Software Distribuït - T8 - WEB-Services

Eloi Puertas i Prats

Universitat de Barcelona
Grau en Enginyeria Informàtica

21 de maig de 2015

UNIVERSITAT DE BARCELONA

Resum de paradigmes de programació distribuïda:

- **Message Passing App.** Aplicació de processos distribuïts que intercanvien dades (ex OpenMPI)
- **Client-Server App.** Aplicació client-servidor amb un protocol prefixat.(ex. mail, ftp, web-server)
- **Web Application.** Aplicació client-servidor amb arquitectura MVC sobre HTTP. Existeixen centernars de Frameworks.
- **RMI/RPC/CORBA App.** Aplicació Distribuïda basada en invocació a mètodes remots.
- **Web Services**. Aplicació Distribuïda basada en serveis sobre HTTP. Existeixen molts Frameworks propis, a part de poder adaptar els de Web Applications.
- **P2P.** Aplicació Distribuïda amb arquitectura Peer2Peer. (ex BitCoins, BitTorrent,...) Existeixen pocs frameworks, la majoria ofereixen infrastructura per a desenvolupar P2P (Tapestry, Pastry, JXTA)

Web Services

- Proveeixen una interficie de servei, equivalent a Objectes Distribuïts, sobre WEB.
- Proporcionen interoperatibilitat per a tota la Internet. Són especialment útils en el camp de la integració B2B (business-to-business).
- S'usen com a middleware per al Cloud Computing.
- Els Web Services no estan pensats per a que siguin accedits pels clients directament via un navegador, sino que siguin aplicacions clients dedicades les que ho facin.

Web Services

- Alhora de crear un WebService s'ha d'especificar:
  - Representació dades. (JSON,XML,SOAP)
  - descripció dels serveis. (Web Service Description Language (WDSL), Services API)
  - trobar els serveis(Directori de Serveis UDDI, publicar APIS...)
  - accedir als serveis.(HTTP, TCP, SMTP..)

Arquitectures Web Services

- XML-RPC
- SOAP + WSDL
- RESTful. (Representational State Transfer):

Representació de dades: SOAP

- Un WebService s'identifica mitjançant una URI.
- SOAP s'usa per a encapsular els missatges que s'intercanviaran en el webService.
- SOAP està formatejat usant sintaxis XML.
- Existeixen llibreries SOAP per a la gran majoria de llenguatges d'aplicacions.
- Per transportar SOAP es pot utilitzar qualsevol protocol, dins de l'embolcall no s'hi defineix cap adreça. Normalment s'usa HTTP amb el mètode POST però.

Missatges SOAP

Està encapsulat dins d'un embolcall (envelope). A dins hi trobem:

- Header:(opcional) Dóna contexte al servei, o per mantenir un historial d'operacions.
- Body: Pot ser o un document XML tal qual adreçat a un servei web en particular o part de la comunicació client/servidor, ésset un Request o Reply segons sigui l'emisor.

Definició esquema SOAP

Exemple missatge SOAP: Request

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

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

Exemple missatge SOAP: Response

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

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">
  <m:GetStockPriceResponse>
    <m:Price>34.5</m:Price>
  </m:GetStockPriceResponse>
</soap:Body>

</soap:Envelope>
```

Descripció dels serveis: WDSL

- WDSL (Web Services Descripction Language). Document XML que descriu un servei Web. Especifica la localització del servei i les operacions (o mètodes) que el servei exposa.

<types> Definicions dels tipus de dades usades pel servei web.
<message> Definició dels missatges que s'enviaran
<portType> Conjunt d'operacions suportades pel servei web.
<binding> Especificació del protocol i format de dades per a un servei concret (portType).

Exemple WDSL

```
<message name="GetStockPriceRequest">
  <part name="StockName" type="xs:string"/>
</message>
<message name="GetStockPriceResponse">
  <part name="Price" type="xs:string"/>
</message>
<portType name="GetStockPrice">
  <operation name="getPrice">
    <input message="GetStockPriceRequest"/>
    <output message="GetStockPriceResponse"/>
  </operation>
</portType>
<binding type="GetStockPrice" name="m">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://www.example.org/stock"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

UNIVERSITAT DE BARCELONA

RESTful Web Services

- REST (Representational State Transfer) és una aproximació als WebServices amb un estil d'operacions molt restrictiu.
- els clients usen URL i els mètodes HTTP GET, PUT, DELETE i POST per manipular els recursos representats en XML o JSON.
- l'ènfasis es posa en la manipulació de les dades, quan un nou recurs es creat (POST), te una nova URL la qual pot ser accedida (GET) o modificada (PUT).

