# Service-Oriented Architecture

Motivation

Service-Oriented Architecture Principles

Service vs. Components

Trends and Summary

# MOTIVATION

# Evolution of Business Computing

Business demands require an enterprise focus

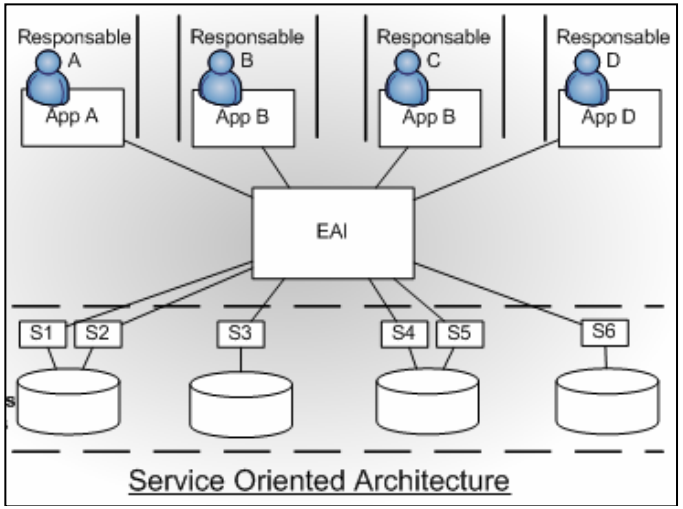
- Business units no longer act in isolation
  - Require information and services from other units
  - Enterprise processes span business units
- The "EAI" (Enterprise Application Architecture) has been bound to various technology stacks over the years, from distributed components (e.g. CORBA, DCOM, EJBs) to services-oriented emphasis (WS-*, ESBs, REST, Microservices…)

Businesses are now becoming more "open" in a data-driven world

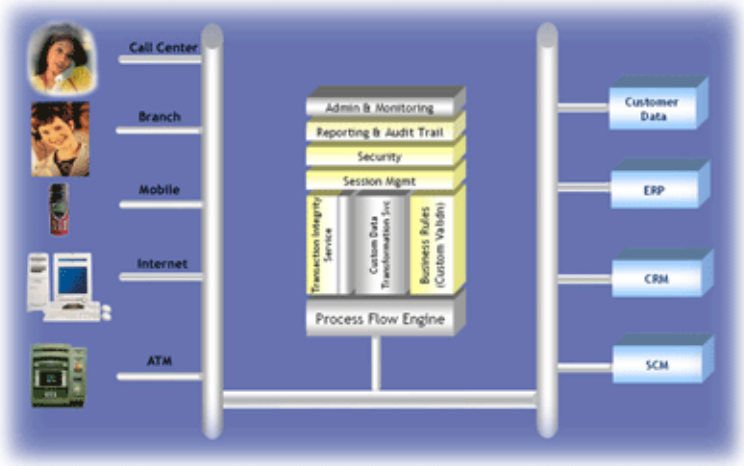
- Open APIs to open (or for rent) data and data processing services
- The nature and complexity of integration is changing
- "Complexity does not go away, it is moved around"

Computing Environments for EAI are evolving

- Old school: Triggers in a shared database
- Legacy: Distributed computing based on shared filesystems
- Past past: Distributed components (CORBA, DCOM, EJB)
- Recent Past: Web Services stack
- Current: REST, Open APIs, Microservices
- Next: Serverless! Direct deployment of behaviors



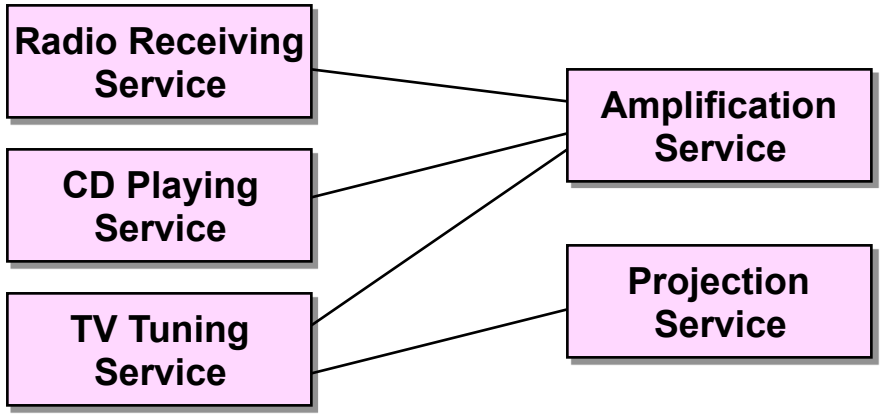
# Integration Challenges




- Heterogeneous systems
  - Platforms, technologies, interfaces
- Managing dependencies
  - EAI by definition includes many dependencies
- Patience
  - Requires incremental adoption
  - Although first increment might feel like a big bang…
- Understanding requirements and business case
  - What needs drive integration
  - What are we integrating, why, what are the dependencies?
- Solution technologies – custom, middle vendor, outsource
  - Maturity of both technologies and developers

