Introduction
Service-Oriented Computing (SOC)
Service-Oriented Architectures (SOA)
Web Services
Summary

What is a Distributed System?
- "If your system fails because of a computer you never heard of, you have a distributed system"
- "A distributed system is a software system in which components located on networked computers communicate and coordinate their actions by passing messages"
- Distributed systems > data communication systems
  - higher level
  - communicating subsystems
  - more advanced functionality
  - more advanced challenges

Distributed System Challenges
- Complexity
  - system abstraction level, scale, granularity
  - resources and system heterogeneity
  - distributed failure types and sources
  - resilience and performance requirements
  - security and trust models
- Illustrating distributed system complexity
  - single-threaded vs multi-threaded
  - single process vs multiple processes
  - single core vs multicore vs manycore
  - single host vs multiple hosts
  - local hosts vs distributed hosts
  - ...add heterogeneity, latency, failures, security, etc...

Message Passing vs Remote Invocation
- Remote invocation exposes an interface over a network
  - invocation semantics left to the invoker
- Message passing uses a (distributed) object model to interpret messages
  - messages parsed at destination (or at intermediaries)
  - invocation semantics determined at object side
- Many different types of message passing, e.g.,
  - asynchronous vs synchronous message passing
  - distributed object models
  - message-oriented middlewares
  - enterprise service buses
  - etc.

Service-Oriented Computing (SOC)
- A computing paradigm revolving around the concept of software as a service
- Assumes that entire systems of software are built and deployed as network-accessible services
- Focus is placed on the utility of software components, rather than on mechanisms of software
- The value of an application is measured in terms of ability to integrate into the environment rather than the application's functional capabilities
The success of Web Services promoted the use of service-oriented architectures (SOAs). This architectural style is used in Service-Oriented Computing (SOC) to build complete systems using services as building blocks. SOAs build complete systems using services as building blocks. Components are identified and assigned responsibilities that clients interact with through "contracted" interfaces.

- Technology neutral
  - use widely available, standardized technology
- Loosely coupled
  - minimize formal knowledge required to use a service
- Location transparent
  - be publicly discoverable
  - provide QoS regardless of location of service or clients

New architectures aimed to leverage standard-based functional services was realized on a number of distributed object models. The success of Web Services promoted the use of a number of other XML-based technologies.

A widespread need for automated business integration
- Multiple tools and techniques merged over time
- A vision of software as a service emerged
- Desirable software properties were identified
  - loose coupling
  - standardization
  - protocol independence
  - distribution of computing applications (scalability)

New architectures aimed to leverage standard-based functional services was realized on a number of distributed object models.

The SOA models service actors and relations & interactions between service actors. Well-designed SOAs provide flexibility in design and deployment, adaptability and integrability in architecture, scalability in implementation and deployment. SOAs build complete systems using services as distributed and modularized components without regard to service implementation. (Composite) applications are built by discovering, invoking, and composing services rather than developing new modules.

A high-enough level of abstraction to view the system as a whole while still provide enough information for analysis and decision making
- Structure that supports required system functionality and behavior
- Conformance to desired system qualities and non-functional requirements (performance, security, interoperability, reliability, flexibility, extensibility, etc)
- Tradeoffs between system qualities must be identified and prioritized
- A view where all implementational details are hidden

Well-designed SOAs provide flexibility in design and deployment, adaptability and integrability in architecture, scalability in implementation and deployment. SOAs build complete systems using services as distributed and modularized components without regard to service implementation. (Composite) applications are built by discovering, invoking, and composing services rather than developing new modules.

Service-Oriented Architectures (SOA)

- An architectural style used in Service-Oriented Computing
- SOA models service actors and relations & interactions between service actors
- Well-designed SOAs provide flexibility in design and deployment, adaptability and integrability in architecture, scalability in implementation and deployment
- SOAs build complete systems using services as distributed and modularized components without regard to service implementation
- (Composite) applications are built by discovering, invoking, and composing services rather than developing new modules
**Service-Oriented Architectures**

- A SOA is based on the combination of and interaction between services:
  - associated with messages
  - governed by policies
- A SOA is designed to:
  - allow developers to overcome complex implementation challenges
  - leverage the potential of distributed applications
  - eliminate component integration barriers
  - produce seamless applications

**SOA Service Characteristics**

- All functions in a SOA are exposed as services
- All services are independent of each other (service operation perceived as opaque)
- Service interfaces are invokable (regardless of location, platform, protocol, etc)
- Services operate with an always-on semantic (no construction or destruction semantics)

