SOA BOF

Dr Mark Little

Red Hat

Overview

- SOA in a nutshell
  - Degrees of coupling
  - The component triad
- Relationship to WS-*
- The JBoss SOA Platform
  - Registries and repositories
    - BRMS
  - Message delivery and transformation
  - Service orchestration

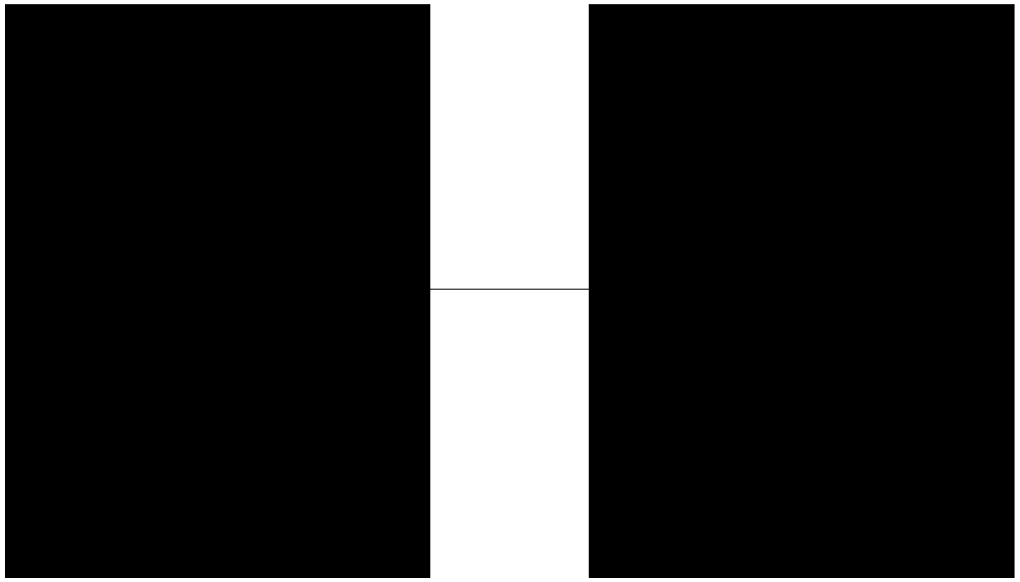
Remote Procedure Call

- 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)***

