I do. I now pronounce ..
- **Two - phase process**
  - voting phase - can you do it?
    - Attempt to reach a common decision
  - action phase - if all vote yes, then do it.
    - Implement the decision

Two-phase commit

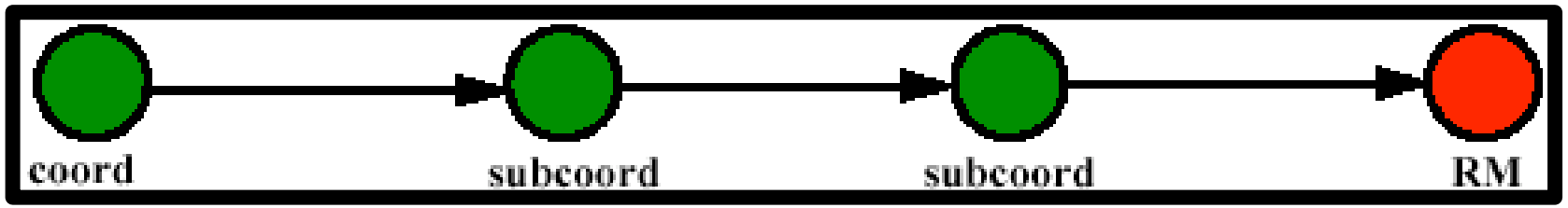


Handling failures

- **Presumed Abort Strategy**
  - can be stated as « when in doubt abort »
  - any failure prior the commit phase lead to abort the transaction
- **A coordinator or a participant can fail in two ways**
  - it stops running (crashes)
  - it times out waiting for a message it was expecting
- **A recovered coordinator or participant uses information on stable storage to guide its recovery**

2PC: optimizations

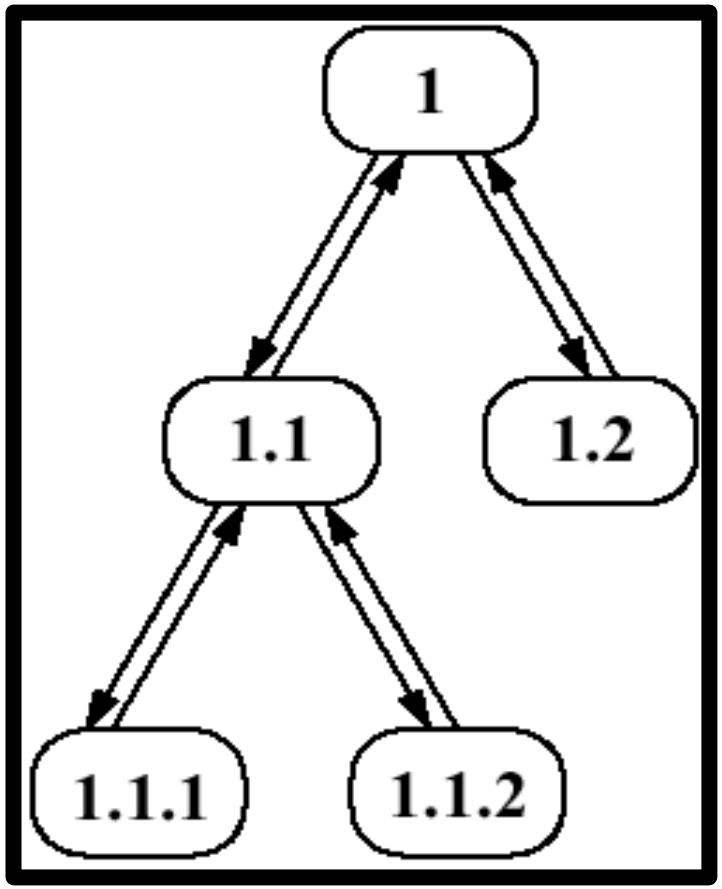
- **one phase commit**
  - no voting if transaction tree is single branch



One Phase Commit

- "read-only"
  - ✓ resource doesn't change any data
  - ✓ can be ignored in second phase of commit

Nested transactions



- **a transaction is *nested* when it executes within another transaction**
- **nested transactions live in a tree structure**
  - parents
  - children
- **implement modularity and containment**

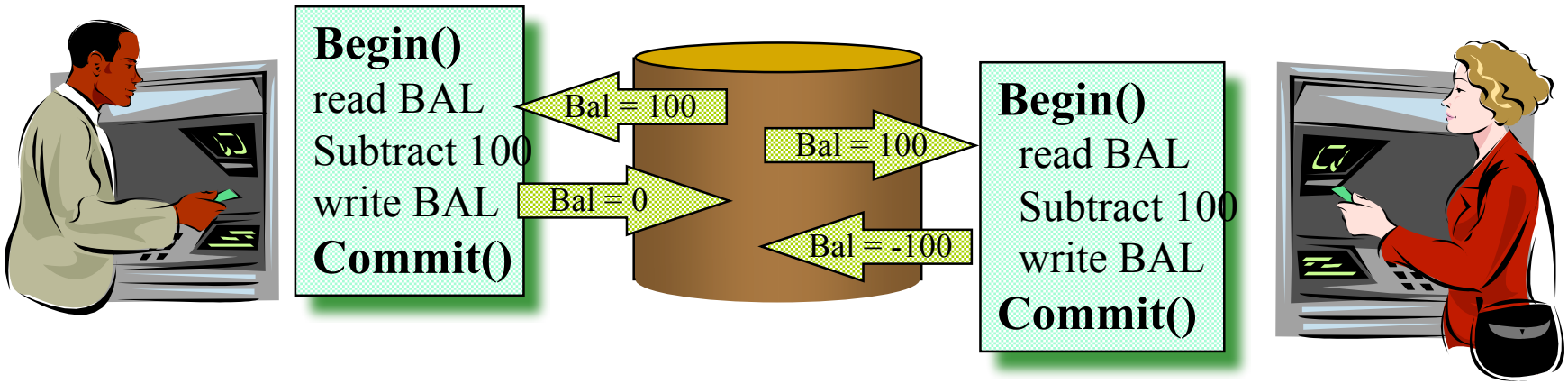
Consistency

- Transactions scope a set of operations
- Consistency can be violated within a transaction
  - Allowing a debit for an empty account
  - Debit without a credit during a Money Transfer
  - Delete old file before creating new file in a copy
- transaction must be correct according to application rules
- Begin and commit are points of consistency

- Consistency preservation is a property of a transaction, not of the TP system (unlike the A, I, and D of ACID)

