# Web applications and web services

2024/25 Q2

***Jaime Delgado, Silvia Llorente* ***

DAC – UPC

* Part of the material originates from other sources

# Contents

## WEB APPLICATIONS AND WEB SERVICES

- Web applications. Dynamic pages, JSP, Servlets, …
- HTTP-based development of applications & services
- Distributed applications. “Remote operations”
- Web services: Model, WSDL, (SOAP), REST
- A real example
- Programming tools

# Contents

## WEB APPLICATIONS AND WEB SERVICES

- Web applications. Dynamic pages, JSP, Servlets, …
- HTTP-based development of applications & services
- Distributed applications. “Remote operations”
- Web services: Model, WSDL, (SOAP), REST
- A real example
- Programming tools

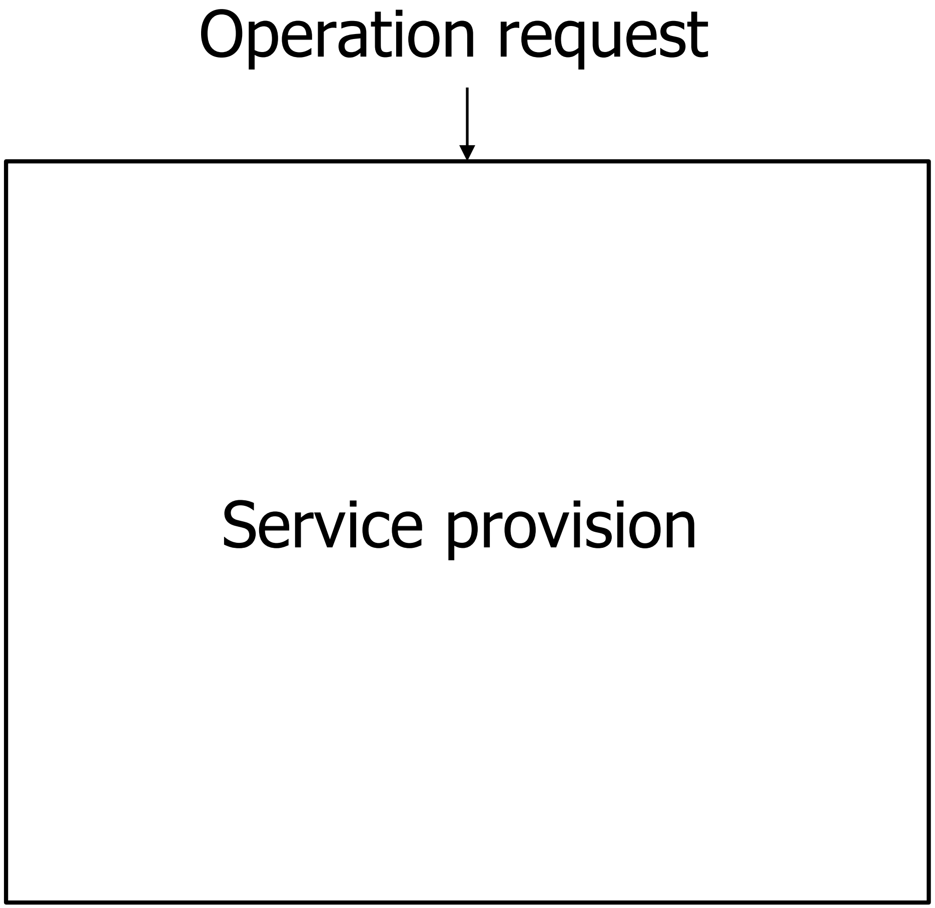
At the laboratory sessions

# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - WSDL / (WADL)
  - (SOAP)
  - REST
- A real example

