Today

Service-Oriented Architectures (SOA)

Properties of Software Architecture

→ 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

Service-Oriented Architectures (SOA)

Software Architecture

→ Software architecture is the high-level structure of a software system - including distributed and service-oriented systems - that is commonly specified in terms of functional components and interactions / interconnections among those components

→ Components are identified and assigned responsibilities that clients interact with through "contracted" interfaces

Service-Oriented Architectures (SOA)

SOA vs MSA

Service-Oriented Architectures

Services and Coupling

Advanced Distributed Systems

2016-11-10

P-O Östberg

Department of Computing Science, Umeå University
**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 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.

**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)

**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 Actors**

- Service providers
  - provide service implementations
  - supply service descriptions
  - provide service support
- Service registries
  - provide (publicly accessible) service information sources
  - houses 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 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
Today
Service-Oriented Architectures (SOA)
Services and Coupling
SOA vs MSA

Service-Oriented Architectures
Conceptually equivalent to a high-level, message-based middleware

ESB Interoperability
➤ 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

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
➤ Publish
   ➤ service description
➤ Find
   ➤ service registry
➤ Bind
   ➤ service client
SOA vs MSA

Service Registration

- Universal Description, Discovery, and Integration (UDDI)
- The Globus Monitoring and Discovery Services (MDS)
- Extensions to the Microsoft Information Index Server (IIS)
- (D)COM service registries
- CORBA naming services
- Java JINI lookup services
- Customized solutions
  - databases
  - configurations
  - web pages
  - web services

Why Discovery and Notifications?

- Polling consumes resources
  - network bandwidth - message 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

Layered SOAs

Quality of Service (QoS)

- Guarantees for service behavior
  - typically expressed in terms of performance
- Qualitative QoS - perceived value
  - user satisfaction
  - 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
1. **Loose Coupling**
   - **Platform-independent**
     - Service implementations realized using any platform
     - 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
   - **Use wide-spread and reliable technology base**
     - XML, SOAP, WSDL, HTTP
   - Designed to provide a universal IPC mechanism

2. **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

3. **Web Service Characteristics**
   - **Tight vs Loose Coupling**
     - **Tightly coupled systems**
       - require knowledge of how peers behave
       - requires agreement and shared contexts for interoperation
       - does not hide interfaces from 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 (loose coupling in time)
       - tend to have coarse-grained communication patterns (document style interaction)

4. **Services vs Web Services**
   - **Service**: a software component accessed over a network that provides functionality to a service requester
   - **Web Service**: a service which publishes a service interface (interface description or API) and uses a message-driven transport protocol (usually via SOAP or REST over HTTP)
   - **SCAP**: Web Services are built using a host of XML-based technologies
     - XML (data exchanged)
     - XML Schema (validation of data exchanged)
     - SOAP (XML-serialized transfer protocol)
     - WSDL (Web Service interface description, XML Schema)
   - Uses a deployment descriptor to configure service
     - (XML-based configuration file for the service container)

5. **Loose Coupling**
   - **Coupling** is a measure of the degree of dependencies in and between systems
   - **Tightly coupled systems**
     - require knowledge of how peers behave
     - requires agreement and shared contexts for interoperation
     - serves 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 (loose coupling in time)
     - tend to have coarse-grained communication patterns (document style interaction)

6. **SLA Examples**
   - **Virtual hosting (web, email, etc)**
     - SLO: Uptime guarantee: 99.7%
     - Compensation model defines target deviation
   - **Dedicated Internet connection**
     - 25% payback for 4/month outage, etc
     - exceptions - force majeure
   - **Construction (houses, bridges, ships, etc)**
     - SLO: completion time & quality / performance of product
     - penalties for missed deadlines / substandard performance
   - **Contract farming**
     - risk reduction
     - farmer sells crops before cultivating them
     - ensures income (farmer) and quality (buyer)

7. **Flexible and Dynamic Services**
   - **Flexible**
     - services can be (re)located on any server
   - **Dynamic**
     - services can be added / removed on demand (load balancing)
   - **Replaceable**
     - service implementations can be replaced (without user disruptions)
   - **Fault-tolerant**
     - upon failures, clients can query registries for alternative services offering the same functionality

---

**Service-Oriented Architectures (SOA) – today**

**Service-Oriented Architectures Department of Computing Science, Umeå University**
### Web Services vs Object Models

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Communication</th>
<th>Endpoint coupling</th>
<th>Infrastructure</th>
<th>Invocation</th>
<th>Brokering</th>
</tr>
</thead>
<tbody>
<tr>
<td>fine-grained</td>
<td>synchronous</td>
<td>tight, APIs</td>
<td>homogeneous</td>
<td>object level</td>
<td>explicit naming</td>
</tr>
<tr>
<td>coarse-grained</td>
<td>asynchronous</td>
<td>loose, interfaces</td>
<td>heterogeneous</td>
<td>wire format level</td>
<td>capability listing</td>
</tr>
<tr>
<td>static</td>
<td>static / dynamic</td>
<td>static</td>
<td>static</td>
<td>static / dynamic</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)

### Micro-Service Architectures (MSA)

- A style of building distributed systems based on stateless, miniaturized services with brief interactions
- Focuses on orchestration and elasticity of applications
- Services are virtualized and minimized
- Commonly built on custom binary services
- Targets continuous deployment and DevOps methodologies
- SOAs are not MSAs, but (some) MSAs are types of SOAs

### SOA Characteristics

- Focus on utility and integration (reuse) of components
- Logical view (service implementation details not revealed)
- Coarse-grained
  - few operations
  - large messages
- Platform (and language) neutral
- Services reused between applications
- Wide-spread technology base (XML, HTTP, TCP/IP)

### MSA Characteristics

- Focus on scaling and distribution of systems
- Logical view (service implementation details not revealed)
- Relatively fine-grained
  - small messages
  - quick interactions
- Optimized for (minimal) write contention
  - updates contained to service subsets
  - stateless services
- Application-specific services
- Custom technology base

### SOA Service Characteristics

- Message-oriented - communicate via messages
  - abstract - interface defined in terms of messages
  - encapsulated - implementation details hidden
- Self-describing: provides machine-readable metadata (advertises capabilities, service interface, protocols etc)
- Discoverable: dynamic "on-demand" service discovery (includes service location, service interface, protocols etc)
- Reusable: shared between applications
- Modular: solves a single, well-defined task
  - self-contained or dependent on other services / resources
- Interoperable: standardized service access
  - standardized protocols and data formats
**Micro-Service Characteristics**

- Message-oriented - communicate via messages
  - constrained message size (single packet)
  - encapsulated - implementation details hidden
  - custom technology
- Stateless (as far as possible)
  - allows orchestration and remediation
  - minimizes write contention
- Application-specific
- Independently deployable
- Often controlled and load balanced in service groups
- Modular: solves a single, well-defined task

**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)