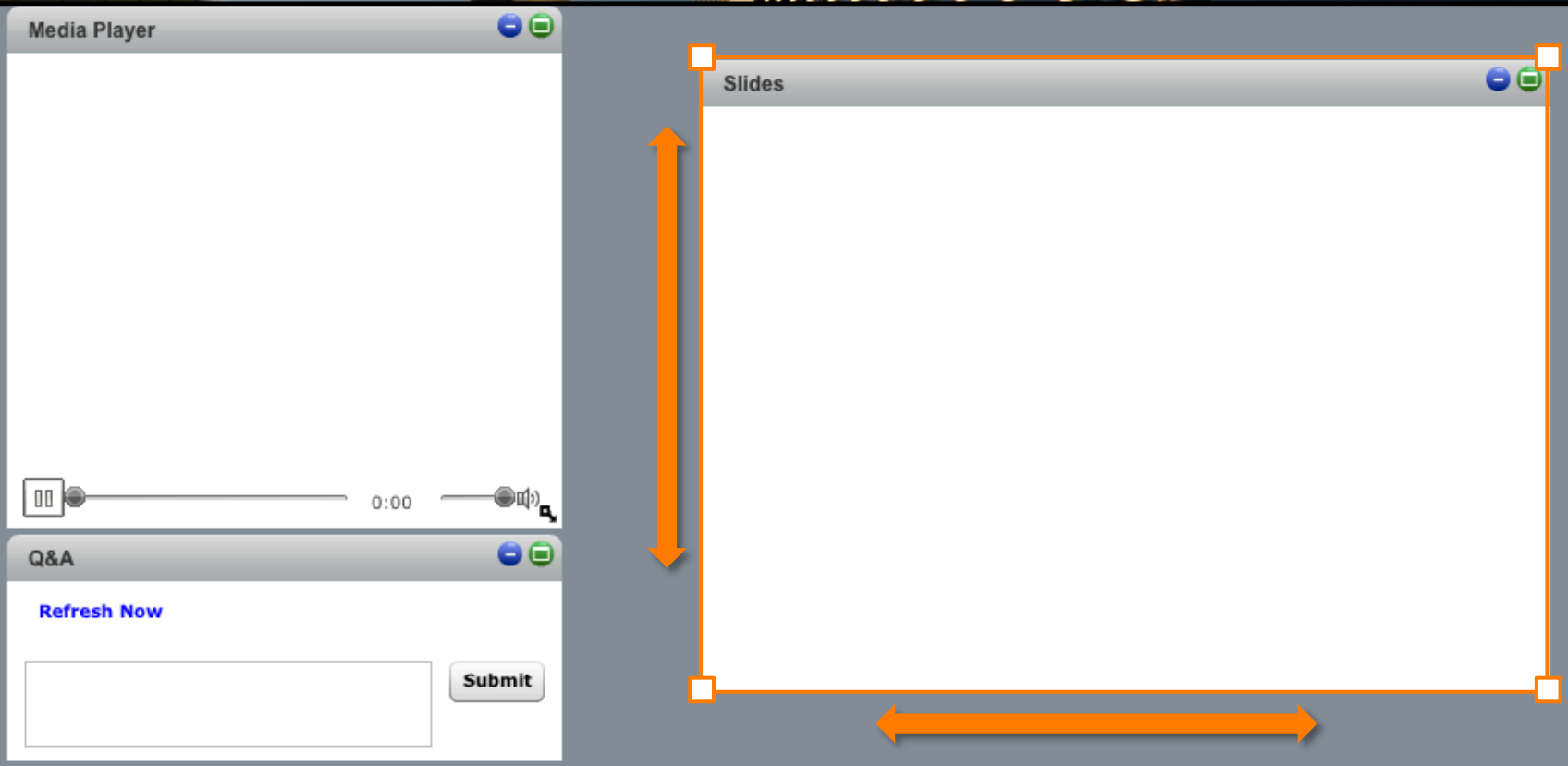


The layout of your screen is completely customizable by you

Today's Speaker

Grace Lewis
Senior member of Technical staff
Software Engineering Institute

Grace Lewis has over 20 years of professional software development experience, mainly in industry. Her main areas of expertise include service-oriented architecture (SOA), cloud computing and mobile applications.

