Telecommunication Services Engineering (TSE) Lab

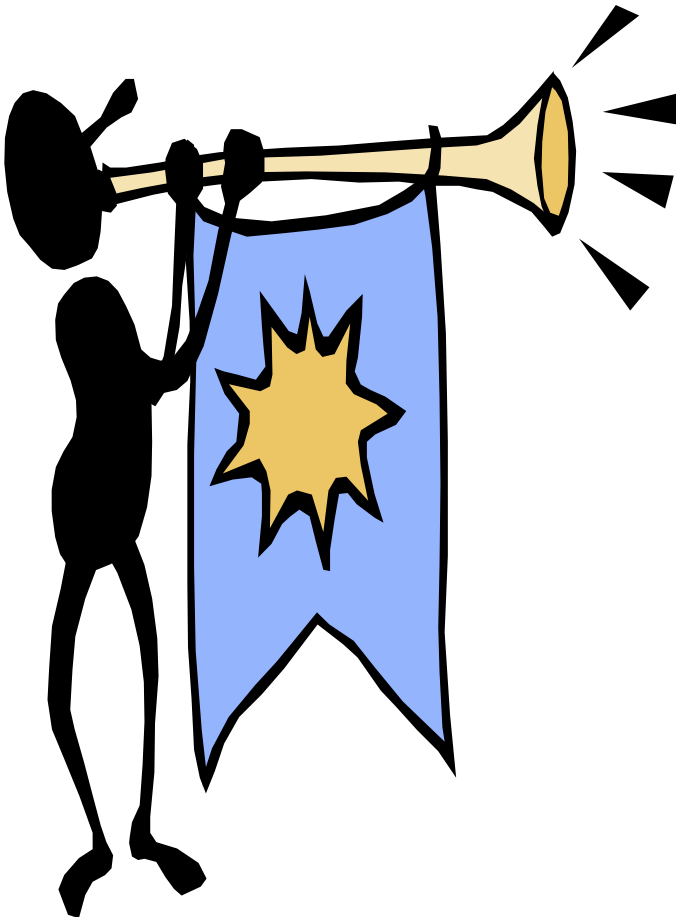
Chapter VI –

RESTful Web Services For Value Added Services (VAS) in NGNs

http://users.encs.concordia.ca/~glitho/

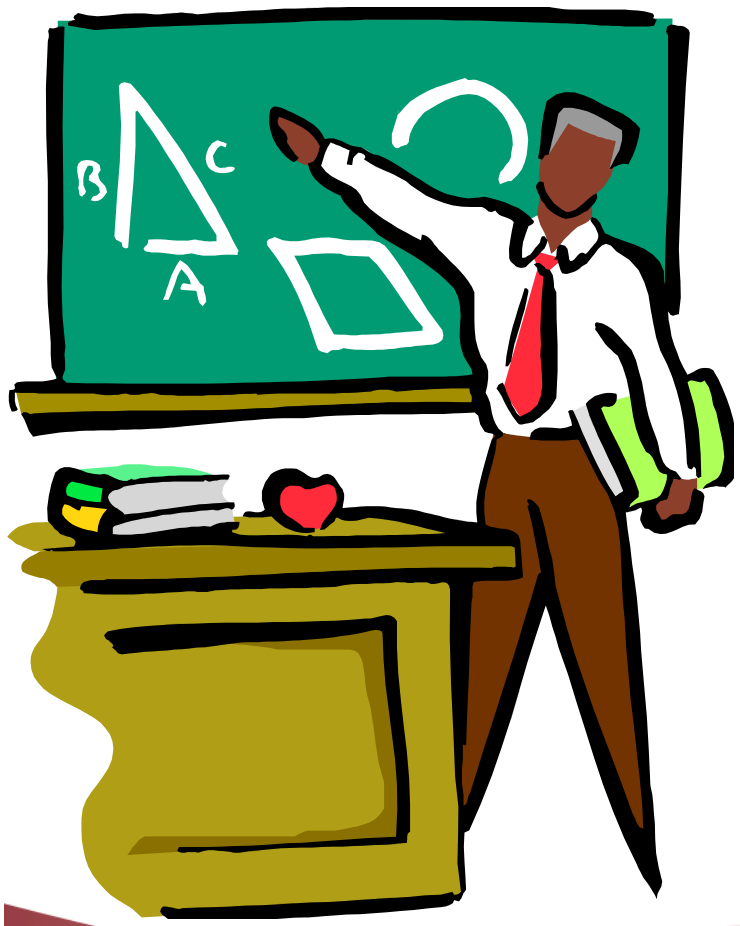
Concordia University
Engineering and Computer Science
Concordia Institute for Information Systems Engineering