# Managing Complexity and Integration

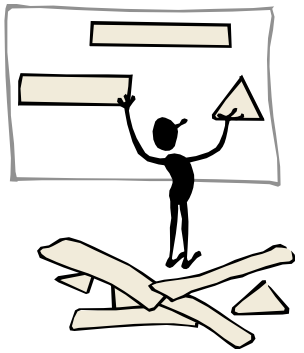
- How do you handle complexity and integration?
  - Services provided through ports with well-known communication standards
    - Provides easy integration, simple restructuring, component reuse, etc.
    - Black box implementations; communication does not imply implementation
  - Operator can easily reconfigure, add/remove services, change service state, start/stop services, etc.
    - *What has changed over time is the nature of the service deployment, from app container "middleware" to lightweight, single server "containerization" (stay tuned…)*



# SERVICE-ORIENTED ARCHITECTURE PRINCIPLES

# Service Oriented Architectures

- Architectural style achieving loose coupling among entities
  - Services are self-contained; not dependent on other services' state
    - Context-free – can be invoked without context of who is invoking
  - Communication achieved through normalized, asynchronous messages
    - *The nature of "normalized" has changed over the past decade*
- Loose coupling between service providers and consumers
  - Communication independent of implementation
- Not bound to Web services
  - However, Web Services are a common solution with a great deal of momentum
  - Web services infrastructure receiving a great deal of support from major technology companies



# Principles of SOA

## Stateless interactions

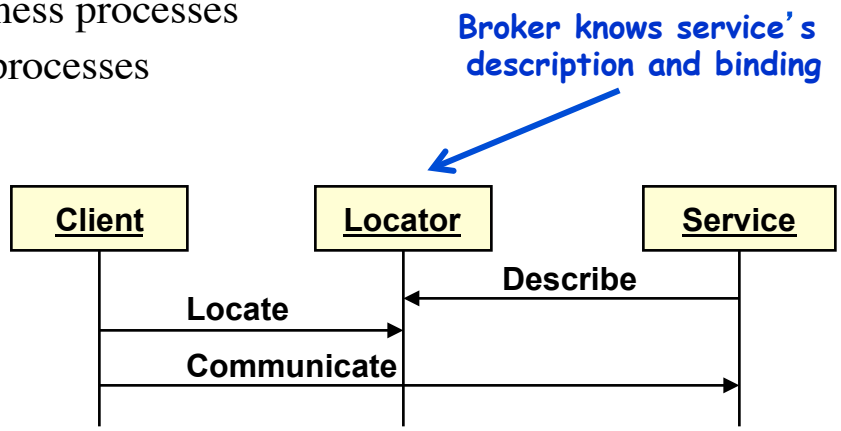
- Services are self-contained and do not store state between invocations
- State should be managed in the business process (e.g. choreography) and provided to individual services

## 3 Modern Flavors

1. Communicate through normalized message broker (ESB)
   - SOAs provide an independent protocol layer (normalized messages)
   - Adapters may be required to convert legacy applications to participate
   - Message-oriented, stateless communication; focus on the message, not the protocol
2. Use REST or similar API-driven services
   - Rely on established best practices instead of heavy standards-based technology stacks
   - Ease the burden of deployment and troubleshooting on your ops staff
   - Scale-up/down (elasticity, cloud) on demand
3. Microservices as a way to promote full lightweight decoupling
   - Assign a team, choose a language/platform, expose a behavior
   - Lightweight migrant deployment – the rise of the container

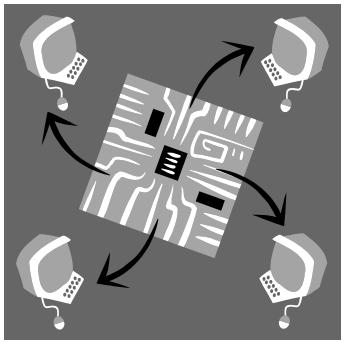
# Basic Elements of SOAs

1. Communication infrastructure (transport)
   - The wire protocol for communication – e.g., SMTP/HTTP
2. Description of service specification
   - Specifies service, messages, responses, QoS guarantees, permissions, access control, etc.
3. Discovery and location
   - Allows clients to discover existing services
   - Criteria may include services properties (e.g., attributes, QoS)
4. Composition
   - Composes services to create higher-level business processes
   - Orchestration defines business and workflow processes
5. Additional infrastructure specifications
   - Security, transactions, event notification, etc.