Isolation

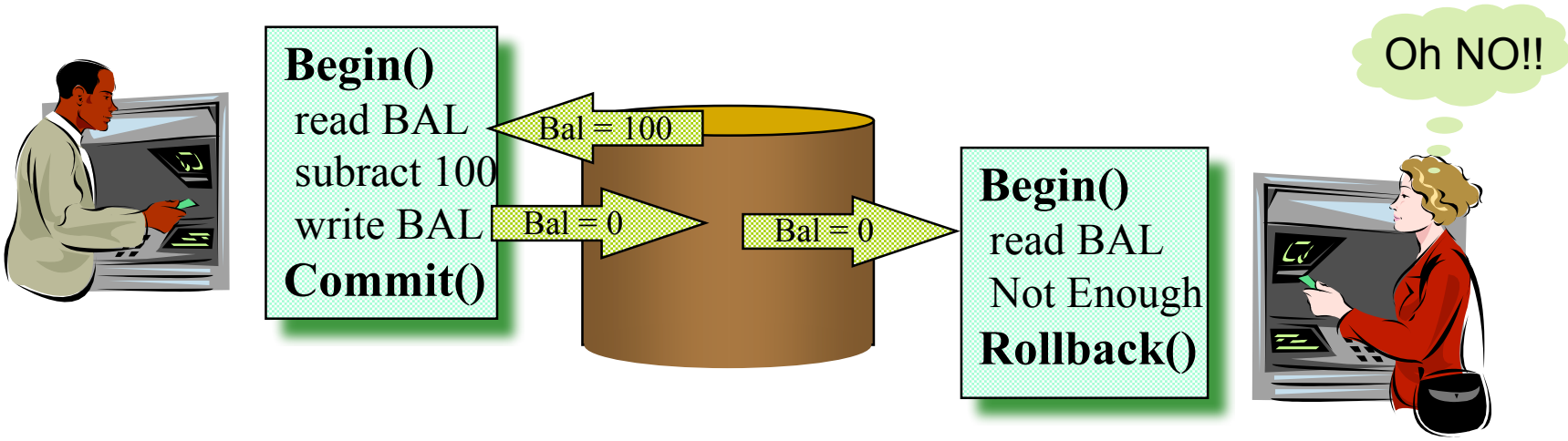
- **Running programs concurrently on same data can create concurrency anomalies**
  - the shared checking account example



Isolation

- **Transaction must operate as a black box to other transactions**
- **Multiple programs sharing data requires concurrency control**
- **When using transactions**
  - programs can be executed concurrently
  - BUT programs appear to execute serially

Isolation


Begin()
read BAL
subract 100
write BAL
Commit()
Bal = 100
Bal = 0
Bal = 0
Begin()
read BAL
Not Enough
Rollback()
Oh NO!!

Durability

- **When a transaction commits, its results must survive failures**
  - must be durably recorded prior to commit
  - system waits for disk ack before acking to user
- **If a transaction rolls back, changes must be undone**
  - before images recorded
  - undo processing after failure

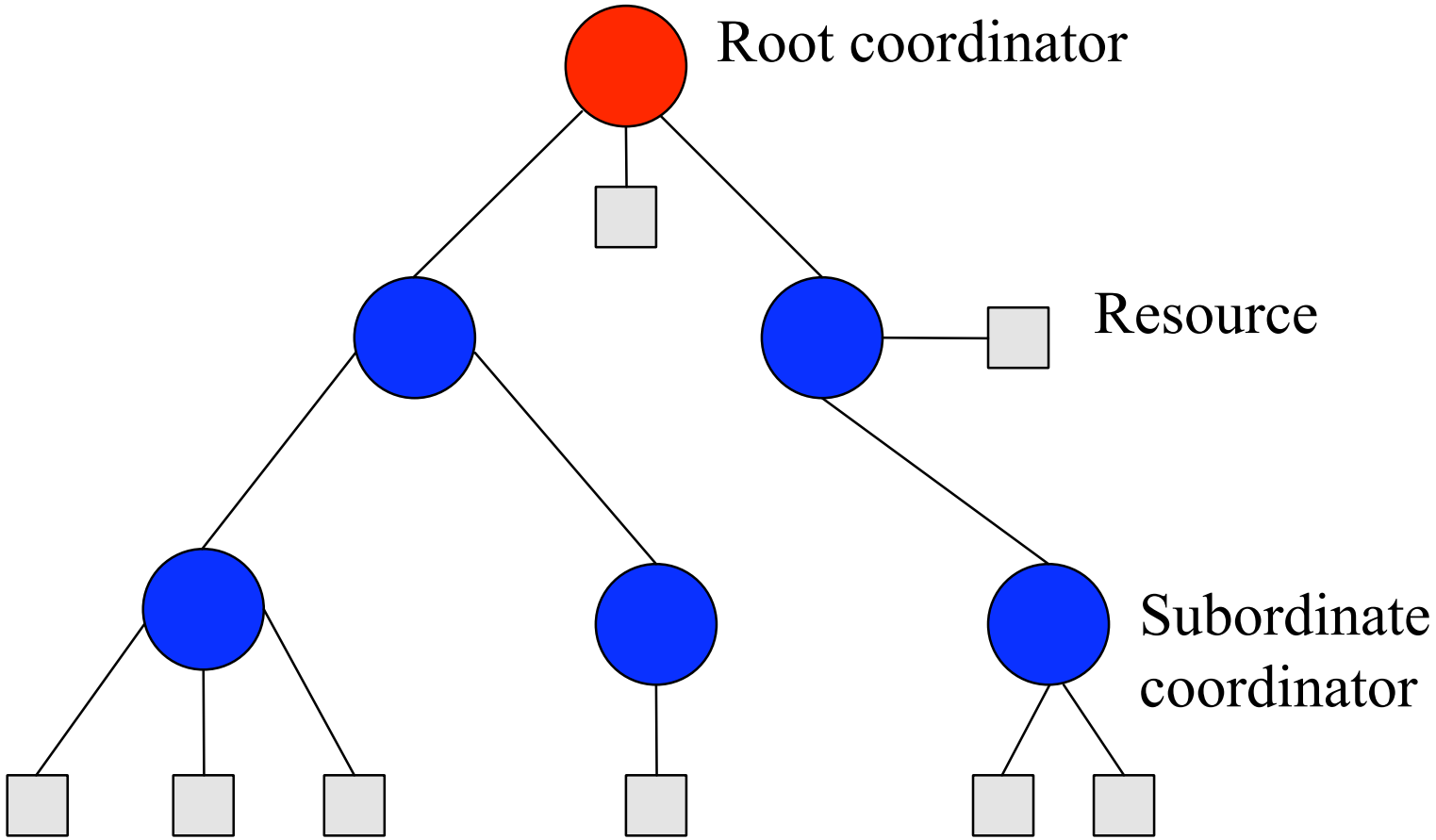
Heuristics

- **Two-phase commit protocol is blocking in order to guarantee atomicity.**
  - Participants may be blocked for an indefinite period due to failures
- **To break the blocking nature, prepared participants may make autonomous decisions to commit or rollback**
  - Participant *must* durably record this decision in case it is eventually contacted to complete the original transaction
  - If the decision differs then the coordinator's choice then a possibly non-atomic outcome has happened: a *heuristic outcome*, with a corresponding *heuristic decision*.