Outline



1. Introduction to Web Services
2. RESTful Web Services Overview
3. A Case Study

Introduction to Web Services

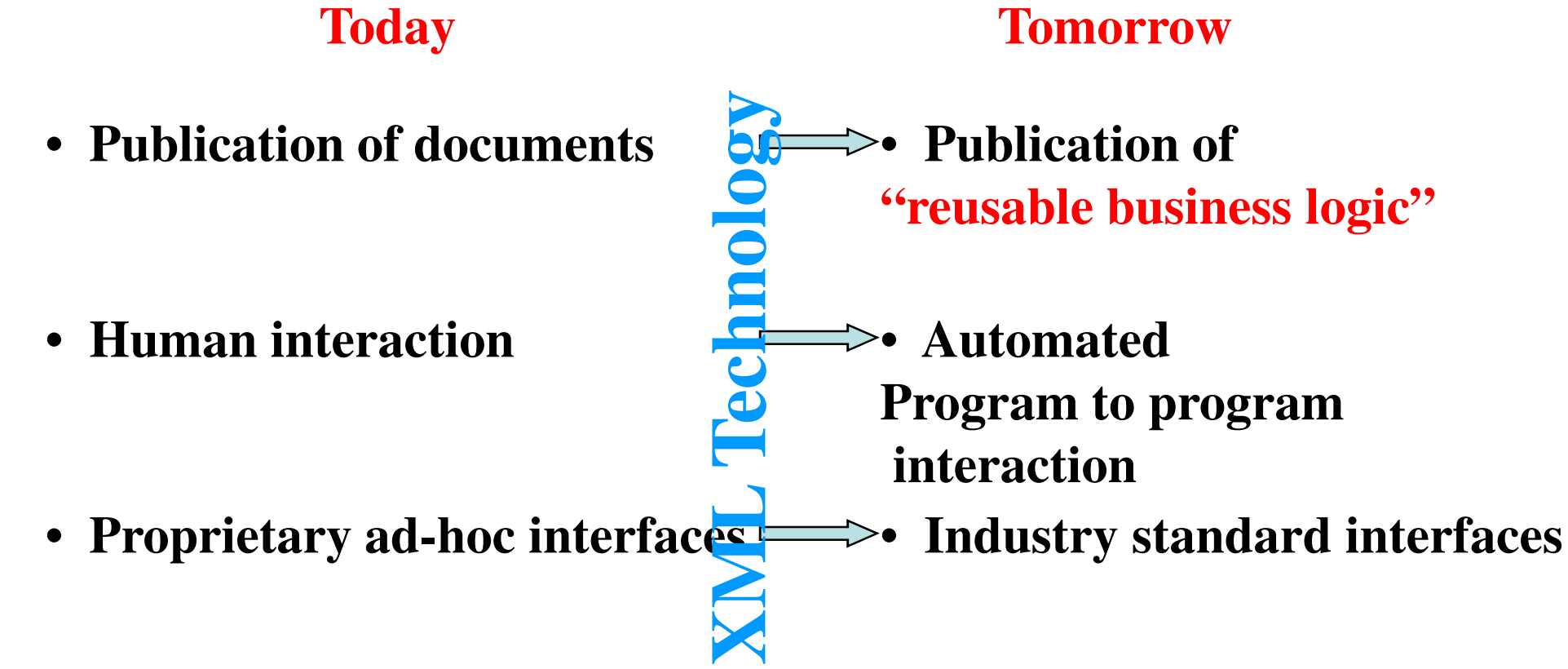


1. Definition and principles
2. Overall business model
3. Technologies

Web Services so far

- **RESTful Web Services**
  - **Most `recent``**
- **SOAP – BASED WEB SERVICES**
  - **Sometimes called `Big`` Web Services**
- This part of the course will discuss the common characteristics

Definitions and principles



Note: There are other technologies such as JSON that may be used

Definitions and principles

"The term Web Services refers to an architecture that allows applications (on the Web) to talk to each other. Period. End of statement"

Adam Bobsworth in ACM Queue, Vol1, No1

Definitions and principles

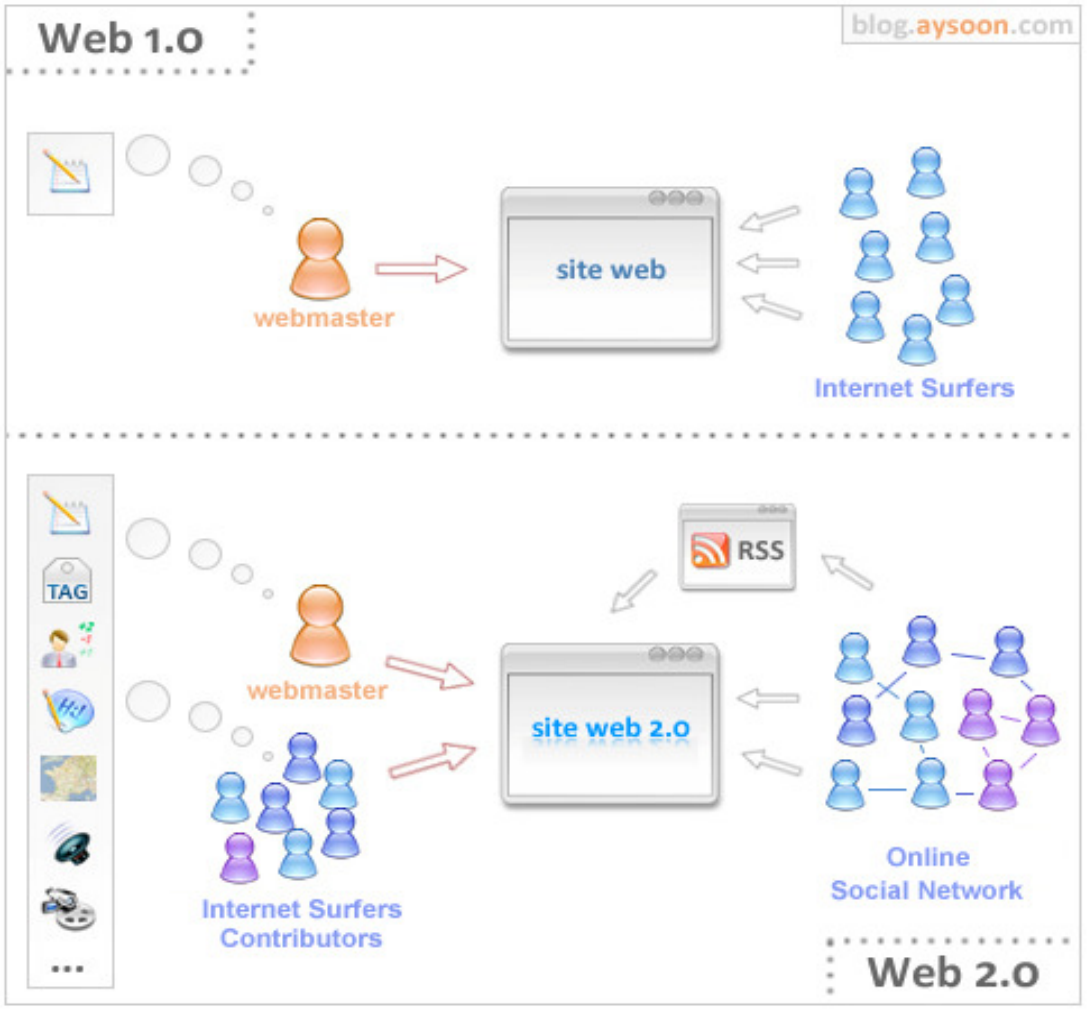
The three fundamental principles, still according to Adam Bobsworth:

1. Coarse grained approach (I.e. high level interface)
2. Loose coupling (e.g. application A which talks to application B should not necessarily be re-written if application B is modified)
3. Synchronous mode of communication, but also asynchronous mode

.

