Today

Web Service Technologies

- Extensible Markup Language (XML)
- JavaScript Object Notation (JSON)
- Hypertext Transfer Protocol (HTTP)
- Java Servlets

REST Web Services

SOAP Web Services
- Web Service Description Language (WSDL)
- SOAP

Web Service Data

- Platform independent
- Text resolved
- Short lived
- Limited in size (kilobytes)
- Descriptive

Extensible Markup Language (XML)

- Designed to structure and describe data
- A family of related technologies
- Widely used in a range of technologies today
- Often used for data validation and migration

XML

- Tags are user specified (not predefined)
- XML documents are hierarchical and extensible
  - applications can parse extended documents
  - unrecognized extensions are ignored
- XML is platform, software, and hardware independent
- XML is used to structure, store, and send information
- XML is both human and machine-readable
**XML Interpretation**

- Well-formed: conforms to XML syntax specifications e.g., tags properly nested, tags closed, attributes quoted
- Valid: conforms to a namespace-defined syntax i.e., document matches a syntax definition (schema)

**XML Example**

```
<root>
  <element1>
    <subelement>content</subelement>
  </element1>
  <element2/>
</root>
```

**XML Parsers**

- Programs that traverse XML documents and retrieve data
- Usually transforms data into a custom format
- Can ignore data and insert default values
- Usually slow & memory consuming
- Should incorporate extensive fault tolerance techniques

**Simple API for XML (SAX)**

- Uses an event-driven parse model
- Performs online parsing of data streams
- Requires very little memory
- Somewhat cumbersome to use

**Document Object Model (DOM)**

- Constructs a node tree from a document
- Allows off-line processing of data
- Requires large amounts of memory
- Simple to use

**XML Path Language (XPath)**

- Expression language for addressing parts of XML documents
- Used for pattern-based searching of XML documents
- Possible to search on elements, attributes, and tag data
- Integrates well with the DOM API
**XML Schema**

- Describes the structure of an XML document
- Is used to validate XML documents
- Is itself an XML document
- Typically difficult to read
- Defines elements, attributes, and data types

**XML Schema Basics**

- Schemas validate documents
- Schemas contain
  - Type definitions
  - Element declarations
  - Attribute declarations
- Simple types
  - Cannot have sub-elements
    - (no child elements or attributes)
  - Predefined type or derived from predefined type
- Complex types
  - Can have sub-element(s) - elements and/or attributes
- Element declarations can reference both simple and complex types
- Attributes can only reference simple types

**JavaScript Object Notation (JSON)**

- A text-based open-standard data representation format
- A lightweight alternative to XML
- Designed for "human readable data exchange"

**JSON Data Types**

- Number (double)
- String (backslash-escaped unicode)
- Boolean
- Array (ordered sequence)
- Object (unordered collection of key-value pairs)
- null (empty)
Hypertext Transfer Protocol (HTTP)

- Text-based
- Application-level protocol (mostly used over TCP)
- Client-driven (requests and responses)
- Stateless (sessions stored in cookies / URL rewrites)
- Can handle text as well as binary data (encoded as text)

HTTP Request

- Request line (method + URI + protocol)
- Headers (request information)
- Body (optional)

HTTP GET Request Example

GET /kursar/3DVO95/VT11/test.html HTTP/1.1
Host: www.cs.umu.se

HTTP Response

- Status line (protocol + status code + reason phrase)
- Headers (response information)
- Body (response data)
HTTP Response Example

HTTP/1.1 200 OK
Date: Thu, 26 Jan 2011 12:31:00 GMT
Server: Apache/2.0.61 (Unix)
Accept-Ranges: bytes
Content-Length: 50
Content-Type: text/html; charset=ISO-8859-1

<html>
<head>
</head>
<body>
text
</body>
</html>

HTTP Request Methods

- HEAD - simulate a get request
- GET - retrieve resource
- POST - submit data to resource
- PUT - upload resource
- DELETE - delete resource
- TRACE - echo request
- OPTIONS - query server for supported methods
- CONNECT - create TCP/IP tunnel

HTTP Transactions

1. A Connection is established
2. A HTTP request is received
3. The (logical) path in the request is translated
4. The requested resource is identified (via the path)
5. A server module handling that resource is invoked
6. The module processes the request and generates a reply
7. A mime-type is provided and a HTTP response is created
8. The HTTP response is sent (possibly in increments)