RESTful Web Services

els 4 punts bàsics són:

- Usar els mètodes HTTP **explícitament**.
- Ésser independent d'estats (Stateless)
- Proporcionar URI com a estructures de directoris.
- Transferir XML, JavaScript Object Notation (JSON), o tots dos.

Usar els mètodes HTTP explícitament

POST	Per a crear un recurs en el servidor.
GET	Per a recuperar un recurs.
PUT	Per a canviar l'estat d'un recurs o per a actualitzar-lo.
DELETE	per a esborrar o eliminar un recurs.

Mètodes HTTP en el request del client

El mètode HTTP en una petició del client és una paraula reservada (en majúscules), que especifica quina operació del servidor, el client desitja fer. Pot ser segur(+) o no(-) i idempotent(+) o no(-).

+ + GET: per a recuperar el **contingut** de l'objecte web al que fa referència la URI especificada.
+ + HEAD: per a recuperar tan sols la **capçalera** d'un objecte web des del servidor, no el propi objecte.
- - POST: utilitzat per a **enviar dades** a un procés del servidor web.
- + PUT: s'utilitza per demanar al servidor **emmagatzemar** el contingut que s'adjunta amb la petició, a la ubicació del fitxer especificat per la URI en el servidor.

i també però menys utilitzats: DELETE, OPTIONS, TRACE, CONNECT

Exemple Obtenir recurs

```
GET /users/Robert HTTP/1.1
Host: myserver
Accept: application/xml
```

Diferència PUT i POST

```
POST /users HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Robert</name>
</user>
```

Afegeix Usuari Robert a /users. POST No és idempotent, podries anar afegint varis usuaris si exectues vàries vegades el mateix.

Diferència PUT i POST

```
PUT /users/Robert HTTP/1.1
Host: myserver
Content-Type: application/xml
<?xml version="1.0"?>
<user>
  <name>Bob</name>
</user>
```

Accedeixes a users/Robert i canvies el nom, ara és Bob, per tant el recurs serà users/Bob.

Ésser independent d'estats (Stateless)

Dissenyar l'aplicació fent el servidor stateless:

- Servidor
  - Genera respostes que inclouen referències cap a altres recursos.
  - Genera respostes que indiquen quan la informació pot ser guardada en caché o no, via HTTP els tags Cache-Control i Last-Modified tags en la capçalera de la resposta.
- Client
  - Fa servir el `Cache-control` rebut a la resposta, per tal de fer servir o no la caché que disposi del recurs.
  - Fa servir Conditional GET, mitjançant el tag `If-Modified-Since` per preguntar al server si el recurs ha canviat.
  - Envia peticions complets que puguin ser servits independentment d'altres recursos. Ha de ser independent de sessions en el servidor.

Proporcionar URI com a estructures de directoris

- URIs han de ser intuitives, directes, predictibles i fàcilment entendibles.
  - `http://www.myservice.org/discussion/topics/topic`
  - `http://www.myservice.org/discussion/year/day/month/topic`
- Manteniu els noms de les URI's sempre que sigui possible! (campusvirtual2 és un mal exemple)
- Eviteu els query strings el màxim possible.

Transferir XML, JavaScript Object Notation (JSON), o tots dos.

- La representació d'un recurs reflecteix el seu estat actual i els seus atributs en el moment en que el client el demana.
- Aquesta representació és sol fer mitjançant un document estructurat XML o mitjançant un JavaScript Object notation (JSON)
- Existeixen llibreries en la majoria de llenguatges de programació per tal de passar d'objectes o estructures pròpies a estructures equivalents en XML o JSON. (GSON: JAVA-JSON)
- El Content-Type de la resposta s'ha d'adaptar al tipus de document de tornada.
  - JSON: application/json
  - XML: application/xml

UNIVERSITAT DE BARCELONA

Exemple WebService Restful

URL servei
http://date.jsontest.com
Resposta JSON:

```
{
      "time": "03:53:25 AM",
      "milliseconds\_since\_epoch": 1362196405309,
      "date": "03-02-2013"
}
```

Frameworks usats per crear Web Services

- BackEnd específic: Restlets.
- BackEnd WebApps: Ruby On Rails, Django, Servlets ...
- Exemple FullStack: JScript & JQuery -> JSON -> Servlets+GSON

Comparativa Web Services vs Objectes distribuïts (RMI)

- Interacció entre client i servidor molt similar:
  - RMI: el client usa referència remota per invocar una operació sobre l'objecte.
  - WebService: el client usa una URI per invocar una operació sobre el recurs.
- WebService està molt lligat a arquitectura client-servidor. Per crear una aplicació distribuïda més complexa tipus P2P o on es necessiti la col·lobaració de varis sistemes, objectes distribuïts és més addient.