Definitions and principles

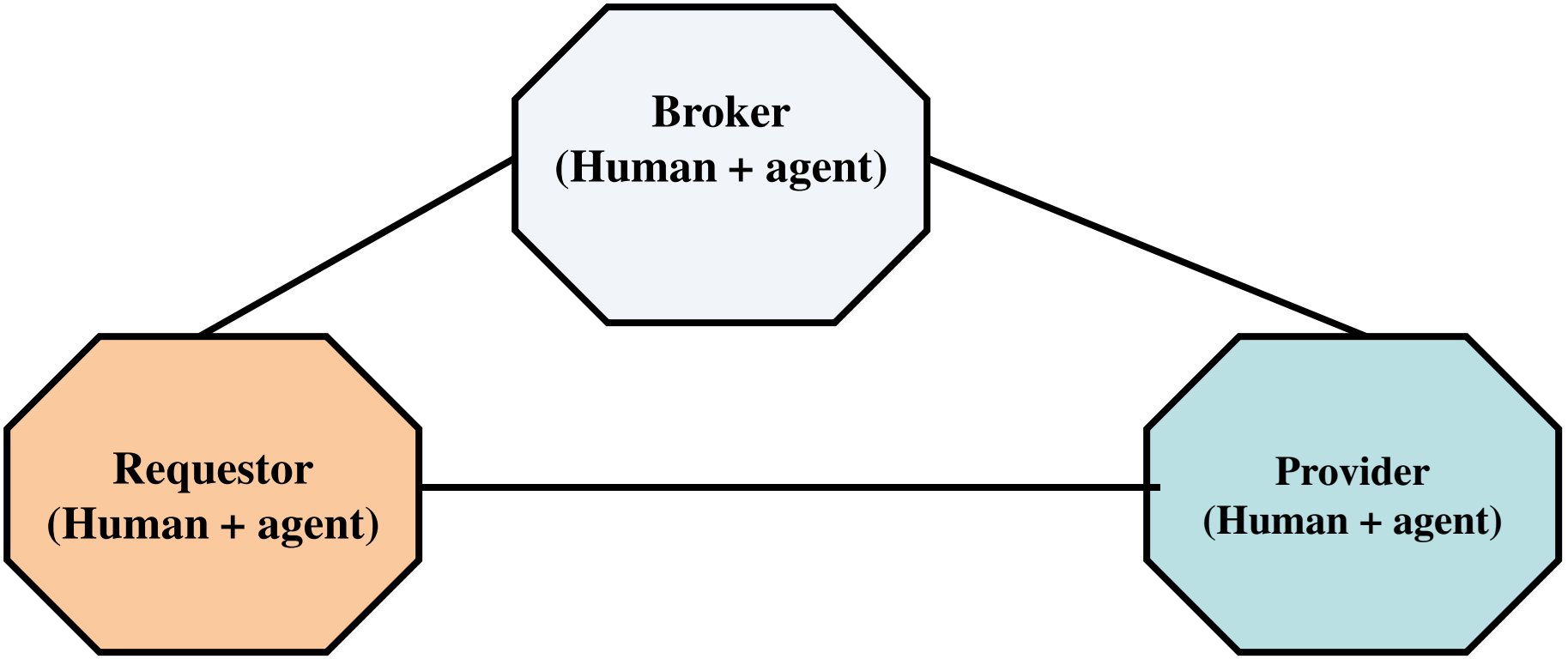
- Web 1.0 , or the human web, is designed for human use.
- Web 2.0, or the programmable web, is designed for consumption by software programs.
- Web 2.0 enables communities and web client participation.



Definitions and principles

Web 1.0	Web 2.0
– Human web	– Programmable web
– Is about HTML	– Is about XML
– Is about client-server	– Is about peer-to-peer
– Is about reading	– about writing
– Is about companies	– Is about communities
–Is about home pages	– Is about blogs
– Is about owning	– Is about sharing
– Is about services sold over the web	– Is about web services
……	…..

Business model



Business model

Requestor

- Person or organization that wishes to make use of a Web service.
- Uses an agent (I.e requestor agent) to exchange messages with both broker agent and provider agent.