Java Servlets

- Java classes
- Implements the Java Servlet API interfaces (predates WS)
- Receives requests and generates responses
- Can be written manually
- Must be thread-safe
- Can be generated for many environments (e.g., JSP)
- Hosted in Servlet containers

The Servlet service() method

- Part of a service pattern
  - init() - called on Servlet instantiation
  - service() - called for each request
  - destroy() - called on container shutdown
- Should not be implemented directly
- Inherit base class and implement handler methods
- Distinct handlers for each HTTP method (e.g., doGet())

1. service() parses request and determines HTTP method
2. service() calls appropriate handler method
3. Handler method processes request

REpresentational State Transfer (REST)

- Originally described in PhD thesis by Roy Fielding (coauthor of HTTP v1.1)
- Describes a web-like architectural style for modelling interaction with stateful resources using HTTP
- Does not define representational models for resources
- Outlines an architectural model, not standardized
Resource-Oriented Architecture (ROA)

- Classic client-server communication model
- Stateless protocols
- Focuses on resources
- Assumes standardized access protocols
- Allows caching of resource representations, and layers of communicating entities (client, proxies, server)

RESTful Web Services

- Services seen as a collection of resources
- Defines:
  - resource set base URI
  - resource representation MIME type (nouns)
  - resource operation set (verbs)
- Service functionality semantics implied by resource
- Like SOAP, interface representation may differ from internal representation
- Also known as a RESTful web API

Design Principles

- Uniform identification of resources (URI)
- Manipulation of resources through representations
- Self-descriptive messages
- Hypermedia as engine of application state
- Freedom from choice in communication
- Freedom of choice in representation

CRUD Semantics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Collection URI</th>
<th>Member URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>List members</td>
<td>Retrieve representations</td>
</tr>
<tr>
<td>PUT</td>
<td>Replace collection</td>
<td>Update / Create resource</td>
</tr>
<tr>
<td>POST</td>
<td>Create entry</td>
<td>Add subordinate</td>
</tr>
<tr>
<td>DELETE</td>
<td>Delete collection</td>
<td>Delete resource</td>
</tr>
</tbody>
</table>

Design

- REST is an architectural style
  - not an API or a standard
- Client-server model
- Stateless interface
  - no need to keep track of sessions
  - all required information attached in call
- REST data format?
  - HTTP with attachments
  - XML
  - JSON
  - plain text
  - binary
  - ...

Interface Design

1. Determine nouns (resource types)
   - verbs (POST, GET, ...) are already defined
2. Define semantics for POST, PUT, GET, DELETE
   - to enable caching, GET should be side-effect free
3. Choose a data binding
4. Define (and document) what error codes apply
REST Advantages

- Simplicity
  - Stateless
  - Easy discovery through hyperlinks
  - HTTPS security
- Good tooling support
  - Only need HTTP (and typically XML)
  - Web browsers works as clients
- Good performance
  - Internet-scalability
  - GET may utilize caching

REST Disadvantages

- No formal (WSDL-like) interface
  - how to know what parameters to pass?
- No input/output data validation
- Only request-response
  - hard to implement one-way/asynchronous messaging
- Poor support for advanced functionality
  - transactions, Covt., reliable messaging, fine-grained / end-to-end security, message routing
- No standardized failure models
- May enforce tight coupling between clients and servers
- Often misunderstood and misused

SOAP

- Formerly known as Simple Object Access Protocol
- Originally designed as a communication protocol without symmetrical requirements, i.e., a protocol that didn’t require communicating nodes to use the same distributed object model
- XML-based protocol to invoke Web Services (XML-serializes Web Service requests / responses)
- Usually transported via HTTP (in HTTP body)
- Can send messages
  - point-to-point (directly to receiver)  
  - via intermediaries (via chains of actors)
- Typically uses request-reply messaging, but can use any communication pattern

SOAP Communication Stack

Service Descriptions

- Provides all information required to invoke a service
- Defines a type system for payload data
- Hosted by service registries
- Provides a mechanism to decouple services and clients
  - Interface abstraction
  - modularization
  - separation of concerns

