Service Oriented Architecture

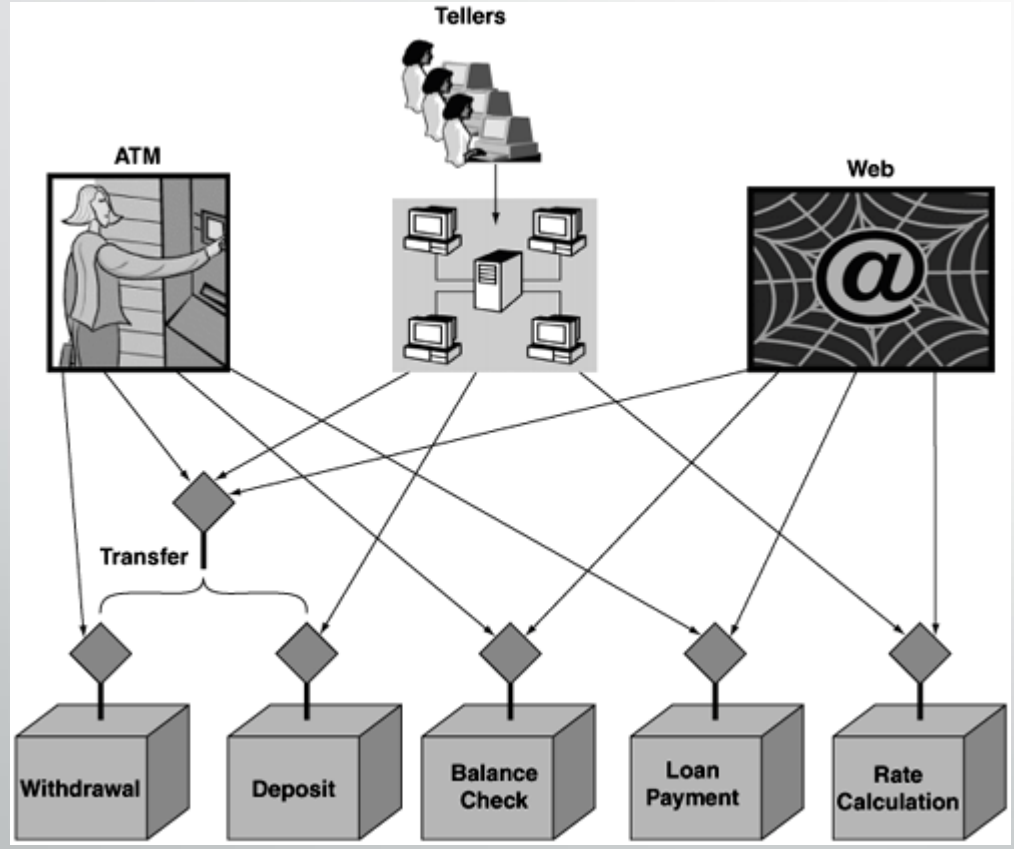
PART I – INTRODUCING SOA

Fundamental SOA

1. The term "service-oriented" has existed for some time, it has been used in different contexts and for different purposes
2. Approach
    1. the logic required to solve a large problem can be better constructed, carried out, and managed if it is decomposed into a collection of smaller, related pieces.
    2. Each of these pieces addresses a concern or a specific part of the problem.
3. **This approach transcends technology and automation solutions. It is an established and generic theory that can be used to address a variety of problems.**

A service oriented analogy

Service Oriented and Composition



The definition of software services aligns with the business services that a bank offers to ensure smooth business operations and to help realize strategic goals such as providing ATM and Web access to banking services in addition to providing them in the branch office.

Architecture

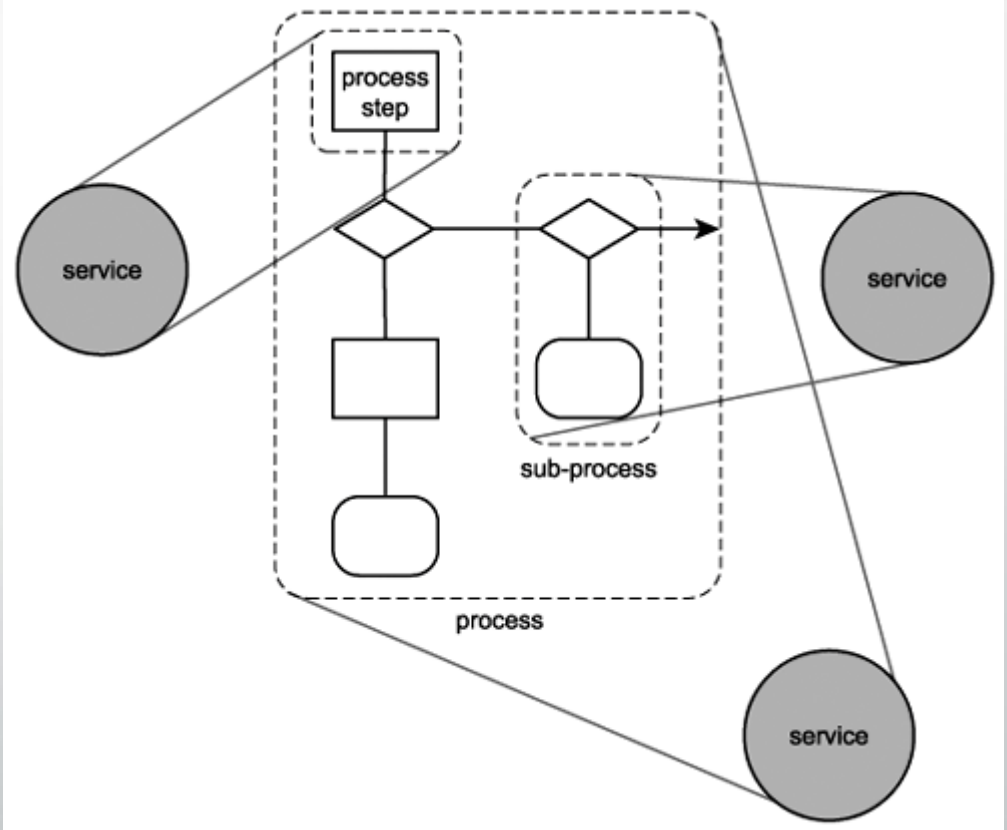
- When coupled with "architecture," service-orientation takes on a technical connotation.
- **"Service-oriented architecture" is a term that represents a model in which automation logic is decomposed into smaller, distinct units of logic.**
- Collectively, these units comprise a larger piece of business automation logic. Individually, these units can be distributed.