Interposition

- **Allows a subordinate coordinator to be created**
- **Interposed coordinator registers with transaction originator**
  - Form tree with parent coordinator
  - Application resources register locally

Interposition



SOA characteristics

- **Business-to-business interactions may be complex**
  - involving many parties
  - spanning many different organisations
  - potentially lasting for hours or days
- **Cannot afford to lock resources on behalf of an individual indefinitely**
- **May need to undo only a subset of work**
- **Need to relax ACID properties**

However …

- Web Services are as much about interoperability as they are about the Web
- In the short term Web Services transactions will be about interoperability between existing TP systems rather than running transactions over the Web

OASIS WS-TX Goals

- **4th attempt at standardising**
- **Support range of use cases**
- **“One-size does not fit all”**
  - “*Make each program do one thing well; to do a new job, build afresh rather than complicate old programs by adding new features”*, Doug McIlroy, inventory Unix pipes
  - Therefore a single protocol cannot cope with all requirements
- **Interoperability with existing transaction infrastructures**

WS-AT/WS-BA

- **Specifications released by Arjuna, BEA, IBM, IONA and Microsoft**
  - Now OASIS standard
- **Separate coordination from transactions**
  - WS-Coordination
- **Define two transaction models**
  - AtomicTransaction
    - Closely coupled, interoperability
  - Business Activities
    - Compensation based, for long duration activities

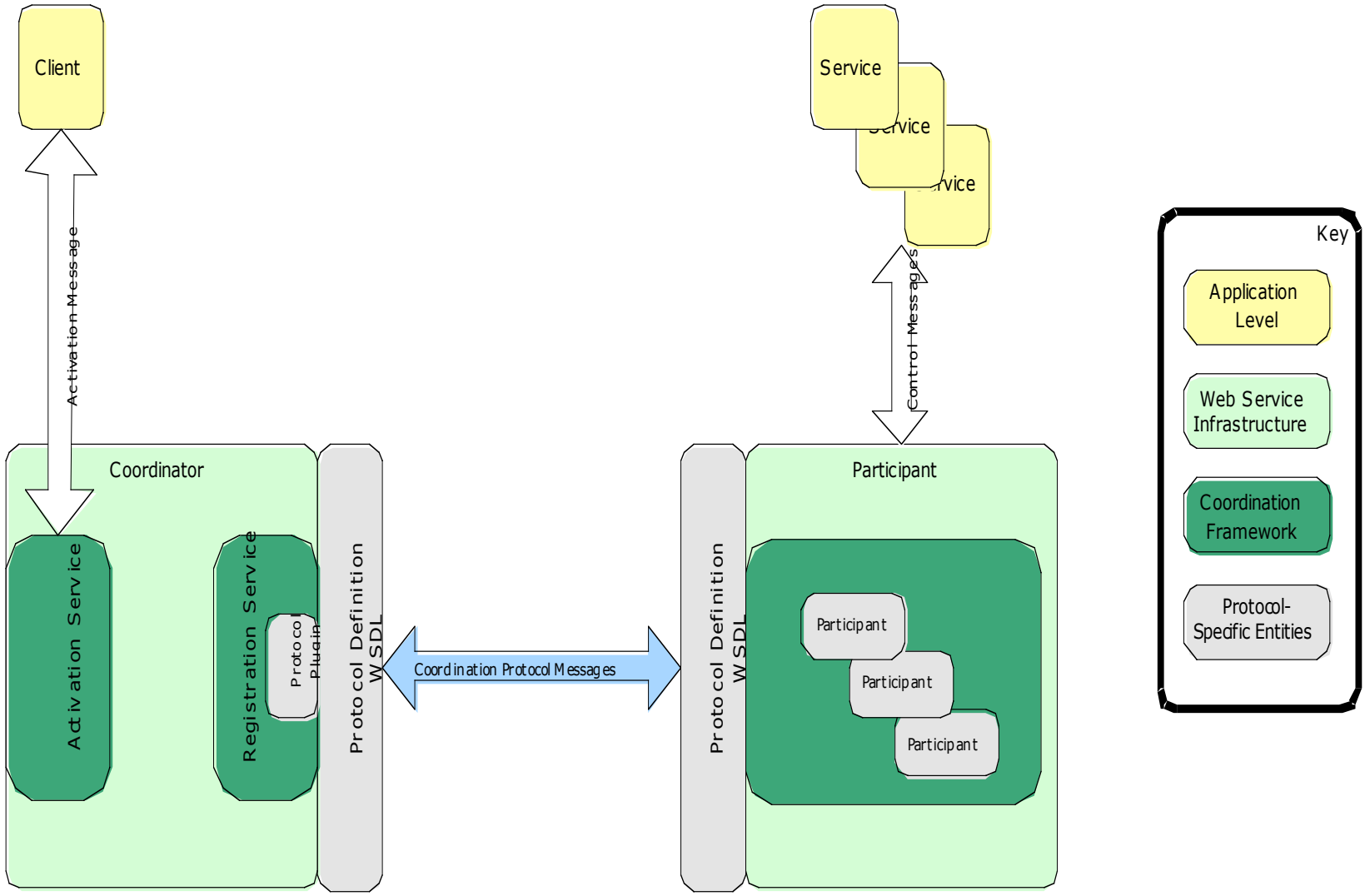
Web Services Coordination

- **Coordination is more fundamental than transactions**
  - Transactions, security, workflow, replication, …
  - But each use may require different protocol
    - Two-phase, three-phase, QoS specific, …
- **Define separate coordination service**
  - Allow customisation for different protocols

WS-Coordination

- **Defines typical roles of coordinator and participant**
  - Coordinator split into two roles
    - Activation service
      - Context
    - Registration service
  - Participant interface is implied by specific protocol

