REST (Representational State Transfer) is an architectural style of distributed applications.

REST is not a standard; it is a set of principles on how to use Web standards, such as HTTP, URIs and MIME types.

The Web is a REST system.

REST is based on the following key principles:

- Give every relevant resource an ID; use URIs to identify everything that is an item of interest.
- URL: http://www.boeing.com/aircraft/747
- A representation of the resource is returned (e.g., Boeing747.html). The representation places the client application in a state.
- Link resources together; navigating links results in state transfers of the client application.
- Use standard methods, such as GET, POST, PUT, DELETE.
- Communication is stateless.
- Resources with multiple representations: a client may specify the formats which it accepts.

GET /customers/1234 HTTP/1.1
Accept: text/xml

An important issue is the choice of the appropriate granularity:

- Small vs. large Web services - thousands vs. a handful of Web services

What are the appropriate reusable, shared business components?

Composition of complex Web Services from simpler reusable Web Services

Dimensions to handle complexity

Web Service Orchestration
Web Services

Web services provide a standard means of communication among distributed software applications based on the Web technology. Standardization by the W3C community.

Motivation - Example

Service Oriented Architecture (SOA)

Web Services - Characteristics

Web Services Architecture

Simple Object Access Protocol (SOAP)

Web Services Description Language (WSDL)

Universal Description, Discovery, and Integration (UDDI)

REST

Web Service Composition

Adopting Web Services

Mashups

Example Web Services

The following lists some available Web services; often registration necessary in order to use them.

Amazon E-Commerce Service (EC2)

FedEx Office and Printing Service

printing of online documents and distribution of paper documents as commercial Web Service

free print plug-in for standard office application; plug-in added to list of printers

Pick up your document at any U.S. location or ship via FedEx for added convenience.

location independent use of printing service

ViaMichelin

Reverse Geocoding Web Service allows users to obtain the closest road segment (named or not) for each supplied geographic coordinates (WGS84).

XMethods - clearinghouse for Web Services
ECS provides access to Amazon's product database with the following types of data:
- Detailed product information.
- Customer-contributed content, e.g., wish list, product reviews.
- Seller information.
- 3rd party product information.
- Shopping cart contents.

ECS supports both SOAP and REST style interactions.

Product operations:
- ItemSearch performs a search for a specific item, typically using a set of keywords.

Other operations:
- ItemLookup: returns a list of similar products to a given product ID (based on product specifications and features).
- ItemGet: access to the data related to a specific product.

Remote shopping cart operations:
- CartCreate: creates a remote shopping cart.
- CartAdd: adds an item to the shopping cart.
- CartModify: modifies an item in the shopping cart.

Further operations are CartGet (obtains content of the cart), CartRemove (removes an item from cart).

Apache Axis supports an environment to implement and provide Web services.

It is a set of client-side APIs for dynamically invoking SOAP Web services (with or without WSDL descriptions).
It can translate WSDL documents into Java frameworks.
Mechanisms for hosting Web services either within a servlet container (e.g., Tomcat) or via standalone servers.
A set of APIs for manipulating SOAP envelopes, bodies, and headers, and using them inside Message objects.
Data binding which enables mapping of Java classes into XML schemas and vice versa.
A transport framework that allows usage of a variety of underlying transport mechanisms (e.g., JMS, email, etc.).

Axis2 is the latest version of the Apache Axis project.

Apache Axis supports an environment to implement and provide Web services.