Services oriented

- **Business Community**
  - impose dependencies → inhibit the potential of individual businesses.
  - empowering businesses to self-govern their individual services → allow them to evolve and grow relatively independent from each other.
  - Independence within our business outlets, and adhesion to certain baseline conventions
  - standardization key aspects of each business for the benefit of the consumers without significantly imposing on the individual business's ability to exercise self-governance.
- **Service-oriented architecture (SOA)**
  - encourages individual units of logic to exist autonomously yet not isolated from each other.
  - Units of logic are still required to conform to a set of principles that allow them to evolve independently, while still maintaining a sufficient amount of commonality and standardization.
  - **Within SOA, these units of logic are known as *services*.**

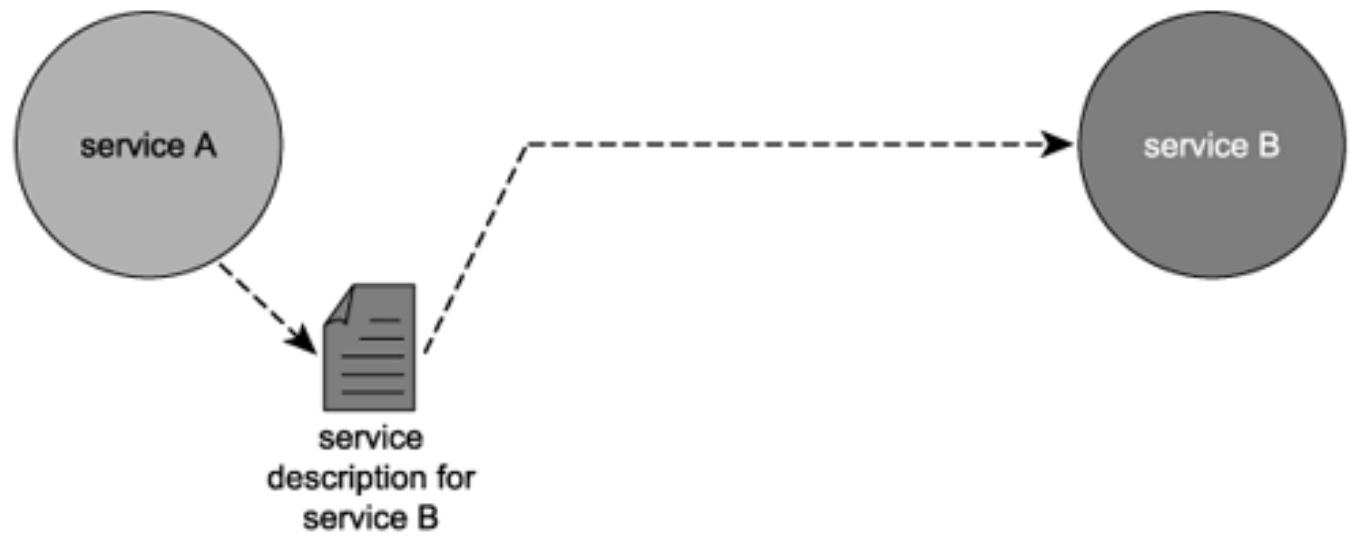
How services encapsulate logic

- each service can encapsulate
  - a task performed by an individual step
  - a sub-process comprised of a set of steps.
- A service can even encapsulate the entire process logic.



How services relate (I)

- The relationship between services is based on an understanding that for services to interact, they must be aware of each other. → ***service descriptions***.
- A service description establishes
  - the name of the service
  - the data expected and returned by the service.



How services relate (II)

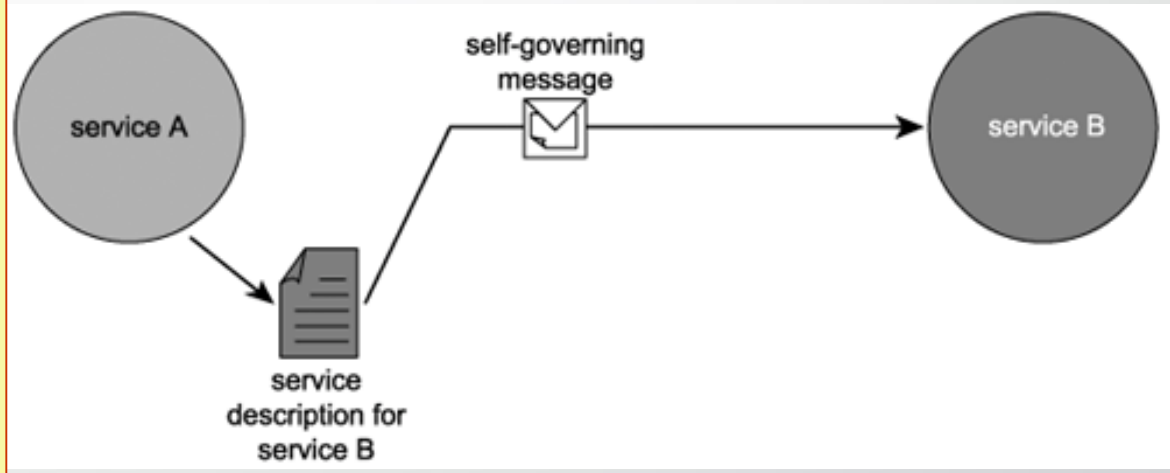
- The manner in which services use service descriptions results in a relationship classified as ***loosely coupled***.
- A communications framework capable of preserving their loosely coupled relationship is therefore required. One such framework is **messaging**.