Before joining the SEI, Lewis was Chief of Systems Development for Icesi University, where she served as project manager and technical lead for the university-wide administrative systems. Other work experience includes Design and Development Engineer for the Electronics Division of Carvajal S.A. where she developed software for communication between PCs and electronic devices; developed embedded software on the microcontroller that was used on the devices; and provided technical assistance to sales personnel during on-site visits to potential and actual clients.

Architecture and Design of Service-Oriented Systems: Goals

Present and discuss

- Basic concepts related to software architecture design
- Impact of service orientation on system qualities
- SOA infrastructure design considerations
- Decomposition of an Enterprise Service Bus (ESB) into patterns and tactics as an example of SOA infrastructure
- Principles of service design

Provide a starting point for using quality attribute requirements to design infrastructure and services for service-oriented systems

Tutorial Agenda

Part 1

- Service-Oriented Architecture: Review of Terminology
- Architecture-Centric Design Fundamentals
- Impact of the SOA Architectural Pattern on System Quality

Part 2

- SOA Infrastructure Design Considerations
- Service Design Considerations

Agenda

Service-Oriented Architecture: Review of Terminology 

Architecture-Centric Design Fundamentals

Impact of the SOA Architectural Pattern on System Quality

What is SOA?

Service-oriented architecture is a way of designing, developing, deploying and managing systems, in which

- Services provide reusable business functionality via well-defined interfaces.
- Service consumers are built using functionality from available services.
- There is a clear separation between service interface and service implementation.
  - Service interface is just as important as service implementation.
- An SOA infrastructure enables discovery, composition, and invocation of services.
- Protocols are predominantly, but not exclusively, message-based document exchanges.

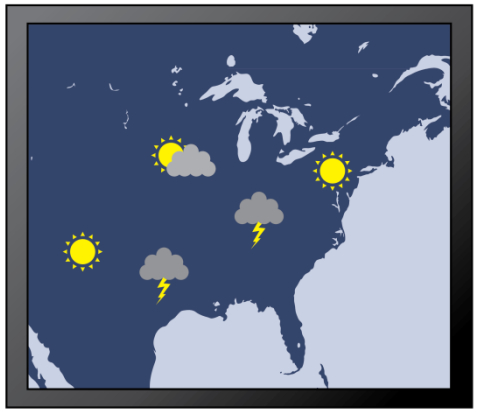
Services

Services are reusable components that represent business/operational tasks.

- Customer lookup
- Credit card validation
- Weather
- Hotel reservation

Services can be

- Globally distributed across organizations
- Reconfigured into new business processes



Service interface definitions are well-defined artifacts available in some form of service registry.

SOA Infrastructure

Set of technologies that bind service consumers to services

- Products, standards and protocols that support communication—typically message-based document exchanges
  - Web Services (WS*: HTTP, SOAP, WSDL; REST)
  - Message-oriented middleware (i.e. IBM Websphere MQ)
  - Publish/subscribe (i.e. Java Messaging Service — JMS)
  - CORBA …
- Infrastructure services available to service providers and/or service consumers to perform common tasks or satisfy QoS requirements of the environment
  - Security, discovery, data transformation, …