# Distributed applications

## LOCAL

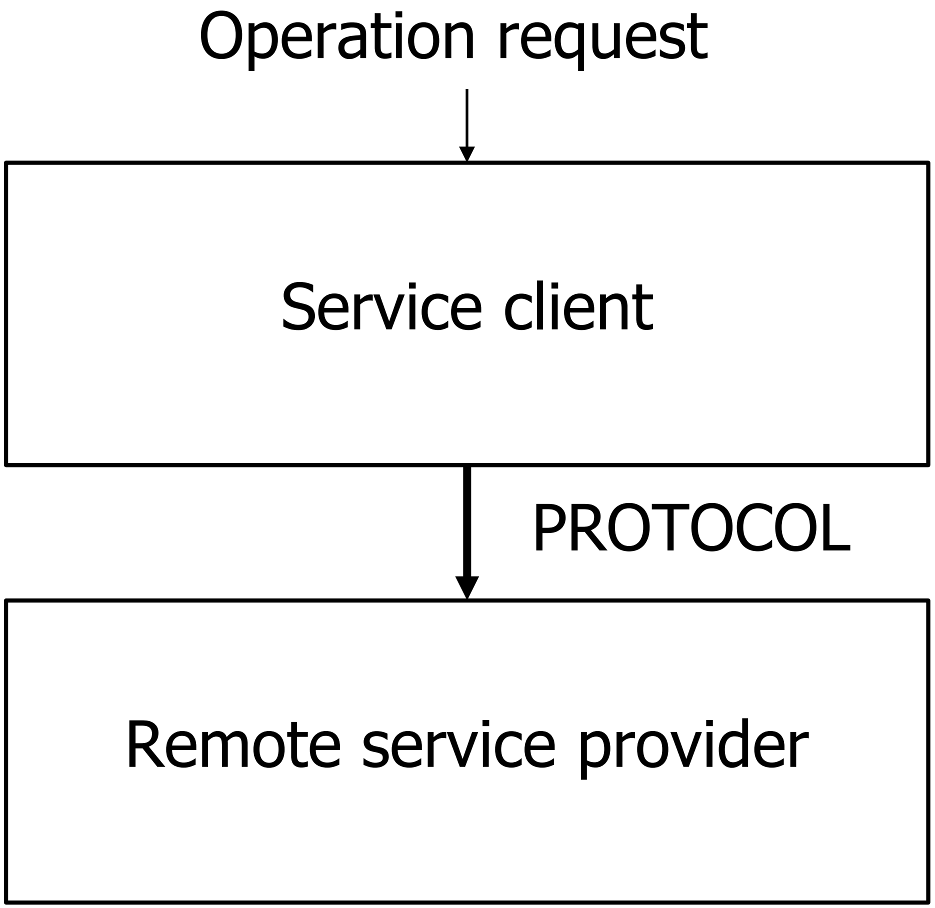


Operation request

Service provision

# Distributed applications

## DISTRIBUTED



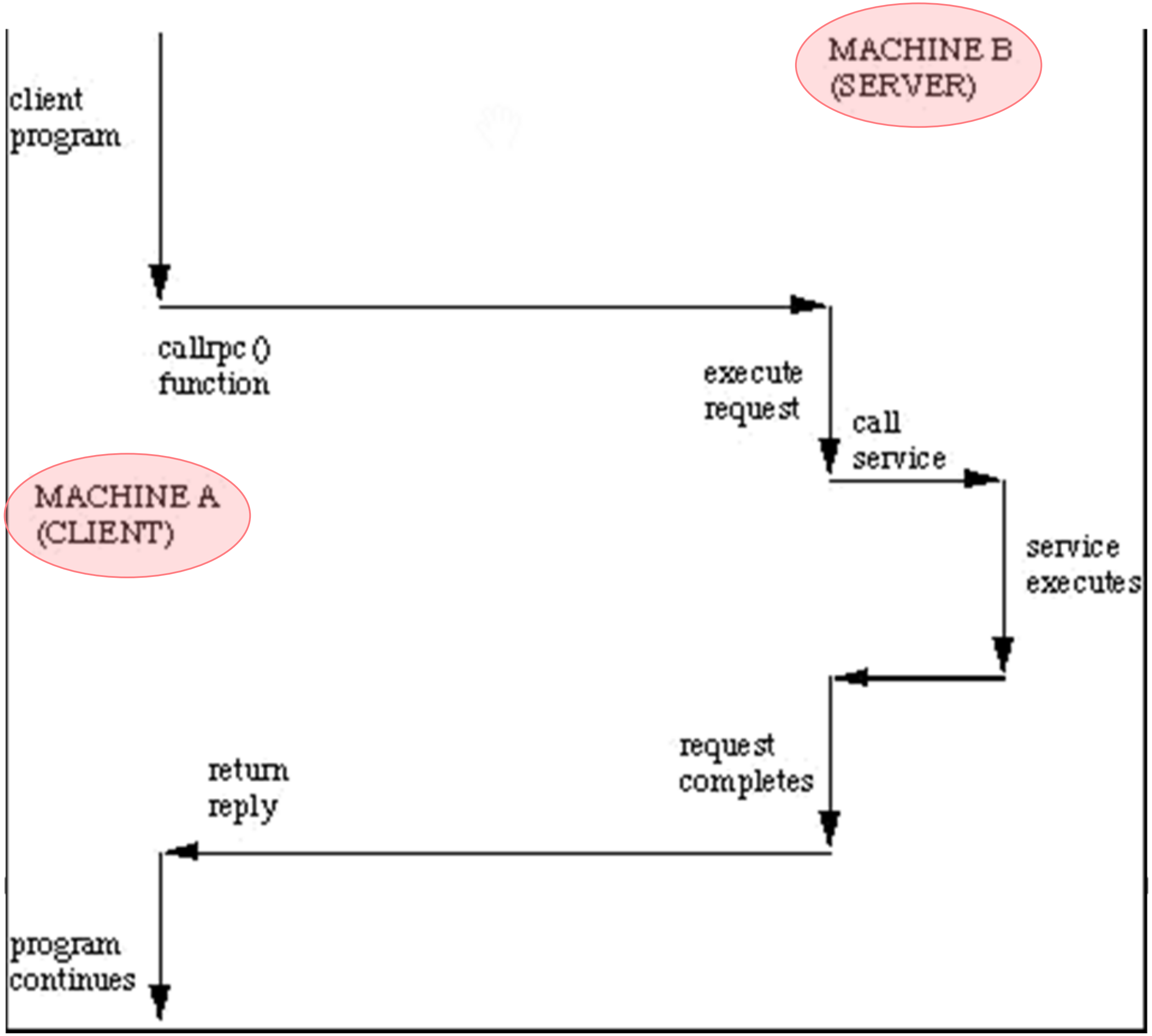
Operation request

Service client

PROTOCOL

Remote service provider

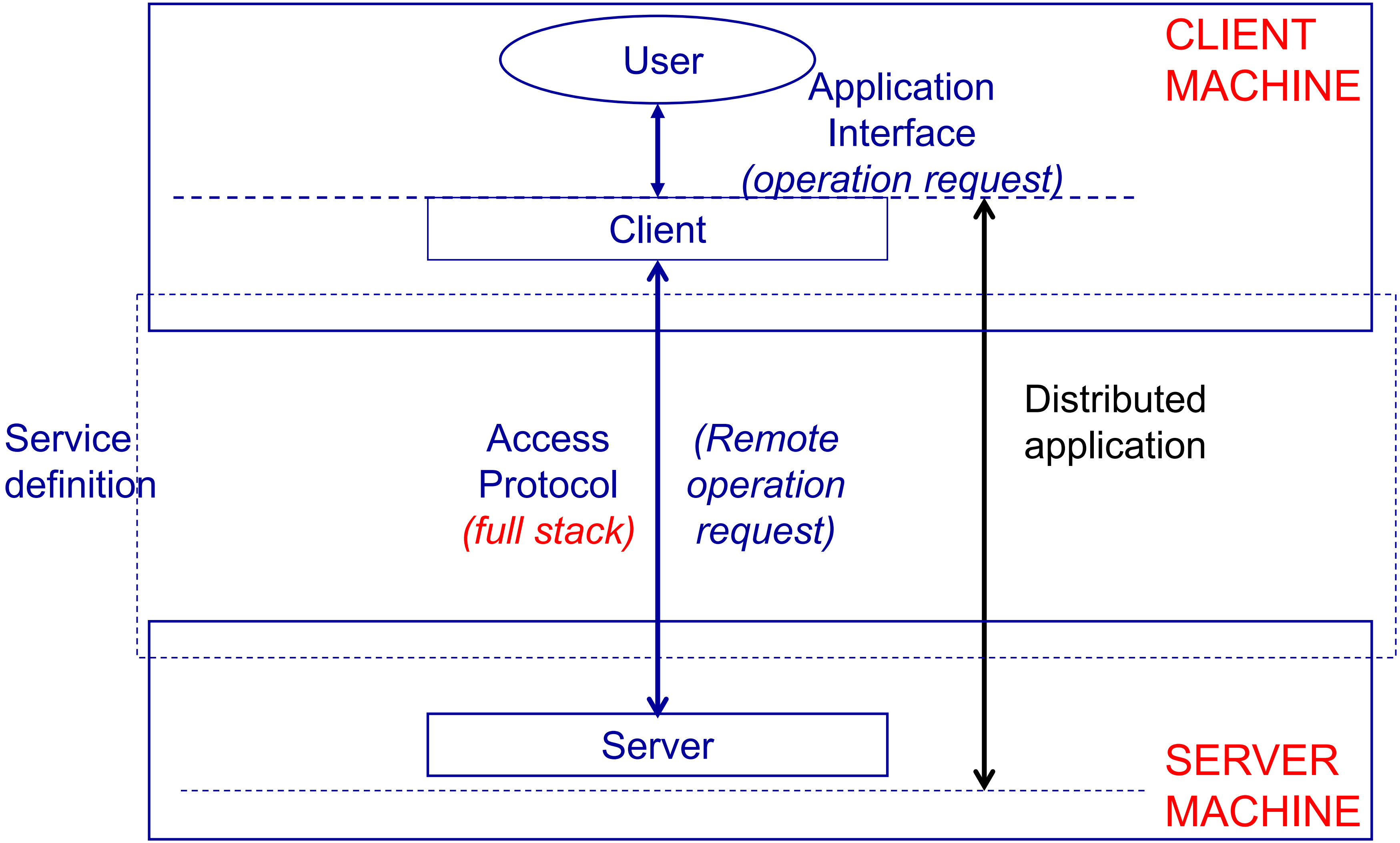
# Remote Procedure Call



# Remote operations

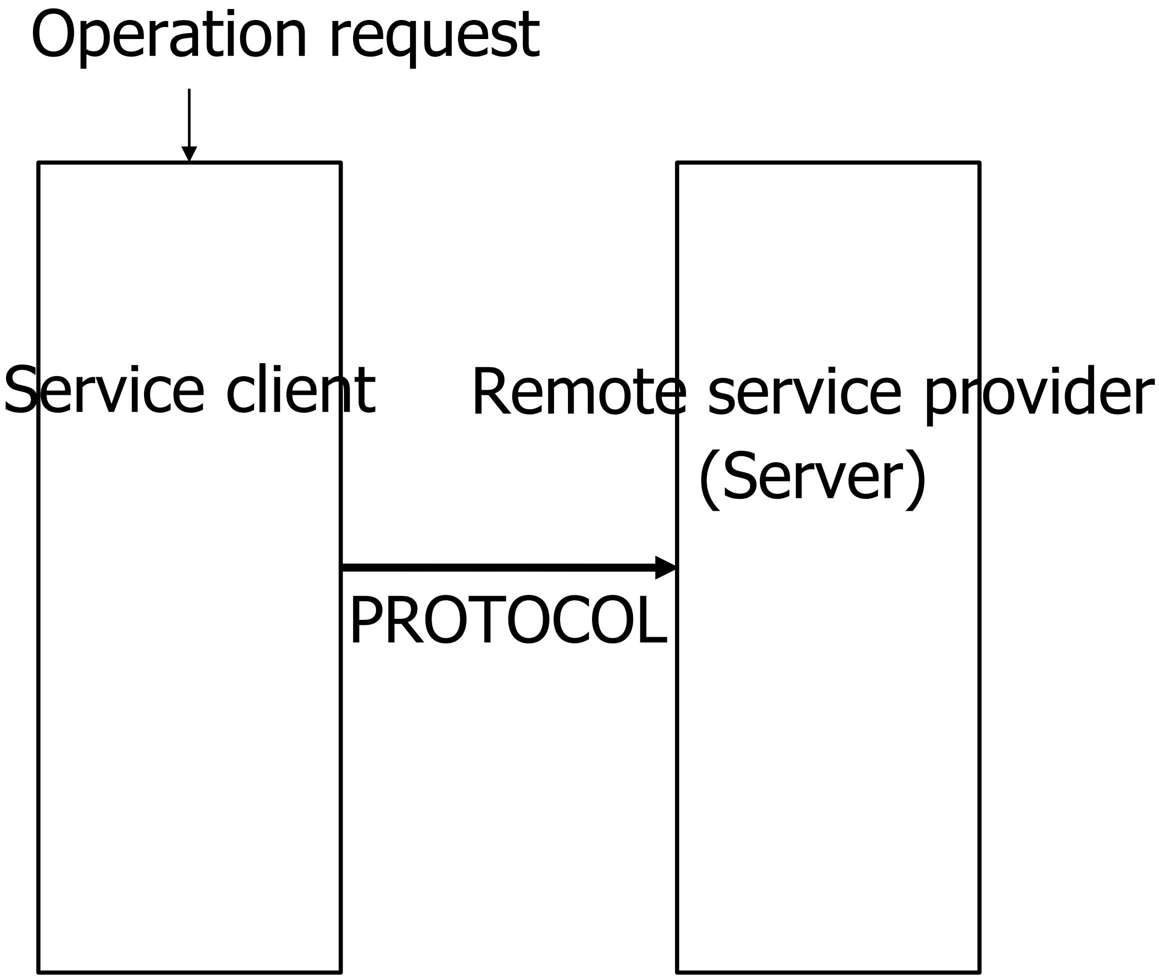
- Remote operations
- Remote invocation
- RPC (Remote Procedure Call)
- …