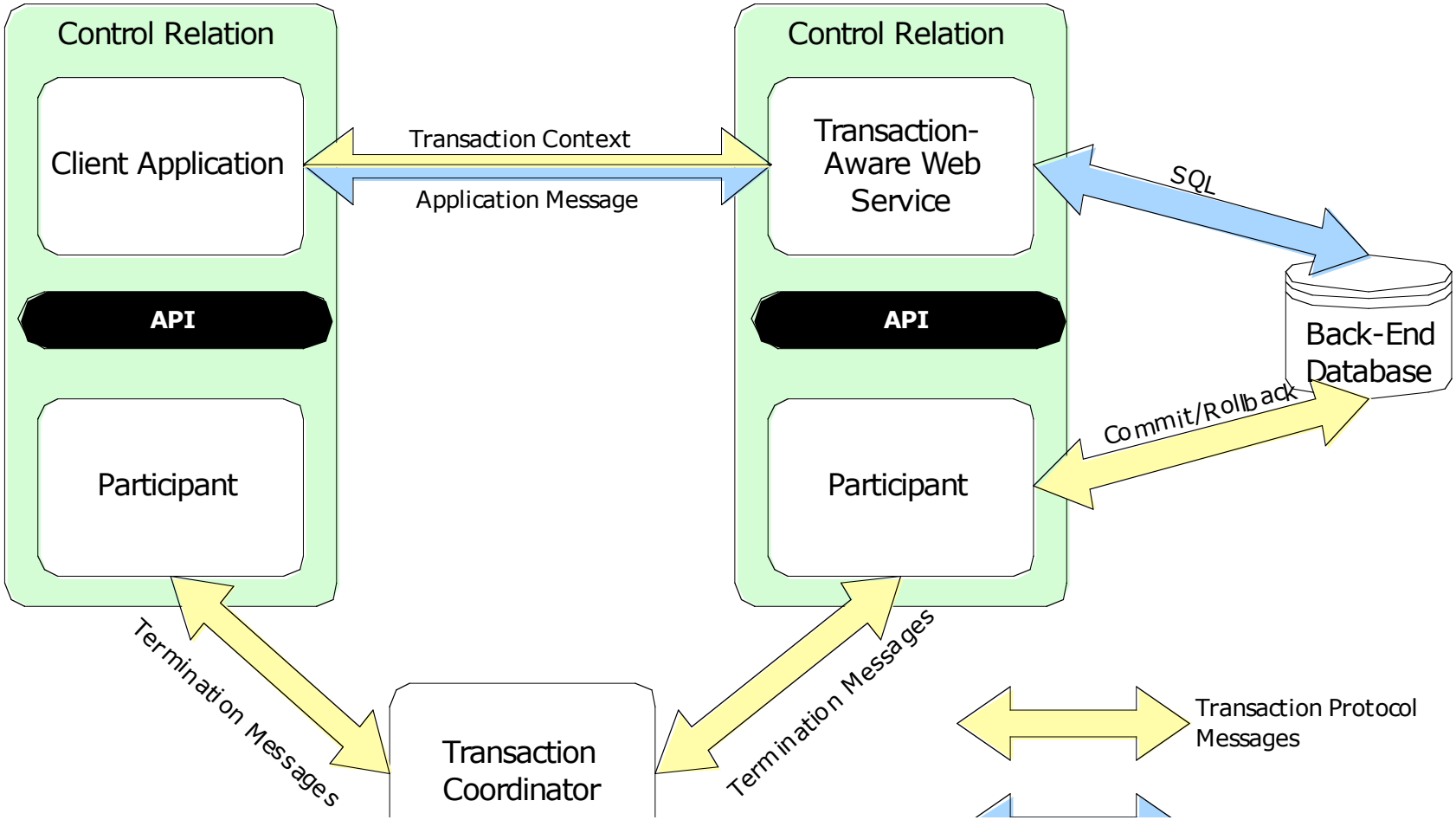
Coordination service



WS-TX protocols

- **Coordinator protocols**
  - Atomic Transaction
    - Completion, DurableTwoPhase, VolatileTwoPhase
  - Business Activity
    - BusinessAgreementWithCoordinatorCompletion, BusinessAgreementWithParticipantCompletion

Services, participants and context



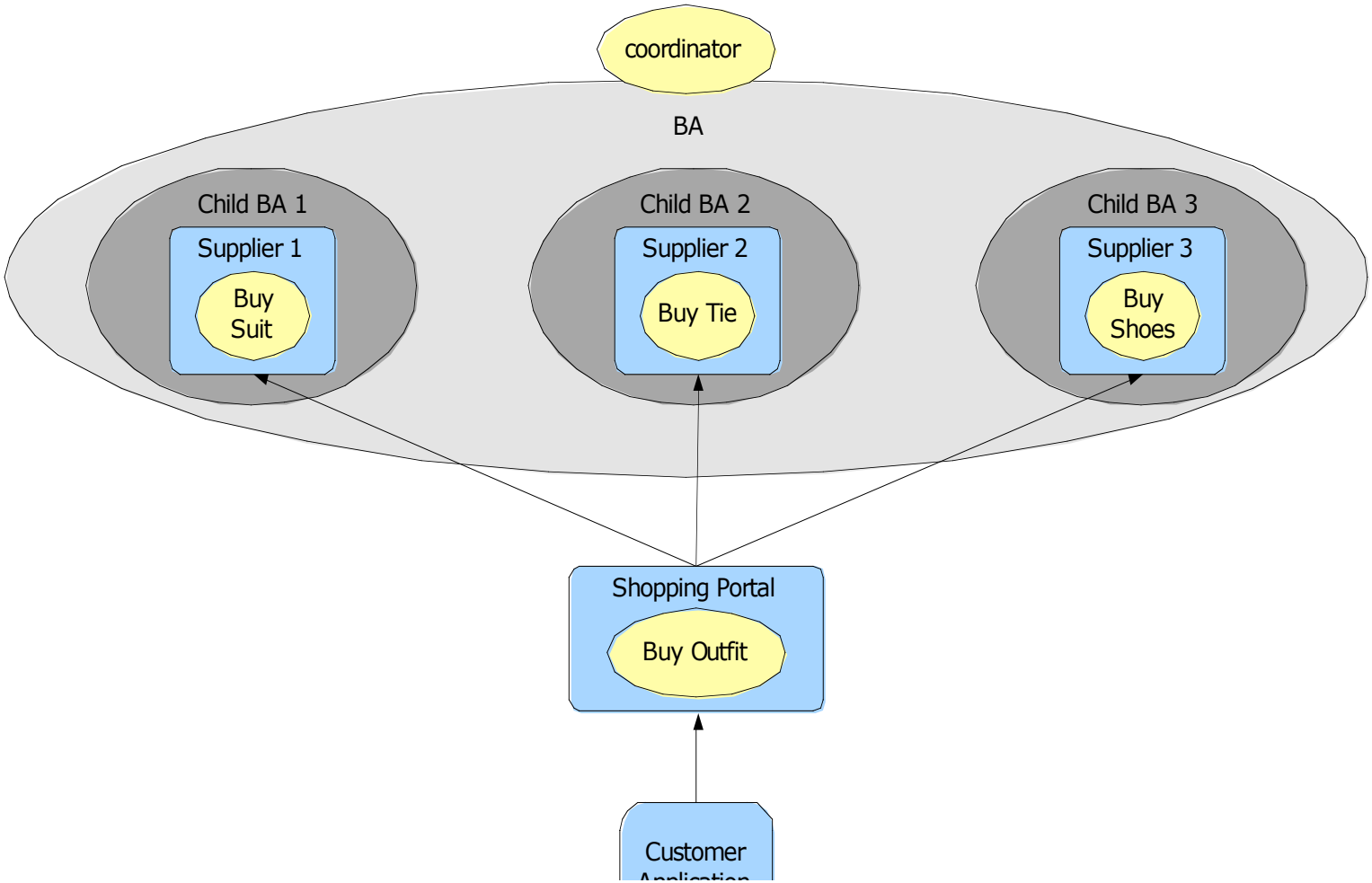
WS-AtomicTransaction

- **Assume ACID transactions**
  - High degree of trust
  - Isolation for duration of transaction
  - Backward compensation techniques
  - Does not allow heuristic outcomes
- **Integration with existing transaction systems**
  - Important to leverage investments
- **Interoperability between transaction systems**
  - Something of a holy grail to date

