Introduction

Mastering Complexity in the Future Internet

Top-down approach:

- Process-driven (incl. service composition) and MDD approaches to master complexity and enterprise-scale change:
  - model/build once -> use many times by many service consumers

Bottom-up approach:

- User-driven composition, Mashup approach for small-scale:
  - build once -> use once

Some Service-oriented Approaches

Jini

Jini is a technology for building service oriented architectures

§ Written in Java

§ Uses RMI and Java Object Serialization

§ Offers network plug and play of services (java objects)

OSGi (Open Service Gateway Initiative)

A Java framework for developing (remotely) deployed service applications

§ Portable byte code (independent of OS or CPU architecture)

§ Security integrated in the language

§ Excellent model for the myriad of customizations and variations that are required for today's devices

UPNP (Universal Plug and Play)

supports Devices and Control Points

§ A device may have multiple Services (e.g. TV ControlService, PictureService)

Enterprise Application Integration (EAI)

§ One of the main challenges in IT

§ Applications shall be integrated to work together

§ Communication via messages (neutral message format)
=> Message Oriented Middleware (MOM)

§ passing of data between applications using a common communication channel that carries self-contained messages

§ messages sent and received asynchronously

Integration Brokers are used to

§ transform,

§ store & route messages (point-to-point or publish/subscribe)

§ apply business rules &

§ respond to events.

Web Services vs. EAI? (http://www.ebizq.net/topics/eai/features/1555.html)

§ Web services are made up of a set of standard

§ EAI groups together a set of methods, technologies and tools (not exclusively based on standards)

§ Web services and EAI are two fully complementary notions: each enriches the other, but they are not mutually exclusive, since Web services can be seen as a technical means of implementing loosely coupled EAI.

Service-Oriented Computing (SoC) and (Web) Services

§ Web services: self-contained Software Entities which are published, discovered, and invoked on the Internet. XML-based languages -> LOOSE COUPLING OF SYSTEMS

§ Virtualization of Resources

§ Agile development through service composition

What is a Service?

§ Standardized interface

§ Self-contained with no dependencies to other services

§ Little integration need

§ Coarse-grained (complex service) or fine-grained (simple service)

§ Context-independent (-> loosely coupled)

§ Can be stateful or stateless

§ Quality of Service(QoS)Attributes which can be measured

Software Services vs Real-world services

§ Software Services equivalent to real-world services

§ Software services should align with business functions/real-world services

§ Service properties apply to software services too

What is SOA?

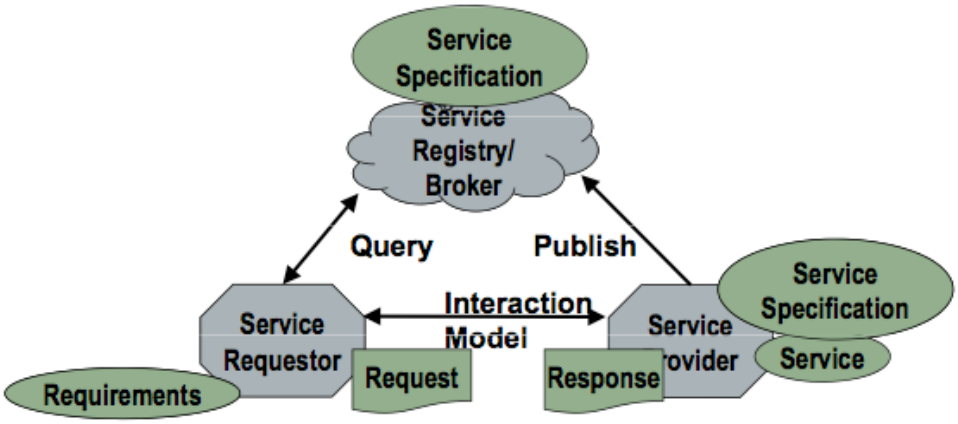
§ Architectural style of software design

§ Guides all aspects of creating and using services

§ Mainparadigms: loose coupling, dynamic, binding, high interoperability

§ Basic Architecture: publish/find/bind

Roles in SOA:



