Presentation Contents

- Rational and historical context
- History of Distributed and service-oriented Computing
- SOA ‘is’ and ‘is NOT’
- Service-Oriented Architectures
- Definition of Web Services
- Examples
- Terms and How Web Services Work
- Benefits & Constraints
- Web Service Standards
- The Future of Web Services
**Service Oriented Architecture**

– A 20 year old concept - Rejuvenated

- Traditionally, inter-application communication is difficult to integrate
  - Synchronous communication (RPC-style)
  - Asynchronous communication (Messaging-style)
  - N (N-1) number of interfaces for N applications

- Adoption of XML, Web Services and Service Oriented Architecture is expected to explode

- Goal of the Service Oriented Architecture
  - Interconnect disparate applications
  - Achieved by reuse of objects and components

---

**SOA historical context**

- 1990s - Enterprise Software Bus
  - Sets of enterprise-wide middleware standards
  - The ideal was: “Ubiquitous, technology-independent, enterprise-wide standard for communication between software modules”
  - Reality:
    - Not a global software bus
    - we now have software heterogeneity and middleware heterogeneity.
    - Nearly as many incompatible middleware systems as application
      - CORBA etc uses to solve point-to-point integration on project-by-project bases
SOA historical rationale

- Service architectures are a Post-RDMS phase of enterprise integration
  - Middleware tools
  - EAI (enterprise application integration) platforms
- SOA moves away from
  - Fine-grained technology-oriented notions such as Java objects and database rows
- Focuses instead on
  - Business level services and business-level transaction orientation

Benefits of SOAs

- Because a service only exposes its interfaces, we can use legacy applications as services.
  - This may require a gateway that translates service requests into a native API.
    - IBM has a Web services gateway.
  - You don't have to modify your legacy applications to use them as services.
  - As your legacy applications evolve, the interfaces can remain the same.
SOA is and is NOT…

- **NOT** an enterprise technology standard
  - Not dependent on a single technical protocol such as IIOP or SOAP
- **It *is*** a architectural blueprint
  - Incorporating many technologies
  - Not requiring specific protocols or bridging technologies
- **Goals:**
  - defining cleanly cut service contracts with a clear business orientation
  - Opening up back end systems and services
The notion of a service

- Webster’s dictionary--
  1. “useful labor that does not produce a tangible commodity” and
  2. “a facility supplying some public demand”
- Technical definition--
  3. “a remotely accessible, self-contained application module.”

They are abstracted from their technical detail including location and discovery and provide business functionality vs technical functionality.

Service oriented architectures

![Diagram of Service Oriented Architecture (SOA) with components: SOA, Application Front end, Service, Service Repository, Service bus, Contract, Implementation, Interface, Business logic, Data.]
Service oriented architecture defined

• A SOA is a software architecture that is based on the key concepts of an application front end, service, service repository and service bus. A service consists of a contract, one or more interfaces and an implementation.

Krafzig et al. 2006

What is a Service-oriented architecture?

• Architecture that leverages open standards to represent software assets as services
• Provides a standard way of representing and interacting with software assets
• Individual software assets become building blocks that can be reused in developing other applications
• The preferred method of building an SOA is via Web services.
An instance of a SOA: Web Services

- **Web Services are free-standing business applications built to operate over the Internet.**
  - Standardized language allows them to interact with other web services within a given company and across company lines.
  - Opportunities for collaboration and co-opetition
  - Opportunity for intra and inter organizational integration
  - Opportunity for reuse on a grand scale

A sample manufacturing process

- Receive order
- Credit check
- Check inventory
- Order parts
- Manufacture
- Check quality
- Send invoice
- Ship

All of these steps are good candidates for integration.
Integration: A credit check

- It would be nice if the system automatically checked the customer's credit.
  - Maybe we contact the accounting department first. If this is a customer in good standing, that might be all the information we need.
  - For new customers, maybe we contact an external credit bureau.
- *All* of the steps in the process could be automated.