# Client / Server model



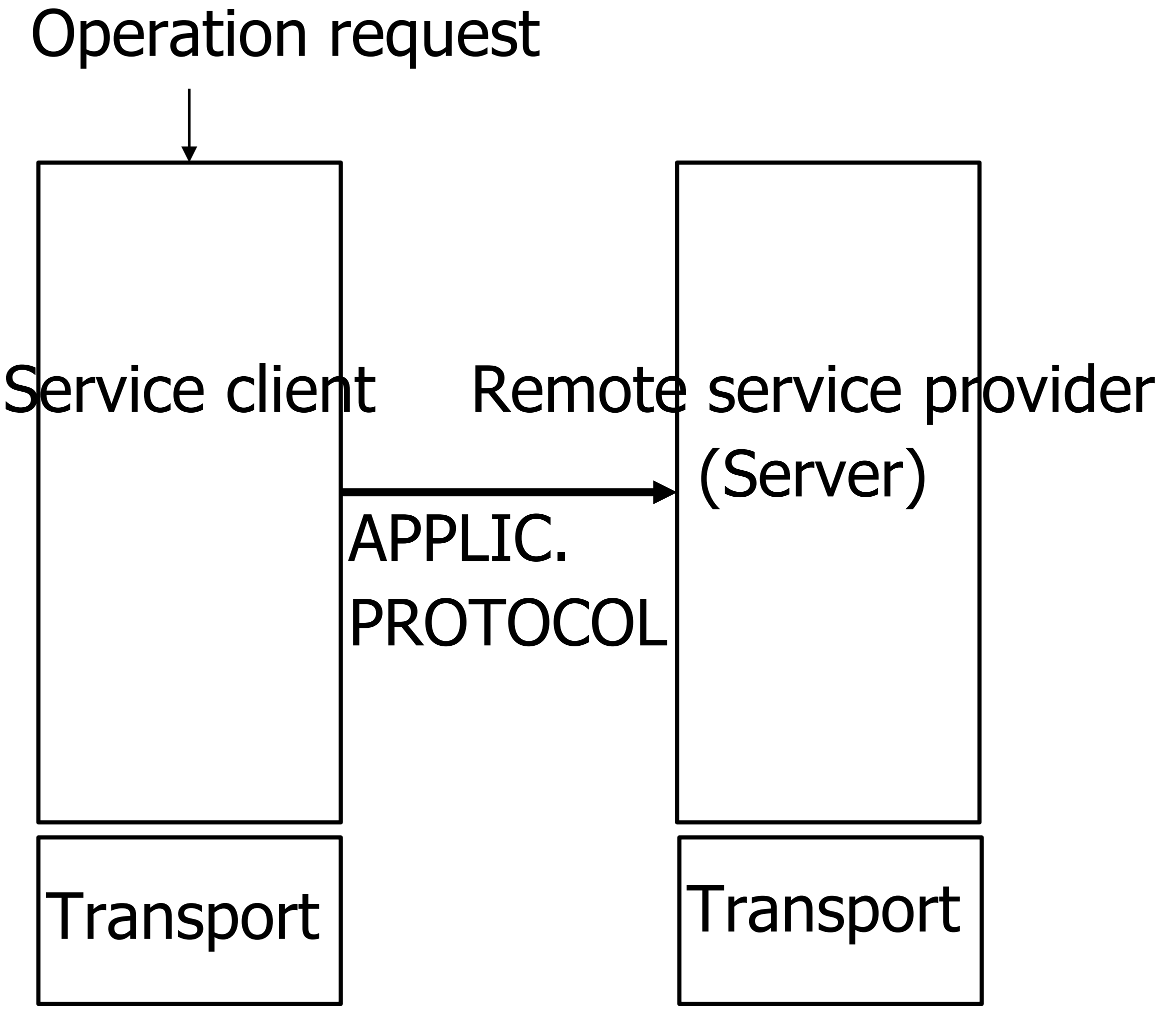
# Distributed applications

**DISTRIBUTED**



# Distributed applications

## DISTRIBUTED – Protocol stack


Operation request

Service client

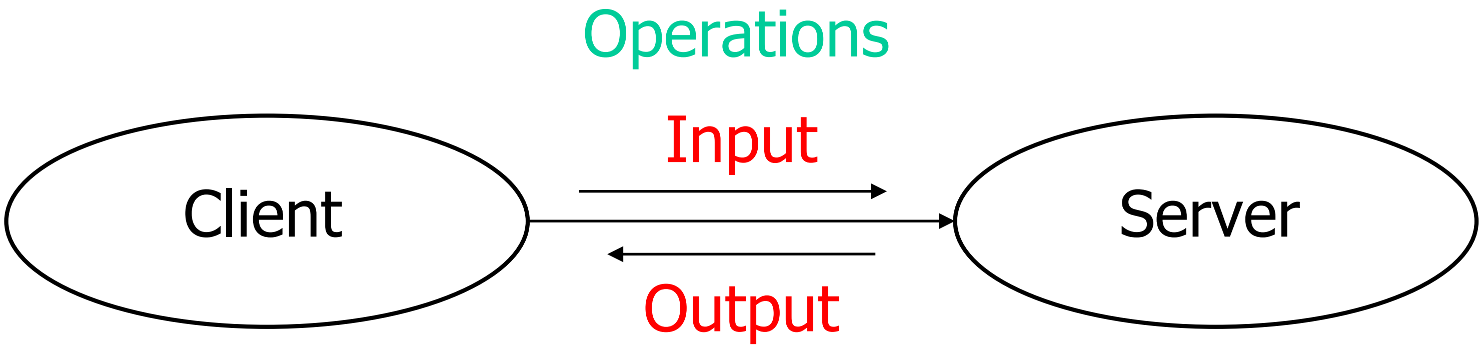
Remote service provider (Server)

APPLIC. PROTOCOL

Transport

Transport

# Distributed applications & Services

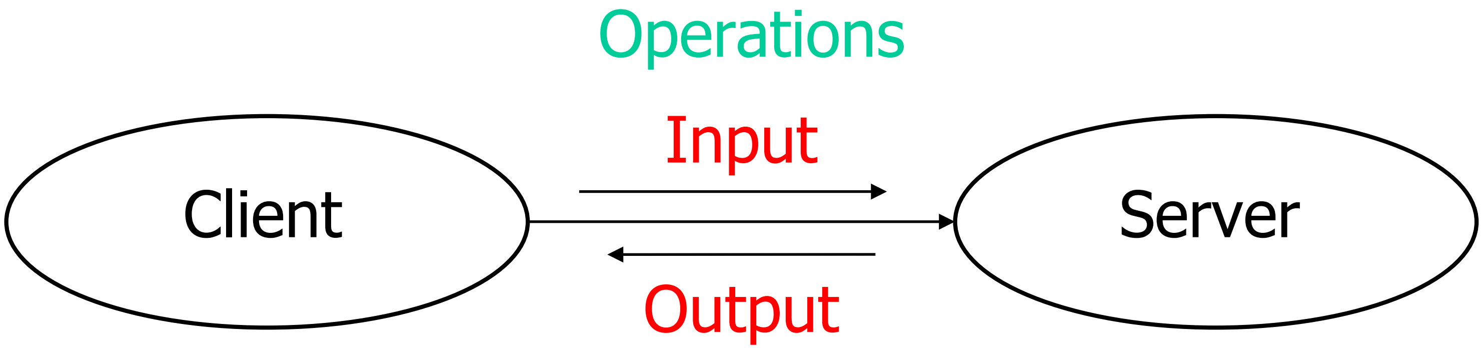


# Distributed applications & Services

- *Remote operations → Services*

- Services:
  - Entities/Modules & Relationships
  - Operations (procedures)
  - Data in operations:
    - Input parameters
    - Results (output parameters)

# Distributed applications & Services → WEB



- **Entities → Web servers/clients**
- **Relationships (Protocols) → HTTP**
- **Representation of the information → XML**

# Web services & technologies

## Modelling:

- Architecture
- Service / Operations
- Data

– *Several valid approaches depending on requirements*

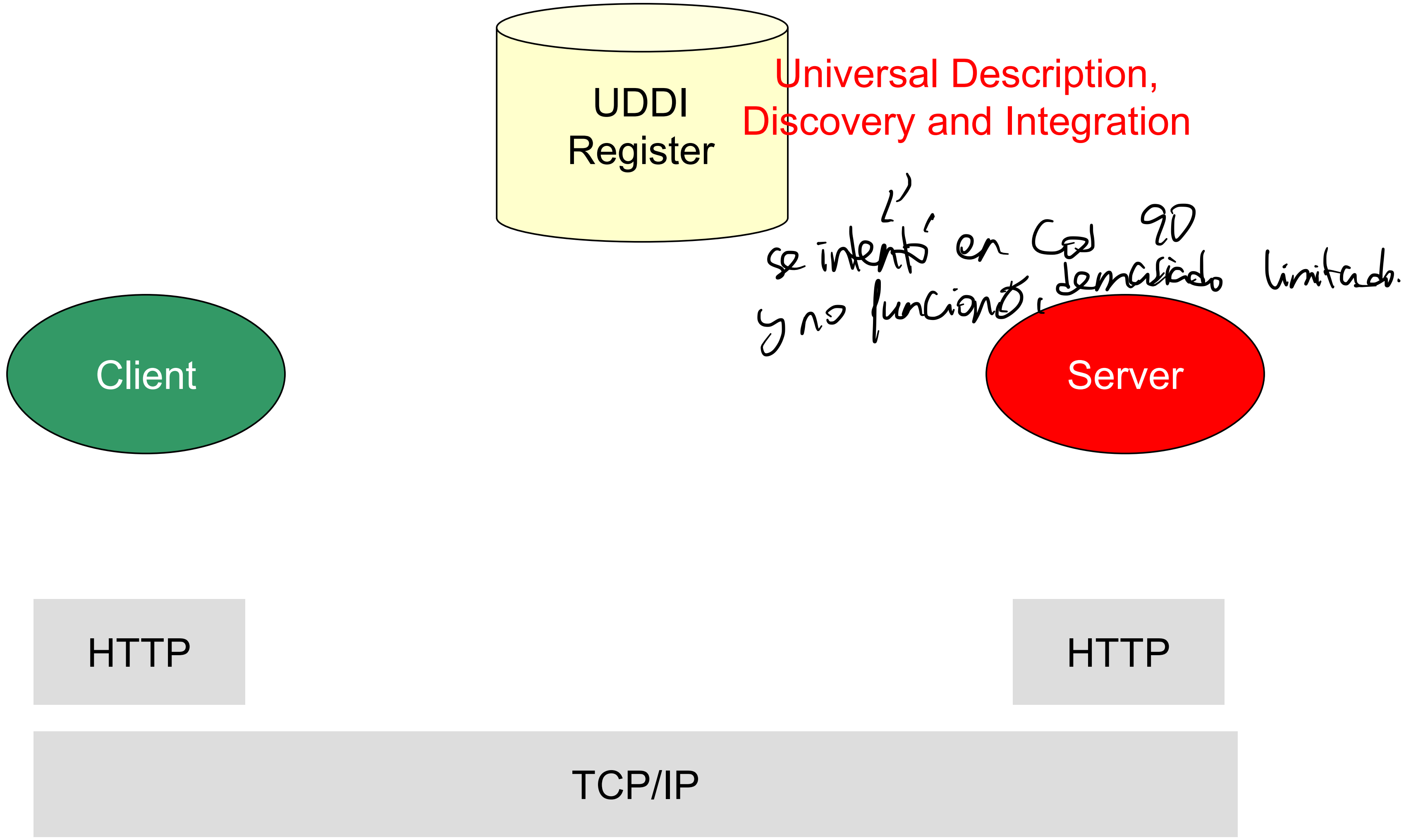
## Implementation:

- Service specification

  esqema XML

  - WSDL (Web Services Description Language)
  - WADL (Web Application Description Language) ## *less used* ##
- Messages interchange
  - SOAP (Simple Object Access Protocol)
- "Just" HTTP Request/Response
  - REST (REpresentational State Transfer)

