Haute école d’ingénierie et d’architecture Fribourg
Hochschule für Technik und Architektur Freiburg

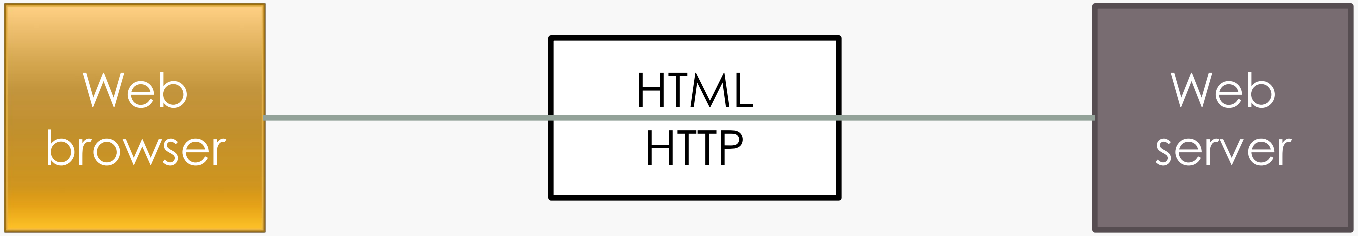
DEVELOPPEMENT BACKEND

WEBSERVICES / RESTFUL APIS

SERGE AYER - HEIA-FR – ISC
CLASSES ISC-ID-2A/D // 2023-2024

WEB SERVICES: A SHORT HISTORY

- Web sites until 2000

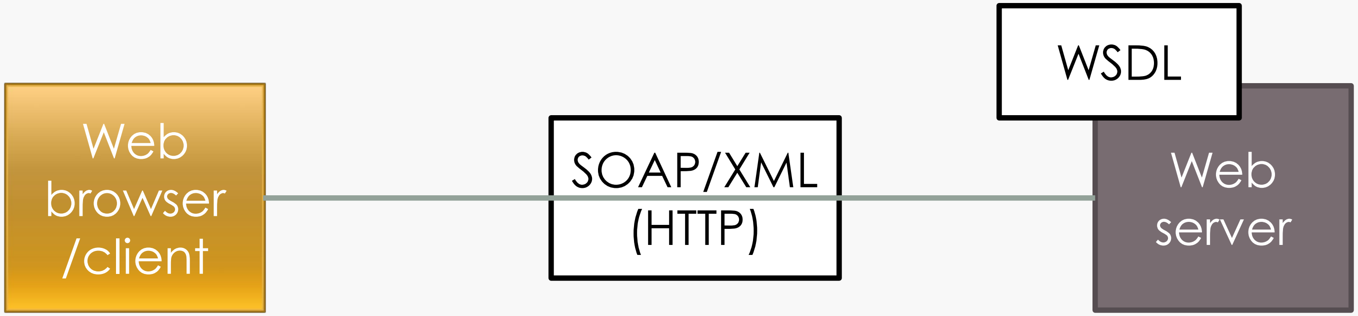


Web browser

HTML
HTTP

Web server

- The Programmable Web (1998)



Web browser /client

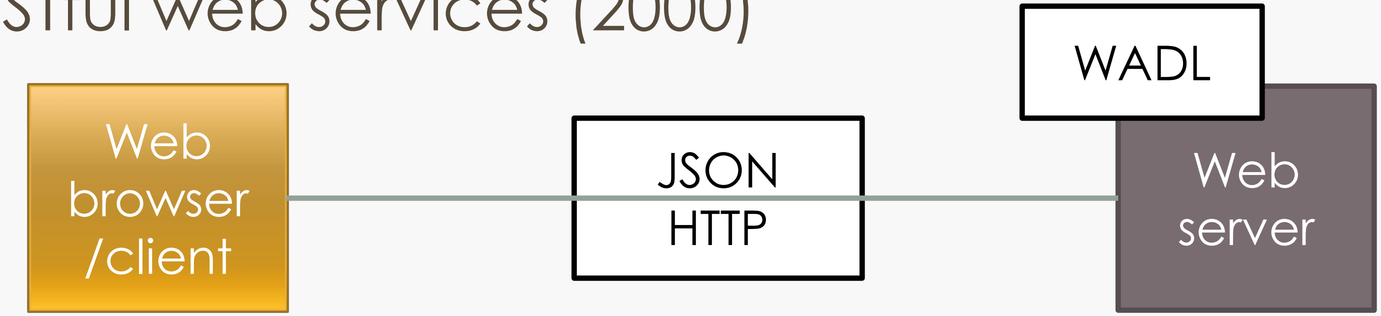
SOAP/XML
(HTTP)