- Google Web Services APIs.
  - Business partners and Affiliates.
    - Build search engines in other Web sites using Google database (4 billion pages).
    - Incorporate Spelling check functionality in other Web sites.
- Amazon Web Services APIs
  - Strategic partners and Associates (Merchants and Publishers).
    - Access catalog data.
    - Create and populate an Amazon shopping cart.
    - Initiate checkout process.
    - Integrate into in-house application for report generation and management reporting.
- Google and Amazon are resources for any application.
Benefits to using/having web services

• Solves three primary concerns:
  – Cost of implementation.
  – Complexity of use.
  – Solution/Application Flexibility.
• Secures real time transaction delivery and consumption.
  – Enables “cross-silo communications” integrating more data.
• Eases use & implementation.
  – Reduces development time.
  – Recycles service components.
• Increases flexibility.
  – Independent of hardware, programming language, operating system, etc.
  – Improves firm’s ROI - maximizes current & future applications.

Benefits: realized how?

• XML and Web Services perfect for realizing SOA
  – Self describing
  – Independent of hardware, programming language, container, etc
  – Accommodates independent change/versioning
  – Standard XML moving through network between applications
Web aspects vs. Services aspects

**Web-aspects:**
- Web-based protocols
  - SOAP over HTTP transport
- Interoperability
- XML-based

**Services-aspects**
- Modular
  - Independent and reusable
- Available
- Described
  - Machine-readable description
- Implementation-independent
- Published
  - Repository based and directory listed

High level examples
Key Protocols: How do Web Services work?

- **XML**
  - Extensible Markup Language
- **SOAP**
  - Simple Object Access Protocol
  - Request-response (RPC) protocol, based on HTTP
    - But other transport mechanisms feasible RMI/IIOP, IM, Web Sphere, etc.
  - XML message format: envelope, address and body
- **WSDL**
  - Web Service Description Language
  - Enforces separation between the interface and the implementation
  - It is extensible; the core specification describes the abstract interface and structure of ports and bindings.
  - Allows templates and skeletons to be used/embedded anywhere
- **UDDI and WSIL**
  - Universal Description Discovery and Integration; a standard for publication of web services
  - Provides for global directory of services; key to finding/identifying services
  - Gives pointers to WDSL or UDDI services listings
  - WSIL—Web Services Inspection Language; a lightweight version of UDDI

What about (D)COM, RMI, and CORBA?

- All are **Good Remote Procedure Call (RPC) technologies**
- All represent **Good Object Models**
- But all have platform, language, and portability issues
- Web Services is not “superior,” but it is more widely accepted and accessible.
  - You can put XML on just about anything…
**Standards:** Standards? What Standards?

- **Demand for Standards**
  - Technology being adopted at amazingly quick pace demands standards

- **Web Services Standards Organization**
  - OASIS - Organization for the Advancement of Structured Information Standards
  - UDDI.org – Universal, Description, Discovery and Integration
  - W3C - World Wide Web Consortium
  - WS-I - Web Services Interoperability Organization
  - IETF - Internet Engineering Task Force

- **Members**
  - IBM, Microsoft Corporation, Sun Microsystems, BEA Systems and other software vendors

**Web services standards**

- **Bus. Process Execution Language**
- **Management**
- **Enterprise**

- **WS-Coordination**
- **WS-Security**
- **WS-Reliable Messaging**
- **WS-Policy**

- **WSDL**
- **UDDI**
- **WSIL**

- **SOAP**
- **SOAP attachments**
- **Other protocols**
- **Other services**

- **Transports (HTTP, HTTPS, JMS, SMTP, FTP, etc.)**

**Description & Discovery**

**Messaging & Encoding**

**Quality of Service**
The basic standards

Quality of service standards
## Enterprise standards

<table>
<thead>
<tr>
<th>Bus. Process Execution Language</th>
<th>Management</th>
<th>Enterprise</th>
</tr>
</thead>
<tbody>
<tr>
<td>WS-Coordination</td>
<td>WS-Security</td>
<td>WS-Reliable Messaging</td>
</tr>
<tr>
<td>WS-Transactions</td>
<td></td>
<td>WS-Policy</td>
</tr>
<tr>
<td>WSDL</td>
<td>UDDI</td>
<td>WSIL</td>
</tr>
<tr>
<td>SOAP</td>
<td>SOAP attachments</td>
<td>Other protocols</td>
</tr>
<tr>
<td>XML, XML Infoset</td>
<td></td>
<td>Other services</td>
</tr>
<tr>
<td>Transports (HTTP, HTTPS, JMS, SMTP, FTP, etc.)</td>
<td></td>
<td>Transport</td>
</tr>
</tbody>
</table>