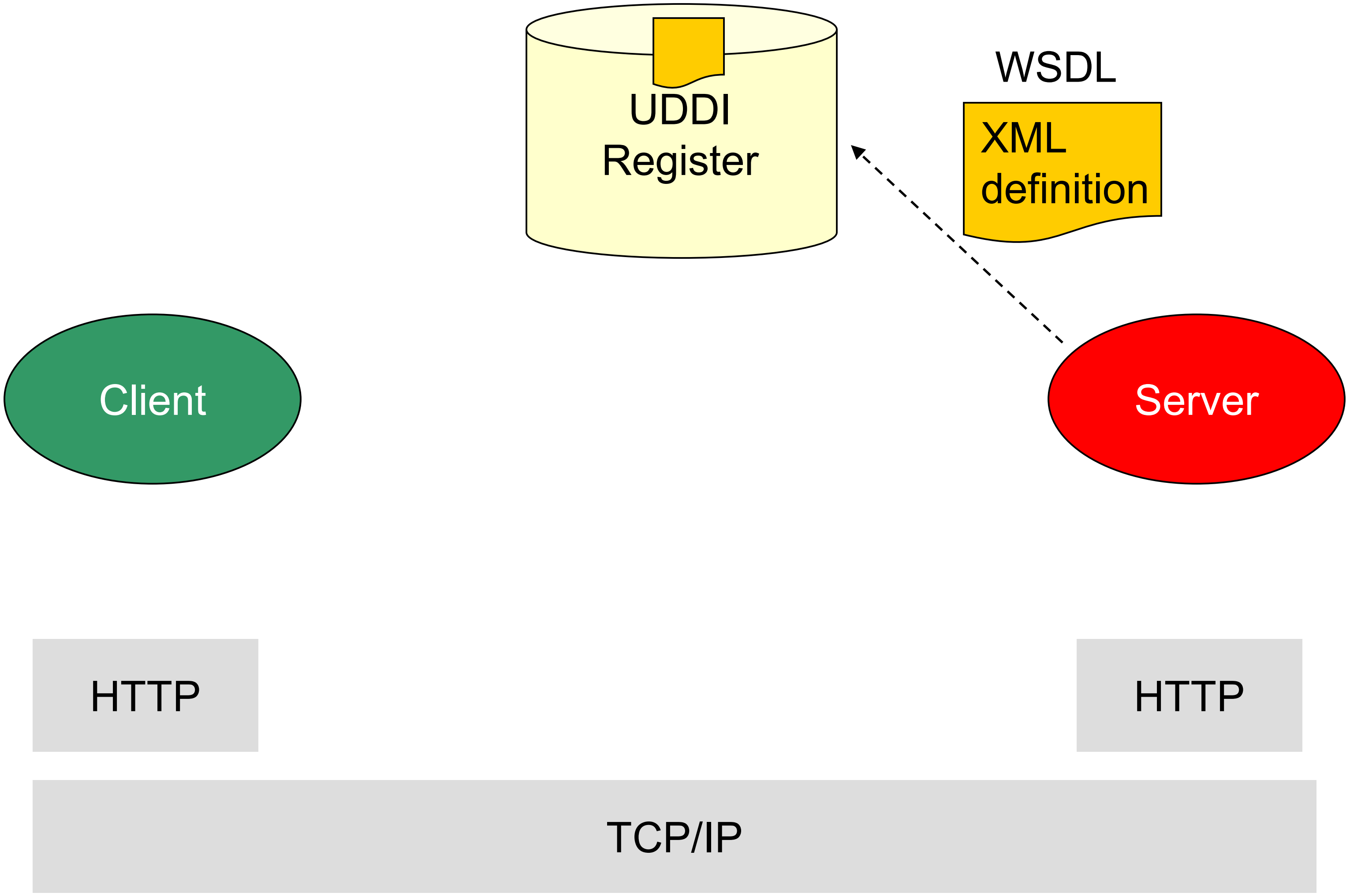
# A general model



Universal Description, Discovery and Integration

se intentó en Col 90 y no funcionó, demasiado limitado.