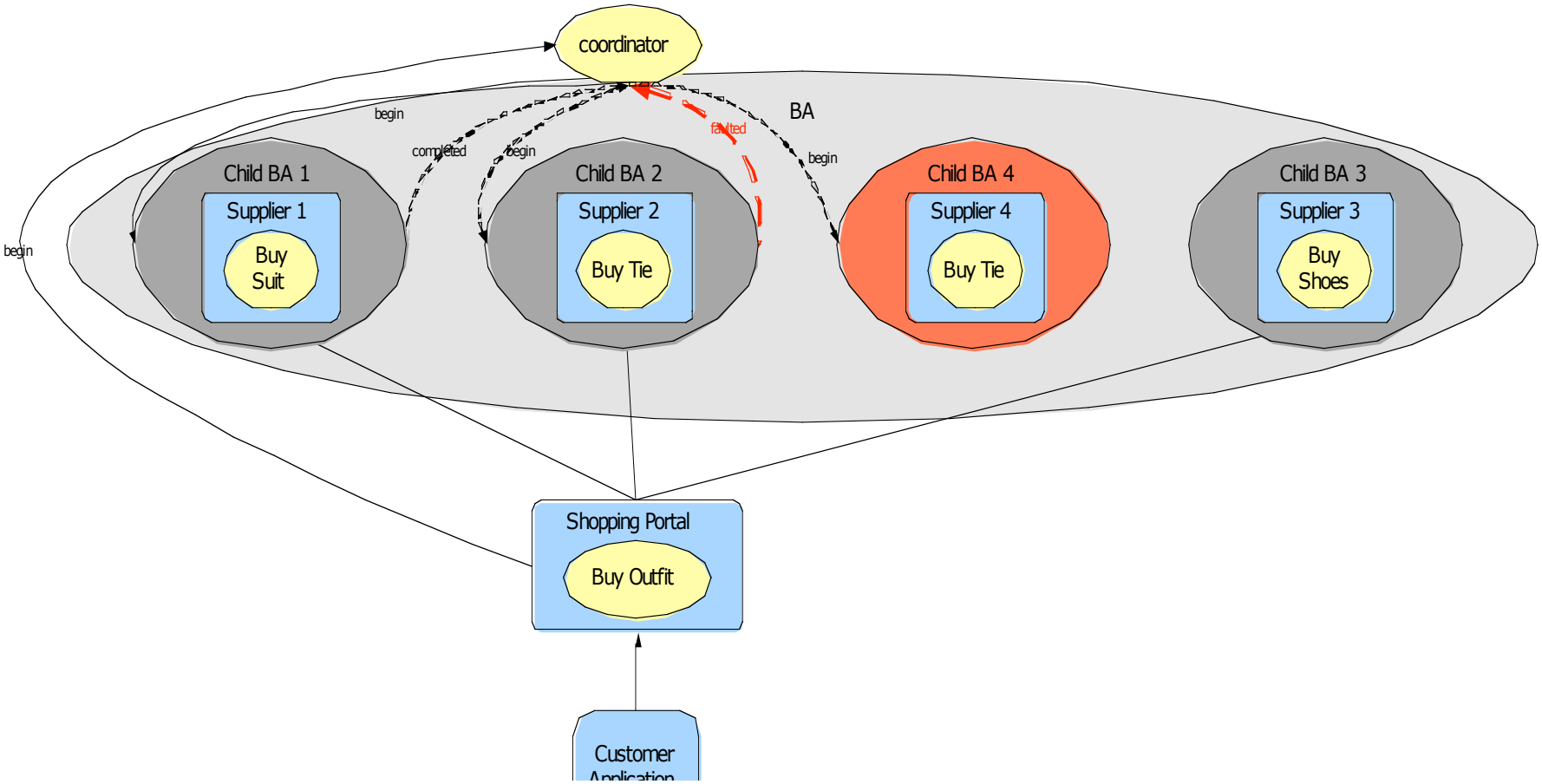
WS-BusinessActivity

- **Workflow-like coordination and management**
- **Business activity can be partitioned into tasks**
  - Parent and child relationships
    - Select subset of children to complete
    - Parent can deal with child failures without compromising forward progress
- **Tasks can dynamically exit a business activity**
- **Tasks can indicate outcome earlier than termination**
  - Up-calls rather than just down-calls

WS-BA example



Compensating BA



What characteristics are right?

- **Need to be able to relax the strict ACID properties**
- **Need to put control of some into hands of service developer**
  - Is consistency (or consensus) important?
- **May need a notion of a central coordinator**
  - But may not!
  - Or something with a fuzzy idea of what's going on
- ***"A comparison of Web services transaction protocols"*, IBM Developer Works, 2003.**

Relaxing isolation

- **Internal isolation or resources should be a decision for the service provider**
  - E.g., commit early and define compensation activities
  - However, it does impact applications
    - Some users may want to know a priori what isolation policies are used
- **Undo can be whatever is required**
  - Before and after image
  - Entirely new business processes

Relaxing atomicity

- **Sometimes it may be desirable to cancel some work without affecting the remainder**
  - E.g., prefer to get airline seat now even without travel insurance
- **Similar to nested transactions**
  - Work performed within scope of a nested transaction is provisional
  - Failure does not affect enclosing transaction
- **However, nested transactions may be too restrictive**

Structuring transactions

- **Could structure transactional applications from short-duration transactions**
  - Release locks early
  - Resulting application may still be required to appear to have “ACID” properties
    - May require application specific means to restore consistency
- **A transactional workflow system could be used to script the composition of these transactions**

Relaxation of consistency

- **ACID transactions (with two-phase commit) are all about strong global consistency**
  - All participants remain in lock-step
  - Same view of transaction outcome (atomic)
- **But that does not scale**
  - Replication researchers have known about this for years
    - Weak consistency replication protocols developed for large scale (number of replicas and physical deployment)
    - Merging of caching and replication protocols
      - Local domains of consistency
    - Cannot “stop the world” and enforce global consistency
  - Some transaction research into this, but industry pushing global consistency
    - Starting to see a change

Heisenberg's Uncertainty Principle

- **Cannot accurately measure both position and momentum of sub-atomic particles**
  - Can know one with certainty, but not the other
  - Non-deterministic measurements
- **Large-scale/loosely-coupled transactional applications suffer the same effect**
  - Can know that all services will eventually see same state, just not when
  - Or at known time can determine state within model/application specific degree of uncertainty
- **Or another way of thinking about it …**
  - No such thing as simultaneity in data space as there isn't in space-time
    - *"Data on the Outside vs. Data on the Inside", by Pat Helland*

No global consensus

- **Split transactions into domains of consistency**
  - Strong consistency within domains
  - Some level of (known) consistency between domains
    - See *"A method for combining replication and caching", Proceedings of International Workshop on Reliable Middleware Systems, October 1999*.
    - *OASIS WS-BusinessProcess specification*, part of OASIS WS-CAF, 2003.
  - Resolve inconsistencies at the business level
    - Don't try and run consensus protocols between domains
- **Consistency related to isolation**
  - Put into the control of the service and application developers

OASIS Business Process

- **All parties reside within *business domains***
  - Recursive structure is allowed
  - May represent a different transaction model
  - No required notion of consistency between domains
- **Business process is split into *business tasks***
  - Execute within domains
  - Compensatable units of work
    - Forward compensation during activity is allowed
      - Keep business process making forward progress
- **Consistency is application (service) dependent**
- **Atomicity (or lack thereof) in the “large” is taken for granted**

SOA or scale?