Service Description Content

- Machine-understandable interface specification
- Details structure, operational characteristics, and non-functional properties of a service
- Describes wire format and transport protocols
- Payload data type system
- Metadata (optional)
  - e.g., behavioral data, policy descriptions
Web Service Description Language (WSDL)

- XML (Schema)-based language for describing Web Service interfaces
- Standardized by the World Wide Web Consortium (W3C)
  - XML Schema provided for WSDL validation
- Completely describes the Web Service interface
- Constitutes a "contract" between the client and the service
- Describes the what, where, and how of Web Services
  - what the service does (what functionality is offered)
  - where the service is located
  - how to invoke the service (data formats, protocols)
- Two major parts
  - abstract / interface (types, messages, interfaces)
  - concrete / implementation (bindings, services)

WSDL Elements

- <wsdl:portType name="HelloWorldService">
  - <wsdl:operation name="Greet"/>
  - <wsdl:message name="GreetRequestMessage">
    - <wsdl:part element="tns:GreetRequest" name="GreetRequest"/>
  - </wsdl:message>
  - <wsdl:message name="GreetResponseMessage">
    - <wsdl:part element="tns:GreetResponse" name="GreetResponse"/>
  - </wsdl:message>
- <wsdl:types>
  - <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
    - <xsd:element name="name" type="xsd:string"/>
    - <xsd:element name="greeting" type="GreetingType" minOccurs="1" maxOccurs="1"/>
  - </xsd:schema>
- <wsdl:fault message="tns:FailureFaultException" name="FailureFault"/>
- <wsdl:service name="HelloWorldService">... </wsdl:service>

WSDL Types

- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:complexType name="GreetingType">
    - <xsd:sequence>
      - <xsd:element name="name" type="xsd:string"/>
      - <xsd:element name="greeting" type="GreetingType" minOccurs="1" maxOccurs="1"/>
    - </xsd:sequence>
  - </xsd:complexType>
- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:element name="GreetRequest" type="tns:GreetRequestType"/>
  - <xsd:element name="GreetResponse" type="tns:GreetResponseType"/>
  - <xsd:element name="FailureFault" type="tns:FailureFaultType"/>
- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:complexType name="GreetingType">
    - <xsd:sequence>
      - <xsd:element name="name" type="xsd:string"/>
      - <xsd:element name="greeting" type="GreetingType" minOccurs="1" maxOccurs="1"/>
    - </xsd:sequence>
  - </xsd:complexType>
- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:element name="GreetRequest" type="tns:GreetRequestType"/>
  - <xsd:element name="GreetResponse" type="tns:GreetResponseType"/>
  - <xsd:element name="FailureFault" type="tns:FailureFaultType"/>
- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:complexType name="GreetingType">
    - <xsd:sequence>
      - <xsd:element name="name" type="xsd:string"/>
      - <xsd:element name="greeting" type="GreetingType" minOccurs="1" maxOccurs="1"/>
    - </xsd:sequence>
  - </xsd:complexType>
- <xsd:schema targetNamespace="http://cs.umu.se/edu/soa/HelloWorldService">
  - <xsd:element name="GreetRequest" type="tns:GreetRequestType"/>
  - <xsd:element name="GreetResponse" type="tns:GreetResponseType"/>
  - <xsd:element name="FailureFault" type="tns:FailureFaultType"/>
SOAP Web Services

REST Web Services

Technologies

Web Service

Today

Java Servlets

Protocol (HTTP)

Hypertext Transfer Notation (JSON)

JavaScript Object Language (XML)

Extensible Markup Language (XML)

WSDL Abstract-Concrete Connection

Web Service Description Languages (WSDL)

WSDL Services

WSDL Bindings

WSDL Services

GOAL: interoperability (favor the WSDL approach)

Two main approaches
- generate WSDL from code
- generate code (stubs) from WSDL

Generated WSDL tend to be platform / tool-dependent
(quick and easy, but incompatibility issues may arise)

Generating stubs from WSDL ensures compatibility
(but require more work from all parties involved)

1. Locate Web Service (discovery)
2. Obtain WSDL description
3. Generate message stubs from WSDL description
4. Use stubs to invoke Web Service methods
5. Use stubs to parse response

Web Service Development

Web Service Invocation (API)

1. Locate Web Service (discovery)
2. Obtain WSDL description
3. Create request message document
4. Use API to invoke Web Service methods
5. Parse response message document
SOAP Web Services