# A general model



UDDI Register

WSDL

XML definition

Client

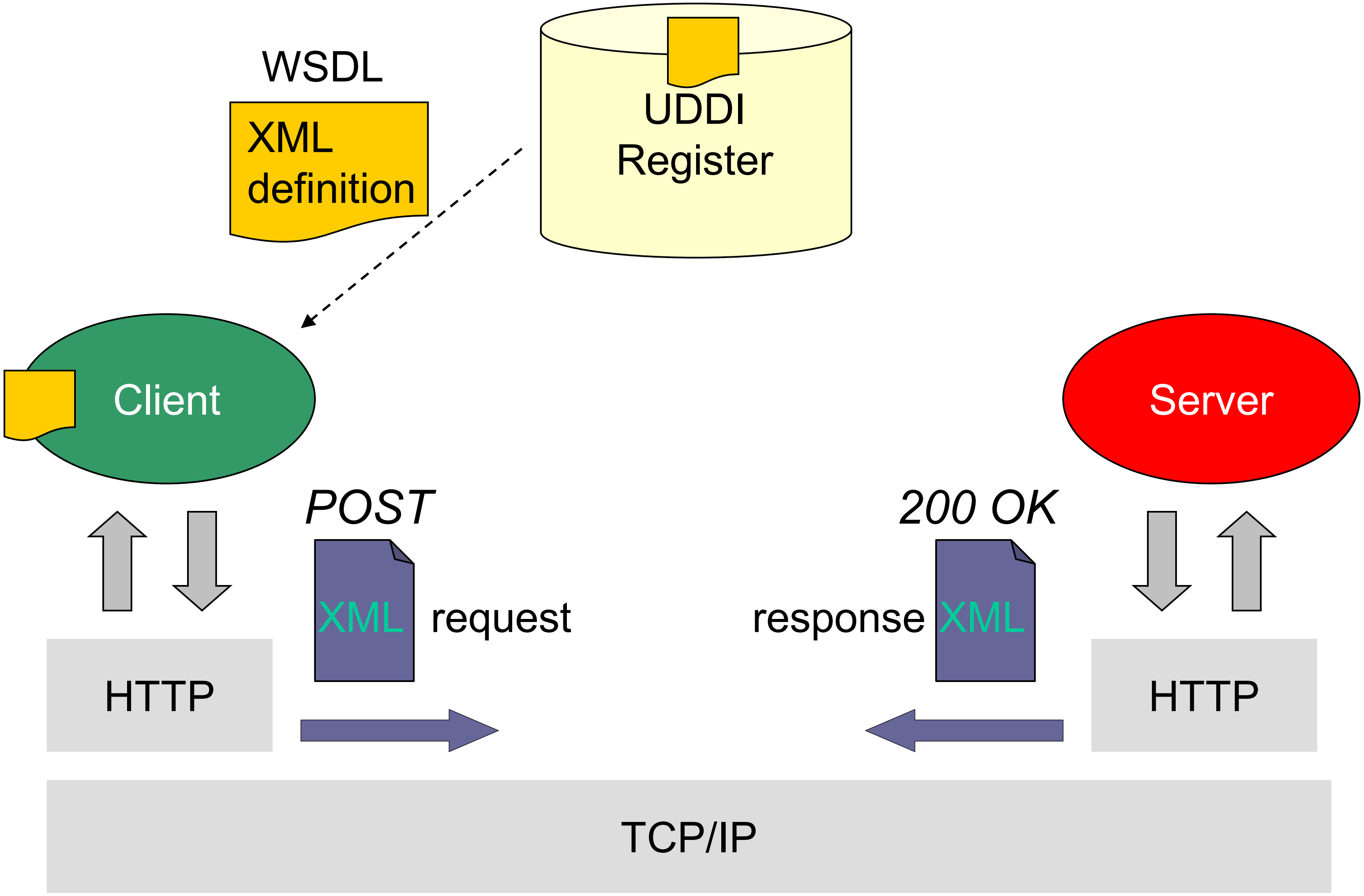
Server

HTTP

HTTP

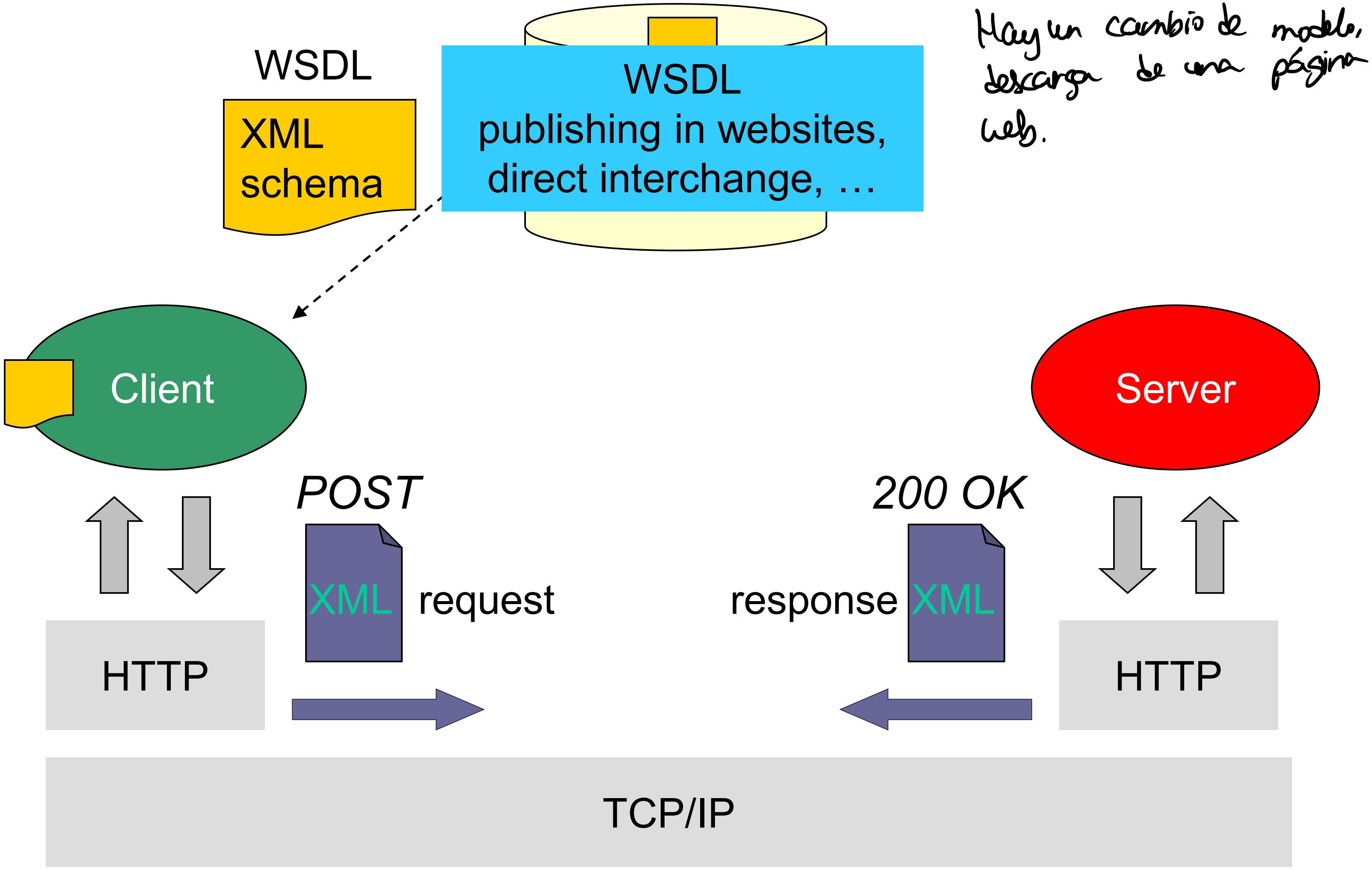
TCP/IP

# A general model

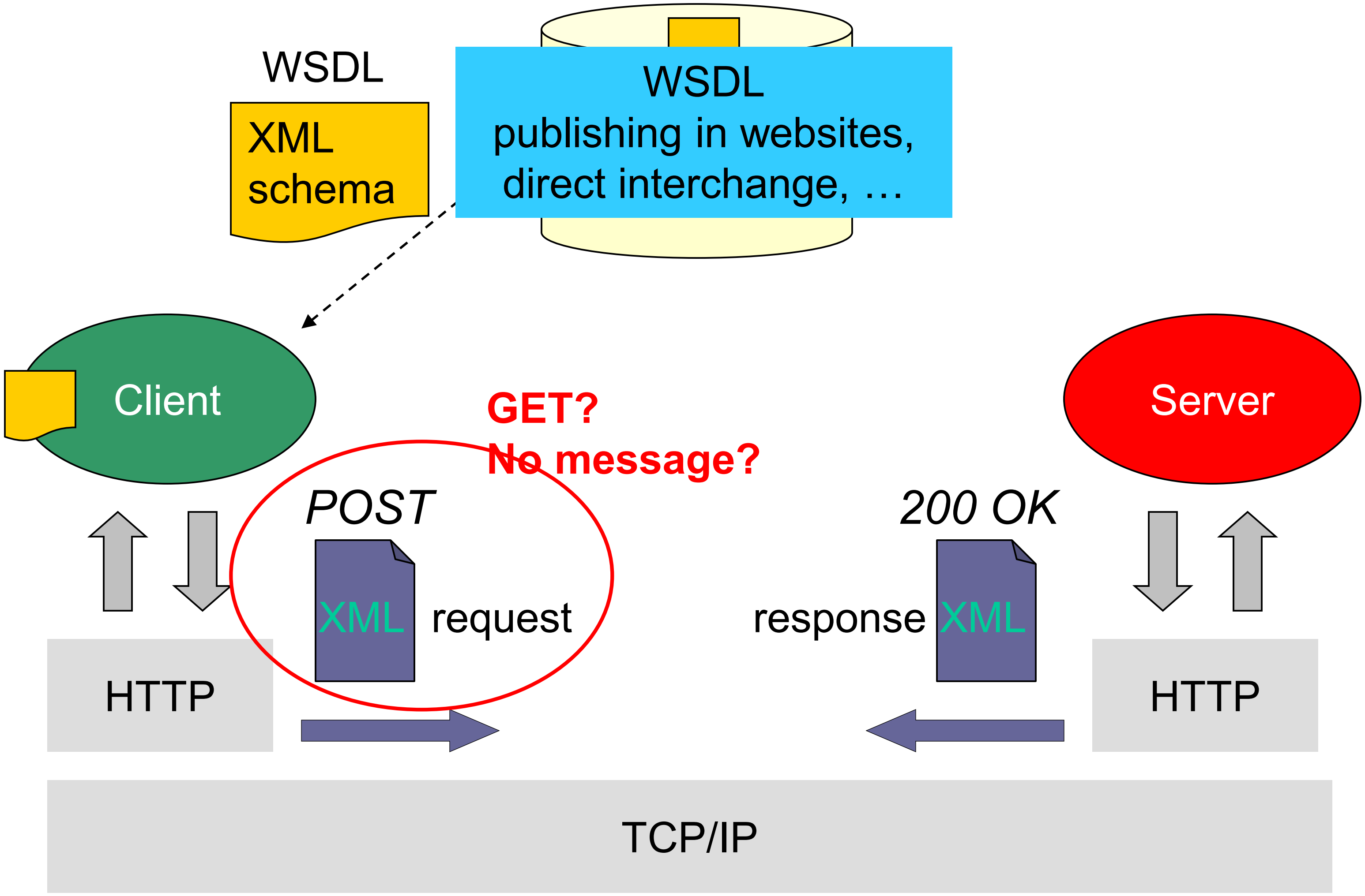


# A general model

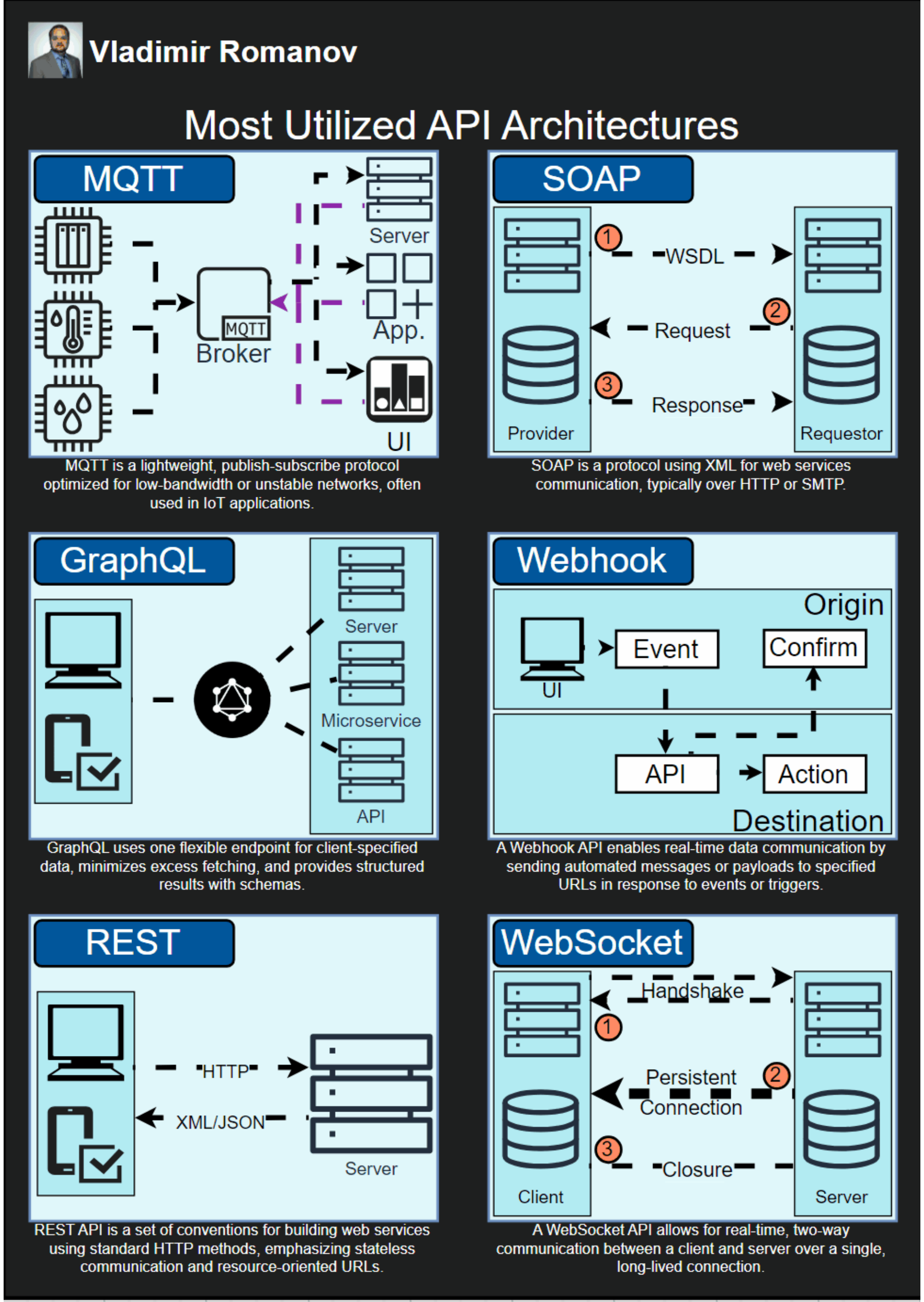
Hay un cambio de modelo, se descarga de una página web.



# A general model



# API architectures



MQTT is a lightweight, publish-subscribe protocol optimized for low-bandwidth or unstable networks, often used in IoT applications.

SOAP is a protocol using XML for web services communication, typically over HTTP or SMTP.

GraphQL uses one flexible endpoint for client-specified data, minimizes excess fetching, and provides structured results with schemas.

A Webhook API enables real-time data communication by sending automated messages or payloads to specified URLs in response to events or triggers.

REST API is a set of conventions for building web services using standard HTTP methods, emphasizing stateless communication and resource-oriented URLs.

A WebSocket API allows for real-time, two-way communication between a client and server over a single, long-lived connection.

Source: Vladimir Romanov (in LinkedIn)

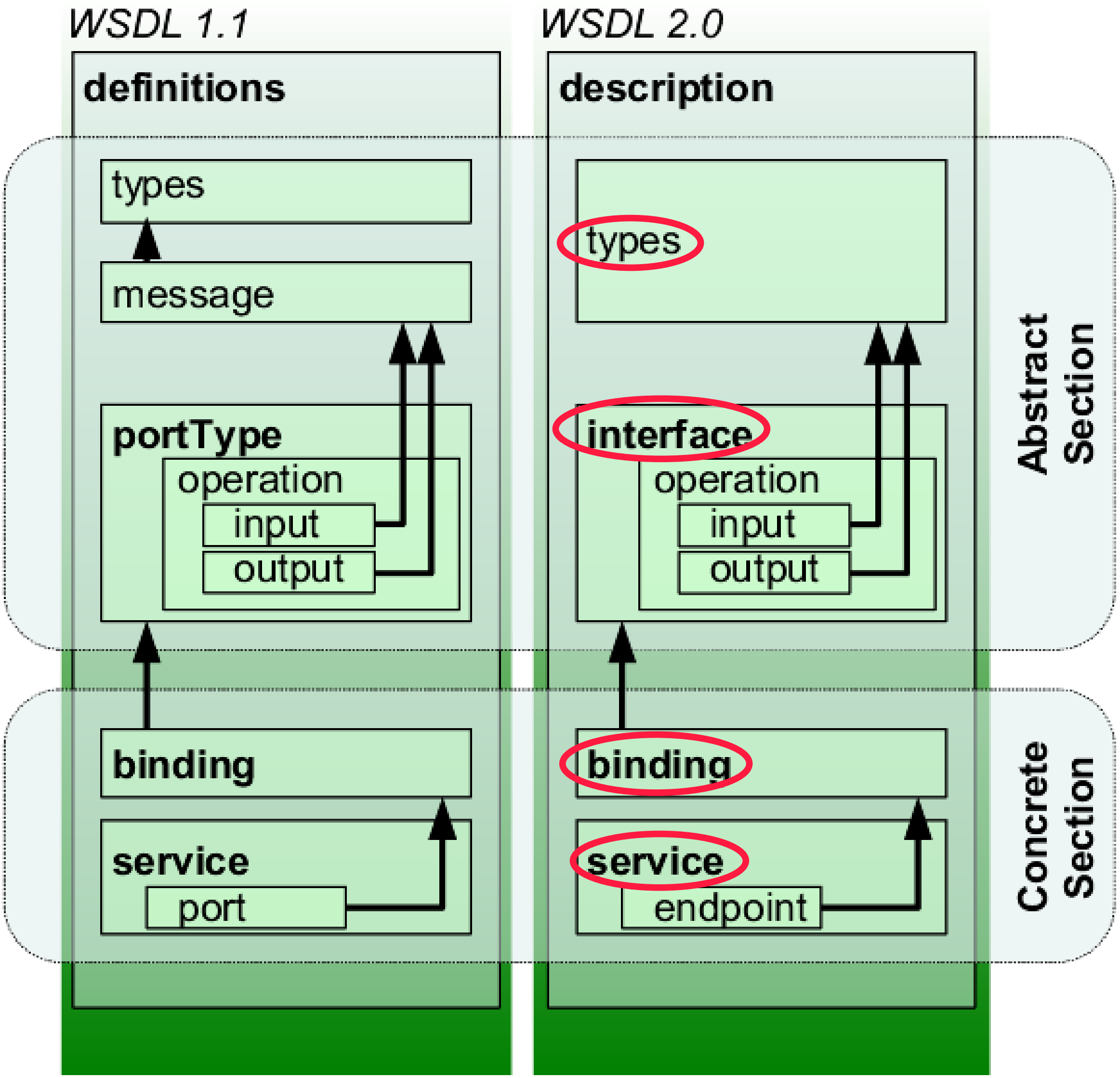
# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - **WSDL** / (WADL)
  - (SOAP)
  - REST
- A real example

# WSDL

- Web Services Description Language (2.0)
- Web Services Definition Language (1.1)

Define cómo se implementan los datos y operaciones.