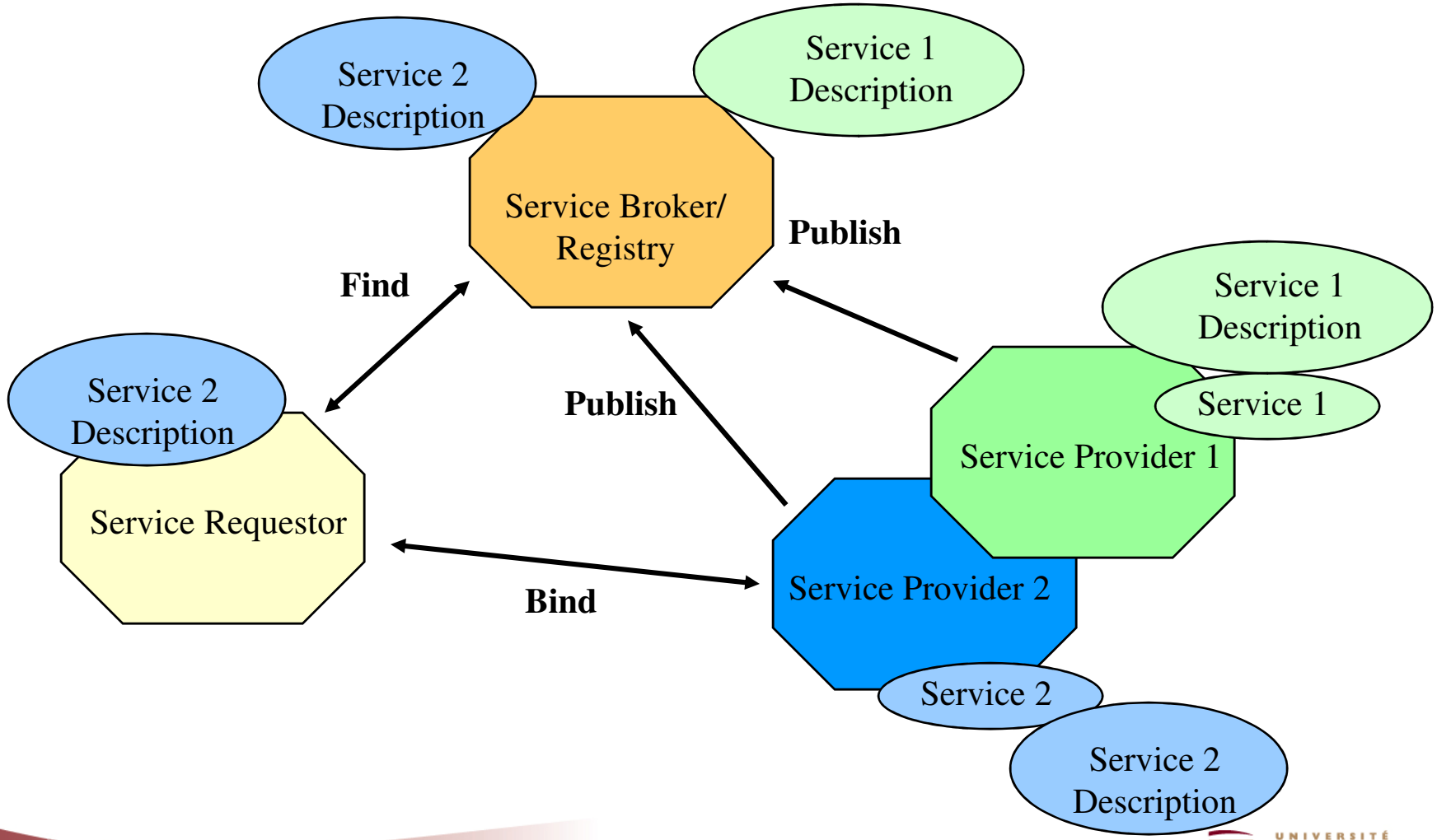
Provider

- Person or organization that owns a Web service it wants to make available for usage
- Use an agent (I.e provider agent) to exchange messages with broker agent and requestor agent.
- The provider agent is also the software piece which implements the Web service (e.g. mapping towards legacy)

Broker

- Person or organization that puts requestors and providers in contact
  - –Providers use brokers to publish Web services
  - –Requestors use brokers to discover Web services
- Use an agent (I.e broker agent) to exchange messages with requestor agent and provider agent

Business model



Technologies

Some of the technologies are mandatory for some Web services while optional for other Web services:

HTTP

- Mandatory for RESTful Web services but optional for SOAP Based Web services

XML

- Mandatory for SOAP Based Web Services but optional for RESTful Web services

HTTP

HTTP (HyperText Transfer Protocol)

- **Is an application-level protocol for distributed, collaborative, hypermedia information systems**
  - **HTTP has been in use since 1990**
  - **HTTP is a request-response protocol**
  - **HTTP requests relates to resources**
    - **A resource is any object or service network that can be identified by a URI (Universal Resource Identifier)**

HTTP

Client

- A program that establishes connections for the purpose of sending requests

User Agent

- The client which initiates a request (e.g. browser)

- Note
  - A request may pass through several servers

HTTP

Server

- An application program that accepts connections in order to service requests by sending back responses
- A given program may be capable of being both a client and a server
- The role depends on connections

HTTP

- Origin server
  - The server on which a given resource resides or is to be created
- Proxy server
  - An intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients
- Gateway server
  - receives requests as if it were the origin server for the requested resource, and forwards the request to another server
  - Is transparent to the client

HTTP

HTTP-message = Request | Response

generic-message = start-line
*(message-header CRLF)
CRLF
[message-body]

start-line = Request-Line | Status-Line

HTTP

HEAD

- retrieve meta-information about a web page, without retrieving the page content (ex: get the date for last modification)

GET

- retrieve the page content

PUT

- store the enclosed content under the supplied Request-URI

POST

- add the entity enclosed in the request as a new subordinate of the resource identified by the Request-URI
  - E.g.
    - Post a message to a mailinglist
    - Extend a database by appending information
    - Transfer a form data

HTTP

DELETE

- Deletes the page

TRACE

- Debug

OPTIONS

- Allows the client to discover the options supported by the server supporte

CONNECT

- Not used currently

HTTP

The built-in HTTP request methods.

Method	Description
GET	Request to read a Web page
HEAD	Request to read a Web page's header
PUT	Request to store a Web page
POST	Append to a named resource (e.g., a Web page)
DELETE	Remove the Web page
TRACE	Echo the incoming request
CONNECT	Reserved for future use
OPTIONS	Query certain options

HTTP

The status code response groups.

Code	Meaning	Examples
1xx	Information	100 = server agrees to handle client's request
2xx	Success	200 = request succeeded; 204 = no content present
3xx	Redirection	301 = page moved; 304 = cached page still valid
4xx	Client error	403 = forbidden page; 404 = page not found
5xx	Server error	500 = internal server error; 503 = try again later

XML

XML is a markup language for documents containing structured information

XML makes use of tags just like HTML.

