Goal:

Offer practical information to help the architecture evaluation of an SOA system
Agenda

Introduction
SOA Architectural Approaches
SOA Design Questions
Closing

What is Service Oriented Architecture?

SOA is an architectural style where systems consist of service users and service providers

A service is a self-contained, distributed component with a published interface that stresses interoperability, is discoverable and dynamically bound.

But what is a service?
SOA and Web Services

SOA is an architectural style
Web Services is a technology used to implement SOA

How Does It Work?

An example...

Key:
- UDDI registry
- MS .NET application
- J2EE service
- SOAP message over http
- service endpoint in WSDL
ATAM

In the analysis, the evaluation team:

- identifies architectural approaches
- asks quality attribute questions about the design decisions
- identifies and records risks and tradeoffs

In SOA systems,
- What architectural approaches could be used?
- What quality attribute questions could the evaluators ask?

Agenda

Introduction
SOA Architectural Approaches
SOA Design Questions
Closing
SOA Communication Approaches

How’s the communication between service user and provider?

Main alternatives:

- Web Services (SOAP)
- REST
- Messaging systems

The SOA environment may involve a mix of these along with legacy protocols

Web Services – RPC-Encoded SOAP

![Diagram of Web Services: Service user, SOAP request, Wrapper, SOAP response, Component (service implementation), Key: Service user component (e.g., .NET Windows application), Service provider component (e.g., EJB), http, Native call-and-return mechanism.]
Web Services – Document-Literal SOAP

- SOAP request: business document in XML
- SOAP response: business document with processing results
- Wrapper that realizes Web services interface
- Validation and transformation of business document
- XML schema (ex: PlaceOrder.xsd)
- Processing request

Key:
- Service user component (e.g., .NET Windows application)
- Service provider component (e.g., EJB)
- http
- Native call-and-return mechanism

Document-Literal vs. RPC-Encoded

<table>
<thead>
<tr>
<th></th>
<th>RPC-Encoded</th>
<th>Document Literal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interoperability</td>
<td>Less interoperable due to SOAP encoding</td>
<td>Recommended by WS-I</td>
</tr>
<tr>
<td>Performance</td>
<td>Processing overhead to encode payloads</td>
<td>No encoding overhead</td>
</tr>
<tr>
<td></td>
<td>Requires DOM parsing</td>
<td>Allows other parsing technologies</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Service interfaces closer to programming language</td>
<td>Harder to implement and debug XML schemas, processing and transformation code</td>
</tr>
<tr>
<td></td>
<td>Clients more susceptible to interface changes</td>
<td>More flexibility in changing definition of business documents</td>
</tr>
</tbody>
</table>
Representational State Transfer – REST

Resource. Examples:
- Current weather for zip code 15219
- Temperature averages for city Pittsburgh in May

Resource URI. Examples:
- http://www.weather.com/current/zip/15219

For each resource, there is a representation
- Format is usually XML

<table>
<thead>
<tr>
<th>Operations on resources</th>
<th>HTTP Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>http post</td>
</tr>
<tr>
<td>Retrieve</td>
<td>http get</td>
</tr>
<tr>
<td>Update</td>
<td>http put</td>
</tr>
<tr>
<td>Delete</td>
<td>http delete</td>
</tr>
</tbody>
</table>

REST Compared to SOAP-Based Web Services

REST is better:
- Interoperability – requires only http support
- Easier to learn
- Modifiability – only the data contract has to be understood, the interface contract is uniform
- Performance – no intermediaries or marshalling required

SOAP-Based Web Services is better:
- Tool support
- Support for security, reliable messaging and transaction management
- "Network knowledge" and skill base due to widespread adoption
Evaluating a Service-Oriented Architecture

Messing Systems

Based on IBM WebSphere MQ, Microsoft MSMQ, Oracle AQ, SonicMQ and similar products
Offer asynchronous message exchanges (point-to-point or pub-sub)

Benefits:
- Reliability
- Loose coupling
- Scalability

Challenges:
- Asynchronous model is more complex
- Interoperability – proprietary messaging systems require bridges to interact