Types of (Web) Services:

§ Informational services: services of relatively simple nature (provide content or expose back-end business applications)

§ Complex service: involve the assembly & invocation of many preexisting services possibly found in diverse enterprises to complete a multi-step business interaction

Two programming styles for services

§ synchronous or remote procedure call (RPC)-style: method call with a set of arguments (Simple informational services, e.g., returning the current price for a given stock)

§ asynchronous or message (document)-style: typically sends an entire document (Business processes, e.g., a purchase order)

Well-definedness of web services:

The Web Services Description Language (WSDL) allows applications to describe to other applications the rules for interfacing and interacting

Service interface:

§ defines service functionality visible to the external world and provides the means to access this functionality (operations available, the parameters, data-typing and the access protocols)

§ realized by service implementation (using any programming language - hidden to service consumer)

Service Level Agreement (SLA):

formal agreement (contract) between a provider and client formalizing the details of use of a Web service

Technical Benefits of Services:

§ Efficient development

§ More reuse

§ Simplified maintenance

§ Incremental adoption (allows step-by-step migration)

Business Benefits of Services:

§ Agility

§ Reduced integration costs (loose coupling, platform independence)

§ Reduced dependency on technolog and vendors

WEB SERVICES

§ ***ONE*** possible implementation technology for SOA (abstract architectural concept -> Web Service - one approach to realization)

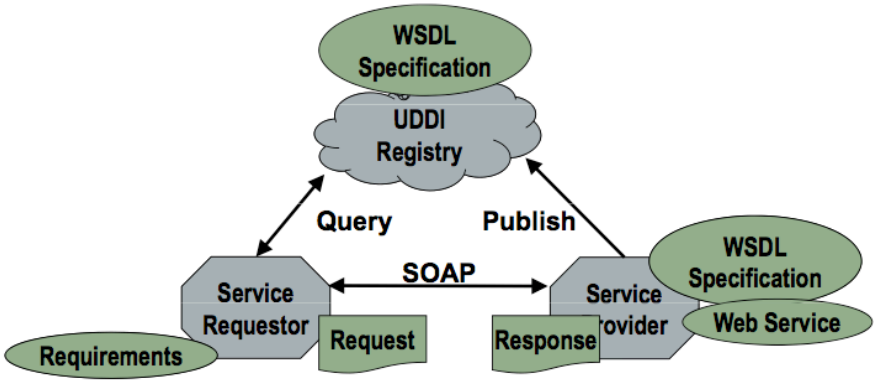
§ machine-to-machine interaction

§ interface described in machine-processable format (WSDL)

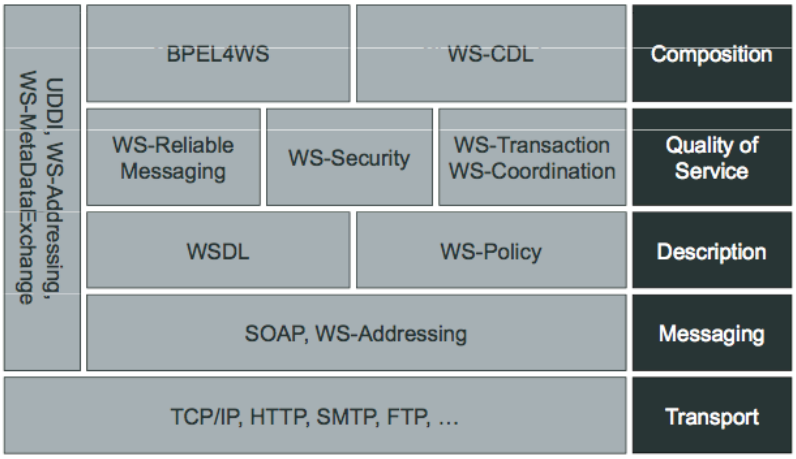
§ interaction using SOAP-mesages (conveyed using HTTP with XML serialization)

§ Core standards (SOAP, WSDL, UDDI) describe basic parts of Web service platform

Core Web Service Architecture:



Web Service Stack (Framework):