Communication Patterns

- One-way messaging
  - SOAP message
- Request/Response messaging
  - SOAP request
  - SOAP response
- Notify/Notification messaging
  - SOAP notification
- Solicitation messaging
  - SOAP request message
  - SOAP answer message

SOAP Terminology

- Method
  - HTTP message (request/response) that complies with the SOAP encoding rules
- Endpoint
  - HTTP-based URL identifying method invocation target
- Intermediary
  - SOAP node operating on a SOAP message
- Message path
  - the SOAP message route (node list)
    - includes sender, intermediaries, and endpoint.
- Fault
  - Detailed error message
  - Comparable to a Java exception

SOAP Messages

- Outer layer
  - technically not part of the SOAP message
    - e.g., HTTP, SMTP, RPC data (SOAP overhead)
- Envelope (message root element)
- Header (optional)
  - message delivery and processing metadata
    - e.g., factorization information, recipient list
    - extensions for security, transactions, QoS, etc, exist
- Body
  - application specific data (message payload)
    - e.g., XML elements, Faults (error messages)

SOAP Faults

- Faults reported in SOAP message body
- Error messages
- Comparable to Java exceptions
- Fault information
  - faultcode / Code: error identifier
  - faultstring / Reason: human readable identifier
  - faultactor: origin of error
  - detail / Details: additional fault information (optional)
SOAP Message Processing

- SOAP node may intercept, alter, and send messages
- SOAP intermediaries take on roles in message processing
- Roles are assigned in SOAP message headers
- Messages marked using the header attribute must be procured according to specification by all nodes, or a SOAP fault must be returned
- Message payloads in the SOAP body may also be altered or processed by any node in the intermediary chain
- Message bodies carry either data or faults, not both
- SOAP faults are propagated back to sender along the same path as the message was delivered

SOAP Communication Styles

- **RPC**
  - Utilizes automated serialization of a method invocation
  - Details serialization rules for method invocations
    - URI identifying transport address for the call is required
    - All parameters modeled as fields in a single struct
    - Parameter names and order must correlate to that of the method invoked
  - Enforces tightly coupled, bilateral interaction between client (requestor) and service
- **Document (Message)**
  - Represents arbitrary message serializations
  - Typically sends entire documents rather than discrete sets of parameters in message payloads
  - Assumes messages are well-formed XML documents
  - Results in loosely coupled, message-driven interactions

SOAP Encoding Styles

- **Literal**
  - XML messages encoded as-is
  - Recognized in WS-I Basic Profile (WS-Interoperability)
- **Encoded**
  - Messages encoded using proprietary encoding schemes
  - Prohibited in WS-I Basic Profile (WS-Interoperability)
  - Recognized in WS-I Basic Profile (WS-Interoperability)

Wire Protocols

- **Wire protocols**
  - Concerned with data format and payload structure
    - e.g., method parameter serialization, message interpretation, etc
- **Transport protocols**
  - Concerned with payload delivery
    - e.g., connection establishment, routing, package size, etc
- **SOAP** is a wire protocol in that it handles XML serialization and encoding of Web Service invocation data
- **SOAP** usually utilizes HTTP as a transport protocol

Wire Protocol Design Criteria

- **Compactness**
- **Protocol efficiency**
- **Coupling**
- **Scalability**
- **Interoperability**

A number of trade-offs between these criteria exist and typically results in that no wire protocol performs well on all criteria
SOAP Advantages

- Simplicity
- Portability
- Firewall-friendly
- Open standards
- Interoperability
- Acceptance
- Resilience to changes

SOAP Disadvantages

- Stateless
- Serialization by value
- Inefficient transport (HTTP)
- Protocol overhead
- Memory requirements
- Poor implementations
- Attack sensitivity
- Inefficient representations (binary data)

SOAP vs. REST

- Passionately discussed, polarized debate
  - Google "SOAP vs REST" > 150k hits
  - Similar to vi vs. emacs discussion
  - Some vendors support both
- Use cases
  - REST
    - Ad hoc Web style integration
    - Simplicity, usability
  - SOAP
    - Enterprise integration
    - Advanced capabilities (QoS, transactions, ...)
- Convergence?
  - WSDL 2.0 supports REST
- Read up, create your own opinion

SOAP Web Service Technology Stack