Challenges with Tight Coupling

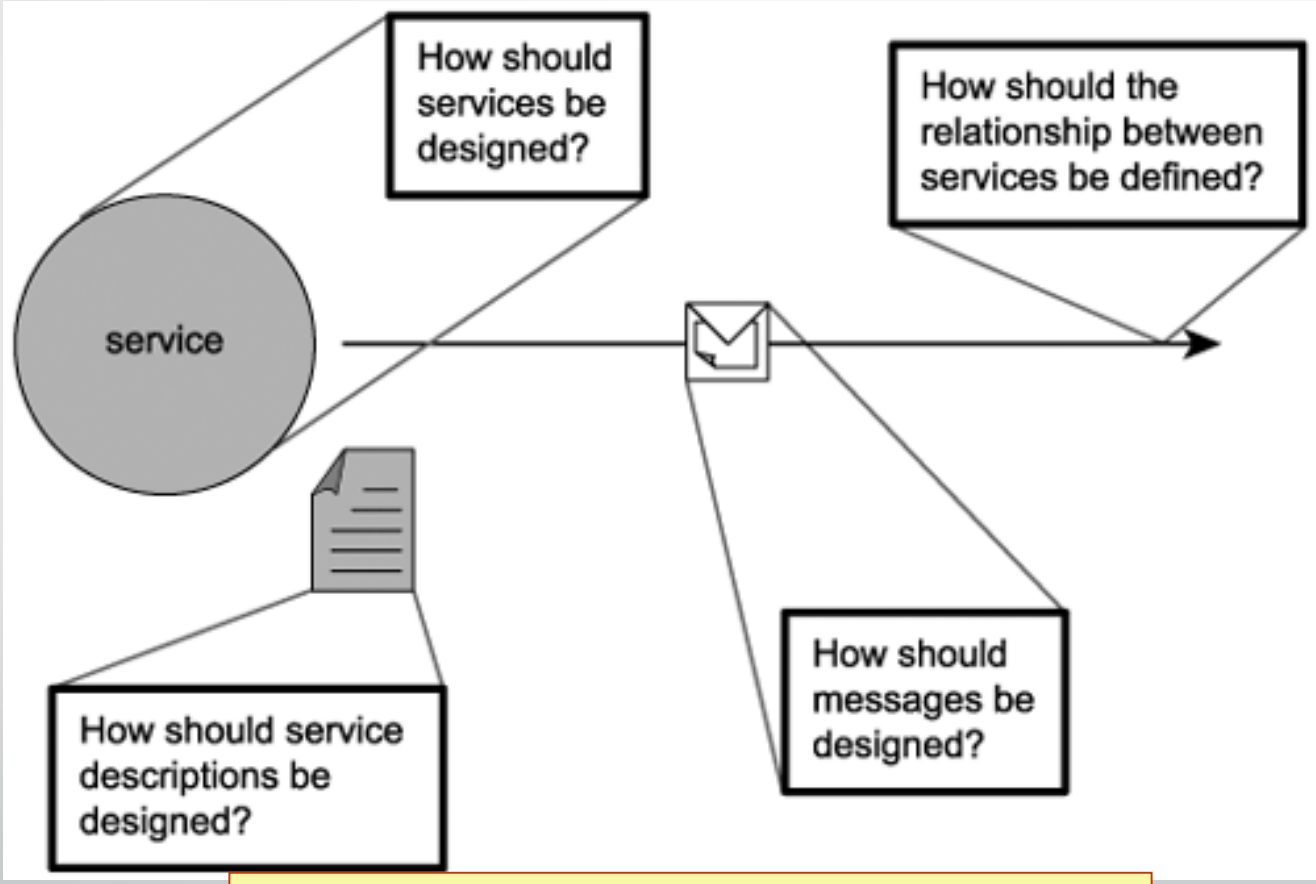
- It's costly to maintain
- Slow and costly to change
- Cost and complexity compounded by multi-party scenarios such as B2B or integration with the public sector
- Cost and complexity of managing and changing a tightly coupled architecture makes business agility difficult (IT can't keep up with business needs, but it's not their fault)
- **Does not support reuse!**

How services communicate

- After a service sends a message, it loses control of what happens to the message thereafter.
- That is why we require messages to exist as "**independent units of communication**."
- Messages, like services, should be autonomous



How services are designed (I)