# 1) Communications Infrastructure

- Infrastructure built on the layered network model
- Transports message and data between client & service
  - Passes message and data and returns data result or fault
- Defines how clients denote service endpoints
  - Technology examples – remote object reference, URL, etc.
- Defines a normalized message format
  - Communication protocol (we have been doing HTTP)
  - Technology examples – JSON, SOAP, etc.
  - Message transported with any protocol
    - May use multiple transports
- Addressing, routing, security
  - May be provided by messaging protocol (WS-* standards)
  - May be provided by transport (e.g., URL, https)



# 2) Service Description

- Defines meta information describing a service's characteristics
  - Information necessary to deploy and interact with a service
- Characteristics
  - Functional characteristics defines messages the service sends & receives
  - Policy characteristics defines the service's execution context
    - Does service require security, transactions, etc.?
    - May also define QoS parameters, e.g., minimal encryption strength
- Technology examples
  - CORBA interfaces and interface repository
  - Web Service's WSDL
  - REST "excellent documentation"

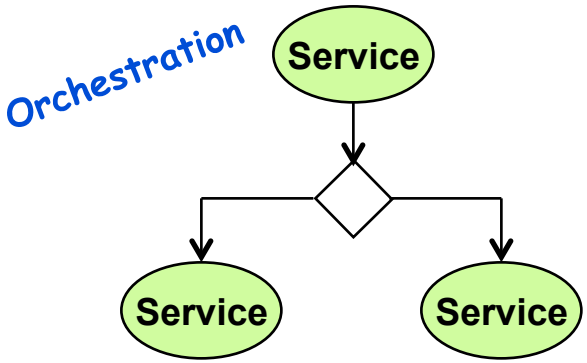
*There is still an active research community but it seems the community has somewhat gotten away from this API "utopia"*

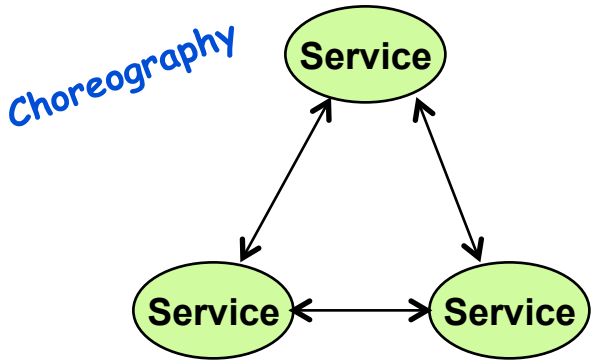
# 3) Service Discovery

- Mechanism by which clients discover & bind to services
  - Must uniquely identify service
  - May also include meta-information about the service
- Technology examples
  - Distributed object '*naming services*' are most common example
  - CORBA: DII
  - Java: JNDI
  - REST: single endpoint + navigation + semantic formats
  - Web Services include UDDI, WS-Policy, DNS-Endpoint Discovery (the white pages for UDDI)
  - Microservices: Client/Server-side Discovery Patterns, Service Registries
- Community currently has weak solutions in this space

# 4) Service Composition

- Create a service from the composition and orchestration of lower-level services
  - Orchestration defines centrally controlled process flow for services (e.g., ESB architectures)
  - Choreography coordinates distributed services (e.g., BPEL)



# 5) Other Common Services Aspects

- Leasing
  - Service usage granted for fixed period
    - Allows service implementation to expire information it may be managing
    - Allows SOA systems to self-heal when services and connections die
- Events
  - Asynchronous communication eliminates need for polling
- Security, reliable messaging, transactions
  - Provides common credentialing and state for these commonly used features

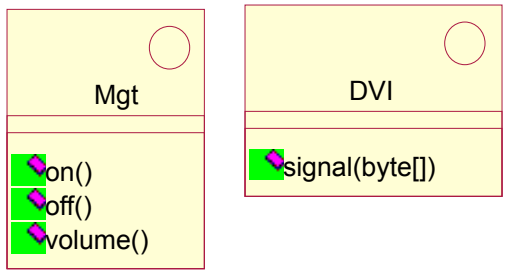
# SERVICES VS COMPONENTS, TRENDS, AND SUMMARY