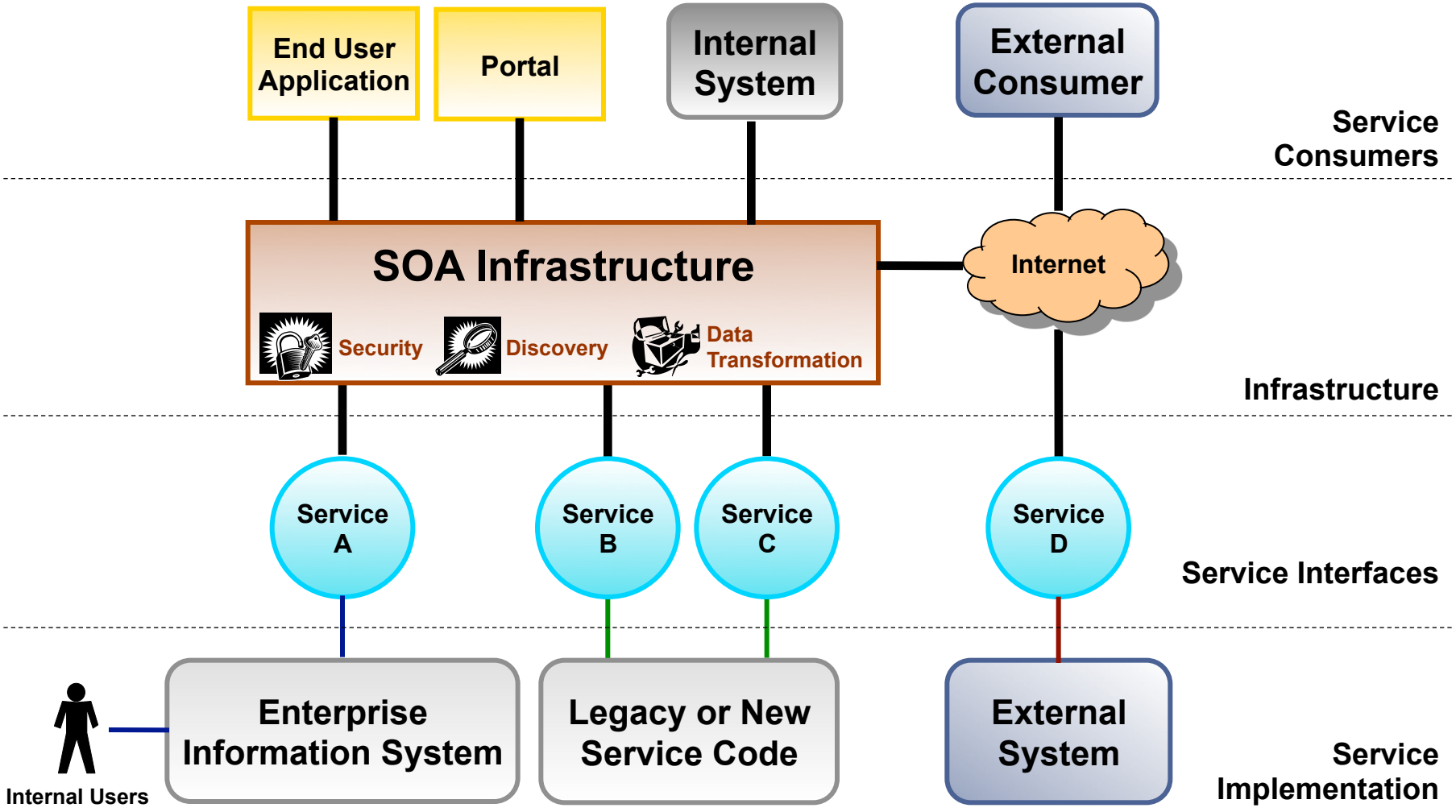
Service Consumers

Clients for the functionality provided by the services

- End-user applications
- Internal systems
- External systems
- Composite services

Consumers programmatically bind to services.

Components of a Service-Oriented System



Benefits Associated with SOA Adoption

Cost-Efficiency

- Services provide functionality that can be reused many times by many consumers
- Services become a single point of maintenance and management for common functionality

Agility

- Via service discovery mechanisms, developers can find and take advantage of existing services to reduce development times

Legacy Leverage

- Separation of service interface from service implementation provides true platform independence

Adaptability

- Separation of service interface from service implementation allows for incremental deployment of services and incremental modernization

Agenda

Service-Oriented Architecture: Review of Terminology

Architecture-Centric Design Fundamentals ⇦

Impact of the SOA Architectural Pattern on System Quality

Software Architecture

The current literature on software architecture offers many definitions

The definition we use in this course is

The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both*

Why create a software architecture?

- Because a system's quality attributes can be predicted by studying its architecture

* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Third Edition. Boston, MA: Addison-Wesley, 2013.