SOAP (Simple Object Access Protocol) – see chapter „SOAP“:

XML-based messaging protocol to exchange messages between computers (using HTTP)

WSDL (Web Services Description Language) – see chapter „WSDL“:

§ XML vocabulary to describe Web services

UDDI (Universal Description, Discovery and Integration) – see chapter „Metadata/UDDI“:

§ Flexible directory/registry service for Web services

Summary:

Web Services are self-contained, modular applications that can be

§ Described: WSDL

§ Published: to UDDI

§ Found: in UDDI

§ Bound: using SOAP

§ Invoked: using SOAP

§ Composed: Orchestration (e.g. BPEL)

Extended Specifications:

§ WS-Addressing

§ Interoperable, transport-independent way of identifying message senders and receivers

§ WS-Policy

§ Define constraints, conditions, service-level assurances and requirements
§ Attach these policies to WS-PolicyAttachment

§ various others: WS-MetaDataExchange, WS-Security, WS-Reliable Messaging, WS-Coordination, WS-Transaction, WS-CDL

§ BPEL (Business Process Execution Language for Web Services)

§ Provides orchestration for Web services
§ Definition and execution of business processes
§ Allows the recursive creation of larger Web services from smaller ones

SOAP (Simple Object Access Protocol)

§ Simple enveloping mechanism

§ Processing model for messages

§ Optional data model and encoding

§ Extensibility scheme

§ Binding mechanism for transport protocols

§ Attachment of non-XML encoded information

To address the problem of **overcoming** proprietary systems running on **heterogeneous infrastructures**, Web services rely on SOAP, an **XML-based communication protocol for exchanging messages** between computers **regardless of their operating systems or programming environment**

Uses XML as an encoding scheme for request and response parameters

Uses HTTP as a means for transport

⇨ standard messaging protocol used by Web services

SOAP covers the following four main areas

§ A **message format** for one-way communication describing how a message can be packed into an XML document

§ A **description** of how a SOAP message should be transported using HTTP (for Web based interaction) or SMTP (for e-mail based interaction)

§ A **set of rules** that must be followed when processing a SOAP message and a simple classification of the entities involved in processing a SOAP message.

§ **A set of conventions** on how to turn an RPC call into a SOAP message and back.

Message Format

Optional header

- Specifies additional handling
- Used by extension protocols,
  e.g. WS-ReliableMessaging, WS-SecuritySOAP Envelope

Mandatory body

- Message payload or business information

```
<env:Envelope
    xmlns:env="http://www.w3.org/2003/05/soap-envelope">

    <env:Header>
        <h:SomeHeader xmlns:h="Some-URI">
            <h:Value>2</h:Value>
        </h:SomeHeader>
    </env:Header>                                    Header

    <env:Body>
        <f:CallFunction xmlns:f="Some-URI">
            <f:Parameter>5</f:Parameter>
        </f:CallFunction>
    </env:Body>                                      Body

</env:Envelope>
```

Nodes and Roles

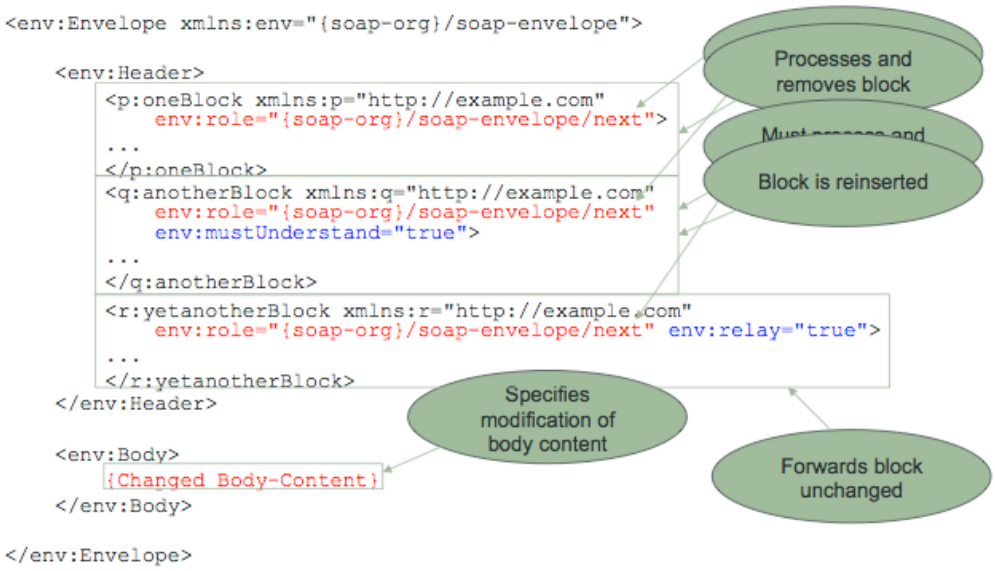
Nodes send and/or receive SOAP messages
Three Types: § Initial sender § Intermediaries § Ultimate receiver