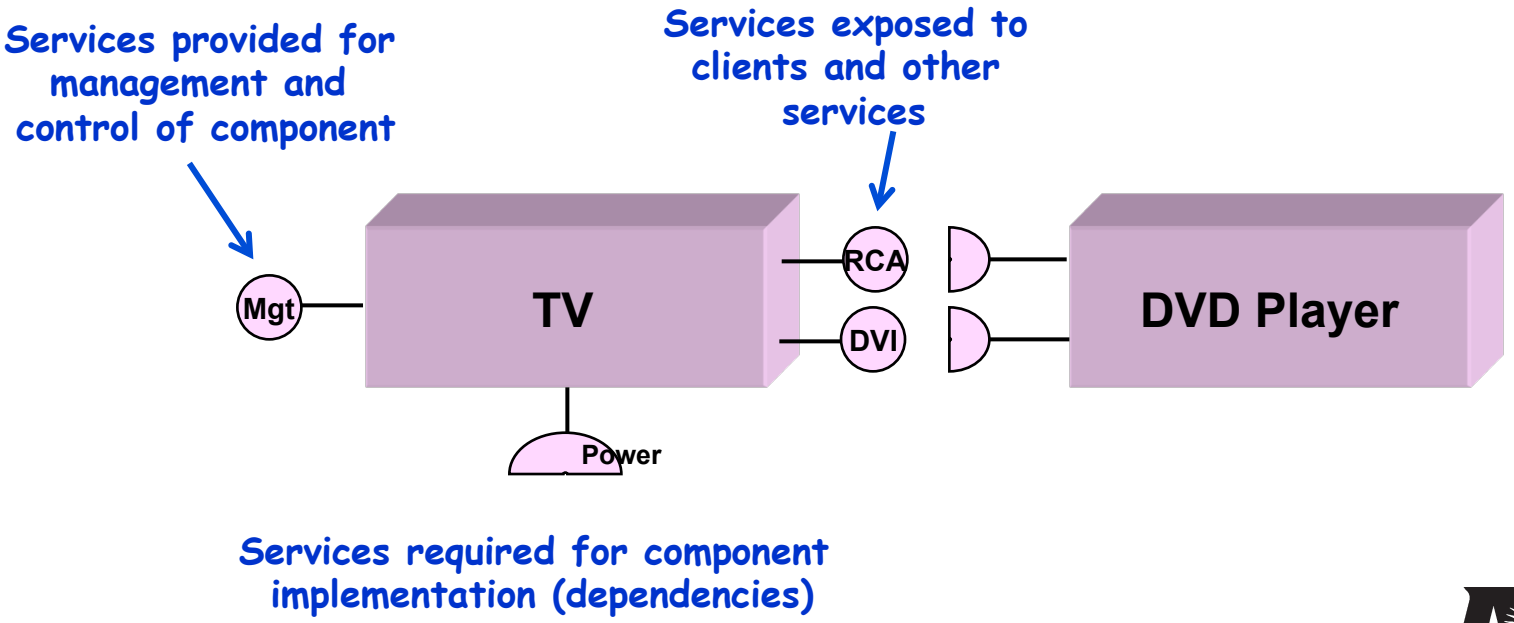
# Services and Components

- Let's back up and talk conceptually…
- What is a service?
  - Specifies course-grained, core business logic
  - Maintains *no user state* and returns result with each request (*context free*)
  - Can be reused across a diverse set of applications
  - Makes no statement *regarding implementation*, *deployment*, etc.
- What is a component?
  - An individually deployable, executable code assembly
  - Commonly *run inside a 'component container'*, which provides lifecycle services and access to resources
  - May have some form of state stored in the component

# Services and Components

- Service-Component relationship
  - Services may be realized by components
  - Components may use/depend on services



# Trends

## The Recent Past

- Application Containers
- Components behind Services
- Assembly and Composition
- Ops: Virtualization
- Values
  - Flexibility (dev & ops)
  - Scalability
  - Decomposition

*We join this show already in progress…*

## The Next Wave

- YAGNI
- Single-process, Single-server
- CCC – Commoditization, the Cloud, and Containerization
- Lean Configuration
- Values
  - Speed (DevOps)
  - Scalability
  - Decomposition – yes, but inversion of the process space

# Summary

- Services-oriented computing is an exercise in managing complexity
- Understand how the 5 SOA Architecture Principles apply in each context
- Solutions driven by economics as much as by technology