DISTINCT DESIGN APPROACH

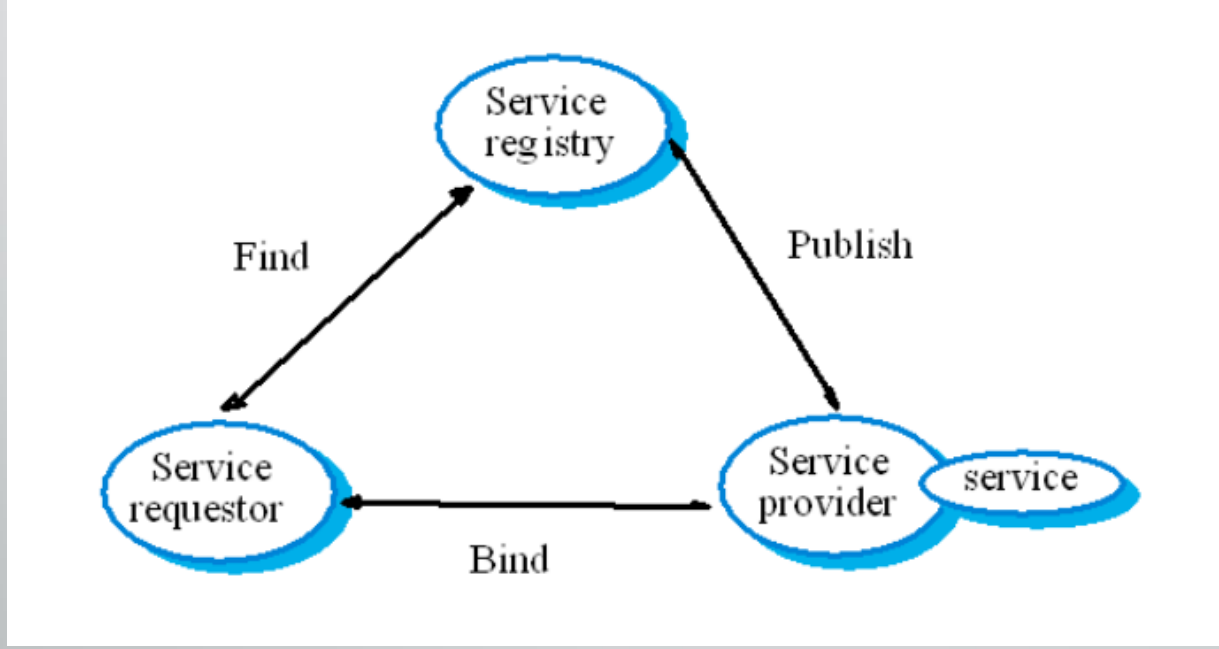
How services are designed: key aspects

- **Loose coupling**: Services maintain a relationship that and only requires that they retain an awareness of each other.
- **Service contract**: Services adhere to a communications agreement
- **Autonomy**: Services have control over the logic they encapsulate.
- **Abstraction**: Services hide logic from the outside world.
- **Reusability**: Logic is divided into services with the intention of promoting reuse.
- **Composability**: Collections of services can be coordinated and assembled to form composite services.
- **Discoverability**: Services are designed to be outwardly descriptive so that they can be found and assessed via available discovery mechanisms.

How services are built

- The term "service-oriented" and various abstract SOA models existed before the arrival of Web services.
- However, no one technology advancement has been so suitable and successful in manifesting SOA than Web services.
- All major vendor platforms currently support the creation of service-oriented solutions, and most do so with the understanding that the SOA support provided is based on the use of Web services.

Primitive SOA



represents a baseline technology architecture that is supported by current major vendor platforms

Contemporaney SOA

- Major software vendors are continually conceiving new Web services specifications and building increasingly powerful XML and Web services support into current technology platforms.
- The result is an extended variation of service-oriented architecture we refer to as contemporary SOA.

Summary 1.1

- SOA and service-orientation are implementation paradigms that can be realized with any suitable technology platform.
- Primitive SOA model represents a mainstream variation of SOA based solely on Web services and common service-orientation principles.

Common Characteristics

SOA is fundamentally autonomous

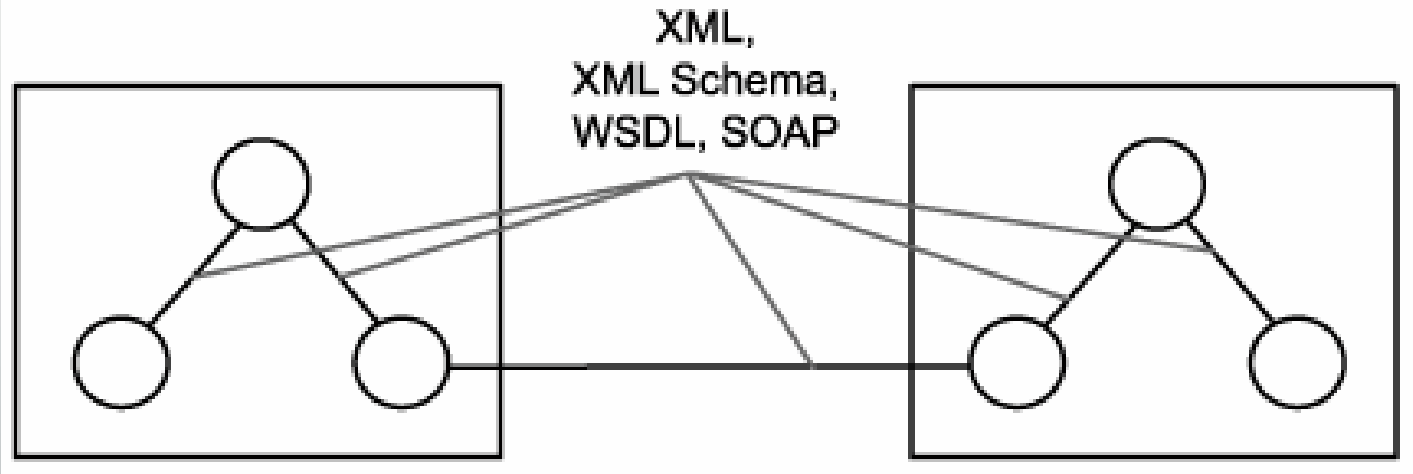
- Individual services be as independent
- self-contained as possible with respect to the control they maintain over their underlying logic.