# WSDL

## INTERFACE DEFINITION (Operations):

```
<interface name="glossaryTerms">
  <operation name="getTerm">
    <input element="getTermRequest"/>
    <output element="getTermResponse"/>
  </operation>
</interface>
```

## TYPES (Parameters):

```
<element name="getTermRequest">
  <element name="term" type="xs:string"/>
</element>

<element name="getTermResponse">
  <element name="value" type="xs:string"/>
</element>
```

# WSDL – binding example

## Implementation of Operations:

```
<binding
        name="glossaryTermsBinding"
        interface="glossaryTerms"
        type="http://www.w3.org/ns/wsdl/soap"
        version="1.1"
        protocol="http://www.w3.org/2006/01/soap11/
                                        bindings/HTTP/">

  <operation
        action="http://example.com/getTerm"/>
</binding>
```

se comunicarán por SOAP, los mensajes se enviarán por este protocolo

Construimos el mensaje en SOAP, en la ida empaqueta el mensaje en SOAP que se envía por HTTP.

# WSDL complete example (1/5)

```
<?xml version="1.0"?>
<wsdl:description
            name="StockQuote"
            xmlns:wsdl=“http://www.w3.org/ns/wsdl”
            targetNamespace="http://example.com/stockquote"
            xmlns:tns="http://example.com/stockquote"
            xmlns:wsoap="http://www.w3.org/ns/wsdl/soap">
```

# WSDL complete example (2/5)

```
<wsdl:types>
  <xs:schema
          targetNamespace="http://example.com/stockquote"
          xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="TradePriceRequest">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockName" type=“xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="TradePriceResponse">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockPrice" type="float"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
```

# WSDL complete example (2/5)

```
<wsdl:types>
  <xs:schema
         targetNamespace="http://example.com/stockquote"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="TradePriceRequest">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockName" type=“xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="TradePriceResponse">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockPrice" type="float"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
```

Prefixed **tns** later

# WSDL complete example (2/5)

```
<wsdl:types>
  <xs:schema
        targetNamespace="http://example.com/stockquote"
        xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="TradePriceRequest">
      <xs:complexType>
        <xs:all>
          <xs:element name="stockName" type="xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="TradePriceResponse">
      <xs:complexType>
        <xs:all>
          <xs:element name="stockPrice" type="float"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
```

A diferencia de antes, estamos creando un esquema definiendo directamente el tipo de los valores. Eso es para poder reutilizar la definición

all : desordenado

sequence : ordenado

**All**: Child elements can appear in any order (not in **sequence**)

# WSDL complete example (2/5)

```
<wsdl:types>
  <xs:schema
         targetNamespace="http://example.com/stockquote"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="TradePriceRequest">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockName" type=“xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="TradePriceResponse">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockPrice" type="float"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
```

# WSDL complete example (3/5)

```
<wsdl:interface
        name="StockQuoteInterface">
    <wsdl:operation name="TradePrice“
        pattern="http://www.w3.org/ns/wsdl/in-out">
        <wsdl:input element="tns:TradePriceRequest"/>
        <wsdl:output element="tns:TradePriceResponse"/>
    </wsdl:operation>
</wsdl:interface>
```

Cuando no necesitamos recibir información
Si hay un error no lo sabemos, pero añadir respuesta genera
overhead ←

*Message Exchange Patterns (WSDL)*: In-Only, Robust In-Only, In-Out.

Robusto a errores, responde si ↙
ha habido un error.

# WSDL complete example (4/5)

```
<wsdl:binding
        name="StockQuoteBinding"
        interface="tns:StockQuoteInterface"
        type="http://www.w3.org/ns/wsdl/soap"
        wsoap:version="1.1"
        wsoap:protocol=
                "http://www.w3.org/2006/01/soap11/bindings/HTTP/">

    <wsdl:operation
            ref="tns:TradePrice"
            wsoap:action="http://example.com/TradePrice"/>

</wsdl:binding>
```

# WSDL complete example (4/5)

```
<wsdl:binding
        name="StockQuoteBinding"
        interface="tns:StockQuoteInterface"
        type="http://www.w3.org/ns/wsdl/soap"
        wsoap:version="1.1"
        wsoap:protocol=
                    "http://www.w3.org/2006/01/soap11/bindings/HTTP/">

    <wsdl:operation
                ref="tns:TradePrice"
                wsoap:action="http://example.com/TradePrice"/>

</wsdl:binding>
```

**tns** prefix used for completeness
(useful when `import`)

# WSDL complete example (5/5)

```
 <wsdl:service
          name="StockQuoteService“
          interface="tns:StockQuoteInterface">

      <wsdl:documentation>My first service</wsdl:documentation>

      <wsdl:endpoint
                     name="StockQuoteEndPoint“
                     binding="tns:StockQuoteBinding“
                     address="http://example.com/endpoint/stockquote"/>

</wsdl:service>


</wsdl:description>
```

# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - WSDL / (**WADL**)
  - (SOAP)
  - REST
- A real example

# WADL

- **Web Application Description Language**

- W3C Member Submission (Sun Microsystems) 31 August 2009

- https://www.w3.org/Submission/wadl/

# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - WSDL / (WADL)
  - (**SOAP**)
  - REST
- A real example

# Web services & technologies

**Modelling:**

- Architecture
- Service / Operations
- Data

– *Several valid approaches depending on requirements*

**Implementation:**

- Service specification
  - WSDL (Web Services Description Language)
  - WADL (Web Application Description Language) ## *less used* ##
- Messages interchange
  - SOAP (Simple Object Access Protocol)
- "Just" HTTP Request/Response
  - REST (REpresentational State Transfer)

# Web services & technologies

**Modelling:**

- Architecture
- Service / Operations
- Data

– *Several valid approaches depending on requirements*

**Implementation:**

- Service specification
  - WSDL (Web Services Description Language)
  - WADL (Web Application Description Language) ## *less used* ##
- Messages interchange
  - SOAP (Simple Object Access Protocol)
- "Just" HTTP Request/Response
  - REST (REpresentational State Transfer)

# SOAP

- Simple Object Access Protocol (Version 1.1, W3C Note, 2000)
- SOAP
  (Version 1.2, W3C Recommendation, 2007)

Capa presente entre layer de aplicación.

# SOAP

```
<?xml version="1.0"?>
<env:Envelope
      xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
      ...
    </env:Header>
    <env:Body>
      ...
       <env:Fault>
          ...
       </env:Fault>
    </env:Body>
</env:Envelope>
```

informar de errores

# SOAP

```
<?xml version="1.0"?>
<env:Envelope
      xmlns:env="http://www.w3.org/2003/05/soap-envelope">
    <env:Header>
       ...
    </env:Header>
    <env:Body>
       ...
        <env:Fault>
           ...
        </env:Fault>
    </env:Body>
</env:Envelope>
```

*Message Exchange Patterns (SOAP)*: request-response, soap-response

# WSDL complete example (2/5)

REMINDER!

```
<wsdl:types>
  <xs:schema
         targetNamespace="http://example.com/stockquote"
         xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="TradePriceRequest">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockName" type=“xs:string"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
    <xs:element name="TradePriceResponse">
      <xs:complexType>
        <xs:all>
          <xs:element name=“stockPrice" type="float"/>
        </xs:all>
      </xs:complexType>
    </xs:element>
  </xs:schema>
</wsdl:types>
```

# SOAP request message example: “stockquote”

```
<?xml version="1.0"?>
<env:Envelope
       xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header> </envHeader>
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceRequest
      <m:StockName>IBM</m:StockName>
    </m:TradePriceRequest>
  </env:Body>
</env:Envelope>
```

# HTTP SOAP Request example

```
POST /stockquote HTTP/1.1
Host: example.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 262
SOAPAction: “http://www.w3.org/2003/05/soap-envelope”

<?xml version="1.0"?>
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header> </envHeader>
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceRequest
      <m:StockName>IBM</m:StockName>
    </m:TradePriceRequest>
  </env:Body>
</env:Envelope>
```

# HTTP SOAP Request example

```
POST /stockquote HTTP/1.1
Host: example.com
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 262
SOAPAction: “http://www.w3.org/2003/05/soap-envelope”

<?xml version="1.0"?>
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header> </envHeader>
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceRequest
      <m:StockName>IBM</m:StockName>
    </m:TradePriceRequest>
  </env:Body>
</env:Envelope>
```

HTTP body

# HTTP SOAP Response example

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 242

<?xml version="1.0"?>
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceResponse>
      <m:StockPrice>34.5</m:StockPrice>
    </m:TradePriceResponse>
  </env:Body>
</env:Envelope>
```

# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - WSDL / (WADL)
  - (SOAP)
  - **REST**
- A real example

# Web services & technologies

## Modelling:

- Architecture
- Service / Operations
- Data

– *Several valid approaches depending on requirements*

## Implementation:

- Service specification
  - WSDL (Web Services Description Language)
  - WADL (Web Application Description Language) ## *less used* ##
- Messages interchange
  - SOAP (Simple Object Access Protocol)
- "Just" HTTP Request/Response
  - REST (REpresentational State Transfer)

# SOAP vs. REST

- SOAP defines complex structures to implement the calls.
- In REST, they are not necessary.

- Example:

# SOAP vs. REST - Example

- SOAP Request, sent with HTTP POST (in the HTTP body)