Integration Approach

There are multiple possible integration approaches
Commonly divided into:
- Point-to-point
- Hub-and-spoke
When to Use Point-to-Point or ESB

Point to point is most acceptable in environments that are:

- Small in number of services and applications
- Homogenous in technology
- Low pace of change (business and technology)

ESBs are most acceptable in environments that are:

- Large
- Technically diverse
- Rapidly changing
Point to Point vs. ESB Tradeoffs - 1

<table>
<thead>
<tr>
<th>Point-to-Point</th>
<th>ESB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modifiability</td>
<td>Changes to a service interface induces change to all connected applications</td>
</tr>
<tr>
<td>Performance</td>
<td>No transformation and routing overhead</td>
</tr>
<tr>
<td>Security</td>
<td>Authentication and authorization managed case-by-case by each service</td>
</tr>
</tbody>
</table>

Point to Point vs. ESB Tradeoffs - 2

<table>
<thead>
<tr>
<th>Point-to-Point</th>
<th>ESB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serviceability</td>
<td>Problem determination spread across applications—no central point to manage connectivity</td>
</tr>
<tr>
<td>Reliability</td>
<td>Strong coupling may result in complex failure modes and unintended dependencies</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Each service to service connection must be compatible</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Agenda

Introduction
SOA Architectural Approaches
SOA Design Questions
Closing

Synchronous or Asynchronous Services?

Choice depends on
• Business requirements
• QA requirements
• Existing components capabilities
Synchronous SOAP-Based WS

Service user

Wrapper that realizes Web services interface

Validation and transformation of business document

Processing request

SOAP request: business document in XML

SOAP response: business document with processing results

Key:
- Service user component (e.g., .NET Windows application)
- Service provider component (e.g., EJB)
- http
- Native call-and-return mechanism

Asynchronous SOAP-Based WS

Service user

Callback endpoint

Wrapper that realizes Web services interface

Validation and transformation of business document

Send client response

SOAP request: business document in XML

SOAP response: http 200 only

SOAP request: business document w/ processing results

SOAP response: http 200 only

Send client response

Back-end processing

Key:
- Service user component (e.g., .NET Windows application)
- Service provider component (e.g., EJB)
- http
- Native call-and-return mechanism
Synchronous vs. Asynchronous Services - 1

<table>
<thead>
<tr>
<th>Synchronous Services</th>
<th>Asynchronous Services</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Modifiability</strong></td>
<td></td>
</tr>
<tr>
<td>☀ Simpler to implement</td>
<td>☀ More complex logic to deal with waiting, callback and correlation</td>
</tr>
<tr>
<td>☀ Behavior (e.g. timing) dependencies beyond interface syntax make replacement more difficult</td>
<td>☀ Lower coupling (components can be more easily replaced)</td>
</tr>
<tr>
<td>☀ More difficult to insert an ESB because of performance or behavior dependencies</td>
<td>☀ Ease of inserting ESB or other brokering into conversations</td>
</tr>
<tr>
<td>☀ Easier control of serialization of parallel requests</td>
<td>☀ Control of sequencing drives complex correlation, exception management and timeout designs</td>
</tr>
</tbody>
</table>

Synchronous vs. Asynchronous Services - 2

<table>
<thead>
<tr>
<th>Synchronous Services</th>
<th>Asynchronous Services</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td></td>
</tr>
<tr>
<td>☀ Designed to achieve better responsiveness</td>
<td>☀ Overhead of messaging</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td></td>
</tr>
<tr>
<td>☀ Poor for large applications</td>
<td>☀ Best scalability for large applications</td>
</tr>
<tr>
<td><strong>Reliability</strong></td>
<td></td>
</tr>
<tr>
<td>☀ More susceptible to complex distributed failures</td>
<td>☀ Better independent operation and fault-tolerance</td>
</tr>
<tr>
<td>☀ Simpler error and exception handling designs</td>
<td>☀ More complex error/retry logic</td>
</tr>
</tbody>
</table>
HTTPS or Message-Level Security?