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***

Stub example



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

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

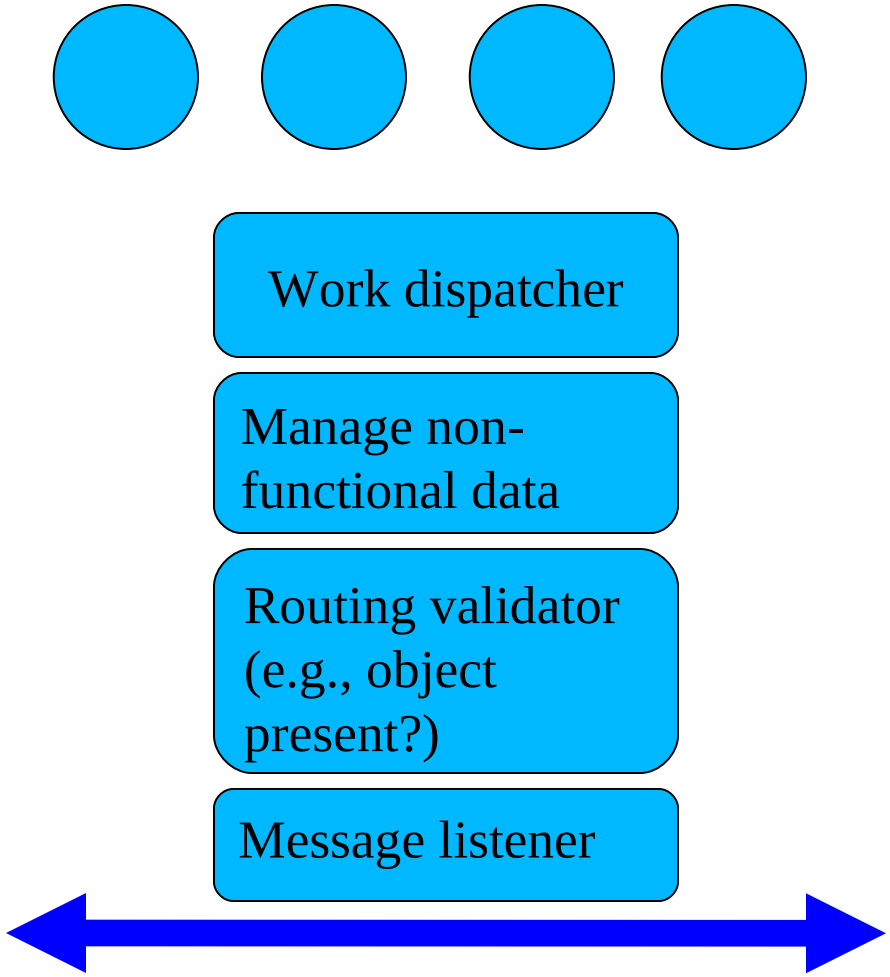
Data Loose Coupling

- **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

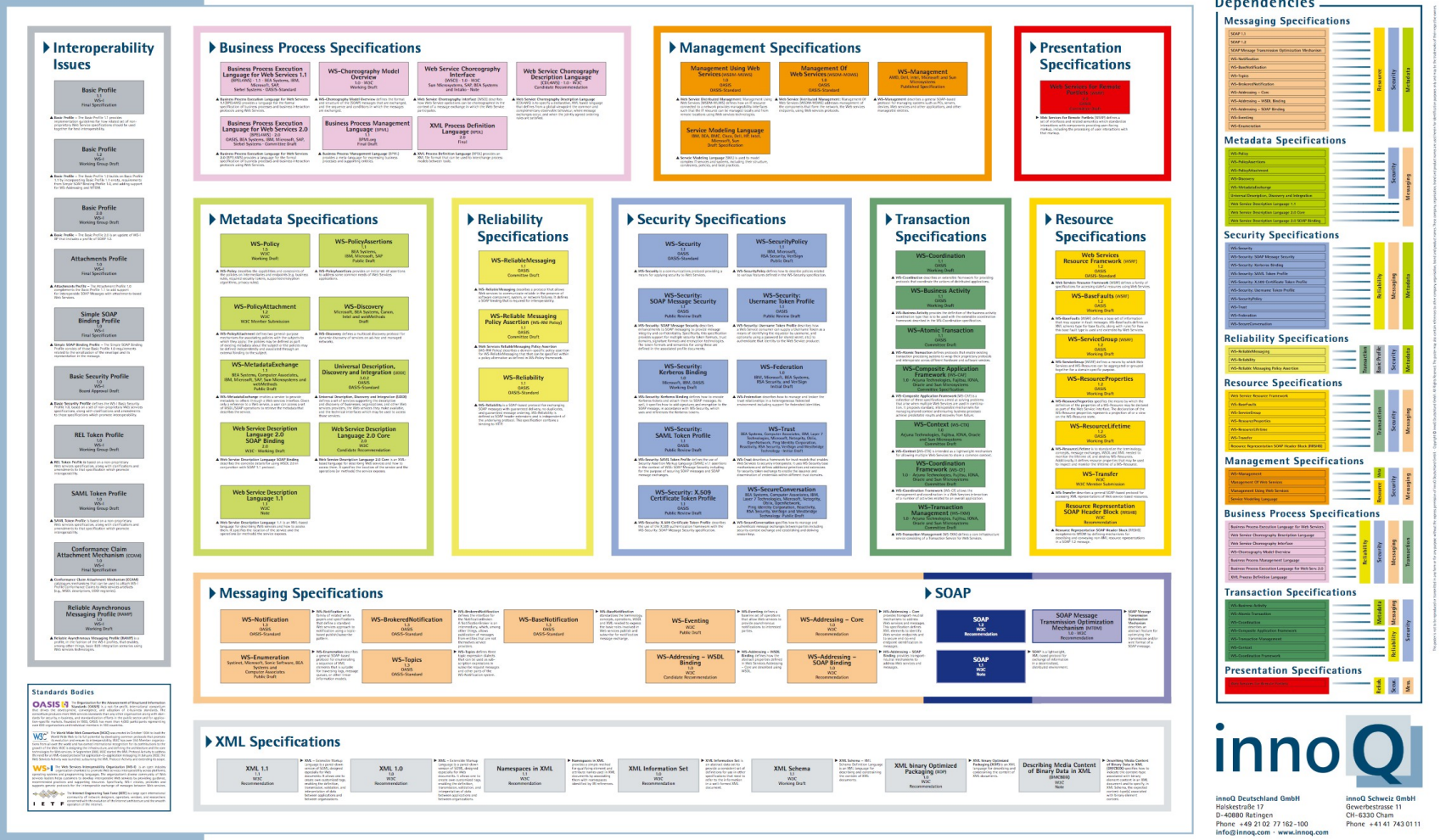
Service stack example