A role is an URI defining what parts of a message a node processes

Node can act in different roles

Message Processing

- Header block may target node by specifying role
  - No role specified → targeted at ultimate receiver
- Node may process header block if targeted
  - Attribute „mustUnderstand“ → block must be processed
  - Block targeted at node must be removed
    - May be reinserted (unchanged/modified) if processed
  - Attribute „relay“ → must be forwarded if not processed
- Body is always targeted at ultimate receiver, but may be changed by intermediaries



Interaction Styles

Synchronicity of request and response does not depend on interaction style!

Document literal

- Body contains business document
- Response (if any) contains another document

Example (SOAP message containing PurchaseOrder document):

```
<env:Envelope xmlns:env="{soap-org}/soap-envelope">
    <env:Body>
        <m:PurchaseOrder xmlns:m="http://example.com">
            <m:Item>XYZ 1138</m:Item>
            <m:Quantity>42</m:Quantity>
        </m:PurchaseOrder>
    </env:Body>
</env:Envelope>
```

Remote procedure call

- Body contains procedure call (name, input parameters)
- Response contains return value, output parameters

Example (SOAP message for call to orderGoods method):

```
orderGoods(String productItem, Number quantity)

<env:Envelope xmlns:env="{soap-org}/soap-envelope">
    <env:Body>
        <m:orderGoods
              env:encodingStyle="{soap-org}/soap-encoding"
              xmlns:m="http://example.com">
            <m:productItem>XYZ 1138</m:productItem>
            <m:quantity>42</m:quantity>
        </m:orderGoods>
    </env:Body>
</env:Envelope>
```

Extensibility

Definition of **features** (e.g. reliability, security, message exchange patterns)

> SOAP provides one-way communication => more advanced message exchange patterns (MEP) can be specified as feature

Binding: how to pass SOAP messages using an underlying protocol (valid for a single hop between nodes) -> HTTP binding uses URI adressing

SOAP Attachments: sending binary data Base64 encoded (en-/decoding is time consuming!)

> **=> SOAP MTOM and XOP**
>
> MOTM (Message Transmission Optimization Mechanism) specifies an abstract feature for optimizing Base64-encoded data
>
> XOP (XML-binary Optimized Packaging) specifies use of MIME for binary parts

Advantages/Disadvantages

PRO

- SOAP hides implementation technolgy
- XML-based
- Platform independent
- Relatively simple
- W3C Standard
- Lots of vendor support

CON

- Too much reliance on HTTP
- Statelessness
- Serialization by value and not by reference

WSDL (Web Service Description Language)

- XML vocabulary to describe Web services
- Highly extensible and adaptable
- Two parts:
  - Abstract: operational behavior ("what?")
  - Concrete: binding, service ("how?", "where?")