Java supports a number of APIs implementing the Web Services standards:
- SAAJ (SOAP with Attachments API for Java)
  - SOAP messages as Java objects
- JAX-WS (Java API for XML based Web-Services)
  - Programming model for Web Services; replaces JAX-RPC
- JUWSDL: Accessing WSDL descriptions
- JAXR (Java API for XML Registries)
  - Accessing Web Services Registries, e.g., UDDI
- JAXP (Java API for XML Processing)
  - Abstract XML-API Specification implemented by e.g., Apache Xalan (XSLT), Apache Xerces2 (XML Parsing (DOM, SAX, ...))
- XWSS (Java Web-Services Security)
  - Signatures, Encryption (roughly for SOAP what SSL is for HTTP)

The following lists some available Web services; often registration necessary in order to use them:

Amazon E-Commerce Service (ECS)
- FedEx Office and Printing Service
  - Printing of online documents and distribution of paper documents as commercial Web Service
  - Free print plug-in for standard office application; plug-in added to list of printers
  - Pick up your document at any U.S. location or ship via FedEx for added convenience.
  - Location independent use of printing service
- ViaMichelin
  - Reverse Geocoding Web Service allows users to obtain the closest road segment (named or not) for each supplied geographic coordinates (WGS84).
- XMethods
  - Clearinghouse for Web Services
Adopting Web Services

There exist already a variety of free of commercial Web services; provided especially by Internet companies, such as Google, Amazon or Yahoo.

Example Web Services
- Apache Axis
- Web Services and Java
- Integration and WS Standards
- Supporting - Restraining Forces
- Distributed Process Architecture
- Semantic Web Services

Integration and WS Standards

<table>
<thead>
<tr>
<th>process management</th>
<th>meta data &amp; additional services</th>
</tr>
</thead>
<tbody>
<tr>
<td>process modeling, execution: BPEL4WS</td>
<td>system management: -</td>
</tr>
<tr>
<td>process control: -</td>
<td>security services: WS-Security, SAML, XML-Encryption, XML-Signature</td>
</tr>
<tr>
<td>user interface integration: WSUW/XSL</td>
<td>development support: -</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>message management</th>
<th>meta database: UDDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>transformation services: XSLT</td>
<td>-</td>
</tr>
<tr>
<td>synchronization: -</td>
<td>-</td>
</tr>
<tr>
<td>transaction services: WS-Coordination, WS Transactions</td>
<td>-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>adapter</th>
<th>middleware</th>
</tr>
</thead>
<tbody>
<tr>
<td>interface</td>
<td>interface description: WSDL</td>
</tr>
<tr>
<td>description: WSDL</td>
<td>reliability: WS Reliability</td>
</tr>
<tr>
<td>messaging: SOAP</td>
<td>Messaging: SOAP, XML</td>
</tr>
<tr>
<td>Transport: HTTP, SMTP, ...</td>
<td>Transport: IP, UDP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>physical network</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>transport layer: TCP, UDP</td>
<td>adapter</td>
</tr>
<tr>
<td>network layer: IP</td>
<td>adapter</td>
</tr>
</tbody>
</table>

Supporting - Restraining Forces

The adoption of Web Services in an organization depends on

Supporting Forces
- interoperable networked applications, i.e. independence of hardware, operating system, application server, ...
- easier exchange of distributed data
- easier access of enterprise wide data
- availability of external services, encapsulation of legacy applications
- cross-organizational computing
- reduced maintenance cost, easier reuse of components
- emerging industry-wide standard

Restraining Forces
- different formats and semantics of data sources
- security issues due to network access
- standards are evolving and not fixed yet
- lack of understanding of effects on operational systems
Semantic Web Services

In order to allow for automatic discovery of appropriate web services and automatic interaction/chainling/incorporation with web services, we need semantic meta-data for web services: Web-Service Ontologies, DataTypes with rich semantics,...

Example: Map-Service
Input: (int, int)
Output: APPLICATION/GIF
Input: (int, int):
  (x,y) of center of map?
  of center of map? Which corner?
  what coordinate system? Wgs84? Gauss-Krueger? ...
Output: APPLICATION/GIF:
  Units of measure?
  candidate technology: **OWL-S** (Ontology Web Language for Web Services)

OWL-based Web service ontology, which supplies Web service providers with a core set of constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form.

1 Mashing on the Web Server

All the work of mashing is done on a Web server while the browser just waits for a response.