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

Relationship to WS-*



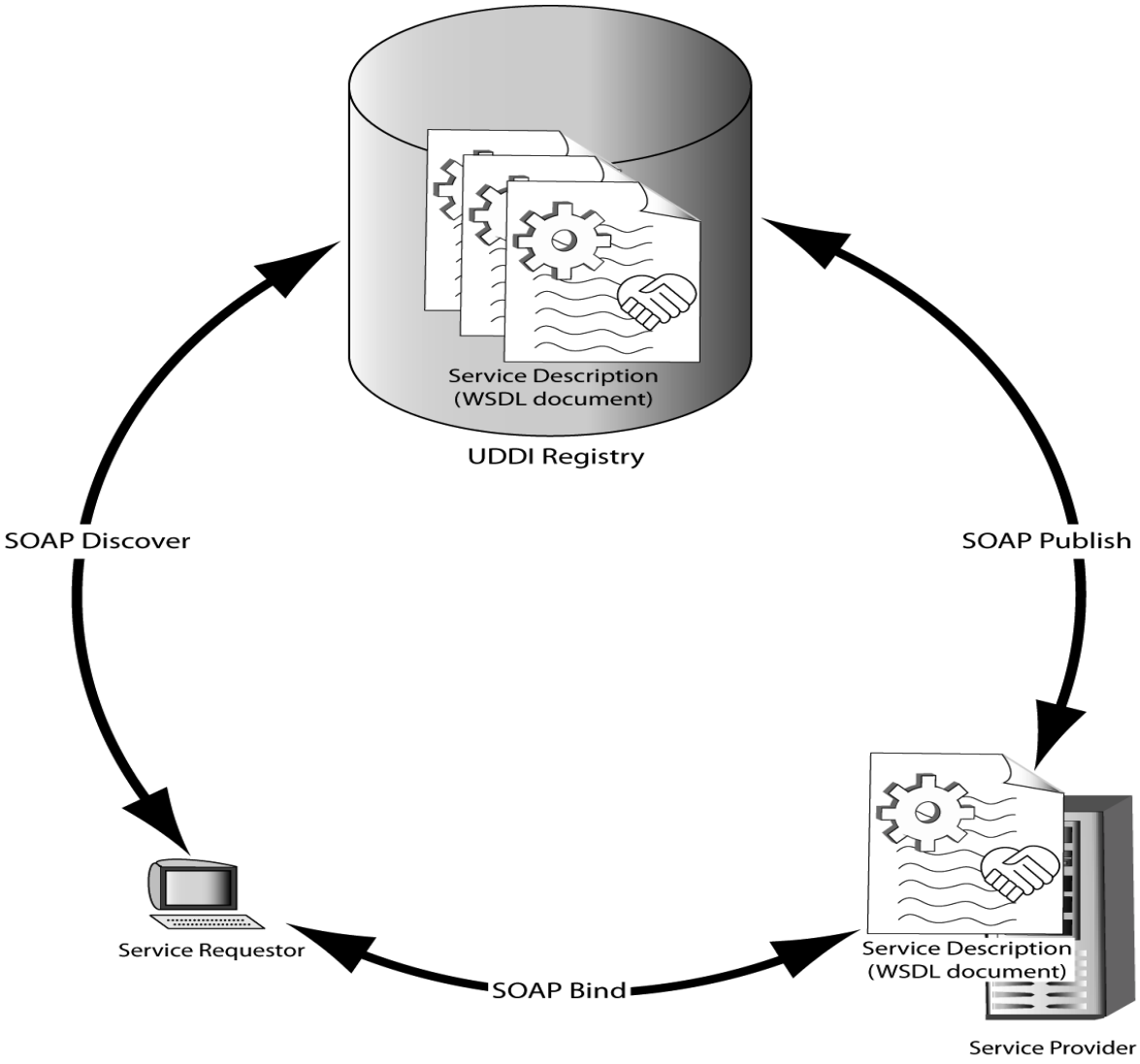
Fortunately ...

- 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**

Component triad



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

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
- SOA-P supports both approaches
  - jBPM
  - WS-BPEL