Scenarios

- Service description for clients
  - Describes published service
  - Abstract and concrete parts
- Description of standard service for implementers
  - Describes standard for service
  - Abstract part only

Concepts

- Extensibility
- Multiple Type Systems
- Messaging and RPC
- Separation abstract – concrete parts
- Multiple protocols and transports
- No ordering
- No semantics

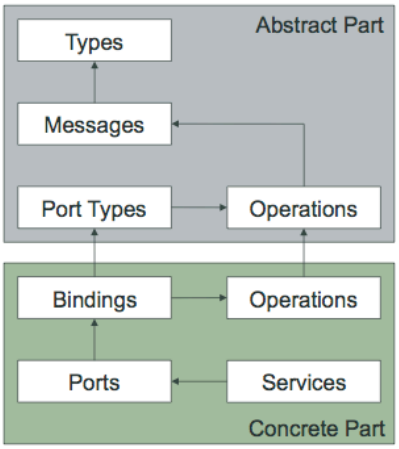
Problems and Limitations of WSDL 1.1

- Messages
  - No variable number of items
  - No choice of alternative message parts
  - Original idea: use also other type systems in addition to XML Schema
    → Functionality equivalent to XML Schema required
- SOAP binding
  - Operation styles and encodings problematic
  - Original idea: bridge message-oriented and RPC-oriented descriptions
- Services
  - Lack clarity w.r.t. granularity (coarse/fine grained)
  - No guidelines for grouping → interoperability problems

§ **Solution: WSDL 2.0**

WSDL 1.1 Language Structure

- Abstract Part
  - Operational behavior ("What?")
  - Plays role of interface definition language
- Concrete Part
  - Bindings ("How?")
    - Mapping of abstract descriptions to concrete protocols
  - Services ("Where?")
    - Locations of service providers



- Types
  - Contains data type definitions
  - Normally XML Schema
  - Extensibility may be used to support other type systems
- Message
  - Contains multiple parts
  - Each part is associated with a type
  - Example: RPC → parts are parameters
- Operation
  - Set of abstract messages
  - Four transmission primitives:
    - One-way
    - Request-response
    - Solicit-response
    - Notification
  - Predefined bindings only support the first two

- Binding
  - Defines message format and protocol details for a port type
- Operation
  - Binding information for corresponding operation in port type
- Port
  - Individual endpoint
  - Single address

- Port type
  - Named set of abstract operations

Abstract ->

<- Concrete

- Service
  - Group of ports
  - Do not communicate
  - Ports are alternatives
    - Protocol
    - Distance
    - …

WSDL 2.0

Simpler to use

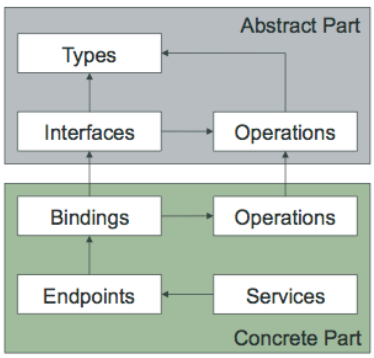
Better specified, more additional features

§ Main differences:

- § Elimination of message construct
- § More Message exchange patterns
- § Interface extensions
- § Flexible include/import concept
- § Features and properties

WSDL 2.0 Language Structure

- Syntactic changes:
  - Root: `description` instead of `definitions`
  - No message construct
  - Interfaces replace port types
  - Endpoints replace ports



Web Service Composition

...creating new processes/applications

- Combine and link existing Web services
- Services to be combined can be:
  - Atomic
  - Composed (recursive composition)

Types of composition

Static: services to be composed are decided at design time

Dynamic: services to be composed are decided at runtime

Orchestration vs. Choreography

Often used interchangeably, overlap somewhat

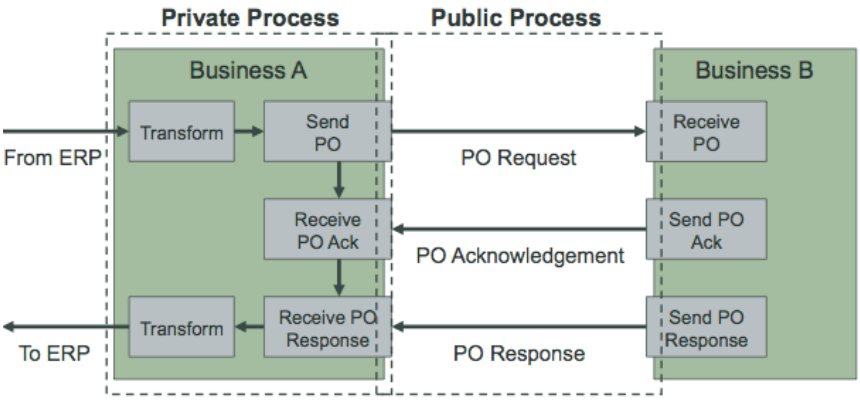
Orchestration

- Compose Web services for business processes
- Define composite services
- Reuse of existing Web services
- Composition in the "part-of" sense