SOA increases quality of service

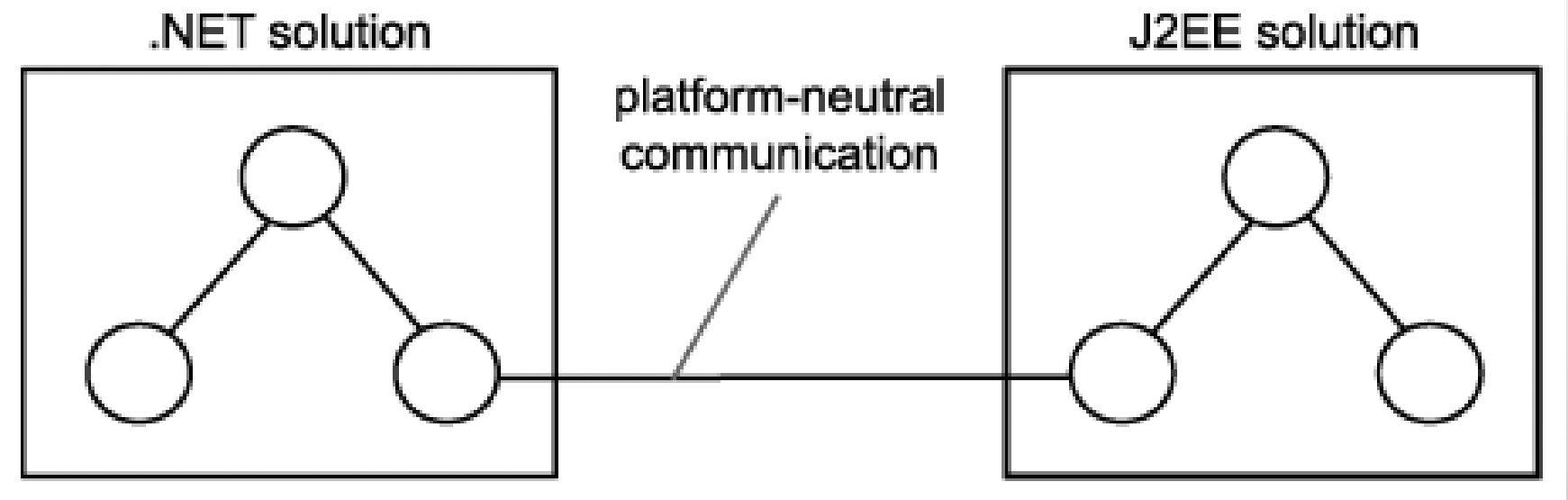
- Ability for tasks to be carried out in a secure manner, protecting the contents of a message, as well as access to individual services.
- Reliability so that message delivery or notification of failed delivery can be guaranteed.
- Overhead imposed by SOAP message and XML content processing does not inhibit the execution of a task.

SOA is based on open standards

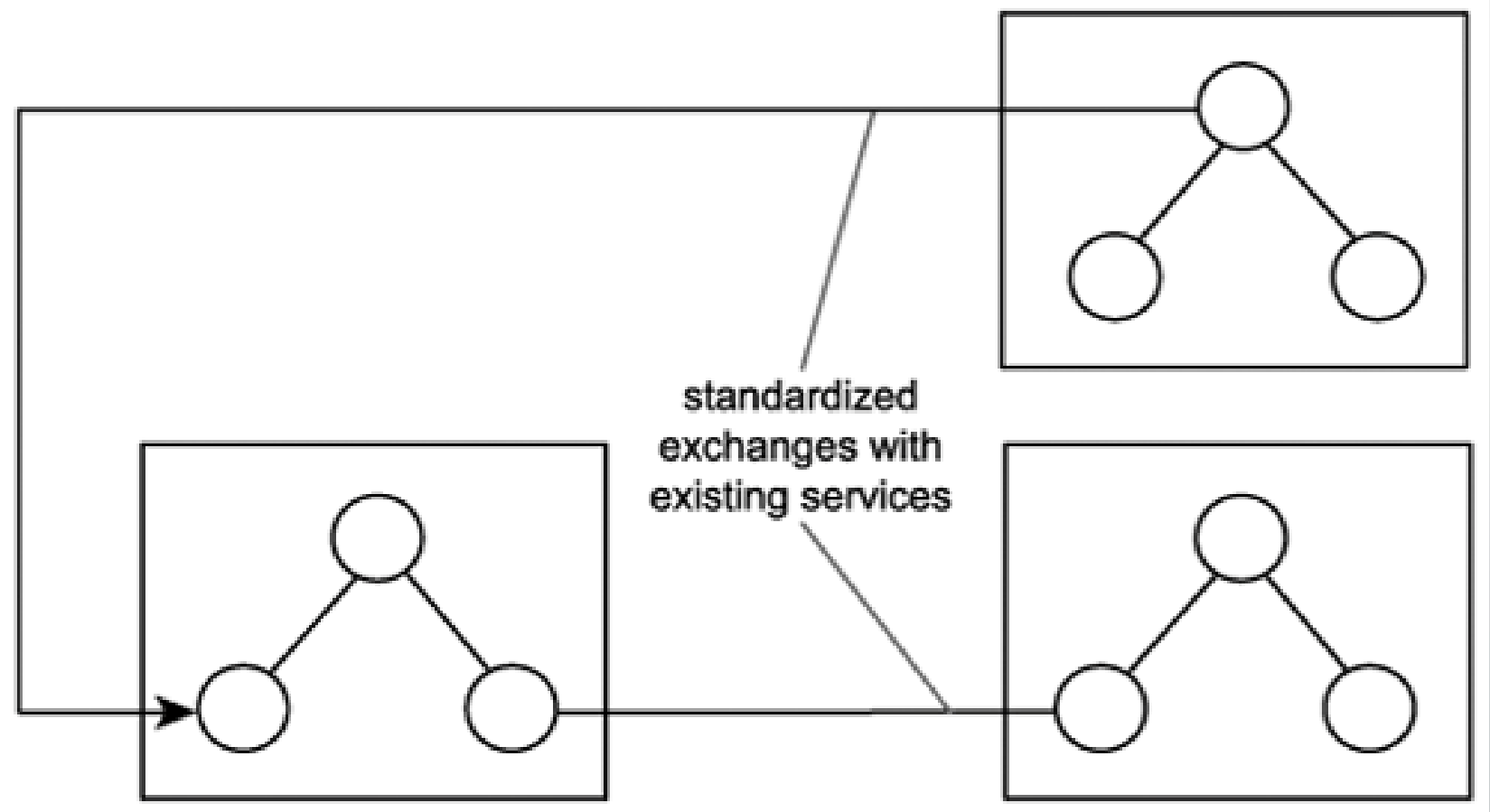
- **Standard open technologies are used within and outside of solution boundaries.**