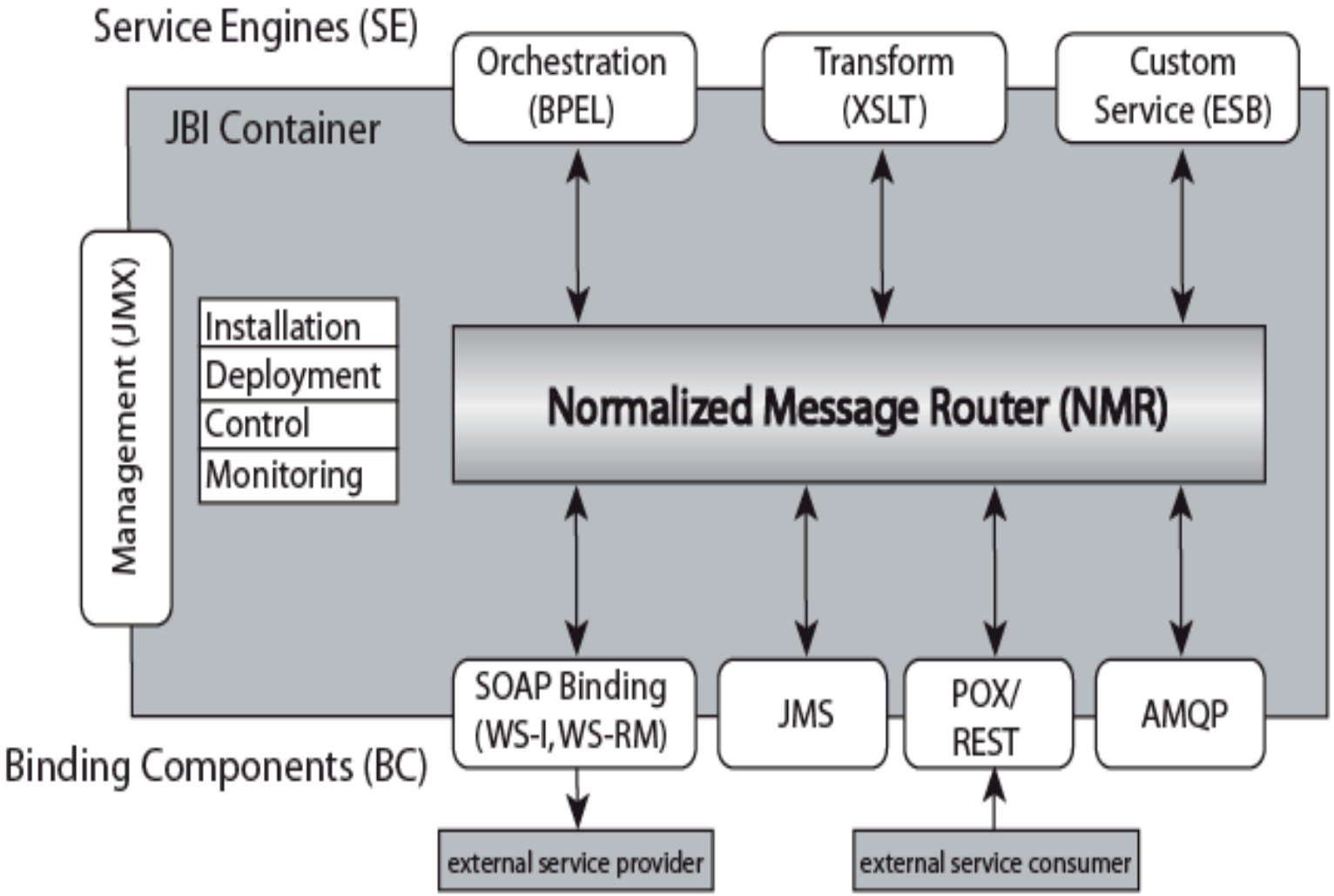
- **Problems with transactions pre-date SOA**
- **Current issues with database technologies are not SOA specific either**
- **Problems are two-fold**
  - Scalability (size and geographic distributed nature)
  - Control over the infrastructure/services
    - Trust comes into this too
- **Much research in the 1990's**
- **SOA (and Web Services) bring this to the foreground**
  - REST would be just as appropriate

Future directions

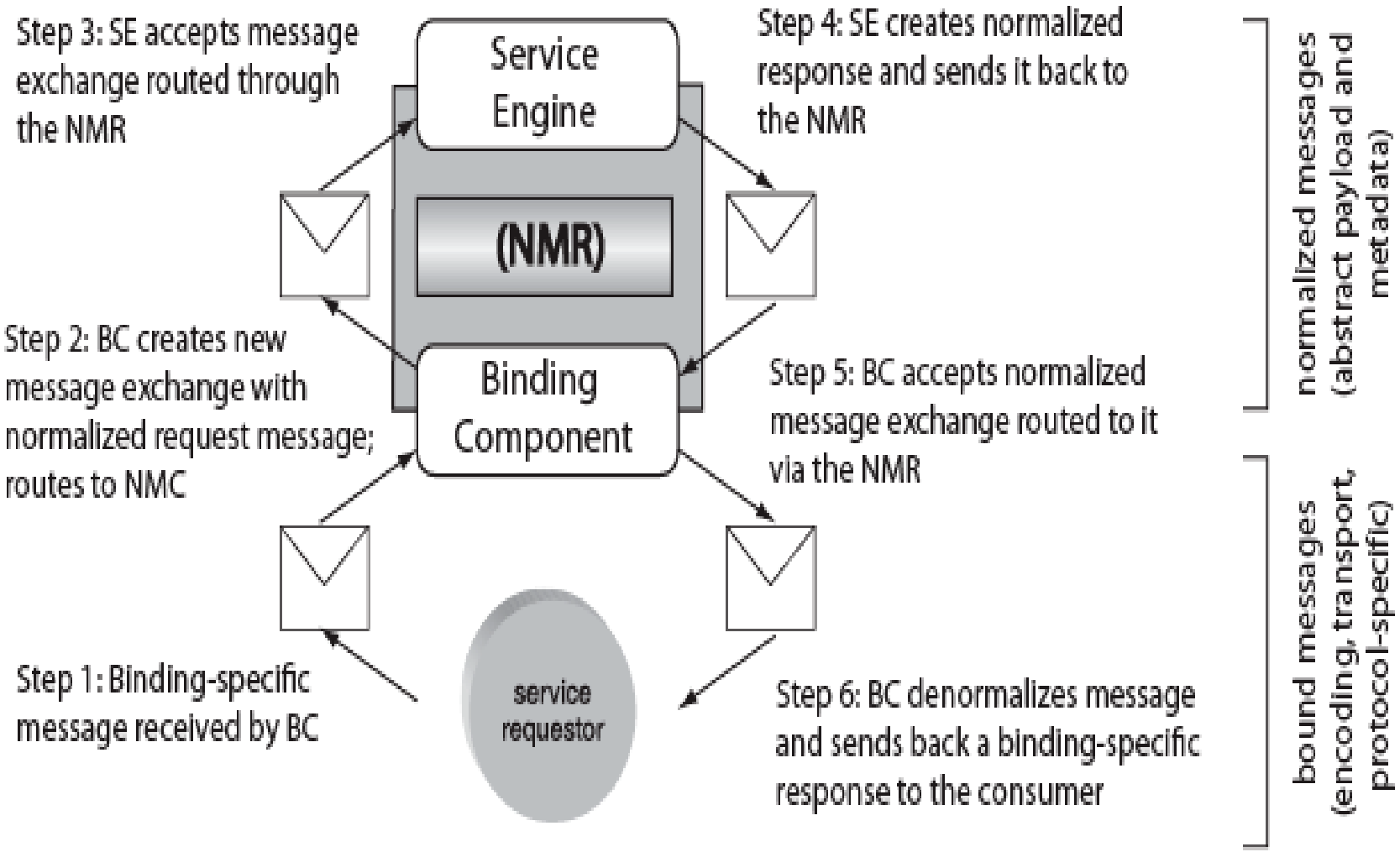
- **One size does not fit all!**
- **Business domains will impose different requirements on implementers**
  - Essentially construct domain-specific models
  - Real-time
- **The range and requirements for such extended models are not yet known**
  - Do not restrict implementations because we don't know what we want yet
- **Still a very active area of research and development**