AJAX i REST

- AJAX[1] és una tècnica popular de desenvolupament web que permet fer pàgines web interactives usant JavaScript.
- Els requests són enviats al server fent servir objectes XMLHttpRequest. La resposta és usada pel codi JavaScript del client per tal de canviar dinàmicament la pàgina actual.
- Seguint la filosofia Rest, cada XMLHttpRequest pot ser vist com a una petició a un servei REST i la resposta sovint és un JSON que el propi JavaScript pot manipular directament.

[1] Article on va aparèixer AJAX per primer cop

AJAX (Asynchronous JavaScript and XML)

- Ens permet fer codi client-side amb recepció no bloquejant.
- **XMLHttpRequest**: Objecte JavaScript que envia peticions HTTP.
  - Inicia la petició:
    - Associa una funció anònima que atendrà la resposta a l'event `onreadystatechange` de la petició.
    - Inicia una petició GET o POST
    - Envia les dades
  - Atén la resposta
    - Comprova que l'estat del `readyState` del request sigui 4 i el codi de resposta 200.
    - Extreu la resposta.
    - Processa la resposta.
- JavaScript no es multi-thread. La implementació del navegador és la que s'encarrega de canviar l'estat del request i col·locar un nou esdeveniment en la cua d'events del JavaScript.
- Hi ha llibreries JavaScript amb les funcions d'AJAX implementades com la JQuery.

Creació d'un objecte XMLHttpRequest en JS

- No hi ha una manera de crear-lo que sigui igual per tots els navegadors

```
function createRequest() {
  var result = null;
  if (window.XMLHttpRequest) {
    // FireFox, Safari, etc.
    result = new XMLHttpRequest();
    if (typeof result.overrideMimeType != 'undefined') {
      result.overrideMimeType('text/xml'); // Or anything else
    }
  }
  else if (window.ActiveXObject) {
    // MSIE
    result = new ActiveXObject("Microsoft.XMLHTTP");
  }
  else {
    // No known mechanism -- consider aborting the application
  }
  return result;
}
```

Utilització de l'objecte XMLHttpRequest en JS

- La funció callback que se li assigna s'invoca diferents cops. No és fins l'últim pas en que es disposarà de la resposta.
- Exemple AJAX usant XMLHttpRequest

```
var req = createRequest(); // defined above
// Create the callback:
req.onreadystatechange = function() {
  if (req.readyState != 4) return; // Not there yet
  if (req.status != 200) {
    // Handle request failure here...
    return;
  }
  // Request successful, read the response as Text
  var resp = req.responseText;
  // ... and use it as needed by your app.
}
```

Enviant la Petició

- Un cop hem creat l'objecte de peticions i hem preparat la funció de callback, podem llençar la petició al servidor

```
// open(Method, URL, Asynchronous?)
req.open("GET", "http://date.jsontest.com/", true);
req.send()

req.open("POST","ajax_test.asp",true);
req.setRequestHeader("Content-type",
                     "application/x-www-form-urlencoded");
// send(Body)
req.send("fname=Henry&lname=Ford");
```