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

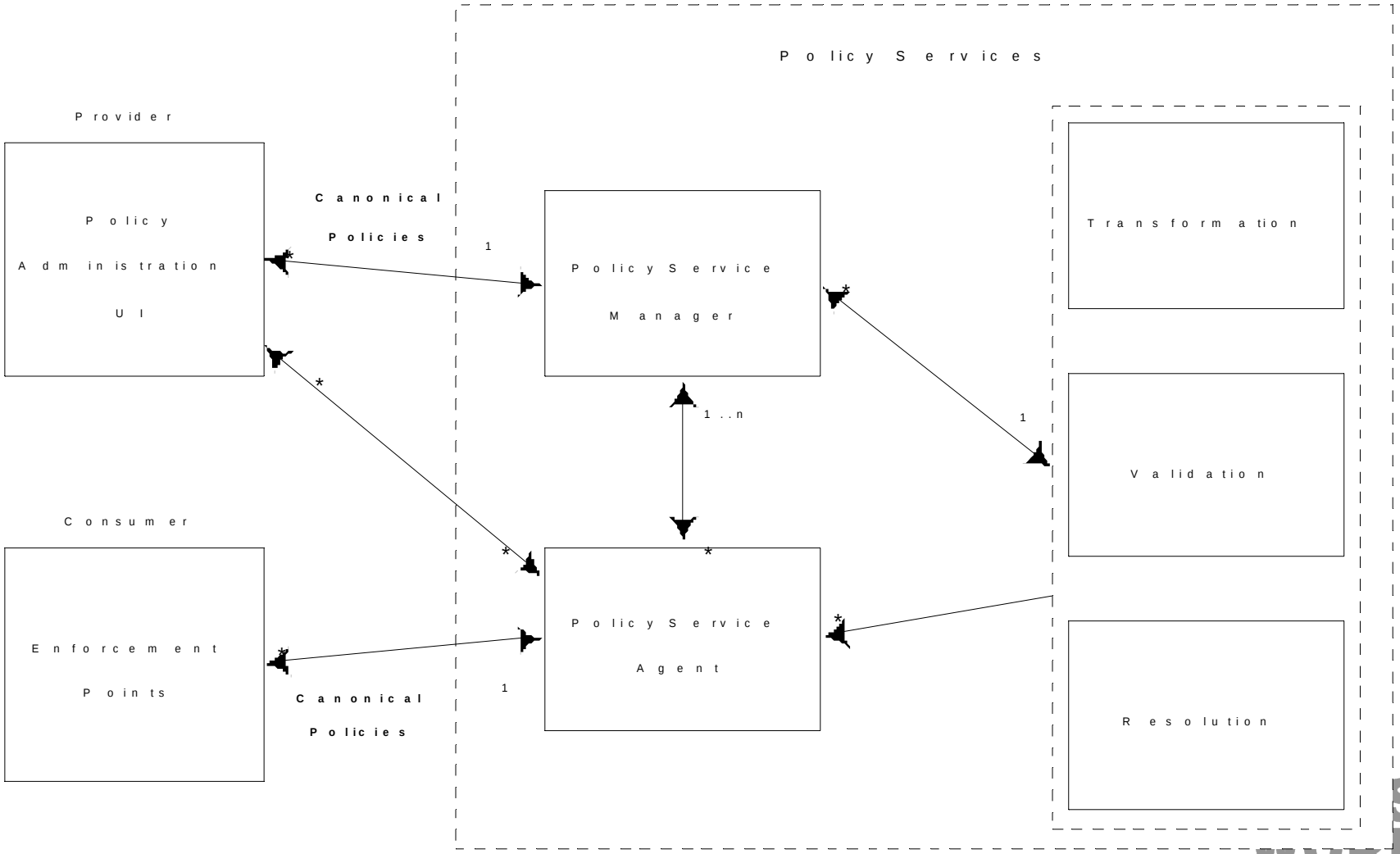
Contracts, policies and SLAs

- **“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