Architecture-Centric Design

An architecture-centric approach to service-oriented systems design builds on a set of fundamental concepts

- **Business Goals**
  - Every system is built to satisfy business goals
  - Business goals determine a system's quality attribute requirements
- **Software Quality**
  - Degree to which software possesses a desired combination of attributes*
  - Examples of attributes are reliability, interoperability, performance
- **Quality Attributes**
  - Quality attribute requirements exert the strongest influence on architectural design
  - Quality attribute requirements can be expressed in terms of scenarios
- **Architectural Tactics**
  - Architectural tactics are an enumeration of techniques that architects use to achieve particular quality attribute responses
  - Tactics form the building blocks for architectural patterns

* IEEE Definition

Sample Quality Attributes

Accessibility
Accountability
Adaptability
Administrability
Affordability
Agility
Auditability
Availability
Credibility
Compliant
Composability
Configurability
Customizability
Degradability
Demonstrability
Dependability
Deployability
Distributability
Durability
Evolvability
Extensibility
Flexibility
Installability
Interchangeability
Interoperability
Learnability
Maintainability
Manageability
Mobility
Modularity
Nomadicity
Operability
Performance
Portability
Predictability
Recoverability
Relevance
Reliability
Repeatability
Reproducibility
Reusability
Safety
Scalability
Seamlessness
Security
Serviceability
Simplicity
Stability
Survivability
Sustainability
Tailorability
Testability
Timeliness
Understandability
Usability

What does each one of these mean?

Quality Attribute Scenarios

A fully-specified quality attribute scenario consists of six parts

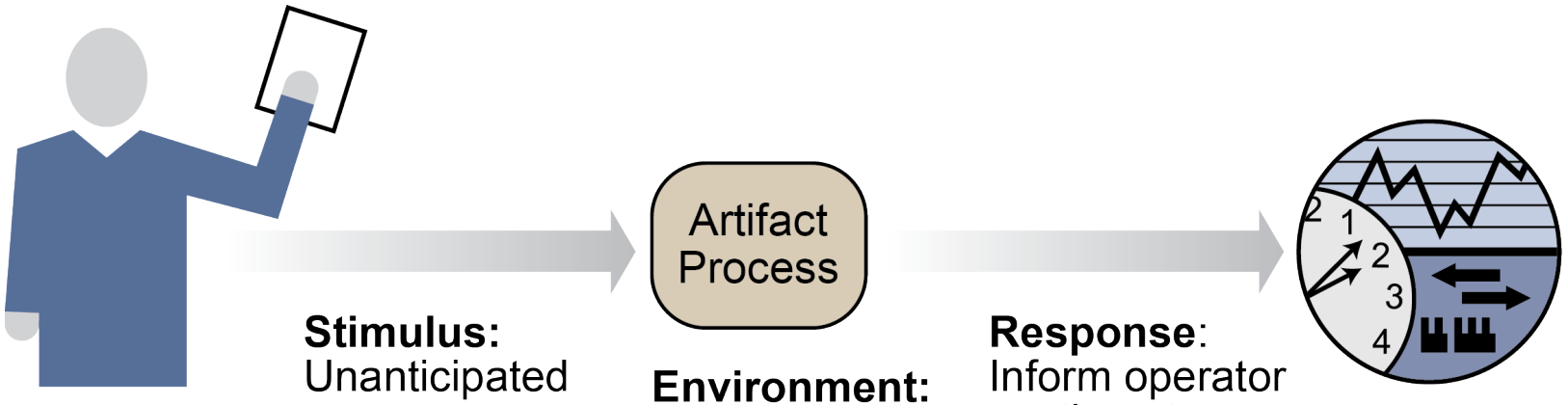
- **Stimulus:** condition effecting the system
- **Response:** activity as a result of the stimulus
- **Source of Stimulus:** entity that generated the stimulus
- **Environment:** condition under which the stimulus occurred
- **Artifact stimulated:** artifact that was stimulated
- **Response measure:** measure by which the system's response will be evaluated

Quality attribute scenarios should be as specific as possible

- A good scenario makes very clear what the stimulus is and what the desired responses of the system are, in order to avoid ambiguity

Example Availability Scenario

An unanticipated external message is received by a process during normal operation. The process informs the operator of the receipt of the message and the system continues to operate with no down time.



Architectural Patterns

Software patterns that describe solutions known to promote particular quality attribute responses

- Also known as architectural styles

An architectural pattern is often composed of multiple architectural tactics

Architectural Pattern	Subset of Tactics
Layers	• Restriction of communication paths • Semantic coherence • Abstract common services
Event Based	• Runtime registration • Use of an intermediary
Service Orientation	• Information hiding • Restriction of communication paths • Adherence to defined protocols

Tradeoff

Design decision that has a positive influence on one or more quality attribute responses and has a negative influence on other quality attribute responses

Example

- A design decision is made to increase the bit level of encryption to promote security (confidentiality) which in turn has a negative impact on performance (latency)

Categories of Design Decisions

The Coordination Model

The Data Model

Allocation of Functionality

Management of Runtime Resources

Binding Time of the Decisions in the Other Categories

Category 1: The Coordination Model

What are the communication mechanisms between the *system and external entities*?