### Description & Discovery
- WSDL
- WSIL

### Messaging & Encoding
- SOAP
- Other protocols

### Transport
- XML, XML Infoset
- Other services

## Web services architecture

![Web services architecture diagram]

- Development environment
- Discovery (UDDI, etc.)
- WSDL Descriptions
- Deployment Descriptors
- System services
- Test client
- Client proxy
- Transport
- Server
- Web service
- Legacy
Application development tools

Developed applications
A SOAP envelope

```
<soapenv:Envelope>
  <soapenv:Header>
    ... 
  </soapenv:Header>
  <soapenv:Body>
    ... 
  </soapenv:Body>
</soapenv:Envelope>
```
Invoking a SOAP service

• A SOAP client builds a request envelope (an XML document), sends it to a service, then typically waits for the response.
  – The response is another envelope that contains information about the request.
  – The client doesn't have to wait for a response. The request could also be a one-way message with no response, but most clients do things synchronously.

Two kinds of SOAP messages

• There are two ways to send a SOAP message:
  – RPC style: The contents of the <Body> mimic a method call in a programming language.
SOAP message structure

- We'll start with RPC-style SOAP applications because they're conceptually simpler.
- There are request and response messages:
  - A request message is a method call on a remote object
  - A response message returns the result of running the requested method