- In HTML, both tag semantics (<p> means paragraph) and tag set are fixed)

XML was designed to overcome the limitations of HTML

- Better support for dynamic content creation and management
- Interactions between programs going further than browser / Web page

W3C recommendation

XML

Main differences between HTTP and XML

- XML was designed to carry data
- XML is not a replacement for HTML
- XML and HTML were designed with different goals
  - XML was designed to describe data and to focus on what data is.
  - HTML was designed to display data and to focus on how data looks.
  - HTML is about displaying information, while XML is about describing information.
- XML is free and extensible (xml tags are not predefined)

XML

XML advantages

- Structure information
- Separate actual data from data representation
- Store data in plain text format
- Share data in software-and-hardware independent way
- Exchange data between incompatible systems
- Create new languages (WAP, WML).
- Platform independent

XML

XML documents

Data objects made of elements

- <element> content </element>

Well-formed Documents

- If it obeys to the XML syntax
  - Exp: - All XML elements must have a closing tag
    - The name in an element's end-tag MUST match the element type in the start-tag.
    - All XML elements must be properly nested

Valid document

- A well-formed document is valid if it obeys to the structural rules of the associated DTD or schema document

XML

XML documents

Schema and DTD (Document Type Definition)

- Provide a grammar for a class of documents
- Define the legal elements of an XML document

Namespace

- An XML namespace is a collection of names, identified by a URI reference
- Provides a simple method for qualifying element and attribute names used in XML documents.