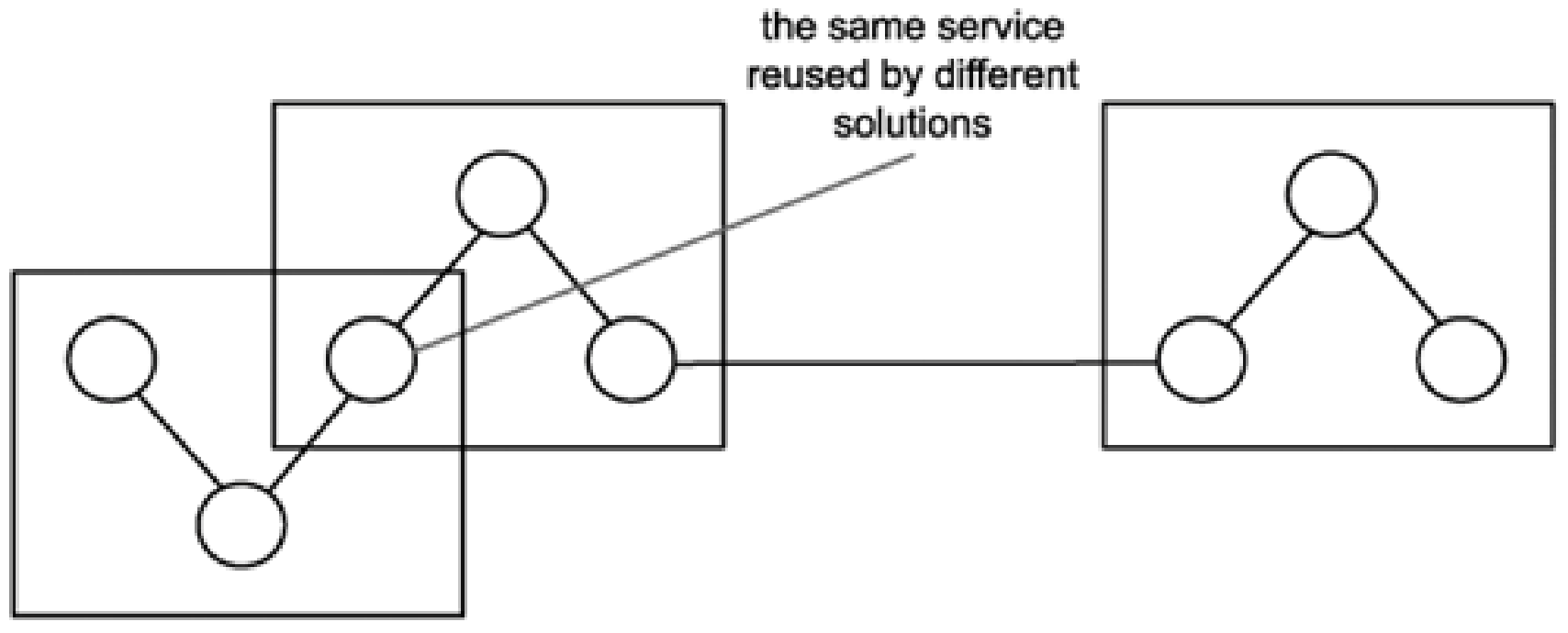
SOA supports vendor diversity



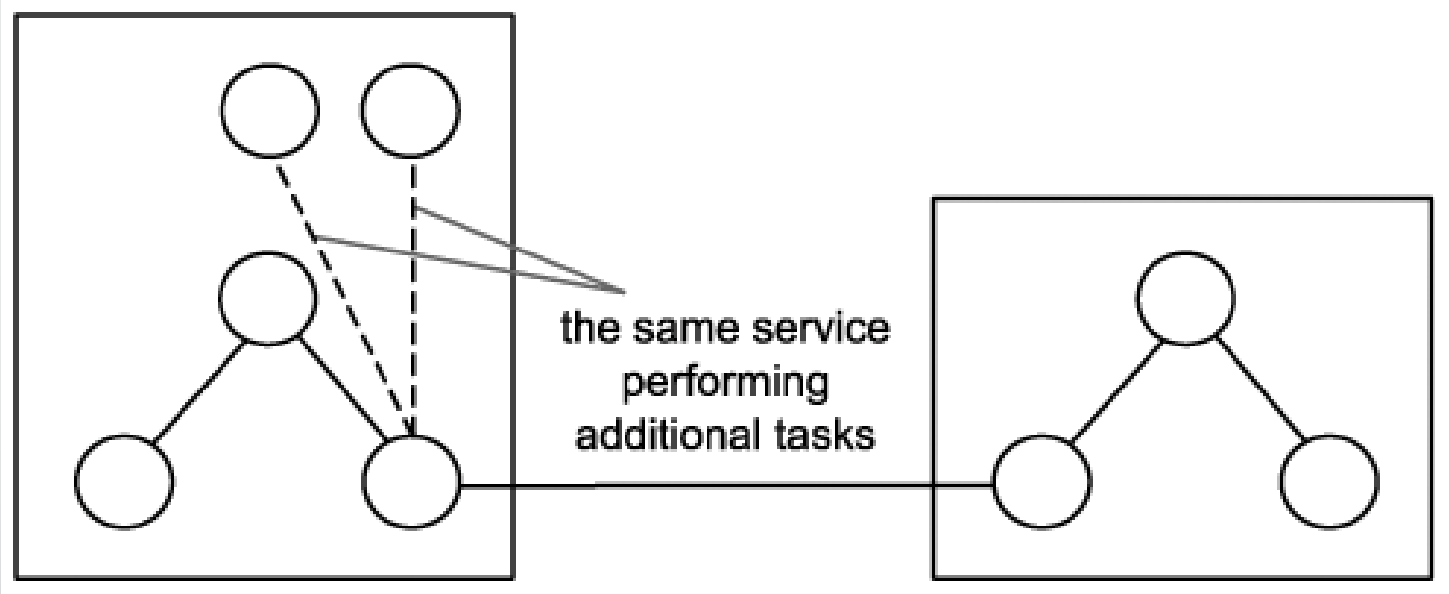
SOA encourages intrinsic interoperability