WSDL

Web server

WEB SERVICES: A SHORT HISTORY

- RESTful web services (2000)



WEB SERVICES

- Definition by W3C: A software system designed to support interoperable machine-to-machine interaction over a network:
  - Involving at least two devices: a (web) server and a client (e.g. a browser).
  - The network is IP, usually using HTTP.
- Usually categorized into two major classes:
  - WS-* or Arbitrary web services
  - REST-compliant web services

WS-* WEB SERVICES

- Functionalities and interfaces declared through WSDL (Web Services Description Language) which is a machine-processable format.
- Client requests and service response objects are encapsulated using SOAP (Simple Object Access Protocol) and transmitted over the network using HTTP.
- These services are usually called WS-* or big web services.

RESTFUL WEB SERVICES

- Manipulation of representations of Web resources using a uniform set of stateless operations
  - At the core: resources
  - Resources are uniquely identified through URIs (Unique Resources Identifiers)
  - Uses URIs to identify resources and HTTP as their service interface
- RESTful web services for connected objects (smart things) is usually known as the Web of Things.

THE PROGRAMMABLE WEB

- The programmable web is not necessarily for human consumption.
  - Its data is intended as input to a software program that does something interesting with it.
- The programmable web is relying on HTTP:
  - HTTP is delivering documents in envelopes.
    - HTTP does not care about what is in the envelope.
  - HTTP is the one thing that all clients and services have in common on the programmable web.

TECHNOLOGIES OF THE PROGRAMMABLE WEB

- HTTP: envelope format
- URI
  - A RESTful, resource-oriented service exposes a URI for every piece of data the client might want to access.
  - A RPC/SOAP service exposes a URI for every process capable of handling the Remote Procedure Call
    - usually called the endpoint and usually unique.
- SOAP: envelope format, on top of HTTP, XML-based
- WSDL (Web Service Description Language): XML vocabulary used to describe SOAP-based web services.
- WADL (Web Application Description Language): XML vocabulary used to describe RESTful web services.
- Today, OpenAPI is often used as the standard for specifying RESTful web services

WS-* VS RESTFUL WEB SERVICES

- The differences are in
  - The way the client convey its intentions to the server:
    - REST: Using the HTTP methods (standardized).
    - SOAP: Using a specific method (like in any programming language)
      - very likely using the POST HTTP method.
  - The way the client tells to the server which part of the data set to operate on (scoping information):
    - REST: Using the URI path (like "…/search?q=REST")
      - resource oriented
    - SOAP: Using the entity-body of the HTTP request.

WS-* VS RESTFUL WEB SERVICES

- When using SOAP:
  - Everything is in the envelope (and if you don’t open it, you don’t understand the request and its response),
- When using a RESTful architecture:
  - The request can be understood from the HTTP method and from the URI.

REST PRINCIPLES

- REST == "**Re**presentational **S**tate **T**ransfer"
- Resource-based rather than action-based
- Representations moved from server to client
- REST is not an architecture but rather a set of design criteria, which are
  - Uniform Interface
    - The method information is kept in the HTTP method.
  - Stateless
  - Cacheable
  - Client-Server
  - Layered System
  - Code on Demand (optional)
- There are a number of architectures that meet those criteria

RESOURCE BASED

- Things vs. actions (for SOAP-RPC)
  - Nouns vs. verbs
  - Example: “user data” vs. “get user data”
- Identified by URIs
  - Multiple URIs may refer to the same resource
- Resources are separate from their representations
  - Very important since there can also be several representations of the same resource

RESOURCE-ORIENTED ARCHITECTURE

- A resource:
  - Something that can be stored on a computer and represented as a stream of data (bits).
  - A physical object
  - An abstract concept
- Examples:
  - Version 2.0 of a software release
  - The latest release of a software
  - The sales numbers for Q4 2015
  - A list of bugs in a bug database
  - A person
  - The relationship between two persons

RESOURCE-ORIENTED ARCHITECTURE

- Resources are on the web:
  - A resource has to have at least one URI (name and address of the resource).
- URIs should be descriptive
  - Examples (from previous slide)
    - http://www.heia-fr.ch/software/releases/2.0
    - http://www.heia-fr.ch/software/releases/latest
    - http://www.heia-fr.ch/sales/2015/Q4
    - http://www.heia-fr.ch/bugs/open
    - http://www.heia-fr.ch/person
    - http://www.heia-fr.ch/relationships/person1;person2