Main difference:
- HTTPs allows point to point security
- Message-level allows end to end security

One doesn’t exclude the other

HTTPS

HTTPS is HTTP over SSL
- Entire message encrypted from point to point
- Reasonable protection from eavesdroppers and “man-in-the-middle” attacks

Problem: message lifecycle usually is longer than point to point
- Multiple hops
- Intermediaries with different policies and controls
- Messages persisted at various points
Message-Level Security

Service users and providers bind security tokens to messages using WS-Security

- Allows encrypting and signing all or just parts of the message
- Tokens represent claims made by the sender (e.g., authentication, authorization, confidentiality, integrity)
- WS-Security does not address security infrastructure such as key management

HTTPS vs. Message Level Security

<table>
<thead>
<tr>
<th>HTTPS</th>
<th>Message Level Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Some performance overhead but generally faster response times</td>
</tr>
<tr>
<td>Complexity</td>
<td>Has been around and is well understood</td>
</tr>
<tr>
<td>Interoperability</td>
<td>More interoperable</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Inflexible all or nothing</td>
</tr>
<tr>
<td>Security</td>
<td>Security is only enforced from point to point</td>
</tr>
</tbody>
</table>
Coarse- or Fine-Grained Services?

Coarse-grained service typically consists of a complete business process
Fine-grained service usually performs small functions

The following should influence service interface design:

- Transactions and state
- QA requirements

Coarse- vs. Fine-Grained Services

<table>
<thead>
<tr>
<th></th>
<th>Coarse Grained</th>
<th>Fine Grained</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>▓ Improved by reducing the number of messages</td>
<td>▐ Requires more message exchanges</td>
</tr>
<tr>
<td>Testability</td>
<td>▓ Simplifies testing by limiting the number of possible paths</td>
<td>▐ Testing is more challenging because the order of operations is not controlled</td>
</tr>
<tr>
<td>Flexibility</td>
<td>▐ Not as flexible</td>
<td>▓ More flexible in assigning authorization for different operations</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▐ Give clients more control over the steps of an operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>▐ Enables service reuse and composition</td>
</tr>
</tbody>
</table>
Static or Dynamic Web Services?

Dynamic WS:

```
Directory of Services

Order Processing Notification

Package Tracking Service

query service

serve contract and address

register service

getPackHistory(#30942)

response

Web store
Service user

Carrier company
Service provider
```

Static WS:

```
Order Processing Notification

Package Tracking Service

getPackHistory(#30942)

response

Web store
Service user

Carrier company
Service provider
```

Static vs. Dynamic Web Services? - 2

<table>
<thead>
<tr>
<th></th>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Performance</strong></td>
<td>- Less overhead because service location is known during design</td>
<td>- Service lookup overhead</td>
</tr>
<tr>
<td></td>
<td>- No WSDL processing</td>
<td>- Overhead of WSDL processing</td>
</tr>
<tr>
<td><strong>Modifiability</strong></td>
<td>- Service user and provider more tightly coupled</td>
<td>- Dynamic binding enables service provider location to change without affecting service user</td>
</tr>
<tr>
<td><strong>Availability</strong></td>
<td>- Failover logic has to be in the service user or other intermediary</td>
<td>- Directory can route service calls (for failover or load-balancing)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Directory can be a SPOF</td>
</tr>
</tbody>
</table>
Agenda

Introduction
SOA Architectural Approaches
SOA Design Questions
Closing

Important Takeaways

ATAM with no changes can be used to evaluate SOAs
ESB versus point-to-point, pros and cons
SOAP is not the only option for SOA communication – REST and Messaging Systems also work
Once you understand the importance of each QA requirement, you can weigh the relevance of each design question
SOAs involve a lot of technical design considerations
Questions – Now or Later

Paulo Merson – pfm@sei.cmu.edu
Phil Bianco – pbianco@sei.cmu.edu
Rick Kotermanski – rek@summa-tech.com

- What’s SaaS?
- What are the typical risks found in an SOA evaluation?
- Is ESB a product, something I have to develop, an infrastructure service of my application server, or something else?