What are the *inter-element* communication mechanisms and what are their properties (e.g., synchronous, asynchronous, hybrid coupling)?

What are the *intra-element* communication mechanisms?

Category 2: The Data Model

What is the *structure*—entities and relations, entity attributes—in the data model?

Which portions of the data model are *used by* which software elements in which *order*?

What are the *access rules* for the data items?

Where are data items *created, modified, and destroyed?*

Category 3: Allocation of Functionality

What are the *major processing steps* necessary to carry out the work of the system?

What is the *division and assignment of functionality* to software elements?

What are the key *abstractions* that can be used to provide the services of the system?

Are the elements *stateful or stateless?*

What are the *activation and deactivation dependencies* among software elements?

Category 4: Management of Runtime Resources

What *scheduling strategies* will be employed?

How much do system elements know about *time?*

What *process/thread models* will be employed?

What resources must be *managed* and what are their *limits?*

Category 5: Binding Time Decisions

The decisions made to resolve the questions on the previous slides can be *bound at a variety of times*

- design time (built in)
- compile time (e.g., compiler switches)
- build time (e.g., replace modules, pick from library)
- load time (e.g., dynamic link libraries [DLLs])
- initialization time (e.g., resource files)
- runtime (e.g., load balancing)

Summary

A software architecture is the earliest life-cycle artifact that embodies significant design decisions: choices and tradeoffs

Design decisions are made in the context of the architecturally significant requirements

- Consider the impact on the architecturally significant requirements
- Identify patterns and tactics that support the architecturally significant requirements

Agenda

Service-Oriented Architecture: Review of Terminology

Architecture-Centric Design Fundamentals

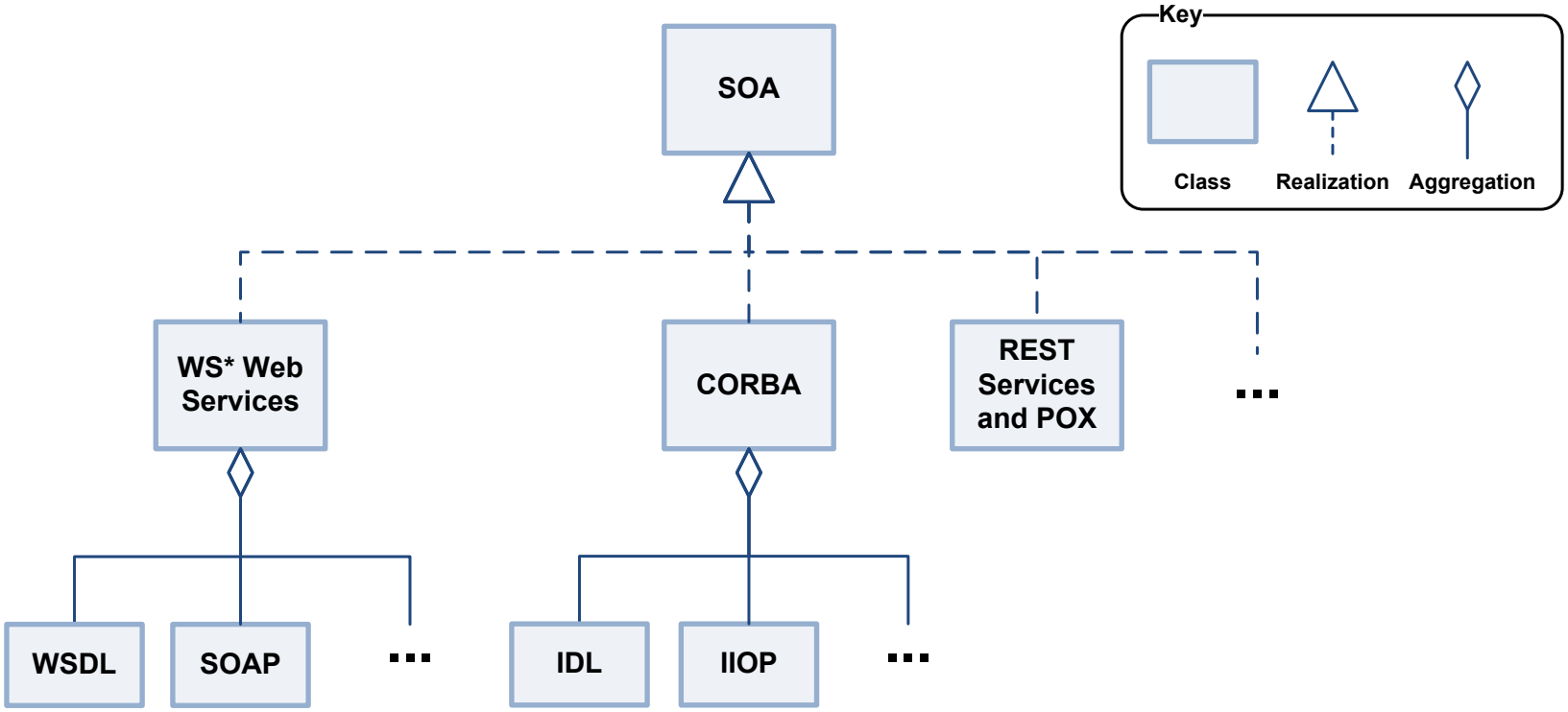
Impact of the SOA Architectural Pattern on System Quality ⇦