RESOURCE-ORIENTED ARCHITECTURE

- Relationship between URIs and Resources
  - Two resources cannot be the same
  - More than one URI may refer to the same resource
    - Example: the latest release may be version 2.0
  - Every URI designates exactly one resource
- Addressability
- Statelessness

REPRESENTATIONS

- (Part of) the current state of the resource
  - Any useful information about the state of a resource.
  - Transferred between client and server.
- There can be multiple representations of the same resource:
  - A book can be represented with its cover image and reviews used for advertise the book.
  - The same book can be represented by an electronic copy of the book that can be downloaded via HTTP when you pay for it.

REPRESENTATIONS

- Typically JSON or XML
  - Can also be HTML or CSV or anything else.
- Example:
  - Resource: person
  - Service: contact information (GET)
  - Representation:
    - Name, address and phone number
    - In JSON or XML format

UNIFORM INTERFACE

- Defines the interface between the client and the server
- Simplifies and decouples the architecture
- Typically,
    - HTTP verbs / methods
        - GET: retrieve the representation of a resource
        - PUT: create a new resource
        - POST: create a new (sub)resource to an existing URI
        - DELETE: delete an existing resource
        - HEAD: retrieve a meta-data only representation (same as GET without the entity-body).
        - OPTIONS: check which HTTP methods a particular resource supports
    - URIs (resource name)
    - HTTP response (status and body – JSON)

STATELESS

- Server does NOT contain any client state.
- Each request contains required context to process the message.
  - Self-descriptive messages.
  - The representation contains the state.
- Any session state is held on the client
  - One should distinguish between:
    - Application state
      - Ought to live on the client.
      - Can vary by client and per request.
    - Resource state
      - Ought to live on the server.
      - At a given time is the same for all clients.

CLIENT-SERVER ARCHITECTURE

- Assume a disconnected system
  - Like any web service based system
- Separations of concerns
  - Don’t mix user interface and web services
- The uniform interface is the link between the client and the server