XML

XML documents Example

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<book>
        <title>Understanding Web Services</title>
        <author>Eric Newcomer</author>
        <price>39.99</price>
</book>
```

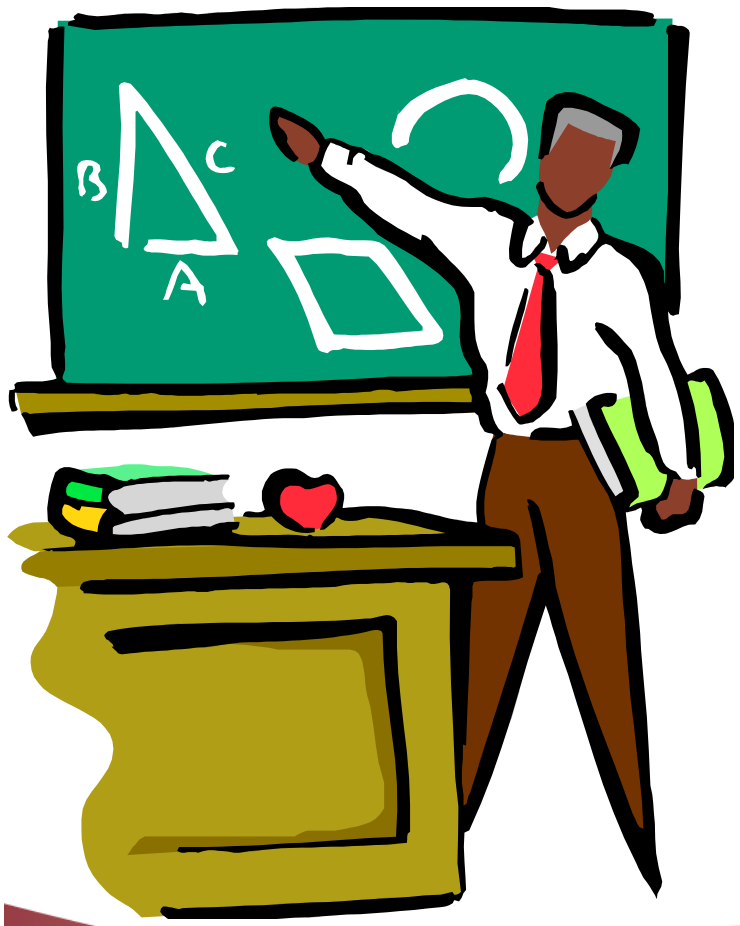
XML

XML processor

- Read XML documents
- Provide access to the content and the structure
- Behaviour described in the XML specifications
- Navigate XML document structure and add, modify, or delete its elements.

- Most popular programming APIs
  - Document Object Model (DOM) from W3C
  - Simple API for XML (SAX) – From XML-DEV mailing list

RESTFul Web Services



1. Introduction
2. Resource Oriented Architecture
3. Resources
4. Properties
5. Tool kits
6. Examples of RESTful Web services

Introduction

- What about using the Web's basic technologies (e.g. HTTP) as a platform for distributed services?
  - This is what is REST about.

Introduction

- REST was first coined by Roy Fielding in his Ph.D. dissertation in 2000
- It is a network architectural style for distributed hypermedia systems.
- It is not an architecture, but a set of design criteria that can be used to asses an architecture

Introduction

- REST is a way to reunite the programmable web with the human web.

- It is simple
  - Uses existing web standards
  - The necessary infrastructure has already become pervasive
  - RESTFull web services are lightweight
  - HTTP traverse firewall

Introduction

- RESTFul web services are easy for clients to use
- Relies on HTTP and inherits its advantages, mainly
  - Statelessness
  - Addressability
  - Unified interface

Resource-Oriented Architecture

- The Resource-Oriented Architecture (ROA)
  - Is a RESTful architecture
  - Provides a commonsense set of rules for designing RESTful web services

Resource-Oriented Architecture

- Concepts
  - Resources
    - Resources names (Unified Resource Identifiers-URIs)
    - Resources representations
    - Links between resources
- Key properties:
  - Addressability
  - Statelessness
  - Uniform interface

Resources

- What's a Resource?
  - A resource is any information that
    - can be named
    - Is important enough to be referenced as a thing in itself
  - A resource may be a physical object or an abstract concept
  - e.g.
    - a document
    - a row in a database
    - the result of running an algorithm.

Resources

- Naming:
  - Unified Resource Identifier (URI)
    - The URI is the name and address of a resource
    - Each resource should have at least one URI
    - URIs should have a structure and should vary in predictable ways

Resource

Representation

- A representation is any useful information about the state of a resource
- Different representation formats can be used (Unlike SOAP based Web services)
  - *plain-text*
  - *JSON*
  - XML
  - XHTML
  - ….

Resource

…

- In most RESTful web services, representations are hypermedia
  - i.e. documents that contain data, and links to other resources.

Properties

- Addressability
  - An application is addressable if it exposes a URI for every piece of information it serves
  - This may be an infinite number of URIs
    - e.g. for search results
      - *http://www.google.com/search?q=jellyfish*

Properties

- Statelessness
  - The state should stay on the client side, and be transmitted to the server for every request that needs it.
    - Makes the protocol simpler
    - Ease load balancing

Properties

- Uniform interface
  - *HTTP GET:*
    - Retrieve a representation of a resource
  - *HTTP PUT*
    - Create a new resource, where the client is in charge of creating the resource URI: *HTTP PUT* to the new URI
    - Modify an existing resource: *HTTP PUT* to an existing URI
  - *HTTP POST:*