Choreography

- Compose Web services for business collaboration
- Peer-to-peer model
- Define how multiple parties collaborate
- Composition in the "sequencing" sense

Difference:



Choreography – The observable public exchange of messages
Orchestration – A private executable business process

Approaches

Process-based

- Compositions are treated like Workflows

Requirements driven

- Compositions are generated out of detailed requirement specifications' documents

AI-based

- Compositions are defined through “reasoning-engines”

Protocols

Orchestration (BPEL)

- Centralized cooperation of services

Choreography (WS-CDL)

- Decentralized federation of services

Event-based federation (WS-Eventing)

- Event-based approach applied to SOA

WS-BPEL (Business Process Execution Language)

- Process-oriented composition language
- Definition of concrete and abstract processes
- Mainly intended for orchestration
- Relies on WSDL
  - Process can be exposed as WSDL-defined service
  - Included services expected to be defined as WSDL port types
  - BPEL processes interact with the services through the WSDLs they expose

Basic component: **activity**

- Primitive activities
- Structured activities
  - Prescribe the order in which a set of activities is executed

Partners

- Actors that are external to the process and with which the process needs to interact
- They can offer to and use services offered by the process
- They are defined by their services
  - Specified through WSDL

Types of Processes

Abstract processes

- Describe externally visible interactions
- Do not necessarily expose internal business logic
- Used for external interface for business partners e.g.

Concrete/executable processes

- External protocol and internal business logic
- Implementation of business processes

WS-CDL (Choreography Description Language)

- Declarative language for defining interaction patterns (not executable)
- Specifies interactions in B2B scenarios
- Intended to complement WS-BPEL with global definitions for message exchange

Transactions

- Mechanism for ensuring that all participants in an application achieve a mutually agreed outcome
- Fundamental concept in distributed systems
- Two-phase commit (2PC)
  - Phase 1: prepare
  - Phase 2: commit
- Requires locking of resources

Transaction Properties (ACID)

- Atomicity
  - Transaction completes successfully: all actions happen
  - Transaction completes unsuccessfully: no actions happen
- Consistency
  - Consistent results are produced
  - Correct transformation of application states at completion
- Isolation
  - Intermediate states invisible to other transactions
- Durability
  - Changes are maintained after successful completion

Transactions for Web services

- Needed for transactions spanning multiple execution environments

- Additional difficulties:
  - Loose coupling
  - Distributed across independent systems
  - Heterogeneity
  - Long runtimes

  => ACID properties not always possible
  => Coordination required

Long-Running Transactions

- Composed of multiple short-time sub-activities

- Sub-activities are committed on completion

- In case of an error (rollback):
  - Previously committed sub-activities are **compensated**

- Compensation is application-specific

- **Limitations**
  - Does not guarantee isolation
    - Effects of committed sub-activities are visible to other applications
  - Handling of errors during compensation

Coordination

Participants - A contributor to an activity

Final outcome of an activity must be consistently agreed to between all of its participants
  - Coordination types & -protocols are used for this

Agreement on the outcome is mediated by a **Coordinator**

Protocol defines exchange of messages (including their order) between a participant and coordinator

WS-Coordination

Defines protocols and services for:
  - **Activating** coordination
  - Providing a **context** to identify operations as part of an activity
  - Allowing **registration** of interest in participating in the activity outcome
  - Selecting a coordination protocol to be performed at **completion** of the activity

WS-AtomicTransaction

§ Handles short-lived activities

§ Satisfies ACID properties

§ Application initiates completion by calling Commit or Rollback on the coordination service