SOA encourages intrinsic reusability

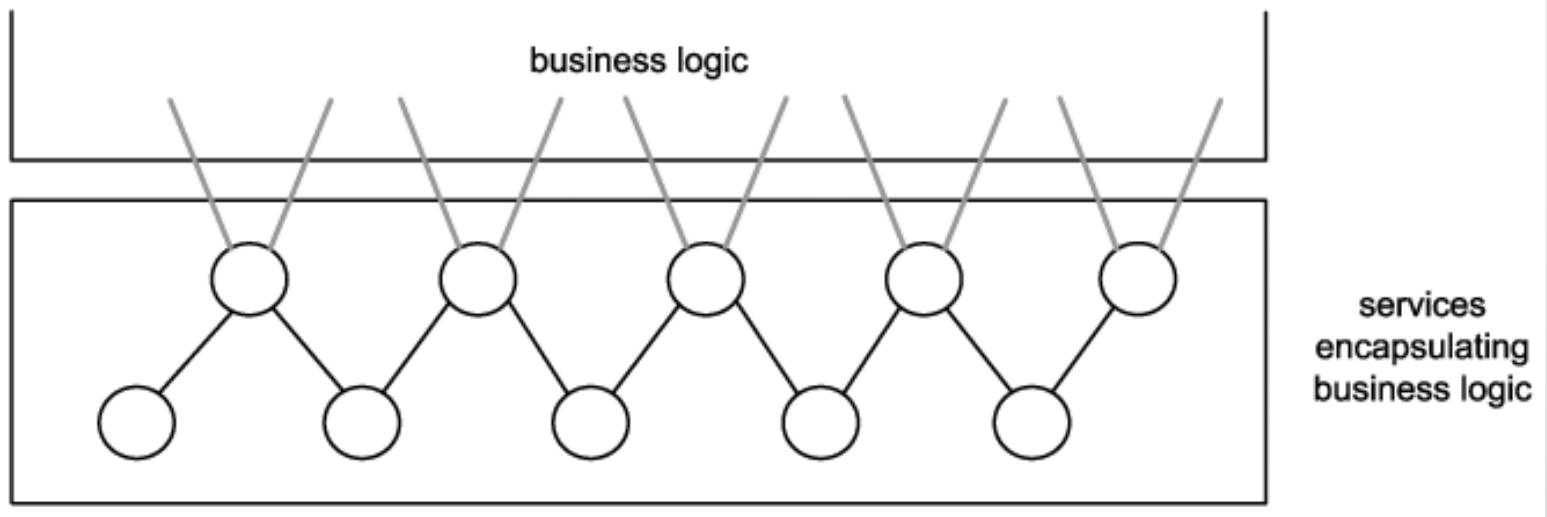


SOA emphasizes extensibility



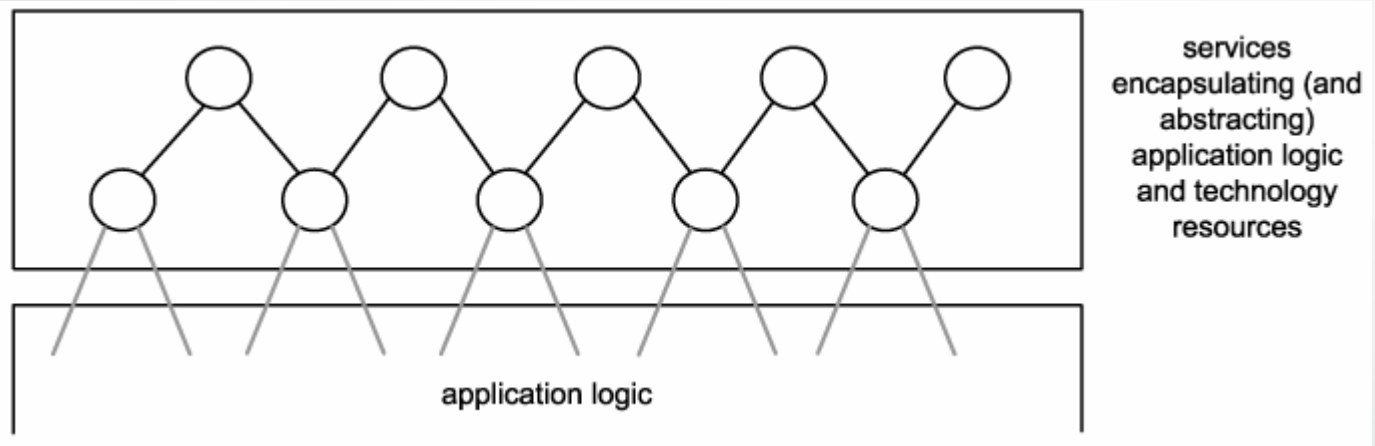
- **Extensible services can expand functionality with minimal impact.**

SOA supports a service-oriented business modeling paradigm



- **Partitioning business logic into services that can then be composed has significant implications as to how business processes can be modeled**

SOA implements layers of abstraction



- **Application logic created with proprietary technology can be abstracted through a dedicated service layer.**

SOA general definition

- SOA is a form of technology architecture that adheres to the principles of service-orientation.
- When realized through the Web services technology platform, SOA establishes the potential to support and promote these principles throughout the business process and automation domains of an enterprise.

SOA Formal definition

- Contemporary SOA represents an open, agile, extensible, federated, composable architecture comprised of autonomous, QoS-capable, vendor diverse, interoperable, discoverable, and potentially reusable services, implemented as Web services.
- SOA can establish an abstraction of business logic and technology that may introduce changes to business process modeling and technical architecture, resulting in a loose coupling between these models.
- SOA is an evolution of past platforms, preserving successful characteristics of traditional architectures, and bringing with it distinct principles that foster service-orientation in support of a service-oriented enterprise.