A SOAP request

```xml
<soapenv:Envelope>
  <soapenv:Body>
    <ns1:askQuestion>
      <ns1:question>Will this work?</ns1:question>
    </ns1:askQuestion>
  </soapenv:Body>
</soapenv:Envelope>
```
A SOAP response

```xml
<soapenv:Envelope>
  <soapenv:Body>
    <ns1:askQuestionResponse>
      <ns1:response0>
        My sources say yes.
      </ns1:response0>
    </ns1:askQuestionResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

RPC-style SOAP services

- With a RPC-style SOAP service, the XML elements in the SOAP envelope have to match the parameters of the Web service’s method.
  - If the `askQuestion` method takes a string as its parameter, the SOAP `<Body>` element must contain one XML element that contains a string.
RPC-style SOAP services

- The Java code:
  ```java
  public String askQuestion(String question)
  ```
- Inside the SOAP envelope:
  ```xml
  <ns1:askQuestion>
  <ns1:question>
    Will this work?
  </ns1:question>
  </ns1:askQuestion>
  ```

Document-style SOAP services

- RPC-style SOAP services work well for method calls with relatively simple parameters.
- What if we want to use an XML document as a parameter?
  - That's where document-style messages come in.
- Common practice is to use an XML schema to validate that document.
Client/service messaging

In this picture, a SOAP envelope is sent from the client to the SOAP engine running on the application server.

How it works

Step 3. The SOAP engine converts the SOAP message to a method call on the appropriate service, passing any data from the client to the service.

The SOAP engine may create new Java objects, or it may use existing ones.
Step 4. The service executes the appropriate method, using the data from the client. The service has no idea it’s being invoked because of a SOAP request.

Step 5. The service returns the results of running the requested method with the data from the client.
How it works

Step 6. The SOAP engine puts the results from the service into a SOAP envelope and sends the response back to the client.

Discovery standards

- **Bus. Process Execution Language**
- **Management**
- **Enterprise**
- **WS-Coordination**
- **WS-Security**
- **WS-Reliable Messaging**
- **WS-Policy**
- **WS-Transactions**
- **UDDI**
- **WSIL**
- **Description & Discovery**
- **WSDL**
- **SOAP**
- **SOAP attachments**
- **Messaging & Encoding**
- **XML, XML Infoset**
- **Other protocols**
- **Other services**
- **Transports (HTTP, HTTPS, JMS, SMTP, FTP, etc.)**
- **Transport**

Enterprise Architectures
Web service discovery : UDDI

• Universal Description, Discovery, and Integration
• Defines standards for a distributed registry of Web Services.
  – You can search by provider information, by service type, by service interface, or create your own categories.
• Almost all production registries are behind firewalls or on VPNs.

WSIL

• Web Services Inspection Language
• With WSIL, we do discovery by asking a known partner for a list of their services.
• Both UDDI and WSIL act as name servers in that they help clients find Web service endpoints.
• See xmethods.net for WSIL and UDDI links to 100s of Web services.
WSDL and service discovery

• If we’re using discovery, the WSDL file tells us that (for example) a given Web service is RPC-style and that it needs two strings and an integer as its arguments.
• **What we can't discover is the business logic to find or create those arguments.**

WSDL and service discovery

• In practice, **we’ll know how to invoke the service before we do discovery.**
  – In other words, we'll write our SOAP client code to a well-known programming interface.
  – What we want to discover is the address of the service's host and the name of the service we're invoking.
The key: Integration

• In all aspects of this process, we need an automated way to:
  – Move structured information from one place to another
  – Communicate with and invoke services
  – Coordinate the way services work together

• A service-oriented architecture makes these things possible.

Figure 1: Key Technologies and Considerations for Building Service-Oriented Applications
Constraints

What Could Undermine Web Services?

- No standardized platform to create Web services
  - Need for businesses to support multiple generations of non-standardized applications & solutions
  - Sun’s Java J2EE environment
  - Microsoft’s C# language on .Net platform
  - Potential Linux development
- Security, operation reliability, & data quality
  - As more systems are inter-dependent, consistency is key
  - “Users & vendors must acknowledge phased change versus big-bang/holistic architectural overhauls.”

Future developments

- Use of Web Services will tremendously increase
  - Compounded growth in bandwidth
  - Increased reliability
  - Increased security standardization
  - Rapid development tools within popular Integrated Development Environments of web services
    - Microsoft, IBM, Sun Microsystems, BEA other ISVs

George Gilder, author of Telecosm, asserts that "bandwidth grows at least three times faster than computer power." This means that if computer power doubles every eighteen months then communications power doubles every six months.
What’s next

- Better Enterprise integrations with web services
- Better standards
  - e.g., Web Services Business Process languages (BPEL4WS)
    - A Joint standard
    - Graphical tool support; XML code generating
- Enterprise adoption and wide spread use

Keep this in perspective

- Web services are another tool for your toolbox.
  - That doesn’t mean you should throw away all of your other tools!
  - The architecture is designed for universal connectivity.
  - Like all design decisions, these things come at a price.
Discussion question:
Where does this fit in an Enterprise-level Architecture?

- Architectural standard
- Part or whole?
- ???
- The concept of ‘connectors’ is important …why?

The Roots…

From Mainframe to Web Services

Applications before the Web (1995)
- Users were expected to spend time learning the system
- Technologies were mixed (COBOL, C++, client/server, etc.)
- Dedicated application hardware
- Integration and changes were difficult
  - Use of EDI, CORBA
  - Lack of standards
- Supply chain management begins
- Security is simpler because network access is limited
- 12-18 month development cycles
Roots: the results of the .com era

- Anybody might use our system (professional users, customers, competitors, random folks)
- Applications built on n-tiered architectures
- Virtualized, distributed servers -
  - If we can run our Web servers anywhere, why can't we run our back-end applications anywhere?
- Real standards emerge in the marketplace
  - (HTTP, HTML, XML, Java, J2EE, etc.)
- Supply chain management drives integration

Roots: The results of the .com era (2)

- Security becomes a major headache (crackers, DoS attacks, etc.)
- 3-6 month development cycles
- Integration is crucial:
  - With our partners
  - With our suppliers
  - With our customers
  - With government agencies
  - With a company we acquire
  - Maybe even with our competitors!
- If we can't integrate our data and applications easily, we're in trouble.
The evolution of e-business

- Access points, transactions, and data are growing exponentially.
- The networked economy is driving the evolution of e-business, and integration is the key.