**

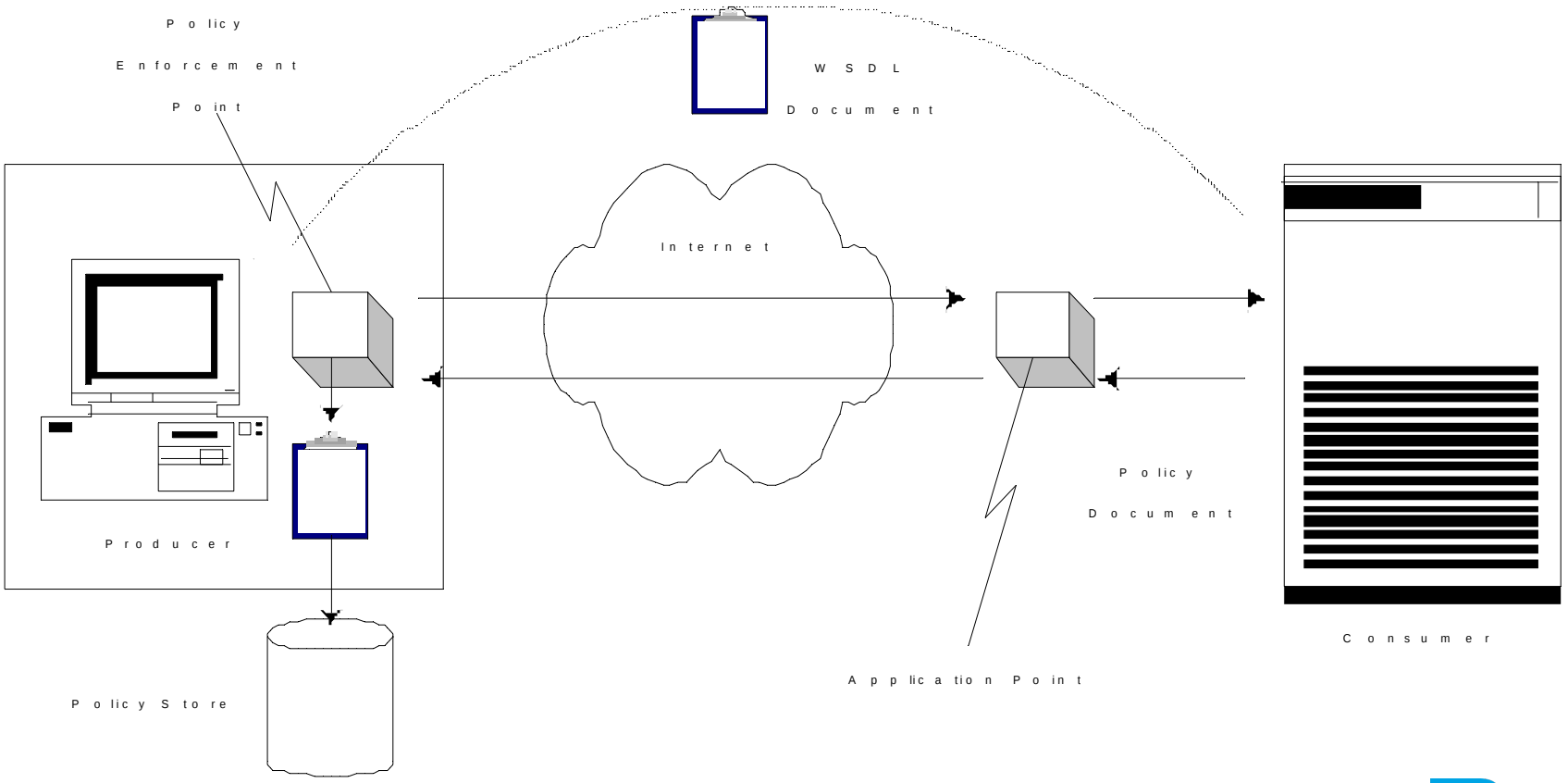
Policies

- 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