SOA is NOT a Specific Technology

SOA is an architectural pattern

- Systems that implement the SOA architectural pattern are called service-oriented systems

Web Services is **one** technology for SOA implementation



Quality Attributes in Service-Oriented Systems

The following slides are examples of common quality attribute scenarios for service-oriented systems, plus an analysis of how the SOA architectural pattern affects those qualities

The legend to indicate the effect is

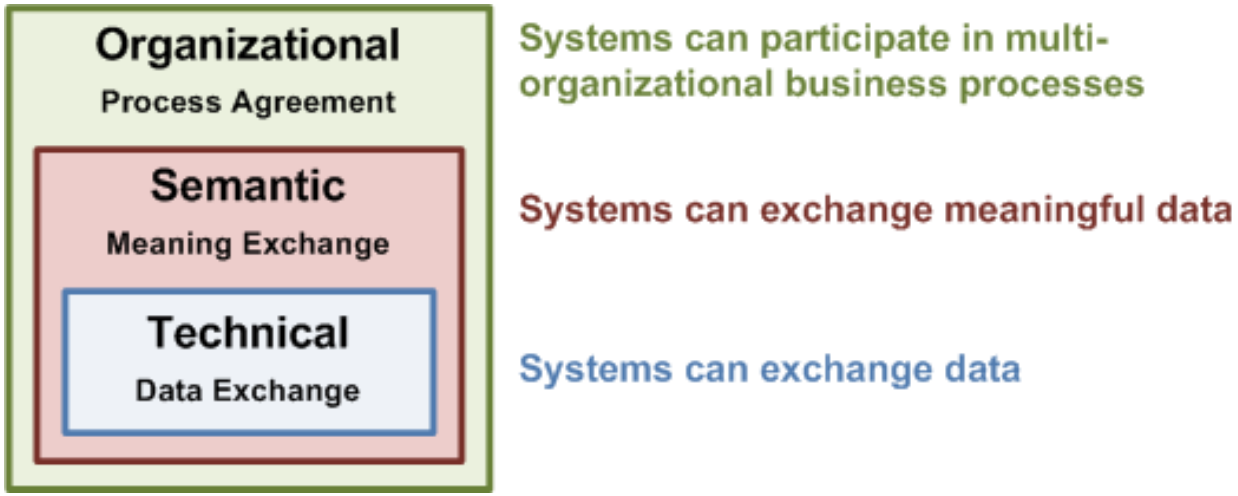
The quality is positively affected

The quality is a challenge

It depends …

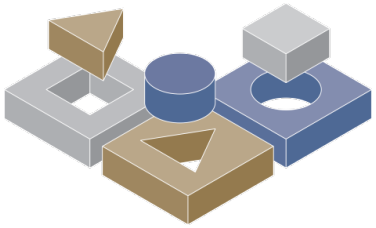
Interoperability

Interoperability is about the degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context*



* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.