```
<?xml version="1.0"?
<env:Envelope
          xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header> </envHeader>
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceRequest
      <m:StockName>IBM</m:StockName>
    </m:TradePriceRequest>
  </env:Body>
</env:Envelope>
```

- REST Request, sent with HTTP GET (in the HTTP header)

  http://example.com/stockquote/**TradePriceRequest/IBM**

# REST example

## Request:

```
GET /stockquote/TradePriceRequest/IBM HTTP/1.1
Host: example.com
Accept: text/xml
Accept-Charset: utf-8
```

# REST example

**Request:**

```
GET /stockquote/TradePriceRequest/IBM HTTP/1.1
Host: example.com
Accept: text/xml
Accept-Charset: utf-8
```

Use of the “Request line” of the HTTP header

# REST example

**Request:**

Use of the “Request line” of the HTTP header

```
GET /stockquote/TradePriceRequest/IBM HTTP/1.1
Host: example.com
Accept: text/xml
Accept-Charset: utf-8
```

Use of the `Path` (and even `Query` and `Fragment`) part of the URL

# REST example

## Response (with SOAP):

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 242

<?xml version="1.0"?>
<env:Envelope
        xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body xmlns:m="http://example.com/stockquote">
    <m:TradePriceResponse>
      <m:StockPrice>34.5</m:StockPrice>
    </m:TradePriceResponse>
  </env:Body>
</env:Envelope>
```

# REST example

## Response (simplified)

**(with XML, but JSON, for example, could be used):**

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 139

<?xml version="1.0"?>
<m:Quote xmlns:m="http://www.example.org/stock">
    <m:StockName>IBM</m:StockName>
    <m:Price>34.5</m:Price>
</m:Quote>
```

# REST example

## Response (even more simplified!):

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 139

34.5
```

# Complex REST requests

- More complex REST request, also sent with GET:

  ```
  http://www.acme.com/phonebook/UserDetails?
  firstName=John&lastName=Doe
  ```

- HTTP methods to implement CRUD operations (Create/Read/Update/Delete):
  - **Create** – HTTP POST method
  - **Read** – HTTP GET method (or POST if the parameters are complex and should travel inside the request message)
  - **Update** – HTTP POST method
  - **Delete** – HTTP POST method

*ALTERNATIVELY:* **PUT, PATCH, DELETE** (for CUD)

# REST with body in the HTTP request

- If parameters too complex or too long for the “Request line” →
  Better to use POST with parameters in the body
- Use of content subtype

  `application/x-www-form-urlencoded`

  (specified in the HTML standard)

# REST resources

- Identified by URL’s
- Accessible
- Resources are key for a correct REST design
  - SOAP: Remote operations
    - GetProductDescription or GetProductPrice
  - REST: Simpler requests
    - Product?field=Description or Product?field=Price
  - Not really complex resources
  - Use links to return additional information

# REST service design ideas

- Format of request/response to be documented and not changed

  → **Specify service details!** (WSDL, *WADL*, …)

Es para lo mismo que JSON, para hacer cosas más complicadas falta formalizarlo un poco más para asegurarse de que va a funcionar y que sea interoperable.

# Web services - Content

- Distributed applications. “Remote operations”
- Web services
  - Introduction
  - Model
  - WSDL / (WADL)
  - (SOAP)
  - REST
- **A real example**

http://openweathermap.org/current

# Web services example - REST

- **API call:**

api.openweathermap.org/data/2.5/weather?q={city name}

api.openweathermap.org/data/2.5/weather?q={city name},
{country code}

- **Parameters:**

**q** city name and country code divided by comma,
use ISO 3166 country codes

- **Examples of API calls:**

api.openweathermap.org/data/2.5/weather?q=London

api.openweathermap.org/data/2.5/weather?q=London,uk http://openweathermap.org/current

# Web services example - REST

- **Request:**

api.openweathermap.org/data/2.5/weather?q=London,uk

http://api.openweathermap.org/data/2.5/weather?q=London,uk&appid=44db6a862fba0b067b1930da0d769e98

- **Response:**

{"coord":{"lon":-0.13,"lat":51.51},"weather":
[{"id":801,"main":"Clouds","description":"few clouds","icon":"02n"}],
"base":"cmc stations","main":{"temp":285.17,"pressure":1018,
            "humidity":66,"temp_min":283.85,"temp_max":286.15},
"wind":{"speed":9.8,"deg":230,"gust":17.5},"clouds":{"all":20},
"dt":1453744044,"sys":{"type":1,"id":5091,"message":0.0102,
       "country":"GB","sunrise":1453708113,"sunset":1453739867},
"id":2643743,"name":"London","cod":200}

# Epilogue …

# Evolution of technologies

- RPCs (initially C language)
- RMI, Remote Method Invocation (Object Oriented RPC, Java)
- CORBA, Common Object Request Broker Architecture (Independent of programming technology)
- *SOAP, Simple Object Access Protocol (XML messages)*
- ***REST, REpresentational State Transfer ("no" messages)***

# Evolution of technologies

- RPCs (initially C language)
- RMI, Remote Method Invocation (Object Oriented RPC, Java)
- CORBA, Common Object Request Broker Architecture (Independent of programming technology)
- *SOAP, Simple Object Access Protocol (XML messages)*
- ***REST, REpresentational State Transfer (“no” messages)***

- ***IS THERE SOMETHING ELSE?***

# Evolution of technologies

- RPCs (initially C language)
- RMI, Remote Method Invocation (Object Oriented RPC, Java)
- CORBA, Common Object Request Broker Architecture (Independent of programming technology)
- *SOAP, Simple Object Access Protocol (XML messages)*
- ***REST, REpresentational State Transfer ("no" messages)***

- ***IS THERE SOMETHING ELSE?***
  - ***Of course! Is it better?***

# Technologies for distributed applications development

- **Models**:
  - based on WSDL, SOAP, REST, …
- **Languages:**
  - Java, JavaScript, JSPs, Servlets, ASPs, PHP, CGI, C#, Ruby, Perl, Python, ColdFusion, Go, …
-
  -
-
  -
-
  -
-
  -
-
  -

# Technologies for distributed applications development

- **Models**:
  - based on WSDL, SOAP, REST, …
- **Languages:**
  - Java, JavaScript, JSPs, Servlets, ASPs, PHP, CGI, C#, Ruby, Perl, Python, ColdFusion, Go, …
- **Databases:**
  - SQL, non-SQL (MongoDB, Cassandra, CouchDB, …)
- **Software stacks**:
  - Java, JavaScript, MEAN, …

# Technologies for distributed applications development

- **Models**:
  - based on WSDL, SOAP, REST, …
- **Languages:**
  - Java, JavaScript, JSPs, Servlets, ASPs, PHP, CGI, C#, Ruby, Perl, Python, ColdFusion, Go, …
- **Databases:**
  - SQL, non-SQL (MongoDB, Cassandra, CouchDB, …)
- **Software stacks**:
  - Java, JavaScript, MEAN, …
- **Frameworks (libraries …):**
  - Spring (Java), AngularJS (JavaScript), .net (Microsoft), …
- **IDEs (Integrated Development Environments):**
  - Netbeans, IntelliJ IDEA (Java), Webstorm (JavaScript), …
- **Runtime environments:**
  - Java, node.js, …

# Technologies for distributed applications development

- **Models**:
  - based on WSDL, SOAP, REST, …
- **Languages:**
  - **Java**, **JavaScript**, JSPs, Servlets, ASPs, **PHP**, CGI, **C#**, **Ruby**, Perl, **Python**, ColdFusion, **Go (Golang)**, …
- **Databases:**
  - SQL, non-SQL (MongoDB,
- **Software stacks**:
  - Java, JavaScript, MEAN, …
- **Frameworks (libraries …):**
  - Spring (Java), AngularJS (J
- **IDEs (Integrated Development Environments):**
  - Netbeans, IntelliJ IDEA (Java), Webstorm (JavaScript), …
- **Runtime environments:**
  - Java, **node.js**, …

Example of APIs offered by a streaming protocol software provider

# Technologies for distributed applications development

- **Models**:
  - based on WSDL, SOAP, REST, …
- **Languages:**
  - Java, JavaScript, JSPs, Servlets, ASPs, PHP, CGI, C#, Ruby, Perl, Python, ColdFusion, Go, …
- **Databases:**
  - SQL, non-SQL (MongoDB, Cassandra, CouchDB, …)
- **Software stacks**:
  - Java, JavaScript, MEAN, …
- **Frameworks (libraries …):**
  - Spring (Java), AngularJS (JavaScript), .net (Microsoft), …
- **IDEs (Integrated Development Environments):**
  - Netbeans, IntelliJ IDEA (Java), Webstorm (JavaScript), …
- **Runtime environments:**
  - Java, node.js, …