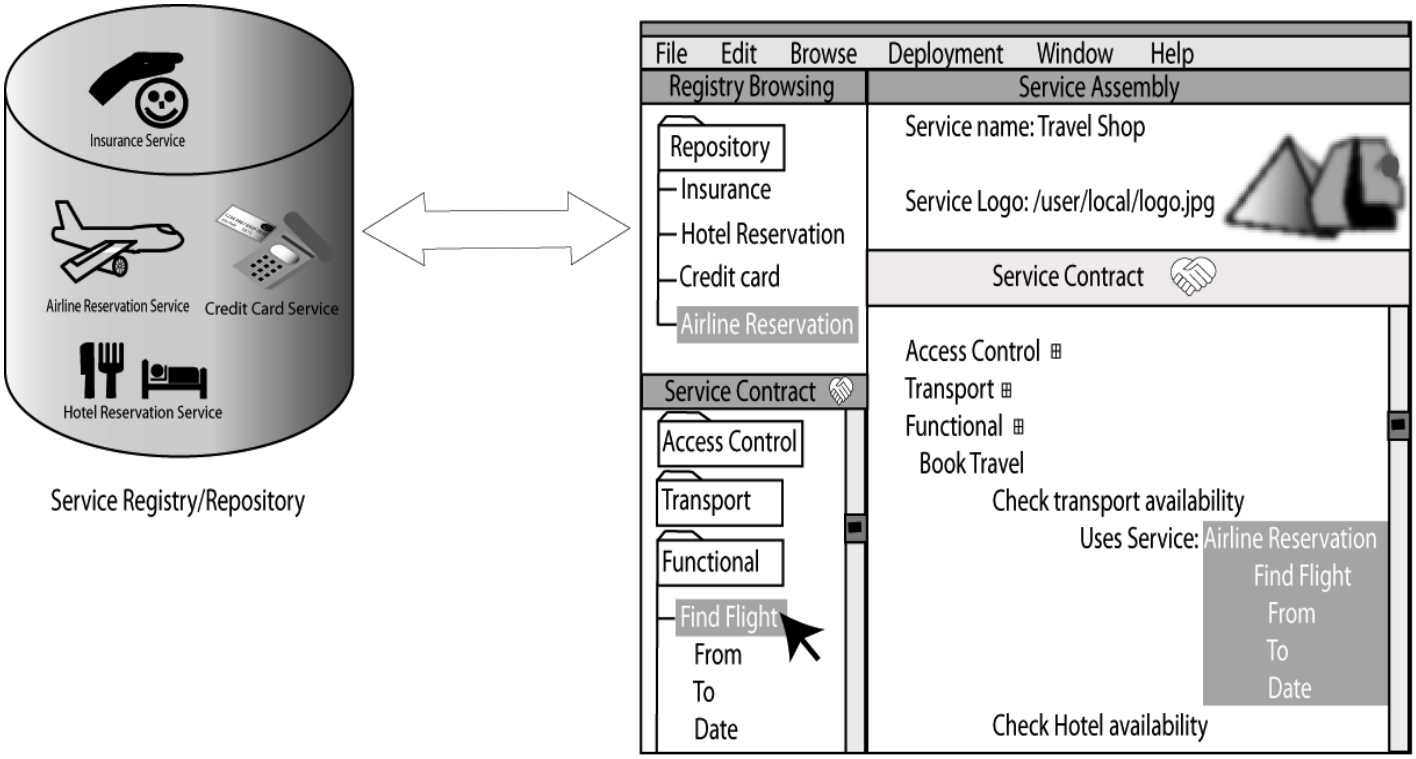
Policy Enforcement



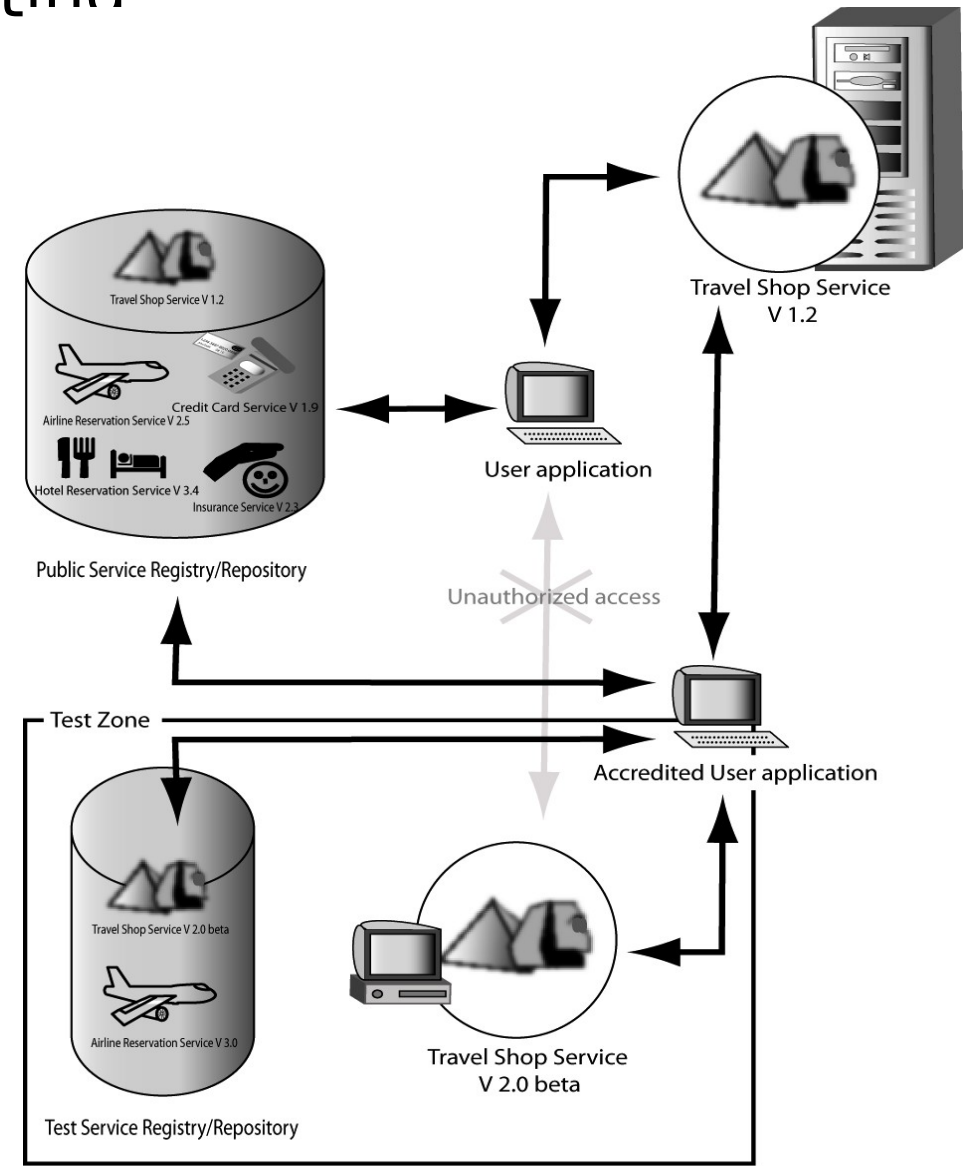
Other meta-data

- 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.