Standards

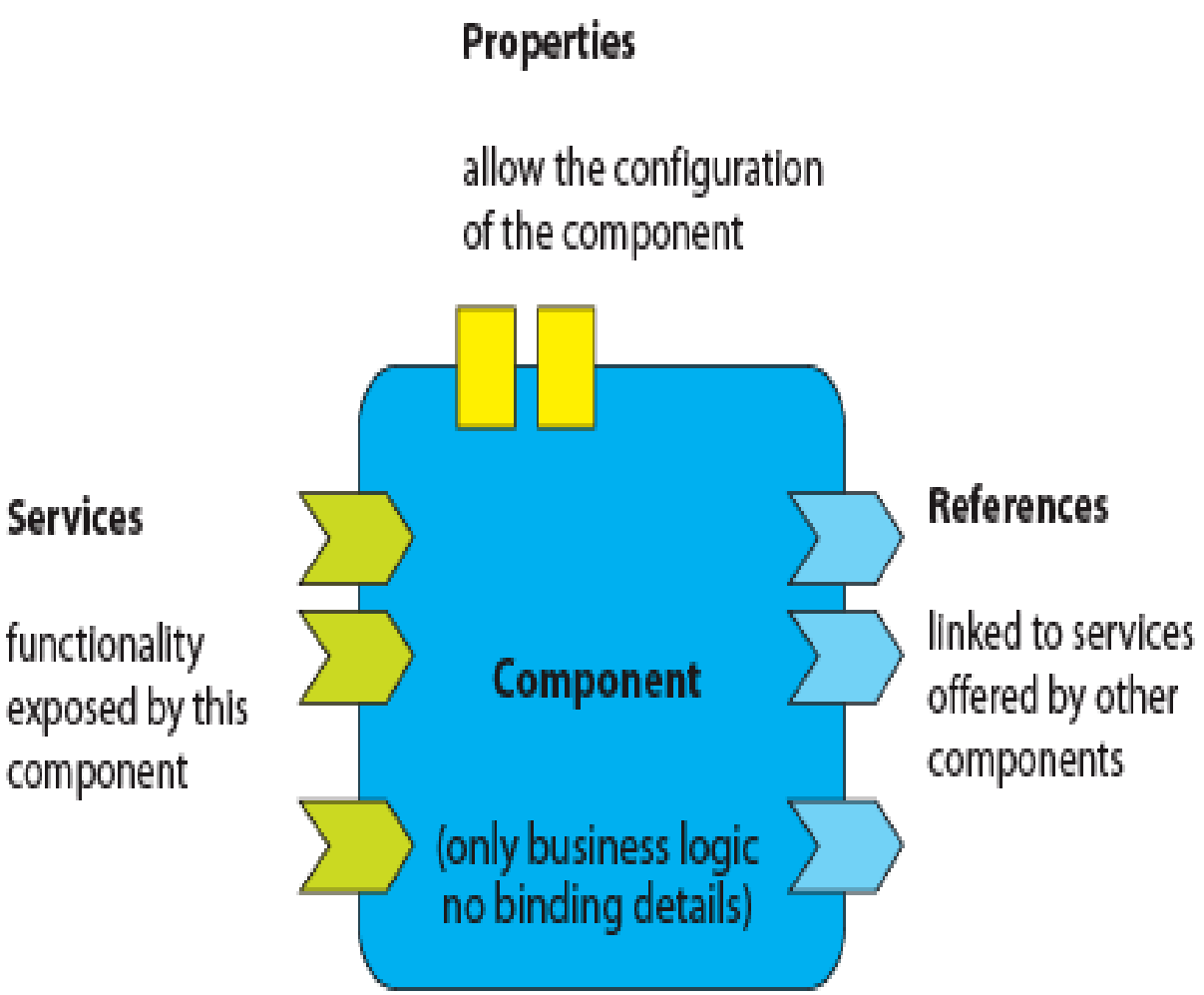
JBI


Service Engines (SE)
Orchestration (BPEL)
Transform (XSLT)
Custom Service (ESB)
JBI Container
Management (JMX)
Installation
Deployment
Control
Monitoring
Normalized Message Router (NMR)
SOAP Binding (WS-I, WS-RM)
JMS
POX/ REST
AMQP
Binding Components (BC)
external service provider
external service consumer