**SOA Actors**

- Service providers
  - provide service implementations
  - supply service descriptions
  - provide service support
- Service registries
  - provide (publicly accessible) service information sources
  - house service meta information (service description, service location, service cost metrics, etc)
- Service clients
  - use service functionality
  - may be individual end-users, organizations, or services
- Service aggregators
  - aggregate services into new services
  - act as brokers and value-add providers

**SOA Services**

- A SOA service is an exposed piece of functionality that:
  - is self-contained (maintains its own state)
  - is platform-independent (interface)
  - can be dynamically located and invoked
- The primary value of a SOA component lies in that it supports reuse of component functionality either as stand-alone or as part of a composed and orchestrated application

**Service Realizations**

- Service containers provide deployment and run-time support environments that make services highly distributed
- Containers provide service management and monitoring facilities as well as lifecycle management contexts
- Containers also allow services to be viewed as logic components in implementation and as abstract services in design
- Techniques such as thread pools and instance reuse allow containers to be highly efficient service hosting environments

**SOA Operations**

- Publication of service description
  - construction of service descriptions
  - publication of service descriptions in service registry
- Service selection
  - location of service descriptions
  - selection of a suitable subset of available services
- Service invocation
  - service invocation semantics establishment (WSDL)
  - service invocation
Services as a Virtualization Technology

- Services can be used without having to worry about
  - service discovery
  - service selection
  - data exchange formats
  - service implementation details
- Services can be aggregated
  - similar services can be hidden behind a common (adapter) interface
  - services can act as brokers to other services

Enterprise Service Bus (ESB)

- Open standards-based message delivery backbone
- Designed to facilitate implementation, deployment, and management of SOA-based applications
- Provides a set of infrastructure capabilities implemented by middleware technology
- Supports service invocations, message, and event-based interactions with appropriate service levels
- Functions as both transport and transformation facilitator to allow distribution of services with increase service interaction granularity
- Conceptually equivalent to a high-level, message-based middleware

ESB Interoperability

Service Usage Patterns

- Service aggregation
  - building applications and logical services by utilizing capabilities offered by services
  - includes creating adapter interfaces to groups of services and service capability brokering
- Service composition
  - constructing services by combining existing services
- Service orchestration
  - creating applications by coordinated use of services
  - focuses on message-level interactions and control flows for individual services
- Service choreography
  - defines workflows for interactions between services
  - describes system-wide collaborations and interactions in terms of message exchanges

Service Registration & Discovery

- A methodology to decouple communicating systems
- Service Registration
- Service Discovery
  - static - design time
  - dynamic - run time
- Service Discovery
  - Locating service providers
  - Retrieving service descriptions
- Service Selection
  - determine what (subset of) services of the discovery result to invoke

Publish, Find, Bind
**Summary**

**Web Services**

**Architectures (SOA)**

**Service-Oriented Computing (SOC)**

**Introduction**

Today

- **Why Discovery and Notifications?**
  - Polling consumes resources
    - network bandwidth - massage bandwidth, congestion packet loss
    - memory - spatial storage space, physical memory exhaustion
    - CPU - message processing load, context switches
    - threads - thread pool exhaustion, synchronization issues
    - sockets - file descriptor exhaustion
  - Techniques to mitigate the need for and impact of polling
    - topological - replace polling with notifications schemes
    - spatial - aggregate interfaces & messages
    - temporal - reduce and overlap poll intervals
  - Subscription-based notifications improve scalability

- **Layered SOAs**
  - SOA projects are often large, involving multiple, disparate organizations and heterogeneous system views
  - SOA deployments will often involve physical and virtual resources owned by multiple parties, and incorporation of various legacy systems
  - To modularize designs and address these issues, SOA systems are often segmented into layers, where working groups collaborate to create limited functionality sets
  - Layered designs work by principle of abstraction, hiding complex functionality and providing customized interfaces in higher layers
  - Applications are typically integrated at high (interface) level rather than a low (implementation) level

- **Quality of Service (QoS)**
  - Guarantees for service behavior
    - typically expressed in terms of performance
    - Qualitative QoS - perceived value
    - service reliability
    - Quantitative QoS - quantifiable value
      - (domain-specific metrics)
        - transactions / second, total usage time (service)
        - bandwidth, latency, jitter, packet loss rate (network)
        - % CPU access, # FLOPS, # RAM (computational)