Sample Interoperability Scenarios



Scenario I1: Service Consumer Integrates Service

A new business partner that uses platform ‘X’ is able to implement a service consumer module that works with our available services in platform ‘Y’ in two person-days.

Scenario I2: Legacy System Functionality Exposed as a Service

A transaction in a legacy system running on platform ‘X’ is made available as a Web service to an enterprise application that is being developed for platform ‘Y’ using Web services technology. The wrapping of the legacy transaction as a service with proper security verification, transaction management, and exception handling is done in 10 person-days.

Comments on Interoperability

Improved interoperability is a prominent benefit of SOA, especially when Web Services technology is considered

- Consumers and providers can easily use and provide services implemented in disparate platforms and different languages

Web Services cross-vendor and cross-platform interoperability

- Mature when basic standards are used: WSDL and SOAP
- Falls short when new standards are in the picture (e.g., WS-Security, WS-Transaction)

WS-I organization was created to promote interoperability (ws-i.org)

- The goal of WS-I profiles is to provide clarifications, refinements, interpretations and amplifications in areas of specific standards that are subject to multiple interpretations

Performance

Performance is about time and a software system's ability to meet timing requirements*

- How long does it take a system to respond when an event occurs?
  - Interrupts
  - Messages
  - Requests from users or other systems
  - Passage of time

Typical response measures

- Latency
- Deadline
- Throughput
- Jitter
- Miss rate
- Data loss

* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.

Sample Performance Scenarios

Scenario P1: Simultaneous Service Requests (Throughput)

The service provider can process up to ‘X’ simultaneous requests during normal operation, keeping the response time on the server less than ‘Y’ seconds.

Scenario P2: Service Request Roundtrip Time

The roundtrip time for a request from a service consumer in the local network to service ‘X’ during normal operation is less than ‘Y’ seconds.

Comments on Performance

Performance in a service-oriented context usually relates to response time or throughput and in most cases is negatively affected

- SOA is a distributed computing paradigm; the network increases response time
- Cross-platform interoperability requires intermediaries to do data marshalling and handle communication

In Web Services, use of XML impacts performance

- Studies show that XML messages can be 10 to 20 times larger than binary messages
- XML requires three CPU- and memory-intensive activities
  - Validation
  - Parsing
  - Transformation

Availability and Reliability

Availability is about minimizing service outage time by mitigating faults*

Typical response measures

- Time interval when the system must be available
- Availability time
- Time interval in which the system can be in degraded mode
- Repair time

Reliability is the degree to which a system, product, or component performs specified functions under specified conditions for a specified period of time**

Typical response measures

- Mean time between failures
- Time interval for failure detection
- Recovery time
- Resource state after recovery

* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.
** * ISO/IEC FCD 25010: Systems and Software Engineering—Systems and software product Quality Requirements and Evaluation (SQuaRE)—System and software quality models

Sample Availability and Reliability Scenarios

Scenario A1: Operation after Incorrect Input

An improperly formatted message is received by a system during normal operation. The system records the message and continues to operate normally without any downtime.

Scenario A2: Degraded Operation during Maintenance

Unscheduled server maintenance is required on server ‘X.’ The system remains operational in degraded mode for the duration of the maintenance.

Scenario R1: Failure Detection

A service becomes unavailable during normal operation. The system detects the problem and restores the service within two minutes.

Comments on Availability

SOA solutions usually rely on mechanisms provided by the execution platform or infrastructure

- Fault detection: monitor, heartbeat, timestamps, etc.
- Fault recovery: redundancy, degradation, reconfiguration, etc.
- Fault prevention: deactivation, transactions, etc.

SLAs for services typically define availability requirements as an Annual Uptime Percentage

Comments on Reliability

There are two types of reliability discussed in service-oriented systems

- Message reliability
  - Messages should be delivered in order and exactly once
  - This is usually a concern of the SOA execution platform, not the service developer
  - Available standards: WS-Reliability and WS-ReliableMessaging
  - WS-I has a Reliable Secure Profile that covers WS-ReliableMessaging

- Service reliability
  - Goal is for the service to operate without failure or to report any failure
  - Typically service have a heartbeat method to ensure service reliability

Scalability

There are two kinds of scalability

- Horizontal (scaling out): adding more resources to logical units
- Vertical (scaling up): adding more resources to a physical unit

Typical response measures

- Difference in quality of service measures
- Time interval in which the system can be in degraded mode
- Time interval for additional resources to become online/offline
- Amount of human intervention

* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.