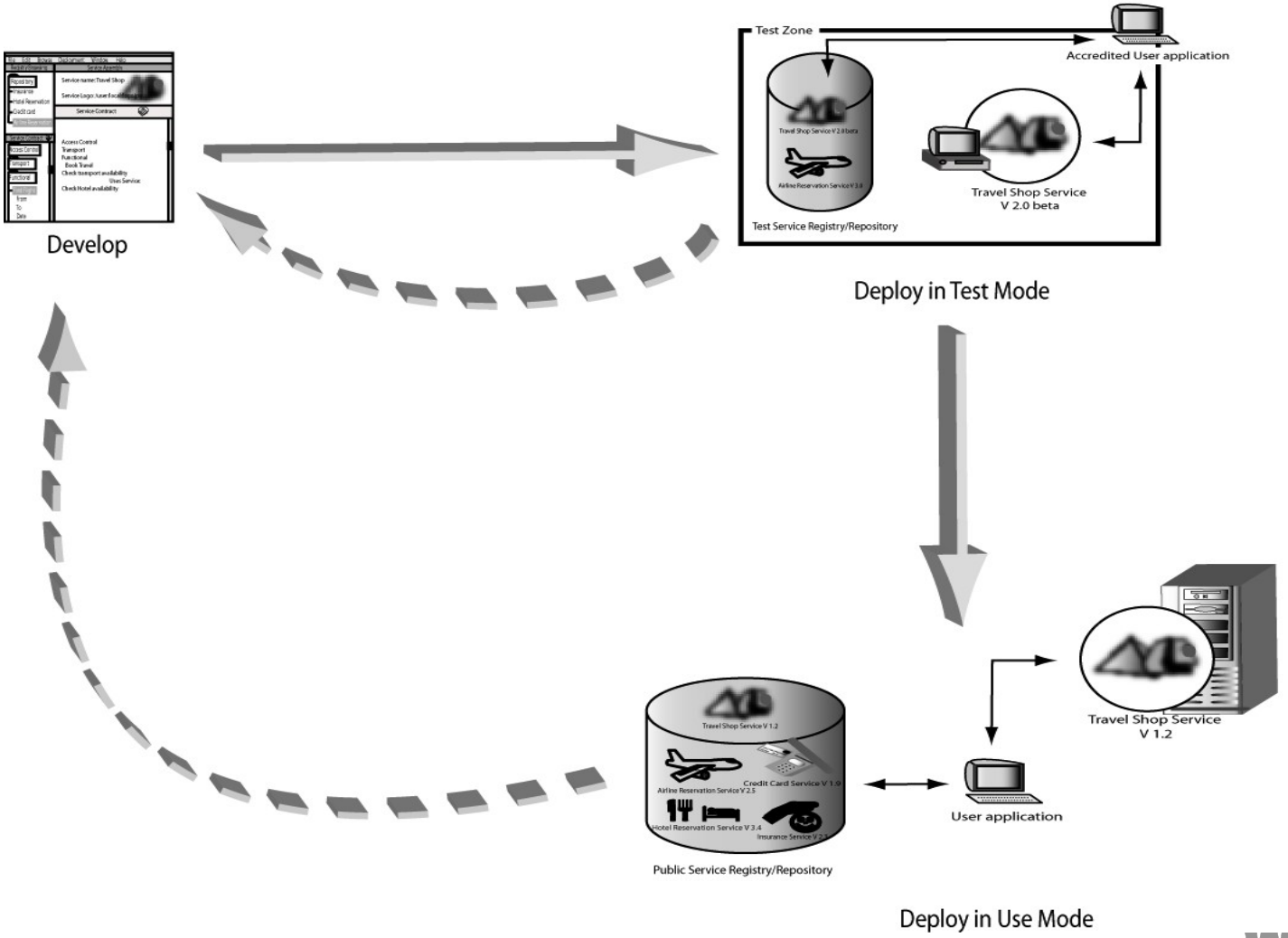
Design-time service discovery



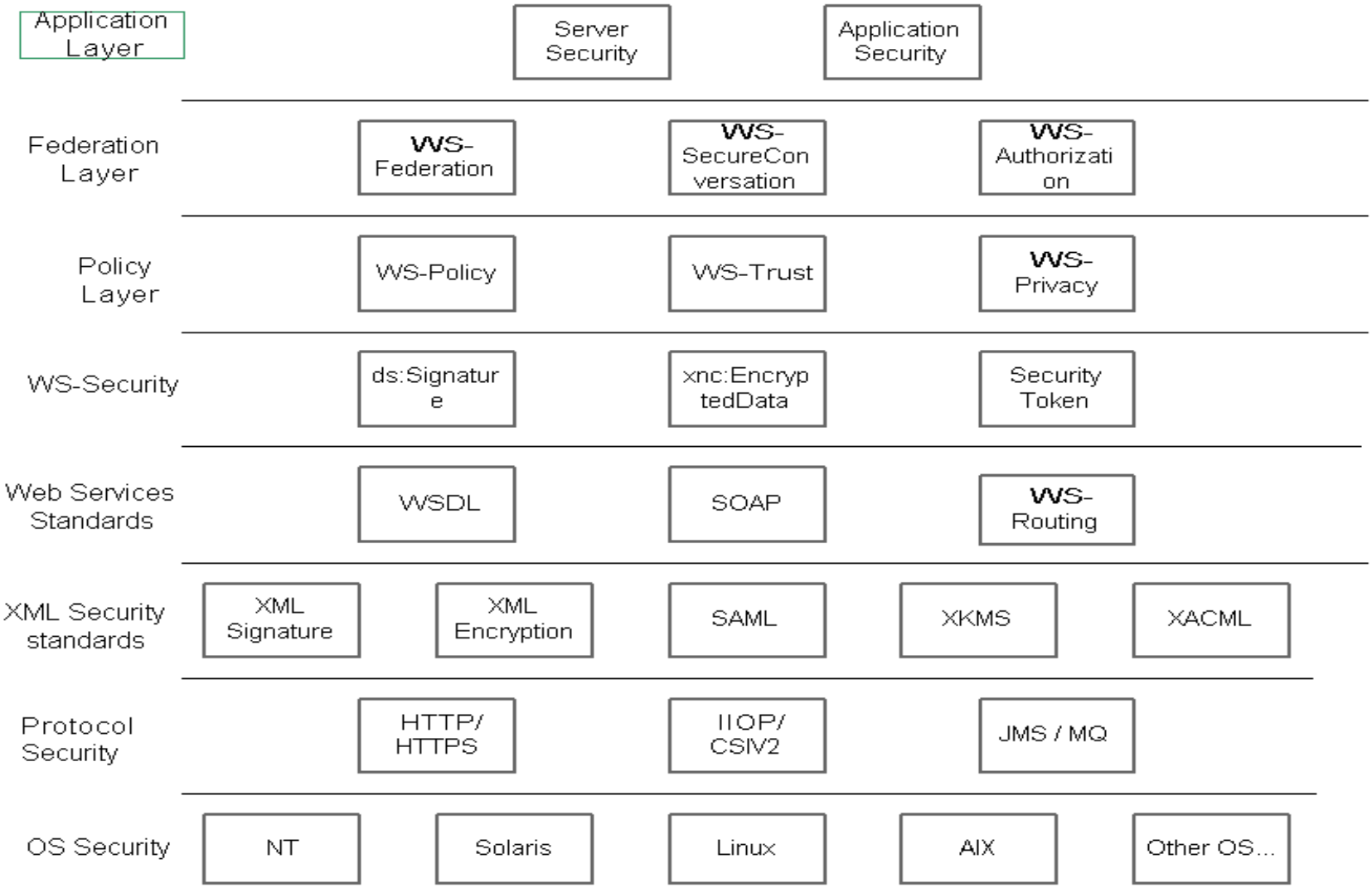
Service testing