# Technologies for distributed applications development

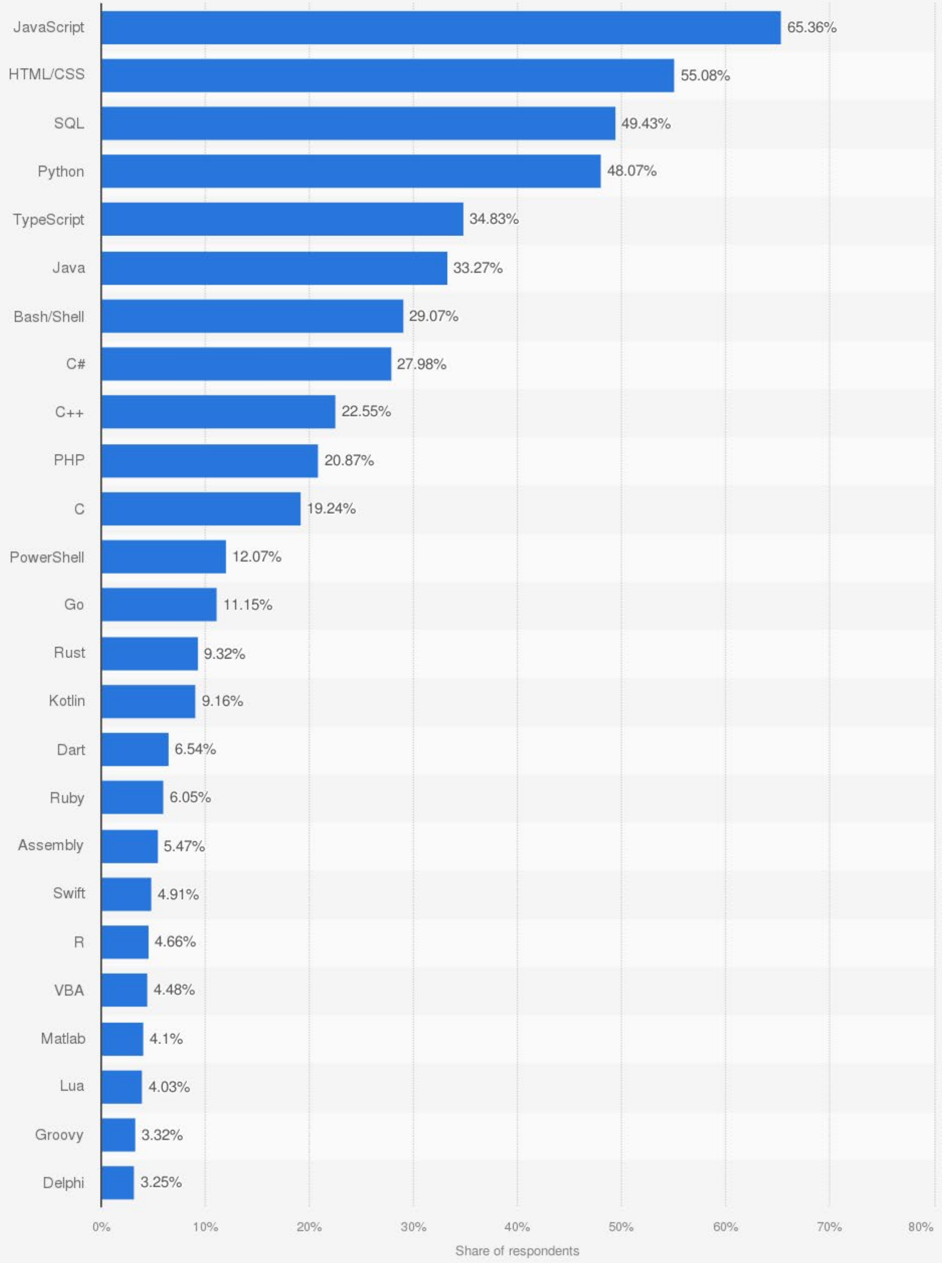
- **MEAN software stack:**
  - *MongoDB*, a NoSQL database
  - *Express.js*, a web application framework that runs on node.js
  - *Angular.js*, a JavaScript MVC framework that runs in browser JavaScript engines
  - *Node.js*, an execution environment for event-driven server-side and networking applications

# Technologies for distributed applications development

- …
- **Specific development environments for mobile applications …**
- …

# Programming languages

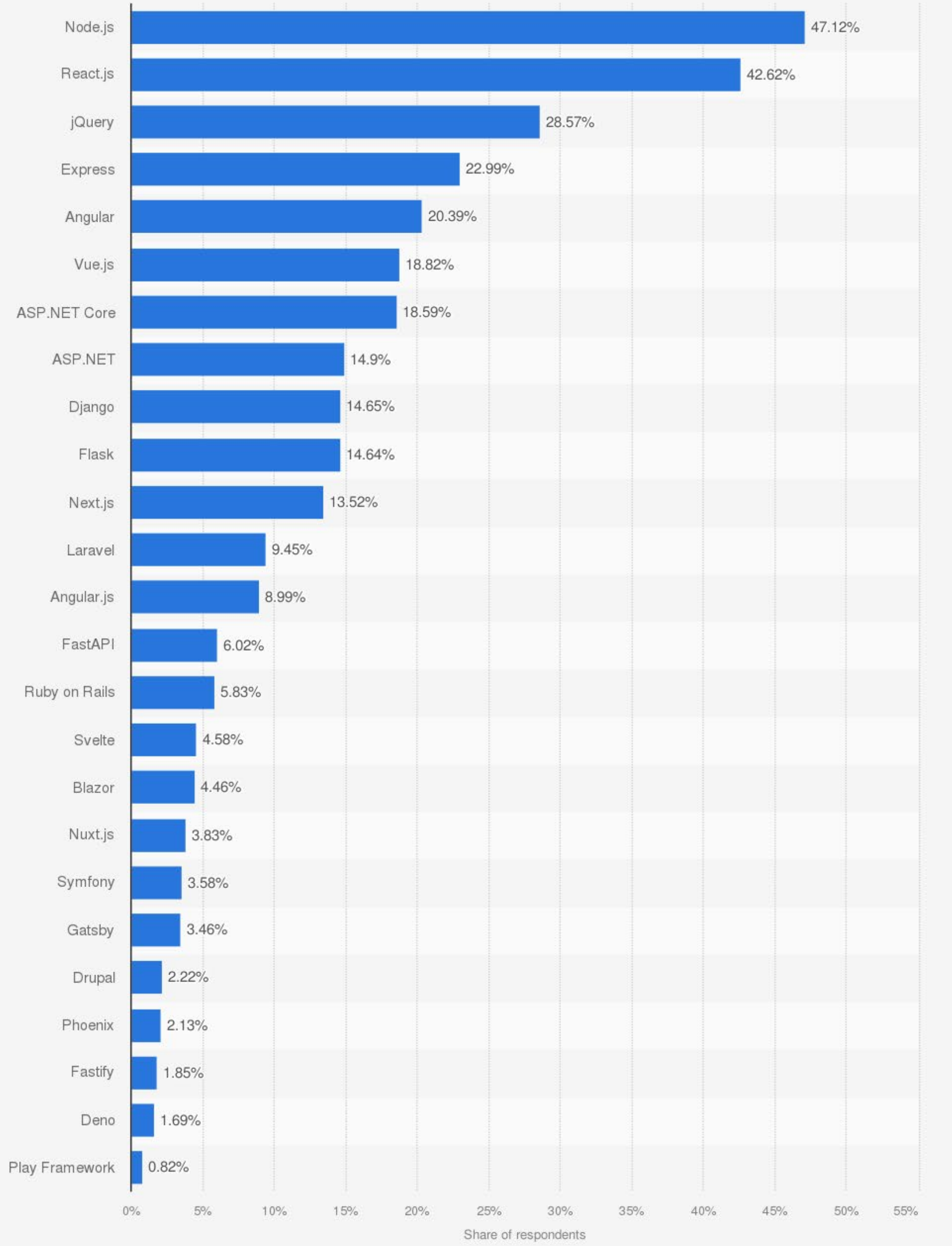
Most used programming languages among developers worldwide as of 2022



Source
Stack Overflow
© Statista 2022

Additional Information:
Worldwide; May 11 to June 1, 2022; 71,547 respondents; Software developers

# Programming frameworks

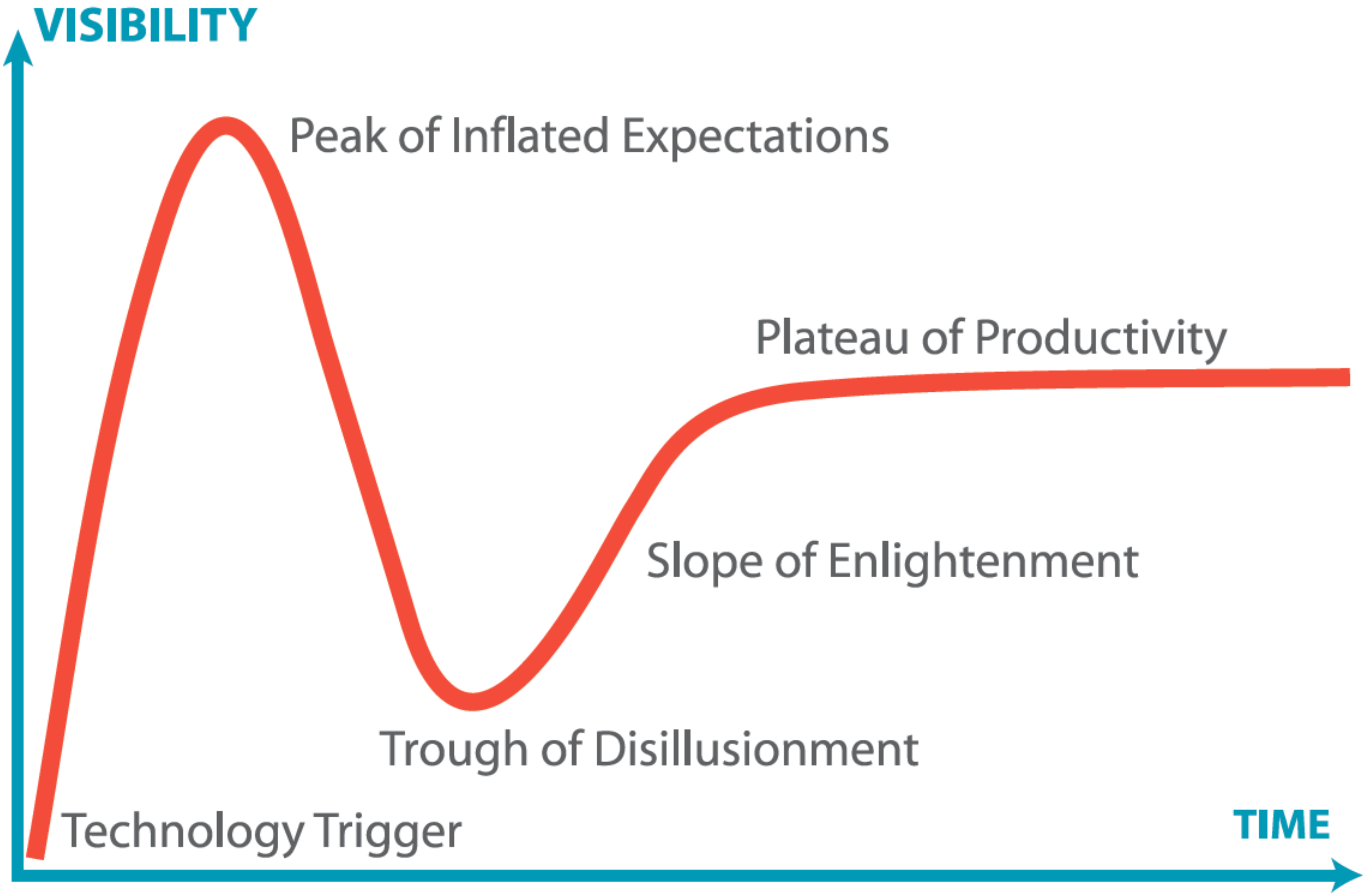


Most used web frameworks among developers worldwide, as of 2022

Source
Stack Overflow
© Statista 2022

Additional Information:
Worldwide; May 11, 2022 to June 1, 2022; 58,743 respondents; Software developers

# Technologies for distributed applications development


VISIBILITY
Peak of Inflated Expectations
Plateau of Productivity
Slope of Enlightenment
Trough of Disillusionment
Technology Trigger
TIME