§ Coordinator performs **2-phase commit protocol**
    § Sends Prepare to all participants
    § If all answer with Prepared, sends Commit
    § If one participant answers with Aborted, sends Rollback to all others
    § Expects Committed or Aborted from all participants

WS-BusinessActivity

§ Handles long-lived activities

§ Participant must be able to compensate activity

§ Once finished, participant sends *Completed* to coordinator
    § If complete activity is completed, coordinator sends *Close* to participants, which answer with *Closed*
    § On error, coordinator sends *Compensate*, answered by *Compensated*

§ Can be combined with Atomic Transactions (e.g. busines activity includes several atomic transactions)

Other Web-Standards (WS-I, REST)

WS-I (Web Service Interoperability)

„Meta“ Web Standards

Main goals:
    § Clarifying ambiguities in existing standards
    § Define best practices
    § ‘Ban’ certain aspects which are known to cause troubles

WS-I Basic Profile (WS-I BP):

§ Main outcome of WS-I

§ Small subset of WS features, which are expected to be interoperable among any (compliant) platform and WS middleware

WS-I Basic Security Profile (WS-I BSP)

§ Similar to WS-BP but focuses on interoperability of Security solutions

REST (Representational State Transfer)

...basic architecture of the WWW and an architectural style for distributed systems

- Resource-Orientation
  - Key elements of any RESTful system are resources
  - Activities are not explicitly modelled

    => Resources are represented in a MIME type (e.g., text/html, text/plain, image/jpeg, ...)

- Statelessness
  - Every request is self-contained

    => HTTP

- Uniform Interface
  - Every resource is accessed through the same interface

    => HTTP interface (GET / POST / PUT / HEAD / DELETE)

- Naming
  - Every resource is associated with an unique and descriptive name

    => Every resource is identified by a URI

    => URIs should be descriptive

- Layering
  - Intermediaries can be inserted transparently

    => HTTP works over caches, proxies, gateways, routers...

From REST to RESTful Web Services

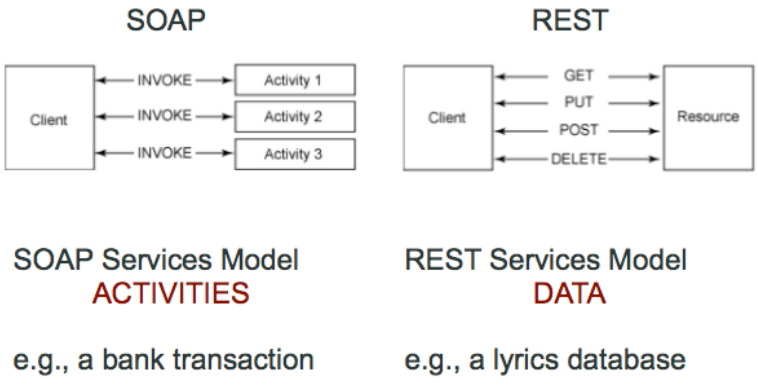
Create Web services that are in line with the Web => Create Web services according to the REST principles

XML

...prevalently used for resource representation

- Other possibilities:
  - JSON (in conjunction with AJAX)
  - HTML
  - Plain text
  - ...

SOAP vs REST



SOAP **vs** **REST**

Main focus:
- § SOA
- § EAI

§ Supports different protocol bindings

§ WS-* stack supports enterprise features

§ Cover everything (and introduce extensibility to cover everything else)