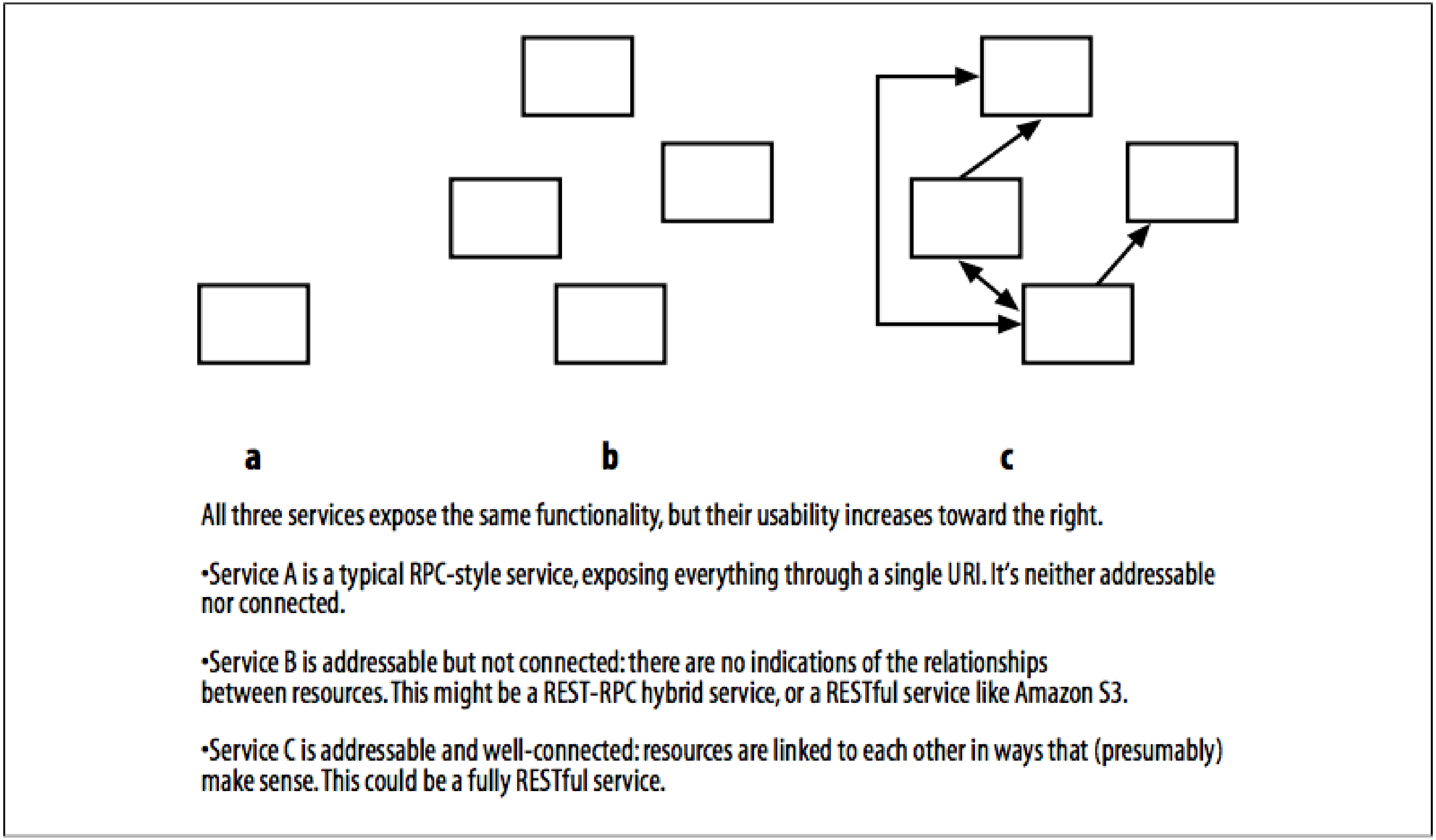
CACHEABLE / LAYERED SYSTEM

- Server responses (or representations) must be cacheable.
  - Implicitly.
  - Explicitly: the server specifies parameters for caching.
  - Negotiated.
- Client can’t assume direct connection to the server.
  - There may be intermediaries between the client and the server.
- This improves scalability.

LINKS AND CONNECTEDNESS

- Very often, representations are hypermedia
  - Documents that contain not just data, but links to other resources.
  - HATEOS or Hypermedia as the Engine of Application State
    - This means that there no HTTP "session" stored on the server as a resource state, but rather that the HTTP "session" is tracked by the client as an application state, and created by the path the client takes through the Web.
  - What it means: resources should link to each other in their representations whenever it makes sense.
    - Counter example: S3 is not connected.

LINKS AND CONNECTEDNESS



All three services expose the same functionality, but their usability increases toward the right.

•Service A is a typical RPC-style service, exposing everything through a single URI. It's neither addressable nor connected.

•Service B is addressable but not connected: there are no indications of the relationships between resources. This might be a REST-RPC hybrid service, or a RESTful service like Amazon S3.

•Service C is addressable and well-connected: resources are linked to each other in ways that (presumably) make sense. This could be a fully RESTful service.

WHY REST ?

- Compliance with the REST constraints allow
  - Scalability
    - Statelessness allows easier scalability and load balancing
      - For instance, the absence of session does not require balancing to worry about session affinity.
  - Simplicity
  - Modifiability
  - Visibility
  - Portability
  - Reliability

REST TIPS

- Use HTTP verbs to mean something.
  - For instance, a GET request must not modify any underlying resource data.
- Provide sensible resource names.
  - Improves understandability of the web service API.
- Use HTTP response codes to indicate status.
- Offer JSON or XML or both for entity-body.
- Create fine-grained resources.
  - If requested, it is easier to create aggregate services – that utilize multiple underlying resources – from fine-grained resources than the other way around.
  - Provide CRUD (Create, Read, Update, Delete) functionality on those fine-grained resources.
- Consider connectedness. HATEOS

RESOURCE URI EXAMPLE

- Insert a new customer in a system
  - POST http://www.example.com/customers
- To read customer with a given customerID
  - GET http://www.example.com/customers/"customerID"
- To create an order for a specific customer
  - POST http://www.example.com/customers/"customerID"/orders
- To read all items of a specific order
  - GET http://www.example.com/customers/"customerID"/orders/"orderID"

RESOURCE URI BAD EXAMPLES

- Avoid single URI to specify the service interface, using query-string parameters to specify the requested operation and/or HTTP verbs.
- Do not use the GET method for operations which are not GET as:
  - GET http://www.example.com/customers/"customerID"/update
- Do not use redundant verbs and resources as:
  - PUT http://www.example.com/customers/"customerID"/update