    - Create a new resource, where the server is in charge of creating the resource URI: *HTTP POST* to the URI of the superordinate of the new resource
  - *HTTP DELETE:*
    - Delete an existing resource:
  - HTTP HEAD:
    - Fetch metadata about a resource
  - HTTP OPTIONS:
    - Lets the client discover what it's allowed to do with a resource.

Examples of tool kits

- RestLet
- Jersey

Examples of RESTful Web Services

- Examples of existing RESTful web services include:
  - Amazon's Simple Storage Service (S3) (*http://aws.amazon.com/s3*)
  - Services that expose the Atom Publishing Protocol (*http://www.ietf.org/html.charters/atompub-charter.html*) and its variants such as GData (*http://code.google.com/apis/gdata/*)
  - Most of Yahoo!'s web services (*http://developer.yahoo.com/*)
  - Twitter is a popular blogging site that uses RESTful Web services extensively.
  - Most other read-only web services that don't use SOAP
  - Static web sites
  - Many web applications, especially read-only ones like search engines

Examples of RESTful Web Services

Resources	URL Base URL: http://{serverRoot}/{apiVersion}/smsmessaging	HTTP action
Outbound SMS message requests	/outbound/{senderAddress}/requests	GET: read pending outbound message requests POST: create new outbound messages request
Outbound SMS message request and delivery status	/outbound/{senderAddress}/requests /{requestId}	GET: read a given sent message, along with its delivery status
Inbound SMS message subscriptions	/inbound/subscriptions	GET: read all active subscriptions POST: create new message subscription
Individual inbound SMS message subscription	/inbound/subscriptions/{subscriptionId}	GET: read individual subscription DELETE: remove subscription and stop corresponding notifications

Table 2. *A subset of ParlayREST SMS resources.*

Examples of RESTful Web Services

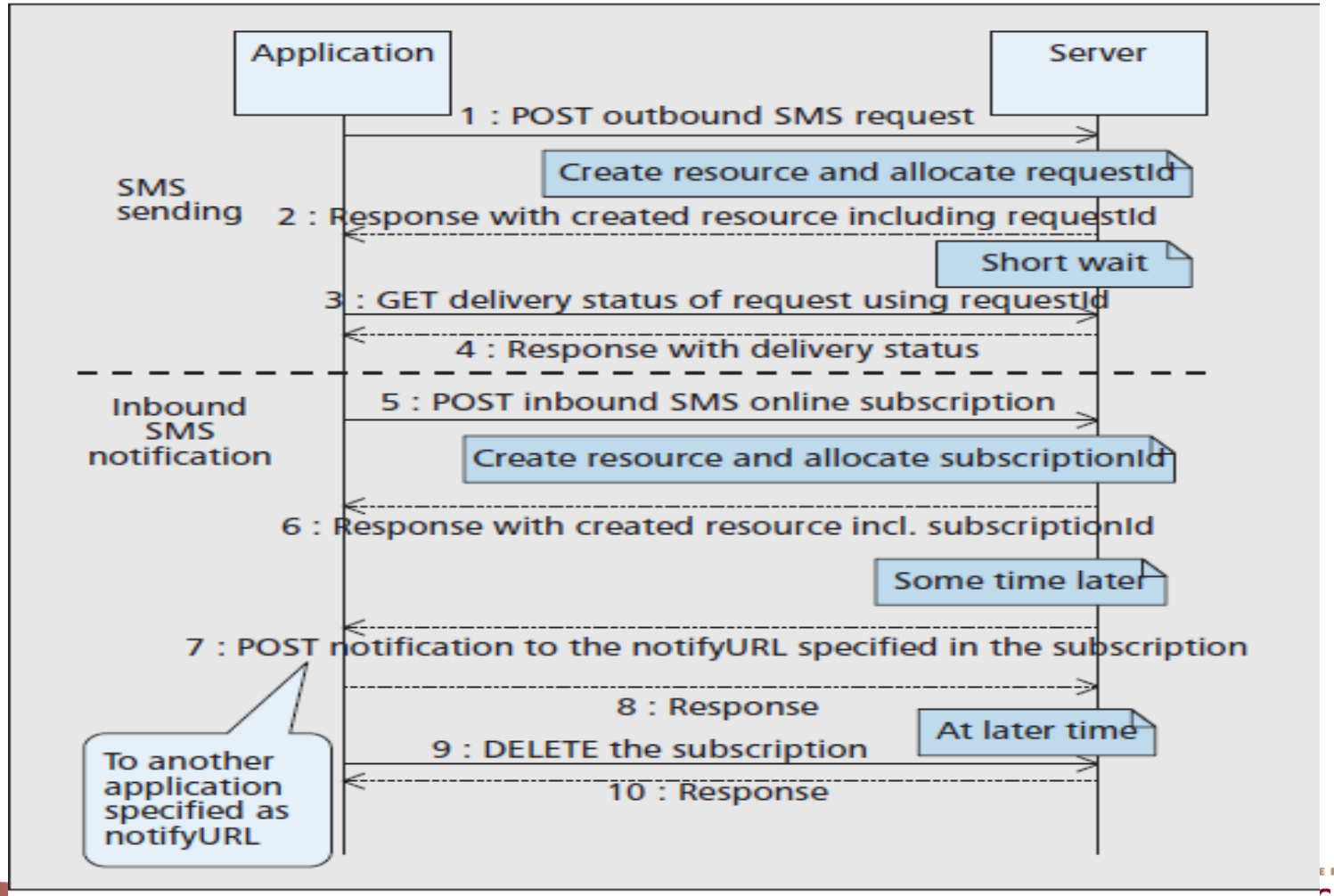
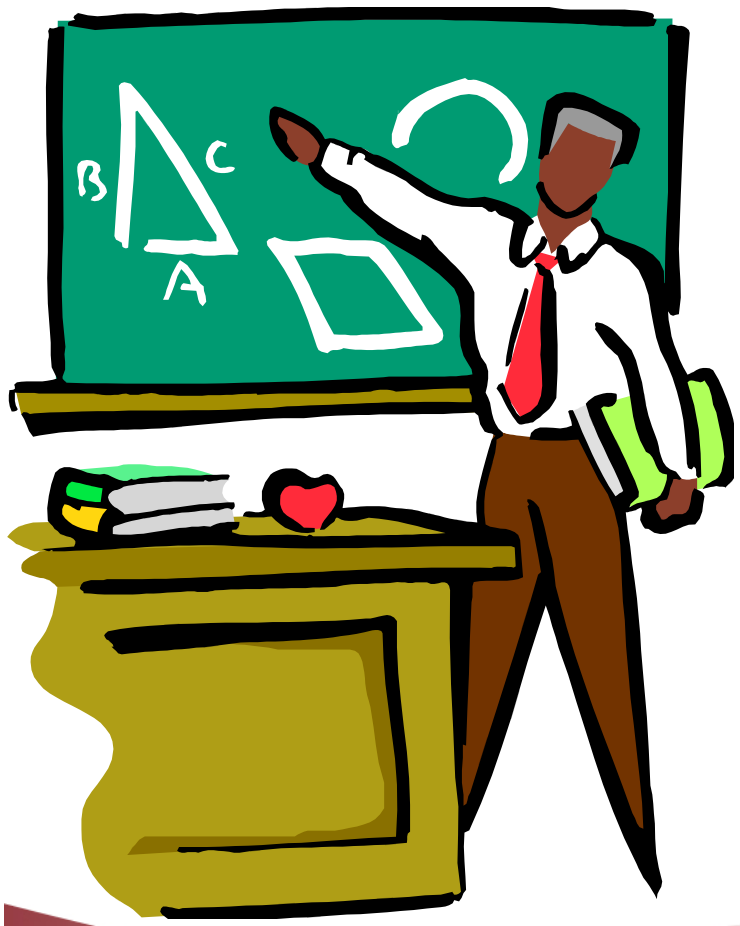


Figure 4. *Sample scenario for SMS handling.*

Illustrative Use Case on conferencing



1. Procedure
2. On conferencing semantics
3. Applying the procedure

The procedure – First Part

- Figure out the data set
- Split the data set into resources

The procedure – Second Part

For each resource:

- Name the resources with URIs
- Identify the subset of the uniform interface that is exposed by the resource
- Design the representation(s) as received (in a request) from and sent (in a reply) to the client
- Consider the typical course of events by exploring and defining how the new service behaves and what happens during a successful execution

On Conferencing semantics

- The conversational exchange of multimedia content between several parties
  - About multimedia
    - Audio, video, data, messaging



  - About participants
    - Any one who wants to participates the conference



On Conferencing semantics

Classification:

- Dial-in / dial-out
- Open/close
- Pre-arranged/ad hoc
- With/without sub-conferencing (i.e. sidebar)
- With/without floor control

On conferencing semantics

- Case considered in the use case
  - Create a service that allows a conference manager to :
    - Create a conference
    - Terminate a conference
    - Get a conference status
    - Add users to a conference
    - Remove users from a conference
    - Change media for a participant
    - Get a participant media

Applying the procedure – First part

1. Data set

   - Conferences
   - Participants
   - Media

Applying the procedure – First part

2. Split the data set into resources
   - Each conference is a resource
   - Each participant is a resource
   - One special resource that lists the participants
   - One special resource that lists the conferences (if we consider simultaneous conferences)