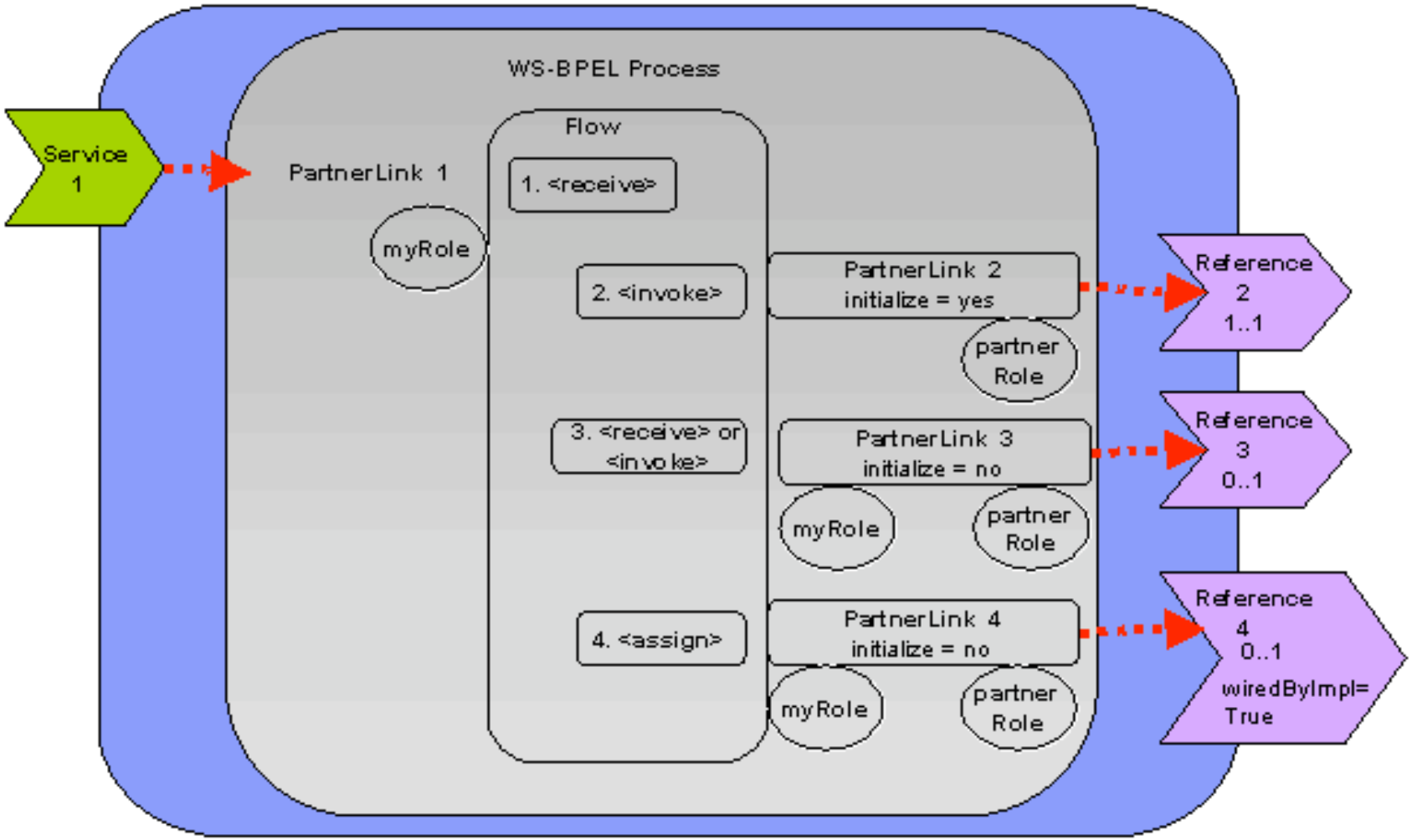
Message handling with JBI



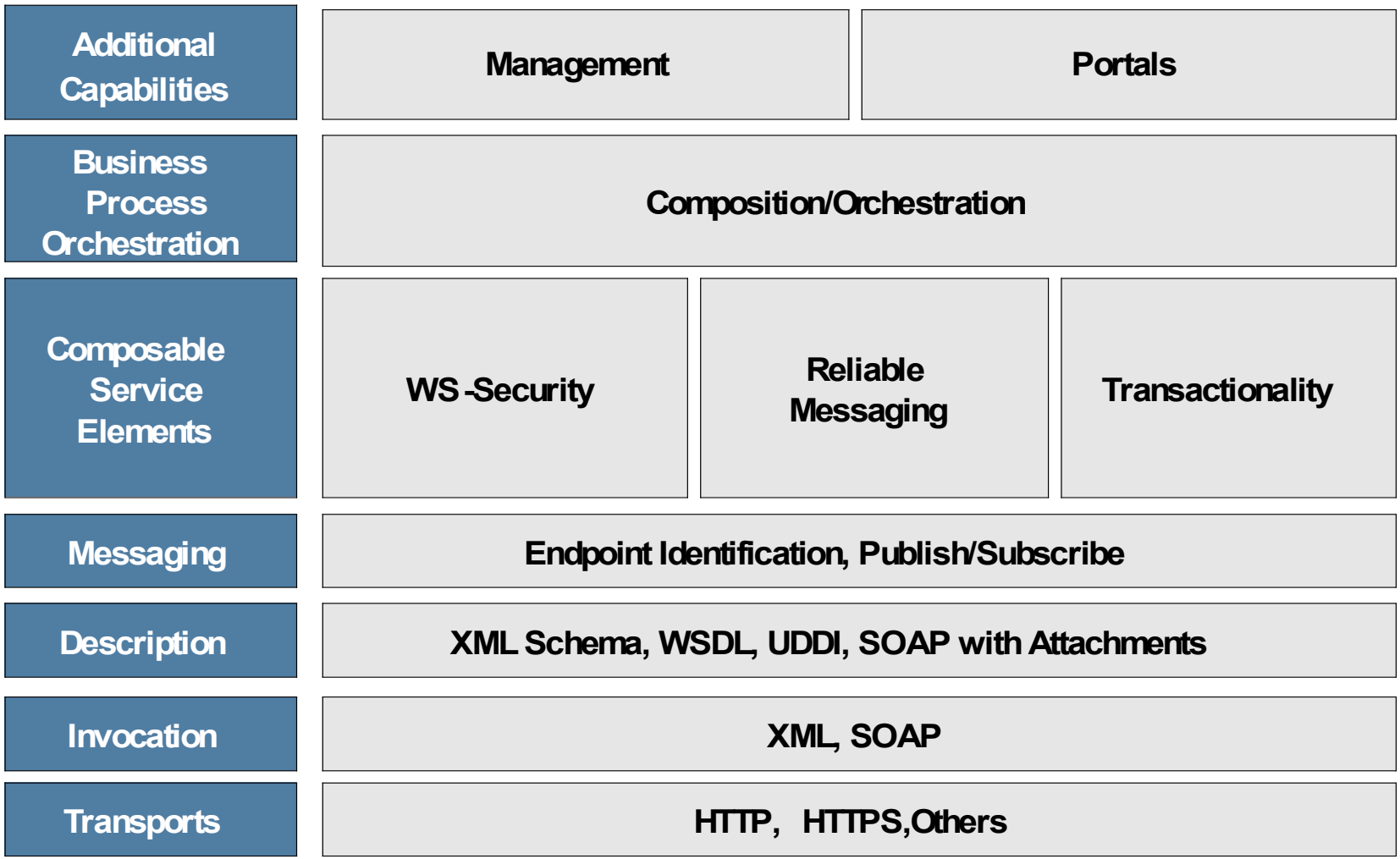
SCA


Properties
allow the configuration
of the component
Services
functionality
exposed by this
component
Component
(only business logic
no binding details)
References
linked to services
offered by other
components

SCA and BPEL


WS-BPEL Process
Flow
Service 1
PartnerLink 1
myRole
1. <receive>
2. <invoke>
3. <receive> or <invoke>
4. <assign>
PartnerLink 2
initialize = yes
partner Role
PartnerLink 3
initialize = no
myRole
partner Role
PartnerLink 4
initialize = no
myRole
partner Role
Reference 2 1..1
Reference 3 0..1
Reference 4 0..1 wiredByImpl= True

WS-I


Additional Capabilities
Management
Portals
Business Process Orchestration
Composition/Orchestration
Composable Service Elements
WS-Security
Reliable Messaging
Transactionality
Messaging
Endpoint Identification, Publish/Subscribe
Description
XML Schema, WSDL, UDDI, SOAP with Attachments
Invocation
XML, SOAP
Transports
HTTP, HTTPS,Others

Any questions?