Characteristics
- Browser is decoupled from the partner sites supplying the data.
- Web server acts as a proxy and aggregator for the responses.
- Browser requests the entire page.
- Scalability problem because server does all the work.

Mashups

Definition: Mashup simply indicates a way to create new Web applications by combining existing Web resources utilizing data and Web APIs.

Mashup Techniques
- Work for the combination of data and services can be done on the server, the client or both of them.
- Mashing on the Web Server
- Mashing using Ajax
- Mashing with JSON

Development Support
- Yahoo Pipes are hosted and executed on a Yahoo server.
- QEDWiki was a Wiki-based mashup maker by IBM; pages are hosted on an IBM server; mostly executed on the client side.
- **ProgrammableWeb** provides a mashup directory and marketplace which let users rank and discuss mashups.

This approach allows a richer user experience; the work is divided between the server and the browser.

Characteristics
- more complex because developers face JavaScript challenges, server communication and asynchronicity.
- Ajax may refresh only a portion of the page.
- navigation mechanism of browser is bypassed.
- approach may result in a rich Internet application.
2 Mashing using Ajax

Characteristics
- more complex because developers face JavaScript challenges, server communication and asynchronicity.
- Ajax may refresh only a portion of the page.
- navigation mechanism of browser is bypassed.
- approach may result in a rich Internet application.
- presentation of results is driven by XSLT style sheet.
- browser is doing most of the work.
- all data are routed through a common point on the server.

Development Support

in order to facilitate and speed up mashup development a number of tools and frameworks have recently emerged. Two dimensions may be distinguished

**component model**: describes the characteristic properties of the mashup components
- a well-defined component interface facilitates reusability of components
- component properties:
  - type: a component can be data, application logic or user interface.
  - interface: create-read-update-delete (CRUD) interface, API for a specific programming language or IDL/WSIDL
  - extensibility: whether the user may extend the component model.

**composition model**: specifies how the components are glued together to create the mashup application
- flow-based: defines the orchestration as sequencing or partial order among components.
- event-based: uses the publish-subscribe model.

Examples for tool-assisted mashup development
- **Yahoo Pipes**: mix data feeds to create data mashups using a visual editor.

Mashups

Definition: Mashup simply indicates a way to create new Web applications by combining existing Web resources utilizing data and Web APIs.

Mashup Techniques

Work for the combination of data and services can be done on the server, the client or both of them.

1. Mashing on the Web Server
2. Mashing using Ajax
3. Mashing with JSON

Development Support

Yahoo pipes are hosted and executed on a Yahoo server.

CloudWiki was a Wiki-based mashup maker by IBM; pages are hosted on an IBM server, mostly executed on the client side.

**ProgrammableWeb** provides a mashup directory and marketplace which lets users rank and discuss mashups.
Software engineering of distributed applications raises interesting issues. In particular, the following problems must be considered:

1. Specification of a suitable software structure
   - Applications must be decomposed into smaller, distributable components; encapsulation of data and functions.
   - Which functionality is provided locally and which remotely?
   - How should we test and debug distributed applications?

2. Mechanisms for name resolution
   - How can an application localize and make use of a remotely provided service?
   - Assignment of names to addresses.
   - What should happen if a client cannot contact the localized server subsystem?

3. Communication mechanisms
   - Selection of the desired communication model, e.g., client-server model, group communication or peer-to-peer.
   - How does the application (both client and server) handle network communication errors?

4. Consistency
   - How can the data be kept consistent, particularly for replicated data?

5. User requirements
   - Functionality and reconfigurability of the distributed application and its components.
   - Service quality, such as security, reliability, fault tolerance and performance.
   - What kind of security mechanisms are provided? Is authentication an issue?
   - What actions will be triggered if a client cannot communicate with its server?
   - What type of heterogeneous system is necessary?
   - What efficiency (performance) is expected?
Design of distributed applications

Issues
Steps in the design of distributed applications
Design - Development environment
Service-Oriented Modeling