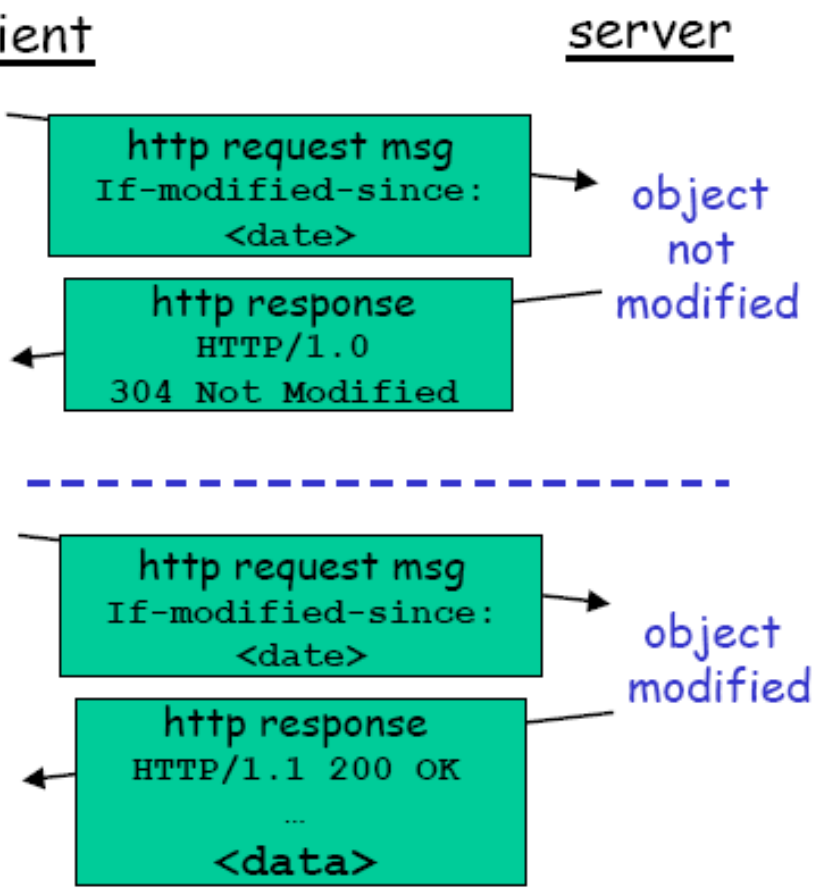
Sistema de Caché

User-server interaction: conditional GET

- Goal: don't send object if client has up-to-date stored (cached) version
- client: specify date of cached copy in http request

  `If-modified-since: <date>`
- server: response contains no object if cached copy up-to-date:

  `HTTP/1.0 304 Not Modified`



2: Application Layer 14

Aplicar Sistema de Caché

ServerSide: Afegir Expires en la capçalera seguit del temps màxim que serà vàlid.

```
response.setDateHeader("Expires",Long.MAX\_VALUE)
```

ClientSide: Afegir `If-Modified-Since` en la capçalera seguit del temps `Last-Modified` de la cache [2]

```
request = new XMLHttpRequest();
var ifModifiedSince =
    cached.getResponseHeader("Last-Modified") ||
    new Date(0); // January 1, 1970
request.open("GET", url, false);
request.setRequestHeader("If-Modified-Since", ifModifiedSince);
request.send("");
if (request.status == 304) {
  request = cached;
}
else {
  cached = request
}
```

[2] Tutorial Caché

Llibreria JQuery

- jQuery és una llibreria JavaScript que fa més simple l'ús d'AJAX, manipulació d'HTML, tractament d'esdeveniments... de forma que sigui portable a un gran nombre de navegadors.
- Tutorial per començar amb jQuery
- Plantilla bàsica per usar JQuery:

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Demo</title>
</head>
<body>
    <script src="static/js/jquery.js"></script>
    <script>

    // Your code goes here.

    </script>
</body>
</html>
```

JQuery i AJAX

- la funció clau és: http://learn.jquery.com/ajax/ ajax()
- Exemple de crida d'api en el mateix servidor

```
// Call a local script on the server /api/getWeather
// with the query parameter zipcode=97201
// and replace the element #weather-temp's html
// with the returned text.
$.ajax({
  url: "/api/getWeather",
  data: {
    zipcode: 97201
  },
  success: function( data ) {
    $( "#weather-temp" ).html(
      "<strong>" + data + "</strong> degrees" );
  }
});
```

JQuery i AJAX.

- En general, les peticions Ajax es limiten al mateix domini on es troba la pàgina que fa la petició (Same Origin Policy)
- Per a poder enviar peticions a clients d'altres dominis (cross-domain requests) el servidor ha d'explicitar-ho en la capçalera del response amb 'Access-Control-Allow-Origin' = * (o llista de dominis en que es pot rebre la sol·licitud)
- Aquest limitació però no aplica a les peticions que es facin via els mètodes AJAX de jQuery.
- Exemple AJAX usant JQUERY

Fer vàries crides simultànees amb AJAX.

- Per poder fer vàries crides simultànees i esperar a que totes acabin, es fa servir la funció`when` de JQuery.
- Se li passen per paràmetre totes les crides AJAX que necessitem
- Retorna un objecte sobre el que es pot fer `done` (en cas d'èxit) o `then` (cas d'èxit o fallada). En tots dos casos se li passa el nom d'una funció que s'executarà quan totes acabin
- Aquesta funció rep tants paràmetres com crides AJAX hàgim fet. Cada paràmetre consta d'un triplet, `[data, status, jqXHR]` on jqXHR és un objecte jquery amb informació sobre la respota.
- en el cas de `then` també li hem de passar el nom d'una funció que tracti el cas de fallada.
- Exemple d'ús.

Entorn de Desenvolupament JavaScript.

- Els navegadors porten eines d'ajuda al desenvolupament (Firefox, Chrome, Safari). Entre d'altres hi trobem:
  - Consola de Javascript
  - Web Inspector
  - Codi
  - Xarxa...
- aquestes eines són indespensables per a poder debugar i testejar el nostre codi.