§ Main focus:
- § Web 2.0

§ Simple usage, ad-hoc

§ Light-weight

§ Let engineers figure out thdetails on demand

Mashups

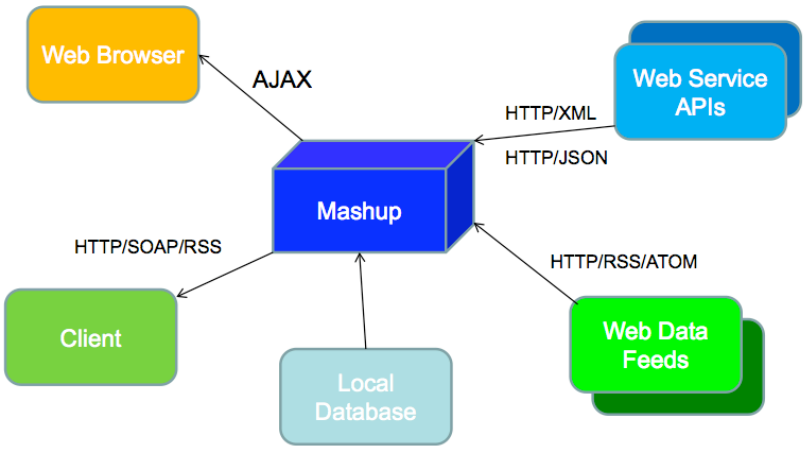
§ Aggregate content from more than one source
- § Public Web Service APIs (flickr.com, maps.google.com, …)
- § Data feeds from other providers (google search, news feeds, …)
- § User-provided information (wikipedia)

§ Lightweight programming effort
- § Numerous toolkits (mostly based on JavaScript and HTML)

§ Interactive Web Application

§ Ad-hoc composition

Architecture



JSON (JavaScript Object Notation)

§ isa lightweight data-interchangeformat

§ easy for humans to read and write § easy for machines to parse and generate

§ Based on a subset of the JavaScript Programming Language

§ A collection of name/value pairs OR an ordered list of values

RSS (Rich Site Summary)

a family of Web feed formats used to publish frequently updated works(blogs, news, headlines, audio and video)

ATOM

Alternative to RSS, because RSS had to remain backward compatible -> advantage in fresh design

Service Mashups

Widespread use of RESTful Web services

§ Strongly influences traditional business process environments

§ Evolve from general Services like Google Search to specialized Services like Twitter, Facebook or Amazon Web services

Metadata and Discovery

Web Service Metadata

§ Data types and structures for messages

§ Message exchange patterns

§ Addressing information for endpoints

§ Required extended features (security, reliability, transactions, etc.)

§ Quality of service attributes

Web Service Metadata Technologies

§ XML Schema
    § Defining data types

§ WSDL
    § Defining messages, message exchange patterns, interfaces, endpoints

§ WS-Addressing
    § Defining Web service endpoint references

§ WS-Policy
    § Declaring assertions for quality of service requirements (reliability, security, transactions, etc.)

§ UDDI
    § Registry/repository for storing/retrieving metadata

§ WS-MetadataExchange
    § Dynamic exchange of metadata

UDDI (Universal Description Discovery and Integration)

§ Flexible directory service/registry for Web services

§ Services described using WSDL and accessed using SOAP

§ Original vision: public directory (**UBR** – Universal Business Registry)

§ Companies register provided Web services

§ Other companies dynamically discover and use these

§ Not (yet) fully realized -> **Meanwhile: intra-enterprise directories**
Most Web services used are internal or shared between business partners / UDDI successful as private/semi-private registry

Changes in V3:

§ Improved security support
§ Support for public and private registries

WS-Addressing

...Addressing mechanism for Web services

Correct delivery to appropriate destination/service endpoint required
=> SOAP does not specify addressing mechanism

§ Specifies:
§ Structure and contents of endpoint references
§ SOAP headers used for encoding addressing information

§ Transport protocol independent

Endpoint References

§ Maps to at most one WSDL port
§ Multiple endpoints can map to the same port

WS-Policy

§ WSDL: functional description
§ Policies: nonfunctional service behavior (e.g. QoS attributes)

=> Changes to policy do not require change of WSDL description

Basic Structure: Assertion (expresses a single behavior)

Vocabulary is the set of all assertions (thus the behavior of the policy)

Capabilities:

§ Grammar for expressing alternatives and composition
§ Merging of multiple policies (= merging of vocabularies)
§ Intersection of policies (determine compatibility between policies)

Policies can be **reused** since they are completely separated from subject.

WS-MetadataExchange

- WSDL interface for exchanging metadata
- designed to support extensibilty and redirection
- supports different metadata dialects

Research Challenges

Self-Healing, Trust and Reputation, Service Management, Service Engineering, ...