Service deployment



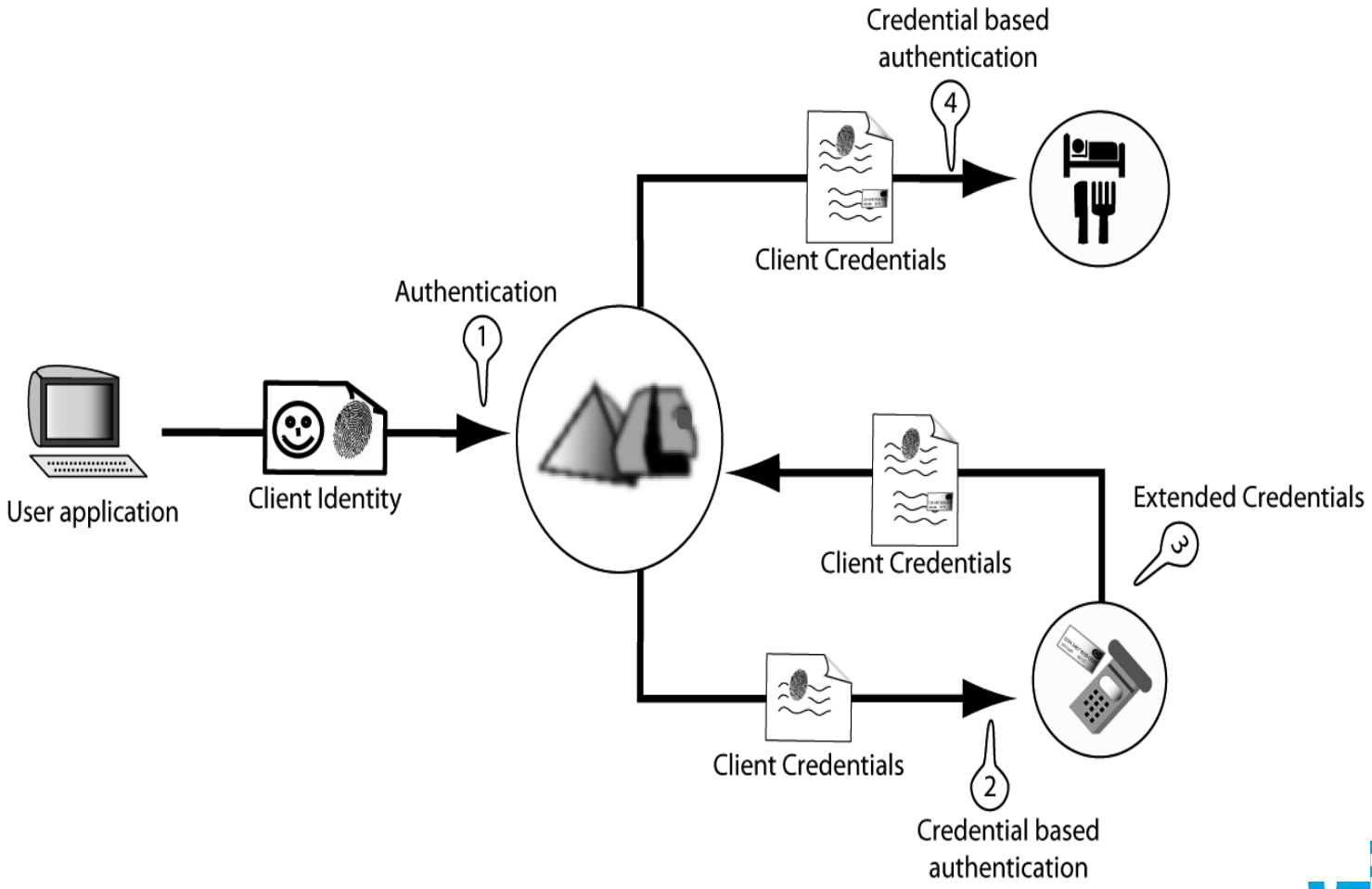
Security



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**