Applying the procedure – Second part

3. Name the resources with URIs

- I'll root the web service at *http://www.confexample.com/*
- I will put the list of conferences at the root URI
- Each conference is defined by its ID: *http://www.confexample.com/{confId}/*
- A conference participants' resources are subordinates of the conference resource:
  - The lists of participants: *http://www.confexample.com/{confId}/participants/*
  - Each participant is identified by his/her URI:

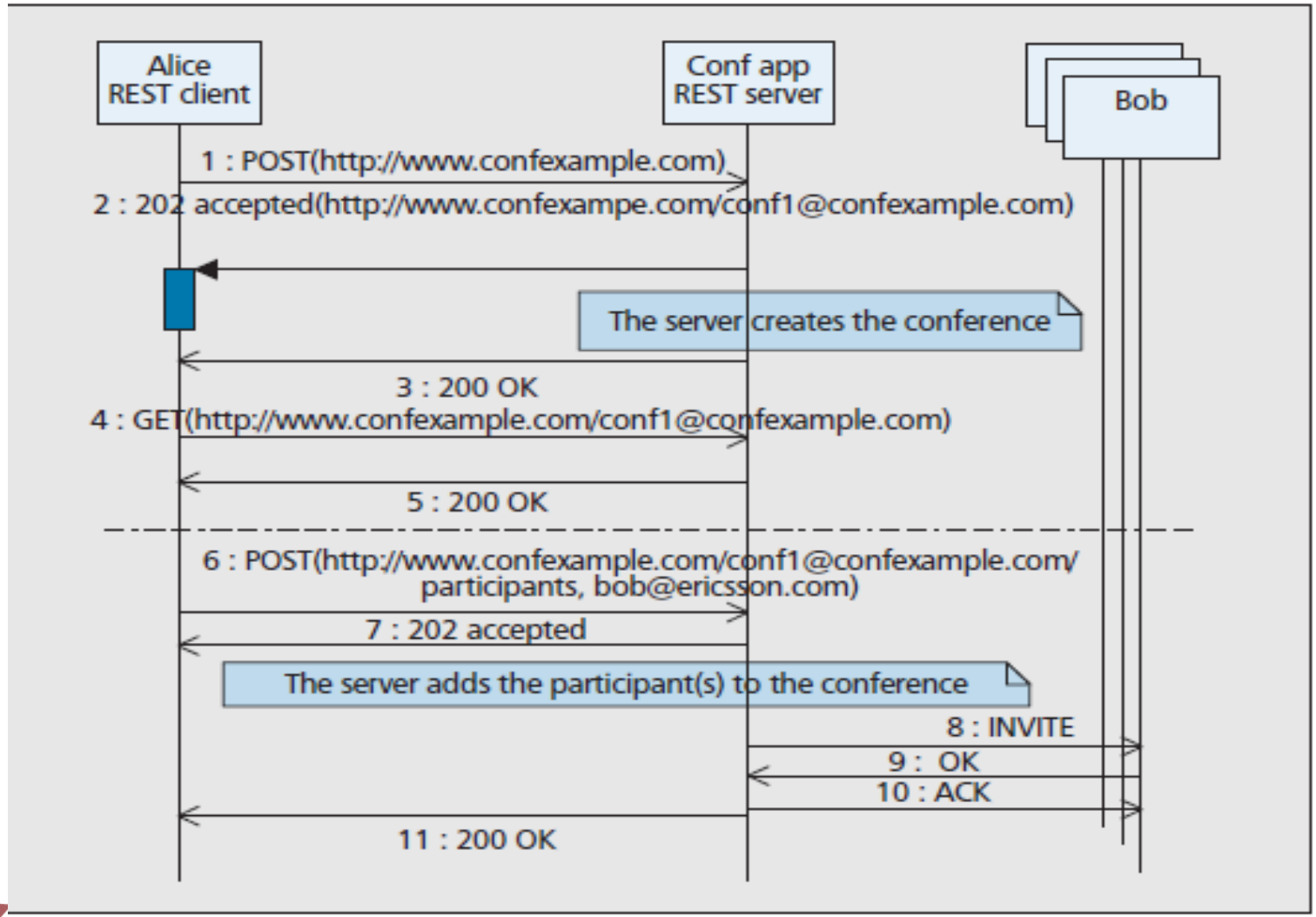
http://www.confexample.com/{confId}/participants/{participantURI}/

Applying the procedure – Second part

Resource	Exposed subset of the uniform interface		Data representation operation	
	Operation	HTTP action	Client->server	Server->client
Conference	Create: establish a conference	POST: http://confexample.com/	<conference> <description> discuss project </description> <maxParticipants>10</maxParticipants> </conference>	http://www.confexample/conf23@example.com
	Read: Get conference status	GET: http://confexample.com/{confId}	None	<status>Active</status>
	Delete: end a conference	DELETE: http://confexample.com/{confId}	None	None
List of participant(s)	Read: Get list of participants	GET: http://confexample.com/{confId}/participants	None	<participants> <participant> <uri>alice@ericsson.com</uri> <status>Connected</status> </participant> .... </participants>
	Create: Add a participant	POST: http://confexample.com/{confId}/participants	<participant> alice@ericsson.com </participant>	<participant> <uri>alice@ericsson.com</uri> <link>http://confexample.com/{confId}/participants/alice@ericsson.com</link> </participant>
	Read: Get a participant status	GET: http://confexample.com/{confId}/participants/{participantURI}	None	<status>Invited</status>
	Delete: remove a participant	DELETE: http://confexample.com/{confId}/participants/{participantURI}	None	None

conference?

Applying the procedure – Second part



Applying the procedure – Second part

9. What might go wrong?
   - Conference

Operation	Server->Client	Way it may go wrong
Create (POST)	Success: 200 OK Failure: 400 Bad Request	The received request is not correct (e.g. has a wrong body)
Read (GET)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	The targeted conference does not exist

Applying the procedure – Second part

9. What might go wrong?
   - Participant(s)

Operation	Server->Client	Way it may go wrong
Create (POST)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct (e.g. has a wrong body) • The target conference does not exist
Read (GET)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist
Update (PUT)	Success: 200 OK Failure: 400 Bad Request Failure: 404 Not Found	• The received request is not correct • The target conference does not exist • The target participant does not exist
Delete (DELETE)	Success: 200 OK Failure: 404 Not Found	• The target conference does not exist • The target participant does not exist

References

- F. Belqasmi, C. Fu, R. Glitho, Services Provisioning in Next Generation Networks: A Survey, IEEE Communications Magazine, December 2011
- L. Richardson and S. Ruby, "RESTful Web Services", O' Reilly & Associates, ISBN 10: 0-596-52926-0, May 2007
- Lightweight REST Framework, http://www.restlet.org/
- C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. "Big"Web Services: Making the Right Architectural Decision", In Proceedings of the 17th International World Wide Web Conference, pages 805–814, Beijing, China, April 2008, ACM Press.
- C. Pautasso and E. Wilde, "Why is the web loosely coupled? A multi-faceted metric for service design", in Proc. of the 18th World Wide Web Conference, Madrid, Spain (April 2009)