Common tangible benefits of SOA

- Improved integration (and intrinsic interoperability)
- Inherent reuse
- Simplified architectures and solutions
- Establishing standardized XML data representation
- Focused investment on communications infrastructure
- Organizational agility

Common pitfalls of adopting SOA

- Building service-oriented architectures like traditional distributed architectures
- Not standardizing SOA
- Not creating a transition plan
- Not starting with an XML foundation architecture
- Not understanding SOA performance requirements
- Not understanding Web services security

Summary 1.2

- We recognize contemporary SOA with a number of common characteristics that build upon and extend the original qualities and principles established by primitive SOA.
- Some of the more dangerous assumptions about SOA are that service-oriented solutions are simple by nature, easy to build, and automatically interoperable
- Many of the pitfalls relate to a limited understanding of what SOA is and what is required to fully incorporate and standardize service-orientation.
- A transition plan is the best weapon against the obstacles that tend to face organizations when migrating toward SOA.

PART II - THE EVOLUTION OF SOA

XML: brief history

- Like HTML, the Extensible Markup Language (XML) was a W3C creation derived from the popular Standard Generalized Markup Language (SGML) that has existed since the late 60s.
- This widely used meta language allowed organizations to add intelligence to document data.
- The language itself was used as the basis for a series of additional specifications.
- XML Schema Definition Language (XSD) and XSL Transformation Language (XSLT) were both authored using XML. (These specifications, in fact, have become key parts of the core XML technology set)

Web Services: a brief history (I)

- In 2000, the W3C received a submission for the Simple Object Access Protocol (SOAP) specification.
- This specification was originally designed to unify (and in some cases replace) proprietary RPC communication. The idea was for parameter data transmitted between components to be serialized into XML, transported, and then deserialized back into its native format.

Web Services: a brief history (II)

- This ultimately led to the idea of creating a pure, Web-based, distributed technology (one that could leverage the concept of a standardized communications framework to bridge the enormous disparity that existed between and within organizations).
- This concept was called Web services.
- The most important part of a Web service is its public interface.
  - It is a central piece of information that assigns the service an identity and enables its invocation

Web Services: a brief history (III)

- One of the first initiatives in support of Web services was the Web Service Description Language (WSDL).
- The W3C received the first submission of the WSDL language in 2001 and has since continued revising this specification.
- IBM and Microsoft were each working on a way to programmatically describe how to connect to a Web Service. After some discussion, protocol proposals from Microsoft and IBM merged. In late 2000, a merged specification, Web Services Description Language (WSDL), was announced.

Web Services: a brief history (IV)

- Completing the first generation of the Web services standards family was the UDDI specification.
- Originally developed by UDDI.org, it was submitted to OASIS, which continued its development in collaboration with UDDI.org.
- This specification allows for the creation of standardized service description registries both within and outside of organization boundaries.
- UDDI provides the potential for Web services to be registered in a central location, from where they can be discovered by service requestors.
- **UDDI** (Universal Description, Discovery, and Integration) is an XML-based registry for businesses worldwide to list themselves on the Internet. Its ultimate goal is to streamline online transactions by enabling companies to find one another on the Web and make their systems interoperable for e-commerce.
- Unlike WSDL and SOAP, UDDI has not yet attained industry-wide acceptance, and remains an optional extension to SOA.

7 Steps to SOA

1. Create/Expose Services
2. Register Services
3. Secure Services
4. Manage (monitor) Services
5. Mediate and Virtualize Services
6. Govern the SOA
7. Integrate Services

1. Create & Expose Services

- Three primary choices
  - Rebuild existing applications using SOA paradigm
  - Expose existing application logic as a set of services
  - A combination of rebuild and expose
- Enterprises typically use a combination of rebuild & expose
  - Solutions exist that facilitate migration of mainframe applications such as CICS to Web Services
- ***Granularity*** is a key criterion for Web Service
  - Functionality must be sufficiently coarse-grained
  - If coarse-grained, potential to be useful to different applications

2. Register Services

- Application architects & developers need to know that a service exists
- Use a registry
- UDDI compatibility important
- Search and Browse capability
- Facilitate quick and accurate discovery of services
- Some vendors have extended registries to repositories

3. Secure Services

- May have inadvertently created gaping security holes
- My have exposed sensitive information
- 5 principles of security
  1. Authentication
     - Basic HTTP authentication, SAML, X.509 signature
  2. Authorization
     - Leverage solutions such as CA SiteMinder, IBM TAM
  3. Privacy
     - XML-Encryption
     - Key & certificate management & distribution capabilities
  4. Non-Repudiation
     - Requestor & Sender cannot deny activities
  5. Auditing
     - Accurate accounting of requests & responses

4. Manage Services

- Look for potential disaster
  - Too many applications consuming a service?
  - Is the load reasonable
  - Is there a degradation in performance?
- Need to be able to monitor for
  - Basic Availability
  - Performance
  - Throughput
  - SLA agreement

5. Mediate & Virtualize Services

- As SOA matures may need to:
  - Introduce new versions
  - Increase capacity by running multiple instances
  - Provision applications to use specific instances of services
- Solution is Virtualization
  - Virtual service is a new service
    - Own WSDL, network address, transport parameters
    - Doesn't implement business logic
    - Acts as proxy to one or more physical services
    - Routes, load-balances, transforms, mediates
- XML transformation can be used to allow consumers to use an old version of service that no longer exists
  - Request & response transformed

6. Govern the SOA

- Use a governance framework
- Design Time Issues
  - What types of services can be published?
  - Who can publish them?
  - What types of schema and messages services can accept?
  - What are the rules for the services?
- Run Time Issues
  - Security
  - Reliability
  - Performance
  - Compliance with policies