Sample Scalability Scenarios

Scenario S1: Process Higher Volumes

Marketing landed several new high-volume accounts that will increase service request volume by a factor of 10. During normal operation, the service requests are processed without affecting the current quality of service.

Scenario S2: Process According to SLAs

Marketing landed several new high-volume accounts that will increase service request volume by a factor of 10. During normal operation, the service requests are processed according to the Service-Level Agreements negotiated with each account.

Comments on Scalability

Similar to availability and reliability, SOA solutions usually rely on mechanisms provided by the execution platform or infrastructure

In addition to horizontal and vertical scaling, a scalability tactic is service scope: configure when a new instance of a service should be created

- Once to serve all requests
- For each new service consumer
- For each new request

Another tactic is to design services to be stateless

- To enable replication and load-balancing

Security

Security is a measure of the system's ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized*

Typical response measures

- Time/effort/resources to circumvent security measures with probability of success
- Probability of detecting attack
- Probability of identifying individual responsible for attack or access/ modification of data and/or services
- Percentage of services still available under denial-of-service attacks
- Restore data/services
- Extent to which data/services damaged and/or legitimate access denied

* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.

Sample Security Scenarios

Scenario 1: Leak of Sensitive Data

An attack is launched attempting to access confidential customer data. The attacker is not able to break the encryption used in all the hops of the communication and where the data is persisted. The system logs the event and notifies the system administrators.

Scenario 2: Identification of Third-Party Service Provider

A request needs to be sent to a third-party service provider, but the provider's identity cannot be validated. The system does not make the service request and logs all relevant information. The third party is notified as well as with the system administrator.

Comments on Security

Some aspects of SOA solutions impact security

- Messages may be transmitted in text format and contain sensitive metadata
- If external services are used
  - Identity of service providers should be authenticated
  - Data should be transmitted and stored securely
- Data in the service registry should be reliable
- Authorization of clients to access services has to be configured

Transport security is usually addressed at the network level (e.g., SSL)

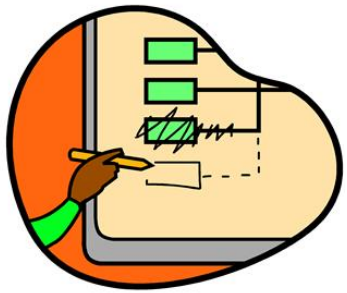
There are tradeoffs with interoperability, modifiability and performance

Modifiability

Modifiability is the ability to make changes to a system quickly and cost effectively*

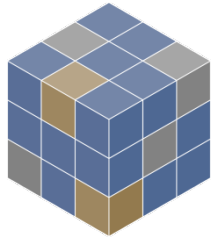
Typical response measures

- Cost in terms of number of elements affected, effort, money
- Extent to which this affects other functions or quality attributes



* Bass, L.; Clements, P.; & Kazman, R. Software Architecture in Practice, Second Edition. Boston, MA: Addison-Wesley, 2013.

Sample Modifiability Scenarios



Scenario M1: Change in Service Implementation

A service provider changes the service implementation, but the syntax and the semantics of the interface do not change. This change does not affect the service users.

Scenario M2: Change in Service Interface

A service provider changes the interface of a service that is publicly available. The old version of the service is maintained for 12 months, and existing service users are not affected within that period.

Modifiability

SOA promotes loose coupling between service consumers and providers

- Services are modular and self-contained
- There are few well-known dependencies, therefore reducing the cost of modifying the services

Changing published interfaces can be a challenge, but SOA solutions deal with these changes through

- Versioning mechanisms
- Flexible contracts specified in XML
- Special components in the infrastructure, such as the registry

Summary

Architectural design patterns are typically chosen to promote one or two qualities that are important to an organization

Service-orientation promotes interoperability and modifiability at the expense of performance

Service-orientation is a starting point that is often augmented by other patterns and tactics to create a complete architectural solution

Join Us for Part 2!

SOA Infrastructure Design Considerations

Service Design Considerations

SATURN 2013

Software Engineering Institute | Carnegie Mellon

Minneapolis, Minnesota

April 29 to May 3, 2013

www.sei.cmu.edu/saturn/2013