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

REST is not a standard; it is a set of principles how to use Web standards, such as HTTP, URLs and Mime Types.

The Web is a REST system.

REST is based on the following key principles:
- give every relevant resource an ID; use URLs to identify everything that is any item of interest.
- 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: client may specify the formats which it accepts.

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

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
Web Service Composition

Composition of complex Web Services from smaller reusable Web Services

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

Dimensions to handle complexity
Web Service Orchestration

Web Service Orchestration

Web Services

Dimensions to handle complexity

component model: defines the sub-services.
orchestration model: defines the order in which the sub-services are invoked.
WS-Coordination is an extensible framework that describes how different Web Services work together reliably. Coordination framework contains

Activation, Registration and Coordination services
data access model: specifies the data exchange between the sub-services.
transactional model: transactional semantics of the composed service.
WS-Transaction specifies the protocols for each coordination type (used by WS-Coordination)

Atomic Transactions: all-or-nothing property, 2-phase-commit.
Business Activity: handle long-lived activities and to apply business logic to handle business exceptions; Business Agreement Protocol.

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

a set of client-side APIs for dynamically invoking SOAP Web services (with or without WSDL descriptions), tools to translate WSDL documents into Java frameworks, mechanisms for hosting Web services either within a servlet container (e.g. Tomcat) or via standalone server, a set of APIs for manipulating SOAP envelopes, bodies, and headers, and using them inside Message objects, a transport framework that allows usage of a variety of underlying transport mechanisms (e.g. JMS, SMTP, etc).

**Axis2**

In the meantime there exists already Apache **Axis2**.

Java-based implementation of both the client and server sides of the Web services.

Axis2 is more flexible, efficient and configurable in comparison to Axis1.x.

Axis2 not only supports SOAP messages, but it also supports RESTful Web services.

**Web Services and Java**

Java provides 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
- **JJWSDL**: 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-Service Security)
  - Signature, Encryption (roughly for SOAP what SSL is for HTTP)

**Semantic Web Services**

In order to allow for automatic discovery of appropriate web services and of automatic interaction / chaining / 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

- (x,y) of center of map?
- of corner of map? which corner?
- what coordinate system? Wgs84? Gauss-Krüger? ...

**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.
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**

- Amazon E-Commerce Service (ECS)
- Apache Axis
- Web Services and Java
- Distributed Process Architecture
- Semantic Web Services

---

**Adopting Web Services**

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.

2. **Mashing using Ajax**

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.

**Mashups**

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

**Mashup Techniques**

- **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.
Characteristics

- the browser communicates directly with the partner site.
- programmers must handle pre-made objects supplied in JSON.
- JSON objects are easier to read than XML.
- there is no data consolidation on the server.

Mashups

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

Mashup Techniques

- Mashing on the Web Server
- Mashing using Ajax
- Mashing with JSON

Development Support

Yahoo Pipes are hosted and executed on a Yahoo server.

GeoWiki 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.
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?
   - Which actions will be triggered if a client cannot communicate with its server?
   - What type of heterogeneity is necessary?
   - What efficiency (performance) is expected?

---

3. Communication mechanisms
   Assignment of desired communication models, 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?

   If a cache is used for performance improvement, then it must be kept consistent with the stored data.

   User interface consistency for the individual components.

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?
   - Which actions will be triggered if a client cannot communicate with its server?
   - What type of heterogeneity is necessary?
   - What efficiency (performance) is expected?

---

Issues

Steps in the design of distributed applications
Design - Development environment
Service-Oriented Modeling
Open Distributed Processing (ODP)

1. Step: development of PIM

PIM models the functionality and behavior of software systems
specifying components, classes, pre-/post conditions, semantics
with UML (Unified Modeling Language) to model information in diagrams.
- use case diagrams
- class and component diagrams
- sequence diagrams
- state diagrams

2. Step: Development Environment

Design - Development environment

3. Step: code generation

generation of specific technological constructs, e.g., Java packages
implementation of system functionality
use of tools for automatic code generation

Platform independent model (PIM)

- PSM: web service model
  - E.g., WSDL, Java constructs

- PSM: Java/EJB model
  - E.g., Java, EJB constructs

- PSM: CORBA model
  - E.g., IDL/C++ constructs

- PSM: ...

Use of Software Engineering concepts, methods and tools to design and develop distributed applications
software development cycle is divided into phases
requirements analysis, specification, design, implementation, test and integration, maintenance
for details see Software Engineering courses

Open Distributed Processing (ODP)
Model Driven Architecture (MDA)
AutoFocus
AutoFocus is a platform to specify distributed systems developed by the group of Prof. Broy, TU München based on formal methods of systems engineering. It integrates hierarchical description techniques allowing distributed and platform independent development.

Project advances include AutoFocus 2, supporting the following functionality:
- Requirement analysis tool (AutoRAID), such as use-cases and scenarios, business and application requirements.
- Design modelling views and editors, such as system structure diagrams, state transition diagrams, message sequence charts.
- Interactive simulation environment, code generation, consistency maintenance support.

Generated by Texpoint.