- **Service-Level Agreements (SLA)**
  - An SLA is a formal agreement (contract) between service providers and clients, that formalizes the details of a service (e.g., content, price, delivery, quality, etc, usually in measurable terms) in a way that meets mutual understandings and expectations of all parties involved
  - SLAs can be static or dynamic, where the latter adapts to current service provisioning
**Web Service Characteristics**

- **Platform-independent**
  - Interfaces and data expressed in XML
- **Loosely coupled**
  - Service implementations hidden behind interfaces
  - Services explicitly self-describing and discoverable
- **Self-contained**
  - Service dependencies abstracted by other services
- **Self-describing**
  - Interface description provides invocation data
  - Services and service registries provide semantic metadata
- **Use wide-spread and reliable technology base**
  - XML, SOAP, WSDL, HTTP
- **Designed to provide a universal IPC mechanism**

**Benefits Of Loose Coupling**

- **Flexibility:** services can be (re)located on any server
- **Scalability:** services can be added / removed on demand (load balancing)
- **Replaceability:** service implementations can be replaced (without user disruptions)
- **Fault tolerance:** upon failures, clients can query registries for alternative services offering the same functionality

**Loose Coupling**

- **Coupling is a measure of the degree of dependencies**
  - **Tightly coupled systems**
    - require knowledge of how peers behave
    - requires agreement and shared contexts for interoperation
    - ties interfaces into other system interfaces
    - makes maintenance and development complex
    - small changes likely to affect other modules
  - **Loosely coupled systems**
    - minimizes knowledge required for interoperation
    - focuses on autonomy of software modules
    - provides agility and ability to survive evolutionary changes
    - tend to use asynchronous communication models
    - tend to have coarse-grained communication patterns

**Tight vs Loose Coupling**

<table>
<thead>
<tr>
<th>Interaction pattern</th>
<th>Tight coupling</th>
<th>Loose coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message style</td>
<td>synchronous</td>
<td>asynchronous</td>
</tr>
<tr>
<td>Message path</td>
<td>RPC style</td>
<td>document style</td>
</tr>
<tr>
<td>Underlying platform</td>
<td>hard coded</td>
<td>routed</td>
</tr>
<tr>
<td>Binding protocol</td>
<td>homogeneous</td>
<td>heterogeneous</td>
</tr>
<tr>
<td>Objective</td>
<td>static reuse</td>
<td>dynamic (late binding)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>flexibility, applicability</td>
</tr>
</tbody>
</table>
**Web Services vs Object Models**

<table>
<thead>
<tr>
<th>Component</th>
<th>Web Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity</td>
<td>fine-grained synchronous</td>
</tr>
<tr>
<td>Communication</td>
<td>tight, APIs homogeneous</td>
</tr>
<tr>
<td>Endpoint coupling</td>
<td>object level</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>explicit naming</td>
</tr>
<tr>
<td>Interface</td>
<td>static capability listing</td>
</tr>
<tr>
<td>Invocation</td>
<td>static / dynamic</td>
</tr>
<tr>
<td>Brokering</td>
<td></td>
</tr>
</tbody>
</table>

**Service Oriented Architectures (SOA)**

- A style of building distributed systems where functionality is provided by modular services
- Focuses on loose coupling between interacting services (i.e., minimizing formal knowledge between components)
- Services are virtualized as much as possible (i.e., focus is placed on interfaces, not implementations)
- Commonly built on (some kind of) Web Services (today)

**SOA Characteristics**

- Logical view - No implementation details are revealed
- Coarse-grained - few operations, large messages
- Platform- and language-neutral
- Wide-spread technology base (XML, HTTP, TCP/IP)

**SOA Service Characteristics**

- Message-oriented - communicate via messages
  - abstract - interface defined in terms of messages
  - encapsulated - implementation details hidden
  - technology independent (platform, OS, API etc)
- Self-describing: provides machine-readable metadata (advertisers capabilities, service interface, protocols etc)
- Discoverable: dynamic "on-demand" service discovery (includes service location, service interface, protocols etc)
- Modular: solves a single, well-defined task
  - self-contained or dependent on other services / resources
- Interoperable: standardized service access
  - standardized protocols and data formats

**Loose Coupling**

- Components minimize built-in knowledge of each other (focus placed on interfaces, not implementations)
- Services are dynamically discovered when needed (includes interfaces, supported protocols, location etc)
- Ideal: zero coupling ("frictionless") (services used without providing any information)

**Next**

- Web Services